La modélisation de processus d'exploration numérique en architecture : design génératif, performance et optimisation

Claire Duclos

HESAM UNIVERSITÉ

le cnam

ÉCOLE DOCTORALE ABBE GREGOIRE
Laboratoire MAP-MAACC

THÈSE

présentée par : **Claire Duclos-Prévet**

soutenue le : **12 Janvier 2024**

pour obtenir le grade de : **Docteur d'HESAM Université**

préparée au : **Conservatoire national des arts et métiers**

Discipline : **Section 18**

Spécialité : **Architecture**

La modélisation des processus d'exploration numérique en Architecture

Design génératif, performance et optimisation

THÈSE dirigée par :
Professeur François GUENA, ENSAPLV

Jury

M. Laurent LESCOP, Professeur à l'ENSA de Nantes	Président
M. Pascal TERRACOL, Professeur HDR à l'ENSA Val de Seine	Rapporteur
M. Philippe MARIN, Professeur HDR à l'ENSA de Grenoble	Rapporteur
Mme Aurélie DE BOISSIEU, Professeur à l'Université de Liège	Examinatrice
Mme Nazila BELKADI, Maîtresse de conférences à l'ENSAPLV	Examinatrice
M. Xavier MARSAULT, Ingénieur de recherche à l'ENSA de Lyon	Examinateur

THÈSE

A mon mari et mon fils,

Remerciements

Je souhaite en premier lieu remercier mon directeur de thèse, François Guéna, Professeur des Ecoles d'Architecture à l'école de l'ENSAPLV qui m'a conseillé et guidé pendant ces trois années de recherche. Ses observations avisées m'ont été précieuses dans la réalisation du présent travail.

Je souhaite également remercier les structures qui m'ont permis de réaliser cette thèse, notamment l'ENSAPLV et le CNAM pour le cadre académique fourni et également l'agence Architecture-Studio de m'avoir accueillie.

Je remercie l'ensemble des membres du laboratoire MAP-MAACC pour leur accueil et leur bienveillance. Merci à Louis Vitalis et à Joaquim Silvestre pour leurs conseils. Merci à Nazila Kahina Belkadi et à Thierry Ciblac pour leur participation à mon comité de suivi de thèse. Un grand merci à Benjamin Loiseau, mon collègue de thèse au laboratoire comme à l'agence.

J'adresse un merci spécial à l'ensemble des associés de l'agence, Martin Robain, Rodo Tisnado, Jean-François Bonne, Alain Bretagnolle, René-Henri Arnaud, Laurent-Marc Fischer, Marc Lehmann, Roueïda Ayache, Gaspard Joly, Marie-Caroline Piot, Mariano Efron, Romain Boursier et Widson Monteiro, pour leur confiance et leur soutien tout au long de ce projet de recherche. Tout particulièrement, je remercie profondément Mariano Efron pour ces encouragements ; lui sans qui ce projet de recherche n'aurait probablement pas été possible. Je remercie l'ensemble des chefs de projets et membres des équipes de projets pour la patience et la bienveillance dont ils ont pu faire preuve lors de nos différentes collaborations. Merci enfin à Isabelle Gunasena et Emmanuelle Lefort pour leur soutien.

Dans un autre registre, je tiens à remercier mes amis pour leur écoute et leur soutien durant ces dernières années si particulières. Merci notamment à Daiana, Catia, François, Daliana, Ruxandra, Ezéchiel, Sarah, Elodie, Matthieu, Fanny, Run, Patricia, Julia, Olivier, Manon, Alexandre et Louis-Marie.

Pour finir, je remercie mon époux pour son support sans faille tout au long de cette aventure, merci pour son enthousiasme à l'égard de cette recherche, merci pour ces conversations enrichissantes, merci pour toutes ces relectures et merci pour tout le reste.

Résumé

Les contextes climatiques et réglementaires imposent aux architectes une prise en compte de plus en plus précoce des contraintes environnementales lors la conception de leurs projets. Depuis la fin des années 1990 des méthodes innovantes qui permettent de prendre en compte les paramètres environnementaux en phase amont de projet en s'appuyant sur la puissance de calcul des ordinateurs sont décrites dans la littérature scientifique. Elles suivent toutes une même approche, à savoir l'élaboration d'une technique générative reliant (1) un modèle génératif, souvent paramétrique, pour définir un espace de solutions, (2) un modèle d'évaluation impliquant l'utilisation éventuelle d'outils de simulation, et (3) un modèle d'exploration nécessitant l'usage d'algorithmes métaheuristiques d'optimisation comme les algorithmes évolutionnaires multicritères.

Ces méthodes cherchent à reproduire automatiquement et rapidement, et dans des quantités beaucoup plus importantes, le processus itératif qui existe entre architectes et ingénieurs dans la méthode de conception traditionnelle. Toutefois, si elles sont très populaires auprès des chercheurs, elles restent peu utilisées en agence d'architecture. Les suppositions énoncées dans la littérature pour expliquer ce « gap » entre leur intérêt théorique et leur utilisation en pratique restent à démontrer dans un contexte professionnel de conception architecturale. Le premier objectif de cette thèse est d'expérimenter ces techniques sur des projets réels. La quarantaine d'expérimentations réalisées a permis d'identifier de nouveaux verrous sur l'usage de ces pratiques dans un contexte professionnel jusqu'ici non relevés par la littérature, notamment des limites « structurelles » liées aux méthodes de travail et d'organisation des architectes, mais aussi à la nature même du projet d'architecture.

D'un point de vue technique, le frein majeur rencontré lors de l'usage de techniques d'optimisation s'est révélé être la difficile intégration de contraintes permettant de restreindre l'espace de solutions à des solutions réalisables. Les méthodes de gestion des contraintes génériques se sont révélées inefficaces lors de nos expérimentations sur ce type de problèmes. Ainsi, le second objectif de cette thèse est d'identifier des méthodes de gestion des contraintes adaptées. Une étude comparative de 7 méthodes a permis de révéler que la méthode des fonctions de réparation qui nécessite l'usage de techniques génératives telles que des modèles à base d'agents peut être très efficace. Ainsi, nous avons entrepris le développement d'une bibliothèque d'outils numériques pour Grasshopper permettant aux architectes d'utiliser des techniques génératives pour faire de l'optimisation multicritère sous contraintes.

Mots clés : Design génératif, architecture, performance environnementale, algorithmes évolutionnaires multicritères optimisation sous-contrainte, modèle à base d’agents, machine learning

Abstract

Climatic and regulatory contexts urge architects to take environmental constraints into account at the earliest stages of design processes. Since the end of the 1990s, the scientific literature has described innovative methods for taking environmental parameters into account using the calculation power of computers. These methods share the same approach based on a generative technique linking (1) a generative model, often parametric, to define a space of solutions, (2) an evaluation model involving the possible use of simulation tools, and (3) an exploration model requiring the use of metaheuristic optimization algorithms such as multi-criteria evolutionary algorithms.

These methods aim at reproducing, automatically and at much larger scale, the iterative process usually established between architects and engineers in the traditional design method. However, while they are very popular with researchers, these methods remain little used in architectural practice. The assumptions made by the literature to explain this "gap" between theory and practice have yet to be tested in a professional architectural design context. The first objective of this thesis is to experiment such techniques on real projects. The forty or so experiments carried out identified new obstacles to the use of these practices in a professional context, hitherto unnoticed in the literature, notably "structural" limits linked to architects' working and organizational methods, but also to the very nature of the architectural project.

From a technical perspective, the major obstacle encountered when using optimization techniques was the difficulty of integrating constraints to restrict the solution space to feasible solutions. Generic constraint management methods proved ineffective in our experiments on this kind of problems. As a consequence, the second objective of this thesis is to identify suitable constraint management methods. A comparative study of 7 methods revealed that the repair function method, which requires the use of generative techniques such as agent-based models, can be very effective. We have therefore undertaken the development of a library of numerical tools for Grasshopper, enabling architects to use generative techniques for multi-criteria optimization under constraints.

Key words: Generative design, architecture, environmental performance, evolutionary multi-criteria algorithms, under-constrained optimization, agent-based model, machine learning

Table des matières

Liste des tableaux

Liste des figures

Liste des annexes

Introduction

L'été 2023 a été la saison la plus chaude jamais enregistrée dans le monde. Les derniers rapports du GIEC dressent un bilan alarmant sur le réchauffement climatique global dont les effets se ressentent partout dans le monde. L'impact environnemental de l'industrie de la construction est conséquent, et ses acteurs, notamment les architectes qui conçoivent et réhabilitent les bâtiments, sont poussés à non seulement prendre en compte les préoccupations environnementales, mais à en faire une priorité au vue des nouvelles exigences. C'est notamment l'objectif de la réglementation environnementale de 2020, qui impose que les bâtiments neufs ne soient plus consommateurs mais producteurs d'énergie.

En parallèle de la transition énergétique, les architectes doivent faire face à la transition numérique. L'avènement récent des applications fondées sur l'utilisation d'algorithmes d'intelligence artificielle générative, comme Midjourney ou Dall-E, qui permettent d'automatiser la génération d'images, sont en train de révolutionner le monde du graphisme, de l'illustration et peut-être bientôt celui de l'architecture (Ali Elfa & Dawood, 2023). Si le sujet préoccupe beaucoup les architectes depuis ces derniers mois, en réalité, les recherches sur le design génératif appliqué à l'architecture existent depuis des décennies (Caetano et al., 2020). Son objectif est d'utiliser la puissance de calcul de l'ordinateur pour explorer numériquement et automatiquement des milliers de propositions architecturales pour, notamment, répondre aux besoins de la transition énergétique. Ces travaux ont donné naissance à un champ de recherche à part entière, le « *design génératif et performanciel* » (Touloupaki & Theodosiou, 2017).

Les résultats de ces précédentes recherches ont montré que ces outils et méthodes numériques innovantes peuvent théoriquement assister un architecte dans la conception de bâtiments performants lors des phases amonts de projet (Bernal et al., 2020). S'il existe différentes méthodes génératives, la plus répandue dans la littérature scientifique consiste à produire un espace de solutions à l'aide d'une modélisation paramétrique reliée à des outils de simulations environnementales, et d'utiliser un mécanisme d'optimisation, pour trouver les valeurs des paramètres permettant de produire les solutions satisfaisant au mieux des objectifs de performance à atteindre (Zhao & de Angelis, 2018).

Ces techniques d'optimisation morphologique ne sont en réalité pas nouvelles, puisqu'elles sont connues et utilisées dans d'autres champs d'application comme l'aérauspatiale ou l'aéronautique depuis des décennies. Aujourd'hui, ces techniques et outils, dont la grande majorité sont disponibles en *open source*, sont accessibles aux architectes, notamment via le logiciel de modélisation Rhinocéros® et son outil de programmation visuel Grasshopper®, pour ne citer que le plus connu. De nombreux efforts ont été réalisés ces dernières années sur les interfaces graphiques pour rendre ces outils facilement appropriables par les architectes. Aussi, il existe déjà des formations au design génératif dispensées dans les écoles d'architecture. Cependant, ces méthodes innovantes passent encore difficilement la porte des agences d'architecture. Rares encore sont celles qui utilisent ces outils. Si quelques chercheurs ont tenté d'expliquer ce phénomène (Attia et al., 2009; Nguyen et al., 2014; X. Shi et al., 2016), il n'existe pas de certitude sur ce qui justifie cet écart entre théorie et pratique, notamment car la grande majorité des cas d'étude présentés dans la littérature scientifique n'ont pas été réalisés dans un cadre professionnel. Ce constat est le point de départ de cette thèse de doctorat, il justifie le besoin d'une recherche approfondie.

Le premier objectif de cette thèse est avant tout de déterminer les éléments qui constituent des freins à la démocratisation du design génératif et performanciel en agence d'architecture. Une fois ces différents freins identifiés, le second objectif est de proposer des méthodologies d'usage adaptées aux particularités de l'activité professionnelle dans ce secteur.

Pour atteindre ces objectifs, il nous a semblé nécessaire d'être immergés dans les activités professionnelles d'une agence afin de confronter l'usage des méthodes de design génératif performanciel aux réalités de l'activité professionnelle. C'est ce qui nous conduit à réaliser notre recherche dans le cadre d'un dispositif de type Convention Industriel de Formation à la Recherche. Une convention a été signée entre le laboratoire MAP-MAACC et l'agence d'architecture parisienne « Architecture Studio » afin de mener la recherche.

Cette agence, créée en 1975 est connue internationalement pour ces grandes réalisations comme le Parlement Européen ou l'Institut du Monde Arabe, mais elle est aussi connue dans le milieu pour sa méthode de travail collaborative incarnée par le tracé rouge, le tracé bleu et pour son organisation particulière, l'agence se présentant elle-même comme un collectif d'architectes. Au démarrage de cette thèse, Architecture Studio était dirigée par 13 associés, elle comptait plus d'une vingtaine de partenaires et plus d'une centaine de collaborateurs au

total issus de différentes cultures et de différentes générations. Les projets réalisés par Architecture Studio sont très divers. Il peut s'agir de projets architecturaux ou de projets urbains, de petits ou de grands projets, pour des clients publics comme privés, sur tous les continents, et pour tous les types de programmes. L'agence ne revendique pas un style d'architecture particulier et les outils de conception utilisés vont du dessin à la main, à la modélisation BIM en passant par la CAO en 2D sur Autocad mais aussi la conception paramétrique sur Grasshopper notamment pour la conception des enveloppes. Avant mon arrivée dans l'agence, certains collaborateurs s'intéressaient déjà à l'exploration numérique et souhaitaient poursuivre ces expérimentations dans un cadre scientifique. Une méthode d'optimisation avait notamment été testée sur un premier projet pour la conception d'un système de protection solaire pour les bureaux Summers à Buenos Aires[1]. Cet intérêt pour les méthodes de conception et cette diversité à tous les niveaux (collaborateurs, projets et outils) font de cette agence un lieu particulièrement adapté pour expérimenter le design génératif et performanciel et a offert un cadre particulièrement approprié pour notre recherche.

Ce mémoire de thèse retrace les différentes étapes de la recherche. La première étape consistait à comprendre pourquoi l'exploration numérique tarde à être démocratisée en agence d'architecture. Cette compréhension nécessitait de répondre à diverses questions : Qu'est-ce qu'est concrètement une méthode de design génératif ? A quoi peut-elle réellement servir ? Et, surtout, comment mettre en place une exploration numérique pour la performance environnementale ? Comment la mettre en œuvre numériquement parlant ? Mais aussi quand et comment l'insérer dans le processus traditionnel de conception des architectes ? La seconde étape s'est appuyée sur les résultats de la première pour proposer des méthodes adaptées qui pourraient permettre une appropriation plus rapide des méthodes de design génératif performanciel dans les agences d'architecture.

La première partie du mémoire de thèse est une synthèse de toutes les connaissances nécessaires pour mettre en œuvre des activités de conception générative et performancielle. Elle contient deux chapitres, un premier sur les connaissances issues de la littérature scientifique, et un second chapitre sur le retour d'expérience des années passées chez Architecture-Studio où nous avons cherché à expérimenter des méthodes d'exploration numérique sur des projets en cours de conception. Le premier chapitre présente un benchmark des outils numériques à la

[1] Ce projet est aujourd'hui construit et a reçu le prix de l'AFEX à Venise en 2020.

disposition des architectes pour mettre en œuvre ces méthodes et identifie une bibliothèque d'outils Grasshopper pour effectuer des analyses environnementales : études d'ensoleillement, analyse de la radiation solaire, calculs de facteur de lumière du jour ou d'autonomie en lumière du jour, analyses de la qualité de vue vers l'extérieur, analyses du confort thermique, de l'écoulement des vents, de la protection contre les précipitations etc..

Le deuxième chapitre présente notre activité de recherche qui peut s'apparenter à une « recherche-action » car nous nous sommes mis à disposition des équipes de projets pour réaliser des analyses, des études paramétriques et résoudre des problèmes d'optimisation sur des projets, principalement pour la conception des enveloppes et le plus souvent lors des concours d'architecture. Le chapitre détaille plus d'une quarantaine d'expérimentations réalisées sur des projets avec des interventions pouvant durer d'une journée à plusieurs semaines sur un même problème. Ces activités de conception expérimentales sont ensuite analysées afin d'identifier les difficultés qui peuvent constituer des freins à leur mise en œuvre dans un contexte professionnel : manque de compétences techniques des équipes de projets, organisation du travail peu adaptée, temps de calculs etc...

De cette première partie du travail de recherche, il ressort que, si certaines difficultés sont surmontables, comme par exemple avec l'usage du Machine Learning pour réduire les temps de calcul, un frein majeur à l'utilisation de cette méthode dans un cadre professionnel est la difficulté d'intégrer les contraintes du projet dans le processus d'exploration. Les espaces de solutions définis à l'aide de modèles paramétriques contiennent bien souvent une part importante de solutions irréalisables, voir aberrantes. C'est ce qui nous a conduit, dans un second temps de la recherche, présenté dans la seconde partie de cette thèse, à étudier, tester, comparer et développer des méthodes de gestion des contraintes dans les processus d'optimisation appliqués aux problèmes de conception architecturale.

La seconde partie de la thèse a pour objet les méthodes d'optimisation multicritères et identifie différentes méthodes de gestion des contraintes compatibles avec ces algorithmes. Certaines de ces méthodes sont issues de la littérature scientifique sur l'optimisation, d'autres sont issues de la littérature sur le design génératif. Ces méthodes sont utilisées sur un cas d'étude issu de la pratique professionnelle d'Architecture Studio afin de les comparer selon différents critères comme leur performance, leur difficulté de mise en œuvre ou la qualité architecturale des solutions générées.

L’une de ces techniques, appelée « réparation », consiste à modifier les solutions ne respectant pas les contraintes pour les rendre faisables en imaginant une heuristique propre à chaque problème. Le quatrième et dernier chapitre présente les résultats de tests de réparation effectués sur des cas rencontrés lors des expérimentations sur les projets de l’agence n’ayant pu aboutir à cause de cette difficulté d’intégrer efficacement des contraintes. Il montre que des fonctions de réparation fondés sur des modèles à base d’agents comme l’intelligence en essaim ou les automates cellulaires peuvent être assez efficaces.

Nous concluons ce travail de recherche par le développement d’une bibliothèque d’outils conçus pour mettre en œuvre des mécanismes d’optimisation multicritère incluant des fonctions de réparation basées sur des modèles à base d’agents. L’ensemble de ces outils prend la forme dans un plugin Grasshopper appelé « Urchin ». Les capacités de ce plugin sont illustrées à la fin de cette thèse avec des applications sur différents sujets inspirés à la fois de la littérature scientifique et de la pratique.

Partie 1

Les connaissances et mise en pratique de l’exploration numérique de solution architecturale

Cette première partie a pour objectif de servir de guide à quiconque souhaite s'initier à l'exploration numérique. Elle est divisée en deux chapitres, un premier qui rassemble les connaissances trouvées dans la littérature scientifique, et un second qui rassemble des connaissances d'une autre nature, celles acquises lors de l'expérimentation de ces méthodes identifiées dans la littérature.

D'abord, nous commencerons par expliquer les raisons qui, selon nous, font de ces méthodes innovantes un sujet d'intérêt pour les architectes. Nous revenons ensuite sur la terminologie propre au design génératif avant de définir ce qu'est en pratique une exploration numérique. Nous chercherons à comprendre ce qui distingue la pratique de l'optimisation chez les architectes de celle des ingénieurs. Nous passerons en revue différents cas d'application d'exploration numérique identifiés dans la littérature scientifique. Nous appréhenderons alors ce qui fait de la modélisation un exercice plus ou moins complexe selon le type de problèmes rencontré (enveloppe, morphologie, urbanisme, et génération de plans). Nous passerons en revue les outils *open source* pour l'évaluation de la performance environnementale à disposition des architectes.

Ensuite nous présenterons l'aboutissement de plusieurs années d'expérimentations sur des cas d'étude issus de la pratique professionnelle. Ces expérimentations ont permis de confronter les connaissances issues de la littérature aux contraintes de la pratique sur des projets réels. Nous chercherons à comprendre pourquoi les techniques d'explorations numériques sont si peu populaires auprès des architectes. Nous verrons que les raisons évoquées dans la littérature scientifique ne sont plus tout à fait d'actualité et que les questions comme le rôle du dessin, les différentes philosophies de conception, le management du projet, la formulation d'un problème d'architecture et surtout l'intégration de contraintes au processus d'optimisation peuvent s'avérer bien plus rédhibitoires que les temps de calcul tant décriés par les chercheurs jusqu'alors. Afin d'appuyer notre propos, nous reviendrons sur les expérimentations les plus notables, qu'elles aient pu aboutir ou non, ce qui permettra d'illustrer les limites que présentent ces méthodes de conception innovantes. Nous finirons cette première partie par l'expérimentation, sur différents problèmes de conception, d'algorithmes d'intelligence artificielle, encore peu utilisés, mais connus pour leur capacité à réduire les temps de calcul. L'objectif sera alors de savoir si les temps de calculs des évaluations représentent encore un réel handicap pour la démocratisation des méthodes d'exploration numérique en agence d'architecture.

Chapitre 1 : Les enseignements de la littérature

Ce premier chapitre est une synthèse de toutes les connaissances issues de la littérature scientifique qui sont nécessaires à la mise en pratique de l'exploration numérique de solutions architecturales pour la performance environnementale (ci-après, ENAPE). Ces éléments sont ici présentés en trois temps.

D'abord, sont exposées les motivations qui conduisent à vouloir appliquer ce type d'approches ainsi que le fonctionnement général d'une ENAPE. Ensuite, une revue de littérature des exemples d'applications intégrant des considérations architecturales est proposée. Enfin, les indicateurs du confort et de l'énergie ainsi que les outils numériques disponibles pour les architectes permettant de les calculer sont identifiés et ont été rassemblés dans une bibliothèque dédiée à la conception générative et paramétrique.

1.1 Transitions écologique et numérique

Dans cette première partie, une première section est consacrée aux problématiques environnementales telles qu'elles sont prises en compte dans la conception traditionnelle. L'enjeu est de présenter au lecteur, d'une part, les données clefs de la transition environnementale dans le secteur du bâtiment ainsi que le rôle que peuvent jouer les architectes dans cette transition ; et, d'autre part, les outils et méthodes utilisées en pratique par la maîtrise d'œuvre pour étudier la performance des bâtiments.

Une seconde section définit les contours et les objectifs de l'exploration numérique de solutions architecturales fondée sur des critères de performance environnementale en resituant cette approche dans le champ de l'architecture computationnelle, et en présentant les grands principes de son fonctionnement technique.

1.1.1. Les problématiques environnementales dans la conception traditionnelle

Le bâtiment, un secteur énergivore

Les données internationales

Les données du rapport des Nations Unies sur les consommations énergétiques des bâtiments et du secteur de la construction (Agence Internationale de l'énergie & Programme des Nations Unies pour l'environnement, 2018) sont particulièrement explicites. L'impact environnemental de cette industrie est gigantesque, supérieur à celui des transports, comme le montre les graphiques de la Figure 1. En 2017, la construction et l'exploitation des bâtiments représentaient 36 % de la consommation finale d'énergie dans le monde et 40 % des émissions de dioxyde de carbone. Seule, l'industrie de la construction représente 6 % de la consommation énergétique mondiale.

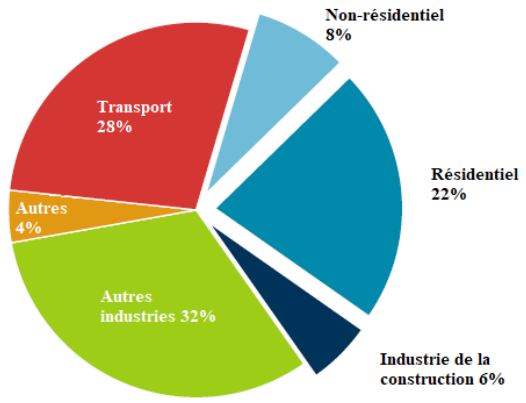


Figure 1 : Part mondiale de l'énergie finale des bâtiments et de la construction en 2017 (source: Agence Internationale de l'énergie & Programme des Nations Unies pour l'environnement, 2018).

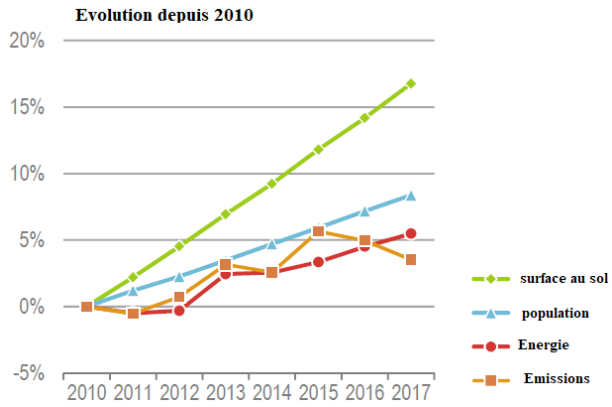


Figure 2 : Evolution des indicateurs de consommations d'énergie, d'émissions de CO2, de la taille de la population et de surface de sol bâti depuis 2010 (source: Agence Internationale de l'énergie & Programme des Nations Unies pour l'environnement, 2018).

Le graphique présenté en Figure 2 montre que malgré les efforts et la baisse des émissions de dioxyde de carbone depuis 2015, la consommation énergétique du secteur du bâtiment continue de croitre, et la surface totale de sol construit augmente bien plus vite que la population mondiale.

Les consommations se répartissent en sept usages différents du plus consommateur au moins consommateur : le chauffage, l'eau chaude sanitaire, les équipements de cuisson, les

équipements et autres, le refroidissement et l'éclairage artificiel (voir Figure 3). L'augmentation significative de la consommation énergétique due aux systèmes de refroidissement observée depuis 2015 serait expliquée pardes étés particulièrement chauds liés aux changements climatiques.

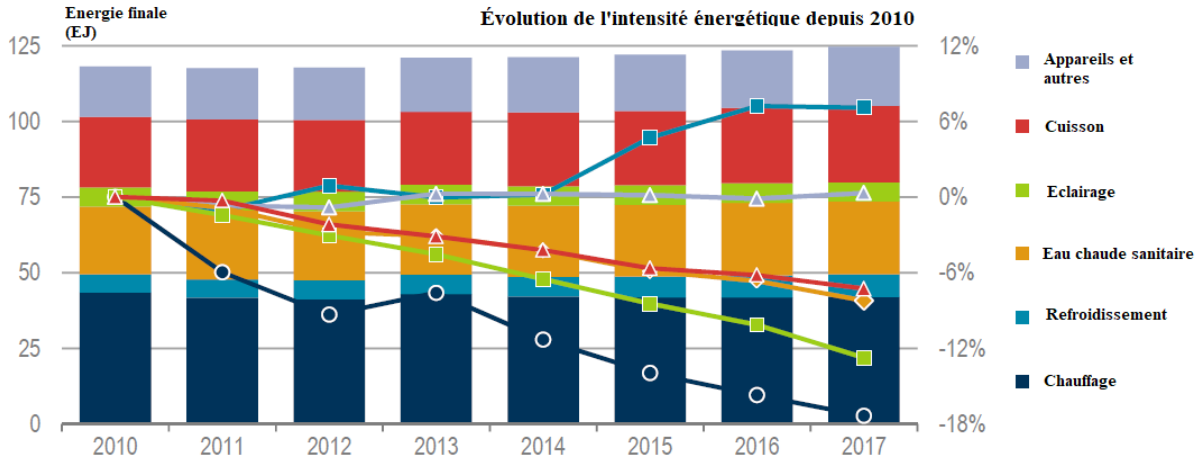


Figure 3: Consommation d'énergie finale des bâtiments dans le monde et évolution de l'intensité par utilisation finale, 2010-17 (source : Agence Internationale de l'énergie & Programme des Nations Unies pour l'environnement, 2018)

Un bâtiment consomme de l'énergie tout au long de son cycle de vie, de l'extraction des matériaux jusqu'à sa démolition en passant par la fabrication et le transport des matériaux, la construction, l'exploitation et la rénovation. La phase d'exploitation est, de loin, la plus énergivore puisqu'elle représente près de 85 % de la consommation d'énergie totale du bâtiment. Le transport , la construction et la démolition additionnés représentent seulement 1 % (Schenk & Amiri, 2022).

Les données françaises

Les conclusions sont aussi inquiétantes en considérant les données françaises. Selon les données de l'agence de la transition écologique (*ADEME Bâtiments*, s. d.), le secteur du bâtiment en France est le premier consommateur d'énergie en prenant en compte le chauffage, la climatisation, l'éclairage, la ventilation et les équipements. Il représente 44 % de la consommation d'énergie en France, dont environ deux tiers pour le secteur résidentiel. Le graphique en Figure 4 montre que la part d'énergies renouvelables utilisée pour ces consommations reste encore faible.

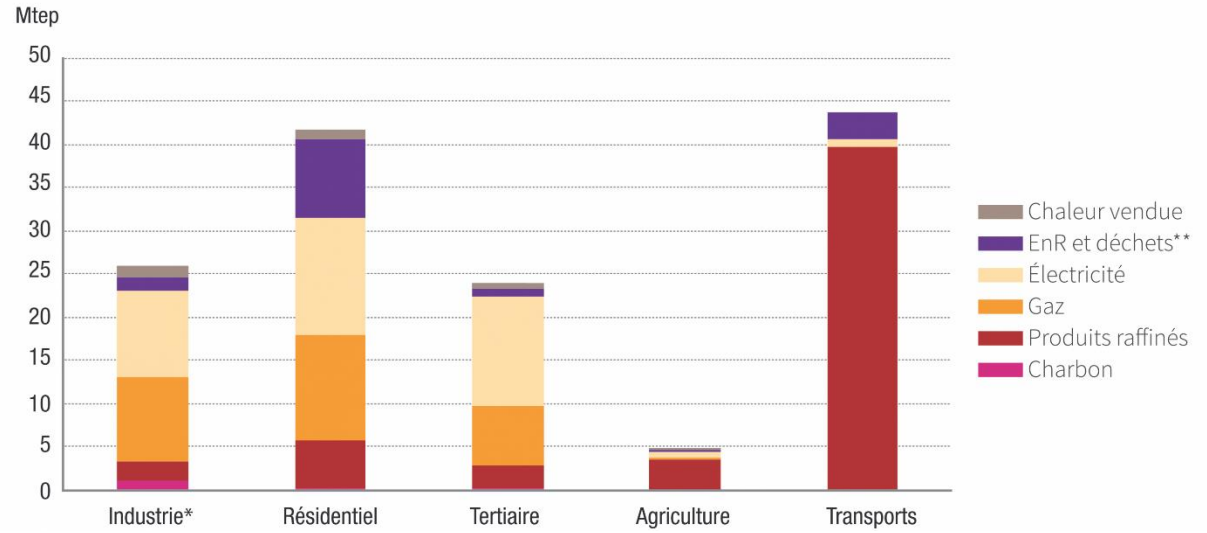


Figure 4: Consommation d'énergie finale de chaque secteur (source: Cerema, ADEME, 2018)

D'après les statistiques du Ministère de la transition écologique et de la cohésion des territoires (*Ceren*, 2018), en 2018, le chauffage et l'électricité spécifique (qui comprend l'éclairage et les équipements) sont les principales sources de consommation énergétique d'un bâtiment tertiaire ou résidentiel. Ils représentent respectivement 42 % et 27 %, en moyenne, de la consommation énergétique d'un bâtiment tertiaire et 64 % et 18 % d'un bâtiment résidentiel (voir Figure 5).

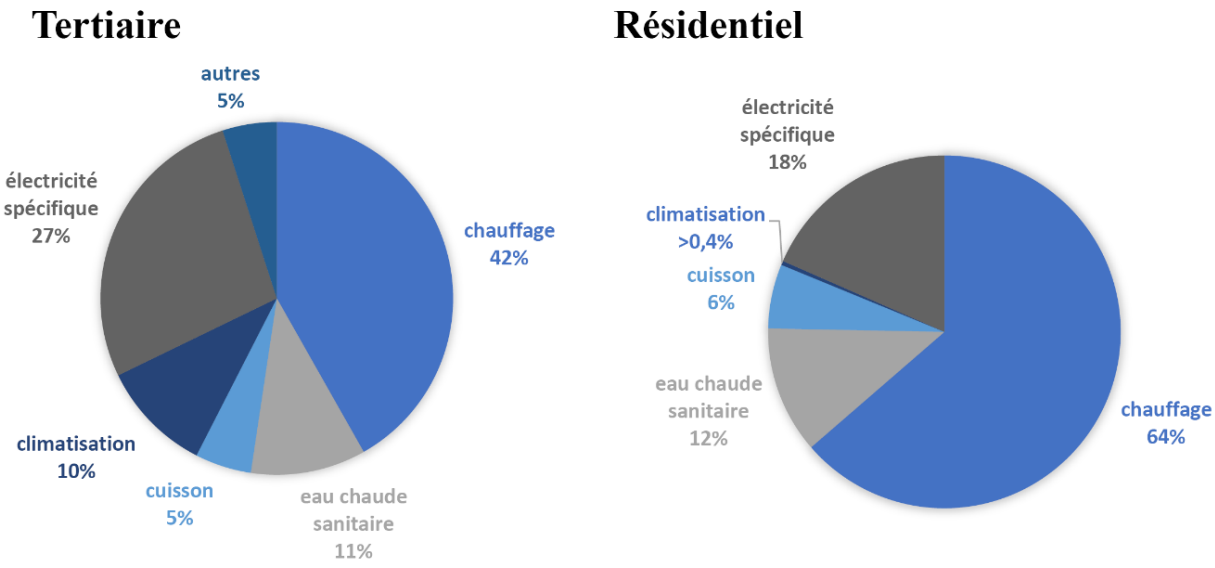


Figure 5 : Répartition des consommations par usage en France en 2020 dans le tertiaire et le résidentiel (source: Ceren, 2018)

Des réglementations toujours plus exigeantes

En conséquence, le secteur du bâtiment se voit imposer des objectifs de réduction des consommations du parc immobilier toujours plus ambitieux (voir Figure 6). En France, depuis janvier 2021, la Réglementation Environnementale 2020 (*RE2020*, s. d.) doit être appliquée dans les logements ;et, depuis Juillet 2022, cette réglementation doit aussi être appliquée dans les bâtiments tertiaires les plus communs (bureaux, écoles). La RE2020 vise à généraliser les normes Bepos (Bâtiment à énergie positive) à toutes les constructions neuves. Toutes les nouvelles constructions de logements (individuels ou collectifs), doivent produire plus d'énergie qu'elles n'en consomment sur cinq utilitaires (le chauffage, l'éclairage, l'eau chaude sanitaire, la climatisation et les auxiliaires techniques) en s'appuyant notamment sur l'usage de panneaux photovoltaïques. Les bâtiments tertiaires spécifiques (hôtels, commerces, gymnases…) seront aussi concernés très prochainement.

Pour atteindre ces objectifs, la RE2020 fixe un objectif de 30 % de réduction des besoins énergétiques par rapport aux exigences de la réglementation précédente (RT2012). Les besoins nets en énergie correspondent à la somme des déperditions de chaleur (parois, vitrages, ventilation, ponts thermiques et ouverture des portes) à laquelle on soustrait la somme des apports de chaleur (rayonnement solaire, appareils électriques et occupants). Les besoins énergétiques sont représentés dans la réglementation par l'indicateur Bbio (pour les besoins bioclimatiques).

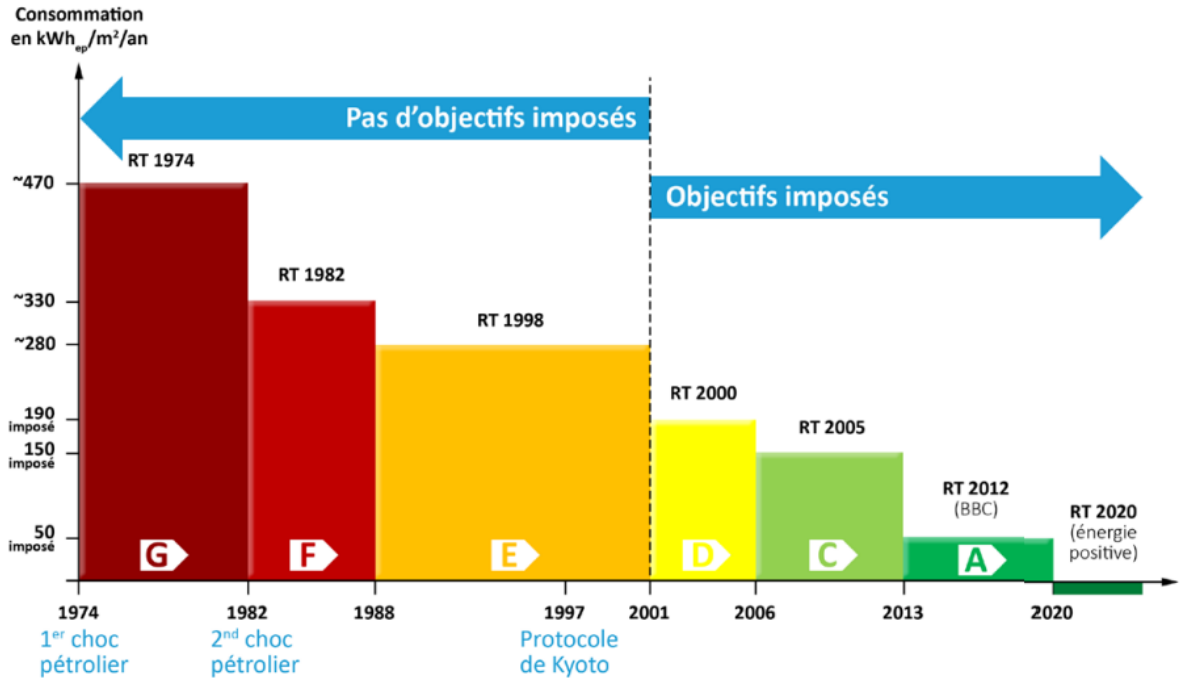


Figure 6: Les réglementations thermiques en France (source: planbatimentdurable.fr)

Un autre indicateur, le Cep sert à mesurer la consommation en énergie primaire. Quatre nouveaux indicateurs sont apparus avec cette nouvelle réglementation : (1) le Cep qui mesure la consommation en énergie primaire non renouvelable, (2) l'Icénergie qui mesure l'impact carbone des consommations d'énergie, (3) l'Icconstruction qui mesure l'impact carbone des produits de constructions ; et (4) le DH (degrés heures) qui mesure la durée et l'intensité de l'inconfort thermique estival.

Le rôle prépondérant de l'architecte

La forme et l'orientation du bâtiment, la conception de son enveloppe et même la disposition des espaces intérieurs ont un impact sur la performance des bâtiments. L'ensemble de ces paramètres architecturaux relèvent des décisions des architectes, leur rôle est donc essentiel dans la lutte contre le réchauffement climatique.

Impact de l'enveloppe et de la morphologie sur la performance

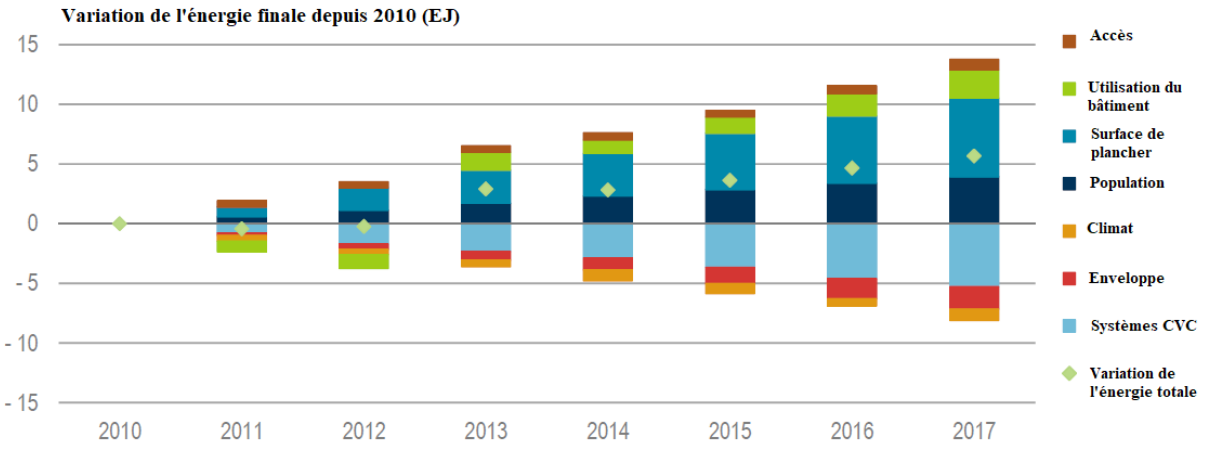


Figure 7 : Influence des facteurs sur la consommation d'énergie des bâtiments dans le monde, 2010-17 (source: Agence Internationale de l'énergie & Programme des Nations Unies pour l'environnement, 2018)

L'impact de l'enveloppe et de l'inertie thermique sur la performance énergétique des bâtiments n'est plus à démontrer (Aste et al., 2009 ; Sadineni et al., 2011 ; Sozer, 2010; Verbeke & Audenaert, 2018). D'après une méta-analyse de 2022 (Latha et al., 2022), le dimensionnement d'un bâtiment peut permettre de réduire de 17 à 35 % sa consommation d'énergie, 35 à 40 % pour le choix du vitrage, et 30 à 60 % pour le détail des menuiserie. Selon le *Global Status Report* de 2018 des Nations Unies (Agence Internationale de l'énergie & Programme des Nations Unies pour l'environnement, 2018), les dernières mesures pour l'amélioration des enveloppes, notamment l'amélioration de l'isolation et des menuiseries, mais

aussi l'amélioration des systèmes CVC (Chauffage, Ventilation, Climatisation) ont permis de fortement compenser l'impact de l'augmentation de la population, de la surface de sol et des activités de services énergétiques dans les bâtiments (voir Figure 7).

De nombreuses études ont montré que dans un climat tempéré comme celui de la France, la forme du bâtiment peut avoir un impact significatif sur sa consommation énergétique (voir en particulier Catalina et al., 2011 ; Izard, 1985). En 1985, Jean-Louis Izard interpelait déjà les architectes dans une étude comparative de la sensibilité des paramètres architecturaux : *« si l'usager a les moyens d'exercer une influence quasi-despotique sur les résultats d'exploitation, notamment par le choix de sa température de consigne […] le rôle de l'architecte en conception thermique est très important »*. Plusieurs paramètres sont comparés dans cette étude : la forme, l'ouverture Sud, l'isolation thermique, l'orientation, l'inertie thermique, la nature et la complexité des systèmes passifs complémentaires, la résistance thermique de l'isolation nocturne, le comportement de l'usager et le site (la météo). Izard conclut que deux paramètres liés à l'architecture sont les plus impactant : l'isolation thermique et la morphologie. Les autres paramètres architecturaux obtiennent des résultats sensiblement égaux. Avec une mauvaise morphologie, le bilan énergétique pourrait être amélioré par des prestations très soignées, donc très coûteuses.

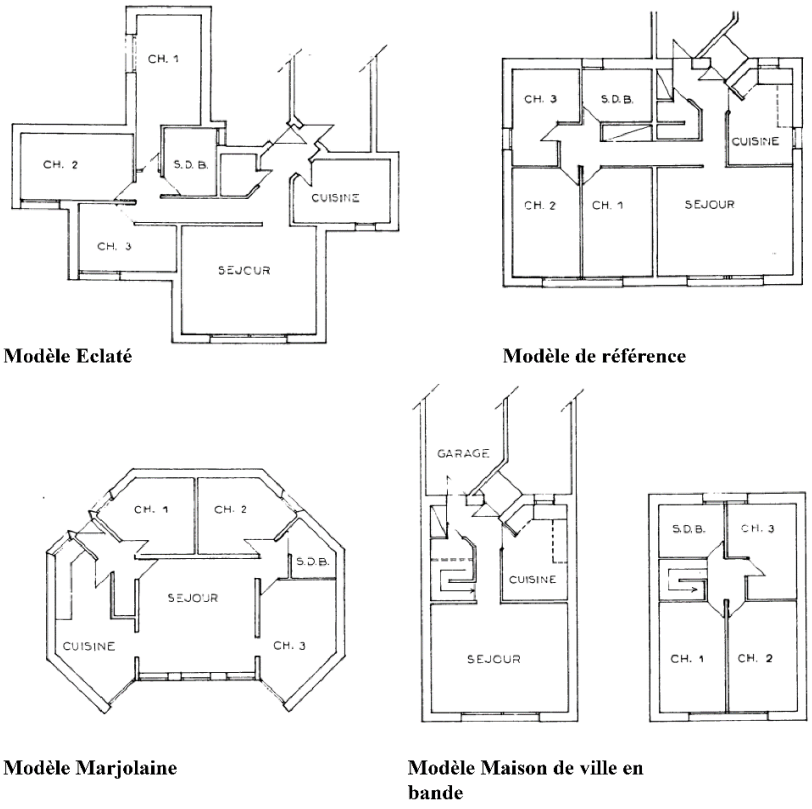


Figure 8 : Les différentes formes comparées dans l'étude de la sensibilité des paramètres architecturaux (source: Izard, 1985)

Ainsi, à isolation thermique et ouverture au Sud égales, la forme a un impact « *assez* » fort sur les besoins thermiques. Son étude comparative porte sur 4 formes de logements de type F4 différents présentés en Figure 8 : un logement de référence rectangulaire sur un niveau, un modèle « *maison de ville* » en bande, un modèle « *marjolaine* », et un modèle « *éclaté* ».

Les besoins peuvent être réduits de 45 % dans un climat océanique et de 64 % dans un climat méditerranéen par rapport au modèle de référence avec un modèle « *maison de ville* » en bande. A l'inverse, ils augmentent de 31 % (climat océanique) à 50 % (climat méditerranéen) avec le modèle éclaté. Ainsi les besoins énergétiques peuvent quadrupler d'une forme à l'autre selon le climat.

Impact de l'agencement des espaces sur la performance

L'influence de l'agencement spatial sur la performance énergétique des bâtiments n'a pas été aussi clairement établie que pour l'enveloppe et la morphologie (Latha et al., 2022).

Certaines études concluent que même l'agencement des espaces intérieurs pourrait avoir une influence sur la performance du bâtiment (Du, Jansen, et al., 2020). En effet, une étude comparative (Du, Jansen, et al., 2020) a montré que l'agencement de l'espace peut permettre de réduire significativement les besoins énergétiques, notamment les besoins en éclairage. Cette étude compare 11 scénarios d'agencement d'espace pour un projet de bureaux (voir Figure 9) analysés dans trois climats différents (Amsterdam, Harbin et Singapore).

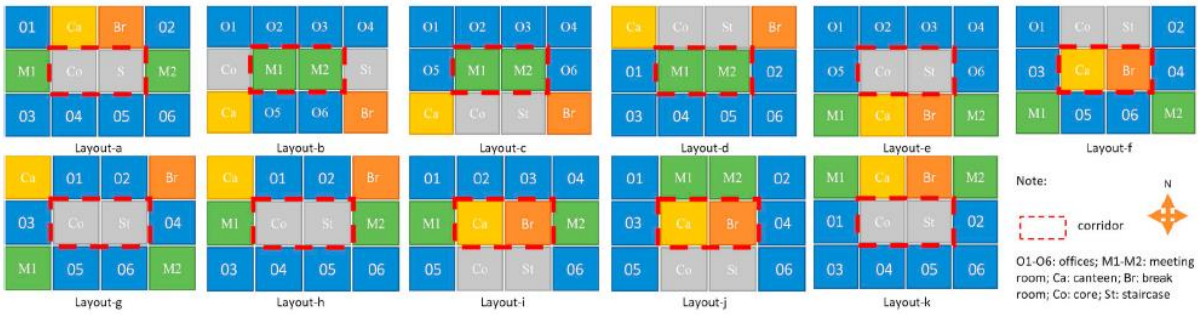


Figure 9: Les différents scénarios d'agencement des espaces étudiés (source: Du et al., 2020)

L'effet de l'agencement diffère selon les climats et les masques solaires. La différence maximale entre les différents scénarios est de 8 % avec ou sans masques solaires avec le climat de la ville d'Amsterdam, en prenant en compte le chauffage, le refroidissement et l'éclairage. Pour l'éclairage seul, la différence peut monter à 48 % sans masque solaire avec le climat de la ville d'Harbin. Le chauffage et le refroidissement sont moins impactés que l'éclairage par la disposition des espaces intérieurs.

Les phases amonts de conception

Il a été démontré que les phases amonts de conception sont des étapes essentielles pour assurer la performance énergétique et environnementale des projets. La courbe de Mac Leamy présentée en Figure 10 (Ilozor & Kelly, 2012) est particulièrement explicite quant à la nécessité de concentrer les efforts du projet sur ces phases initiales afin d'optimiser à la fois la qualité fonctionnelle du projet et ses coûts.

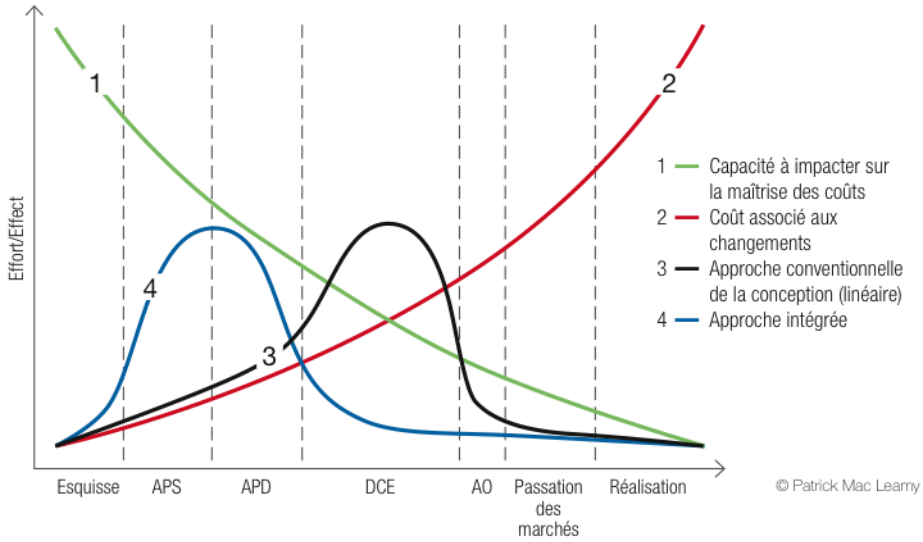


Figure 10: La courbe de Mac Leamy (source : Ilozor & Kelly, 2012)

En effet, lors de ces phases amonts, les architectes jouent un rôle prépondérant (Lu et al., 2015). Elles sont déterminantes du point de vue créatif pour un projet d'architecture. Ces moments de conception, essentiels à la qualité architecturale d'un projet, se caractérisent par des phases de « *doute exploratoire* », où l'approche intuitive des architectes se mêle plus difficilement aux méthodes calculatoires des ingénieurs.

Outils et méthodes de conception existants

Les outils de simulation de la performance des bâtiments

L'usage d'outils numériques dédiés que sont les logiciels de simulation de performance des bâtiments (SPB) a été essentiel aux ingénieurs et architectes pour avancer vers une conception plus écologique (Touloupaki & Theodosiou, 2017). La simulation est une méthode qui cherche à reproduire le comportement physique des systèmes. C'est la méthode la plus utilisée pour évaluer la performance énergétique des bâtiments (Solmaz, 2019). Elle est utilisée pour évaluer le confort thermique et visuel, les besoins et consommations énergétiques, ou

encore pour faire des analyses environnementales, autrement appelées des analyses de cycles de vie (ACV). Il y a deux types de simulation : statique et dynamique. La plupart des outils utilisés aujourd'hui utilisent la méthode dynamique (STD) qui donne des résultats plus précis. Il existe de nombreux outils pour faire de la simulation environnementale disponibles sur le marché (*Building Energy Software Tools*, s. d.). La plupart des outils de simulation ont été développés à l'origine par des chercheurs. Bien que concernés par la performance environnementale de leurs projets, les architectes sont peu familiers avec les outils de simulation (Punjabi & Miranda, 2005).

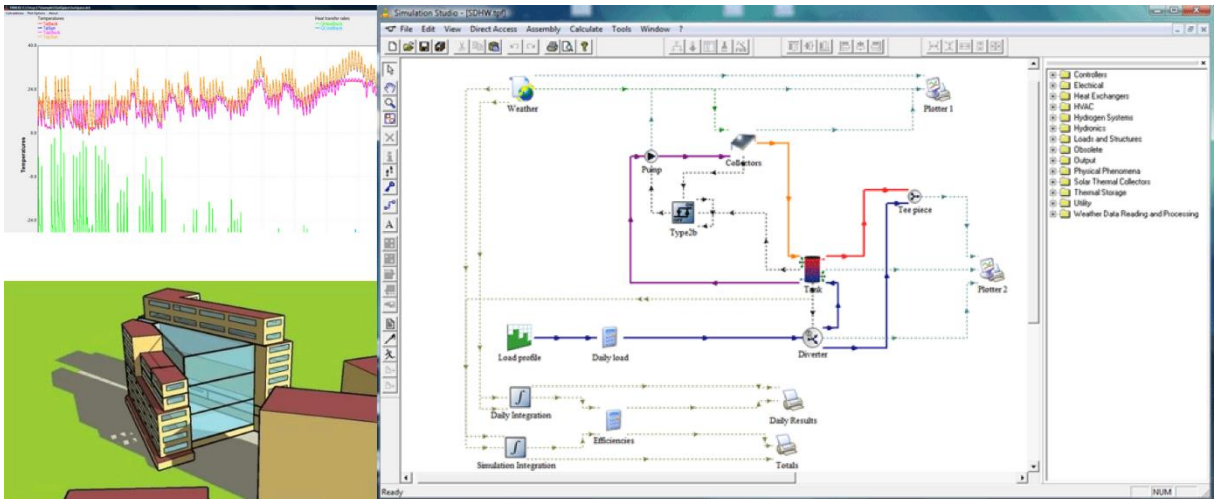


Figure 11: Interface du logiciel de STD TRNSYS (source: https://www.trnsys.com/)

Ces logiciels, aux interfaces souvent opaques et difficiles d'accès (voir Figure 11 l'exemple du logiciel TRNSYS) sont complexes, et les architectes ne sont pas formés pour pouvoir se les approprier facilement (Weytjens et al., 2011).

En 2009, Attia et ses co-auteurs (Attia et al., 2009) ont interrogés 249 architectes américains utilisant des outils SPB sur leurs pratiques de ses logiciels. La moitié des répondants étaient relativement connaisseurs du sujet puisqu'ils détenaient une accréditation LEED. Les outils de simulation préalablement sélectionnés par les auteurs sont ceux considérés comme les plus utilisés par les architectes aux Etats-Unis. Il s'agit d'ECOTECT (qui n'est plus commercialisé depuis 2015), HEED, Energy 10, Design Builder (DB), eQUEST, GREEN Building Studio (GBS), IES VE, EnergyPlus SketUp plug-in (EPSU), DOE-2 et EnergyPlus (EP). Comme le montre le graphique en Figure 12, les outils considérés par les répondants comme les plus adaptés aux phases amonts sont aussi les moins utilisés (GBS, E10, HEED, DB). Ceux considérés comme dédiés aux phases conceptuelles et de développement sont les

plus utilisés (Ecotect, eQUEST et IES VE). Les autres (DOE-2, EP et EPSU) sont considérés comme adaptés aux phases avals (développement et optimisation du projet). D'après les répondants de cette étude, parmi les éléments essentiels pour qu'un outil SPB soit adapté aux architectes on trouve la présence de :

(1) Une représentation graphique des résultats,

(2) Des rapports comparatifs pour plusieurs alternatives,

(3) Des lignes directrices pour la conformité aux réglementations et aux systèmes d'évaluation, et

(4) Des analyses énergétiques rapides pour appuyer la prise de décisions.

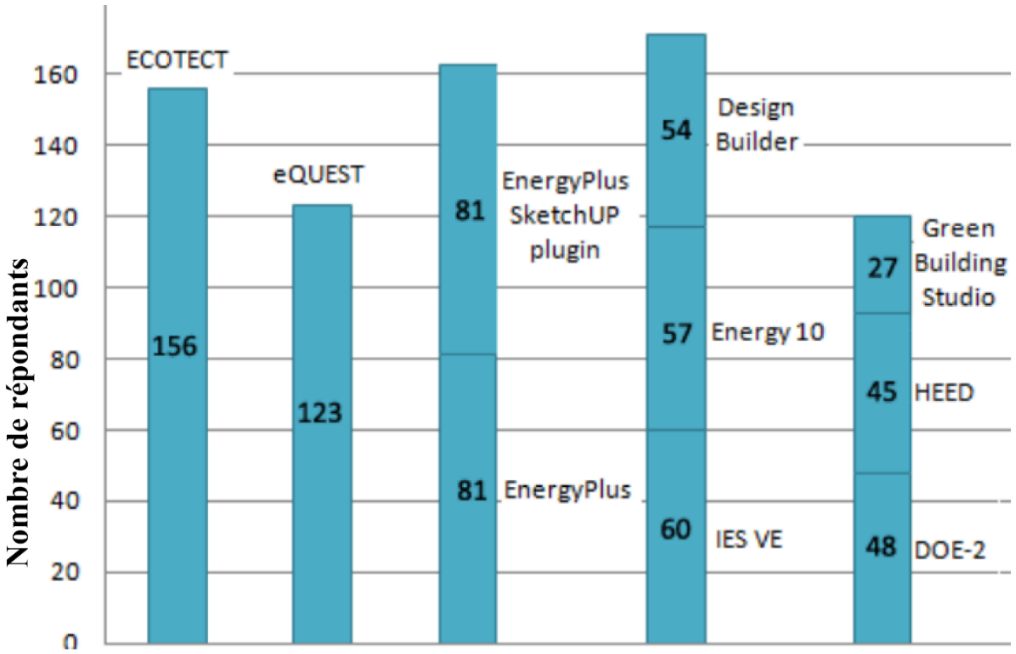


Figure 12: Outils SPB utilisés par les répondants (source: Attia et al., 2009)

Dans une étude comparative datant de 2019 (Solmaz, 2019), Aslihan Solmaz analyse les points forts et points faibles de 9 outils de simulation couramment utilisés et validés scientifiquement. Parmi eux, trois requièrent des connaissances poussées en physique du bâtiment et simulation (EDSL-Tas ; EnergyPlus et ESP-r), ce que n'ont pas les architectes. Trois autres nécessitent des informations précises sur le bâtiment (comme les systèmes CVC) pour créer le modèle (eQUEST, OpenStudio et TRNSYS), ce qui n'existe pas toujours lors des phases amonts de projet. Un autre, Green Building Studio, est décrit comme peu adapté à la conception architecturale car incapable de générer des modèles de bâtiment complexe. Ainsi, IES-Virtual Environnement (IES VE) et Design Builder sont décrits comme étant les plus adaptés à la pratique des architectes.

Les méthodes de conception actuelles

Dans la pratique, les agences d'Architecture se sont construits leurs propres outils adaptés aux phases amonts de conception, notamment pendant la phase d'esquisse où l'usage de l'informatique est le plus souvent exclu, que ce soit pour le dessin d'architecture ou l'analyse environnementale. En effet, en France, les architectes n'utilisent pas d'outils SPB, la tâche est déléguée aux ingénieurs lors des phases de développement du projet où les simulations réalisées sont très détaillées. Durant ces phases, l'activité de conception architecturale peut être schématisée comme un processus itératif entre les équipes d'architectes et d'ingénieurs (voir Figure 13). L'esquisse modélisée numériquement par les architectes sera évaluée par les ingénieurs à l'aide d'outils de simulation, puis modifiée par l'architecte, avant d'être de nouveau évaluée par l'ingénieur. Toutefois, les délais accordés lors de ces phases ne permettent pas toujours aux équipes de réaliser ces itérations suffisamment tôt dans le processus de conception. Ainsi, les performances exigées peuvent devenir inatteignables. Si l'intervention des ingénieurs est trop tardive, alors la simulation sert plus d'outil de validation que d'outil d'aide à la décision (Attia, 2011).

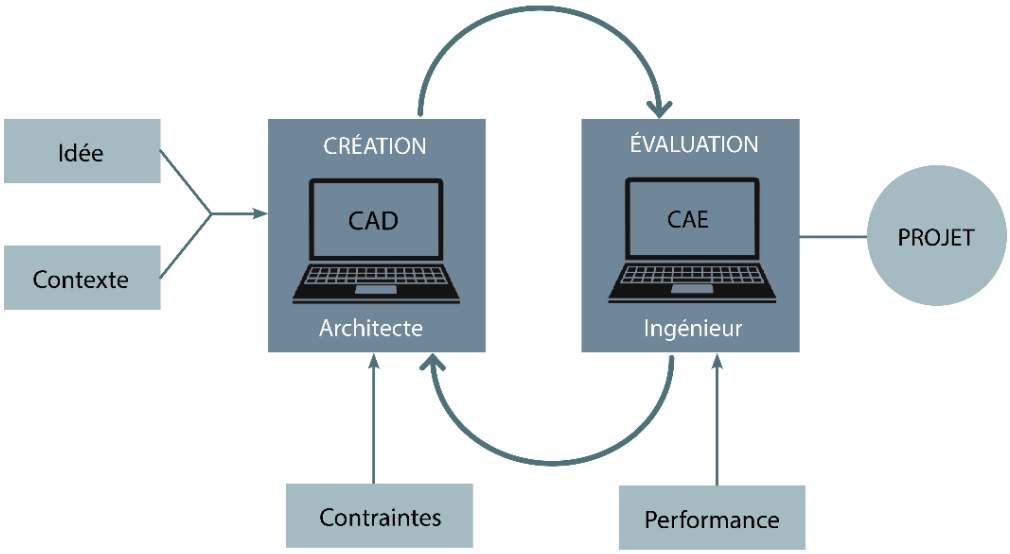


Figure 13: Méthode de conception traditionnelle

Le Building Information Modeling (BIM), présenté comme incontournable dans la profession permet la création de maquettes paramétriques 3D informées supposées faciliter la collaboration entre les acteurs de la conception, notamment l'évaluation de la performance environnementale. En réalité, dans la plupart des agences d'architecture, les outils numériques utilisés se bornent à l'aide à la représentation (Marsault, 2018) et à la collaboration, mais

rarement à l'aide à la conception lors des phases amonts. Lors d'une enquête réalisée en 2020 auprès de 892 agences d'Architecture française (Hochscheid & Halin, 2020), il a été montré que la majorité des agences n'utilise pas de BIM, car elle considère cette démarche trop complexe et trop coûteuse. Certains ont même testé puis abandonné le BIM (7 % des répondants). AutoCAD reste le logiciel le plus utilisé par les agences d'architecture en France (voir Figure 14).

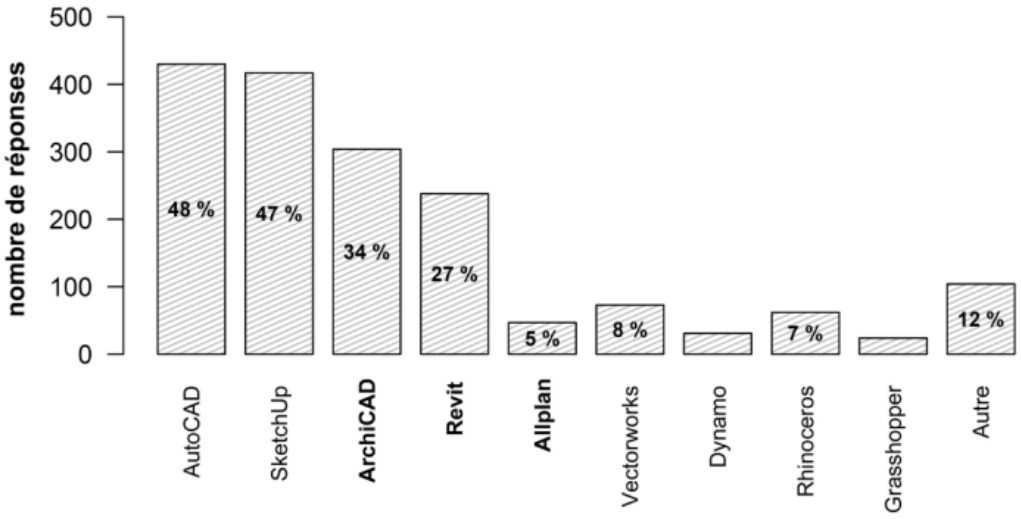


Figure 14: Logiciels de Conception Asistée par Ordinateur utilisés par les agences (source: Hochscheid &Halin, 2020).

Sur les 42 % des agences qui utilisent le BIM, près des deux tiers ne font pas d'échanges avec leurs partenaires. Deux tiers des architectes interrogés estiment que le BIM n'est pas adapté aux phases de conception, voir pour un tiers qu'il n'est pas du tout adapté au métier d'architecte. D'autres études scientifiques ont aussi montré que la démarche BIM reste peu adaptée aux phases amonts de conception (Gouezou, 2018). Il est donc difficile de considérer aujourd'hui en France que le BIM permette de faciliter le processus itératif entre architectes et ingénieurs et donc de faciliter la prise en compte des problématiques environnementales en phases amonts de projet.

Dès lors, apparaît très nettement la nécessité de fournir aux architectes des outils de simulations adaptés aux phases d'esquisses qui soient compatibles avec leurs méthodes de conception.

L'usage d'outils numériques par les architectes lors des premières esquisses permettrait de faciliter une collaboration en aval avec les ingénieurs afin de pouvoir, à tout moment, confronter les décisions aux contraintes environnementales. Il existe donc un véritable besoin

d'identifier ou d'élaborer de nouvelles méthodes et outils adaptés aux phases amonts de conception.

1.1.2. L'Exploration Numérique de solutions Architecturales basée sur des critères de Performances Environnementales

Depuis la fin des années 1990, sont décrites dans une littérature scientifique presque exclusivement anglo-saxonne, des méthodes innovantes permettant de prendre en compte les paramètres environnementaux en phase amont de projet en s'appuyant sur la puissance de calcul des ordinateurs.

Les noms donnés à ces méthodes diffèrent selon les auteurs : « *generative exploration* » (Marin et al., 2008), « *integreted environnemental design* » (Bechthold et al., 2011), « *performative generative architectural design* » (Grobman, 2011), *« parametric systems for performative design exploration* » (Gursel Dino, 2012), « *performance-based parametric design explorations* » (Ercan & Elias-Ozkan, 2015), ou encore « *performance-driven design using an evolutionary multi-objective optimization approach* » (F. Fathy & Fareed, 2017), « *computational design exploration » (Yang et al., 2018)*». Malgré cette grande disparité sémantique, ces méthodes décrivent toutes des processus d'exploration numérique de solutions architecturales fondés sur des critères de performance environnementale (ENAPE).

Le paradigme de la conception computationnelle

Les terminologies employées par les praticiens et les chercheurs divergent tellement lorsque l'on parle de ce type d'approche que Caetano et ses co-auteurs (Caetano et al., 2020) viennent à parler de « *paradigme de la conception computationnelle* ». Ainsi, avant de décrire ce que nous entendons par ENAPE, nous commencerons par définir les différents termes issus de l'anglais que sont la « *conception computationnelle* », la *« conception paramétrique* », la « *conception générative* », la « *conception intégrée* », la « *conception environnementale* » et enfin « *la conception performancielle* ».

La conception computationnelle (CC)

Si le terme existe depuis les années 1950, il ne devient récurant dans la littérature scientifique qu'à partir de la fin des années 1990 (voir Figure 15) et sa définition peut varier selon les auteurs. Toutes les méthodes de conception impliquant l'usage d'un ordinateur, comme lorsqu'on utilise un logiciel de conception assistée par ordinateur (CAO), ne sont pas

considérées comme des méthodes de conceptions dites « *computationnelles* ». La conception computationnelle implique un usage autre que l'aide à la représentation.

A partir d'une revue de littérature exhaustive, Caetano et ses co-auteurs (Caetano et al., 2020) décrivent la conception computationnelle comme :

« Un processus de conception qui tire parti des capacités de calcul de l'ordinateur pour les activités suivantes :

(1) l'automatiser des procédures de conception […],

(2) paralléliser certaines taches de conception et la gestion efficace d'une grande quantité d'informations,

(3) intégrer des modifications de façon rapide et flexible,

(4) assister l'architecte avec des processus de recherche formelle avec feedback automatique comme la cartographie de résultats de simulation. » (traduit de l'anglais).

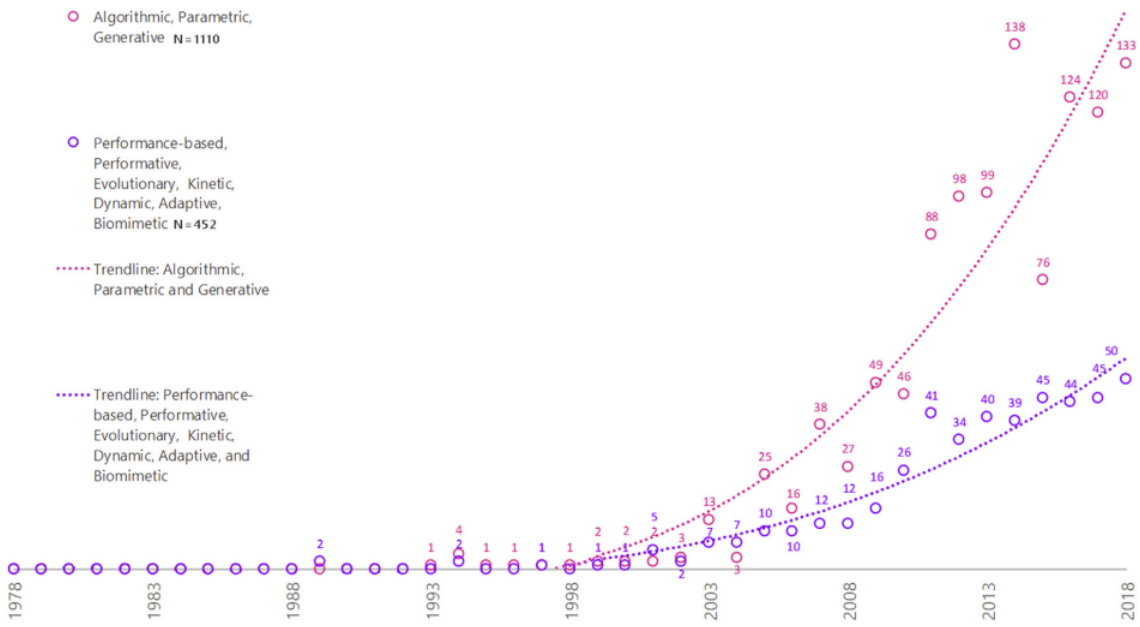


Figure 15: Fréquence d'utilisation des mots clés relatifs à la CC entre 1978 et 2017 (source: Caetano et al., 2020)

La conception paramétrique (CPA)

En architecture, la conception paramétrique est souvent associée aux outils numériques comme *Grasshopper*® sur *Rhinocéros*® ou encore *Dynamo*® sur *Revit*® (voir Figure 16). Ces outils sont en réalité des langages de programmation visuelle (Celani & Vaz, 2012) créés pour rendre la programmation informatique accessible à des non spécialistes tel que les architectes.

Ils permettent de coder dans les logiciels de modélisation pour lesquels ils ont été créés, de la même façon qu'il est possible d'utiliser le langage Visual Basic for Applications (VBA) pour coder dans Excel ou Word. Ainsi, utiliser *Grasshopper*® ou *Dynamo*®, ne veut pas forcément dire « *faire de la conception paramétrique* ». On peut utiliser *Grasshopper*® pour faire de l'analyse environnementale par exemple, ou *Dynamo*® pour gérer les données d'une maquette numérique.

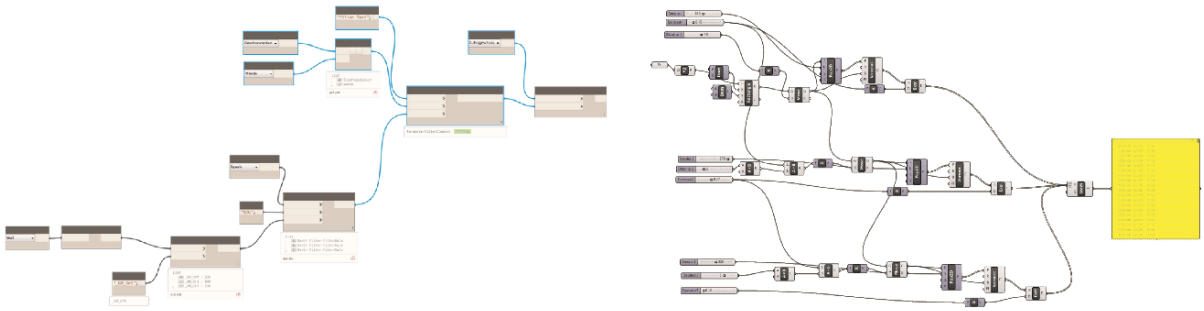


Figure 16: Capture d'écran d'un script Dynamo à gauche et d'un script Grasshopper à droite.

Selon la définition de l'encyclopédie Wikipédia :

« La conception paramétrique est un mode de fonctionnement des logiciels de conception assistée par ordinateur actuels. Il s'agit de définir une entité par des paramètres qui peuvent être modifiés facilement. »

Selon cette définition, dès qu'il y a un système de paramètres, on peut parler de conception, ou modélisation paramétrique. Ainsi, les outils BIM comme *Revit*® avec son système de familles est un logiciel de conception paramétrique sans même avoir besoin d'utiliser *Dynamo*®. La conception paramétrique n'est pas réservée à l'architecture. De nombreux logiciels de CAO ont un mode de fonctionnement paramétrique (*CATIA*®, *FreeCAD*®, *Top Solid*®, *SALOME*®).

La conception générative (CG)

De nouveau, la définition de la conception générative est variable selon les auteurs. Il est certain que celle-ci diffère de la conception paramétrique (voir Figure 17), et nécessite des algorithmes bien plus complexes, au point que certains parlent d' « *algorithmes non-linéaires* » par opposition aux « *algorithmes linéaires* » de la conception paramétrique (Ma, 2015a). Un processus génératif est un processus itératif où une instruction est exécutée en boucle et s'arrête lorsqu'un critère précis est satisfait (Caetano et al., 2020). Il existe de nombreuses techniques génératives notamment des techniques bio-inspirées. Certains auteurs parlent de « *systèmes naturels* » (Harding & Shepherd, 2017). Les processus génératifs peuvent générer des résultats complexes à partir de règles simples. Ils sont « *souvent destinés à faciliter l'exploration* »

(Singh & Gu, 2012). L'une des grandes caractéristiques des techniques génératives est leur capacité d'émergence (Stiny, 1994), c'est-à-dire la capacité qu'ont ces techniques de générer des formes qui n'avaient pas été préfigurées ou imaginées par le concepteur. Les algorithmes d'optimisation évolutionnaires très utilisés pour faire de l'exploration de solutions sont souvent catégorisés comme techniques génératives (Singh & Gu, 2012) et sont probablement les plus employés en architecture.

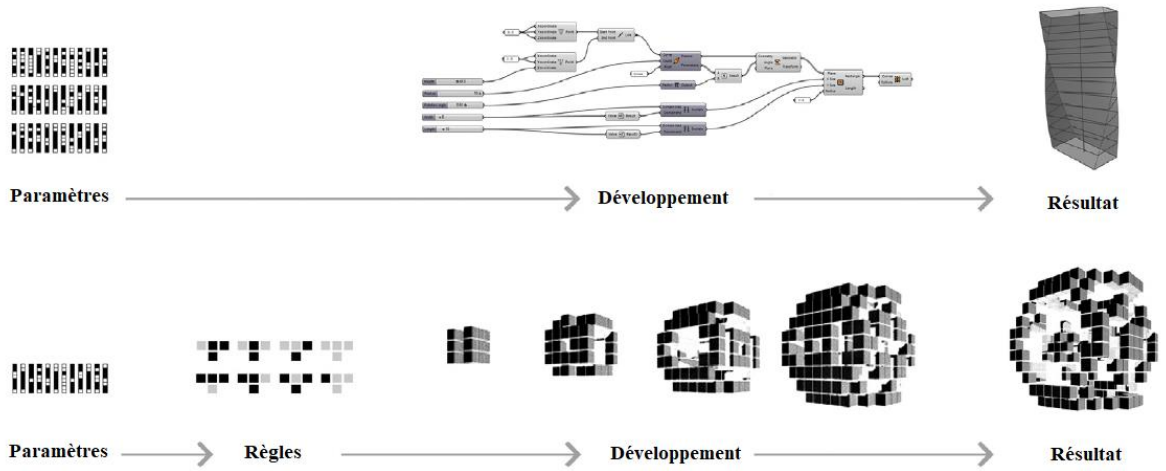


Figure 17: Comparaison entre un modèle paramétrique et un modèle génératif (source: Harding & Sheperd, 2017)

La conception intégrée (CI)

Il s'agit d'une approche holistique de la conception (Papanek, 1972) où les échanges entre les différents acteurs de la construction sont favorisés dès les premières phases du projet pour créer plus de synergie afin d'optimiser les objectifs du projet (architecture, ingénierie, environnement, budget, planning, etc.). La conception intégrée n'est pas un terme spécifique à l'architecture, il peut être employé pour la conception de tout type de produits. Ce terme ne renvoie pas à une méthode de conception en particulier mais plutôt à une philosophie de conception. Bien que l'objectif reste souvent de construire des bâtiments plus durables, le terme « *conception intégrée* » ne cible pas à l'origine spécifiquement la question des contraintes environnementales.

La conception environnementale (CE)

Le terme « *conception environnementale* », littéralement traduit de l'anglais « *environnemental design* », reste peu employé dans la littérature scientifique francophone. On lui préfère le terme d' « *éco-conception* » (Peuportier, 2008), ou encore d' « *architecture écologique* » (Gauzin-

Müller, 2001) ou d' « *architecture durable* » (Hoyet, 2020). Selon le Ministère de la transition écologique et de la cohésion des territoires (*L'éco-conception des produits*, 2019),

« L'éco-conception consiste à intégrer la protection de l'environnement dès la conception des biens ou services. Elle a pour objectif de réduire les impacts environnementaux des produits tout au long de leur cycle de vie : extraction des matières premières, production, distribution, utilisation et fin de vie. ».

La conception performancielle (CPE)

Le terme « *conception performancielle* » est absent de la littérature francophone. Il vient de l'anglais « *performance design* » or « *performative architecture* » (Kolarevic & Malkawi, 2005). En architecture performancielle, la performance n'est pas nécessairement environnementale, elle peut par exemple être structurelle (Oxman, 2009). Cependant, elle implique l'usage d'outils digitaux, elle est donc toujours computationnelle. Selon Oxman (Oxman, 2006),

« Dans la conception basée sur la performance, l'objet est généré en simulant sa performance». (traduit de l'anglais)

Oxman explique qu'il existe deux types d'approches performancielle. Une première approche nommée « *formation based design* » lorsque la simulation est utilisée pour piloter le processus de conception, et une seconde impliquant l'usage d'un processus génératif.

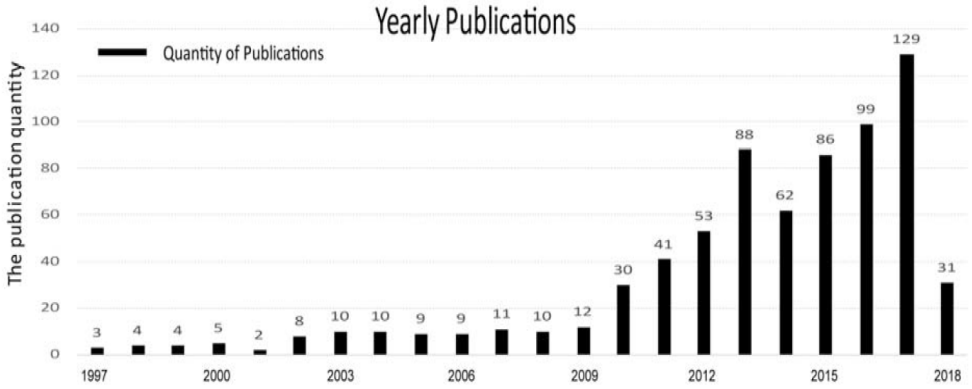


Figure 18: Nombre de publications par année sur les processus d'exploration fondés sur des critères de performances (source: S. Zhao et E. de Angelis, 2018)

Le développement des langages de programmation visuelle au milieu des années 2000 avec la création de *Grasshopper*® en 2007 a permis, comme le montre la Figure 18 (Zhao & de Angelis, 2018), une démultiplication du nombre d'études et de publications scientifiques traitant à la fois de conception paramétrique et de performance faisant ainsi de la *« conception computationnelle et environnementale »* un champ de recherche à part entière (Touloupaki & Theodosiou, 2017).

Définition et fonctionnement général d'une méthode ENAPE

Définition et modélisation de l'ENAPE

L'ENAPE est une méthode de conception computationnelle à la fois générative et performancielle où l'un des critères au moins est environnemental, impliquant l'usage de la conception paramétrique, et permettant potentiellement de faire de la conception intégrée (voir Figure 19). Pour faire de l'ENAPE, il s'agit d'élaborer un script pour un problème de conception architectural donné permettant de générer une grande quantité de solutions évaluées selon un ou plusieurs critères de performance environnementale qu'il convient alors d'explorer pour en déduire un optimum.

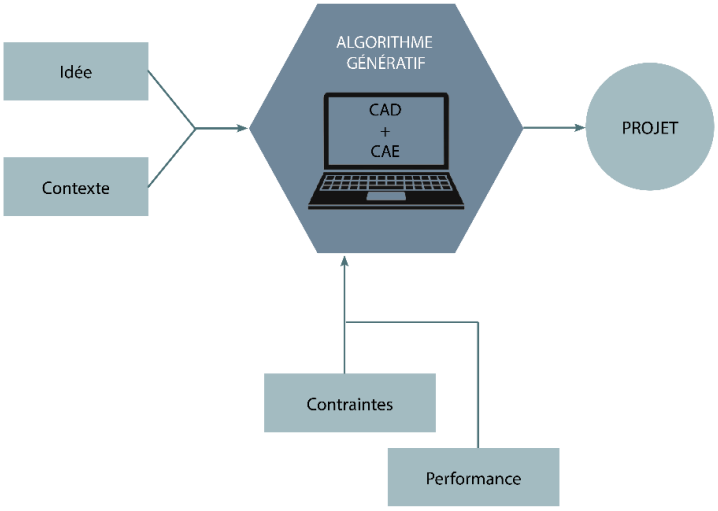


Figure 20: Méthode de conception générative et performancielle

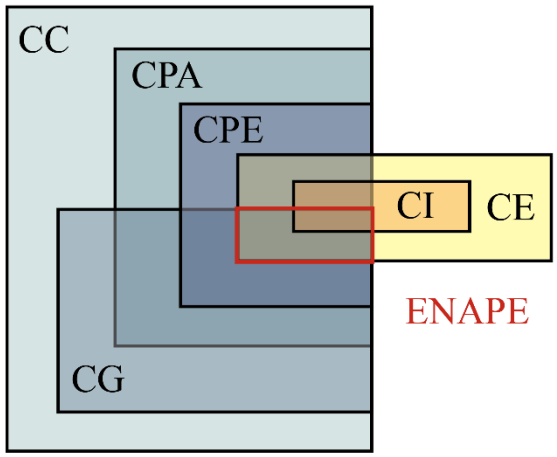


Figure 19: Diagramme des terminologies de la conception computationnelle

Cette méthode cherche à reproduire automatiquement et rapidement, et dans des quantités beaucoup plus importantes, le processus itératif qui existe entre architectes et ingénieurs dans la méthode de conception traditionnelle schématisée en Figure 13. Une méthode d'ENAPE est un algorithme génératif dans lequel les outils de modélisation des architectes sont directement reliés aux outils de simulation des ingénieurs.

Comme le décrit la Figure 21 (Sharaidin et al., 2012), pour modéliser informatiquement un processus d'ENAPE, trois sous-modèles sont nécessaires. En premier lieu, il est nécessaire d'élaborer un modèle qui permet de créer différents scénarios de projet, que l'on nomme le modèle génératif. Dans la grande majorité des cas décrits dans la littérature, il s'agit d'un modèle paramétrique simple.

Ensuite, l'utilisateur met en place un mécanisme d'évaluation pour relier le modèle paramétrique à des logiciels existants ou à des méthodes simplifiées d'évaluation.

Enfin, si l'ensemble de solutions ne peut être entièrement exploré dans un délai raisonnable, un algorithme d'optimisation peut être utilisé à l'aide d'un solveur d'optimisation comme *Galapagos*® (Rutten, 2011) pour déterminer la combinaison de paramètres permettant d'approcher un optimum global.

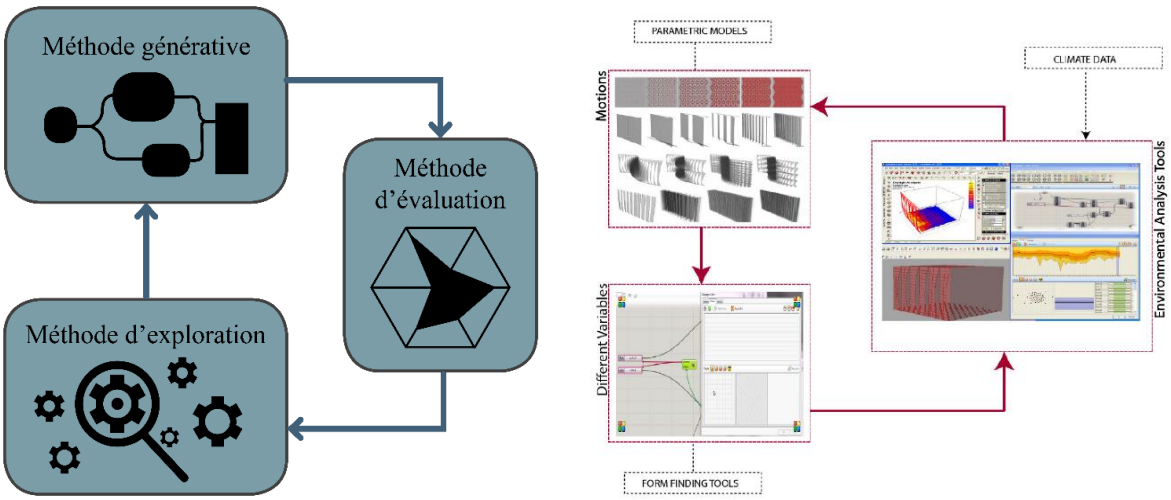


Figure 21: A gauche le fonctionnement général d'une méthode d'exploration, à droite un exemple d'application (source: Sharaidin et al., 2012)

Les cinq étapes de mise en pratique de l'ENAPE

Si la modélisation est constituée de trois sous-modèles interfacés informatiquement, la mise en pratique s'organise en cinq étapes présentées en Figure 22 (Shen et al., 2018).

(1) Une première étape (« analyser ») d'analyse est essentielle pour formuler le problème (Zhao & de Angelis, 2018). Elle comprend notamment une analyse des exigences réglementaires et des exigences de la maîtrise d'ouvrage, mais aussi des données météo. Cette analyse permet de définir les critères d'évaluation, de distinguer les paramètres qui resteront figés de ce qui peuvent varier (les variables). Cette étape cruciale est réalisée parallèlement à la recherche d'une première idée architecturale, mais ne peut complètement la précéder. En effet, la démarche ne peut pas être mise en place à partir d'une feuille blanche, il faut un début d'esquisse, un premier concept architectural. Tous les problèmes que posent un projet d'architecture ne peuvent être résolus à l'heure actuel avec une unique exploration. Formuler le problème revient donc parfois à le décomposer en sous problème.

(2) La seconde étape (« prototyper ») consiste à élaborer le modèle génératif qui permet de définir un ensemble de solutions à explorer appelé « *espace de solutions* » ou

« *espace de recherche* ». Sa forme la plus simple est un modèle paramétrique (El Sheikh & Gerber, 2011 ; Ercan & Elias-Ozkan, 2015 ; F. Fathy & Fareed, 2017), mais il existe d'autres techniques génératives spécifiques qui peuvent être utilisées comme des grammaires de formes (Caldas, 2008) ou des automates cellulaires (Kim, 2015). Cette étape permet déjà de visualiser des représentations 3D de solutions possibles pour le projet. Elle peut servir à affiner un concept esthétique (Hudson et al., 2011) et à restreindre la taille de l'espace de recherche en bornant les paramètres pour des questions architecturales, réglementaires ou fonctionnels.

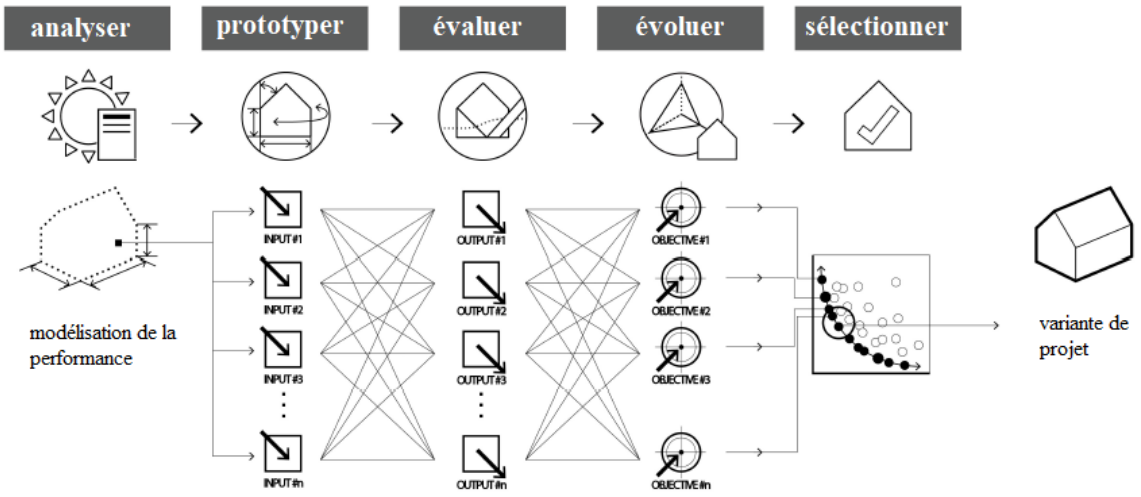


Figure 22: Les cinq étapes de la mise en pratique d'un processus d'ENAPE (source: Shen et al., 2012)

(3) La troisième étape (« évaluer ») est celle de l'élaboration du modèle d'évaluation. Les critères d'évaluation doivent se préciser sous la forme d'indicateurs. Le choix des indicateurs dépend de la phase de conception, du degré de complexité géométrique du modèle 3D à analyser, des réglementations selon la localisation des projets, mais aussi des délais, car les temps de calcul diffèrent selon les indicateurs qu'on souhaite évaluer. De nouveaux paramètres peuvent aussi être intégrés lors de cette phase comme la matérialité. Cette étape peut impliquer une phase d'analyse de sensibilité (Kheiri, 2018) qui sert à déterminer les paramètres qui ont le plus d'impact sur les critères d'évaluation. Enfin, chacun des critères qui sera pris en compte dans l'exploration doit être exprimé sous la forme d'une unique valeur numérique. Il s'agit de la « *fitness function* », ou « *fonction d'évaluation* » en français.

(4) La quatrième étape (« évoluer ») est celle de l'exploration qui peut être ou non automatisée. Le terme « évoluer » fait référence aux algorithmes d'optimisation évolutionnaires souvent utilisés pour faciliter l'exploration de solution. Nous verrons qu'il existe plusieurs algorithmes à disposition des architectes sur des plateformes comme *Grasshopper*® (Dissaux, 2018) qui restent abordables pour les non-mathématiciens. Cette étape consiste principalement à laisser tourner l'algorithme, sachant que les temps de calcul peuvent se compter en journées (Ercan & Elias-Ozkan, 2015).

(5) La dernière étape (« sélectionner ») est l'étape de sélection d'une solution. Elle peut être chronophage notamment dans le cas d'une exploration impliquant plusieurs critères (fait systématique dans la pratique professionnelle). Dans ce cas, il n'y a pas une solution optimale, mais bien plusieurs solutions qui se démarquent. Il convient alors de les visualiser pour pouvoir les comparer et sélectionner une solution parmi l'espace de recherche. Plus généralement, l'ENAPE permet de générer une grande quantité de données qui, si elles sont correctement analysées, peuvent aider le concepteur à mieux comprendre les corrélations entre paramètres architecturaux et critères environnementaux restés jusqu'alors invisibles pour l'œil de l'architecte.

Les différents types d'outils numériques nécessaires à l'ENAPE

Les outils d'analyses

Pour explorer et ainsi comparer des centaines, voire des milliers de scénarios différents, il est indispensable d'être capable de les évaluer quantitativement et rapidement. On utilise alors des « *critères d'optimisation* » ou « *critères d'évaluation* » que l'on évalue à l'aide d'outils d'analyse. Un critère d'optimisation est un paramètre à prendre en compte dans le processus de conception que l'on souhaite intégrer « *du mieux possible* », de « *manière optimale* ». Il est à distinguer d'une contrainte (notion que nous détaillerons plus tard) qui est un paramètre devant impérativement être respecté et qui se formule différemment dans un processus d'ENAPE.

Les critères que peuvent rencontrer les architectes qui souhaitent faire de l'ENAPE ne sont pas systématiquement des critères environnementaux parce qu'il existe beaucoup d'autres paramètres qui entrent en compte dans la conception pouvant être en contradiction avec ces critères comme des critères économiques, fonctionnels, structurels ou esthétiques. Les méthodes d'analyse diffèrent selon le type de critères. Nous distinguons 3 grandes catégories

d'outils d'analyses : (i) l'analyse géométrique, (ii) les outils de simulation et (iii) les méthodes simplifiées.

(i) L'analyse géométrique

Des fonctions déjà intégrées aux outils de programmation visuelle permettent de faire de l'analyse de la géométrie. Il y a notamment les fonctions qui permettent de faire des analyses quantitatives comme le calcul de surfaces, les mètres linaires et volume de matériaux, les distances entre des éléments. Cela permet par exemple de définir des critères économiques comme le coût de construction lié aux matériaux, ou des critères fonctionnels comme la distance entre deux espaces. Ces critères sont alors calculés « *en temps réel* », c'est-à-dire que les temps de calcul sont tellement courts qu'ils sont à peine perceptibles pour l'utilisateur. Il est aussi possible d'utiliser une fonction de lancer de rayons pour créer ses propres indicateurs comme pour l'évaluation de la qualité de la vue. Les temps de calcul peuvent être un peu plus long selon le nombre de rayons mais restent acceptables pour faire de l'exploration.

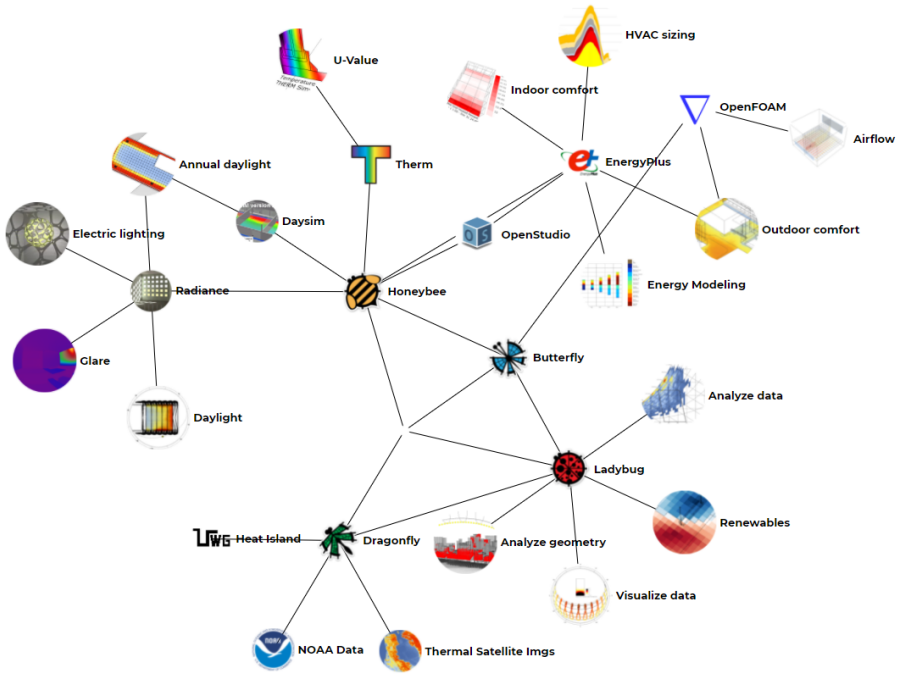


Figure 23: Ecosystème des plugins Ladybug (source: Ladybug Tools, 2016)

(ii) Les outils de simulations

Ils peuvent être reliés à un modèle paramétrique avec l'utilisation de *plugins* spécialisés dans l'interopérabilité comme *Ladybug*® et *Honeybee*® (Roudsari et al., 2013) dont l'écosystème est illustré en Figure 23 (Ladybug Tools, 2016). Ces outils

permettent de réaliser des passerelles avec les logiciels de simulation de performance comme *Energy Plus*® pour la simulation thermique dynamique, ou *Radiance*® pour la simulation de l'éclairage. Il existe des outils équivalents pour l'analyse structurelle.

Les simulations utilisées pour faire de l'ENAPE sont à distinguer de celles utilisées pour faire des calculs réglementaires qui servent à valider ou invalider un scénario, et non pas à comparer rapidement des multitudes de *scenarii*. Ainsi le paramétrage des simulations doit être adapté pour réduire au maximum les temps de calcul sans que cela n'augmente trop significativement les marges d'erreur des analyses, ce qui implique de s'écarter des exigences réglementaires. Les temps de calculs peuvent être de quelques minutes par simulations, au-delà il faut s'intéresser à d'autres méthodes.

(iii) Les méthodes simplifiées

Pour certains critères, simuler les comportements physiques n'est pas la méthode adaptée car les temps de calculs sont beaucoup trop longs pour faire de l'exploration en phase amont de conception. Dans ce cas il est possible d'utiliser des méthodes simplifiées, encore appelées modèles de substitution, qui demandent d'utiliser des algorithmes de *Machine Learning* comme les modèles de régression (Catalina et al., 2009; Nault et al., 2016) ou des modèles de *Deep Learning* (Singaravel et al., 2018) pour prédire les consommations énergétiques d'un bâtiment par exemple. Ces techniques peuvent aussi être utilisées pour définir des critères très spécifiques difficiles à quantifier comme le critère esthétique (Y. K. Yi, 2019). Ces techniques impliquent une phase préparatoire, appelée « *phase d'apprentissage* » durant laquelle de véritables simulations sont réalisées pour pouvoir entraîner les algorithmes de *Machine Learning* afin qu'ils puissent, dans une seconde phase, évaluer des scénarios en temps réel.

<u>Les outils d'optimisation</u>

Bien que nous parlions de « *processus d'exploration* », chercher parmi un ensemble de scénarios de projets possibles celui qui serait le mieux adapté pour répondre à une problématique est un « *problème d'optimisation* ». En élaborant un modèle paramétrique dans lequel certaines données géométriques sont fixées et d'autres sont variables, nous définissons ainsi un espace de solution, dont la taille (le nombre de solution) est déjà définie. Rechercher

la meilleure solution dans cet espace revient à faire de l'« *optimisation* », et rechercher les meilleures solutions selon différents critères (qui peuvent être géométrique, environnemental, structurel, esthétique) revient à faire de l' « *optimisation multicritère* ». Pour explorer aisément un espace de solutions, il peut être utile d'utiliser des algorithmes d'optimisation mathématiques notamment lorsqu'on veut faire de l'optimisation multicritère. L'optimisation, en conception computationnelle, est considérée comme une technique générative parmi d'autres (Singh & Gu, 2012).

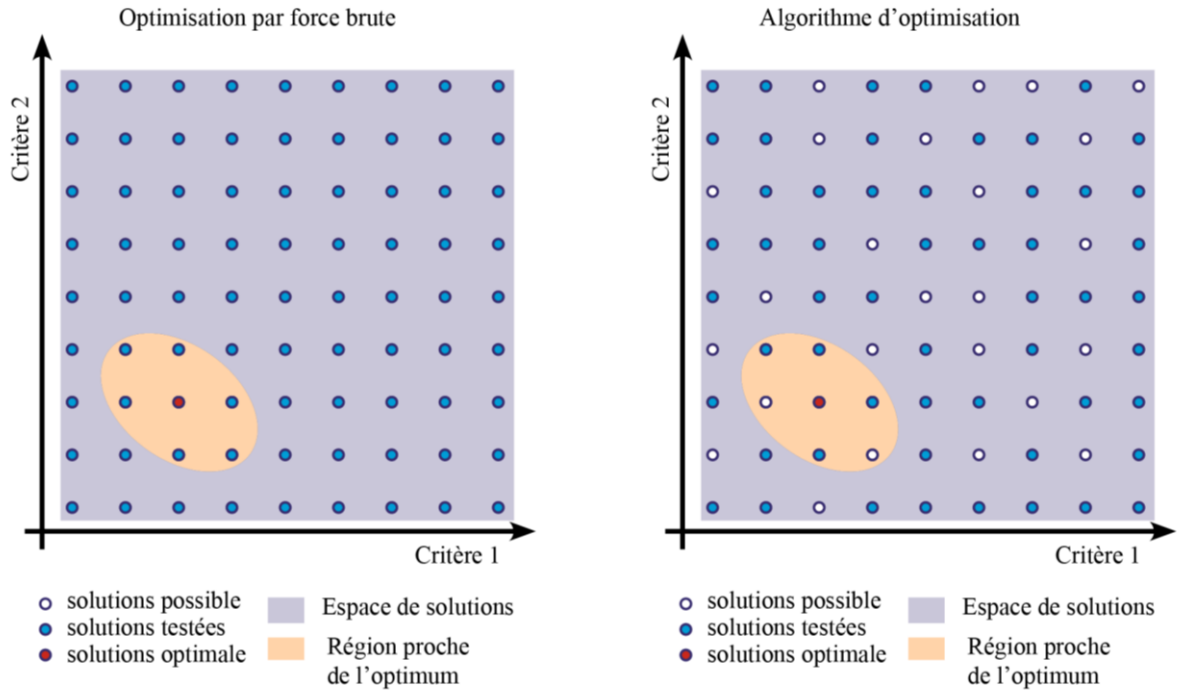


Figure 24: Méthode d'optimisation par force brute versus algorithme mathématique d'optimisation

Lorsque le nombre de solutions est raisonnable, il est possible de toutes les analyser afin de sélectionner la meilleure selon un ou plusieurs critères. On parle alors d'optimisation par force brute ou d'approche exhaustive. Lorsque cet espace de solutions est trop grand, les temps de calcul sont alors trop longs. On fait alors appel à des algorithmes mathématiques afin d'évaluer une partie de l'espace de solutions. Le rôle de ces algorithmes est de chercher dans cet espace les meilleures solutions sans avoir à toutes les analyser (voir Figure 24).

Il existe de nombreux types d'algorithmes d'optimisation, dont l'efficacité diffère selon le type de problème rencontré. De nombreux algorithmes sont à disposition des architectes, principalement sur *Grasshopper*® et *Dynamo*®. Malgré cette diversité, les algorithmes les plus couramment utilisés en Architecture sont les algorithmes génétiques (Nguyen et al., 2014).

Parmi les algorithmes accessibles aux architectes via la programmation visuelle des modeleurs 3D, ils sont les seuls qui permettent de faire de l'optimisation multicritère.

Les algorithmes génétiques (Mirjalili, 2019) sont connus pour être facilement accessibles à des non-mathématiciens, c'est pourquoi ils sont si populaires. Ces algorithmes s'inspirent de la théorie de l'évolution, de l'idée que les espèces évoluent et survivent grâce à la sélection naturelle. Il existe aujourd'hui de nombreux algorithmes génétiques décrits dans la littérature scientifique (Deb et al., 2002; Zitzler et al., 2001) mais leur fonctionnement général suit toujours le même schéma logique. Cette suite d'instructions que l'on nomme « *algorithme* » a été élaborée par un chercheur du domaine, souvent un mathématicien. L'outil qui permet d'utiliser cet algorithme s'appelle un « *solveur d'optimisation* ». Il s'agit d'un outil informatique créé par un développeur qui permet de démocratiser l'usage de l'algorithme à l'aide d'une interface graphique. Un même solveur peut parfois contenir plusieurs algorithmes différents, c'est le cas de *Galapagos®,* le solveur le plus populaire sur *Grasshopper®*.

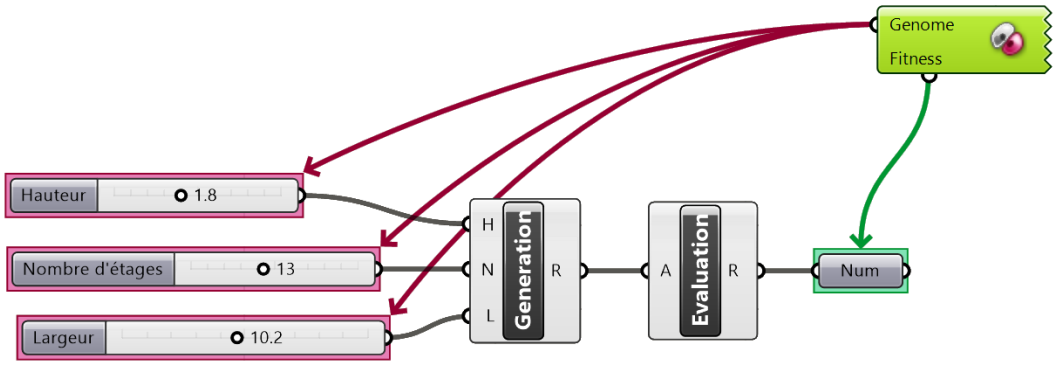


Figure 25: Script *Grasshopper®* pour le fonctionnement du solveur *Galapagos®*.

Comme on peut le voir sur la Figure 25, sur le logiciel de programmation visuelle *Grasshopper®,* deux types de données doivent être connectés au solveur d'optimisation : les variables qui prennent la forme de curseurs (en rouge), et les résultats des évaluations qui prennent la forme d'un nombre réel (en vert). En double cliquant sur le composant *Galapagos*, une première interface permet d'initialiser l'exploration. Une fois l'algorithme lancé, il est possible de suivre l'exploration à l'aide d'un tableau de bord visible sur la Figure 26. Un graphique y est représenté avec la moyenne en rouge, l'écart type en orange et l'ensemble des scores des solutions testées en jaune pour chaque génération. L'historique de toutes les solutions générées est conservé, ainsi chaque solution testée peut être regénérée dans *Grasshopper®*.

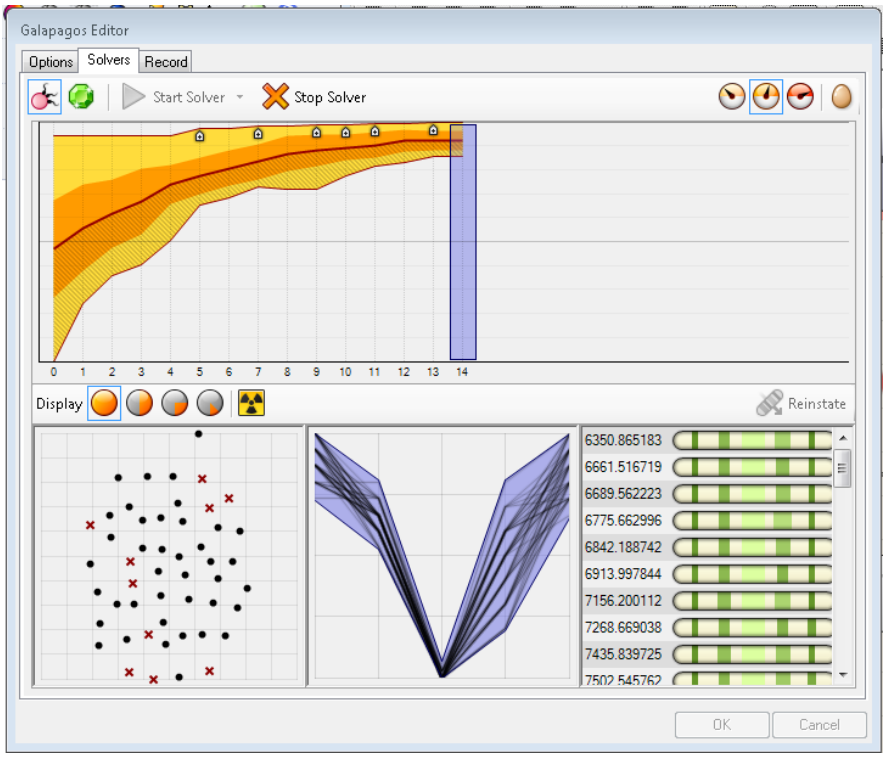


Figure 26: Interface graphique du solveur d'optimisation *Galapagos*® sur *Grasshopper*®.

Lorsqu'on souhaite faire de l'optimisation multicritère, il faut utiliser un solveur spécifique. Le plus populaire sur *Grasshopper*® est le solveur *Octopus*®. Il fonctionne comme *Galapagos*®, seulement il permet de connecter plusieurs valeurs d'évaluations aux composants pour prendre en compte plusieurs critères. Dans ce cas, le tableau de bord est un peu différent. La représentation des résultats est plus complexe car les algorithmes multicritères n'ont pas pour objectif de déterminer une unique solution optimale, mais un ensemble contenant les meilleures solutions qui se trouvent sur le « *front de Pareto* » (Van Veldhuizen & Lamont, 1998). Il s'agit de toutes les solutions dites « *non-dominées* », c'est-à-dire qu'il n'existe pas une autre solution dans l'ensemble des solutions testées qui soit meilleure pour chacun des critères d'optimisation. Comme le montre la Figure 27, il est possible de représenter le front de Pareto avec un graphique en nuage de points en 2D lorsqu'on étudie deux critères, ou en 3D lorsqu'on étudie trois critères.

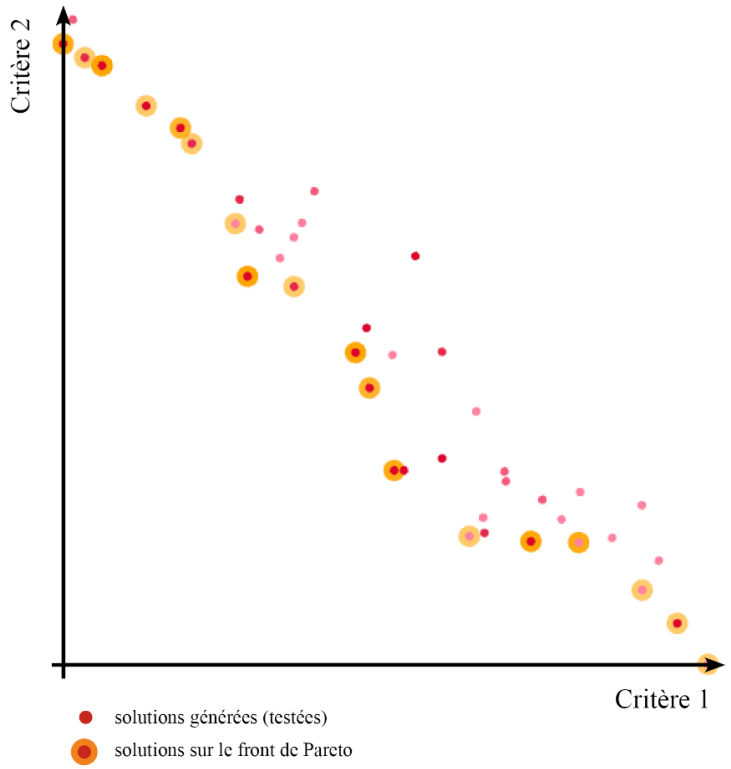


Figure 27: Illustration du Front de Pareto

Les outils de data visualisation

Lorsqu'on fait de l'exploration, il est nécessaire de s'intéresser à l'ensemble des solutions qui ont été analysées et pas uniquement à la solution optimisée. D'abord, parce qu'en optimisation multicritère il n'y a jamais une unique solution mais bien des solutions non-dominées qu'il faut hiérarchiser. Ensuite, parce que d'autres solutions, proches de la solution optimisée ou des solutions optimisées en termes de performances environnementales, pourraient se révéler plus intéressantes selon d'autres critères pas toujours quantifiables (esthétiques…). Enfin, parce que l'analyse de l'ensemble des données pourrait nous permettre de mieux appréhender le projet en révélant des corrélations entre géométrie et performance. Cela peut notamment permettre de définir des règles de conception ou des contraintes de dimensionnement.

Ainsi, il existe des outils numériques de visualisation des données à disposition des architectes qu'on appelle les outils de « *post-traitement* » (Zhao & de Angelis, 2018). Ils permettent de représenter les solutions testées ainsi que leur performance de façon à faciliter le plus possible l'interprétation des résultats. Ces outils utilisent notamment des graphiques rendus plus ou moins interactifs grâce au numérique. Les deux graphiques les plus populaires pour faire de

l’exploration sont : le graphique en nuage de points laissant apparaître le front de Pareto précédemment présenté et le diagramme de coordonnées parallèles.

Le diagramme de coordonnées parallèles (Wegman, 1990) permet de trouver des corrélations entre les paramètres géométriques explorés et les différents critères d’optimisation. Dans un diagramme de coordonnées parallèles, chaque ligne représente une unique solution. Il est possible de visualiser les valeurs de chacune des variables et de chacun des critères utilisés pour l’exploration. En attribuant un dégradé de couleur à une donnée (une variable ou un critère), il est possible de découvrir des tendances générales.

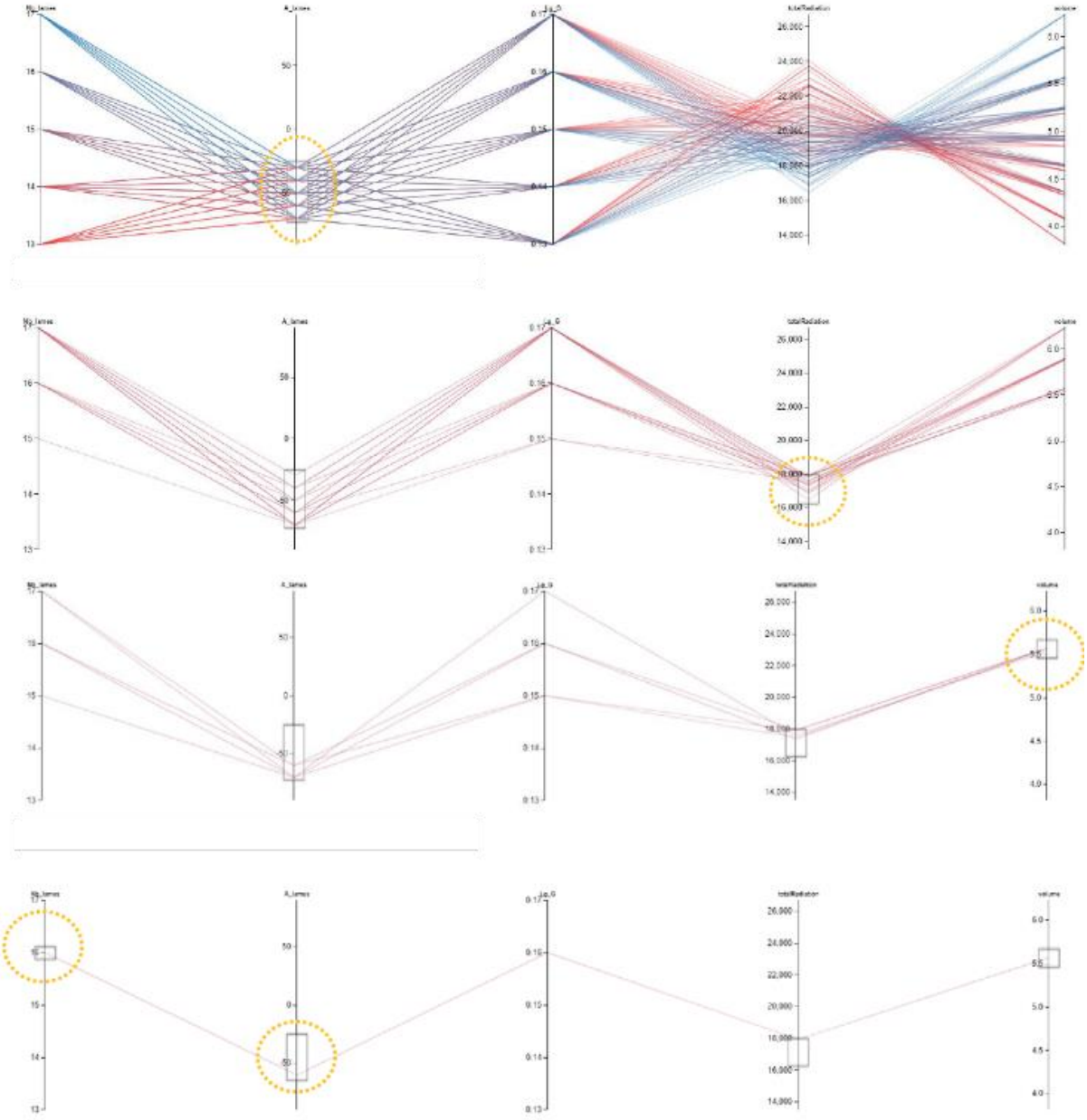


Figure 28: Captures d'écran successives lors de l'utilisation du digramme de coordonnées parallèles interactif de *Design Explorer2®*

Il existe d'autres types de représentation graphiques utilisés pour analyser des résultats d'optimisation. (Attia et al., 2013) ont identifiés les 14 types de représentation les plus utilisés par des experts de l'optimisation interviewés pour leur étude (voir Figure 29):

(1) fitness de la solution, (2) probabilités de dispersion des solutions, (3) gamme de solutions, (4) ligne, (5) espace des solutions (front de Pareto), (6) poids paramétriques, (7) série temporelle, (8) diagramme en barre, (9) arbre des solutions, (10) compromis linéaire, (11) tableau, (12) dendrogramme, (13) diagramme de convergence, et (14) courbe de niveau thermique.

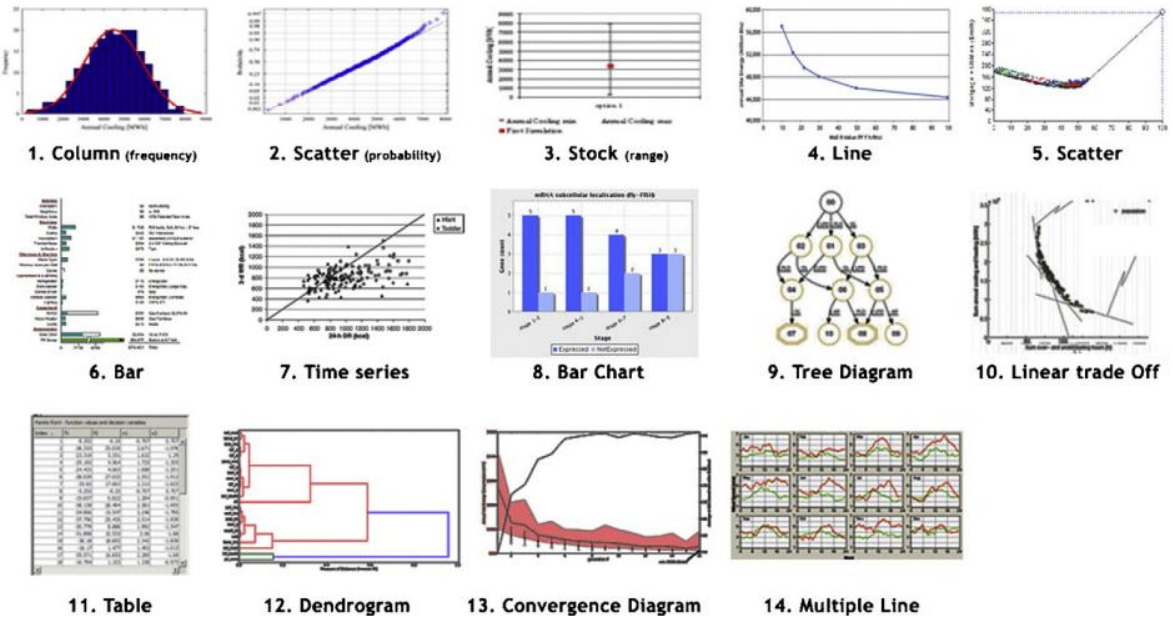


Figure 29: les 14 différentes visualisations pour l'analyse des résultats (source: Attia et al., 2013)

La création de ce type de graphiques réalisables avec des outils comme *Microsoft Excel*® ou *R*® n'est pas une activité familière des architectes. Aussi, il existe des outils de data visualisation, encore appelé « *outils de post-traitement* » qui permettent d'automatiser la génération de ce type de graphiques. Notamment, l'application web *DesignExplorer2*® (Thornton Tomasetti, 2019) donne accès à une version interactive du diagramme de coordonnées parallèles et du graphique en nuage de points. Les graphiques y sont reliés au catalogue des solutions testées qu'il est possible d'explorer grâce à un système de filtres. La Figure 28 montre comment filtrer les solutions avec un diagramme de coordonnées parallèles interactif. Pour pouvoir l'utiliser avec *Grasshopper*®, l'exploration doit être effectuée en

utilisant le plugin *Colibri*® qui permet de réaliser des captures d'écran de chaque solution testée ou des images 3D au format 3djson et d'enregistrer les données de performance dans un format csv. L'interface graphique de *DesignExplorer2*® est présentée en Figure 30.

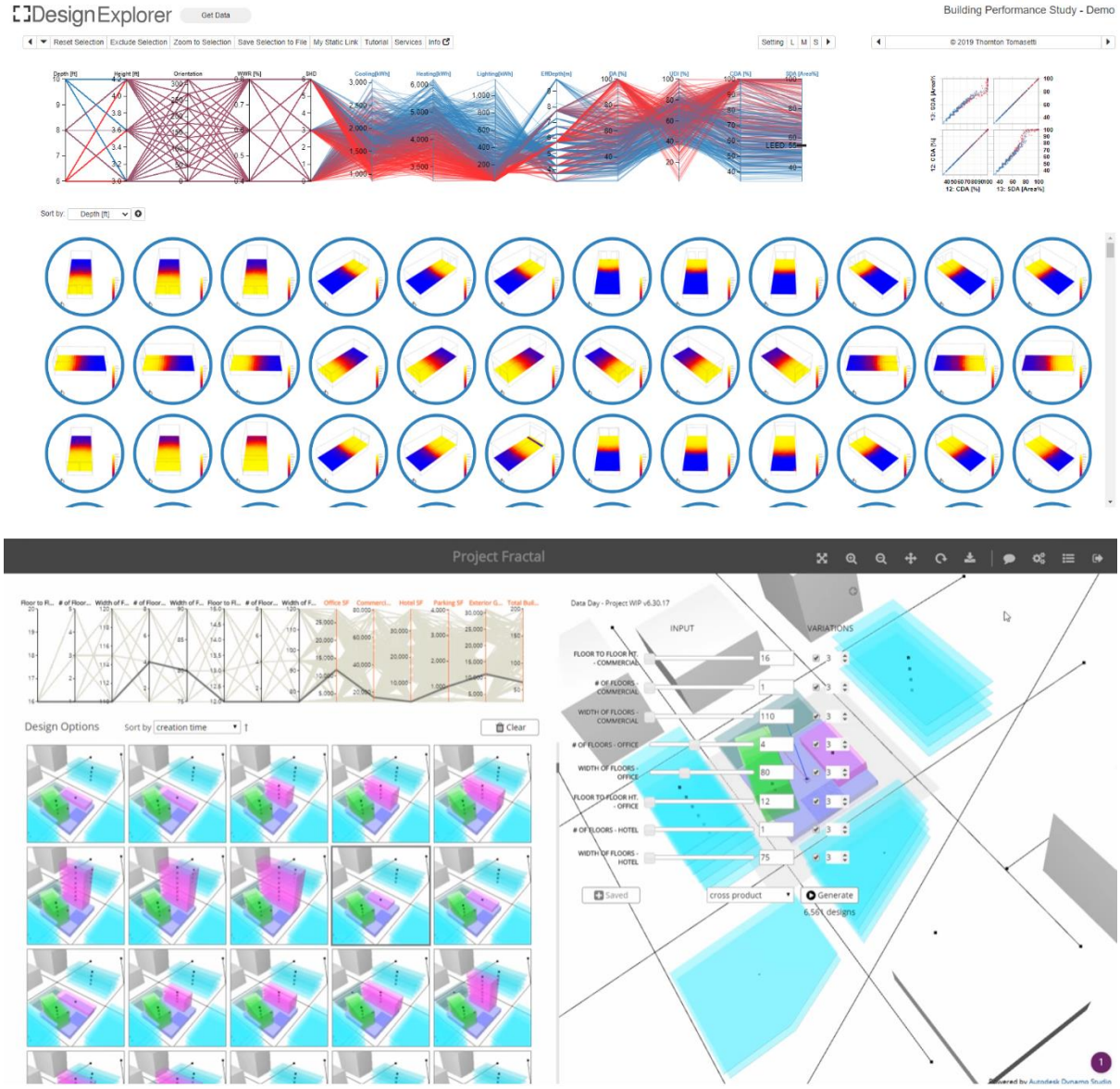


Figure 30: Interface graphique des applications web de data visualisation *Design Explorer2*® et *Project Fractal*®

Il existe un équivalent pour le BIM sur *Dynamo*® et *Revit*®. A l'origine, l'application web créée en 2017, *Project Fractal*®, présentée en Figure 30, est similaire à *DesignExplorer2*®, mais elle ne fonctionne qu'avec des données générées avec *Dynamo Studio*®. Elle a depuis été intégrée à *Project Refinery*®, un outil de conception génératif développé par *Autodesk*® aujourd'hui directement intégré à *Dynamo*® et *Revit*® et renommé depuis *Generative Design Primer*®.

Le diagramme de flux des données

Actuellement, pour mettre en œuvre une ENAPE, cinq types d'outils numériques sont ainsi nécessaires : (1) un modeleur 3D et sa plateforme de programmation visuelle, (2) un outil auxiliaire permettant l'interopérabilité (3) avec les outils de simulation environnementale, (4) un solveur d'optimisation et (5) un outil d'analyse en post traitement. Deux environnements aujourd'hui ciblent les architectes : celui de *Rhinocéros3D*® avec *Grasshopper*® et celui de *Revit*® avec *Dynamo*®.

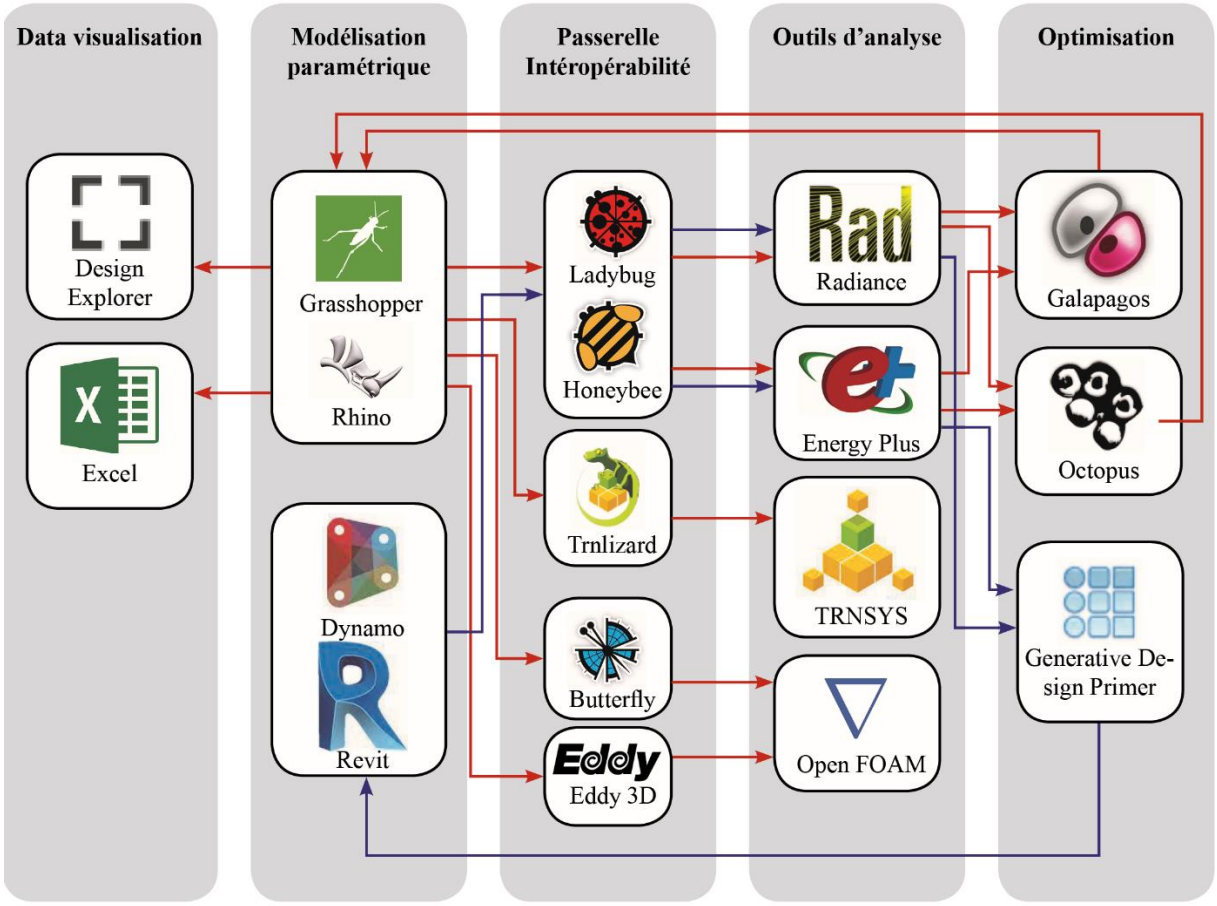


Figure 31: Diagramme de flux des données pour faire de l'ENAPE

La Figure 31 présentent le diagramme de flux des données pour ces deux environnements. Les données fluctuent ainsi entre le modeleur 3D, les outils de simulation et les algorithmes d'optimisation de manière automatique. Seule la transmission des données aux outils de post-traitement nécessite encore une intervention manuelle impliquant de sortir de l'environnement de base, uniquement pour *Grasshopper*®. Il existe d'autres outils pour faire de l'optimisation et de l'analyse que ceux représentés dans ce diagramme, que nous présenterons plus tard dans

cette thèse. Il s'agit ici des plus connus et reconnus auprès des chercheurs pour faire de l'optimisation de la performance énergétique.

Au terme de cette première partie, il apparaît que les architectes peuvent jouer un rôle essentiel dans la lutte contre le réchauffement climatique. Bien qu'il existe de nombreux outils sur le marché pour l'évaluation de la performance énergétique, ils restent peu adaptés aux méthodes de conception intuitives et créatives des architectes, notamment pour les phases amonts de projet, qui sont des étapes pourtant stratégiques pour la performance des bâtiments.
Depuis le développement de la programmation visuelle dans les modeleurs 3D, les chercheurs ont décrit en quantité des méthodes de conception computationnelle qui peuvent être qualifiées à la fois de méthodes génératives, intégrées, performancielles et environnementales, appelées méthodes d'exploration numérique de solutions architecturales basées sur des critères de performances environnementales. Ces approches conçues pour reproduire le processus itératif de conception d'une équipe de maîtrise d'œuvre se composent informatiquement de trois modèles : un modèle génératif, un modèle d'évaluation et un modèle d'exploration. Cinq types d'outils numériques sont nécessaires à sa mise en œuvre : les outils de modélisation, d'interopérabilité, d'évaluation, d'exploration et de visualisation.
La seconde partie de ce premier chapitre fait état des exemples d'applications de ces approches qu'il est possible de trouver aujourd'hui dans la littérature scientifique.

1.2. Les applications théoriques

Dans cette seconde partie du premier chapitre, nous proposons un état de l'art des applications de méthodes d'exploration numérique de solutions architecturales basée sur des critères de performance environnementale. Cette revue de littérature est constituée de 61 exemples d'applications qui intègrent des préoccupations architecturales. Quatre types de problèmes sont recensés : les problèmes de conception d'enveloppe, les problèmes de planification urbaine, les problèmes de morphologie du bâtiment et les problèmes d'agencement des espaces intérieurs.

1.2.1. Classification des cas d'études décrits dans la littérature scientifique

De l'approche ingénieur à l'approche architecte

Les méthodes d'ENAPE s'appuyant sur l'usage d'algorithme d'optimisation ont été utilisées par les ingénieurs bien avant les architectes et le développement de *Grasshopper*®. Ainsi, des exemples d'applications existent par centaines dans la littérature scientifique et des dizaines de revues de littératures ont déjà été publiées sur le sujet.

Les revues orientées ingénierie

La grande majorité des revues de la littérature scientifique sur l'optimisation des bâtiments ont été réalisées avec un œil d'ingénieur (Attia et al., 2013 ; Evins, 2013 ; Y. Huang & Niu, 2016 ; Kheiri, 2018 ; Nguyen et al., 2014). Ces derniers disposent de nombreux outils permettant de faire de l'optimisation tels que *Matlab*®, *GenOpt*®, *BEOpt*®, *Opt-E-Plus*®, *GENE_ARCH*®, *Modefrontier*®, *MutiOpt*®, *ParaGen*® (Kheiri, F., 2018). Les solveurs disponibles sur *Grasshopper*® ou *Dynamo*® ne sont même pas cités dans ce type d'étude. Les sujets abordés sont relativement éloignés des préoccupations des architectes. Trois types de sujets sont récurrents : les systèmes CVC, les paramètres physiques des matériaux et les paramètres géométriques (Nguyen et al., 2014). Parmi les variables interrogées, on retrouve l'orientation des stores (Mahdavi et al., 2005), la configuration du système CVC, les points de consigne, le ratio de surface vitrée ou les propriétés du vitrage (Evins, 2013). Un exemple d'application pour la conception de stores est illustré en Figure 32.

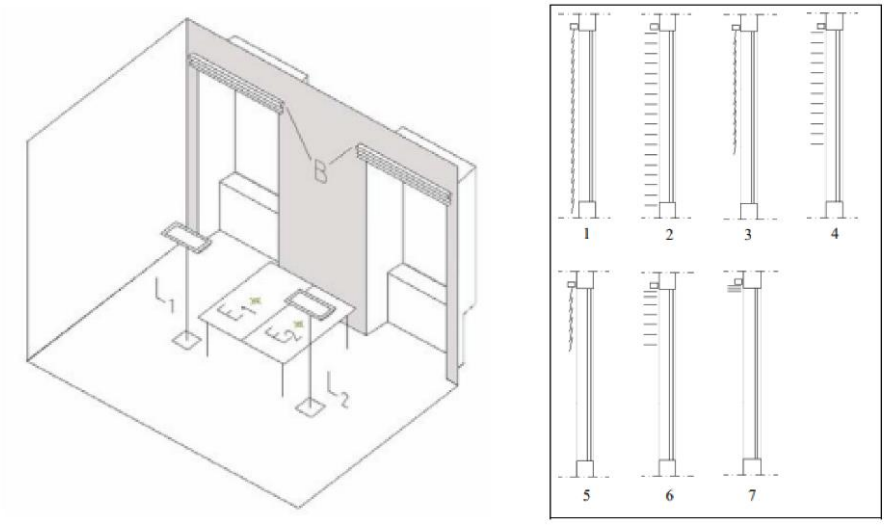


Figure 32: Exemple de cas d'étude inclus dans la revue de Attia et al. (2013) (source: Mahdavi et al., 2005)

Dans ce type de revues, les auteurs se concentrent essentiellement sur le sujet de la consommation énergétique et moins sur les questions de confort ou d'éclairage, ainsi les modèles 3D analysés sont très souvent simplifiés. Ces auteurs s'accordent sur le fait que les algorithmes génétiques sont les algorithmes d'optimisation les plus employés. Certaines revues comptent des dizaines de références comme l'étude d'Attia et de ses co-auteurs qui porte sur 165 publications (Attia et al., 2013). Malgré ces nombreuses recherches, l'optimisation est très peu utilisée dans la pratique par les ingénieurs. Cela s'expliquerait, d'après les auteurs, par des temps de calcul trop importants et un manque de méthode systématique (Attia et al., 2013). Certains l'expliquent aussi par le manque d'expérimentations sur des cas réels ne permettant pas de démontrer la performance de la méthode (Nguyen et al., 2014).

En 2018, les choses ont évoluées, Farshad Kheiri identifie dans une revue très orientée sur l'informatique (Kheiri, 2018) que le modeleur 3D Rhinocéros est devenu l'outil de visualisation le plus utilisé pour l'optimisation des enveloppes. Les modèles de substitution qui remplacent la simulation et permettent de réduire les temps de simulation sont évoqués mais restent rarement utilisés. Les algorithmes génétiques sont toujours les plus couramment utilisés même si d'autres méthodes d'optimisation existent (Y. Huang & Niu, 2016).

Les revues spécialisées

Avec le temps, les revues publiées se concentrent sur un sujet ou un angle particulier. Huang et Niu ont notamment analysés 52 références sur l'optimisation de l'enveloppe (Y. Huang & Niu, 2016) pour les consommations énergétiques, l'analyse du cycle de vie, le confort thermique et visuel (exclusivement relatif à la lumière). Ils ont notamment fait état des méthodes d'optimisation. En 2017, Eltaweed et Yuehong publient une revue sur la conception paramétrique et l'éclairage naturel (Eltaweel & Yuehong, 2017). L'usage de l'optimisation y est encore rare, et les principales applications portent sur les enveloppes modélisées avec *Grasshopper*®. Les références sont classées par types d'applications : les stores, lucarnes, masses, fenêtres, façades, intégration de panneaux photovoltaïques.

D'autres auteurs se concentrent sur l'échelle urbaine (Miao et al., 2020; Qeisi & Al-Alwan, 2021; Z. Shi et al., 2017). En 2017, Shi et ses co-auteurs passent en revues trois sujets appliqués exclusivement à cette échelle : les techniques génératives, les méthodes de conception performancielle et l'optimisation énergétique (Z. Shi et al., 2017). Ils passent ainsi en revue les critères et les outils d'analyse spécifiques à l'urbanisme. En 2020, Miao, Koenig et Knecht examinent l'optimisation architecturale et urbaine d'un point de vue historique (Miao et al., 2020). L'optimisation y est comparée à d'autres techniques génératives. Ils identifient

quatre grandes périodes : (1) les prémices avec l'optimisation monocritère dans les années 1960, 1970 et 1980, (2) les développements avec des méthodes logiques centrées sur l'Homme dans le courant des années 1990, (3) l'apparition de problèmes plus complexes et les méthodes d'optimisation métaheuristiques dans les années 2000, (4) et enfin les premières mises en pratique réussies avec l'intelligence artificielle dans les années 2010. Plus récemment, Qeisi et Al-Alwan vont identifier quatre catégories de méthodes de conception urbaine générative : (1) la déformation d'une forme simple, (2) la simulation de la croissance urbaine, (3) l'étude de la morphologie des îlots indépendamment du réseau viaire, et (4) l'analyse d'une structure spatiale existante (Qeisi & Al-Alwan, 2021).

Comme pour l'urbanisme, l'agencement spatial ou « *allocation spatiale* » est un sujet moins traité, ou traité à part et par des chercheurs qui se sont spécialisés sur ce type d'application plus complexes. C'est le cas de Du et de ses co-auteurs qui ont publié en quantité sur la relation entre performance énergétique et agencement spatial (Du et al., 2018 ; Du, Jansen, et al., 2020 ; Du, Turrin, et al., 2020). Ils ont recensé sept méthodes de génération de plans en s'appuyant sur 22 références de la littérature scientifique et font état d'une dizaine de références qui couplent les processus génératifs pour l'allocation spatiale avec la performance énergétique.

Approches orientées architecture

Depuis peu de temps, des revues de littérature orientées sur les questions architecturales ont été publiées (Ekici et al., 2019 ; S. Li et al., 2020 ; X. Shi et al., 2016 ; Touloupaki & Theodosiou, 2017 ; Zhao & de Angelis, 2018). Les premiers, à notre connaissance, à revendiquer une approche réalisée d'un point de vue de l'architecte sont Shi et ses co-auteurs qui publient en 2016 une revue de littérature exhaustive contenant 116 références. Néanmoins, elle contient des applications qui traitent de paramètres non géométriques, des modèles 3D très simplifiés et des outils d'optimisation peu connus des architectes comme *Matlab*® et *Modefrontier*®. En 2017, Touloupaki et Theodosiou se sont intéressés uniquement aux exemples pratiques qui concernent l'architecture. Seulement 13 exemples pratiques sur différents sujets sont analysés dans cette revue (Touloupaki & Theodosiou, 2017). Shaoxiong Li et ses co-auteurs ont réalisé en 2020 une revue systématique avec 99 références sur trois sujets : l'enveloppe, la forme et les protections solaires (S. Li et al., 2020). Bien qu'ils précisent prendre un point de vue d'architecte, 31 % des auteurs seulement proviennent de départements d'architecture, et de nombreuses références traitent de questions qui concernent dans la pratique les ingénieurs. En 2018, Zhao. et De Angelis ont sélectionnés et analysés 40 articles

scientifiques sur 5 types d'applications différentes dans le but d'identifier des corrélations entre les méthodes générative, d'évaluation et d'optimisation pour tenter de mieux appréhender la formulation des problèmes de conception (Zhao & de Angelis, 2018).

Tableau 1: Les paramètres en conception architecturale computationnelle et performancielle

Catégorie 1 : Paramètres géométriques	**Catégorie 2 : Autres paramètres**
Matrice d'adjacence	Température de consigne
Nombre de bâtiments	Points de consigne
Forme du bâtiment	Propriétés physique des matériaux
Conception du plafond	Fuites d'air
Conception de la façade	Générateur de chaleur
Hauteur de plancher	Humidité relative
Réflecteur de lumière	Débits d'air
Orientation	Délais de démarrage et d'arrêt
Forme de la toiture	Système de ventilation
Structure de la toiture	Energies renouvelables
Protection solaire	Taus d'infiltration
Dimensions et disposition d'une pièce	Taux d'absorption solaire
Trame 2D/3D	Système CVC et variables de contrôle
Dimensions et disposition de la fenêtre	
Ratio de surface vitrée	

Enfin, la revue qui nous semble la plus parlante et la plus complète pour s'adresser aux architectes est aussi la plus récente. En 2019, Ekici et ses co-auteurs proposent une taxonomie constituée de 100 études sur la conception architecturale computationnelle et performancielle (Ekici et al., 2019). Les critères de performance pris en compte dans cette étude sont la structure, le coût, la fonctionnalité et la durabilité. Les auteurs distinguent deux catégories de paramètres : (1) les paramètres d'« *exploration de la forme* » que nous appelons paramètres « *géométriques* » et (2) les autres paramètres que nous appellerons les paramètres « *thermiques* ». Ces différents paramètres sont présentés dans le Tableau 1. La première catégorie de paramètres est celle qui concerne réellement les architectes. Toutes les études de cette taxonomie portent sur au moins un paramètre géométrique. 58 % des études présentées dans cette revue de littérature concernent l'enveloppe, 23 % la forme du bâtiment et 19 % l'agencement spatial. Comme la montre la Figure 33, les auteurs ont pu identifier quels étaient les paramètres géométriques et les critères de performances associés les plus étudiés.

Néanmoins les auteurs soulignent eux-mêmes en conclusion que certaines références se concentrent sur des préoccupations architecturales, mais que d'autres ont des préoccupations plus d'ingénierie, ou d'informatique. Ils précisent notamment que les questions de ratio de surfaces vitrées et les systèmes de protection solaire sont abordés sans préoccupations architecturales dans de nombreux articles.

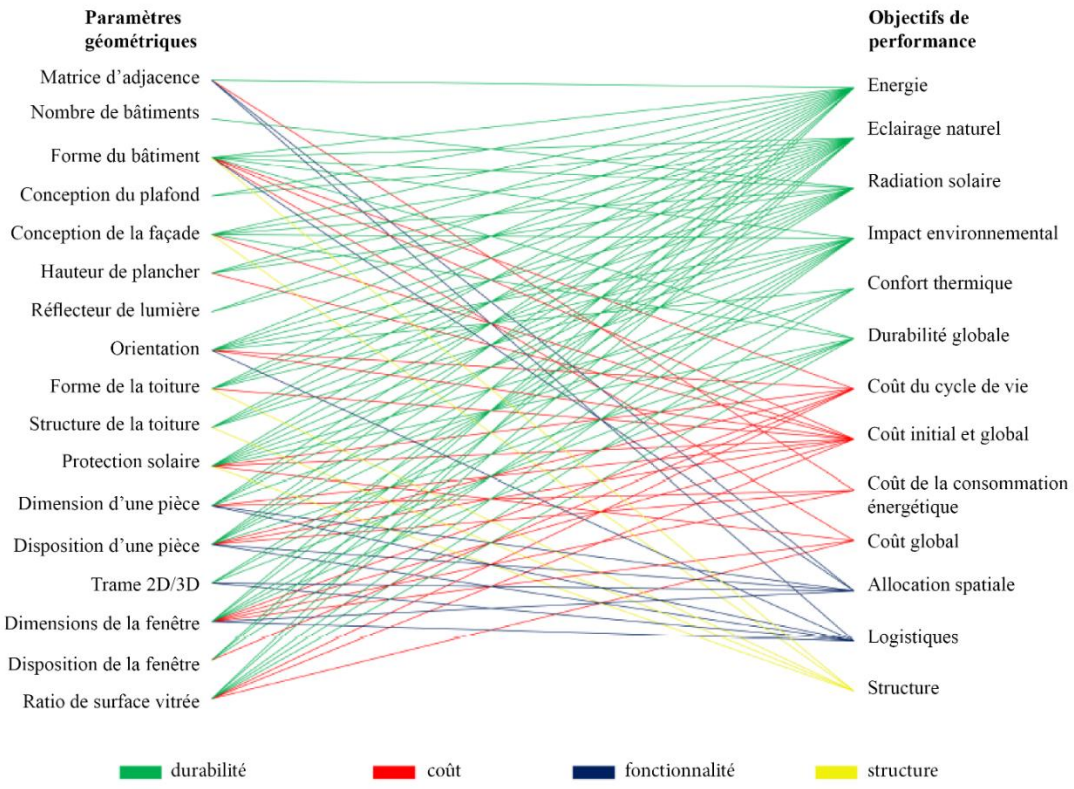


Figure 33: Relations entre paramètres géométriques et critères de performance (source: Ekici et al., 2019)

Méthode d'élaboration de la revue de littérature

Méthode de sélection des références

Ainsi, nous proposons notre propre revue de littérature contenant des références abordant spécifiquement des préoccupations architecturales sur des sujets de conception d'enveloppe, de planification urbaine, de morphologie de bâtiments et d'îlots urbains, et des problèmes d'agencements d'espaces. L'objectif est de montrer la diversité des approches, plus que de faire une revue de littérature exhaustive. Ainsi, nous nous appuyons sur des applications concrètes susceptibles d'inspirer les praticiens, et donc de faciliter la mise en application de ces méthodes dans un contexte professionnel. Nous avons sélectionné parmi les études citées dans les revues

de littérature précédemment présentées, celles qui incluaient à la fois l'étude de paramètres géométriques, et des préoccupations architecturales. Toutes les études portant sur des paramètres de catégorie 2 listés dans le Tableau 1 ont été exclues. Les sujets où certains critères géométriques sont traités comme des paramètres thermiques comme de simple ratio de surface vitrée ont été considérés comme ne présentant pas d'intérêt architectural et ont été exclus du corpus de référence. Ce dernier a ensuite été augmenté par une recherche sur les plateformes académiques en ligne *Google Scholar* et *Cumincad*. A cette fin, plusieurs combinaisons d'au moins trois mots clés ont été utilisés (et sont présentés dans le Tableau 2). Celles-ci incluent un terme pour évoquer un type d'application, un terme pour évoquer une notion d'évaluation et un dernier pour évoquer la notion d'exploration.

N'ont été retenus que les articles présentant les caractéristiques précédemment décrites. En définitive, 61 articles scientifiques ont été retenus. Ils sont listés par ordre chronologique dans le Tableau 3.

Tableau 2 : Mots clefs utilisés pour l'élaboration de la revue de littérature

Termes - Application	**Termes - Evaluation**	**Termes - Exploration**
« architectural design »	« performance »	« generative design »
« urban design »	« integreted design »	« optimization »
« facade design	« sustainable design »	« exploration design »
« building envelope »	« data driven design »	« parametric design »
« building form »	« energy performance »	
« urban block »	« thermal performance »	
« street network »	« daylighting performance »	
« space layout »		
« space allocation »		

La carte d'identité visuelle des références

Les références ont ensuite été cataloguées selon quatre caractéristiques qui permettent de les classifier : (1) le type d'application, (2) la technique générative, (3) les critères de performance et (4) les techniques d'exploration. Pour chaque référence du corpus, une carte d'identité graphique a été réalisée, comme celle présentée en Figure 34. Elle contient les illustrations du cas d'étude contenues dans l'article scientifique, les quatre caractéristiques du problème présenté sous forme de logos et des précisions sur ces caractéristiques.

Tableau 3: Tableau des références du corpus (partie 1)

	RÉFÉRENCE	TYPE	APPLICATION	PROGRAMME	CLIMAT	GÉNÉRATION	CRITERES D'EVALUATION	MÉTHODE D'EVALUATION	NB DE CRITÈRES	MÉTHODE D'EXPLORATION
1	(Kämpf et al., 2010)	application		logements	montagnard			physique	●●	
2	(M. El Sheikh & Gerber, 2011)	théorique		bureaux	aride			physique	●●	
3	(Hudson et al., 2011)	expérimentation		stade	océanique			physique	●●	
4	(Bechthold et al., 2011)	théorique		_	continental			physique	●●	
5	(Henriques et al., 2012)	théorique		_	océanique			physique	●	
6	(Gagne & Andersen, 2012)	théorique		_	continental			physique	●●	
7	(Menges, 2012)	application		_	tropical			physique	●●●●	
8	(Menges, 2012)	théorique		_	_			physique	●●●●●	
9	(Sarvani, et al., 2013)	théorique		logements	méditerranéen			physique	●●	
10	(Gokmen, 2013)	théorique		_	_			physique	●	
11	(Parascho, 2013)	théorique		_	_			physique	●●●	
12	(Elghazi et al., 2014)	théorique		logements	aride			physique	●●	
13	(H. Yi, 2014)	application		commerces	continental			physique	●●	
14	(Gerber & Lin, 2014)	théorique		_	_			physique	●●●	
15	(António et al., 2014)	théorique		_	_			physique	●	
16	(H. Yi & Yi, 2014)	application		logements	_			physique	●●●	
17	(Ercan & Elias-Ozkan, 2015)	application		bureaux	méditerranéen			physique	●●	
18	(Kim, 2015)	théorique		_	océanique			physique	●	
19	(Conti, Z. et al., 2015)	théorique		logements	méditerranéen			physique	●●	
20	(Negendahl & Nielsen, 2015)	expérimentation		bureaux	océanique			substitution	●●●	
21	(Suyoto, W., et al., 2015)	expérimentation		bureaux	équatorial			physique	●	
22	(Y.-S. Huang et al., 2015)	application		_	tropical			physique	●●	
23	(Y. K. Yi & Kim, 2015)	application		logements	continental			physique	●	
24	(Taleb & Musleh, 2015)	application		logements	aride			physique	●	
25	(Su & Yan, 2015)	théorique		hôpital	_			physique	●●	
26	(Fathy et al., 2015)	théorique		ecole	aride			physique	●●	
27	(Narangerel et al., 2016)	théorique		bureaux	continental			physique	●●	
28	(Abdelwahab & Elghazi, 2016)	théorique		logements	aride			physique	●●	
29	(M. F. Mohamadin et al., 2016)	théorique		laboratoires	aride			physique	●●	
30	(H. Yi, 2016)	application		logements	continental			physique	●●●	
31	(Fathy & Fareed, 2017)	application		bibliothèque	aride			physique	●●	

	RÉFÉRENCE	TYPE	APPLICATION	PROGRAMME	CLIMAT	GÉNÉRATION	CRITERES D'EVALUATION	MÉTHODE D'EVALUATION	NB DE CRITÈRES	MÉTHODE D'EXPLORATION
32	(Tomasowa & Sjarifudin, 2017)	application		bureaux	équatorial			physique	●	
33	(Gerber et al., 2017)	théorique		logements	montagnard			physique	●●	
34	(Sleiman et al., 2017)	théorique		hôpital	_			physique	●●	
35	(Shen, 2018)	théorique		bureaux	continental			physique	●●●●	
36	(Shen et al., 2018)	expérimentation		laboratoires	méditerranéen			physique	●●	
37	(Jayathissa et al., 2018)	théorique		logements	montagnard			physique	●●●	
38	(Zarrabi et al., s. d.)	théorique		_	continental			physique	●	
39	(Youssef et al., 2018)	application		_	aride			physique	●	
40	(Li et al., 2018)	application		université	continental			physique	●●	
41	(Chen et al., 2018)	application		_	équatorial			physique	●●	
42	(Haymaker et al., 2018)	théorique		école	méditerranéen			physique	●●●●●●●●	
43	(Vermeulen et al., 2018)	expérimentation		_	_			physique	●	
44	(Nagy et al., 2018)	expérimentation		logements	océanique			physique	●●	
45	(Y. K. Yi, 2019)	application		logements	méditerranéen			physique	●●	
46	(Tabadkani et al., 2019)	théorique		bureaux	aride			physique	●	
47	(Agirbas, 2019)	théorique		logements	continental			physique	●	
48	(T. Chen et al., 2019)	théorique		bureaux	équatorial			physique	●	
49	(Hosseini et al., 2019)	théorique		bureaux	aride			physique	●	
50	(Chatzikonstantinou et al., 2019)	expérimentation		laboratoires	océanique			physique	●●	
51	(Marsault & Torres, 2019)	application		logements, bureaux	_			substitution	●●●●	
52	(Showkatbakhsh, M., et Kaviani, S., 2020)	application		bureaux	aride			physique	●●●●	
53	(Ciardiello et al., 2020)	théorique		logements	_			physique	●●●	
54	(Mukkavaara & Sandberg, 2020)	application		logements	montagnard			physique	●●●●●	
55	(R. Koenig et al., 2020)	application		_	océanique			synthaxe spatiale	●●●●	
56	(Duering et al., 2020)	application		_	océanique			substitution	●●●●	
57	(Rizi & Eltaweel, 2021)	théorique		_	continental			physique	●●●	
58	(A. G. Fathy et al., 2021)	théorique		_	aride			physique	●●	
59	(W. Wang et al., 2021)	application		_	continental			substitution	●●●	
60	(Schwartz et al., 2021)	théorique		logements	océanique			physique	●●●	
61	C. Huang et al., 2022	théorique		_	_			substitution	●●	

L’ensemble de ces fiches constitue un catalogue de références destiné à aider les architectes à intégrer ce type d’approches en phases amonts de conception. Dans la pratique, les architectes travaillent beaucoup avec des références graphiques. Le projet est une démarche particulière qui se distingue de la démarche scientifique. Aussi, les architectes n’ont pas le temps en phase esquisse ou en concours pour rechercher, ou étudier les articles scientifiques qui pourraient correspondre au problème de conception qu’ils rencontrent. Ce catalogue a donc été pensé pour servir d’intermédiaire entre la démarche créative et intuitive des architectes et les méthodes scientifiques d’ENAPE.

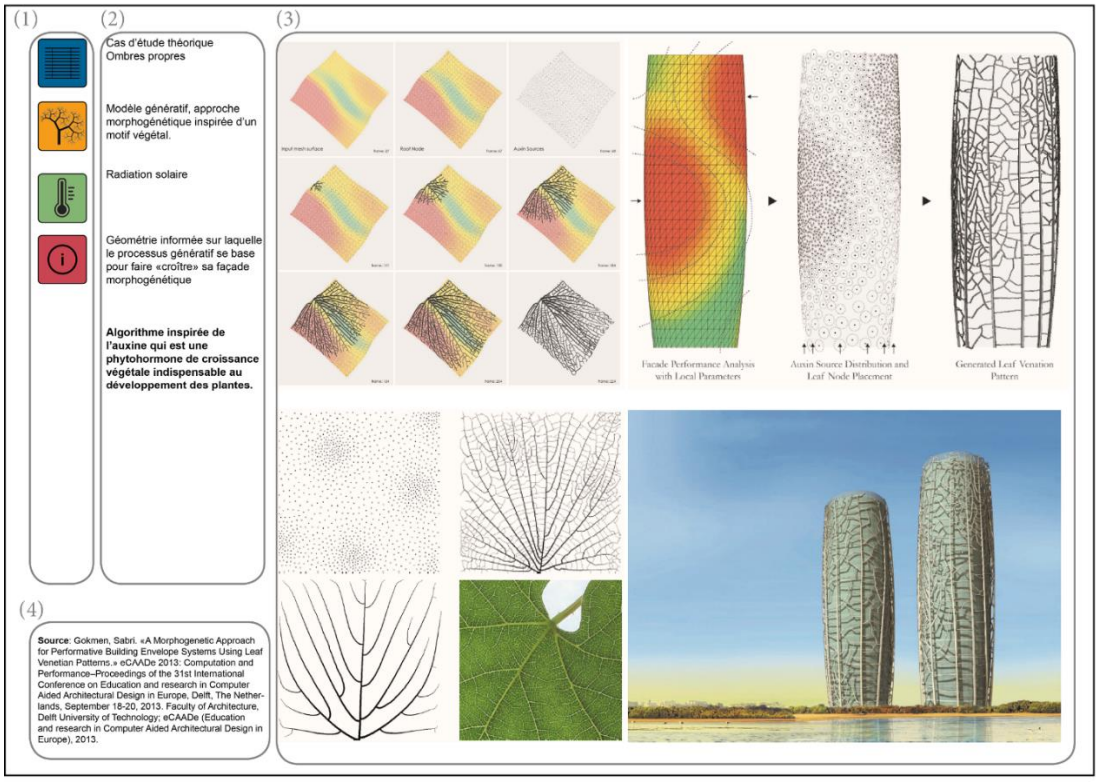


Figure 34: Carte d'identité visuelle de la référence (Gokmen, 2013)

Les critères de classification des cas d’études

Les différents types d’applications

L’ENAPE a été adaptée à différentes échelles et à différents objectifs de performance, faisant émerger ainsi des spécificités selon les catégories d’applications. Nous avons identifié quatre catégories d’application :

(1) les applications pour l’enveloppe du bâtiment, qui peuvent se faire à l’échelle de toute une façade ou seulement sur un détail, qui est la catégorie la plus représentée dans la

littérature et qui comprend l'étude des protections solaires, de la disposition du vitrage et la forme de la peau.

(2) les applications à l'échelle urbaine qui comprennent la génération du réseau viaire d'une part et la génération du bâti dans les îlots d'autre part.

(3) les applications pour explorer la morphologie du bâtiment. Il peut s'agir d'explorer la forme et l'orientation d'un unique bâtiment ou d'explorer la morphologie et la disposition de plusieurs éléments, parfois la génération d'îlots urbains entiers.

(4) les applications pour l'agencement des espaces. Ce peut être l'agencement d'un groupe d'espaces comme des appartement, ou l'agencement de pièces ou de mobilier.

Dans le catalogue, les différents types d'applications sont représentés en bleus aux tonalités différentes. Les logos permettent de distinguer des sous-catégories (voir Figure 35).

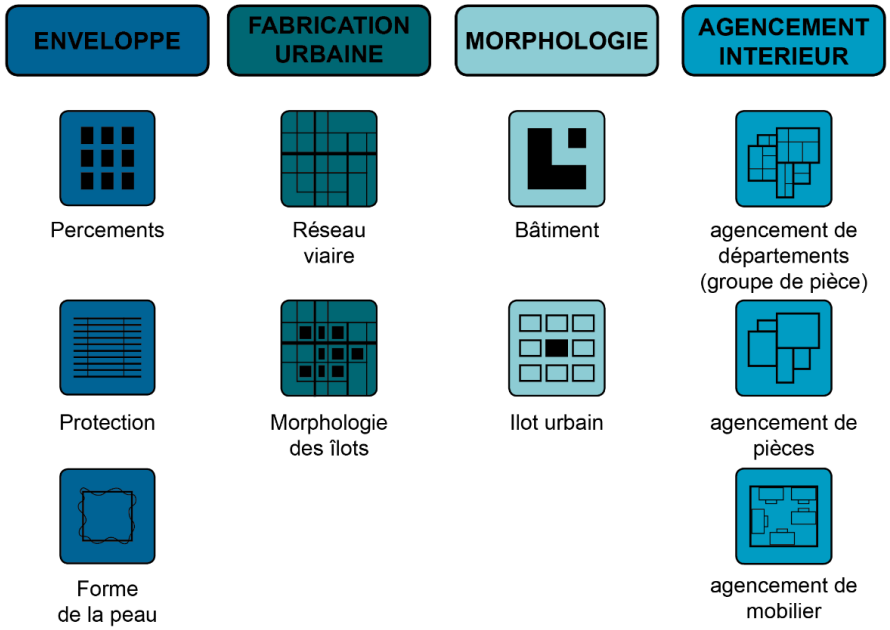


Figure 35: Logos pour distinguer les différents types et sous-types d'applications.

Comme le montre le graphique en Figure 36Figure 36: Statistiques sur les types d'application et de cas d'étude du corpus de référence, le sujet de l'enveloppe est de loin le plus représenté puisque 54 % des références s'appliquent à l'enveloppe du bâtiment. Il y a ensuite les applications sur la morphologie qui représentent 28 % du corpus de référence, suivi par les applications à l'échelle urbaine (10 %) et l'agencement spatial (8 %). On observe aussi que les applications à l'échelle urbaine sont plus récentes que celles appliquées à l'échelle du bâtiment. Aussi, nous distinguons 3 types de cas d'étude : (1) les cas d'études théoriques où le problème

d'architecture est une invention de l'auteur sans qu'il n'existe même de site, (2) les applications qui s'appliquent à un vrai problème qu'il serait possible de rencontrer dans la pratique avec une localisation, un terrain, un contexte urbain, et enfin (3) les expérimentations qui sont des mises en pratique dans un cadre professionnel. Les cas d'étude théoriques sont majoritaires (54 %), suivi par les applications (34 %). Les expérimentations restent encore rares (12 %).

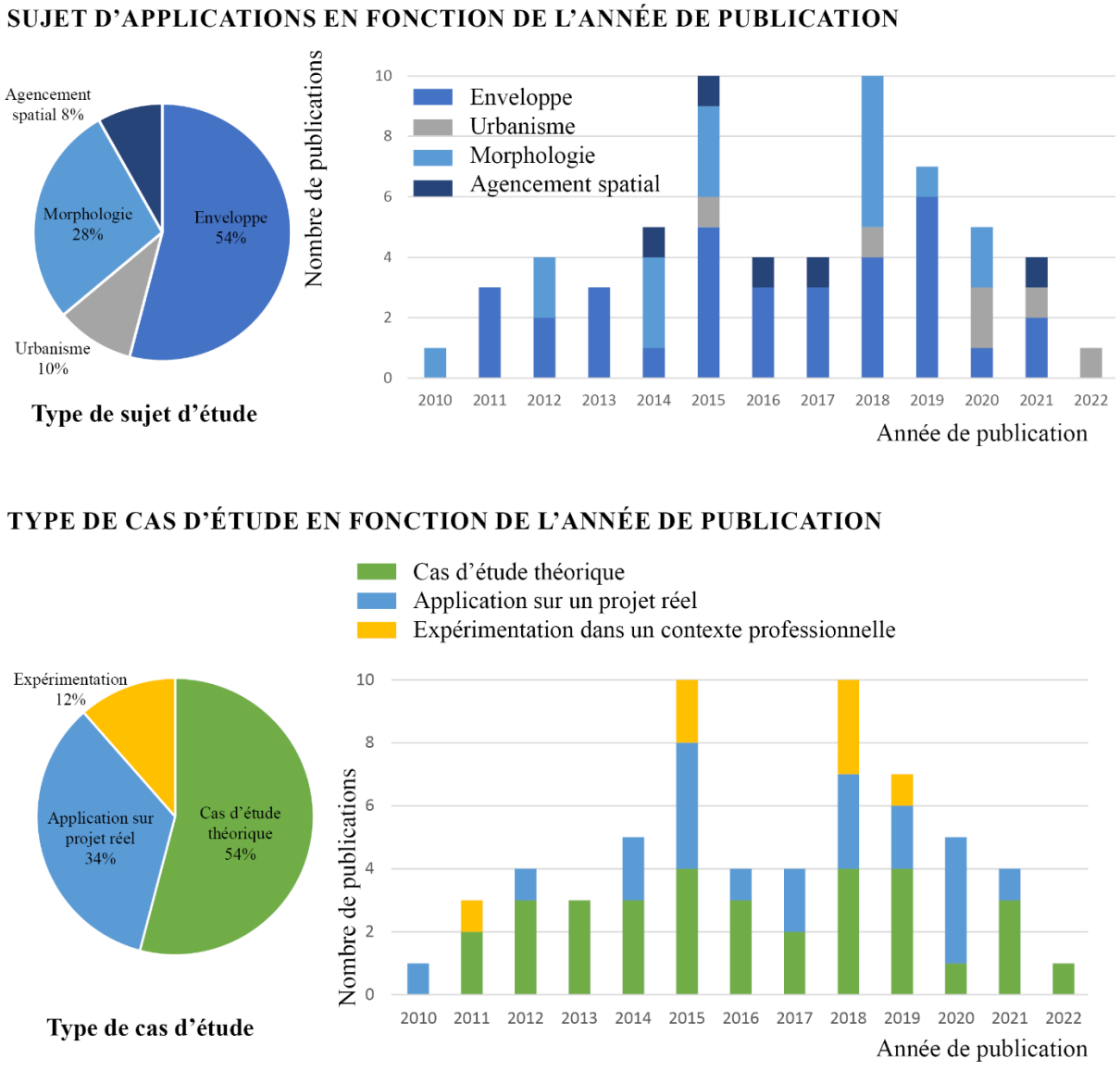


Figure 36: Statistiques sur les types d'application et de cas d'étude du corpus de références

Ces méthodes sont appliquées à des sujets dans différents climats. Si pour 23 % la localisation du fichier météo n'est pas précisée dans l'article, 20 % ont lieu dans un climat équatorial, 20 % continental, 13 % océanique, 10 % méditerranéen, 6 % équatorial, et 5% dans un climat montagnard. Comme le montre le graphique en Figure 37, en climat aride ou

méditerranéen, il s'agit surtout d'applications sur l'enveloppe. A l'inverse, en climat montagnard, la morphologie du bâtiment est plus étudiée. Les applications à l'échelle urbaine sont testées sur les villes européennes, donc un climat océanique. Les programmes ne sont pas renseignés pour 40 % des applications. Pour le reste, les applications sur les logements représentent 23 % du corpus. C'est le seul programme étudié pour tous les sujets (enveloppe, forme, urbanisme et agencement). La majorité des applications se font sur des espaces de travail : 18 % pour des bureaux, 8 % pour des écoles, bibliothèques ou université, et 5 % pour des laboratoires.

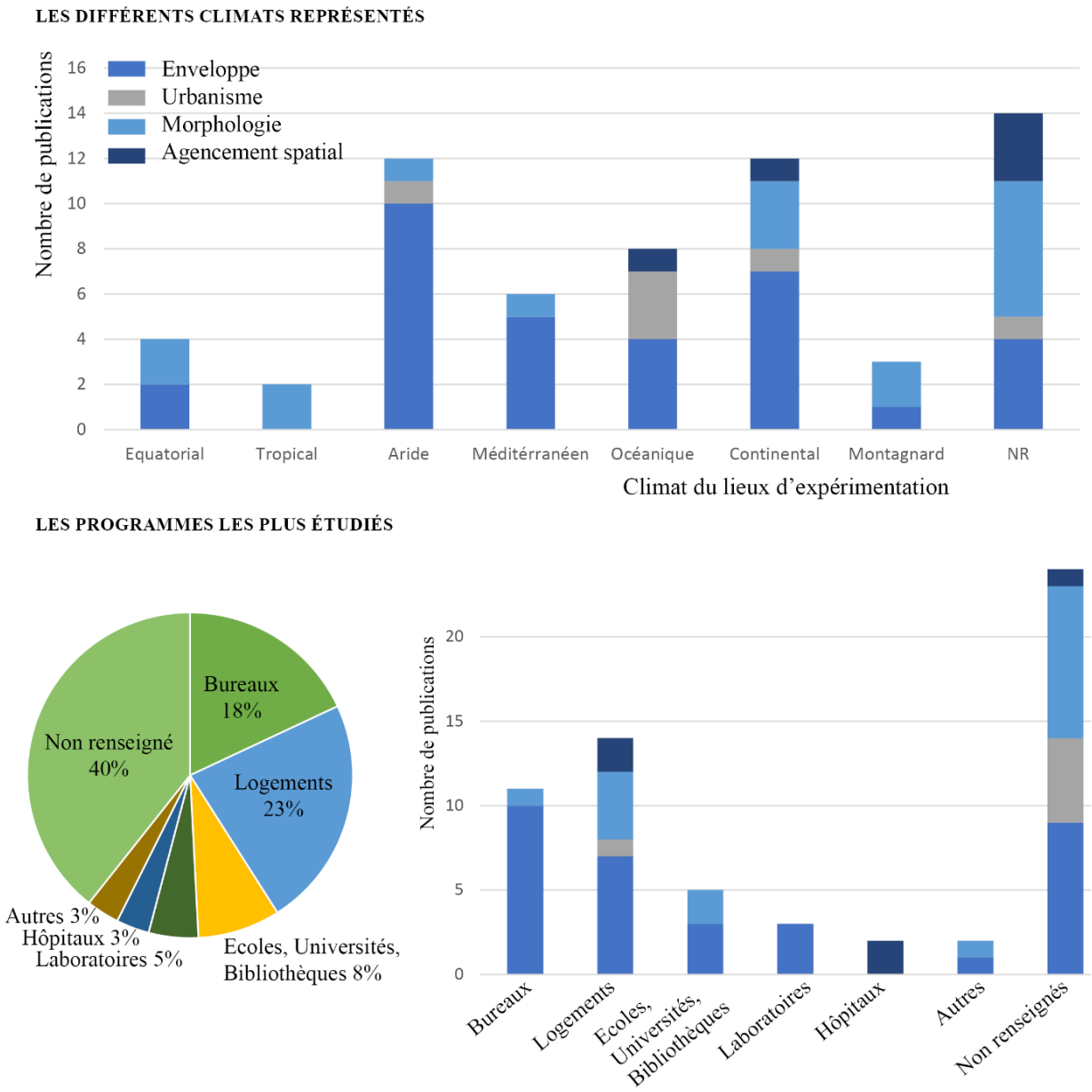


Figure 37: Statistiques sur les différents climats et programme du corpus de références

Les différentes méthodes génératives

Différentes méthodes sont utilisées pour définir l'espace de solution à explorer. Nous distinguons deux grandes catégories de modèles : les modèles paramétriques et les modèles génératifs (voir Figure 38). Pour chacune de ces catégories nous avons identifié des sous-catégories.

Nous distinguons 4 types de modèles paramétriques : les modèles paramétriques simples et trois types de modèles plus complexes : les modèles avec un système de discrétisation (comme des panneaux en façade), les motifs géométriques complexes (les patterns) et les structures en origami.

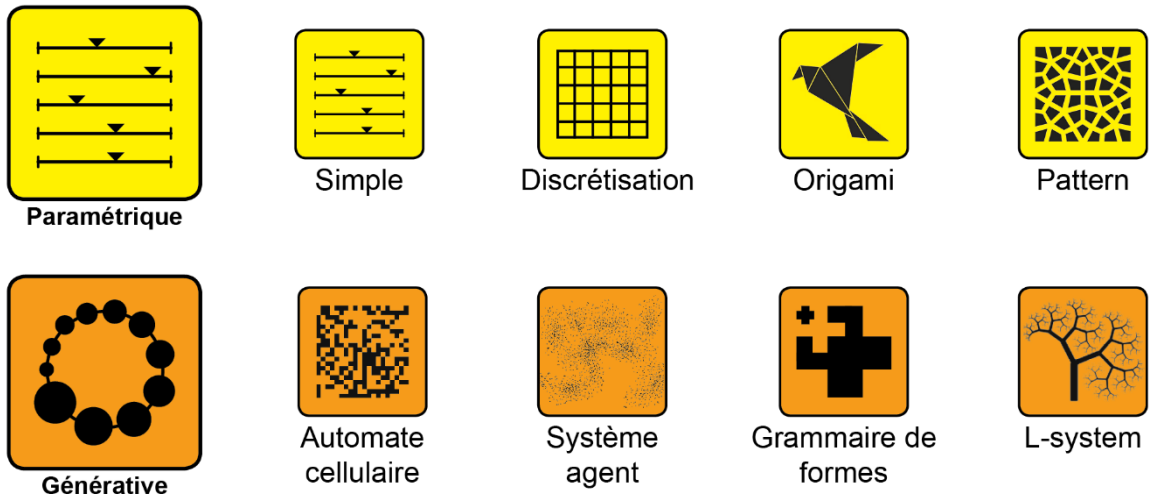


Figure 38: Logos pour distinguer les différentes méthodes génératives

Selon (Singh & Gu, 2012), il existe quatre techniques de modèles génératifs différentes utilisées en architecture en plus des algorithmes génétiques : les modèles à base d'agents (MBA), les automates cellulaires (AC), les grammaires de formes (GF), et les systèmes de Lindemeyer (LS).

Il existe de nombreux termes qui font référence aux MBA : « *système d'auto-organisation* », « *modélisation à base d'agents* », « *simulations à base d'agents* » ou encore « *systèmes multi-agents* ». Selon C.M. Macal (Macal, 2016) tous ces termes renvoient à des méthodes inspirées de l'intelligence artificielle distribuée. Un MBA est un processus qui permet de résoudre des systèmes complexes constitués de multiples entités en interaction appelées agents. En exécutant des règles en fonction d'informations locales, les agents peuvent atteindre un objectif global en coopérant entre eux et en fonction de leur environnement. Les agents doivent être des individus modulaires identifiables, autonomes, sociables puisqu'ils interagissent entre eux et ont un état qui peut varier au cours du temps (Macal & North, 2005). En architecture, les programmes les

plus utilisés sont ceux inspirés de phénomènes observables dans la nature comme les nuées d'oiseaux en vol, les colonies de fourmis, les essaims d'insectes ou les bancs de poissons (Ma, 2015).

Un AC est un MBA particulier avec des caractéristiques distinctes. Cet objet mathématique peut, à partir de règles très simples, générer des formes complexes. Il est défini par une grille à une, deux ou trois dimensions fabriquant ainsi des cases nommées « *cellules* ». Chaque cellule peut prendre un nombre d'états préalablement définis (au minimum 2). L'évolution de l'état d'une cellule impacte l'état des cellules voisines selon un rayon qui définit l'intervalle permettant qu'une cellule interagisse avec une autre (Von Neumann & Burks, 1966). A partir de ces trois paramètres, on définit un AC pour lequel il existe un nombre fini de règles applicables.

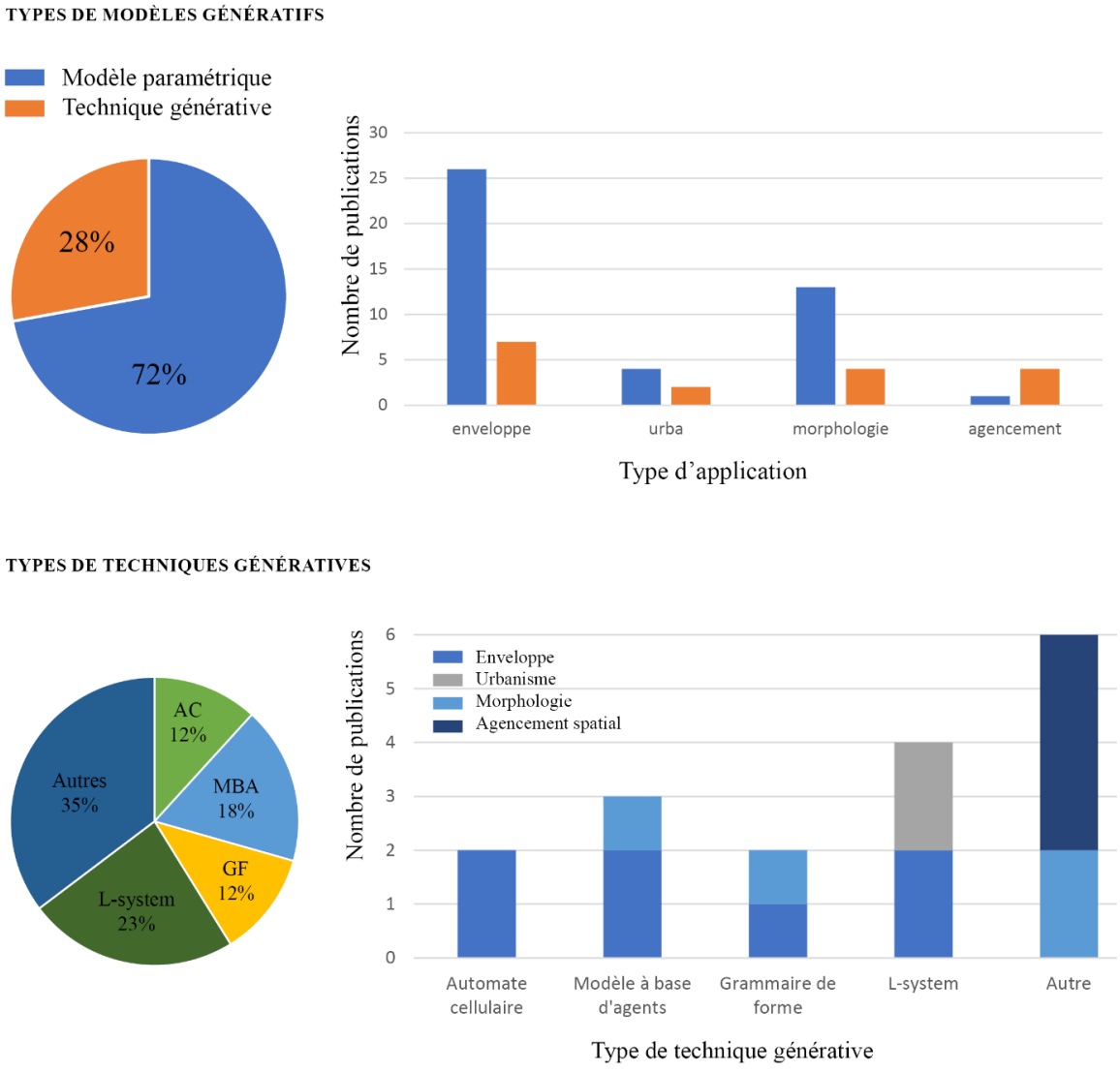


Figure 39: Statistiques sur les différentes méthodes génératives du corpus de références

Une GF est une méthode qui consiste à faire émerger des compositions en appliquant successivement un ensemble de règles de transformation à une forme de départ, appelée forme initiale (Knight, 2003). Cette technique n'a pas toujours été assistée par un ordinateur. Elle était à l'origine utilisée pour faire de l'analyse historique des styles en architecture avant de servir pour la production architecturale ou encore comme un outil d'aide à la conception (Tepavčević & Stojaković, 2012).
Le LS est un système de réécriture inventé par le biologiste Lindenmayer. Cette grammaire générative permet de modéliser le processus de croissance des plantes à partir de règles et de symboles (Lindenmayer, 1968).

Pour faire un modèle génératif, il n'est pas nécessaire d'utiliser une des techniques présentées ci-dessus. Certains problèmes nécessitent le développement d'un modèle spécifique. Comme la montre la Figure 39, la plupart des références de notre étude utilisent des modèles paramétriques (72 %) qui sont beaucoup plus faciles à modéliser et à contrôler. Dans notre corpus, les AC sont exclusivement utilisés pour la modélisation de l'enveloppe. Les problèmes d'agencements n'utilisent pas de techniques génératives spécifiques mais d'autres techniques. L'échelle urbaine utilisent exclusivement des LS. Certaines applications interrogeant la morphologie du bâtiment utilisent des GF et MBA où d'autres méthodes.

Les différents objectifs de performance

Les critères de performance étudiés différent selon le type d'application. Nous distinguons 5 types de critères (voir Figure 40). Les deux premiers sont essentiels pour la performance environnementale. Il s'agit des critères d'évaluation du confort, et des critères d'évaluation de l'énergie et d'analyse environnementale. Trois autres types de critères peuvent aussi être utilisés : les critères métriques, économiques et esthétiques. L'analyse du confort est la plus représentée avec 55 % des références, suivie par l'énergétique (27 %) , puis métrique (11 %) et enfin économique (6 %). Le critère esthétique est beaucoup plus rare puisqu'il n'est intégré que dans une référence (Y. K. Yi, 2019).

La notion de confort est essentielle en conception environnementale car c'est le manque de confort qui entraine un surplus de consommations énergétiques. Dans notre étude, seuls deux types de confort sont représentés : thermique et visuel. Aucun article retenu n'inclut une analyse du confort acoustique, mais il existe des recherches sur l'exploration pour la performance acoustique (Spaeth, 2011). Le confort thermique est étudié principalement à l'aide de trois

indicateurs différents : (1) la radiation solaire ou de l'ensoleillement direct pour favoriser les apports solaires lors des périodes froides et les minimiser lors des périodes chaudes, (2) l'indice de confort thermique PMV (Predicted Mean Vote) qui prédit la valeur moyenne des votes d'un grand groupe de personnes sur l'échelle de sensation thermique, (3) la vitesse des vents pour le confort des espaces extérieurs en favorisant les vents faibles lors des périodes de basses températures et inversement pour les hautes températures. L'analyse de la radiation solaire est un indicateur récurrent dans notre corpus d'étude, l'analyse des vents est beaucoup plus rare (voir Figure 41).

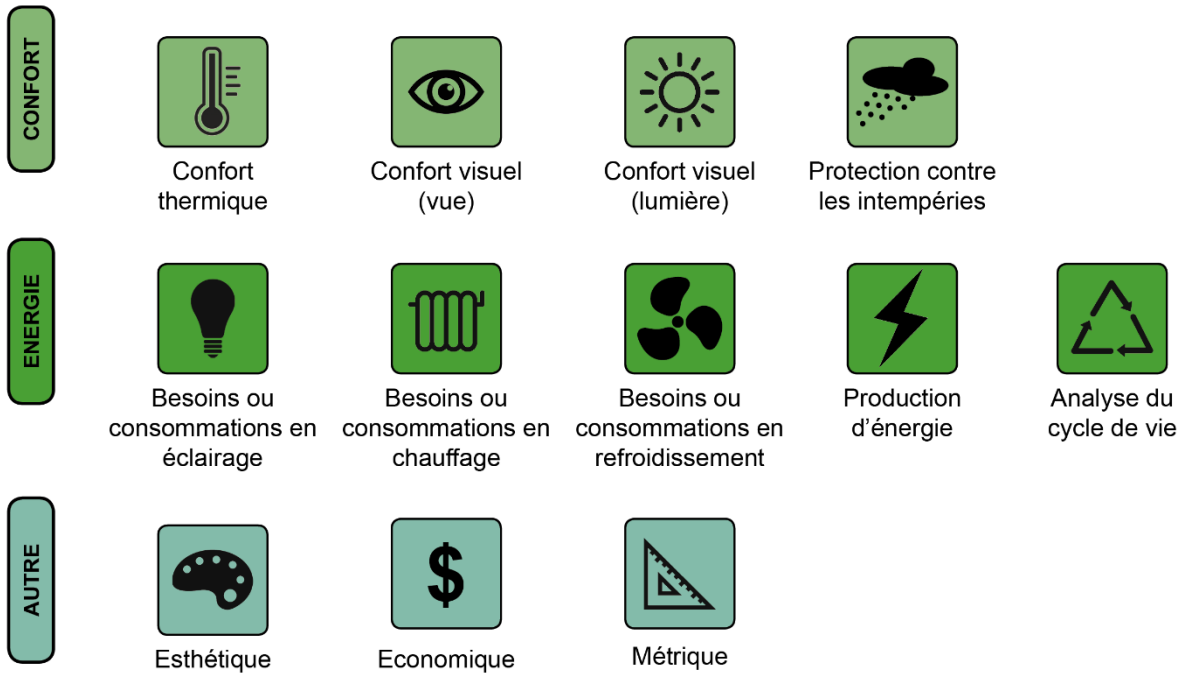


Figure 40: Logos pour distinguer les différents objectifs de performance

Pour le confort visuel nous distinguons l'évaluation de la lumière naturelle et de la qualité de vue. De nombreux indicateurs différents sont utilisés pour étudier le confort lumineux : le facteur de lumière du jour, l'autonomie en lumière du jour, l'éclairement à une date et heure précise, l'analyse de l'éclairement utile annuel, l'intensité lumineuse, ou encore les indices d'éblouissement. L'analyse de la vue sont des analyses uniquement quantitatives. Il s'agit principalement d'analyse de l'obstruction, appelée visibilité. La prise en compte de la qualité de vue reste encore rare (voir Figure 41).

Les analyses énergétiques sont les plus courantes après l'évaluation du confort thermique ou visuel. Il peut s'agir de l'étude des besoins énergétiques ou des consommations, les consommations en chauffage et refroidissement étant les plus représentées. Pour cela, la plupart des auteurs utilisent la simulation physique. La production d'énergie est parfois étudiée

à l'aide d'une simple analyse de la radiation. Certains auteurs intègrent l'évaluation de l'empreinte carbone avec une analyse du cycle de vie. Ce type d'analyse peut être très chronophage, c'est pourquoi la simulation physique est parfois remplacée par des modèles de substitution. Cependant, ils restent très peu représentés dans notre littérature sur les ENAPE.

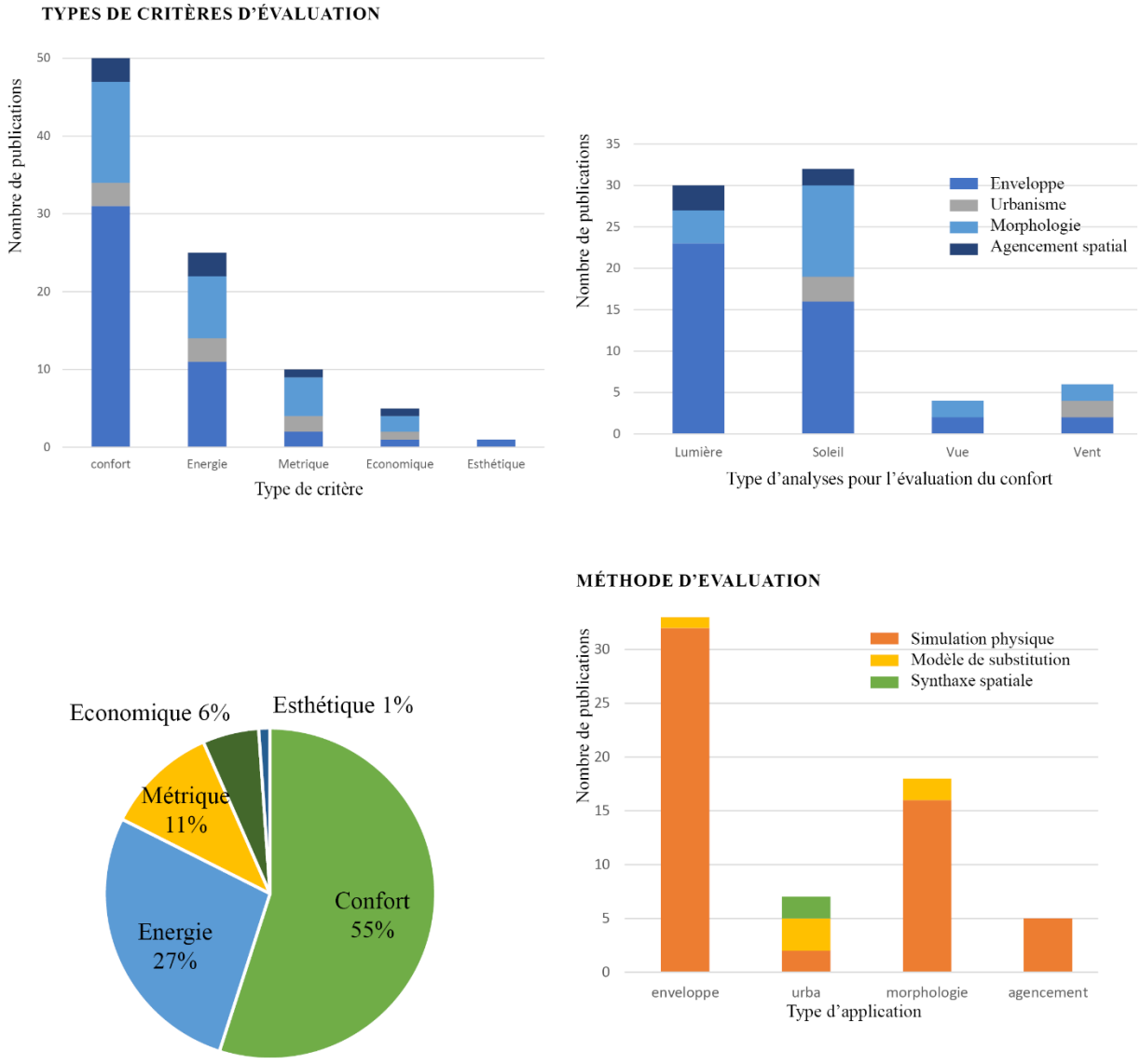


Figure 41: Statistiques sur les critères de performance étudiés du corpus de références

Les critères métriques sont l'ensemble des critères quantitatifs facilement mesurables comme les surfaces de plancher, les volumes construits, la distance entre deux éléments. Ces critères peuvent aussi avoir un impact sur les performances énergétiques et environnementales, avec des indicateurs comme la compacité et un volume représentant une quantité de matière. Les indicateurs d'analyse de l'échelle urbaine comme l'accessibilité, l'orthogonalité sont aussi classés dans cette catégorie. Les critères économiques comme le coût de construction ou le coût d'exploitation, ou encore l'estimation du profit sont quelquefois pris en compte mais restent

rarement intégrés. L'esthétique est un critère difficilement quantifiable, ce pourquoi il est très difficile à intégrer dans ce type d'approche, bien que certains aient tenté l'expérience en utilisant des algorithmes de *Deep Learning* (Y. K. Yi, 2019).

Les différentes techniques d'exploration

Si les algorithmes d'optimisation, notamment les algorithmes génétiques sont les techniques les plus représentées pour faire de l'exploration, il en existe d'autres listées en Figure 42. Il y a d'abord des techniques d'exploration très simples, qui ne nécessitent pas l'usage d'outils mathématiques ou informatiques particuliers, nous parlons des techniques « *triviales* », comme l'exploration « *manuelle* » où l'utilisateur explore l'espace de solutions à la main en faisant varier les paramètres, et où une petite quantité de scénarios est analysée. Il est possible d'utiliser directement les résultats mappés sur la géométrie d'une analyse pour les intégrer à un modèle paramétrique. Nous qualifions cette méthode de « *géométrique informée* ». La méthode manuelle et la méthode de la géométrie informée représentent ensemble 18 % des applications du corpus de référence. Une autre méthode consiste à utiliser une fonction aléatoire pour analyser automatique une partie de l'espace de solutions choisi au hasard (5 % des références). Si tous les scénarios d'un espace de solutions sont analysés, il s'agit d'une optimisation par force brute, autrement appelée approche exhaustive (8 % des références).

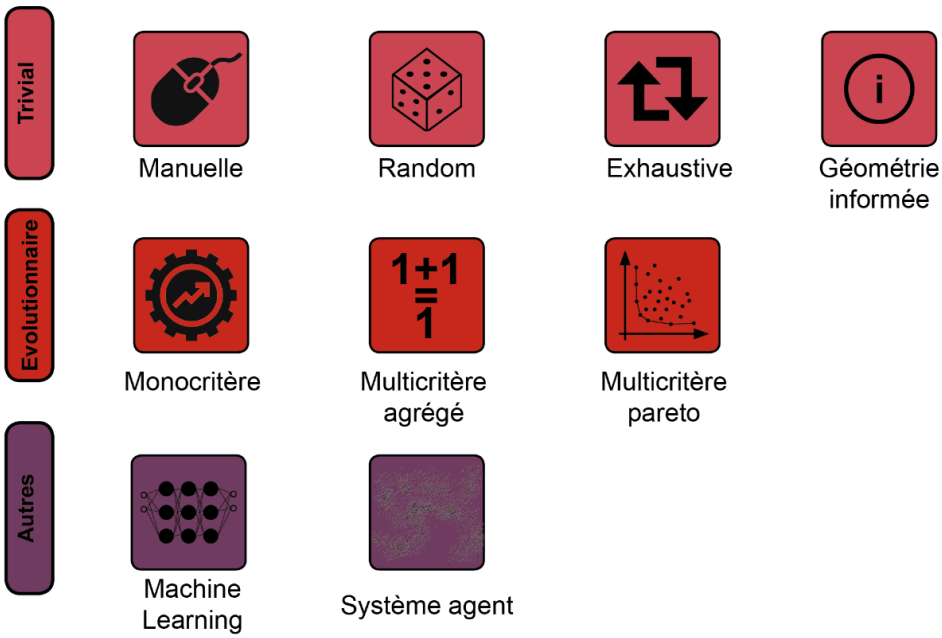


Figure 42: Logos pour distinguer les différentes méthodes d'exploration

Lorsque les auteurs utilisent des algorithmes d'optimisation, par exemple des algorithmes évolutionnaires, comme c'est le cas pour 69 % des références, nous distinguons trois approches : (1) l'optimisation monocritère où un seul critère de performance est pris en compte dans l'optimisation et où un algorithme d'optimisation monocritère est utilisé (14 %), (2) l'optimisation multicritère avec agrégation, où plusieurs critères de performance sont agrégés pour ne former qu'une seule fonction d'évaluation qui sera optimisée avec un algorithme monocritère (14 %), (3) l'optimisation multicritère avec méthode de Pareto où plusieurs critères sont pris en compte et où un algorithme d'optimisation multicritère est utilisé (7 %).

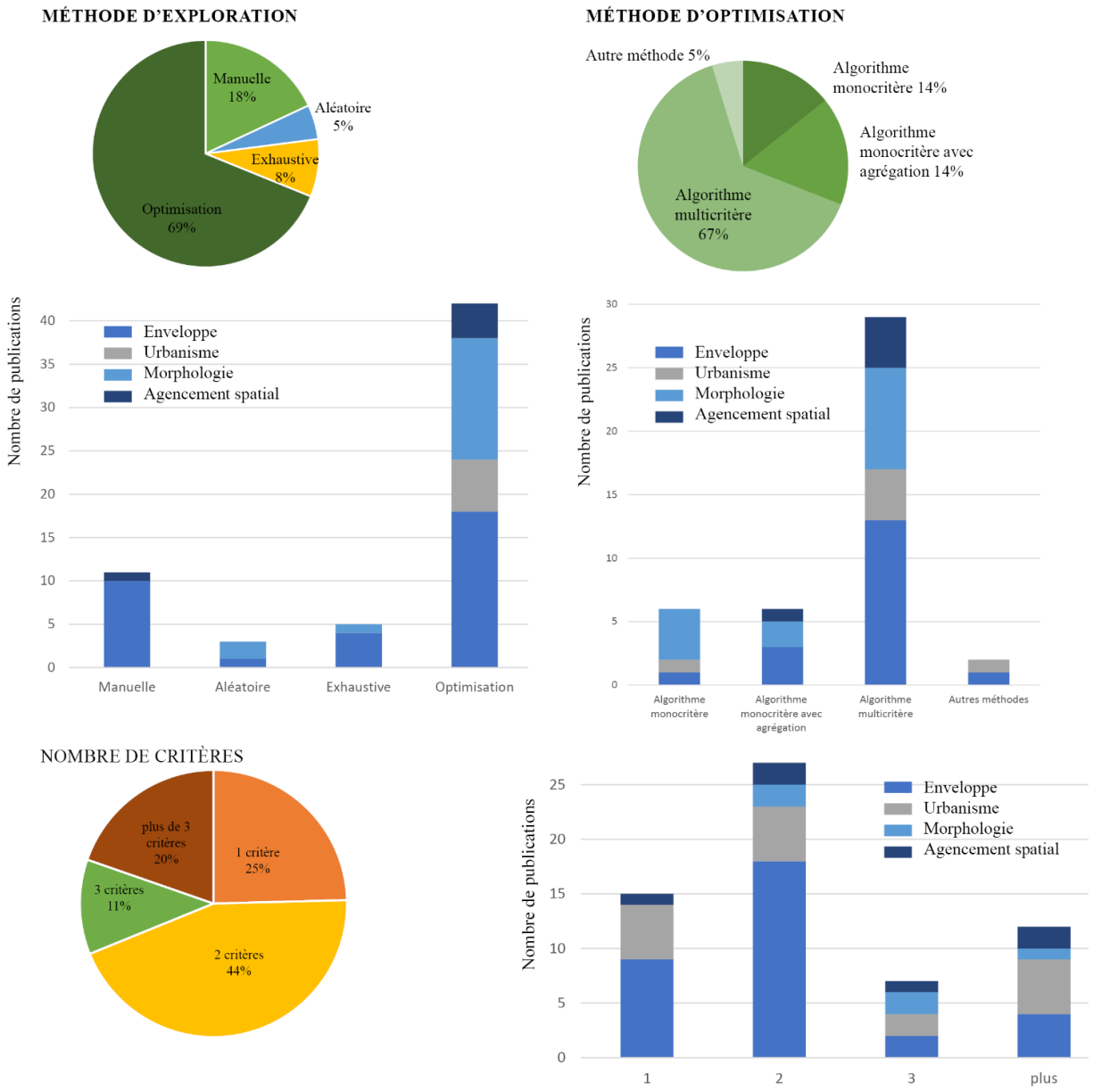


Figure 43: Statistiques sur les méthodes d'exploration étudiées du corpus de références

D’autres méthodes sont parfois utilisées (5 % des références), comme les systèmes multi-agents ou les algorithmes de *Deep Learning*. Comme le montrent les graphiques de la Figure 43, quelle que soit la technique d’exploration utilisée, 75 % des applications prennent en compte au minimum deux critères de performance ; et 20 % des applications prennent en compte plus de 3 critères de performance, dont la majeure partie des applications se font à l’échelle urbaine. Les applications sur la morphologie sont, quant à elles, systématiquement multicritères.

1.2.2. Applications à l’enveloppe

Sur les 61 références de notre corpus, 33 références s’intéressent à l’enveloppe du bâtiment. Nous verrons dans cette partie le détail les différentes caractéristiques propres à ce type d’application.

Les différents types de problèmes pour l’enveloppe

Protection solaire, percements et forme

Les applications sur les enveloppes peuvent porter sur trois types de sujet : (1) 76 % des références portent sur le système de protection solaire, (2) 18 % portent sur l’allocation du vitrage (les percements), et (3) 6 % portent sur la forme de l’enveloppe. Une grande partie (35 %) des cas d’études portant sur le système de protection solaire ont pour objectif de concevoir une façade adaptative (El Sheikh & Gerber, 2011 ; Henriques et al., 2012 ; Hosseini et al., 2019 ; Hudson et al., 2011 ; Jayathissa et al., 2018 ; Rizi & Eltaweel, 2021 ; Shen et al., 2018 ; Tabadkani et al., 2019 ; Tomasowa & Sjarifudin, 2017), c’est-à-dire une façade dont la géométrie, ou les caractéristiques évoluent dans le temps pour s’adapter au climat.

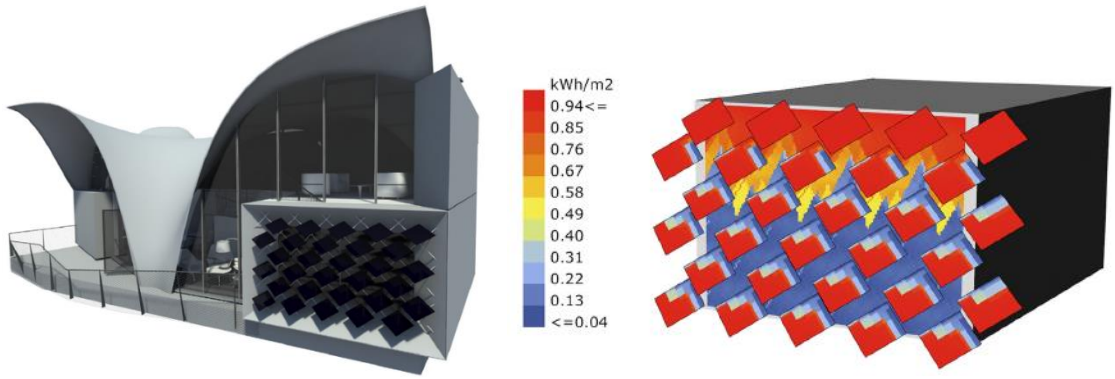


Figure 44: Façade adaptative pour l'amphithéâtre de l'unité de recherche Hilo (source: Jayathissa et al., 2018)

Par exemple, en 2018, Jayathissa et ses co-auteurs (Jayathissa et al., 2018) expérimentent une méthode d'exploration pour optimiser un système de protection et de production solaire adaptatif pour l'amphithéâtre de l'unité de recherche Hilo à Dübendorf en Suisse.

L'ENAPE peut aussi bien s'appliquer à la conception de stores simples (El Sheikh & Gerber, 2011), à des éléments courbes (Bechthold et al., 2011), ou à des systèmes plus complexes comme des moucharabiehs en brique (Abdelwahab & Elghazi, 2016), ou avec un motif islamique (Mohamadin et al., 2016), ou encore des structures pliables à l'image d'un origami (Elghazi et al., 2014; A. G. Fathy et al., 2021; Hosseini et al., 2019).

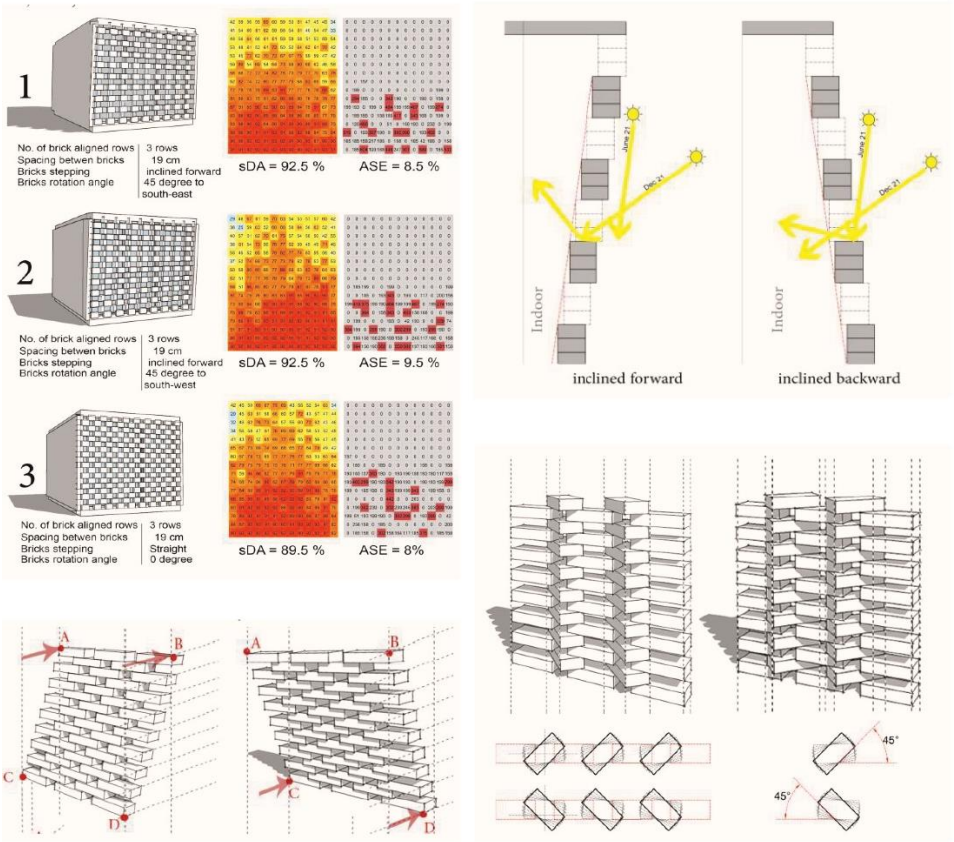


Figure 45: Etude pour la conception d'un moucharabieh en brique (source : Abdelwahad & Elghazi, 2016).

Certains auteurs traitent plusieurs sujets de l'enveloppe simultanément, comme les systèmes de protection solaire avec l'emplacement du vitrage (Gagne & Andersen, 2012). D'autres interrogent l'enveloppe et la morphologie du bâtiment (Conti, 2015; Sarvani & Kontovourkis, 2013; Youssef et al., 2018). Ce type d'applications reste rare car elles entrainent souvent une explosion combinatoire, c'est-à-dire une explosion de la taille de l'espace de solutions (et donc des temps de calcul).

Les applications qui cherchent à optimiser la forme de l'enveloppe sont moins fréquentes que les deux autres sujets. Elles sont à distinguer de celles qui optimisent la forme du bâtiment. Dans le premier cas, il s'agit simplement de déformer la peau du bâtiment pour faire varier sa prise au vent (Parascho, 2013) ou l'exposition des parties vitrées (Negendahl & Nielsen, 2015) sans pour autant faire varier la forme générale du bâtiment.

Les différentes échelles de modélisation

Selon les cas, l'exploration a lieu sur un détail de l'enveloppe, ou sur toute la façade du bâtiment. Lorsque la peau est plate et verticale, s'il s'agit d'une façade régulière pour un programme homogène comme des bureaux ou des salles de classes, la modélisation se fait à l'échelle de la pièce (F. Fathy et al., 2015 ; Gerber et al., 2017 ; Shen, 2018). Lorsqu'il s'agit d'élaborer un motif régulier qui sera répété sur l'ensemble de la façade, seulement un morceau de façade est alors modélisé pour réduire les temps de calcul des simulations (Abdelwahab & Elghazi, 2016; Elghazi et al., 2014; Kim, 2015; Tabadkani et al., 2019).

Dans plusieurs cas, il est nécessaire d'explorer l'intégralité de la façade si on cherche à optimiser la performance environnementale. Nous avons identifié trois conditions :

(1) Si le contexte urbain est hétérogène (T. Chen et al., 2019), alors les apports solaires en façade sont hétérogènes, optimiser un détail à reproduire sur toute la façade n'a alors plus de sens, comme pour le projet présenté en Figure 46.

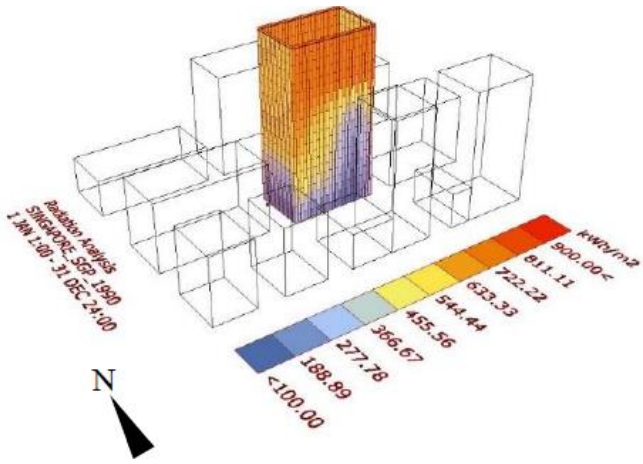


Figure 46: Optimisation de la disposition de panneaux solaire pour une tour dans un contexte dense (source: T. Chen et al., 2019)

(2) Si la forme de l'enveloppe est courbe (A. G. Fathy et al., 2021 ; Gokmen, 2013; Parascho, 2013 ; Y. K. Yi, 2019) ou complexe (avec des éléments en retrait ou en

saillie), elle aura tendance à créer de l'ombre et donc à homogénéiser les apports solaires en façade. L'enveloppe doit alors s'adapter pour être performante. Pour les applications dans les climats chauds, cette recherche d'ombre peut même justifier l'usage de l'optimisation comme le font Showkatbakhsh & Kaviani (2021) pour la rétro-conception des tours Al-Bahr à Abou Dabi aux Emirats Arabes Unis (voir Figure 47).

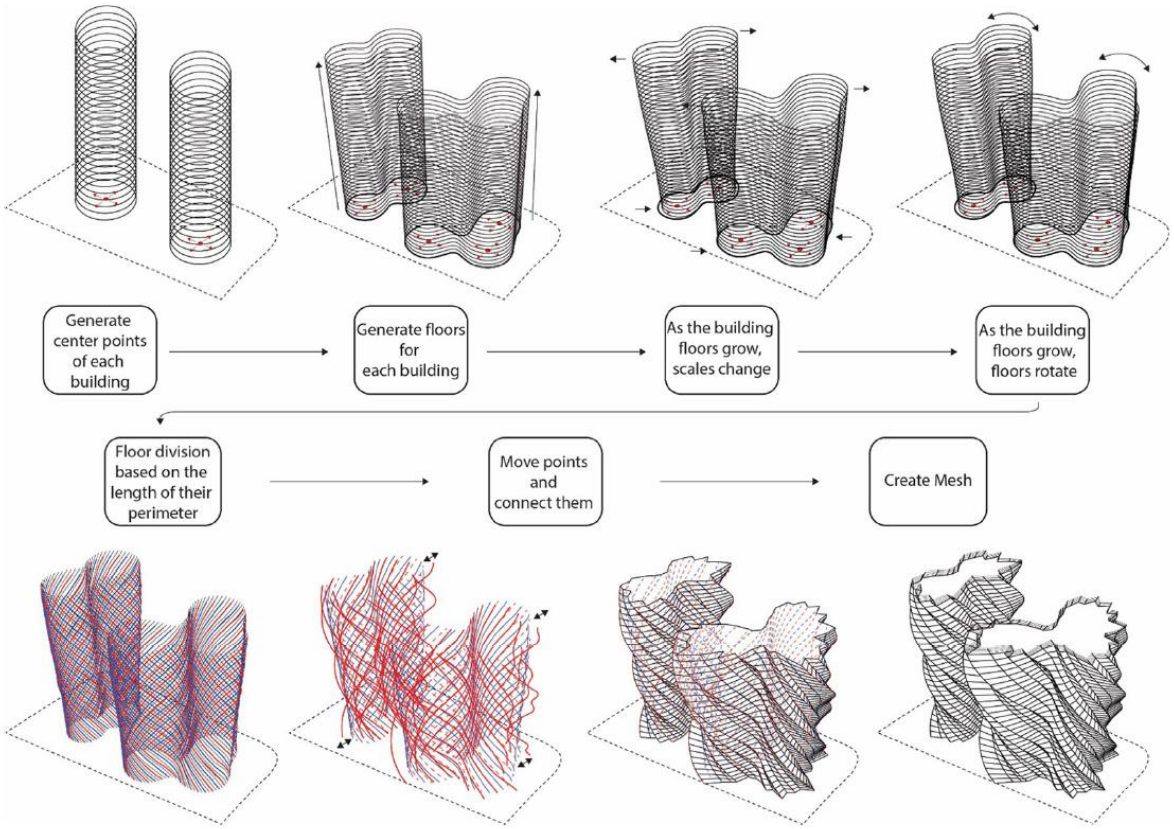


Figure 47: Méthode générative de déformation de l'enveloppe pour maximiser l'ombrage sans créer d'obstruction à la vue (source : Showkatbakhsh & Kaviani, 2021)

(3) Enfin, il faut parfois étudier l'enveloppe dans sa globalité pour des préoccupations purement architecturales. La quête du non-standard, la recherche d'effets aléatoires, de texture, ou de dégradés sont courantes chez les architectes qui conçoivent des façades « *paramétriques* ». Le cas d'étude d'Ercan & Elias-Ozkan (2015) illustre bien cette situation. Ils cherchent en déformant des lames verticales en aluminium, à maximiser une lumière indirecte dans des bureaux (voir Figure 48).

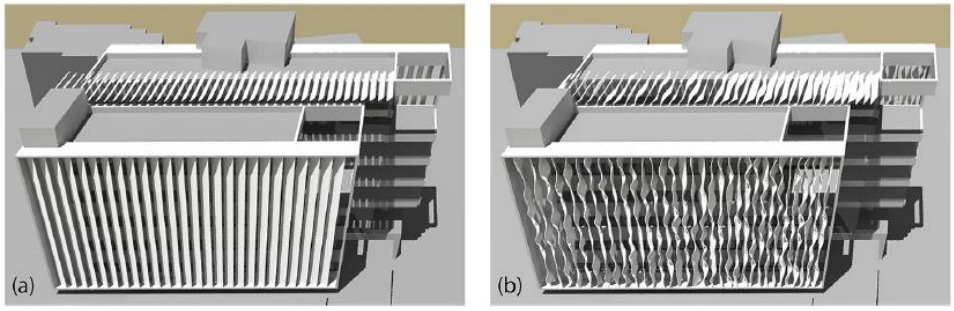


Figure 48: Système de protection solaire pour un immeuble de bureau à Lacarna en République de Cyprus (source : Ercan & Elias-Ozkan 2015)

Différentes méthodes génératives

Selon le type du problème rencontré, différentes méthodes peuvent être utilisées pour définir l'espace de solution. Elles peuvent être simplement paramétriques ou nécessiter des techniques génératives spécifiques.

Des techniques génératives spécifiques pour l'esthétique

Certains auteurs utilisent des techniques génératives (AC, MBA, GF, LS) pour générer des systèmes de protection solaire aux motifs originaux.

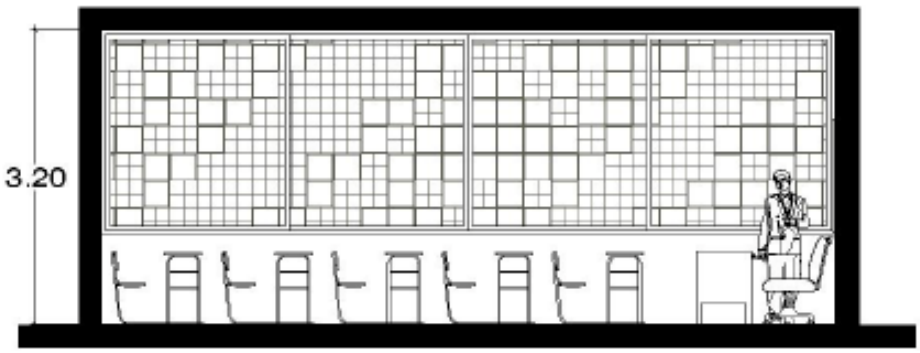


Figure 49: Exemple de façade conçue à l'aide d'un automate cellulaire (source : F. Fathy et al. 2015)

Un AC à deux dimensions et deux états (vitré ou opaque) permet de générer de nombreux motifs différents, rendant cette technique facilement applicable à la conception des enveloppes. F. Fathy et al. (2015) et Kim (2015) ont utilisé des AC pour explorer des scénarios de façades performantes pour le confort visuel. Cependant, les façades conçues à l'aide de ces AC (voir Figure 49) ont toutes un style apparenté. Dans ce cas, l'utilisation de cette technique relève plus d'une démarche esthétique que performancielle.

L'usage d'un MBA peut se justifier, comme pour les AC, uniquement pour des raisons esthétiques. C'est le cas de la façade de Agirbas (2019) générée à l'aide d'un algorithme de

« *swarm inetlligence* » (simulation en essaim). Mais cela peut aussi être utile lorsqu'il existe des contraintes difficiles à intégrer dans un modèle paramétrique, car les MBA peuvent intégrer des règles simples pour contrôler les interactions des agents entre eux. Zarrabi et al. (s. d.) ont élaborés une méthode pour la conception d'une façade piézolélectrique à Stockholm (voir Figure 50). Cette tour de bureaux est recouverte de barres articulées qui bougent en fonction du vent et produisent ainsi de l'électricité. La taille des barres et la distance qui les sépare sont variables. Pour trouver la disposition idéale des barres et générer un maximum d'énergie tout en évitant les collisions, un MBA a été utilisé. Cette méthode générative est chronophage, ainsi l'exploration de l'espace de solutions sur cette application s'est faite manuellement sur quelques scénarios.

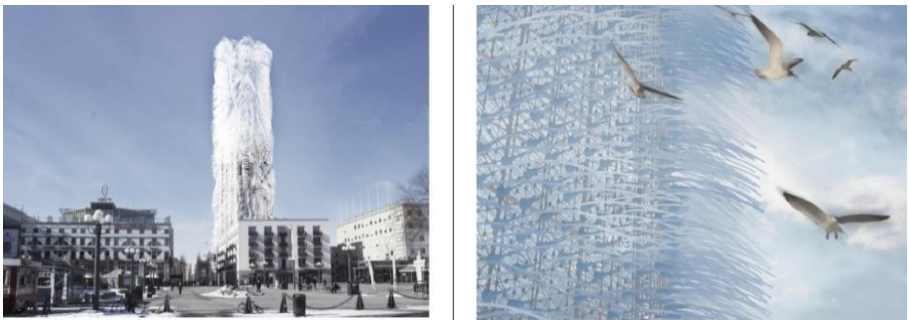


Figure 50: MBA pour une façade piézoélectrique (source: Zarrabi et al.,s. d.)

En 2017, Tomasowa & Sjarifudin (2017) utilisent une GF pour optimiser les protections solaires de la façade adaptative d'une tour de bureaux à Jakarta. Les règles de cette grammaire n'intègrent pas de condition de performance mais permettent de générer des scénarios topologiquement différents, difficilement reproductibles avec un modèle paramétrique.

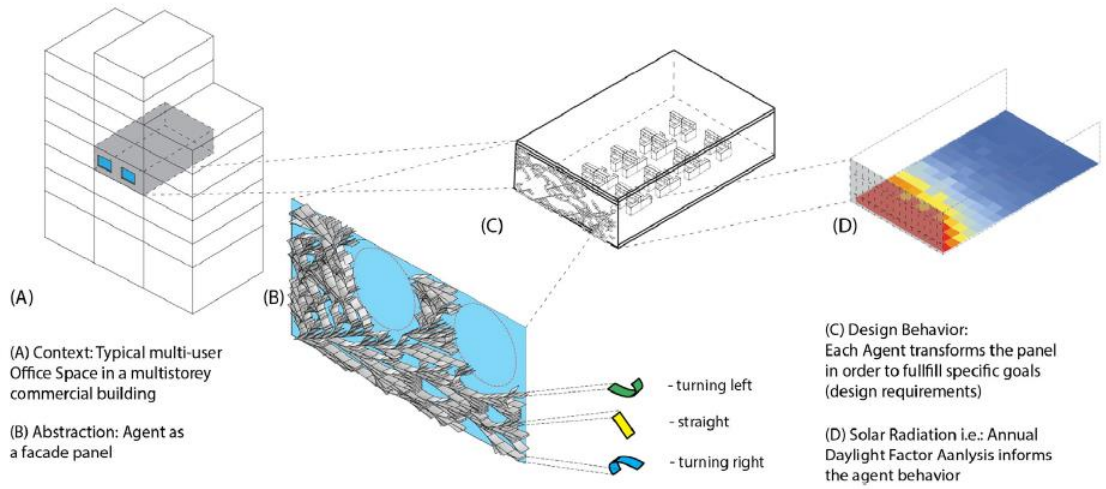


Figure 51: Système de protection solaire reproduisant la croissance d'une plante (source: Gerber et al., 2017)

Deux références de notre corpus utilisent des processus génératifs inspirés de la croissance des plantes (qui ne sont pas des LS à proprement parler, mais ont un objectif comparable). Gokmen (2013) utilise des données de radiation pour guider la croissance des protections solaires sur deux tours, et Gerber et al. (2017) utilisent aussi ce type de procédé pour le moteur génératif d'une façade optimisée à l'aide d'un système multi-agent (voir Figure 51).

Les modèles paramétriques

Figure 52: Protection solaire adaptative (source: El Sheikh & Gerber, 2011)

La majorité des exemples pratiques identifiés pour l'enveloppe utilisent des modèles paramétriques, dont la grande majorité est développée à l'aide de *Grasshopper*®, outil de programmation visuel du modeleur 3D *Rhinoceros*®. Seulement un exemple a été développé sur *Dynamo*® (A. G. Fathy et al., 2021). La moitié de ces modèles sont « *simples* », c'est-à-dire qu'ils ne nécessitent pas de méthode de modélisation particulière comme des outils de discrétisation, des systèmes d'attraction avec des champs de force, ou par exemple des générateurs de motifs islamiques.

Un modèle paramétrique simple contient des paramètres géométriques standards comme une longueur, largeur, hauteur, épaisseur, un angle de rotation ou un rayon de courbure. Quel que soit le sujet à étudier, le modèle est souvent constitué d'une multitude d'éléments architecturaux : plusieurs lames, plusieurs parties vitrées, plusieurs briques… etc. Avec un modèle paramétrique simple, les paramètres de l'ensemble des éléments varient de façon uniforme, et génèrent des façades au design régulier. Avec ce type de modèles, la taille de l'espace de solutions est souvent raisonnable, comme pour la façade adaptative de El Sheikh & Gerber (2011), illustrée en Figure 52 où deux paramètres sont utilisés pour gérer l'orientation de toutes les lames.

Lorsqu'on souhaite générer des scénarios plus complexes avec un modèle paramétrique simples, alors le nombre de paramètres se multiplient avec le nombre d'éléments. La taille de l'espace de solutions est alors décuplée, ce qui peut, selon la méthode d'évaluation utilisée, fortement impacter les temps de calcul et ainsi remettre en question la faisabilité de l'approche. Ercan & Elias-Ozkan (2015) ont utilisé un modèle paramétrique simple pour explorer à l'échelle globale une façade au design irrégulier (voir Figure 48). Dans ce cas, 33 lames verticales subissent toutes des torsions à l'aide d'un paramètre d'angle de rotation pour chacun des 16 points de chaque lame. Ainsi, le modèle est composé de 528 paramètres différents pouvant prendre 90 valeurs différentes (de -45° à +45°). Alors un nombre immense de scénarios sont possibles pour la façade principale du bâtiment, 90^{528} scénarios très précisément (voir Figure 53). On parle alors d'« *explosion combinatoire* ». Avec un espace de solutions de cette dimension, il est selon nous, avec la puissance de calcul des ordinateurs de notre époque, illusoire d'espérer obtenir des résultats satisfaisants dans une temporalité compatible avec la pratique professionnelle.

$90^{528} =$

69190263406982301020261252279923524203295835
03333541160493114652357003821505320239185219
85589205120343540467249134997700698176361299
54834040549796568473333729184396997909759676
68786318165883179199487913369478961304466918
98684967683024892823611116787824407916003378
22680429851274616149871170748835651811950118
19138655071572833986240358430082734093168146
74374709691204184388350905153096471288914735
86396227228988995429987629464273511111623496
96153403779605835094784694786858123737528347
63240619802203131521000000000000000000000000
00000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000
00000000000000000000

Figure 53: Taille de l'espace de solution de l'application de Ercan et Elias-Ozkan (2015)

Certaines techniques de modélisation paramétrique plus complexes à mettre en œuvre facilitent la génération de compositions irrégulières en évitant l'explosion combinatoire. Les structures pliées, ou en origami permettent d'assembler les éléments de la façade dans un unique système où toutes les parties sont imbriquées. Ainsi, quelques paramètres suffisent pour faire varier toute une façade. En 2015, Negendahl & Nielsen (2015) ont utilisé cette technique afin de concevoir une façade de bureaux pour un projet non dévoilé de l'agence BIG. Sur la Figure 54, on peut observer en plan la diversité de scénarios envisageables avec seulement trois paramètres.

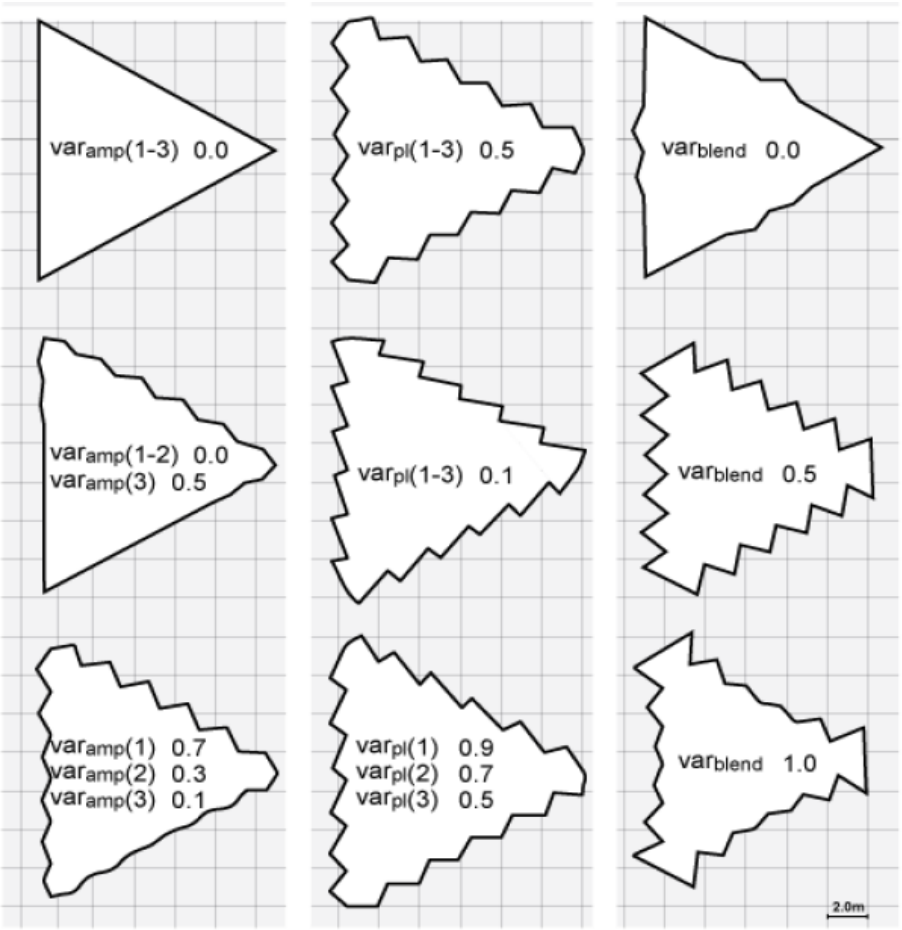


Figure 54: Déformation de l'enveloppe avec 3 paramètres (source : Negendahl & Nielsen, 2015)

Une autre technique consiste à utiliser des outils de discrétisation qui facilitent le découpage de la façade en panneaux à partir de surfaces pouvant être courbes. Cette technique de modélisation paramétrique est récurrente da s notre corpus (Abdelwahab & Elghazi, 2016 ; T. Chen et al., 2019 ; F. Fathy & Fareed, 2017 ; Hosseini et al., 2019 ; Hudson et al., 2011 ; Narangerel et al., 2016 ; Rizi & Eltaweel, 2021). Lorsque qu'on cherche à optimiser une enveloppe discrétisée, la combinatoire peut rapidement exploser. Si tous les panneaux sont

identiques (Narangerel et al., 2016), ou s'il y a peu de panneaux (Hosseini et al., 2019) alors l'espace de solutions reste gérable. Cependant, si les panneaux sont nombreux ou tous différents, et si la façade doit être étudiée dans sa globalité alors la tache devient complexe. Cette problématique a été soulignée par Chatzikonstantinou et al. (2019) dans l'étude d'une façade modulaire pour des laboratoires à Delft aux Pays-Bas. Un système de champs de forces avec des points d'attraction est utilisé pour contrôler l'orientation des panneaux 3D, comme l'illustre la Figure 55. Cette contrainte permet de créer des effets de dégradé en façade tout en réduisant le nombre de paramètres et la taille de l'espace de solutions.

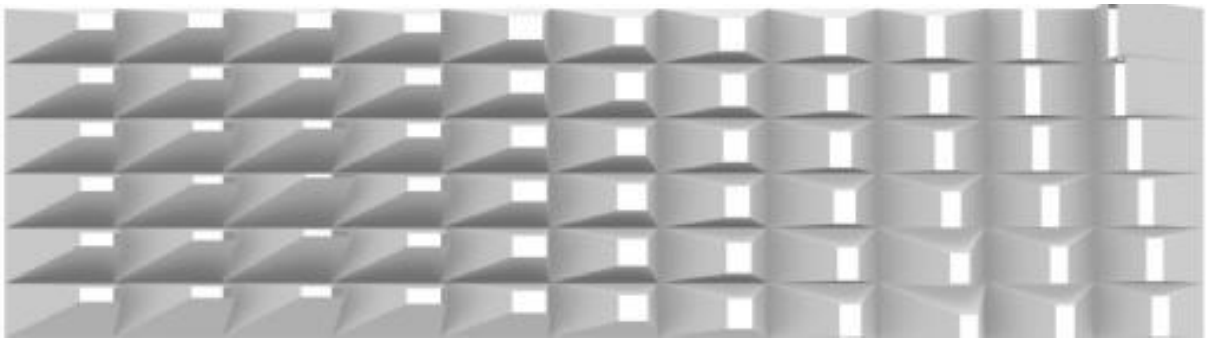


Figure 55: Modèle paramétrique avec des panneaux contrôler par un système de champ de force (source: Chatzikonstantinou et al., 2019)

Les indicateurs pour l'évaluation de la performance de l'enveloppe

L'analyse de l'éclairage naturel est de loin le critère le plus représenté pour l'optimisation des enveloppes puisqu'il représente plus de 70 % des références du corpus. L'ensoleillement pour le contrôle des apports solaires est étudié dans plus de 35 % des cas et les consommations énergétiques dans presque 23 % des cas. D'autres critères sont beaucoup plus rarement employés comme la qualité de la vue vers l'extérieur rencontrée dans seulement deux références (Conti, 2015; Showkatbakhsh & Kaviani, 2021) et le critère esthétique rencontré à une unique occasion (Y. K. Yi, 2019). Les critères les plus souvent associés au confort visuel sont la prédiction des consommations énergétiques ou l'analyse de l'ensoleillement.

Pour évaluer un même critère, les auteurs utilisent parfois des indicateurs différents, comme le facteur de lumière du jour et l'autonomie en lumière du jour, pour le confort lumineux. A l'inverse, certains indicateurs peuvent répondre à des objectifs différents. Par exemple, l'analyse de la radiation solaire peut servir à réduire les apports solaires pour maximiser le confort thermique d'été, ou encore à maximiser les apports solaires sur des surfaces photovoltaïques pour maximiser la production d'énergie.

Comme on peut le voir sur la Figure 56, huit indicateurs différents au total sont utilisés pour la mesure de l'éclairage naturel, le plus populaire étant l'éclairement utile (présent dans 43 % des cas). On retrouve aussi souvent l'autonomie en lumière du jour (35 %), le facteur de lumière du jour (26 %), l'éclairage naturel (22 %), l'exposition annuelle à la lumière du soleil (17 %) et plus rarement l'intensité lumineuse (4 %), les indicateurs d'éblouissement (4 %) et l'éclairage disponible significatif (4 %). Souvent plusieurs indicateurs sont pris en compte dans une même étude. Le choix de ces indicateurs est rarement justifié et renvoie surtout à des réglementations ou labels environnementaux. De même, certains font des simulations annuelles (57 %), là où d'autres se concentrent sur une saison (4 %), sur une journée type (4 %) ou sur les équinoxes et solstices (26 %). Ici aussi, le choix n'est que rarement expliqué. On observe tout de même que ceux qui font des analyses sur une journée seulement (jour type ou équinoxes et solstices) utilisent systématiquement des algorithmes d'optimisation. On imagine donc que cette simplification s'explique par le besoin de réduire les temps de calcul.

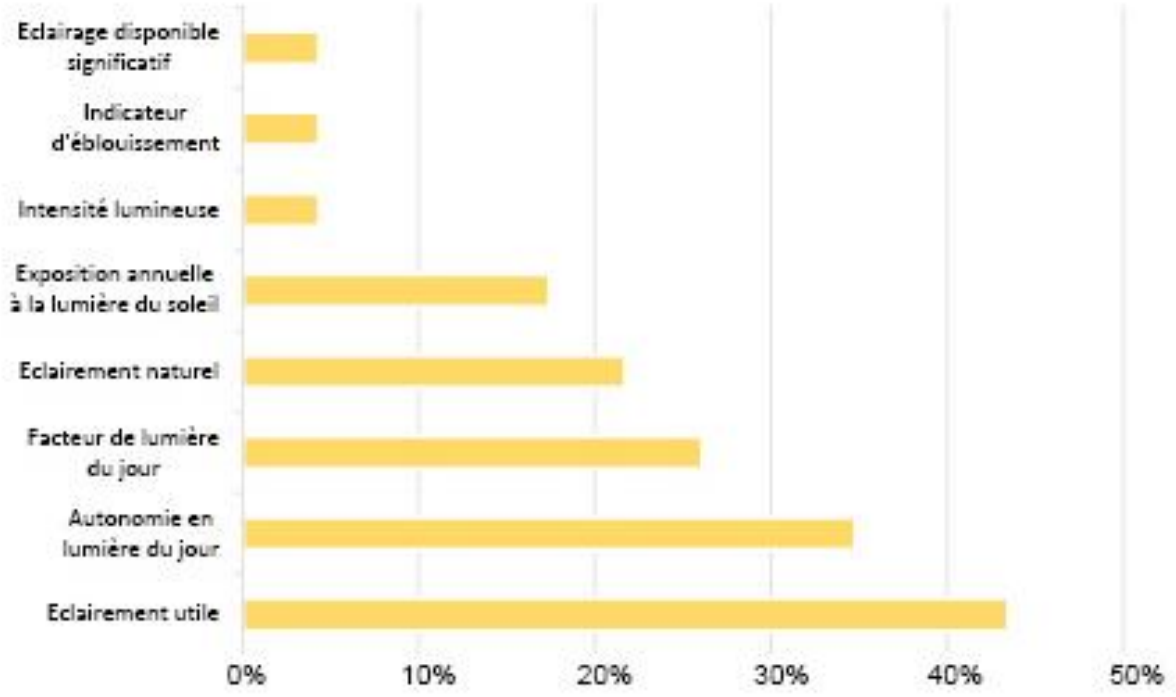


Figure 56: Les indicateurs de l'analyse de l'éclairage naturelle utilisés dans les applications sur l'enveloppe du bâtiment

Les techniques d'exploration pour l'enveloppe

L'algorithme d'optimisation est la technique la plus utilisée pour l'exploration des façades. Dans notre corpus, l'usage de l'optimisation pour les façades adaptatives est moins

courant que pour les façades traditionnelles. Même si les algorithmes d'optimisation sont dans certains cas seulement évoqués, ils ne sont jamais utilisés avec d'autres techniques génératives (comme les GF, MBA, LS, AC) mais uniquement avec des modèles paramétriques. Lorsque l'algorithme d'optimisation est précisé, il s'agit le plus souvent d'un algorithme génétique. Le paramétrage des algorithmes utilisés est rarement précisé, et jamais expliqué. Les temps de calculs sont rarement précisés, lorsqu'ils le sont, il s'agit de plusieurs dizaines d'heures (Abdelwahab & Elghazi, 2016; Chatzikonstantinou et al., 2019; Ercan & Elias-Ozkan, 2015).

Une méthode spécifique à l'application sur les enveloppes a été identifiée. Elle permet de créer de la diversité en façade sans augmenter la combinatoire et consiste à utiliser la « *géométrie informée* », ou « *informed geometry* » en anglais (Anton & Tănase, 2016). Dans ce cas, les irrégularités en façade proviennent de données attribuées à la géométrie à l'aide d'une évaluation antérieure. Quatre références de notre corpus utilisent cette technique. En 2013, (Gokmen, 2013) utilise les données résultant d'une étude de la radiation en façade pour dimensionner des protections solaires (voir Figure 57). En 2019, (T. Chen et al., 2019) font de même pour dimensionner des panneaux photovoltaïques. En 2018, (Zarrabi et al., s. d.) utilisent les données aérauliques de la pression et la vitesse des vents pour une façade piézoélectrique. Pour ces trois exemples, la géométrie qui doit être « *informée* » est une donnée fixée du problème, le critère utilisé n'a donc pas besoin d'être évalué plusieurs fois.

Parfois, la combinatoire risquant fortement d'exploser, il est possible de mélanger plusieurs techniques. C'est par exemple le cas lorsqu'on cherche à interroger à la fois la forme du bâtiment et le design de sa façade. Showkatbakhsh & Kaviani (2021) ont utilisé la technique de la « *géométrie informée* » pour créer des panneaux 3D adaptés à l'exposition solaire avant d'optimiser la forme de l'enveloppe pour la vue et l'ensoleillement (voir Figure 47). Dans ce cas, la radiation solaire et la qualité de la vue sont évaluées plusieurs fois, à chaque fois qu'une nouvelle forme pour les bâtiments est générée. Ces évaluations, même si elles peuvent impliquer des outils similaires, sont à distinguer de celles de la méthode d'évaluation. L'intégration de ces évaluations dans la méthode générative constitue alors une forme de « *préoptimisation* ». Les critères utilisés en phase générative peuvent ou non être réemployés en phase d'évaluation pour être intégrés dans le processus d'optimisation.

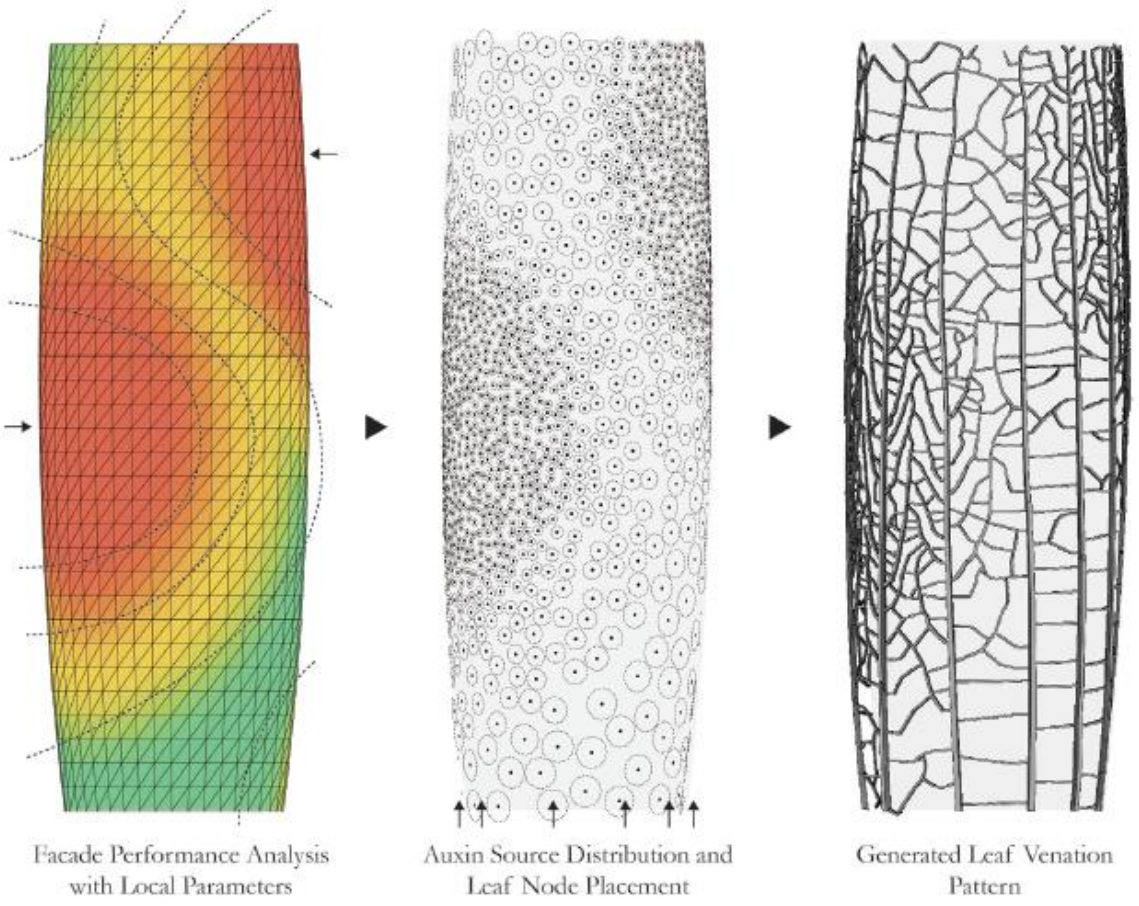


Figure 57: Modèle génératif inspiré de la croissance de plantes et géométrie informée (source : Gokmen, 2013)

1.2.3. Fabrique urbaine, morphologie et agencement spatial

Nous avons identifié trois autres types d'application nettement moins répandus que l'enveloppe dont nous avons présenté les caractéristiques dans cette dernière partie du chapitre. Nous présentons ces applications en partant de l'échelle la plus large à la plus petite.

Fabrique urbaine

Historiquement, l'optimisation a d'abord été appliquée à l'échelle architecturale avant l'échelle urbaine. Nous avons identifié 3 types d'approches génératives à l'échelle urbaine :

(i) la simulation de l'expansion de la ville (Nugroho & Al-Sanjary, 2018),

(ii) la génération de formes urbaines à l'échelle de l'îlot urbain qui ne prend pas en compte les problématiques liées à l'accessibilité,

(iii) les processus génératifs de la fabrication urbaine qui comprennent à la fois l'optimisation du réseau viaire, du parcellaire, et la morphologie des bâtiments ainsi que leur agencement dans la ville.

La première catégorie n'est pas exploratoire et nous ne l'aborderons pas dans cette thèse. La deuxième catégorie est la plus représentée en conception performancielle, nous évoquerons ce type d'applications dans la partie sur la morphologie des bâtiments. La troisième catégorie, plus ambitieuse, compte peu d'exemples dans la littérature (6 dans notre corpus) que nous exposerons dans cette partie après avoir présenté brièvement les méthodes génératives du réseau viaire et des bâtiments.

L-système pour le réseau viaire

Pour générer un plan urbain, il faut commencer par générer les rues, la subdivision en îlots puis le parcellaire. Un générateur pour une ville avec trame orthogonale est facilement modélisable avec un modèle paramétrique. Générer des tissus urbains s'insérant dans des villes organiques est informatiquement parlant beaucoup plus complexe. La trame orthogonale est adaptée pour faire des études paramétriques, définir des grands principes de conception, mais pas pour faire de la conception urbaine générative pour des villes européennes.

Plusieurs études ont déjà abouti à des outils disponibles sur *Grasshopper*® pour générer des tissus urbains organiques :

(1) DecodingSpaces (R. T. Koenig, 2013) qui s'inspire des travaux de Parish & Müller (2001) et Vanegas et al. (2009).

(2) City generator (Mei et al., 2021) qui s'inspire des travaux de Parish & Müller (2001)

(3) Components for parametric urban design in Grasshopper (Schneider et al., 2011) qui s'inspirent directement de la méthode de Weber et al. (2009).

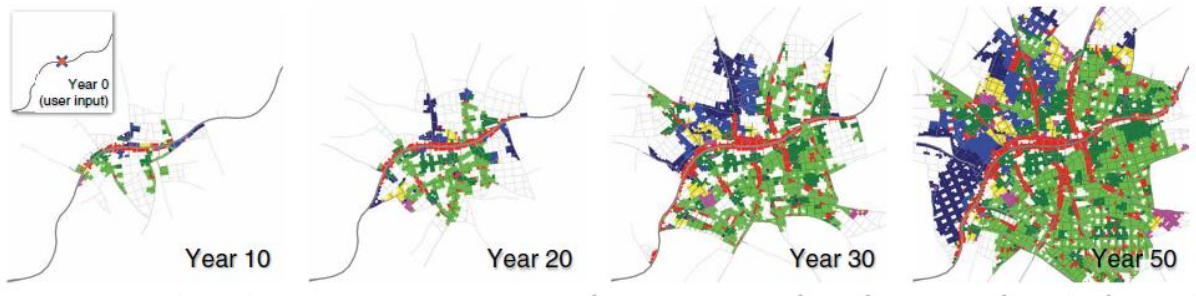


Figure 58: Simulation de l'expansion d'une ville (source: Weber et al., 2009)

Paris et Müller s'appuyant eux-mêmes sur les travaux de Weber et al. (2009), tous les outils cités ci-dessus s'inspirent donc de la méthode de Weber et de ses co-auteurs qui utilisent des LS pour générer le réseau viaire. Ces chercheurs ont notamment développé en 2008 le modeleur 3D d'environnement urbain connu et reconnu : *CityEngine*®. Il est utilisé en géoconception, dans l'industrie du cinéma, du jeux vidéo, par l'armée, les chercheurs en urbanisme, et par quelques concepteurs (Foster+Patner). Ce logiciel utilise une approche procédurale à base de GF et de LS pour simuler l'expansion des villes (voir Figure 58). Le LS fonctionne en deux étapes à chaque itération. Une première où le processus génère une solution générique en fonction des contraintes globales comme la densité urbaine ou le type de tissu urbain (basique, rectangulaire ou concentrique) qui sera ensuite modifiée si besoin en fonction de contraintes locales (voir Figure 59).

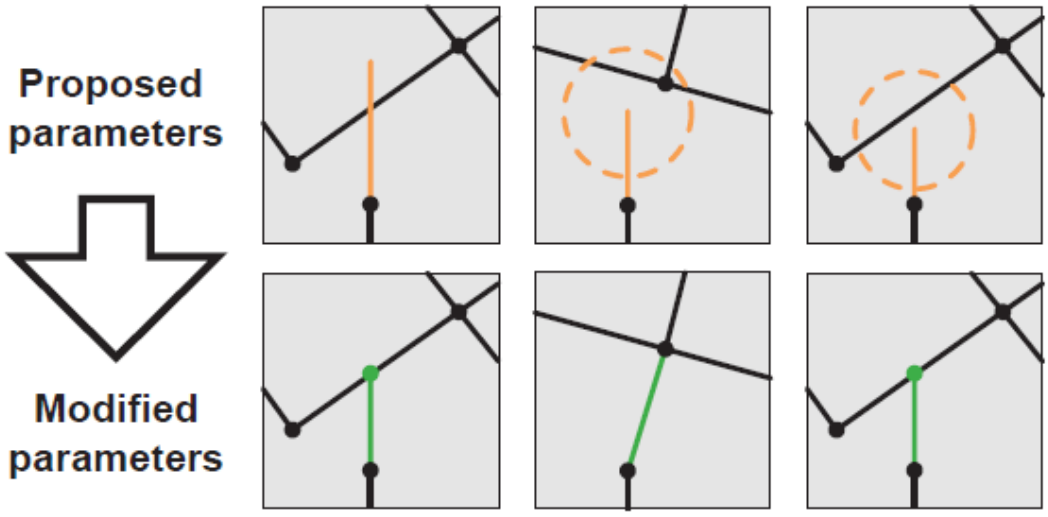


Figure 59 : Contraintes locales pour la génération d'un réseau viaire organique (source: Weber et al., 2009)

Des typologies pour les bâtiments 3D

Après la génération du réseau viaire, il faut subdiviser les îlots en parcelles, distribuer le programme dans les îlots, puis générer la géométrie de chaque îlot. Dans le logiciel *CityEngine*®, les îlots sont divisés en lot. Un bâtiment sera ensuite généré par lot. Trois types de bâtiments peuvent être générés : gratte-ciel, commerces, et logements. Une empreinte est générée arbitrairement, puis un LC vient déformer l'empreinte pour créer des géométries variées. Cette méthode est adaptée pour faire de la simulation de croissance mais pas pour l'optimisation où l'espace de solutions ne peut pas être infini.

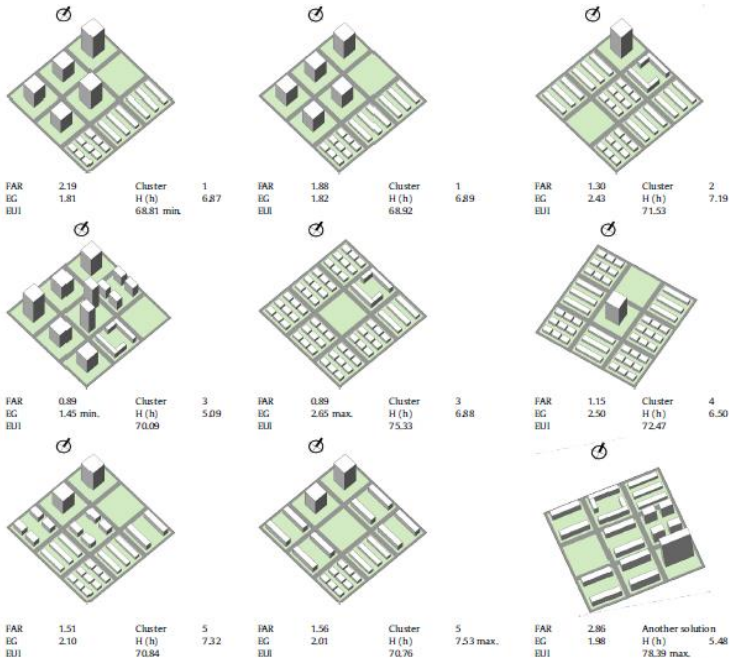


Figure 60: Génération de 9 îlots avec un système de typologies (source : W.Wang et al., 2021)

Pour chaque îlot l'espace de solutions peut être immense, pour optimiser la performance, les chercheurs travaillent avec des géométries qui restent simples. Pour pouvoir proposer des topologies différentes il est possible de fonctionner par typologies. La méthode la plus triviale (voir Figure 60) consiste à définir des typologies sans variables (W. Wang et al., 2021), qui peuvent parfois s'adapter à la forme de la parcelle et à la taille du programme (C. Huang et al., 2022). L'unique variable par îlot est alors celle de la typologie. Les typologies courantes sont l'îlot fermé sans ou avec interruptions, l'îlot ouvert, ou mixte, l'immeuble de grande hauteur et les maisons en bande. Lorsqu'il y a peu d'îlots on peut ajouter des paramètres tels que la hauteur et longueur de certains bâtiments (Duering et al., 2020). Il est aussi possible de déformer des bâtiments à partir d'une première esquisse (Taleb & Musleh, 2015).

Deux références sur six interrogent simultanément le réseau viaire et l'emplacement où la forme des bâtiments (Duering et al., 2020; Nagy et al., 2018), trois interrogent uniquement le parcellaire (C. Huang et al., 2022), ou la morphologie du bâtiment (Taleb & Musleh, 2015; W. Wang et al., 2021), et un seul interroge l'un puis l'autre (R. Koenig et al., 2020).

Les critères de performances à l'échelle urbaine

Le développement des techniques de syntaxe spatiale, qui permettent d'analyser rapidement les configurations spatiales, a augmenté le nombre de méthodes d'analyse appliquées à l'échelle urbaine et donc la quantité de critères d'optimisation (Miao et al., 2020). Les critères ne sont pas les mêmes qu'à l'échelle architecturale. Z. Shi et al. (2017) ont listé les métriques utilisées (autres que la performance énergétique) en conception urbaine performancielle et générative (voir le Tableau 4). Une grande quantité de ces critères peut facilement être reproduit avec un outil de programmation visuel et n'implique pas des temps de calcul importants.

Tableau 4 : Liste des critères d'optimisation pour l'échelle urbaine (source: Z.Shi et al., 2017)

longueur des îlots	densité des rues
densité des îlots	distance entre les commerce et les parcs
score d'accessibilité pour les piétons, vélos, transit	accessibilité des commerces et transports en commun pour les piétons
le dégradé de densité d'occupation des sols	le ratio de surface de plancher
la taille de la parcelle	la densité urbaine
la profondeur de la parcelle	facteur de vue du ciel
ratio surface/volume	index de forme urbaine
mixité de l'utilisation des sols	densité d'occupation des sols

Lorsque de véritables analyses énergétiques sont effectuées à l'échelle urbaine, l'usage de méthodes de substitution comme le *Machine Learning* semble s'imposer, même lorsqu'il s'agit d'optimiser seulement quelques îlots (Duering et al., 2020 ; C. Huang et al., 2022 ; W. Wang et al., 2021). Lorsque que l'échelle est presque celle d'une ville entière, les méthodes d'analyse se restreignent à la syntaxe spatiale (R. Koenig et al., 2020).

Morphologie des bâtiments

L'optimisation de la forme des bâtiments est la seconde application la plus fréquente de notre corpus après l'enveloppe, avec 7 références pour l'optimisation de l'îlot urbain et 11 références pour l'optimisation d'un unique bâtiment. Il existe désormais plusieurs outils génératifs payants en ligne pour aider les architectes, et maîtres d'ouvrage en phases amonts de projet (kreo.net, testfit.io, iamplooto.com, autodesk.com/products/spacemaker/overview).

Explorations à l'échelle de l'îlot urbain

Générer une grande variété de géométries différentes à partir d'une feuille vide est complexe d'un point de vue informatique. Lorsqu'on travaille à l'échelle du quartier, il est nécessaire de restreindre cette diversité pour éviter l'explosion combinatoire, mais lorsqu'on travaille à l'échelle de l'îlot il est souvent souhaitable de pouvoir prendre en compte un maximum de morphologies différentes.

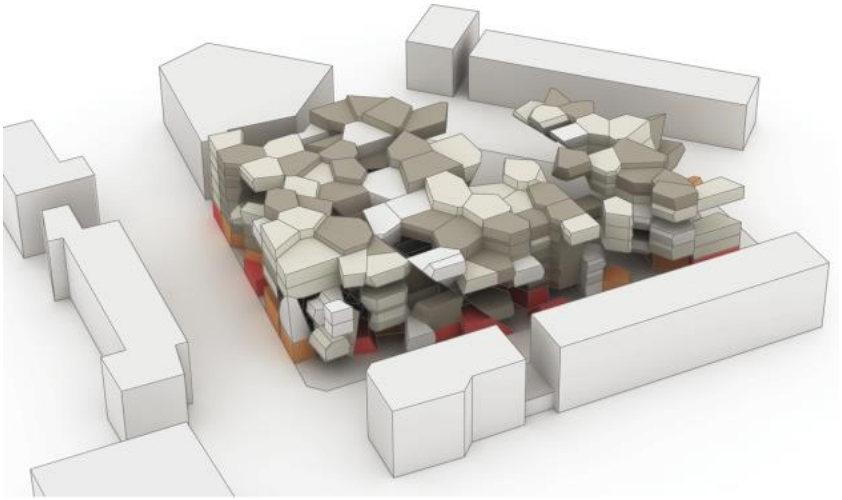


Figure 61 : Voxels et diagramme de Voronoï pour la génération d'îlot (source: Menges, 2012)

La méthode générative employée dépend de l'objectif. Comme pour l'échelle urbaine, certains auteurs s'appuient sur des typologies, comme Kämpf et al. (2010) qui comparent des terrasses plates, des terrasses inclinées et des patios pour l'optimisation des apports solaires des espaces extérieurs. D'autres cherchent à interroger la disposition des bâtiments entre eux. Pour éviter les problèmes de superposition des bâtiments, certains chercheurs utilisent un système de discrétisation en cellules 2D (António et al., 2014; Vermeulen et al., 2018), là ou d'autres développent une approche particulière avec des points de contrôles (Y. K. Yi & Kim, 2015).

Pour pouvoir explorer un maximum de configurations possibles, l'utilisation du voxel (un pixel en volume, soit une discrétisation 3D) est efficace. Marsault & Torres (2019) utilisent

cette approche avec des voxels cubiques pour générer des îlots et Menges (2012) propose une version du voxel avec un diagramme de Voronoi 2D extrudé (voir Figure 61).

Cependant, ce type de méthodes est difficile à appliquer dans la pratique professionnelle où les contraintes, notamment urbaines, sont nombreuses. Mukkavaara & Sandberg (2020) développent un générateur d'îlots pour répondre à un problème spécifique qui contient notamment des contraintes urbaines comme l'alignement aux rues et les hauteurs maximums. La génération des bâtiments se fait en trois temps, illustrés en Figure 62. Une première étape consiste à générer un volume limite avec 5 paramètres pour chaque rue. La deuxième étape consiste à inscrire des parallélépipèdes dans le volume limite. La troisième étape fusionne les bâtiments définis comme trop proche en extrudant un bâtiment.

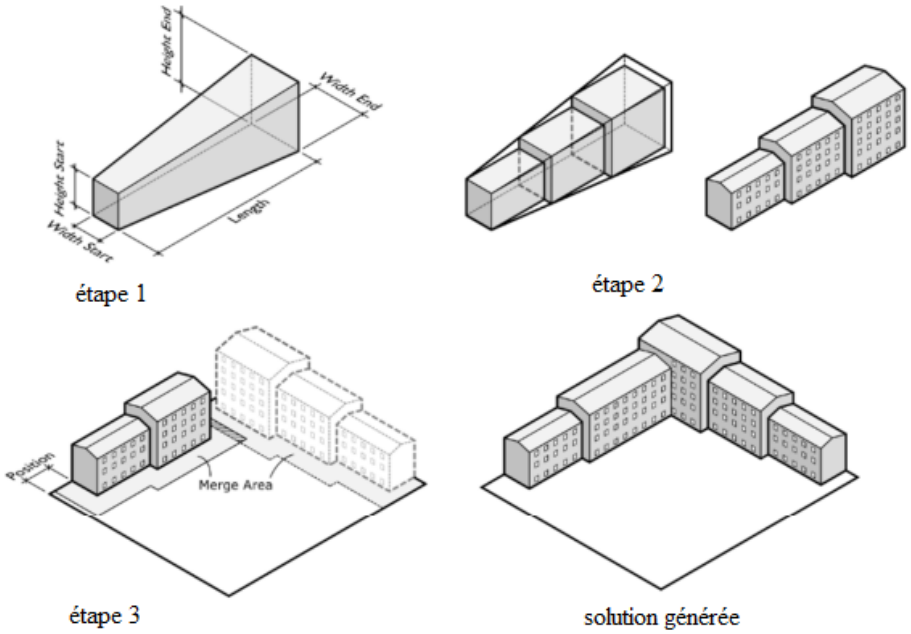


Figure 62: Méthode générative d'îlot en trois étapes (source: Mukkavaara et al., 2020)

Le critère de performance le plus interrogé à cette échelle est le confort thermique avec l'étude des apports solaires. Lorsque les consommations énergétiques sont évaluées, une méthode de substitution est utilisée, il s'agit ici d'une régression (Marsault & Torres, 2019). Dans notre corpus, 3 références sur 6 sont des optimisations monocritères, qui étudient les apports solaires (António et al., 2014 ; Vermeulen et al., 2018 ; Y. K. Yi & Kim, 2015). La seule référence qui intègre des contraintes urbaines n'utilisent pas d'algorithme d'optimisation, il s'agit d'une exploration aléatoire (Mukkavaara & Sandberg, 2020).

Exploration à l'échelle du bâtiment

A cette échelle, le risque d'explosion combinatoire est moins important. Il existe différentes méthodes génératives selon qu'on parte d'une feuille blanche ou d'une première esquisse. Certains interrogent la forme du bâtiment en jouant simplement sur les paramètres de son empreinte (K. W. Chen et al., 2018). Il s'agit donc de déformer une courbe plane fermée. D'autres utilisent la même méthode en 3D où plusieurs courbes planes fermées sont déformées à des altitudes différentes (Gerber & Lin, 2014; Menges, 2012; Suyoto et al., 2015; H. Yi, 2014). La forme résultante est la surface lissée à partir des différentes courbes planes. Cette méthode est la plus utilisée de notre corpus de 11 références sur la morphologie du bâtiment.

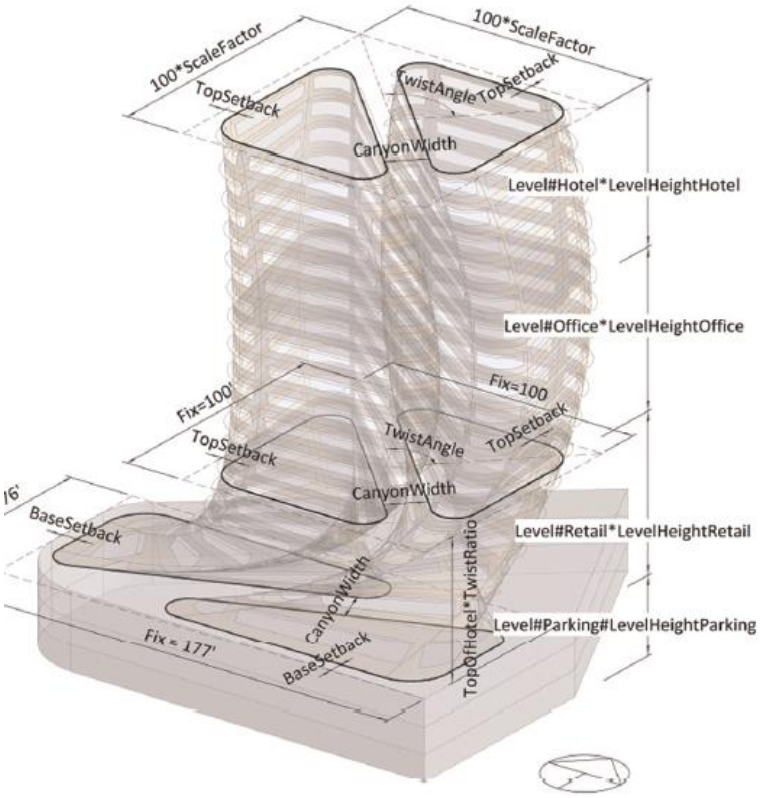


Figure 63: Exploration à partir de la déformation d'une esquisse (source: Gerber & Lin, 2014)

Le problème se complique lorsqu'on souhaite respecter des contraintes comme les règlements urbains (Y.-S. Huang et al., 2015), ou conserver une surface de plancher constante.

Comme pour l'échelle urbaine, à l'échelle de l'îlot, lorsqu'on souhaite faire de l'exploration sans partir d'une idée prédéfinie, il faut tester différentes typologies comme les formes en L, H, I, U, T, la croix, le trapèze (Ciardiello et al., 2020 ; Z. Li et al., 2018 ; Youssef et al., 2018). L'inconvénient avec cette méthode est qu'on reste sur des géométries très classiques (voir Figure 64).

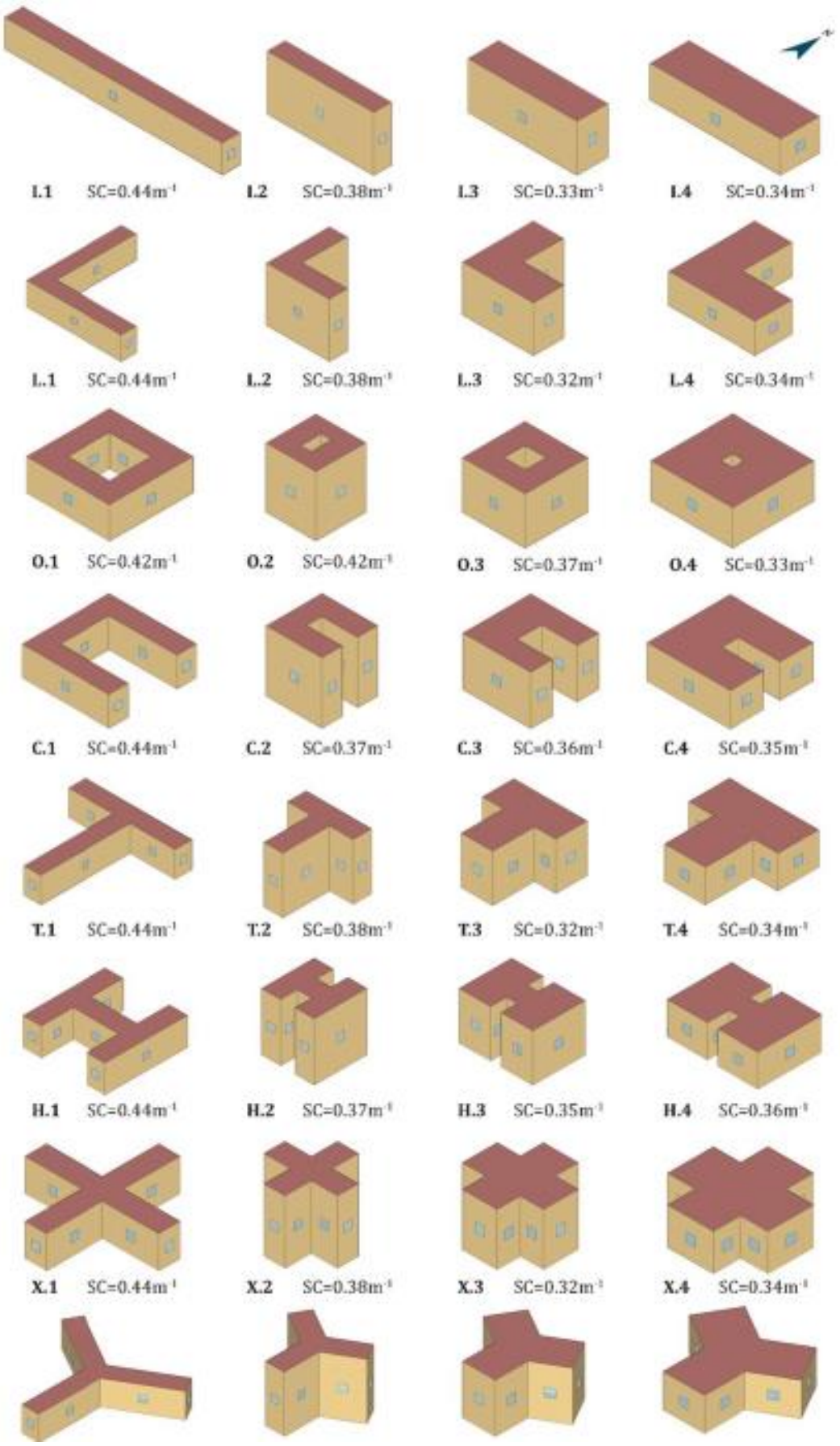


Figure 64 : Génération de forme simples (sources: ciardello et al., 2020)

A cette échelle, les critères de performance sont beaucoup plus axés sur l'énergétique. Certains intègrent même une analyse de cycle de vie (Haymaker et al., 2018). Pour faire des simulations sur les besoins et consommations énergétiques, il est nécessaire de faire des hypothèses sur la géométrie et la matérialité de l'enveloppe. La plupart du temps ces paramètres ne sont pas variables pour éviter l'explosion combinatoire. Lorsqu'on utilise des modèles physiques pour prédire des consommations, plus la forme est complexe (notamment courbe), plus les temps de calcul augmentent de façon exponentielle (Gerber & Lin, 2014).

La génération de plans

Il y a peu d'exemples dans la littérature d'ENAPE appliqués à la génération de plan. Nous en avons identifié seulement 5 dans notre corpus. L'exploration numérique de solutions appliquée à la génération de plans est un sujet déjà très complexe en soi sans avoir à y ajouter l'optimisation de la performance. La génération de plans est sans doute le problème le plus complexe en architecture computationnelle. Les scientifiques qui publient sur ce type de problèmes proviennent parfois de facultés bien éloignées des départements d'architecture comme l'informatique avancée ou le génie mécanique.

Les différentes méthodes pour faire de la génération de plans

Il existe de nombreuses méthodes pour faire de la génération de plans (un peu moins pour l'agencement d'espaces en 3D). Il peut s'agir d'agencement de mobilier, d'espaces ou de groupes d'espaces. Il y a souvent plusieurs étapes, et donc plusieurs méthodes nécessaires pour générer les plans (i) la représentation du problème, (ii) une résolution topologique (schématique) du problème, (iii) une résolution géométrique du problème.

Nous distinguons 6 familles de méthodes (Du, Turrin, et al., 2020):

(1) Le modèle physique de type simulation particulaire (Z. Guo & Li, 2017), voir Figure 64.

Les règles d'attraction et de répulsion permettent de définir des forces entre les espaces représentés par des cercles ou des rectangles reliés par des cordes (ou ressorts). Pendant la simulation, les espaces se repoussent ou s'attirent les uns les autres jusqu'à atteindre un équilibre. Le *plugin* de simulation physique *Kangaroo*® permet ce type de modélisation, et cette méthode peut être appliquée en 2D ou en 3D.

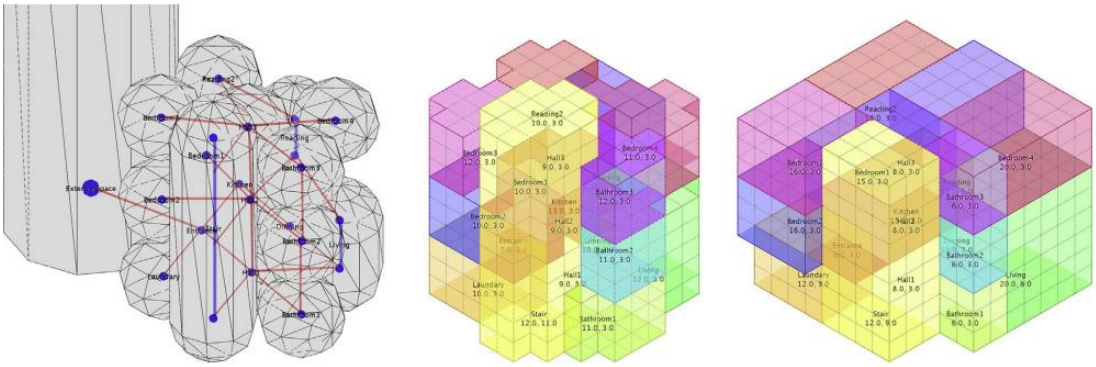


Figure 65 : Application avec un modèle physique pour un logement (source: Guo & Li, 2017)

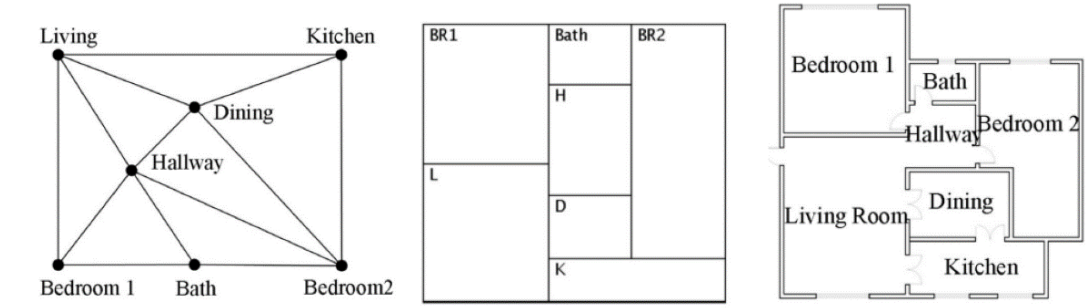
(2) La génération procédurale (Anderson et al., 2018), voir Figure 66.

Développée pour un problème spécifique, les espaces ou mobilier sont disposés les uns après les autres selon des règles strictes.

Figure 66 : Application d'une méthode procédurale appliquée à l'agencement de mobilier (source: Anderson et al., 2018)

(3) La théorie des graphes (X.-Y. Wang et al., 2018), voir Figure 67.

Le problème est représenté à l'aide d'un tracé de graphe planaire où chaque nœud représente une pièce et où chaque arête représente les règles d'adjacences entre les espaces. Ce graphe est dit planaire car aucune arête ne se croise. Les algorithmes de la théorie des graphes sont ensuite utilisés pour générer des plans à partir du graphe.



Graphe planaire — Plan schématique généré — Plan ajusté par le concepteur

Figure 67 : Application de la théorie des graphes pour un logement (sources: X.Y. Wang et al., 2018)

(4) La méthode d'assignement de cellules (Boon et al., 2015), voir Figure 68.

Une matrice permet de définir les règles d'attraction et de répulsion entre les différents espaces (méthode de représentation). Le bâtiment est discrétisé en cellules 3D (voxels), puis les différents espaces sont assignés aux voxels. Un mécanisme d'optimisation permet de générer des solutions qui respectent au mieux les règles définies dans la matrice. Cette méthode peut facilement être appliquée en 3D.

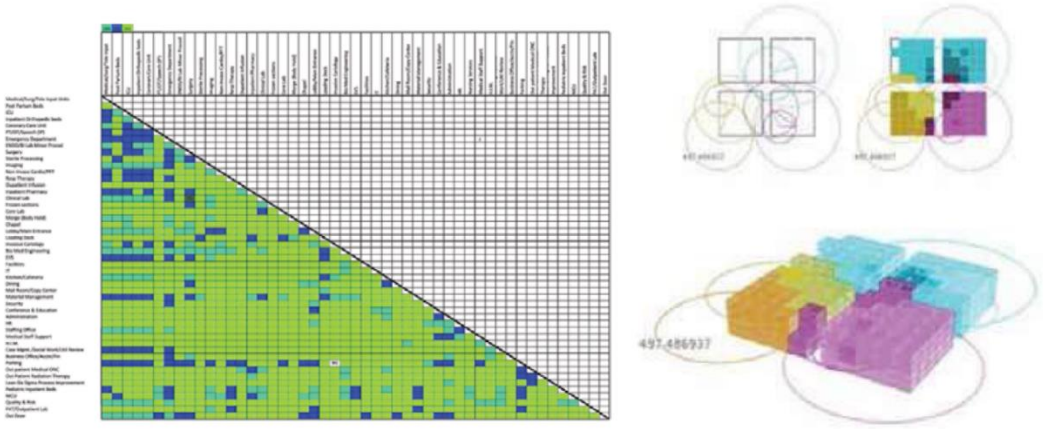


Figure 68 : Méthode d'assignement des cellules pour un hôpital (source Boon et al., 2015)

(5) La méthode de fractionnement de l'espace (Das et al., 2016), voir Figure 69.

Un plan d'étage est subdivisé de façon itérative à l'aide d'une structure en arborescence qui définit les relations entre les espaces. Les nœuds de l'arbre représentent des espaces. L'arbre contient aussi les paramètres de l'espace comme ses dimensions ou sa surface. A chaque itération, un espace est ajouté. Cette méthode permet une approche hiérarchisée des espaces.

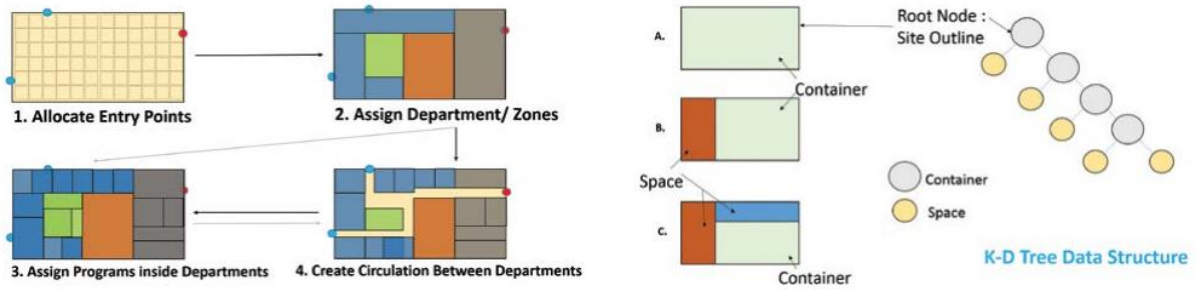


Figure 69: Application de la méthode de fractionnement (source: Das et al., 2016)

(6) Le *Machine Learning* (Chaillou, 2020)

Des centaines, voire milliers d'images de plans sont analysées par un algorithme de *Machine Learning* (type réseau de neurones) pour l'entrainer à générer des plans. Cela permet de reproduire le processus de conception de l'architecte sans avoir besoin de comprendre sa logique en profondeur. Il est souvent nécessaire de retravailler les images

de la base de données pour qu'elles soient lisibles pour l'algorithme, par exemple en coloriant les plans selon la fonction des espaces. Stanislas Chaillou (2020) propose une méthode générative en 3 étapes (i) un algorithme entrainé pour générer des empreintes de bâtiment, (ii) un algorithme entrainé pour générer un agencement des espaces, (iii) un dernier pour la disposition du mobilier.

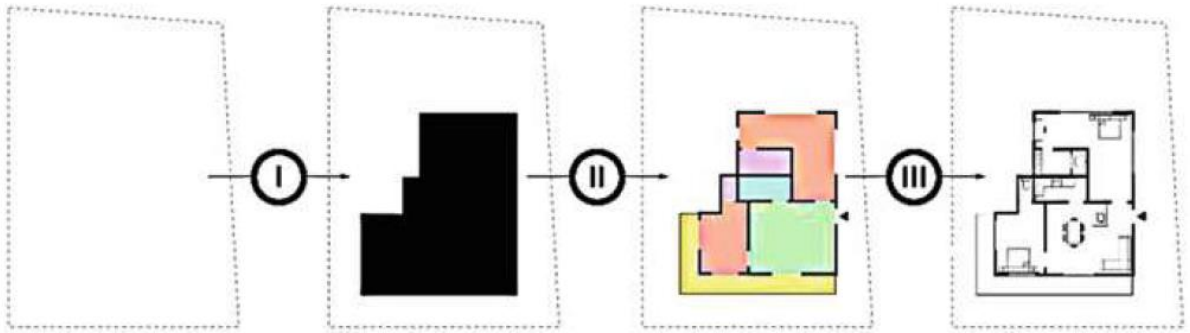


Figure 70 : Génération de plans de logements avec du *Machine Learning* (source: Chaillou, 2020)

Générations de plans et performance

Parmi nos 5 références, une seule utilise un modèle paramétrique (Su & Yan, 2015) pour optimiser la disposition des chambres hospitalières d'une unité de soins. Cette approche semble peu applicable dans la pratique. Une autre référence s'intéresse à la génération de plan hospitalier (Sleiman et al., 2017), cependant il ne s'agit pas véritablement d'exploration puisque seulement deux variantes sont analysées.

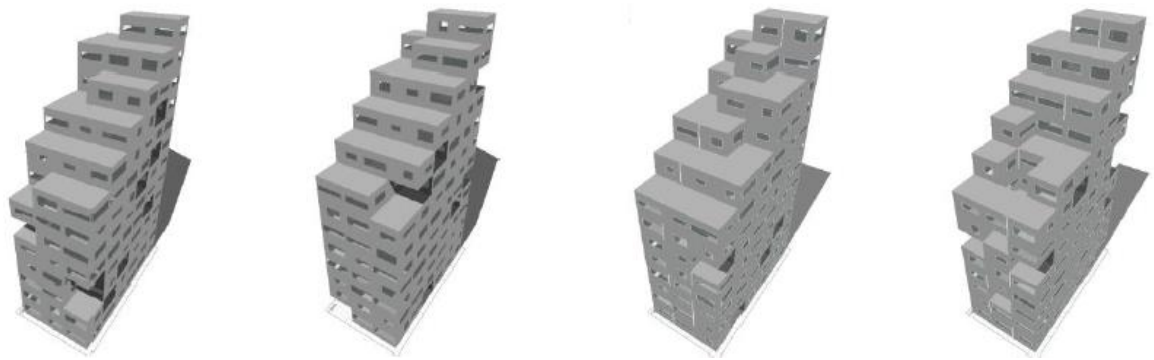


Figure 71 : Quatre solutions d'agencement d'appartements générées (source: H. Yi & Yi, 2014)

Deux autres références (Schwartz et al., 2017; H. Yi & Yi, 2014) interrogent l'agencement d'espace en 3D pour des logements. De leurs différents agencements découlent différentes formes urbaines. Il s'agit donc aussi d'une optimisation de la morphologie du bâtiment. La forme des espaces à agencer restent très simple, un parallélépipède ou une forme en L (voir Figure 71).

Finalement, une seule référence d'allocation spatiale avec subdivision de l'espace intègre l'optimisation de la performance (H. Yi, 2016).
La génération de plans performants est un sujet encore très peu traité dans la littérature scientifique et représente un vrai enjeu pour des recherches futures. Les méthodes décrites dans la littérature semblent difficilement applicables dans la pratique, d'autant que si l'architecte requiert une aide informatisée pour l'assister dans un exercice complexe d'agencement d'espaces, ce ne sera certainement pas pour agencer les 5 pièces d'un appartement ou d'une petite maison.

Au terme de cette seconde partie du premier chapitre, il apparaît que s'il existe beaucoup d'exemples d'applications de l'ENAPE dans la littérature scientifique, celles qui intègrent des préoccupations architecturales ne sont pas si nombreuses. Parmi elles, les expérimentations ayant lieu dans un contexte professionnel sont très rares. Contrairement aux applications orientées pour les ingénieurs, celles qui s'adressent aux architectes traitent plus souvent du confort que des consommations énergétiques. Les problèmes traités sont principalement multicritères, leurs résolutions s'appuient sur l'usage d'algorithmes d'optimisation souvent génétiques.
Une application sur deux est appliquée à l'enveloppe. Il s'agit principalement d'optimisation du système de protection solaire. La morphologie du bâtiment et un détail d'enveloppe peuvent être explorées à l'aide de simples modèles paramétriques tant qu'il n'est pas nécessaire d'intégrer des contraintes. L'échelle urbaine nécessite d'utiliser des méthodes génératives (LS) et des méthodes d'évaluation spécifiques comme les indicateurs issus de la syntaxe spatiale ou des modèles de substitution pour l'énergétique. L'allocation spatiale pour l'agencement des espaces intérieurs nécessite l'usage de modèles génératifs particulièrement complexes, pas toujours au point pour des problèmes comptant un grand nombre d'espaces comme dans le cas du programme hospitalier.
Aussi, nous retenons que très peu de ces exemples d'applications ne traitent de la question de l'intégration des exigences réglementaires autres que les exigences environnementales qui constituent des contraintes inviolables dans la pratique de l'architecture. Enfin, les problèmes intégrant des préoccupations architecturales présentent des modèles génératifs pouvant être bien plus complexes que les problèmes centrés sur des préoccupations uniquement thermiques, obligeant les architectes souhaitant pratiquer la conception générative à acquérir une solide culture informatique.

1.3. Bibliothèque d'outils d'évaluation pour la conception paramétrique et générative

L'objectif de cette dernière partie n'est pas de présenter tous les outils de simulation qui existent sur le marché mais les outils qui nous semblent les plus adaptés pour une transition d'une pratique de la conception paramétrique vers une pratique de la conception générative et performancielle en agence d'architecture. Pour pouvoir tester dans un contexte professionnel les méthodes présentées dans la partie précédente, il était nécessaire de constituer une bibliothèque d'outils d'analyse compatibles avec les outils de modélisation utilisés quotidiennement par les architectes praticiens, mais aussi de comprendre comment ils fonctionnent et comment ils peuvent être en pratique des outils d'aide à la décision. Ce chapitre fait la synthèse de la veille technologique réalisée tout au long du doctorat et du pré-doctorat sur les outils d'évaluation pour la conception paramétrique et générative.

1.3.1. Méthode d'élaboration de la bibliothèque

Nous nous sommes intéressés en priorité aux méthodes et outils facilement interopérables avec les modeleurs 3D utilisés quotidiennement par les architectes de notre terrain d'étude, l'agence Architecture Studio (AS). Deux outils sont utilisés pour la modélisation 3D : le logiciel de CAO *Rhinocéros*® et le logiciel CAO BIM multi-métiers du BTP *Revit*®. Chez AS, certaines équipes de projet travaillent exclusivement sur *Revit*® (dès les phases de concours), notamment pour la commande de grands équipements publics français (hôpitaux, prisons, écoles, palais de justice…), d'autres équipes travaillent la modélisation 3D exclusivement sur *Rhinocéros*® notamment les projets urbains, les immeubles de bureaux et les concours internationaux. Enfin, certaines équipes utilisent les deux logiciels. *Rhinocéros*® peut être utilisé en phases amonts (notamment pour le concours), et si le projet est gagné, la maquette 3D sera entièrement reprise sur *Revit*®. Parfois, certains éléments du projet, comme les enveloppes, sont travaillés sur *Rhinocéros*® (connu pour être plus performant pour la modélisation de géométries complexes) puis exportées dans *Revit*® à l'aide d'outils d'interopérabilité comme *RhinoInside*® ou les plugins *Hummingbird*®, ou *Grevit*®.

Pour notre bibliothèque, l'environnement *Rhinocéros/Grasshopper*® a été préféré à l'environnement *Revit*/*Dynamo*® pour plusieurs raisons : (1) beaucoup plus de plugins ont été développés pour *Grasshopper*®, (2) qui est donc beaucoup plus utilisé par la communauté scientifique, (3) la grande majorité des plugins développés pour cet environnement sont open-source, enfin (4) *Rhinocéros*® est plus stable, et *Grasshopper*® peut être utilisé dans *Revit*® via *RhinoInside*®.

La bibliothèque a été constituée au fur et à mesure de l'avancement de la thèse. Les indicateurs ont été sélectionnés en fonction des différents besoins rencontrés dans les projets de l'agence ArchitectureStudio donnant lieu à des expérimentations, et en fonction des exemples issus de la littérature scientifique. Ces besoins étaient souvent le reflet des demandes des maîtres d'ouvrages, ou des exigences décrites dans les programmes. L'objectif était de rassembler une grande diversité d'indicateurs. En effet, dans la pratique, les critères peuvent être nombreux comme le montre la Figure 72 sur les différentes sollicitations de l'enveloppe. Ainsi, nous avons cherché à évaluer dans l'ordre l'ensoleillement, la lumière, l'énergie, les vents, les précipitations, la vue, et enfin l'impact carbone. La plupart des outils ont été identifiés grâce : (1) à la revue de littérature présentée dans la partie 1.2 du chapitre 1, mais aussi (2) à la lecture d'articles scientifiques spécialisés sur un type d'évaluation (Carlucci et al., 2015 ; Hu et al.,

2023 ; Säwén et al., 2022), et enfin (3) sur la plateforme en ligne for4rhino.com (*Food4Rhino*, s. d.). La plupart des plugins cités dans ce chapitre ont été testés et comparés. Pour chaque critère, le plugin le plus adapté a été retenu et intégré à notre pratique. Les critères qui ont permis de sélectionner les outils étaient : les temps de calcul, l'adaptabilité aux phases amonts de projet, le coût de la licence, la visualisation, l'adaptabilité à la réglementation française. Le plugin le plus complet et que nous avons utilisé sur des bases régulières pour cette thèse est la suite *Ladybug*® (Roudsari et al., 2013) développée depuis 2012 par Mostapha Sadeghipour Roudsari, co-fondateur de *Ladybug Tools LLC*.

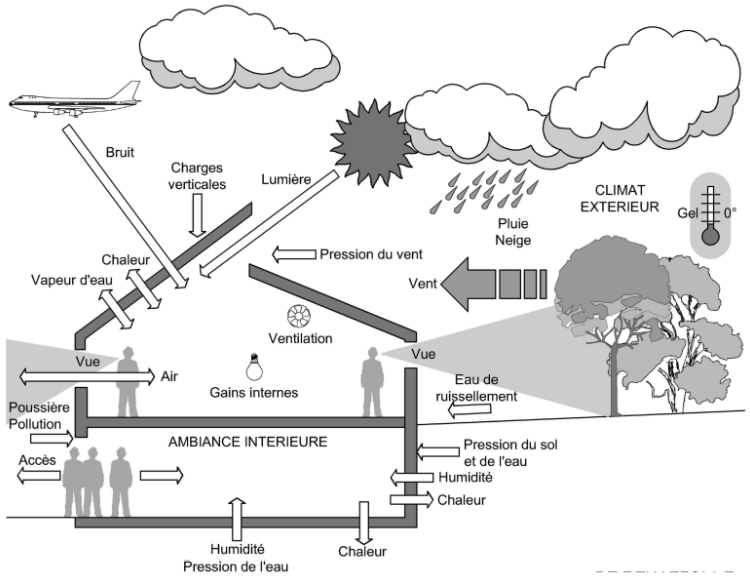


Figure 72: Les différentes fonctions de l'enveloppe extérieur (source: Hauglustaine et al, 2018)

Lorsque pour certaines évaluations simples, il n'existait pas (à notre connaissance) de plugin *Grasshopper*® dédié, nous avons développé nos propres méthodes. Ainsi, certains outils ont été développés en python notamment pour l'évaluation de la qualité de vue, et l'étude des précipitations. Nous avons listé dans le

Tableau 5 l'ensemble de ces outils identifiés ou développés. Pour chacun d'entre eux, nous avons indiqué le niveau de difficulté que représente leur mise en œuvre dans un processus d'optimisation selon quatre critères : la modélisation, les temps de calcul, l'interprétation des résultats, et la définition de la fonction d'évaluation.

Chaque critère intégré à la bibliothèque a nécessité un état des lieux des connaissances, des méthodes et des logiciels existants sur le sujet avant de pouvoir être expérimenté en pratique. La suite de ce chapitre fait la synthèse de ces connaissances minimales nécessaires

pour la compréhension et l'utilisation de ces outils par des architectes dans un processus d'optimisation.

Tableau 5: Bibliothèque d'outil d'évaluation pour l'optimisation de la performance environnementale

Critère	Indicateur	plugin	modélisation	temps de calcul	interprétation des résultats	fonction évaluation	Avantage/Limite pour l'optimisation
lumière	Eclairement lumineux	honeybee Radiance	**	*	**	**	Une date et heure précise
	Facteur de lumière du jour	honeybee Radiance	**	**	**	**	Ne prend pas en compte la météo local et l'orientation
	Autonomie en lumière du jour	honeybee Radiance	**	**	*	**	
	Eclairement utile	honeybee Radiance	**	**	**	**	Peu prendre en compte les risques d'éblouissement
éblouissement	Daylight Glare Index	honeybee Radiance	**	*	*	*	Un seul point de vue
	Daylight Glare Probability	honeybee Radiance	**	*	*	*	Un seul point de vue
vue	Visibilité	ladybug	*	*	**	**	
	Vue sur un élément remarquable	_	*	*	*	**	
	Vis-à-vis	_	*	*	*	**	
confort thermique	Predicted Mean Vote	honeybee Energy+	***	***	*	**	Uniquement pour le bureau
	Predicted Percentage of Dissatisfied	honeybee Energy+	***	***	*	*	Uniquement pour le bureau
	Universal Thermal Index	honeybee Energy+	***	***	*	**	Echelle urbaine, temps de calcul trop long
ensoleillement	Radiation solaire simplifiée	ladybug	*	*	**	**	Ne prend pas en compte la radiation réfléchie
	Radiation solaire	honeybee Radiance	**	**	**	**	
	Heures d'ensoleillement	ladybug	*	*	*	**	
aéraulique	Confort au vent	Eddy 3D OpenFOAM	**	***	**	***	Coquille qui empêche de faire des boucles
	Pression des vents en façade	Eddy 3D OpenFOAM	**	***	**	***	Coquille qui empêche de faire des boucles
précipitations	Trajectoire des gouttes de pluie	_	**	**	*	**	
énergie	Prédiction des besoins énergétiques	honeybee Energy+	***	***	*	*	Pas adapté à la réglementation française
	Prédiction des consommations énergétiques	honeybee Energy+	***	***	*	*	Pas adapté à la réglementation française
ACV	Impact carbon des matériaux de construction	One Click LCA	**	*	*	*	licence payante
	Impact carbone des consommations énergétiques	Bombyx LCA	**	***	*	*	Pas adapté à la réglementation française

Modélisation
* à partir de la maquette des architectes
** nécessite d'adapter l a maquette des architectes
*** nécessite la création d'un modèle 3D spécifique

Temps de calcul
* se compte en s secondes
** se compte en minutes
*** se compte en heures

Interprétation des résultats
* limpide, même pour un novice
** nécessite d'être initié
*** nécessite une formation

Définir la fonction d'évaluation
* la fonction est le résultat de l'évaluation
** les résultats nécessitent d'être normalisés
*** nécessite de formuler et de tester une fonction spécifique

1.3.2. Le Confort visuel

Dans cette partie, nous apportons des précisions sur les indicateurs et outils de la bibliothèque qui concernent la lumière naturelle, l'éblouissement et la qualité de la vue. Ces trois critères traitent du confort visuel.

L'analyse de l'éclairage naturel

Définition et grandeurs photométriques

Selon la norme européenne EN_12665, le confort visuel est une *« condition subjective de bien-être trouvant son origine dans l'environnement lumineux »*, la luminosité est l'*« attribut d'une perception visuelle selon lequel une surface paraît émettre ou réfléchir plus ou moins de lumière »*, et l'éblouissement correspond aux *« conditions de vision dans lesquelles on éprouve de l'inconfort ou une réduction de l'aptitude à distinguer des détails ou des objets, par suite d'une répartition défavorable des luminances ou d'un contraste excessif »*. Que ce soit pour le confort, la santé, la productivité (Edwards & Torcellini, 2002; Rashid & Zimring, 2008) ou pour réduire la consommation d'éclairage artificiel, il est essentiel de pouvoir contrôler la lumière naturelle dans un bâtiment. Il s'agit d'un enjeu d'autant plus important que la densité urbaine peut avoir un impact considérable sur son potentiel d'utilisation.

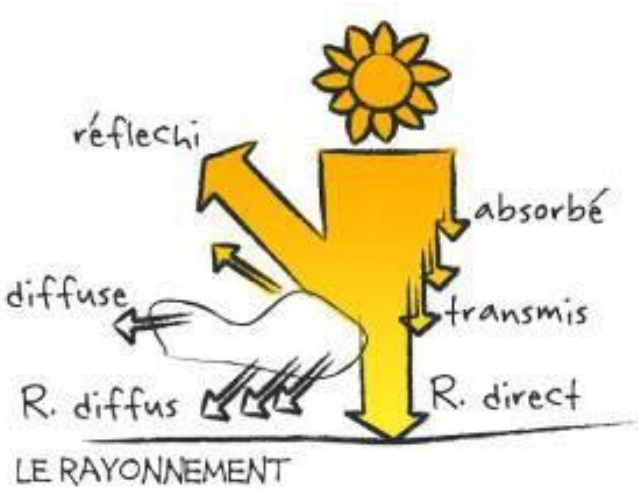


Figure 73: Les différents types de rayonnement solaires (source: https://www.guide-clea.fr/clea_projet/climat/)

La lumière naturelle correspond à la partie visible du rayonnement solaire. D'après le guide ICEB-Arene, la composition du rayonnement varie aux cours des saisons mais en moyenne entre 48 et 51 % des rayons du soleil seraient visibles (AREC, s. d.). Ce rayonnement visible est en partie filtré et diffusé par la voûte céleste, une partie des rayons est réfléchie ou absorbée, une autre est transmis ou diffusée. Ainsi, on distingue deux types de rayonnement, le rayonnement direct et le rayonnement diffus. L'éclairement dépend donc du type de ciel (ensoleillé avec soleil visible ou non ou par intermittence, ciel nuageux, uniforme…), mais aussi de la position du soleil.

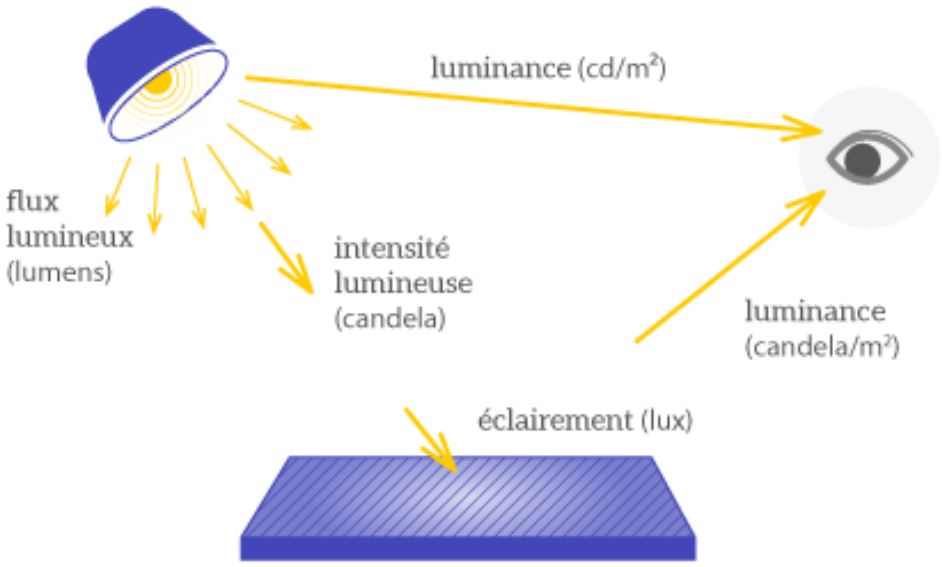


Figure 74 : Les grandeurs photométriques pour mesurer l'éclairage (source: https://leclairage.fr/th-photometrie/)

Quatre grandeurs photométriques permettent de mesurer la lumière :

(1) Le flux lumineux est la puissance lumineuse émise par une source dans toutes les directions. Il est pondéré par la sensibilité de l'œil humain. Il se mesure en lumens (lm). Cette grandeur permet de définir toutes les autres.

(2) L'intensité lumineuse exprime le flux lumineux d'une source ponctuelle dans une direction donnée. Elle se mesure en candela (cd).

(3) L'éclairement lumineux correspond au flux lumineux reçu par une unité de surface. Elle se mesure en lux (lx) ce qui équivaut à des lumens par mètre carré.

(4) La luminance porte sur la notion d'éblouissement. D'après le dictionnaire Le Robert, elle correspond au *« quotient de l'intensité lumineuse d'une surface par l'aire apparente de cette surface pour un observateur lointain »*.

Le confort visuel dépend de la physiologie de l'œil humain, des quantités physiques décrivant la quantité de lumière et sa distribution dans l'espace et de l'émission spectrale de la source lumineuse. Les facteurs pris en compte dans son évaluation sont la quantité de lumière, l'angle d'incidence de la lumière, l'intensité de la lumière et l'uniformité de la lumière (Carlucci et al., 2015)

Les indicateurs de l'évaluation de la lumière

D'après une étude réalisée en 2014 (Galasiu & Reinhart, 2008) auprès de concepteurs (architectes, architectes d'intérieurs et concepteurs d'éclairage), ingénieurs et chercheurs, plusieurs méthodes sont utilisées pour traiter la question de l'éclairage naturel dans les projets. Pour les concepteurs, l'expérience est la méthode la plus employée, suivie par la simulation numérique, les règles d'or, les guides de conception, les informations du fabricant, et les maquettes physiques. La simulation numérique permet d'évaluer à partir d'une maquette numérique 3D des dizaines d'indicateurs différents.

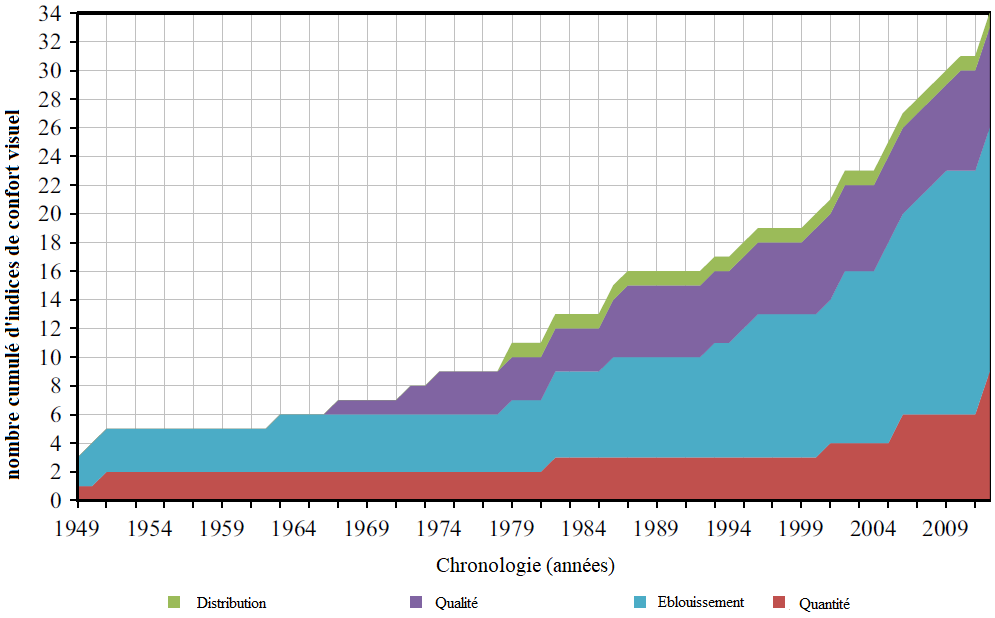


Figure 75 : Evolution du nombre d'indices de confort visuel (source: (Carlucci et al., 2015))

D'après une revue sur les indices d'évaluation du confort visuel réalisée en 2015 (Carlucci et al., 2015), il existe plusieurs types d'indices : les indices permettant d'évaluer la quantité de lumière, les indices permettant d'évaluer la distribution de la lumière, les indices pour l'évaluation de l'éblouissement et les indices pour évaluer la qualité de la lumière. Ils diffèrent les uns des autres selon plusieurs dimensions, telles que le champ d'application de l'évaluation, les quantités physiques impliquées, la période de calcul, la source lumineuse, le critère d'acceptabilité et la présence d'un seuil. En optimisation, les indicateurs les plus utilisés sont ceux de l'évaluation de la qualité de la lumière et ceux de l'éblouissement

Nous présentons ici la liste des différents indicateurs que nous avons rencontrés dans la pratique au cours de nos expérimentations chez Architecture Studio.

L'éclairement lumineux peut être mesuré à une date et heure donné en un point donné d'une surface donnée. Il dépend de la localisation, de l'orientation, des surfaces environnantes et de leur coefficient de réflexion lumineuse, et du type de ciel. Il est souvent mesuré à 70 cm du sol, ce qui correspond à la hauteur d'une surface d'un bureau. D'après la norme européenne EN 12464-1 sur la lumière et l'éclairage des lieux de travail, on considère que pour être confortable, un espace de circulation doit être éclairé à 100 lux et que des espaces de travail doivent être éclairés entre 300 et 500 lux. Lorsqu'on souhaite évaluer l'éclairage naturel dans le cadre d'une optimisation, cet indicateur peut être choisi car il est bien plus rapide à évaluer qu'une simulation annuelle. Notamment, il peut être pertinent d'étudier l'éclairement lumineux aux solstices pour connaitre les valeurs d'éclairage dans les pires conditions.

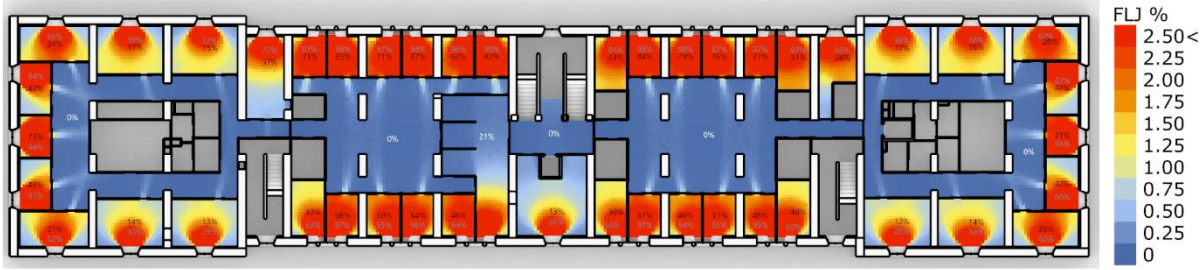


Figure 76 : Evaluation du FLJ dans des bureaux pour une réhabilitation

Le facteur de lumière du jour (FLJ) est *« le rapport de l'éclairement intérieur reçu en un point du plan de référence (souvent plan de travail ou sol), à l'éclairement extérieur simultané sur une surface horizontale en site parfaitement dégagé »* (De Luminae, s. d.). La simulation se fait avec un ciel couvert, le rayonnement direct est donc exclu du calcul. Cet indicateur a l'avantage de pouvoir être calculé rapidement, cependant il a des limites. La localisation, la météo locale et l'orientation ne sont pas prises en compte dans les prédictions. Il ne permet pas non plus de prendre en compte le risque d'éblouissement (Carlucci et al., 2015). Le résultat est exprimé en pourcentage. Un FLJ inférieur à 1 % est considéré comme très faible, il est faible entre 1 et 2 %, moyen entre 2 et 5 %, élevé entre 5 et 10 % et très élevé au-dessus de 10 %. Des valeurs d'au moins 5 % sont conseillés dans les bureaux.

L'autonomie en lumière du jour (ALJ) dans un bâtiment est définie *« comme le pourcentage de périodes d'occupation par an pendant lesquelles un niveau d'éclairement minimal peut être maintenu par la lumière du jour. Il s'agit d'une mesure de performance qui prend en compte toutes les conditions du ciel tout au long de l'année lorsque le bureau est occupé »* (C. F. Reinhart et al., 2003). Cette simulation annuelle prend en compte la localisation, la météo,

l'orientation, et la réflexion de la lumière sur les surfaces environnantes. Les temps de calcul peuvent être longs. Le niveau d'éclairage minimal (100, 300 lux…), ainsi que la période d'occupation dépendent des fonctions des locaux évalués.

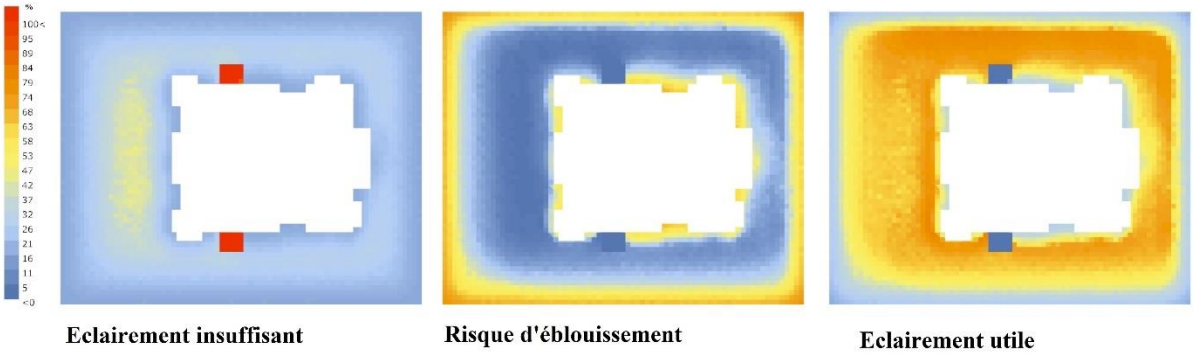


Figure 77 : Evaluation de l'éclairement utile pour un concours de bureaux

L'éclairement utile est défini, d'après (Carlucci et al., 2015), comme la fraction du temps dans l'année où l'éclairement naturel horizontal intérieur à un point donné se situe dans une plage donnée (souvent entre 100 et 2000 lux). Le calcul est le même que pour l'ALJ, c'est la façon de présenter les résultats qui diffère. Ici, deux seuils sont utilisés, un premier en dessous duquel l'éclairement est considéré comme insuffisant, et un second au-dessus duquel l'éclairement est considéré comme trop important, donc à risque d'éblouissement, et entre les deux, l'éclairement est considéré comme « *utile* ». Il existe aussi l'indice de fréquence du confort visuel, son concept est similaire à celui de l'éclairement utile.

Les indicateurs de l'évaluation de l'éblouissement

Il existe différents indicateurs pour évaluer l'éblouissement (Carlucci et al., 2015), nous nous sommes intéressés à ceux identifiés dans notre corpus présenté au chapitre précédent : il s'agit du Daylight Glare Index (DGI) et du Daylight Glare Probability (DGP)

Le DGI est le principal indicateur de l'éblouissement (Bellia et al., 2008) et est couramment utilisé. La formule pour le calculer est complexe et prend en compte la luminance de la source, des surfaces environnantes et de la fenêtre, l'angle solide de la fenêtre et de la source. Le DGI s'évalue pour un point de vue donné. L'éblouissement est perceptible avec un index de 16 ou 18, acceptable à 20, borderline à 22, inconfortable à 24 ou 26 et intolérable à 28, mais il est connu pour surestimer l'éblouissement dans la plupart des cas.

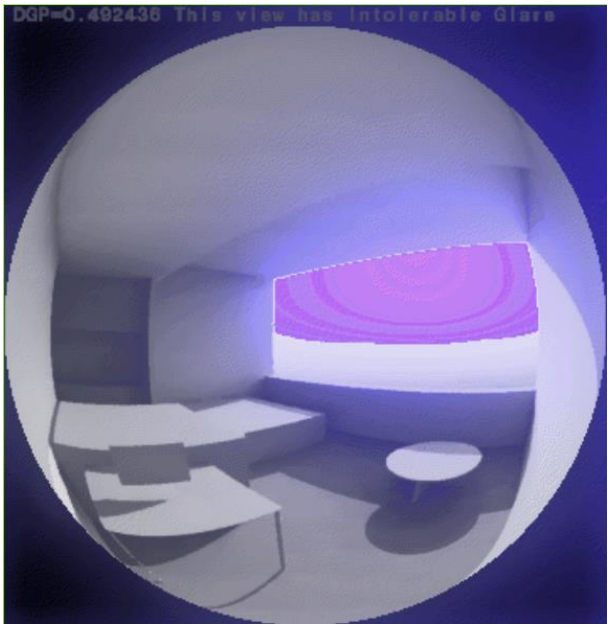


Figure 78 : Evaluation du DGP avec *Ladybug*®

Le DGP est un indicateur créé en 2006 par Wienold et Christofferse qui *« utilise la probabilité qu'une personne soit dérangée par l'éblouissement »* comme mesure de l'éblouissement (Wienold & Christoffersen, 2006). Il s'agit d'une formule empirique dont les principales variables sont l'éclairement vertical au niveau de l'œil, la luminance de la source d'éblouissement, de son angle solide et son indice de position. Il se mesure en pourcentage pour un point de vue donné. Avec un DGP en dessous de 35 %, on considère que l'éblouissement est imperceptible, entre 35 et 40 % il est perceptible, entre 40 et 45 % il est dérangeant, et au-dessus de 45 %, il n'est plus tolérable.

Les logiciels de simulation de la lumière et plugins d'interopérabilité

Pour pouvoir évaluer la lumière naturelle ou l'éblouissement, il faut avant tout un logiciel de simulation de l'éclairage. Le choix du moteur de simulation peut avoir un très fort impact sur les résultats de simulation. Il existe plusieurs logiciels qui ont un fonctionnement similaire en 3 étapes explicitées en Figure 79 (C. Reinhart, 2019) : (1) il faut pouvoir y définir les données d'entrée : la scène (géométrie, contexte, matériaux), le domaine d'intérêt à évaluer, les différents usages et le type de ciel ; (2) un moteur de simulation fait les calculs ; et (3) les résultats sont traités pour être traduits sous forme d'indicateur.

Le plus connu de ces logiciels, *Radiance*® utilise la méthode du lancer de rayons. A notre connaissance, il est le seul qui soit entièrement *open source*. Il existe 14 logiciels propriétaires, selon Wikipédia (« List of Lighting Design Software », 2022) dont *DIALux*®, *LightStanza*®,

ou encore *Relux*®. Certains ont des versions déjà intégrées aux outils CAO comme *ReluxCAD*® pour *Revit*® qui est plutôt destiné aux électriciens.

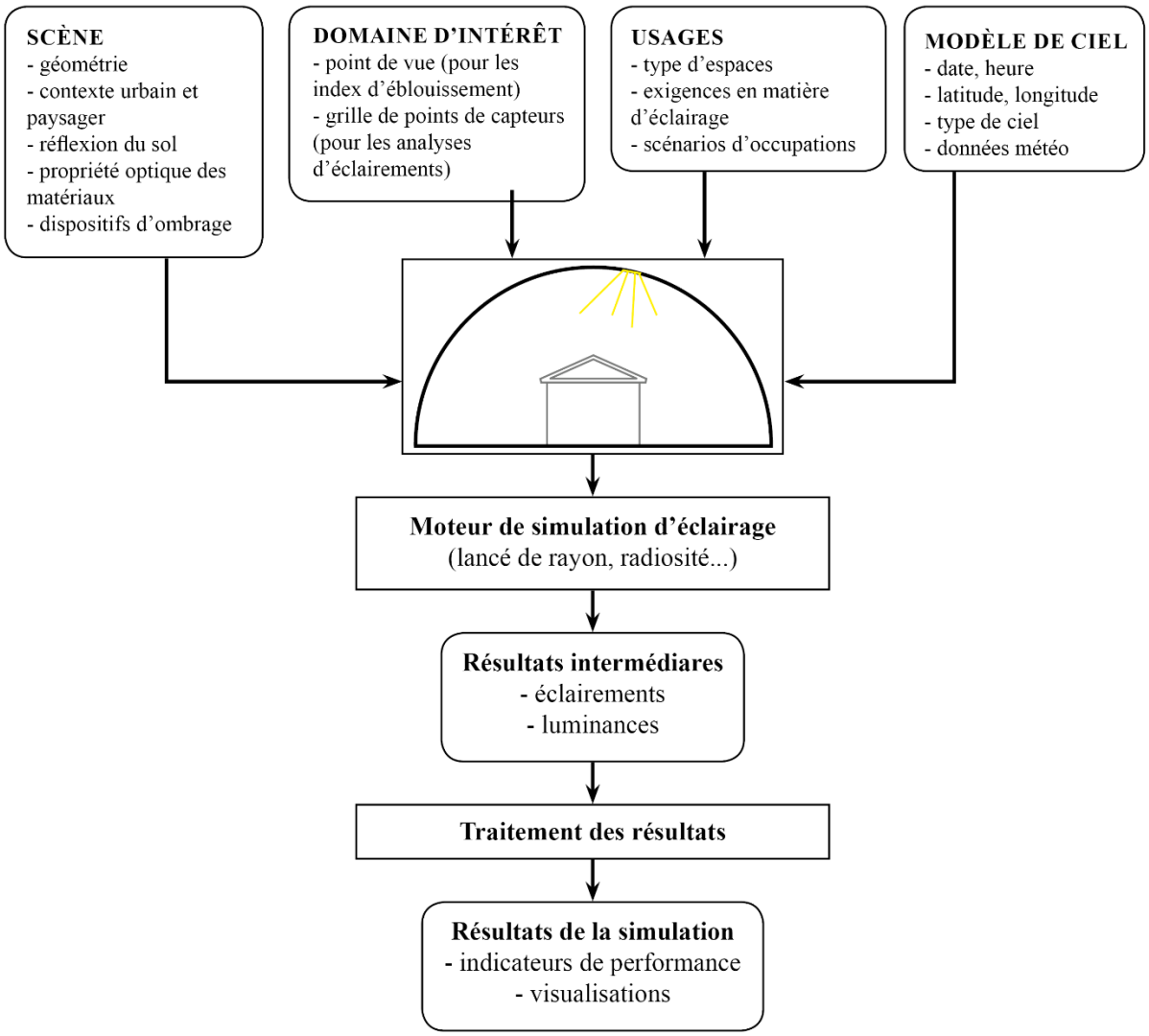


Figure 79 : Fonctionnement d'un programme de simulation d'éclairage, image adaptée de (C. Reinhart, 2019)

Il existe des logiciels qui utilisent le moteur de *Radiance*® et proposent des interfaces graphiques et fonctionnalités qui rendent son usage plus convivial et ergonomique (*DesignBuilder*®, *Integreted Environnmental Solutions*®,…). Cependant, ces outils restent plutôt destinés aux ingénieurs.

Des plugins ont été développés pour pouvoir faire de la simulation de l'éclairage avec *Radiance*® depuis certains modeleurs 3D comme le plugin *OpenStudio*® pour *Sketchup*®, ou *ClimateStudio*® pour *Rhinocéros*® (anciennement Diva-for-Rhino). Pour faire de l'optimisation nous recommandons le plugin *Ladybug*© disponible pour *Grasshopper*® et *Dynamo*®. Il s'agit d'une suite d'outils destinée à la prise en compte des problématiques

environnementales. L'outils *Ladybug*® permet de faire de la datavisualisation et *Honeybee*® de lancer des calculs *Radiance*® depuis une plateforme de programmation visuelle (voir Figure 80). Les résultats peuvent ensuite être facilement liés à des solveurs d'optimisation. Tous les indicateurs présentés précédemment peuvent être calculés avec la version *Grasshopper*® de *Ladybug*®.

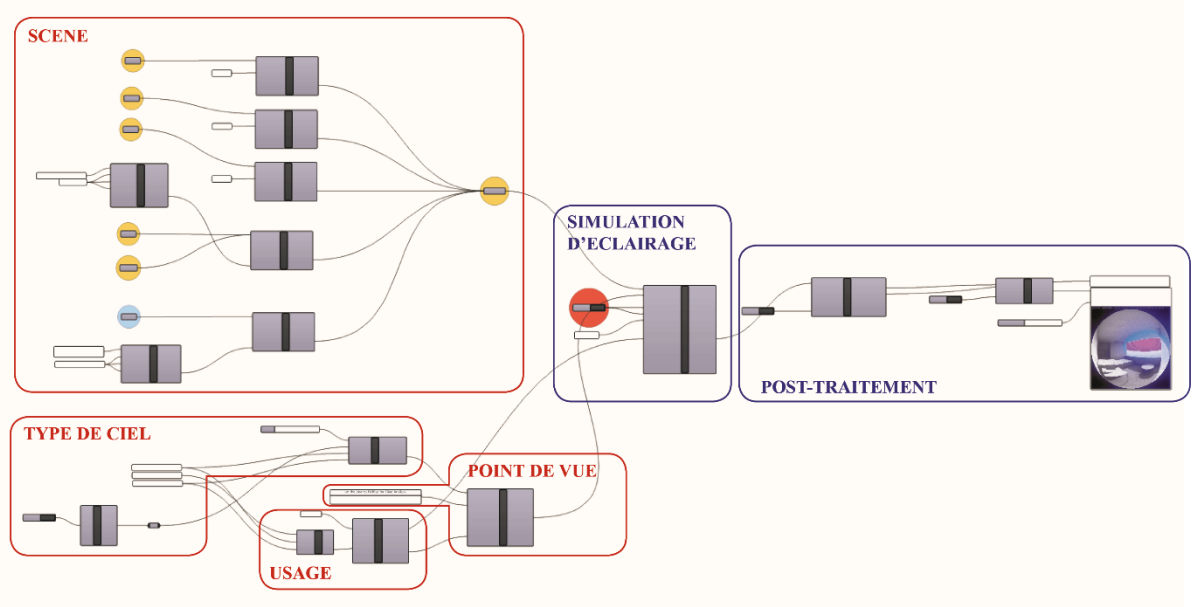


Figure 80 : Simulation d'éclairage via Ladybug sur Grasshopper

Le paramétrage de la simulation

La modélisation de la scène n'est pas particulièrement complexe. Il suffit d'importer dans *Grasshopper*® des surfaces modélisées dans *Rhinocéros*®. Elles doivent avoir été préalablement classées selon leur matériau de revêtement. En phase amont de projet, nous distinguons notamment, les planchers, plafonds, murs intérieurs, revêtements de façade, vitrage intérieur, double vitrage, végétation, sol extérieur, et les bâtiments du contexte. L'étape la plus délicate consiste à déterminer le coefficient de réflexion lumineuse pour les parois opaques et le coefficient de transmission lumineuse pour les parois translucides. De nombreux guides de conception ou sites internet donnent des références en fonction des matériaux. Pour une surface en double vitrage clair le guide Bio-tech sur l'éclairage naturel de l'ARENE préconise par exemple une transmission lumineuse de 0.81. La norme EN 12464-1 donne des préconisations pour les surfaces réfléchissantes intérieures : 0.7 à 0.9 pour les plafonds, 0.5 à 0.8 pour les murs, 0.2 à 0.4, pour les sols.

Le type de ciel dépend de l'indicateur choisi. La commission International de l'Eclairage (CIE) propose 15 modélisations de types de ciels différentes pour la simulation. Pour un calcul de

FLJ, on utilise un ciel uniforme standardisé (CIE). Pour les autres indicateurs, il est possible de générer un type de ciel à partir d'un fichier météo. Les usages concernent surtout les scénarios d'occupation notamment pour des indicateurs comme l'ALJ. Le domaine d'intérêt lorsqu'il s'agit d'une surface peut être discrétisé en grille de points. Les cellules de la grille peuvent être plus ou moins grandes. Plus elles sont petites, plus la simulation sera précise et plus les temps de calcul seront longs. Selon les exigences du référentiel HQE, le FLJ doit par exemple être calculé tous les 50 cm.

Pour faire de l'optimisation du confort visuel, l'étape la plus complexe est celle de la définition de la fonction d'évaluation. Pour les indicateurs de l'analyse de l'éclairement, il y a autant de valeurs en données de sortie que de points évalués. Pour pouvoir faire de l'optimisation, il est nécessaire de transformer ces valeurs en un unique nombre réel. Ainsi, il y a plusieurs méthodes entre lesquelles il faut choisir en fonction de l'objectif à atteindre. Il est possible de faire la moyenne de ces valeurs, ou bien de calculer la surface de plancher qui respecte un seuil limite, ou s'il a plusieurs locaux, de calculer le nombre de locaux qui atteignent une surface de plancher qui respecte un seuil limite. Pour les indicateurs de l'analyse de l'éblouissement, la question est moins complexe puisque les indicateurs calculent un score par point de vue.

L'analyse de la vue

Accès à la vue et qualité de la vue

En conception architecturale et urbaine, la vue est un critère essentiel. Elle n'a pas d'impact sur les consommations énergétiques mais elle entre souvent en conflit avec des critères qui peuvent en avoir beaucoup, notamment les apports solaires. De plus, la qualité de la vue a un impact sur la santé (Raanaas et al., 2012), le bien-être et la productivité des usagers (Sundstrom & Sundstrom, 1986), mais aussi sur le prix de vente des biens immobiliers.

Lorsqu'on parle d'analyse de la vue, il peut s'agir de deux approches distinctes :

(1) l'accès à la vue, ou l'analyse de la visibilité qui cherche à mesurer au combien une vue peut être obstruée par des éléments de son environnement. Elle peut se mesurer à l'aide d'outils comme les lancers de rayons et les isovists (Benedikt & Burnham, 1985). Il s'agit d'une évaluation quantitative de la vue, elle est donc facilement implémentable numériquement.

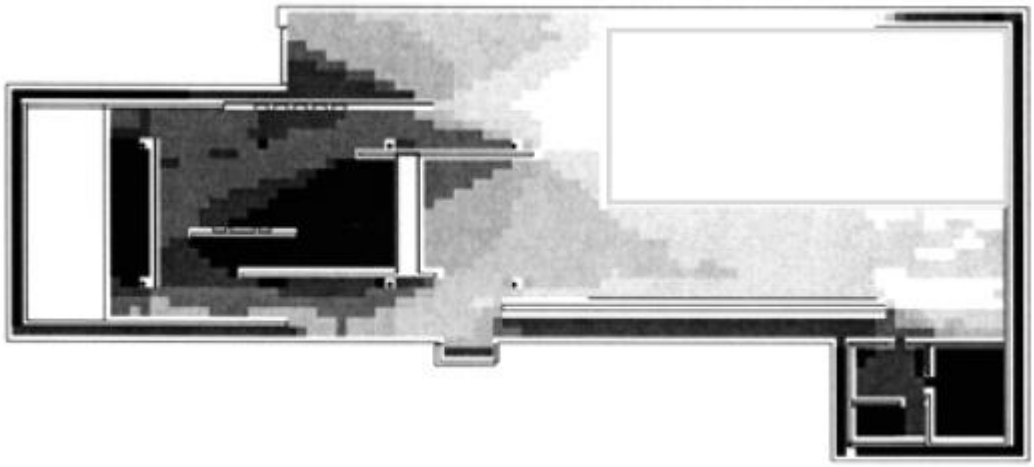


Figure 81 : Analyse de la visibilité dans le pavillon de Mies van der Rohe à Barcelone (source: Turner et al., 2001)

(2) la qualité de la vue qui dépend de la nature de ce qui est vu et des goûts de chacun. Ce critère particulièrement subjectif est difficile à évaluer. Il existe des méthodes qui s'appuient sur l'usage de questionnaires à choix multiples (Hellinga & Hordijk, 2014) permettant d'attribuer un score à la vue. (W. Li & Samuelson, 2020) proposent une méthode numérisable qui permettrait d'évaluer la vue à partir d'images issues de *Google Earth*® et des préférences de l'utilisateur (voir Figure 82).

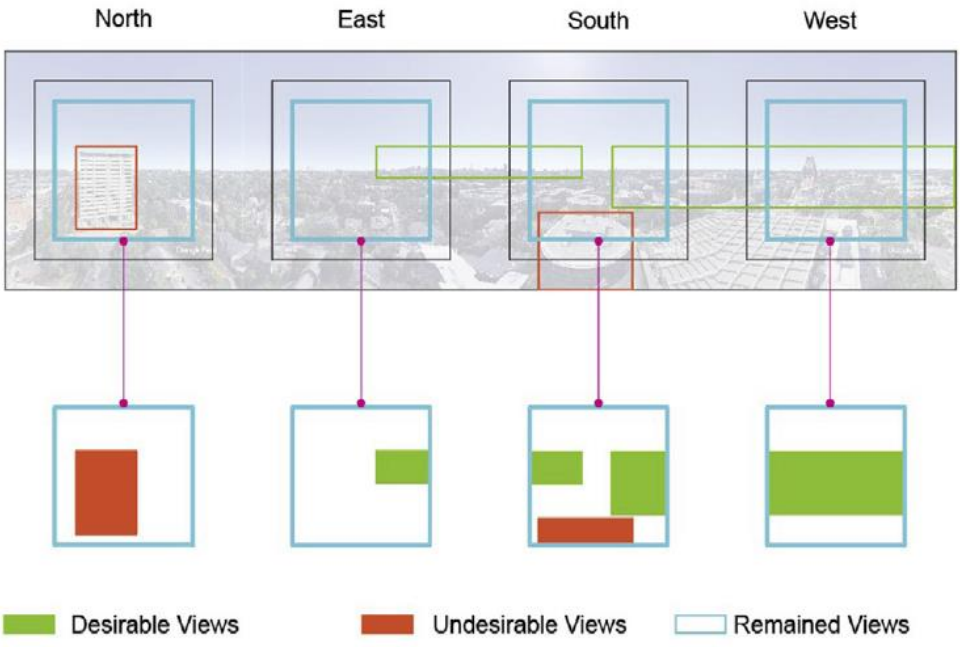


Figure 82 : Zone de préférences des vues définies par l'utilisateur (source: W.Li & Samuelson, 2020)

Outils d'analyse de la vue pour l'optimisation

Les plugins *Ladybug®*, *DesCodingSpaces®* pour *Grasshopper®* contiennent des composants qui permettent d'évaluer la visibilité avec des isovists 2D ou 3D. La modélisation est particulièrement simple puisque seulement deux données d'entrée sont nécessaires : la géométrie à évaluer et la géométrie impactant la visibilité. Les temps de calcul sont rapides comparés à une analyse de l'éclairage.

Grasshopper contient une fonction de lancer de rayons. Elle permet de créer facilement des méthodes d'évaluation de la vue adaptée à différents contextes de projets comme le font par exemple (Zargar & Alaghmandan, 2019) avec leur outil CORAL pour la conception de stade.

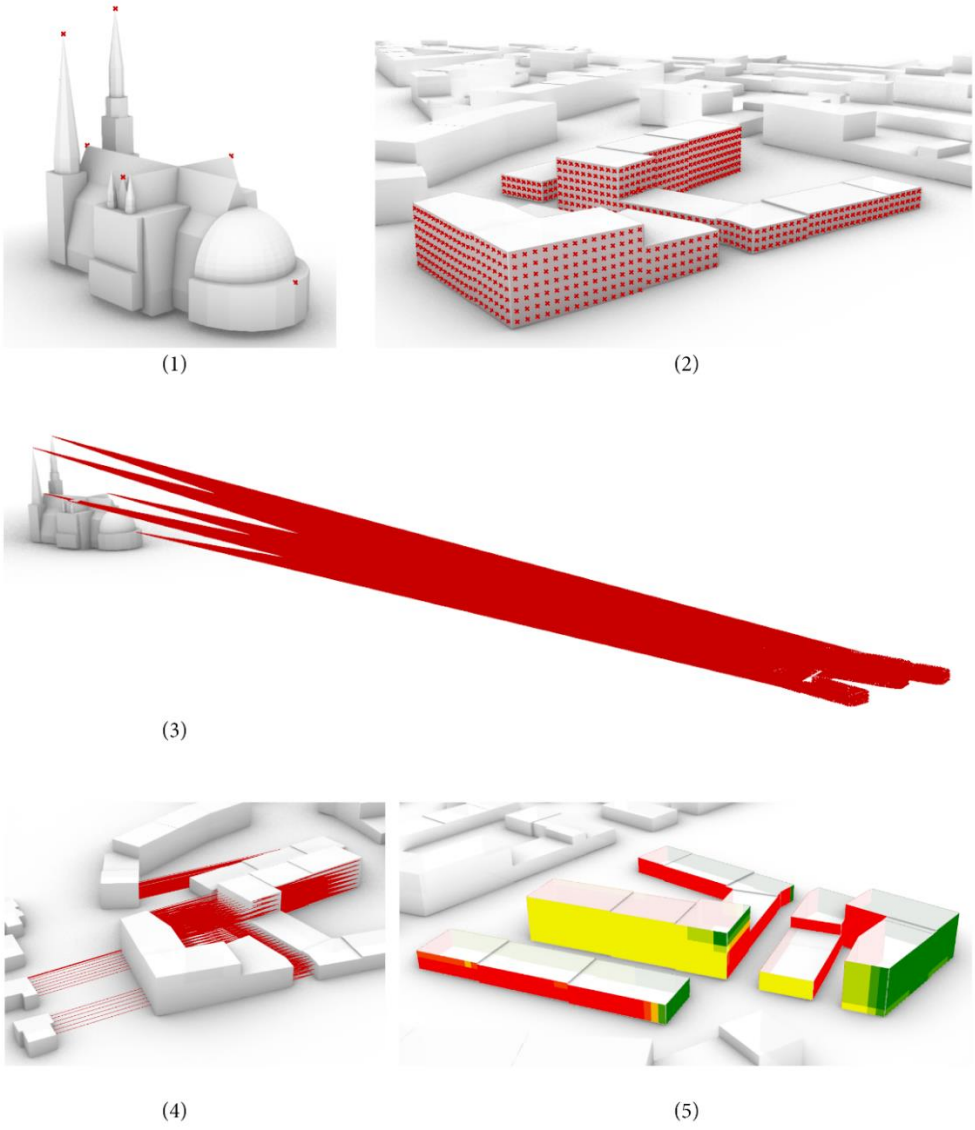


Figure 83 : Les étapes pour l'analyse de l'accès à la vue sur un élément remarquable

Ainsi, à l'occasion d'une application pour la ville de Chartres, nous avons développé un script *Grasshopper*® pour l'analyse de l'accès à la vue sur un élément remarquable (ici, la cathédrale). La méthode fonctionne en 5 étapes présentées en Figure 83: (1) définir des points sur la géométrie de l'élément remarquable, (2) générer une grille de points sur les surfaces à évaluer, (3) effectuer un lancer de rayons depuis les points de la grille vers les points de la géométrie remarquables, (4) comptabiliser les rayons qui ont été interceptés par la géométrie du contexte, (5) visualiser les résultats sur la géométrie originelle.

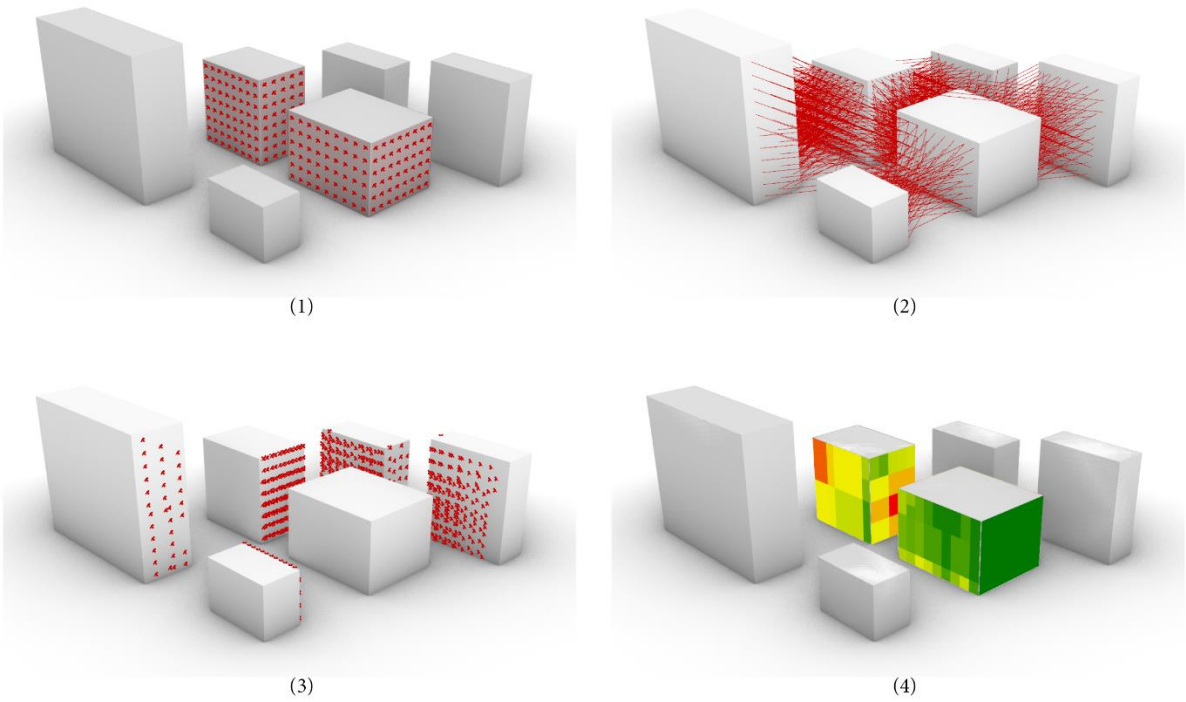


Figure 84 : Méthode d'évaluation des vis-à-vis

Un second outil à base de lancer de rayons a été développé sur *Grasshopper*® pour évaluer les vis-à-vis. Cet indicateur est destiné à l'optimisation de la morphologie d'un bâtiment ou d'un ou plusieurs îlots. La méthode fonctionne en 4 étapes (voir Figure 84) : (1) générer une grille de points sur les surfaces à évaluer, (2) lancer quelques rayons dans des directions différentes depuis les points de la grille, (3) comptabiliser les rayons qui ont été interceptés par la géométrie du contexte, (4) sommer les distances pour chaque point de la grille et visualiser les résultats sur la géométrie originelle.

1.3.3. Le confort thermique

Avant de s'intéresser aux consommations énergétiques du bâtiment, nous nous sommes intéressés à une notion qui lui est lié, le confort thermique. La réglementation environnementale 2020 intègre désormais un indicateur de confort thermique DH (degrès.heures) qui évalue le niveau d'inconfort perçu par les occupants en été. *« Un bâtiment, incluant sa structure, son éclairage, son système de production énergétique, etc., se doit d'offrir les conditions intérieures les plus confortables pour l'être humain. La notion de confort thermique est dès lors étroitement liée à la performance énergétique dans le bâtiment »* (Lavoye & Thellier, 2008). Le confort thermique dépend de nombreux paramètres : le métabolisme de l'homme, son habillement (sa résistance thermique), la température ambiante, la température moyenne des parois, l'humidité relative de l'air et la vitesse de l'air (« Confort thermique », 2007). Un diagramme psychométrique permet de représenter les caractéristiques de l'air humide. En Figure 85, Roland Fauconnier utilise ce type de diagramme pour définir des plages de confort hygrothermique pour les bâtiments tertiaires (Fauconnier, 1992). La zone 1 est à éviter pour la sécheresse, les zones 2 et 3 sont à éviter pour le développement des champignons et des bactéries, la zone 3 est à éviter pour les acariens et la zone 4 correspond au polygone de confort hygrothermique.

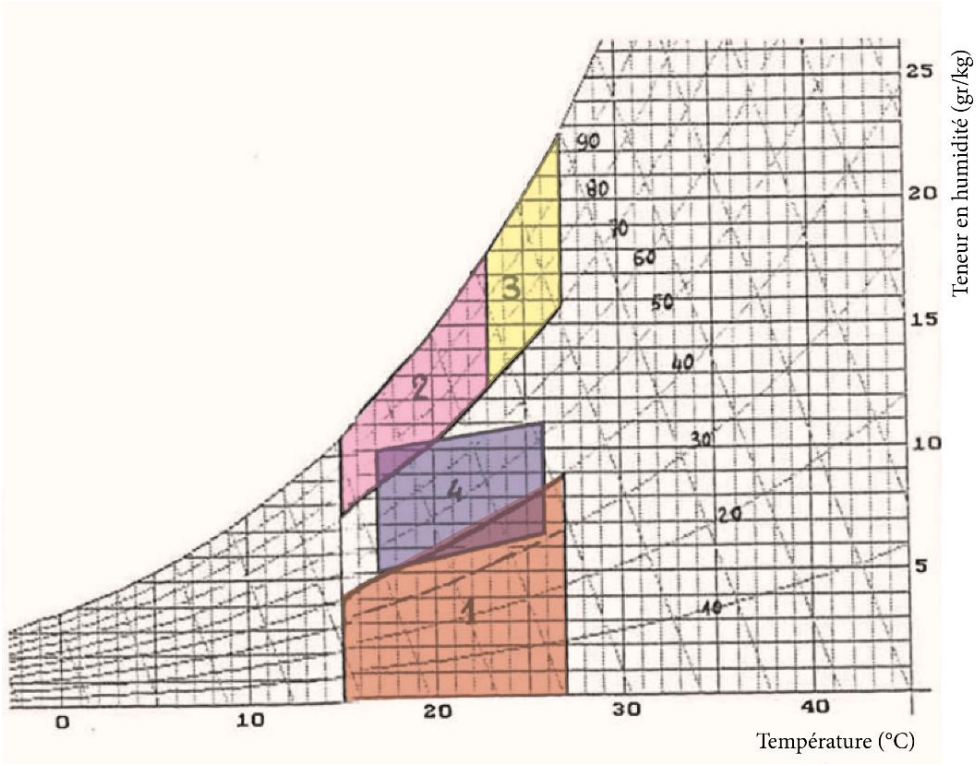


Figure 85: Plages de confort hygrothermique (source: Fauconnier, 1992)

La simulation thermique pour le confort

Le confort thermique peut être évalué en faisant un bilan thermique à l'aide de simulations physiques. Il existe plusieurs indicateurs du confort thermique.

Les indicateurs du confort thermique

D'après la revue de littérature sur les indicateurs du confort thermique de (Enescu, 2017), il en existe de nombreux. Les deux indicateurs les plus classiques sont le *Predicted Mean Vote* (PMV) et le Predicted Percentage of Dissatisfied (PPD). L'indice PMV est défini par une équation complexe explicitée dans la norme ISO 7730. Il inclut les six paramètres précédemment énoncés. Il cherche à prédire la réponse moyenne d'un grand groupe de personnes si on leur demandait de noter leur ressenti en termes de confort thermique : très chaud (+3), chaud (+2), légèrement chaud (+1), neutre (0), légèrement frais (-1), frais (-2), froid (-3). Une note moyenne comprise entre -0.5 et 0.5 est considérée comme confortable. Cet indice est valable pour les adultes en bonne santé (pas les malades et les enfants) et pour des espaces intérieurs climatisés (de type bureaux) dans des conditions stables.

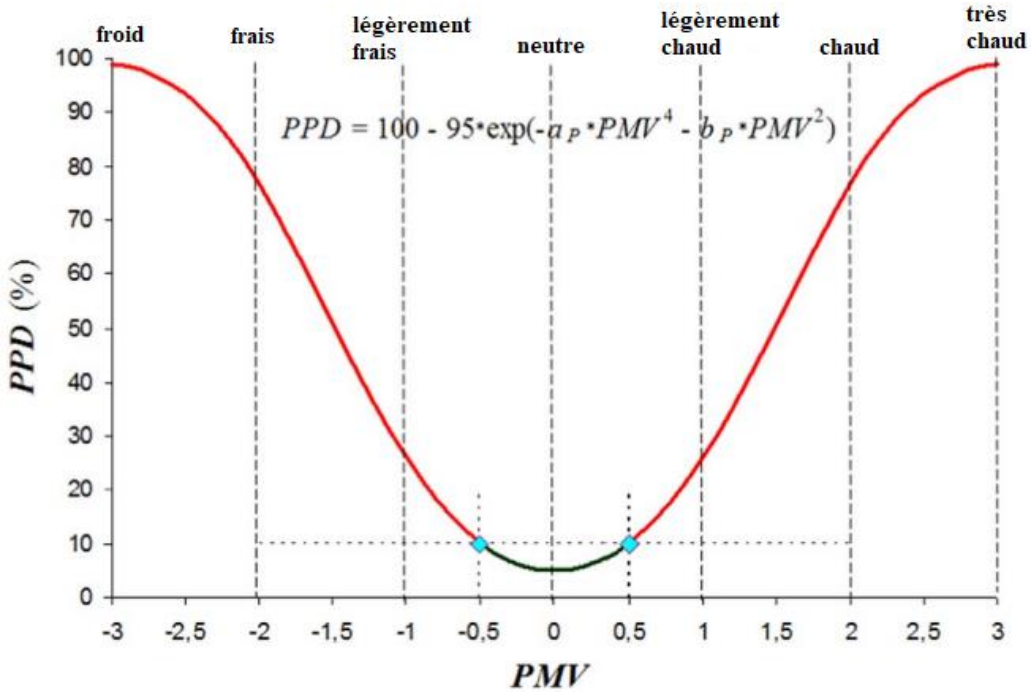


Figure 86: Dépendance du PPD au PMV (source: Enescu, 2017)

L'indice PPD s'appuie sur l'indice PMV pour prédire le pourcentage de personnes qui seraient insatisfaites du niveau confort thermique. Pour un PMV compris entre -0,5 et 0,5, cela correspond à moins de 10 % de personnes insatisfaites pour qu'un espace intérieur soit considéré comme confortable.

Pour l'évaluation du confort thermique des espaces extérieurs, l'indice le plus utilisé est l'indice universel du climat thermique, en anglais, *Universal Thermal Index* (UTCI) (Błażejczyk et al., 2013). Il permet de prédire le stress thermique ressenti par l'Homme en prenant en compte la température de l'air, la température radiante moyenne, la vitesse des vents, l'humidité et l'humidité relative. L'UTCI s'exprime en degrés et se décline en dix catégories de stress présentée dans le Tableau 6.

Tableau 6 : Température équivalente UTCI catégorisée en termes de stress thermique (source: Błażejczyk et al., 2013)

UTCI (°C)	**Catégorie de stress**
au dessus de +46	Stress thermique extrême
entre +38 et +36	Très fort stress thermique
entre +32 et +38	Fort stress thermique
entre +26 et +32	Stress thermique modéré
entre +9 et 26	Absence de stress thermique
entre +9 et 0	Léger stress dû au froid
entre 0 et -13	Stress modéré dû au froid
entre -13 et -27	Fort stress dû au froid
entre -27 et -40	Très fort stress dû au froid
en dessous de -40	Stress dû au froid extrême

Outils d'analyse du confort thermique pour l'optimisation

Dans les deux cas, ces indicateurs nécessitent pour être calculés de modéliser et simuler les transferts thermiques dans le bâtiment. Il faut donc faire de la simulation thermique avec des outils comme ceux présentés dans le chapitre sur Les outils de simulation de la performance des bâtiments (p. 44). Comme pour la simulation de la lumière, des plugins permettent l'interopérabilité entre des outils de simulation thermique et des modeleurs 3D. Openstudio permet de faciliter les échanges entre *Google Sketchup 3D*® et *Energy Plus*®. Le couple *Ladybug/Honeybee*® permet de relier *EnergyPlus*® à *Grasshopper*® ou *Dynamo*®. *TRNLizard*® permet d'utiliser *TRNSYS 18*® avec *Grasshopper*® (*TRNLizard*, 2017).

Ladybug/Honeybee® dans sa version 1.6. pour *Rhinocéros 7*®, propose un composant pour le calcul du PMV, « HB PMV Comfort Map » (*PMV Comfort Map*, s. d.). Il utilise *Radiance*® et *Energy Plus*® pour prédire les températures intérieures et de surface ainsi que l'humidité à partir des données météo de température de l'air extérieur, d'humidité relative et de vitesse de

l'air. L'indice PMV est calculé pour chaque capteur (chaque point de la grille modélisée) de la pièce. Comme le montre la Figure 87, les résultats enregistrés dans un fichier csv peuvent être visualisés en 2D sur une carte de la même manière que pour un calcul de visibilité ou de FLJ.

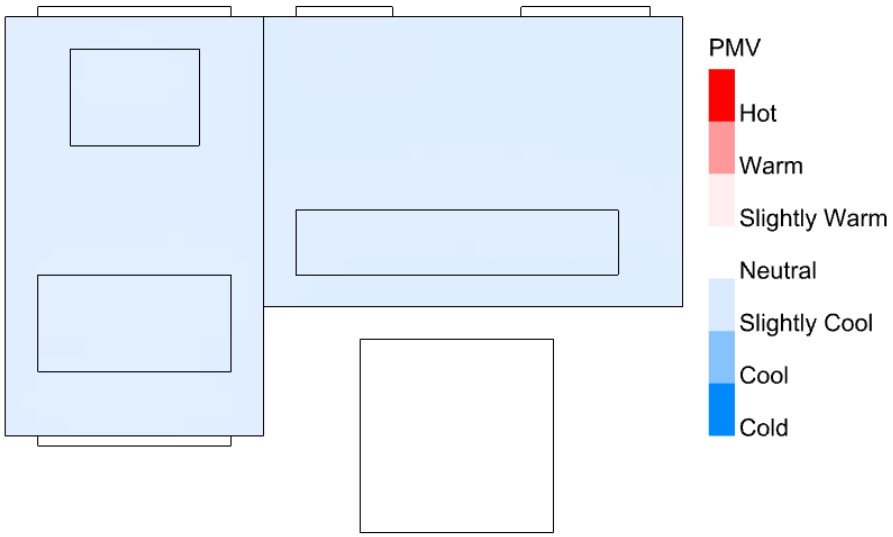


Figure 87 : Evaluation de l'indice PMV pour deux pièces

Si les temps de calcul sont raisonnables pour deux pièces à la géométrie très simple (environ 4 minutes avec un processeur Intel® Xeon® CPU E5-2630 v3 @ 240 GHz sur 16 cœurs), ils sont beaucoup plus longs si on travaille à l'échelle d'un bâtiment, ou si on intègre un système de protection solaire. De plus, la modélisation pour ce type de simulation est particulière, peu connue des architectes. Il s'agit de modéliser des zones thermiques, ce qui implique de reprendre presque entièrement la modélisation 3D. Il existe un composant équivalent pour le calcul de l'UTCI, « HB UTCI Confort Map » qui présente les mêmes difficultés en termes de temps de calcul et de complexité de modélisation. De plus, ce composant ne prend pas en compte l'évapotranspiration des végétaux et l'impact des masses sur la vitesse des vents extérieurs ce qui a un fort impact sur la qualité des résultats.

Ainsi, pour faire de l'optimisation, certains auteurs, au lieu d'utiliser un indicateur du confort, analysent certains paramètres pris en compte dans l'évaluation du confort (Duering et al., 2020), comme la radiation solaire qui impacte la température et la vitesse des vents.

L'analyse de la radiation solaire

Définition du rayonnement thermique

Il est essentiel de contrôler les apports solaires d'un bâtiment pour le confort thermique en été comme en hiver. Maximiser les apports solaires lors des saisons froides permet de réduire les

consommations de chauffage. A l'inverse, minimiser les apports en été permet d'éviter l'usage de la climatisation. Ainsi, « *pour aboutir à des constructions dont les besoins de chauffage ou de rafraîchissement sont limités, il est nécessaire : de permettre au bâtiment de garder au maximum la chaleur […] ; (et) d'utiliser au mieux l'énergie gratuite fournie par le rayonnement solaire* » (Courgey & Oliva, 2006).

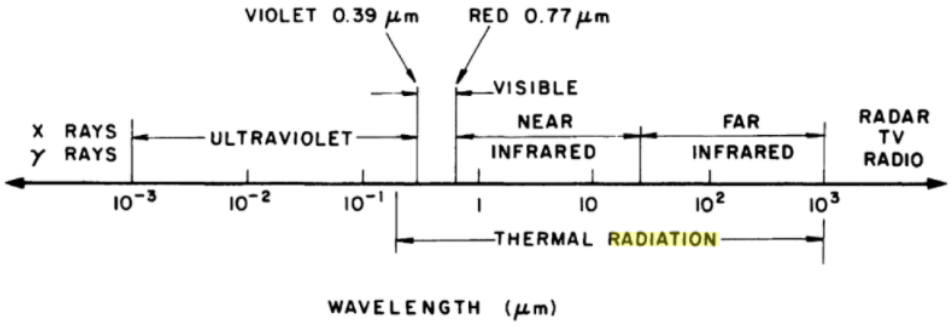


Figure 88 : Spectre électromagnétique de la radiation (source : Iqbal, 2012)

Etudier les apports solaires d'un bâtiment, revient à étudier la quantité d'énergie arrivant sur une surface du bâtiment. Cette énergie dépend de l'intensité de la radiation solaire, encore appelée rayonnement solaire. Muhammed Iqbal explique dans son *livre An introduction to solar radiation* (Iqbal, 2012) que la majorité des rayonnements qui atteignent la Terre sont des rayonnements thermiques. Il s'agit d'un rayonnement électromagnétique émit par l'agitation atomique et moléculaire. Son spectre contient différents types de rayons (gamma, X, ultraviolet…). Les rayons visibles par l'œil humain correspondent à ce que l'on appelle communément la lumière. La chaleur englobe plus de rayons que la lumière (rayons visibles pour l'Homme) mais ne corresponde qu'à une partie du rayonnement solaire comme l'illustre la Figure 88.

Comme pour les rayons lumineux, une partie de ce rayonnement thermique atteint la Terre, c'est le rayonnement direct. Une partie de ce rayonnement est réfléchi par l'atmosphère, les nuages et la Terre, on parle d'albédo planétaire. Une partie est absorbée par les molécules atmosphériques. Une partie est diffusée par l'atmosphère et les nuages. Et enfin, une partie est réfléchie sur les surfaces avoisinantes notamment le sol (l'albédo). L'ensemble des rayonnements (direct et diffus) qui atteignent une surface est appelé le rayonnement global.

L'intensité du rayonnement solaire dépend du lieu où on se trouve sur Terre. Ainsi, le premier outil à la disposition des architectes pour étudier le rayonnement solaire est le diagramme

solaire qui permet de connaître l'emplacement du soleil pour un lieu donné selon la date et l'heure (voir Figure 89).

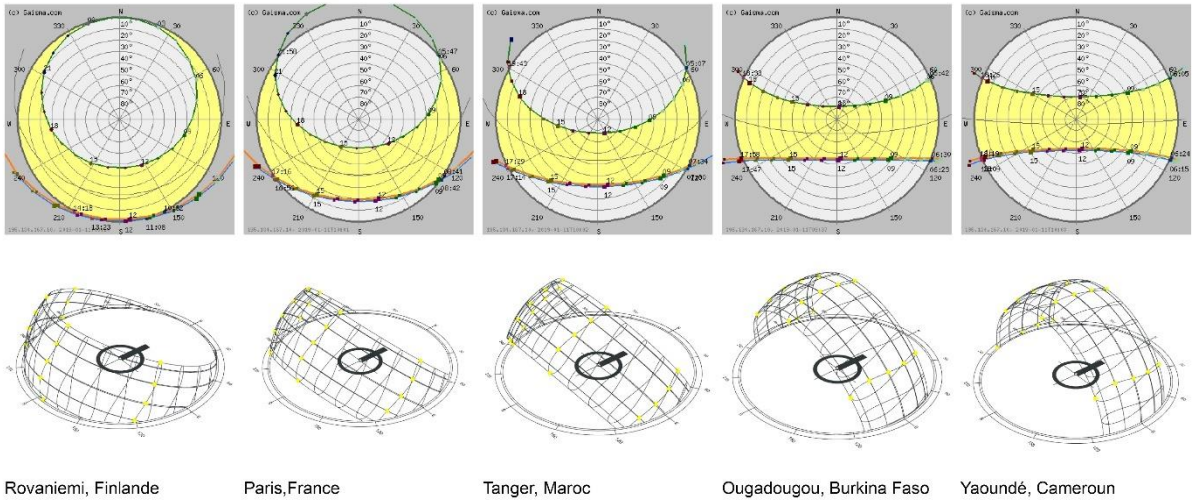


Figure 89: Diagramme solaire 2D et 3D dans ville de Rovaniemi, Paris, Tanger, Ougadougou, Yaoundé source: gaisma.com)

La quantité d'énergie solaire sur une paroi dépend de l'angle entre le rayonnement solaire et la paroi. Les matériaux et couleurs ne transmettent pas tous la même quantité de rayonnement. Cette propriété des matériaux est quantifiée avec le coefficient d'absorption. Les matériaux transparents comme le verre sont ceux qui transmettent la plus grande partie du rayonnement solaire. Ainsi, la situation géographique, la nébulosité (quantité de nuage), la géométrie, le contexte urbain, et les matériaux sont des paramètres nécessaires pour l'évaluation des apports solaires d'un bâtiment.

Outils d'évaluation de la radiation solaire

Les outils de simulation de l'éclairage naturel précédemment présentés comme *Radiance*® permettent de faire des évaluations de la radiation solaire poussées en prenant en compte les 3 composantes du rayonnement : le rayonnement direct et diffu en fonction de la localisation et des données météo et les rayons réfléchis sur les surfaces modélisées en fonction de leur coefficients d'absorption et de réflectance.

Le plugin *Ladybug*® propose un composant qui permet de faire un calcul de la radiation solaire simplifiée destiné à l'étude de géométrie simple. Il s'agit du composant « LB Incident Radiation » dans la version 1.6. pour *Rhinocéros*® 7. Il ne prend pas en compte la réflexion de l'énergie solaire en dehors de l'irradiation réfléchie par le sol de manière grossière comme le

précise le code source du composant. La modélisation est extrêmement simple. Il y a deux types de géométrie : la surface à évaluer discrétisée en une grille de points plus ou moins fine selon la précision souhaitée, et les géométries du contexte qui créent des masques solaires. Aucun autre paramètre n'est nécessaire en dehors d'un fichier météo. Les résultats calculés rapidement (quelques secondes) sont ensuite mappés directement sur les surfaces évaluées.

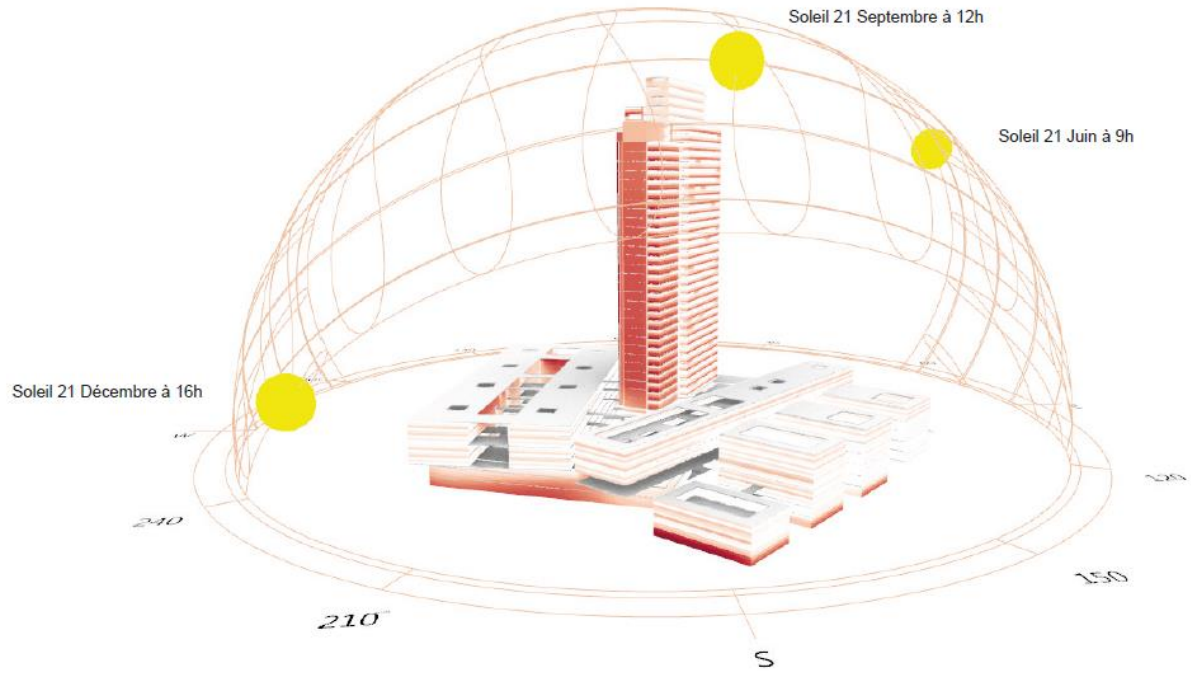


Figure 90 : Evaluation de la radiation solaire et diagramme solaire pour un hôpital dessiné par AS en phase concours

Pour une évaluation plus précise ou avec une géométrie complexe, le composant *Honeybee*® « HB Annual Irradience » qui permet d'utiliser Radiance est plus adapté. La modélisation et le fonctionnement sont similaires à une analyse de l'éclairement.

Les résultats de l'analyse sont donnés en kWh/m^2. Les apports solaires variant selon la localisation des projets (voir Figure 91), les résultats ne sont pas toujours faciles à interpréter. Il est intéressant de comparer les résultats avec ceux d'une analyse du même bâtiment sans masque solaire pour comprendre leur impact. On peut aussi effectuer une autre analyse avec un fichier météo différent, par exemple d'une ville où on a une connaissance empirique des principes de conception bioclimatique.

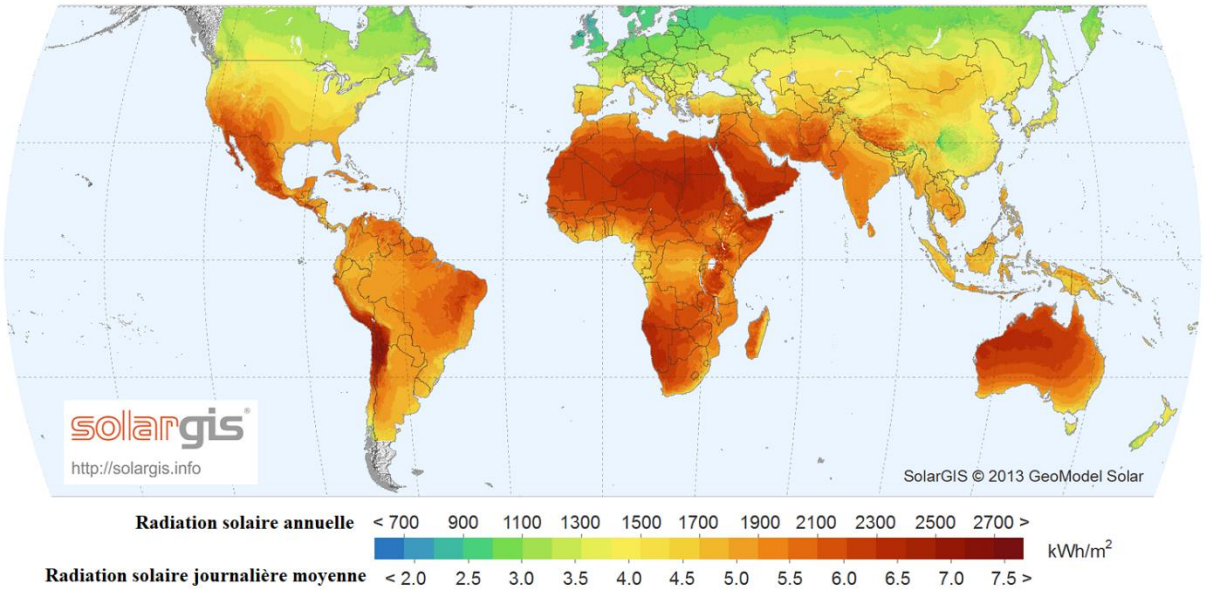


Figure 91 : Carte mondiale de l'irradiation solaire globale (source: Wikipédia)

Pour être appliqués à de l'optimisation, il est intéressant de distinguer les apports en saison chaude, qu'il faut alors minimiser, des apports en saison froide, qu'il faut maximiser. Si on souhaite plus de précisions qu'en utilisant les saisons, il est possible sur *Grasshopper®* d'utiliser les données météo, pour définir les périodes d'analyses et ne prendre en compte dans le calcul que les jours où la température de l'air extérieur est supérieure ou inférieur à un certain seuil. L'évaluation de la radiation solaire peut aussi être utilisée pour la production d'énergie, il faut alors indépendamment des saisons chercher à maximiser les apports.

L'étude des vents

La modélisation CFD

Que ce soit pour les espaces extérieurs ou intérieurs, la ventilation naturelle a un impact sur la qualité de l'air et le confort thermique. Prendre en compte la ventilation naturelle peut permettre de réduire les consommations énergétiques. Le vent peut aussi être utilisé pour la production d'énergie (Zarrabi et al., s. d.). Comme le montre la Figure 92, les formes architecturales ont un fort impact sur les flux de vent, et certains architectes considèrent le vent comme un outil de conception (Kabošová et al., 2020). En 2015, (W. Guo et al., 2015) ont montré à l'aide d'une étude comparative que la simulation aéraulique (CFD) jouait un rôle clef dans l'optimisation de la performance des bâtiments sur trois aspects : le plan masse, la forme du bâtiment et la conception de l'enveloppe.

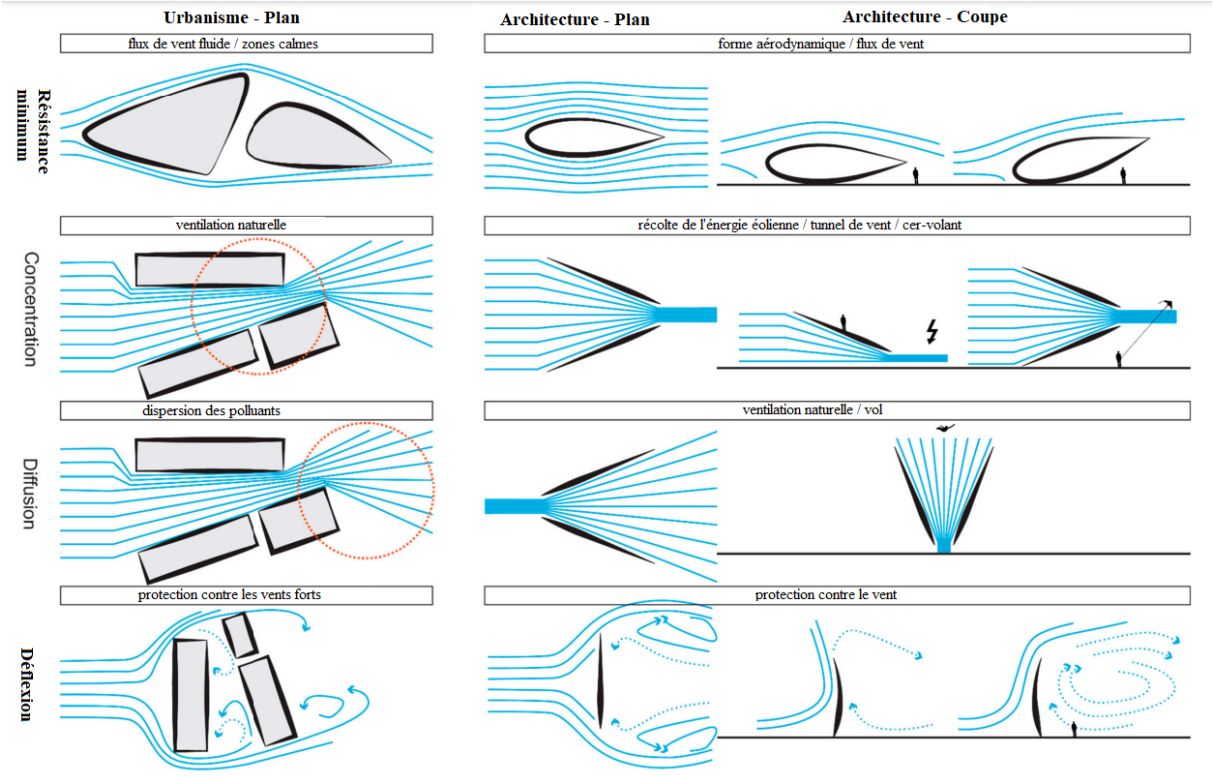


Figure 92 : Les formes architecturales affectant les flux de vent (Kabošová et al., 2020)

L'étude numérique des vents pour le bâtiment implique aujourd'hui de faire de la simulation aéraulique, en anglais, *computational fluid dynamics* (CFD). La modélisation CFD est particulièrement complexe, nécessitant des connaissances en physique, elle est longtemps restée inaccessible pour les architectes. Aussi, elle requiert une importante puissance de calcul, avec des temps de calcul longs restant importants. Il existe plusieurs moteurs de calcul aéraulique notamment *OpenFOAM®* (Jasak, 2009). Il s'agit d'un solveur open source puissant et robuste développé à l'origine sur Linux (il fallait passer par une machine virtuelle pour l'utiliser sur Windows). Aujourd'hui il en existe une version Windows, mais il reste difficile d'accès car les guides d'utilisation ne sont pas destinés aux non-experts. D'après (Bouhelal & Smaïli, 2022), il existe d'autres solveurs opensources comme *OpenFlower®*, *Code_sature®*, *OpenLB®*, *SLFCFD* ou commerciaux comme *COMSOL Multiphysics®* (Multiphysics, 1998), *Ansys-Fluent* (Matsson, 2022), *Ansys-CFX®*, *EasyCFD®*, *SimFlow®*, *CONVERGE®* ou encore *CHAM PHOENICS®*.

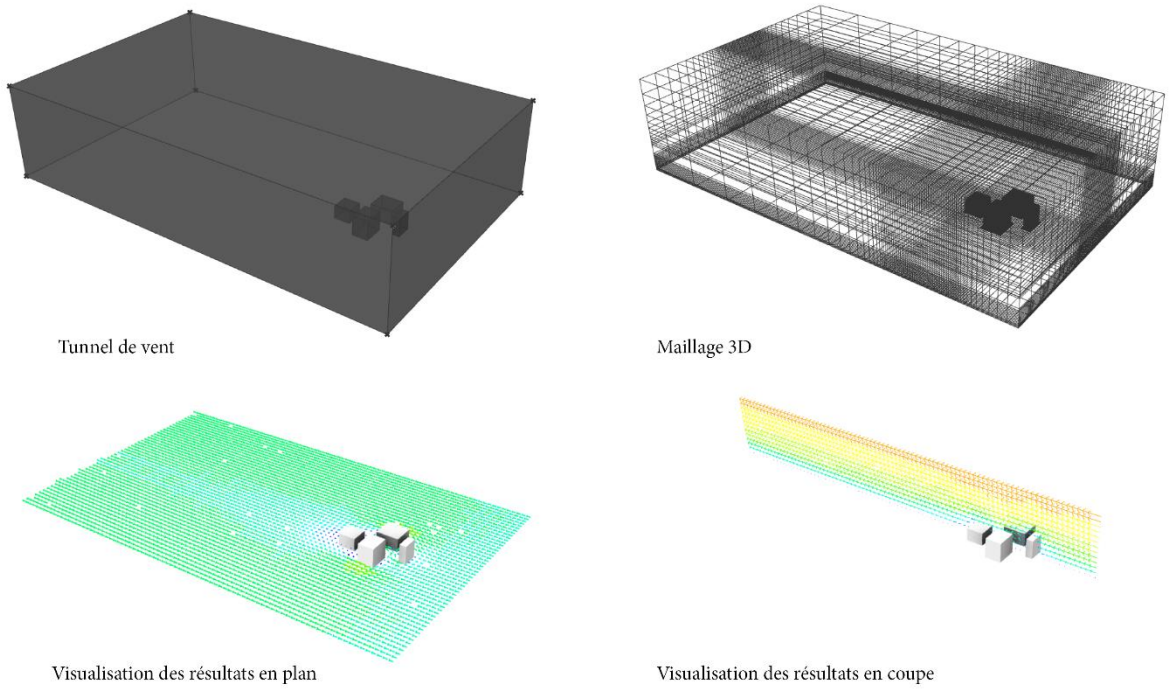


Figure 93: Modélisation CFD et visualisation

Ils ont tous un mode de fonctionnement similaire. La modélisation se passe en 5 étapes : (1) la modélisation CAO du problème, (2) la modélisation d'un tunnel de vent, c'est-à-dire un volume 3D dans lequel les mouvements de vent vont être simulés, (3) la discrétisation adaptative 3D du tunnel, c'est-à-dire un maillage 3D structuré ou non (qui peut être orthogonal, tétraédrique, prismatique ou pyramidal) s'affinant à certains endroits pour plus de précisions notamment aux abords de géométries complexes, (4) la simulation, résolution d'équations différentielles de façon itérative à chaque nœud du maillage, (5) le processus de post traitement pour la visualisation des résultats (voir Figure 93).

Les plugins CFD

D'après (Hu et al., 2023), il existe des plugins CFD qui permettent aux architectes de faire de la CFD en réduisant la nécessité d'acquérir des connaissances en physique fondamentale, en fournissant des guides d'utilisation et des interfaces graphiques adaptées. Ces plugins ont été développés pour *Revit*®, *Rhinocéros*®, *Grasshopper*® et *Sketchup*® (voir Figure 94).

Nous nous sommes particulièrement intéressés aux plugins de *Rhinocéros*® et *Grasshopper*® car les autres plugins ne permettent pas de coupler les résultats de simulations à un solveur d'optimisation. *RhinoCFD*® s'appuie sur le moteur de calcul de *CHAM*

PHOENICS®. Il utilise un maillage orthogonal qui est facile à paramétrer mais pas toujours efficace pour la géométrie complexe. Le paramétrage est accessible pour des utilisateurs aux connaissances limitées comme un architecte (Chronis et al., 2017). Cependant, la licence coute 2000 $, et il n'est pas encore relié à *Grasshopper*®, on ne peut donc pas l'utiliser pour faire de l'optimisation.

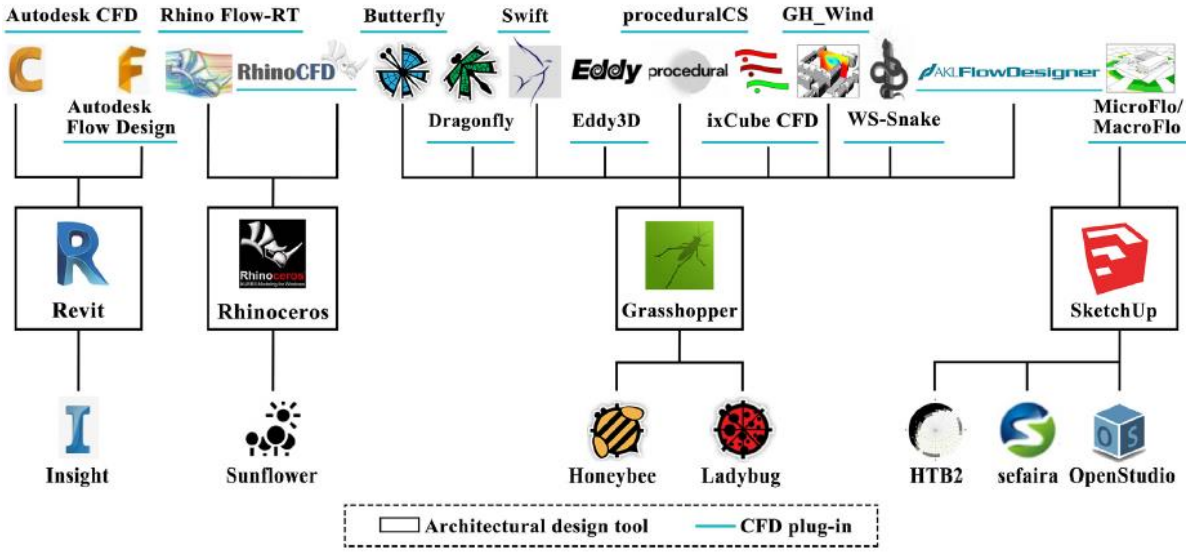


Figure 94: Diagrammes des plugins CFD (source: Hu et al., 2023)

Le plugin *Butterfly*® appartient à la suite *Ladybug*®. Il nécessite un bagage de connaissance en modélisation CFD pour être utilisé. Même s'il existe des configurations de base pour les utilisateurs novices, il reste complexe à prendre en main. Tous les paramètres de simulation sont ouverts à la personnalisation ce qui permet de faire de la modélisation avancée.

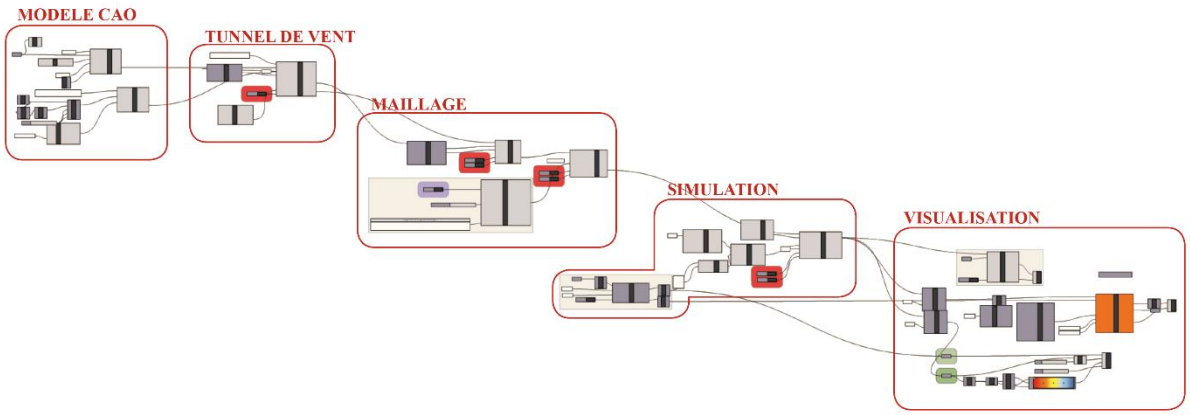


Figure 95 : Script Butterfly sur Grasshopper pour l'analyse des flux de vent extérieurs

Eddy3D® (Kastner & Dogan, 2022) a été développé dans le cadre d'un projet de recherche pluridisciplinaire de l'Université de Cornell, USA. *Eddy3D*® est beaucoup plus

facile d'accès, les temps de calculs sont plus faibles mais l'instabilité de certains composants de post-traitement des résultats rend difficile son usage dans une boucle itérative automatique comme un algorithme d'optimisation.

Le plugin *Swift*® pour *Rhinocéros*® est développé par ODS ingénierie. Il fonctionne avec une version linux de *OpenFOAM*® et sous Rhino 5. Il n'y a pas eu de nouvelle version depuis 2017. ixCube® CFD est un plugin payant qui permet d'utiliser *OpenFOAM*®. *ProceduralCS*® est un outil payant qui permet de faire de la simulation en ligne, notamment de la CFD.

GH_wind® (Waibel et al., 2017) utilise la méthode FFD (fast Fluid Dynamics) codée directement dans *Rhinocéros*® et *Grasshopper*®, ce qui permet de réduire les temps de calcul. Il est utilisable uniquement pour les flux d'air dans un contexte urbain (autour des bâtiments).

Ws-*snake*® développé en 2019 permet de calculer la pression des vents en façade. La méthode de calcul est fondée sur le Code National du Bâtiment canadien de 2015. Le plugin a été développé pour être utilisé avec le plugin *Karamba3D*® pour faire des calculs structurels. Il est notamment adapté pour la conception de façades vitrées.

Tableau 7 : Les plugins CFD de Grasshopper

Plugin CFD	**CFD Solveurs**	**Optimisation**	**Prix**
Butterfly®	OpenFoam sur Windows	Possible, mais temps de calcul important	gratuit
Swift®	OpenFoam sur Linux	Possible	gratuit
Eddy 3D®	OpenFoam® sur Windows	Boucle impossible (instabilité d'un composant). Etude urbaine uniquement.	gratuit
ProceduralCS®	OpenFoam®	Impossible car les calculs sont réalisés dans le cloud	gratuit
ixCube CFD®	OpenFOAM®	Non testé	100 €/an
GH_wind®	Méthode FFD	Possible, uniquement pour les études urbaines	gratuit
WS-snake®	_	Possible, uniquement pour les études de façade	gratuit

Pour faire une analyse du confort des espace extérieurs nous recommandons d'utiliser le plugin *Eddy3D*®. Pour faire de l'optimisation du confort extérieur, nous recommandons

d'utiliser *GH_wind*®. Pour l'analyse et l'optimisation des espaces intérieurs, et pour coupler simulation CFD et énergétique, le plugin le plus adapté aujourd'hui est *Butterfly*®.

La protection contre la pluie

Les précipitations entraînées par le vent sur les enveloppes affectent la durabilité des bâtiments. Aussi, certains espaces extérieurs d'un bâtiment nécessitent d'être protégés contre les intempéries, il faut alors concevoir des éléments architecturaux qui répondent à cette fonction. C'est pourquoi il est intéressant de pouvoir vérifier la capacité d'un dispositif à protéger de la pluie. La modélisation CFD des mouvements de pluie entrainée par le vent est bien trop complexe pour des études en phase amont de projet (AREP L'hypercube, s. d.), mais nous pouvons dans un premier temps utiliser des lancers de rayons pour analyser l'intersection entre une pluie battante et une façade.

Pour cela il est nécessaire de calculer la trajectoire des gouttes de pluie. L'hypercube, la cellule de recherche de l'agence AREP propose un premier graphique pour estimer la vitesse de déplacement des gouttes de pluie en fonction de leur taille basé sur les travaux de (Gunn & Kinzer, 1949) et un second graphique pour estimer leur taille en fonction de l'intensité de pluie à partir des travaux de (Best, 1950). Ces deux graphiques sont présentés en Figure 96.

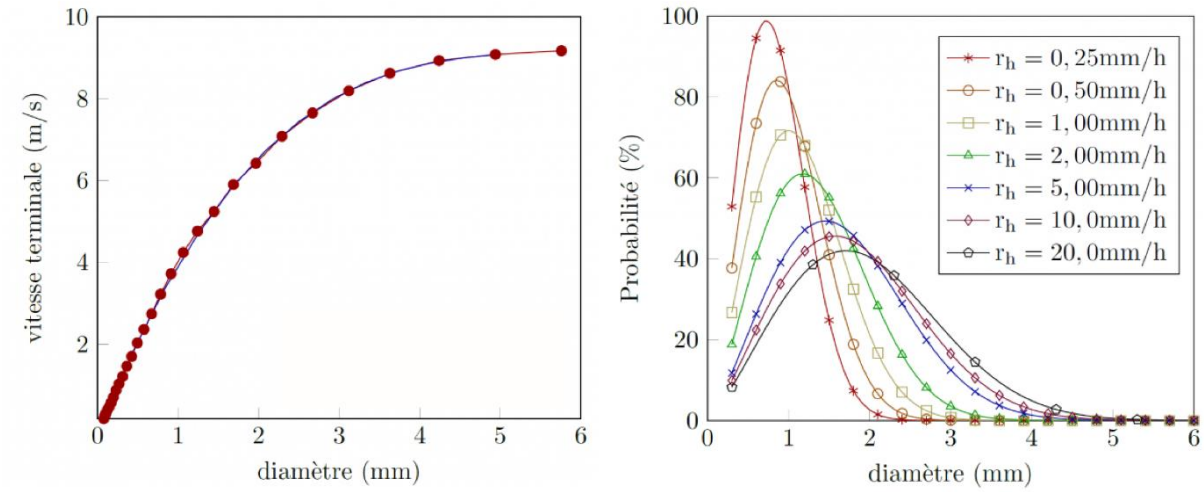


Figure 96 : A gauche vitesse des gouttes de pluie en fonction de leur taille, à droite distribution des tailles de gouttes pour différentes intensités de pluie (source: https://lhypercube.arep.fr/aeraulique/pluie-entrainee-par-le-vent-2/)

Les paramètres de l'orientation du vent et la vitesse des gouttes permettent de définir la trajectoire des gouttes de pluie entrainées par le vent avec la définition de (Blocken & Carmeliet, 2004) illustrée en Figure 97.

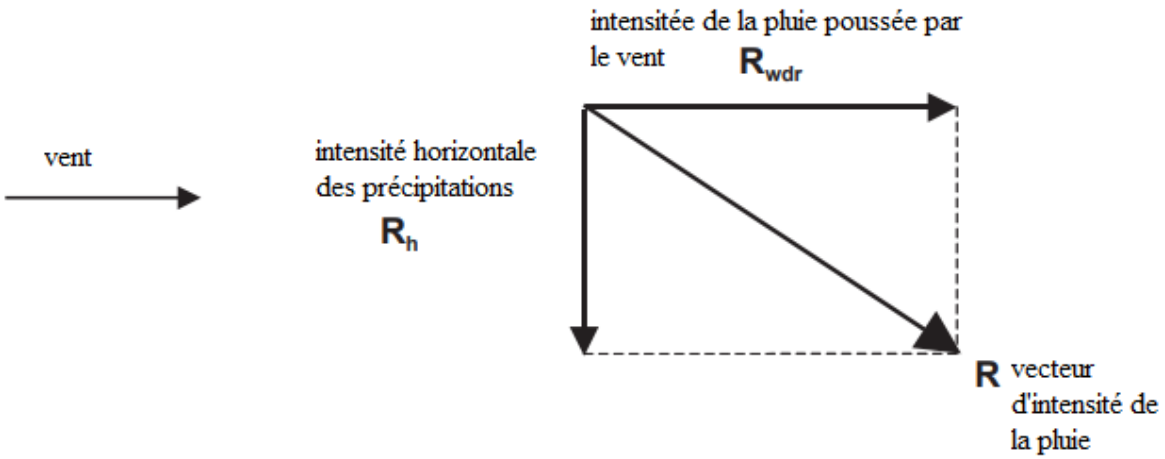
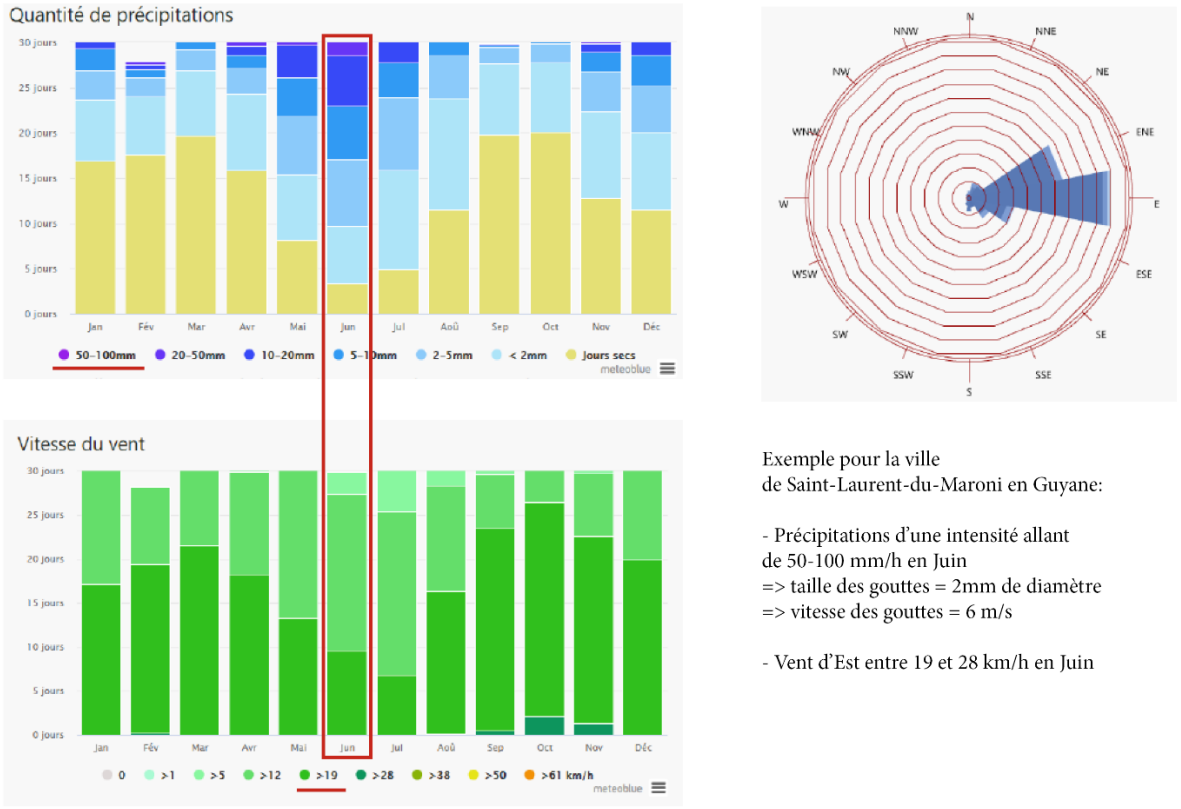


Figure 97 : Définition du vecteur d'intensité de la pluie (source: Blocken & Carmeliet, 2004)

Ainsi, à partir des données météo locales, comme celles illustrées en Figure 98, l'intensité des précipitations, la vitesse et l'orientation des vents, nous pouvons définir un vecteur pour notre outil à base de lancer de rayonq développé sur *Grasshopper*®.



Exemple pour la ville
de Saint-Laurent-du-Maroni en Guyane:

- Précipitations d'une intensité allant
de 50-100 mm/h en Juin
=> taille des gouttes = 2mm de diamètre
=> vitesse des gouttes = 6 m/s

- Vent d'Est entre 19 et 28 km/h en Juin

Figure 98 : Données météo de Saint-Laurent-de-Maroni-Guyane française (source: www.meteoblue.com)

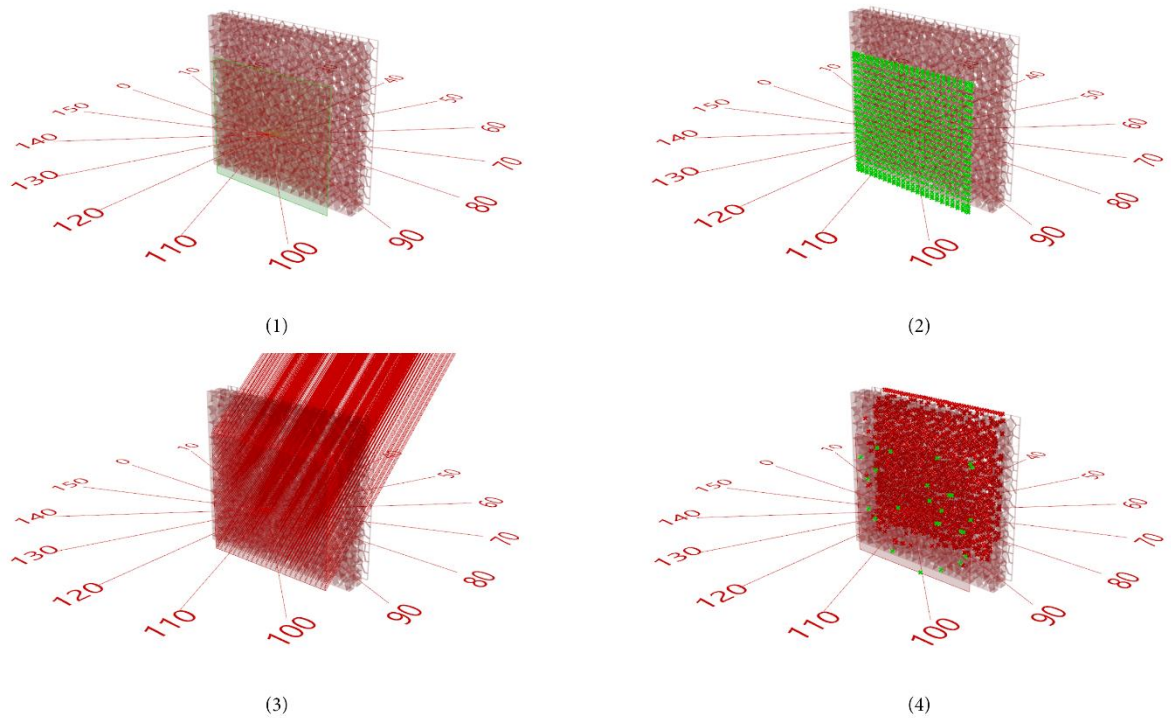


Figure 99 : Méthode paramétrique d'analyse des précipitations

La méthode fonctionne en 4 étapes présentées dans la Figure 99 : (1) modélisation 3D du problème et paramètres (vitesse des gouttes et orientation et vitesse du vent), (2) discrétisation en grille de points de la surface à étudier, (3) simulation de la trajectoire des gouttes avec un lancer de rayons, (4) visualisation des gouttes atteignant la surface analysée.

1.3.4. Energie et carbone

Pour assurer la performance énergétique et environnementale d'un bâtiment, la situation idéale serait de pouvoir prédire finement ses consommations d'énergie et son impact carbone. Ce type d'évaluation n'est pas toujours adapté aux phases amonts de conception et à l'optimisation pour plusieurs raisons. D'abord car les données nécessaires pour ce type de simulations ne sont pas toujours disponibles en phase esquisse, ensuite car les temps de calculs sont souvent conséquents (Marsault, 2018). Enfin, nous pouvons ajouter que ces évaluations sont plus techniques et donc moins facilement abordables pour les architectes. Cependant, il existe aujourd'hui des recherches et outils pour tenter de démocratiser ce type d'évaluations auprès des architectes en phases amonts de projet.

La simulation énergétique

Pour étudier la performance thermique d'un bâtiment il existe des méthodes détaillées qui s'appuient sur des outils de simulation (TRNSYS, DOE2, Energy Plus) présentées dans la partie

sur Les outils de simulation de la performance des bâtiments (p44), et des méthodes simplifiées comme celle de la RE2020 (Salomon et al., 2005). La simulation énergétique permet de modéliser le comportement thermique d'un bâtiment. Il existe différents types de simulation notamment la simulation statique, ou la simulation thermique dynamique (STD) réputée plus précise. Aujourd'hui, la STD est de plus en plus intégrée au processus de conception d'un bâtiment. Dans l'état de l'art des applications présenté dans la partie 1.2. de ce chapitre, les prédictions des consommations, des besoins énergétiques et du confort thermique ont été majoritairement réalisées à l'aide de STD.

Lorsqu'on parle de prédiction des besoins nets en énergie, il s'agit du niveau d'énergie que le système de chauffage et/ou de climatisation doivent fournir pour maintenir une certaine température et du niveau d'énergie que le système d'éclairage doit fournir pour maintenir un certain niveau d'éclairement. Les besoins ne dépendent pas des installations techniques, contrairement aux consommations. Dans la RE2020, les besoins correspondent à l'indicateur du Bbio (indicateur de besoin bioclimatique) qui se calcule selon la formule suivante : (2x besoins de chauffage) + (2x besoins en refroidissement) + (5x besoin en éclairage).

La prédiction des consommations d'énergie finalees prend en compte dans son calcul les déperditions liées au mode la production d'énergie, à la distribution et à l'émission. Dans la RE2020, deux indicateurs sont utilisés : le Cep pour l'ensemble des consommations en énergie primaire, et le Cep.nr pour les consommations en énergie primaire non renouvelables. Les consommations pour le chauffage, le refroidissement, la production d'eau chaude sanitaire, d'éclairage, de ventilation et auxiliaires sont prises en compte dans le calcul Cep.

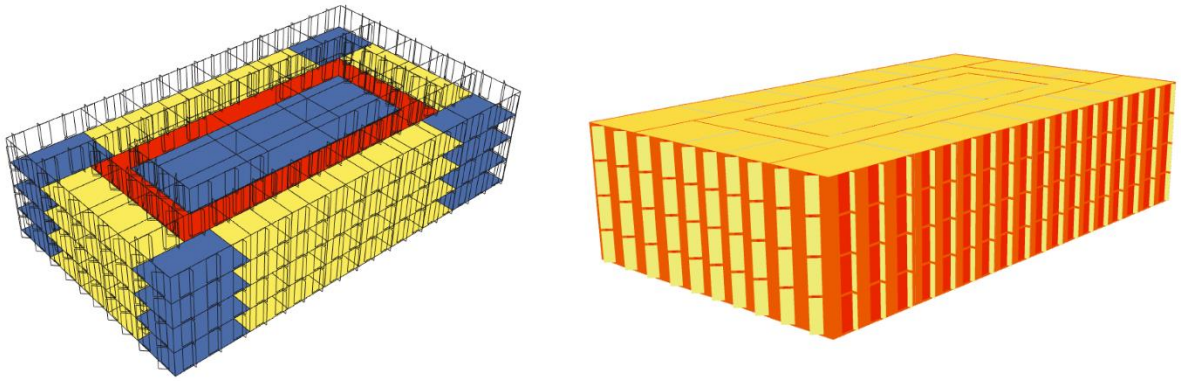


Figure 100 : Exemple de modèle thermique modélisé avec *Honeybee*®

Les outils et plugins qui permettent de faire de la simulation ont déjà été présentés dans le chapitre sur le confort thermique. Parmi les références citées en partie 1.2. de ce chapitre, *Honeybee*® couplé à *Energy Plus*® est le plugin *Grasshopper*® le plus utilisé pour la

simulation thermique. Pour faire de la simulation thermique, il faut un modèle 3D du bâtiment bien différent des modèles 3D que modélisent et utilisent les architectes. On parle de « *modèle thermique* ». Dans un modèle thermique (voir Figure 100), chaque espace est représenté comme un volume fermé à la forme simple, il s'agit du « *zonage thermique* ». Les murs sont représentés par des surfaces sans épaisseurs. L'élément le plus détaillé du modèle est l'enveloppe où on trouve des surfaces pour représenter le vitrage (appliquée sous forme de ratio de surface vitrée) et des surfaces à la géométrie très simple pour représenter des systèmes de protection solaire. Cette modélisation peut se faire sur *Rhinocéros*®.

Il y a ensuite beaucoup de paramètres à définir depuis l'interface *Grasshopper*® (voir Figure 101). En effet, de nombreux paramètres ont un impact sur le comportement thermique d'un bâtiment : les paramètres physiques des matériaux, l'exposition à la lumière naturelle et aux rayonnements solaires, la météo locale, les scénarios d'occupation et le comportement des usagers sont à prendre en compte pour prédire les besoins énergétiques d'un bâtiment. Pour prédire les consommations d'énergie, il faut aussi ajouter les systèmes CVC.

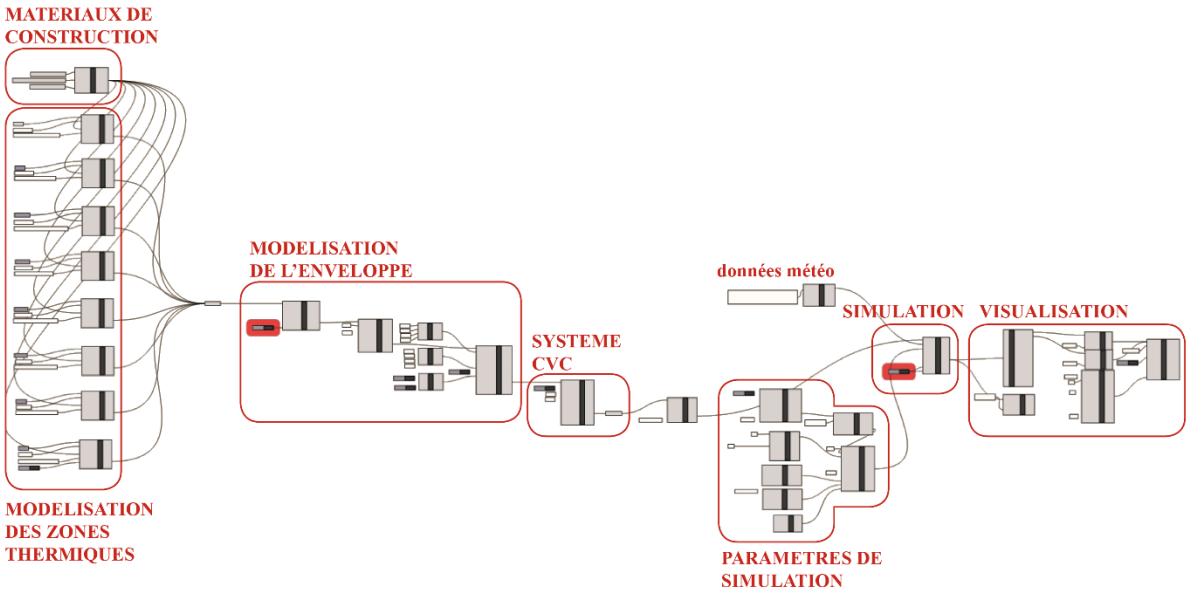


Figure 101 : Script *Honeybee*® pour la prédiction des consommations

Au fur et à mesure des versions du plugin, le paramétrage a été facilité. Des paramètres par défaut pour les matériaux et scénarios d'usage ont été définis pour rendre cet outil particulièrement adapté aux phases amonts de conception, cependant ces paramètres sont basés sur des standards américains.

Une fois la simulation lancée, les temps de calcul sont très variables selon la taille du modèle à évaluer mais aussi, comme le montre la Figure 102, sa complexité géométrique (Gerber & Lin, 2014). Pour calculer les besoins et consommations en éclairage, la simulation thermique peut être utilisée avec une simulation d'éclairage avec *Honeybee*® et *Radiance*®, ce qui allonge encore les temps de calcul.

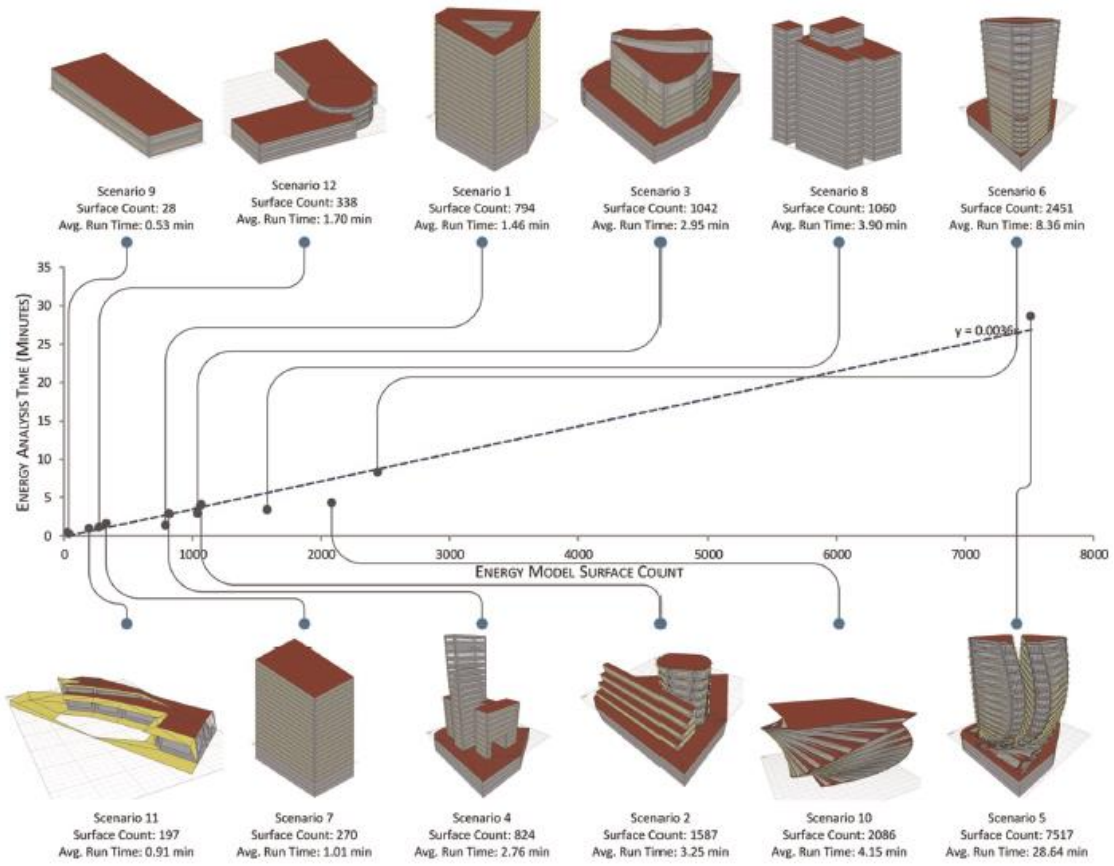


Figure 102 : Temps de calcul en fonction de la complexité géométrique (source: Gerber & Lin, 2014)

Au final, le plugin permet de visualiser la balance énergétique du bâtiment et le détail des besoins énergétiques au fil des mois et par zone thermique.

Analyse de cycle de vie (ACV)

D'après le site inies.fr sur l'ACV bâtiment, *« l'analyse de cycle de vie (ACV) est une méthode d'évaluation environnementale multi étape et multicritère permettant de quantifier l'impact environnementale d'un produit, d'un service ou d'un procédé ou d'un ouvrage sur l'ensemble de son cycle de vie »* (voir Figure 103). L'ACV d'un bâtiment est composée de la somme des ACV des matériaux de constructions utilisés, de l'eau et de l'énergie consommées en phase d'exploitation, ainsi que des impacts liés au chantier. Une ACV bâtiment consiste donc à faire

un inventaire des produits utilisés pendant la phase de construction et à prédire les consommations en phase d'exploitation. La norme NF EN 15978 détaille la méthode de calcul de l'évaluation de la performance environnementale des bâtiments. Les résultats d'une ACV bâtiment dépendent de la base de données des impacts environnementaux utilisée. En France, la base de données de référence pour l'évaluation de l'impact carbone est Inies (inies.fr). Elle contient une base de données sur les produits de construction (FDES) et sur les équipements de construction (PEP).

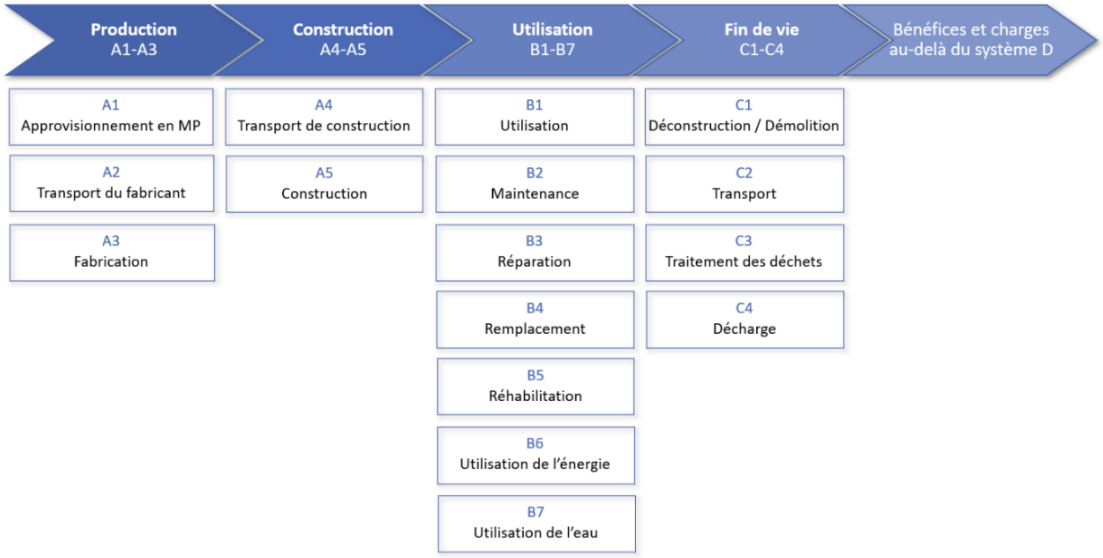


Figure 103 : Les cinq étapes du cycle de vie d'un bâtiment (source : https://www.inies.fr/inies-pour-le-batiment/lacv-batiment/)

Dans la RE2020 quatre contributions sont prises en compte : (1) les produits de construction et équipements, (2) les consommations d'énergie et d'eau, et le traitement des déchets pendant le chantier, (3) les consommations d'énergie de la phase d'exploitation, (4) les consommations et gestions de l'eau de la phase d'exploitation. Elles sont traduites sous la forme de deux indicateurs : Icénergie et Icconstruction. Icénergie est l'indice carbone énergie, il mesure l'impact carbone des consommations d'énergie pendant toute la durée de vie du bâtiment (50 ans). Icconstruction mesure l'impact carbone des produits de constructions et du chantier. 13 lots doivent être pris en compte : (1) voiries et réseaux divers, (2) fondations et infrastructures, (3) superstructure/maçonnerie, (4) couvertures, étanchéité, charpente, zinguerie, (5) cloisonnements, doublage, plafonds suspendus, menuiseries intérieures, (6) façades et menuiseries extérieurs, (7) revêtements des sols, murs et plafonds, chape, peintures, produits de décoration, (8) chauffage, ventilation, refroidissement, eau chaude sanitaire, (9) installations

sanitaires, (10) réseaux d'énergie, (11) réseaux de communication, (12) appareils élévateurs et autres équipements de transport intérieur et (13) équipement de production locale d'électricité.

Il existe de nombreux logiciels pour faire de l'ACV, chaque pays développant ses propres outils avec leurs propres bases de données. Certains intègrent des fonctions de modélisation 3D, ou des méthodes de prédiction des besoins énergétiques, d'autres ne prennent en comptent que le calcul d'impact des matériaux (Hollberg & Ruth, 2016). Le site inies.fr ressence 8 outils pour réaliser une ACV bâtiment : *ClimaWin* par BBS Slama, *SustainEcho*, *OneClick LCA* par Biovana Ltd, *Pleiades* par IZUBA énergies, *U21Win V6* par Logiciels PERRENOUD, *Béa* par Bastide Bondoux, *Vizcab Eval* par Combo Solutions et *Nooco*.

De nombreux plugins pour faire de l'ACV sur *Grasshopper*® à partir d'une maquette *Rhinocéros*® ont été développés ces dernières années. (Säwén et al., 2022) ont identifié 13 outils *Grasshopper*® pour l'analyse du cycle de vie en phases amonts. Certains sont des outils d'interopérabilité qui permettent d'utiliser un logiciel d'analyse de cycle de vie depuis *Grasshopper*® et d'autres sont des outils d'analyse directement intégrés dans *Grasshopper*®. Les outils de la seconde catégorie sont décrits par les auteurs comme les mieux adaptés pour une ACV à grande échelle. Ils comparent 4 outils open-source (*Carbinal LCA*®, *Tortuga LCA*®, *Bombyx*®, et *BHoM LCA*®) qu'ils ont jugés les plus matures parmi les 13 identifiés. Finalement, Cardinal LCD et Bombyx seraient les plus adaptés aux phases amonts de projet d'après les auteurs, nous les avons donc étudiés plus en détail.

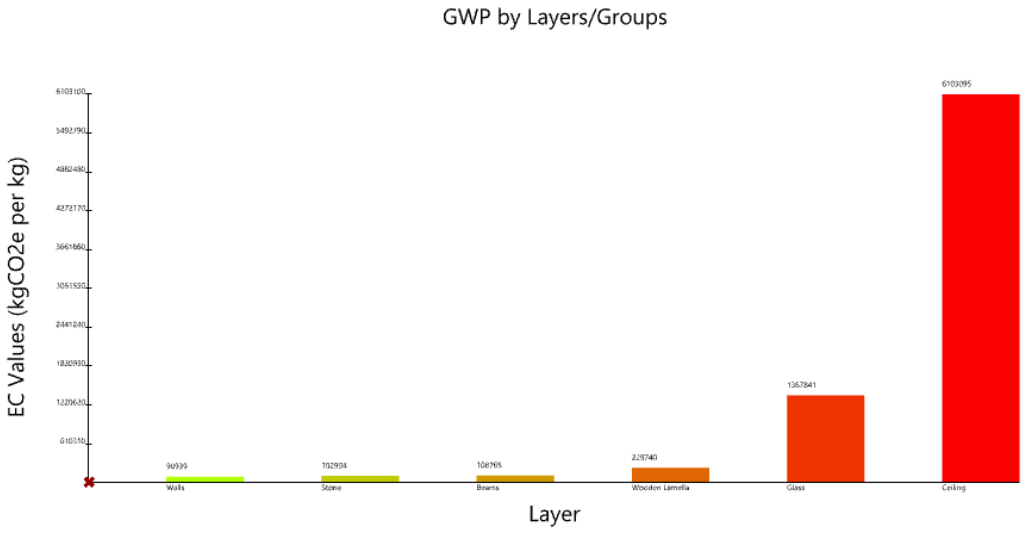


Figure 104 : Visualisation des résultats d'une ACV avec le plugin Cardinal

Cardinal® ne prend en compte que l'impact carbone des matériaux de construction de la structure et des éléments de façade. Il utilise une base de données américaine (EC3) et une base anglaise (ICE 2019). Il s'agit d'une boîte noire, il est notamment difficile de connaitre la période de référence. Il semble très simple d'utilisation et ne permet de calculer qu'un indicateur : le GWP (Global Warming Potential) illustré en Figure 104.

Bombyx®, développé par l'ETHZ est le plugin le plus complet. Il permet de calculer l'impact carbone des consommations d'énergie pendant la phase d'exploitation du bâtiment et l'impact carbone des matériaux de la fabrication à la fin de vie du bâtiment, en prennant en compte les transports. Il utilise la base de données suisse KBOB. Les consommations énergétiques sont calculées à partir d'une méthode SIA, mais peuvent aussi être calculées à partir d'un plugin permettant de faire de la simulation thermique (type *Honeybee/EnergyPlus*®). La période de référence est paramétrable.

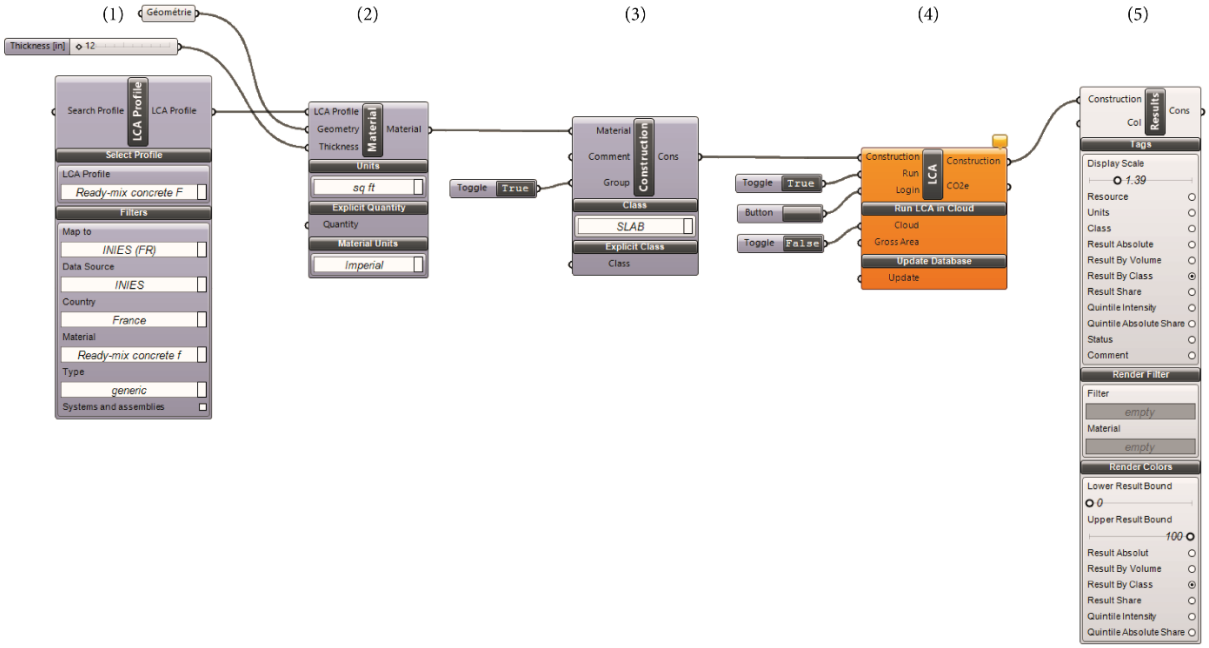


Figure 105: Script Grasshopper pour une ACV avec One Click LCA

Jusqu'ici, aucun de ces plugins ne permettait de faire une ACV avec la base de données Inies et d'intégrer la réglementation française. Depuis septembre 2022, un nouveau plugin permet d'utiliser le logiciel *One Click LCA*® depuis *Grasshopper*®. Il s'appuie sur la plus grande base de données du monde, il est notamment compatible avec la réglementation RE2020. Cinq composants, présentés en Figure 105, sont nécessaires pour créer un modèle avec ce plugin : (1) un premier composant permet de sélectionner une base de données, (2) un second permet

de définir un matériau à partir de la base de données sélectionnée et de la géométrie, (3) un troisième sert à définir des constructions (une pour chaque lot du bâtiment) connecté aux différents matériaux créés, (4) un quatrième composant permet de lancer les calculs, (5) et le cinquième sert à visualiser les résultats.

Les calculs sont effectués en ligne (dans le cloud) et visualisable dans *Grasshopper*®. Il peut tout de même être utilisé avec des algorithmes d'optimisation multicritère. Il a notamment été utilisé avec *Octopus*® (Apellániz et al., s. d.). Le plugin peut être téléchargé et installé gratuitement, mais une licence « expert » est nécessaire pour lancer les calculs et visualiser les résultats.

Au terme de cette dernière partie, il apparaît qu'il existe de nombreux outils opensource dans l'environnement Grasshopper® conçus pour aider les architectes à mieux appréhender les contraintes environnementales, notamment pour l'évaluation de la lumière, de la vue, de l'éblouissement, du confort thermique, de l'ensoleillement, du confort au vent, des besoins et consommations énergétiques et de l'analyse de cycle de vie. La plupart de ces critères peuvent techniquement être intégrés à un processus d'optimisation même si certains demandent un certain niveau d'expertise pour être utilisés et des temps de calcul conséquents pas toujours compatibles avec la pratique professionnelle.

Au terme de ce premier chapitre sur les connaissances issues de la littérature nécessaires à la mise en pratique des méthodes d'ENAPE, il apparaît que le rôle de l'architecte est essentiel pour la transition énergétique et que les logiciels connus pour la prise en compte des performances environnementales ne sont pas utilisés car pas adaptés à ces derniers. Aussi, la prise en compte des contraintes bioclimatiques arrive trop tard dans le processus de conception du projet. Depuis la démocratisation de la programmation visuelle dans les milieux professionnels et académiques, des méthodes d'ENAPE ont été développées pour tenter de reproduire et améliorer le processus itératif qui existe aujourd'hui entre architectes et ingénieurs. Elles se composent informatiquement de trois modèles : un modèle génératif, un modèle d'évaluation et un modèle d'exploration. Cinq types d'outils numériques sont nécessaires à sa mise en œuvre, les outils de modélisation, d'interopérabilité, d'évaluation, d'exploration et de visualisation.

Des applications à l'enveloppe, à la morphologie du bâtiment, sont décrites dans la littérature scientifique, ainsi que des applications à l'échelle urbaine et pour l'agencement des espaces intérieurs, même si ces deux sujets sont plus complexes à mettre en œuvre informatiquement parlant. A l'origine, les paramètres étudiés étaient surtout des paramètres thermiques destinés à l'aide à la décision pour les ingénieurs. Bien que les sujets qui intéressent les architectes soient souvent plus complexes à modéliser, de plus en plus de cas d'étude intègrent des préoccupations architecturales. Cependant, la mise en œuvre de ces méthodes de conception générative et performancielle demande un certain niveau d'expertise, très peu d'entre elles permettent d'intégrer des contraintes, et très peu sont expérimentées dans des contextes professionnels.

Pourtant, le succès est tel auprès des universitaires que de nouveaux plugins issus de la recherche rendant l'évaluation plus accessibles aux architectes sont mis en ligne tous les mois. Aujourd'hui, si l'on pratique la programmation visuelle sur *Grasshopper*®, il est possible pour un architecte de faire des analyses de FLJ, ALJ, d'éclairement utile, de visibilité, de qualité de vue, d'éblouissement, du confort thermique intérieur et extérieur, des apports solaires, du confort au vent, du vecteur d'intensité des précipitations, des besoins et consommations énergétiques, de l'impact carbone des matériaux de construction et des consommations en phase d'exploitation. Ce foisonnement d'outils rend la conception générative et performancielle beaucoup plus accessibles pour les architectes. Il faut désormais la confronter au monde réel.

Chapitre 2 : Les enseignements issus de la mise en pratique dans un contexte professionnel

Dans ce chapitre, nous mobilisons toutes les connaissances issues de la littérature scientifique et de la revue des nouveaux outils numériques à la pratique de la conception architecturale en agence présentés dans le Chapitre 1 pour tenter d'identifier pourquoi il subsiste un écart entre la théorie – pour laquelle ces nouveaux outils sont utiles aux architectes - et la pratique – dans laquelle ils sont peu employés.

Dans un premier temps, nous revenons sur les limites de ces méthodes identifiées dans la littérature et présentons la structure organisationnelle de l'agence Architecture-Studio dans laquelle nous avons expérimenté différentes techniques d'exploration ainsi que la méthode utilisée pour mettre en œuvre des expérimentations. A partir de cette expérience, nous proposons une version actualisée de la liste des difficultés qui peuvent être rencontrées en agence, freinant la mise en pratique des ENAPE.

Dans un deuxième temps, nous présentons les expérimentations portant à nos yeux le plus d'information, certaines ayant abouties à l'usage de méthodes d'exploration et d'autres n'ayant pas abouties aux résultats escomptés.

Dans un troisième temps, nous étudions les différentes techniques permettant de réduire les temps de calcul des simulations en utilisant des algorithmes de *Machine Learning* pour créer ce qu'on appelle des modèles de substitution.

Plusieurs passages des deux premières parties qui composent ce chapitre ont fait l'objet d'une publication (en cours) dans un numéro des Cahiers Ramau.

2.1. La mise en place des expérimentations

Dans cette première partie du chapitre 2, nous proposons une classification des éléments décrits dans la littérature scientifique comme des freins à la mise en pratique de l'optimisation par les architectes.

Nous présentons également l'ensemble des expérimentations conduites chez Architecture-Studio sur des projets réels en cours de conception pour évaluer si ces limites sont toujours d'actualité et, le cas échéant, en identifier d'autres.

2.1.1 Contexte et méthode pour la conduite des expérimentations

Les difficultés relevées dans la littérature scientifique

Une enquête de 2017 conduite dans 165 cabinets d'architecture a révélé que 87 % des répondants souhaiteraient utiliser l'optimisation dans leur activité de conception (Cichocka, 2017). Malgré ces déclarations, ces méthodes sont encore sous-utilisées par les agences d'architecture (S. Li et al., 2020) ; même s'il existe quelques exceptions notables (Haymaker et al., 2018; Shen et al., 2018).

Classification des limites

Avant d'exposer les limites pour une mise en pratique des ENAPE en agence d'architecture identifiées grâce à plusieurs années d'expérimentation chez Architecture-Studio, nous avons d'abord synthétisé les arguments énoncés dans la littérature scientifique.

D'après les revues de littérature sur l'optimisation appliquée à la conception environnementale des bâtiments (Attia, 2011; Nguyen et al., 2014; X. Shi et al., 2016; Zhao & de Angelis, 2018), les raisons de cette sous-utilisation peuvent être multiples. Dans le Tableau 8, nous avons recensé les différents arguments évoqués dans cette littérature. Nous les avons classés en quatre catégories :

(1) **les limites de la catégorie « pédagogie »** sont liées à un manque de compétences et de connaissances dans les agences d'architectures qui freine la mise en place de ces méthodes. La réponse à ces limites est plutôt à chercher du côté des écoles d'architecture, même si la recherche peut conduire à l'élaboration de guides et de manuels techniques, et de méthodes d'enseignement,

(2) **les limites de la catégorie « interface »** qui concernent à la fois la capacité des interfaces homme-machine (IHM) des outils numériques à répondre aux besoins des profils créatifs comme les architectes, mais aussi le manque d'interface entre les différents outils eux-mêmes (génération, évaluation, optimisation et visualisation) qui permettent de faciliter les flux de données et ainsi de rendre l'approche plus fluide,

(3) **les limites de la catégorie « robustesse »** sont celles qui remettent en question la stabilité de la performance de l'approche, notamment la qualité des évaluations et la convergence des algorithmes d'optimisation,

(4) **les limites de la catégorie « technique »** sont celles où la technique informatique n'est pas suffisamment au point pour pouvoir répondre aux besoins des architectes et justifient

la nécessité de poursuivre les recherches en conception computationnelle avant de pouvoir espérer utiliser ces approches dans un contexte professionnel.

Tableau 8: Liste des limites qui freinent la mise en pratique professionnelle des ENAPE, d'après la littérature

Références	**Verrous à la mise en pratique**	**Type de verrous**
(Attia, 2011), (Shi et al., 2016), (Attia, 2011; Z. Shi et al., 2017; Touloupaki & Theodosiou, 2017)	Demande un haut niveau d'expertise, des connaissances logiciels et de programmation, besoin de formation	Pédagogie
(Attia, 2011; Nguyen et al., 2014; Touloupaki & Theodosiou, 2017)	Temps de calculs pas compatibles avec les phases amonts	Technique
(Attia, 2011; Nguyen et al., 2014; Stevanović, 2013)	Manque d'outils d'interopérabilité entre outils d'analyses et d'optimisation	Interface
(Attia, 2011; Zhao & de Angelis, 2018)	La formulation d'un problème est complexe.	Pédagogie et technique
(X. Shi et al., 2016; Touloupaki & Theodosiou, 2017)	Les problèmes d'architecture sont plus complexes que ceux purement thermiques.	Technique
(Attia, 2011; Nguyen et al., 2014)	Incertitude des paramètres de simulation et d'optimisation.	Robustesse
(Attia, 2011)	Absence de méthode systématique.	Pédagogie
(X. Shi et al., 2016)	Manque d'outils de post traitement pour l'analyse des résultats	Interface
(Attia, 2011)	Faible confiance dans les résultats. Manque de retour venant de la pratique.	Robustesse
(X. Shi et al., 2016)	Les résultats d'une optimisation multicritère ne sont pas toujours faciles à interpréter.	Pédagogie
(Nguyen et al., 2014)	Conflits entre paramètres d'évaluation et d'optimisation.	Robustesse
(Nguyen et al., 2014)	Les plateformes d'optimisation (type Matlab) ne sont pas adaptées aux architectes.	Interface

<u>Un manque de pédagogie</u>

Les années s'étant écoulées depuis la publication de ces différentes revues, les choses ont depuis beaucoup évoluées. Concernant les arguments de la catégorie « pédagogie », on a pu énoncer précédemment que de plus en plus de chercheurs s'intéressent à ces approches avec un œil

d'architecte (voir p.73) mais aussi que de plus en plus d'applications dans la littérature intègrent des préoccupations architecturales (voir section sur les applications théoriques p. 69), et enfin que les outils de simulation à disposition des architectes nécessitent moins qu'auparavant de connaissances en physique fondamentale (voir le section sur la bibliothèque d'outils d'évaluation). De plus, ces approches sont depuis quelques années enseignées dans certaines écoles d'architecture. Quelques écoles, notamment anglo-saxonnes, proposent même des masters dédiés à ces nouvelles méthodes de conception. Le Tableau 9 donne une liste non exhaustive des masters en conception digitale ouverts aux étudiants en Architecture qui existent aujourd'hui. Ces formations durent entre un et deux ans. Il s'agit parfois de master en architecture (March), de master en science (MSc), de master de recherche (Mres) ou de post-master (PM). Même si les choses semblent avoir progressé, une grande majorité des étudiants en architecture aujourd'hui ne sont pas formés à ces technologies, notamment en France où l'enseignement de l'Architecture, issu des Beaux-Arts, peut-être très éloigné des méthodologies scientifiques et technologiques. Aussi, la formation en conception générative implique un prérequis, celui de maitriser la conception paramétrique, or même si elle est enseignée, elle reste très peu pratiquée dans la majorité des agences d'architecture en dehors des plus grosses entreprises (Stals et al., 2022).

Tableau 9 : Liste des masters en conception digitale pour l'architecture

Ecole	**Pays**	**Type**	**Nom de la formation**	**Durée**
Architectural Association School of Architecture (AA school)	RU	March MSc	Emergent Technologies and Design	12-16
Universitat Politecnica de catalunya	Espagne	PM	Parametric design in architecture	12
Welsh School of Architecture	RU	MSc	Computational Methods in Architecture	12
The Bartlett School of Architecture	RU	MSc Mres	Architectural Computation	12
Carnegie Mellon University	USA	PM	Advanced Architectural Design	24
Weitzman School of Design	USA	MSc	Advanced Architectural Design	18
Dr. Bhanuben Nanavati College of Architecture for Women	Inde	MArch	Digital Architecture	24
Nottingham Trent University	RU	MSc	Digital Architecture and Construction	12
NMIMS Balwant Sheth School of Architecture	Inde	March	Advance Architecture Design	24
University of Kent	RU	MSc	Bio-Digital Architecture	12
Ecole des ponts et chaussées	France	PM	Digital Building Design	12

Un manque d'interfaces

Les arguments concernant les sujets d'interface ne nous paraissent plus d'actualité. En effet, concernant l'environnement Rhinocéros/Grasshopper beaucoup de progrès ont été réalisés pour gérer les flux de données entre les outils de simulation et de modélisation comme nous avons pu le décrire dans la section sur la bibliothèque d'outils d'évaluation (p.116) et pour faciliter la visualisation des résultats comme nous l'avons vu dans la sous-section sur Les outils de data visualisation (p63). Enfin, nous verrons dans le prochain chapitre, que des progrès importants ont aussi été réalisés pour les algorithmes d'optimisation.

Un manque de robustesse

Concernant le sujet de la robustesse de la méthode, et notamment la question du manque de retours sur des expérimentations issues de la pratique, il existe désormais quelques exemples de mise en pratique dans un contexte professionnel notables (Haymaker et al., 2018; Shen et al., 2018). En 2020, Marcelo Bernal et ses co-auteurs comparent ce qu'ils appellent les « *approches intuitives* » (méthodes traditionnelles des architectes) et « *approches systématiques* » (exploration numérique de solution) dans un contexte professionnel (agence Perkins + Will) et montrent une amélioration significative et constante des performances des projets après l'utilisation des approches systématiques (Bernal et al., 2020). Les explorations sont réalisées dans l'environnement Grasshopper® avec des outils similaires à ceux que nous utilisons dans cette thèse. Pour l'exploration, des algorithmes de *Machine Learning* ont été utilisés (et non pas des algorithmes génétiques). Les équipes de conception étaient constituées d'un architecte, d'un chef de projet et d'un ou deux ingénieurs avec au moins 20 ans d'expériences et d'autres membres (architectes et dessinateurs) moins expérimentés.

L'étude est conduite sur trois cas où l'objectif était de maximiser l'éclairage naturel et de minimiser les besoins énergétiques. Il s'agissait d'une enveloppe pour un lycée, d'une empreinte d'un bâtiment pour des laboratoires et de la façade d'un bâtiment résidentiel. Les solutions proposées par les équipes de conception et les solutions optimisées ont ensuite été comparées sur des critères de performance environnementale. Pour chaque cas d'étude, les chercheurs ont récupéré la solution proposée par les équipes de maîtrise d'œuvre, les ont analysées et ont défini un espace de solution à partir de cette première esquisse. Comme le montrent les résultats présentés en Figure 106, les solutions ont toutes été améliorées.

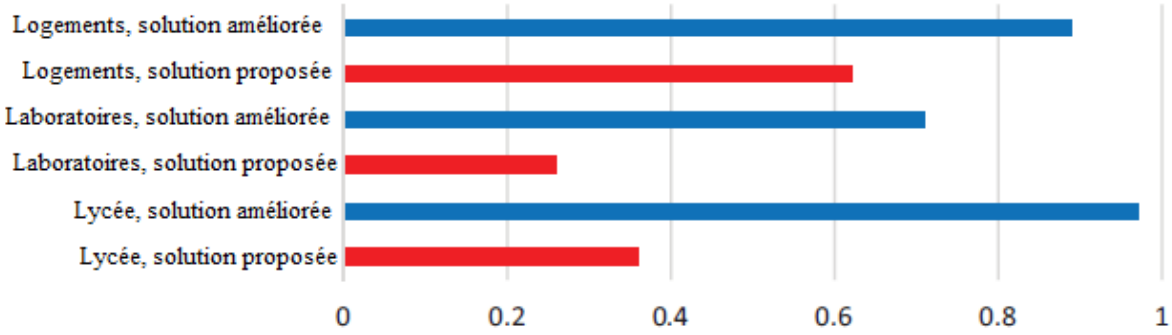


Figure 106 : Résultats de l'étude comparative de (Bernal et al., 2020)

Des limites techniques

Il reste encore un état des lieux à réaliser pour les limites techniques. La limite la plus fréquemment citée est celle des temps de calculs qui ne seraient pas adaptés aux phases amonts de conception. De nombreux travaux ont déjà été réalisés pour tenter de trouver des solutions à ce problème, l'utilisation de modèles de substitution semble être une réponse prometteuse même s'ils restent encore peu utilisés, nous consacrons une section entière à ce sujet à la fin de ce chapitre.

Description du terrain : Architecture-Studio

Cette thèse a été réalisée en collaboration avec l'agence d'architecture, d'urbanisme et de design d'intérieur Architecture studio (AS). Le temps consacré à la recherche pendant les trois années du doctorat a été réparti entre du temps d'expérimentation et de développement en agence, et du temps de recherche plus théorique au laboratoire.

Créée en 1973 par Martin Robain, qui sera rapidement rejoint par Jean-François Galmiche et Rodo Tisnado, AS est une agence renommée, implantée dans un ancien atelier du centre de Paris, avec des bureaux à Shanghai et à Zug. Parmi ses réalisations notables se trouvent l'Institut du monde arabe à Paris (1987), le bâtiment Louise-Weiss du Parlement européen de Strasbourg (1999), le plan directeur de l'Exposition universelle de Shanghai (2010), l'aménagement urbain du Fort d'Issy-les-Moulineaux (2013), Le Grand auditorium de la Maison de la Radio et de la Musique (2014), les bureaux "Summers" à Buenos Aires (2019), ou encore le CHU de Tanger (2021). L'agence a reçu de nombreux prix qui récompensent ses réalisations, notamment l'Equerre d'argent en 1987, la Médaille d'or Interarch en 2012 et le Grand Prix AFEX en 2020.

La gouvernance collégiale d'AS

Connue pour son mode de gouvernance particulier, elle compte aujourd'hui 13 associés qui sont des directeurs d'agence, et 21 partenaires qui sont pour la plupart des chefs de projet expérimentés qui constituent le noyau stable de l'agence. Au total, AS rassemble 150 personnes de 20 nationalités différentes autour de projets d'architecture et d'urbanisme du monde entier. Les dirigeants y pratiquent une gouvernance collégiale à laquelle ils tiennent et qu'ils décrivent comme faisant partie de l'ADN de l'agence. Bryan Lawson pourrait qualifier l'organisation de leur pratique de « *semi-autonome et fédérée* » (Lawson, 2006) ; chaque associé est responsable de l'avancement de ses propres projets, mais le collectif décide, s'entraide, partage et confronte les idées. Tous les projets sont collectivement signés. Ces moments de collaboration sont institutionalisés dans des réunions hebdomadaires appelées « *revues de projet* ». Il arrive que certains associés travaillent en binôme sur certains projets, et se remplacent lors de congés. Certains associés ont développé des expertises sur différents types de programmes spécifiques (hôpitaux, prisons, urbanisme…), et d'autres pour différentes régions du monde (Suisse, Moyen-Orient, Amérique latine, Afrique, Asie…).

Les équipes de projet, les équipes supports (pôle BIM, infographie, R&D, communication) et tous les frais de fonctionnement sont partagés. Les équipes de projet sont de tailles variables. Elles sont généralement *a minima* constituées d'un chef de projet (partenaire lorsqu'il s'agit d'un grand projet), auquel peut s'ajouter un ou plusieurs assistants de projet et / ou un ou plusieurs stagiaires. La taille des équipes est très variable au cours de la vie d'un projet. Les collaborateurs sont amenés à changer régulièrement de projet s'étant spécialisés au fil du temps sur certaines phases (concours, conception ou chantier), mais les architectes ayant eu l'occasion de travailler avec l'ensemble des associés restent rares. L'ensemble des collaborateurs travaille à la vue de tous sur des mezzanines éclairées par une grande verrière. Les échanges s'y font à haute voix et les informations se partagent le plus souvent lors de réunion informelle. Les espaces de travail de l'agence ont été conçus pour favoriser cette méthode de travail collaborative qui a même su résister à la crise du COVID.

Les outils de conception d'AS

Au fil des années, les associés ont développé une méthode de travail commune qui facilite la collaboration et qui a fait leur réputation. Le « *tracé rouge* » est ce qui fait la marque de fabrique d'AS, dont la meilleure description reste celle qu'en font les associés *:*

« Concrètement, nous sommes réunis autour d'un grand calque et apportons tous notre touche au dessin avec – la précision a son importance – des stylos de marque Tempo. Ceux-ci ont l'avantage d'être à la fois suffisamment précis et imprécis pour dessiner à grands traits les intentions d'un projet tout en étant à l'échelle. Grâce à leurs différentes couleurs (dont chacune correspond à la nature de l'élément tracé), nous pouvons superposer des dessins. Ainsi émerge un palimpseste d'éléments techniques et architecturaux. Le résultat est ensuite figé en un tracé rouge formé de milliers de traits, plus que nécessaire. Tous ceux qui travailleront à partir de ce dessin choisiront des traits, mais ne pourront pas en ajouter. C'est donc à la fois un dessin ouvert et fermé, une ouverture des possibles dans la rigueur. Nous réalisons le tracé rouge en plan et en coupe, selon une logique qui prédétermine l'esthétique ou le principe du projet. Le tracé rouge est une radiographie de tous les éléments qui composent un bâtiment et y jouent un rôle structurant : enveloppe, périmètre, poteaux, escaliers, gaines, etc. Il positionne tous les éléments invariants du projet, niveau par niveau, auxquels on peut ajouter les éléments variables. […] La méthode du tracé rouge nous a aidés à opérer notre mutation vers le dessin informatique au début des années 1990. Ces nouveaux outils nécessitent en effet de constituer une référence externe, à savoir un fichier matrice à partir duquel on peut dessiner tous les niveaux du projet. C'était déjà, en quelque sorte, le principe du tracé rouge.» (Robain & AYACHE, 2013)

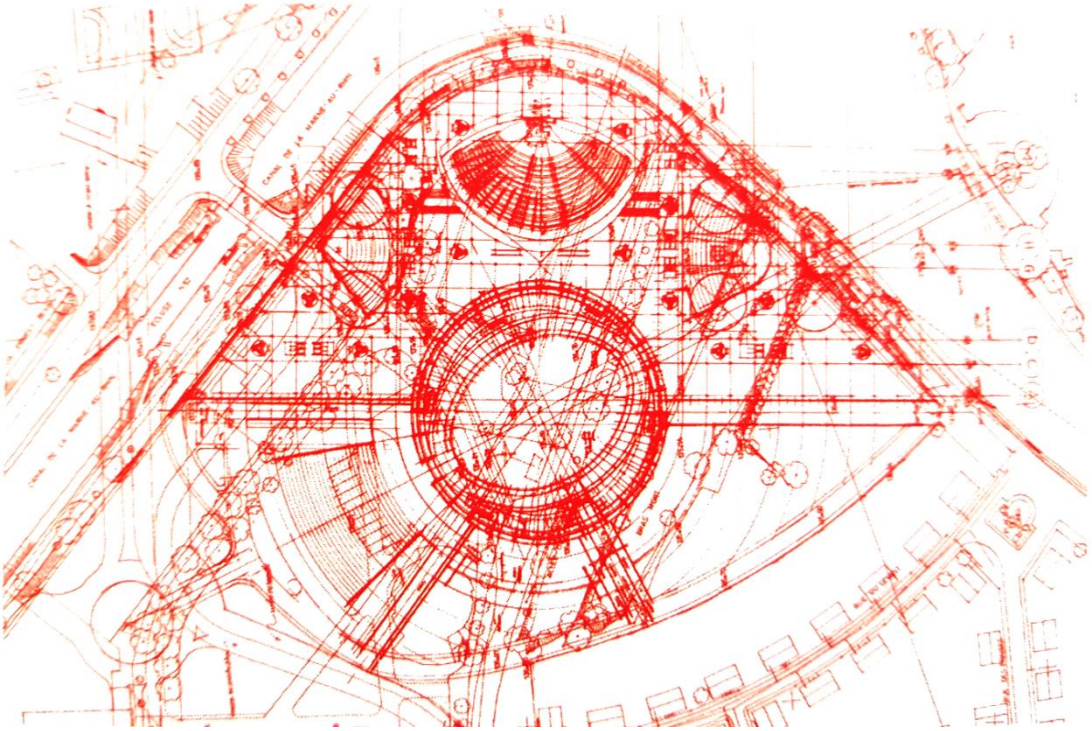


Figure 107 : Tracé rouge du grand auditorium de la maison de la radio (source: Architecture Studio)

Aujourd'hui, le Tracé Rouge est le seul outil de conception et de représentation véritablement partagé par toutes les générations qui travaillent chez Architecture-Studio. L'évolution rapide de la conception architecturale assistée par ordinateur (CAAO) a créée des barrières entre les différentes générations dans les agences d'architecture auxquelles Architecture-Studio n'a pas échappée. Les doyens de l'agence ont été formés au dessin à la main, les jeunes diplômés sont à l'aise avec les logiciels de la modélisation BIM ou la modélisation paramétrique, et entre les deux générations, les architectes, aujourd'hui souvent devenus chef de projet, ont été formés aux logiciels de CAO en 2D et ont appris par la pratique la modélisation 3D et ses outils afférents.

Ainsi, il n'est pas rare qu'un projet soit dessiné trois fois, notamment lors des concours. Dans un premier temps, les intentions du projet sont dessinées à la main avec la méthode du tracé rouge. Puis les plans et les coupes sont redessinés et précisés à l'aide du logiciel Autocad® dans lequel un scan du Tracé Rouge est importé et où les surfaces exactes et leurs adéquations avec les exigences du programme peuvent être vérifiées une première fois. Cet outil est aussi souvent utilisé pour le rendu des pièces graphiques de dessins d'architectures (plans, coupes, élévations). Enfin, une modélisation 3D, appelée « *maquette 3D* » est réalisée et liée au Tracé Rouge préalablement vectorisé dans Autocad®. En phase concours, elle sert principalement à la production d'images de synthèse photoréalistes appelées « *perspectives* ». Elle peut être réalisée dans un logiciel CAO comme Rhinocéros® ou, si le chef de projet le souhaite, ou si la maîtrise d'ouvrage l'exige, elle peut être directement réalisée sur un logiciel BIM comme Revit®, on parle alors chez Architecture-Studio de « *maquette BIM* » qui permettra, notamment dans les phases suivantes, de faciliter les modifications et les échanges avec les collaborateurs externes. A cette phase, les logiciels utilisés sont donc plus des outils de représentation que des outils de conception.

En 2019, l'usage d'outils de conception numérique en phases amonts est donc encore rare chez Architecture-Studio. Certains outils ont été développés pour assister l'architecte dans des phases spécifiques comme un outil Dynamo© qui permet de générer automatiquement des familles Revit d'objet 3D parallélépipédiques utilisés pour la production d'organigrammes permettant de mieux appréhender la complexité d'un programme hospitalier. Les collaborateurs ne pratiquent pas la simulation environnementale que ce soit pour l'éclairage naturel, la thermique, la qualité des vues ou l'étude des vents, qui sont sous-traitées à des bureaux d'études. Seuls des petits outils permettant de faire des études d'ombrage ou d'afficher des héliodons sont parfois utilisés par les architectes qui les connaissent.

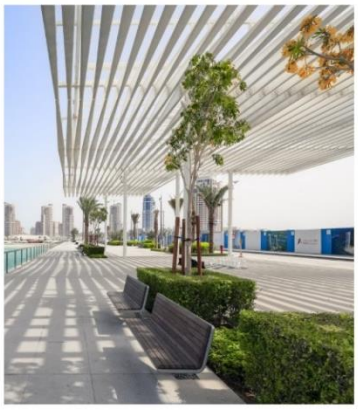
Ilôt Effeil, Créteil, France

Mosquée du roi Abdallah, La Mecque, Arabie Saoudite

La cathédrale de Créteil, France

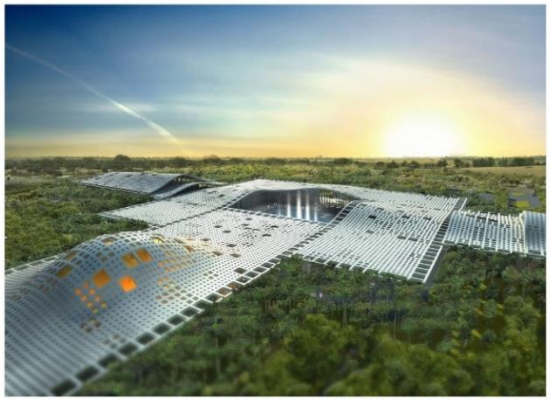
Tour résidentiel M1 62, Dubaï, Emirats

Immeuble Om, Issy-les_Moulineaux, France



Espaces publics de Lusail, Qatar

James'Unnas Complex, Damman, Arabie Saoudite



Centre culturel d'Oman, Mascate, Oman

Stade Arena 92, Nanterre, France

Institut Faire Face, Amiens, France

Figure 108 : La courbe dans les projets d'Architecture Studio (source: Architecture Studio)

Les équipes de projet préfèrent utiliser des outils de rendu photoréaliste pour analyser la qualité de leur projet, même si les perspectives finales ne sont pas réalisées par un membre de l'équipe, puisqu'elles sont soit sous-traitées, soit réalisées par le pôle spécialisé de l'agence.

La pratique de la programmation visuelle d'AS

La programmation visuelle, notamment avec Grasshopper®, est régulièrement utilisée chez Architecture-Studio, notamment pour la conception des enveloppes. Avant cette thèse, son usage était pratiquement exclusivement réservé à la recherche de formes. Nous avons identifié deux catégories usages :

(1) **La maîtrise des formes courbes**, qui peuvent être de simple ou double courbure appliquées à la forme du bâtiment, à des éléments de façade ou à la trame d'un plan d'urbanisme. Cet usage n'est pas systématique dans la pratique de l'agence car ce type de projets, souvent difficiles à construire, sont coûteux. La Figure 108 présente la dizaine de projet d'Architecture-Studio où l'on retrouve l'emploi de formes courbes.

(2) **Les « jeux combinatoires »** pour la conception des enveloppes. Cet usage est fréquent dans les projets d'Architecture-Studio. La modélisation paramétrique est utilisée pour générer, dans un espace contrôlé, de la complexité en jouant sur la disposition d'une multitude d'éléments architecturaux. Ici, la complexité vient de l'assemblage des éléments architecturaux et non de leur forme. Une sélection de douze projets utilisant ces jeux combinatoires est présentée en Figure 109.

Un cas particulier existe. Avant de débuter ce travail de doctorat, un premier projet, aujourd'hui construit et récompensé (AFEX 2020), que sont les bureaux Summers à Buenos Aires, a fait l'objet d'une ENAPE. Si ce projet est typique de la catégorie des jeux combinatoires, il est le seul à avoir fait l'objet d'une optimisation de son système de protection solaire. Des lames verticales vitrées et sérigraphiés sont utilisées pour protéger la façade d'un bâtiment courbe. L'orientation des lames a fait l'objet d'une optimisation monocritère à l'aide de l'algorithme génétique du solveur Galapagos. La méthode de la géométrie informée a ensuite été utilisée pour concevoir la sérigraphie et proposer différents types de lames plus ou moins translucides en fonction des apports solaires. Une photographie de la façade construite est présentée en Figure 109.

Bureaux Summers, Buenos Aires, Argentine

Folie Richeter, Montpellier, France

Tour Abdali Gates, Amman, Jordanie

Projet connexe à la gare du GPE, Créteil, France

CHU Tanger, Maroc

Siège Caisse d'Epargne, Bordeaux, France

Centre Culturel de Jinan, Jinan, Chine

Complexe Culturel de Zhangjiakou, Chine

Cité scolaire internationale, Marseille, France

Complexe Akwaba, Abidjan, Côte d'Ivoire

Campus de Bordeaux, Secteur Biologie Santé, France

Musée des sciences naturelle du Tibet, Lhasa, Chine

Figure 109: Les jeux combinatoires dans les projets d'Architecture Studio

Dans la pratique, pour modéliser des projets de la première ou de la seconde catégorie, les collaborateurs ne sont pas contraints par leur hiérarchie à utiliser la programmation visuelle. Le choix de passer ou non par un script est laissé à celui qui a la charge de la modélisation. Celle-ci sera simplement beaucoup plus fastidieuse et chronophage pour celui qui ne maîtrise pas la programmation visuelle. Les collaborateurs qui la pratiquent sont peu nombreux et ne font pas partie du noyau dur de l'agence Architecture-Studio (associés et partenaires). Cette compétence peu répandue ne fait pas partie des exigences pour le recrutement.

Parmi les collaborateurs qui pratiquent le développement de scripts, nous avons pu distinguer trois types de développeurs :

(1) **Les développeurs « experts »** peuvent créer des fonctions personnalisées (appelées composants ou nœuds selon qu'il s'agisse de Grasshopper® ou de Dynamo®) car ils maîtrisent un ou plusieurs langages de programmation informatique (python, C#, VBA…). Actuellement, il n'y a pas de collaborateur architecte à ce niveau chez Architecture-Studio. Lorsqu'un architecte a ce type de compétence, il s'agit très souvent d'un autodidacte, car la programmation informatique n'est pas enseignée dans les écoles d'architecture.

(2) **Les développeurs « confirmés »** maîtrisent parfaitement les fonctions de gestion des données. Ils connaissent les fonctions de base des plateformes comme Grasshopper® ou Dynamo® et utilisent des *plugins* (autrement appelés *packages* ou *librairies*) pour traiter des problèmes spécifiques (génération de panneaux, motifs islamiques, origami, optimisation…). Selon les périodes, l'agence peut compter entre 1 et 3 collaborateurs au maximum à ce niveau.

(3) **Les développeurs « débutants »** qui ont déjà pu assister à une formation en programmation visuelle, peuvent élaborer des scripts simples tant que les arborescences des listes de données restent restreintes à une branche. Ils sont capables d'utiliser les scripts développés par les développeurs de niveau expert ou confirmé, et de les modifier à la marge. Leurs connaissances en programmation sont suffisantes pour utiliser les *plugins* permettant de créer des interfaces entre les outils de simulation et les modeleurs 3D présentés dans la section 3 du chapitre 1 (p 116), mais pas suffisantes pour créer un modèle paramétrique permettant de faire des jeux combinatoires en façade par exemple. Le nombre de développeurs débutants est très variable chez Architecture-Studio. Des formations sont proposées par l'agence aux collaborateurs les plus expérimentés depuis l'année 2020.

Parmi les collaborateurs ne pratiquant pas le développement de scripts, nous distinguons de nouveau trois niveaux de connaissances :

(1) **Les « utilisateurs »,** ce sont les collaborateurs capables d'utiliser un script. Ils ne peuvent pas le modifier eux-mêmes, mais sont en mesure de jouer avec les différents paramètres après une brève présentation de l'outil. Ils sont aussi capables de formuler une demande précise de script, ou de formuler une demande précise de modification de script. Certains partenaires et chefs de projet chez Architecture-Studio sont des utilisateurs de scripts.

(2) **Les « initiés »**, ce sont les collaborateurs qui ne savent pas utiliser les scripts mais qui comprennent la logique paramétrique, ses atouts et ses limites. Ils sont capables d'identifier une situation où l'usage de la programmation visuelle pourrait être utile. On trouve des initiés à tous les échelons de l'échelle hiérarchique chez Architecture-Studio.

(3) **Les « novices »**, ce sont les collaborateurs qui ne connaissent pas ou très peu de chose sur la conception paramétrique. Il en existe à tous les échelons de l'échelle hiérarchique et pour toutes les générations chez Architecture-Studio.

L'hétérogénéité, et le manque de stabilité dans le niveau de connaissance et d'expérience de la pratique de la programmation visuelle chez Architecture-Studio a nécessairement eu un impact sur les expérimentations d'ENAPE, à commencer par la méthode choisie pour leur mise en œuvre.

Méthodologie et objectifs

Les trois axes de la thèse

En parallèle de l'état de l'art sur les applications des ENAPE dans la littérature scientifique, nous avons rapidement démarré les expérimentations sur des projets réels en cours de conception dans un contexte professionnel, celui de l'agence Architecture-Studio. Pour cela, nous avons utilisé une méthodologie de recherche que l'on peut qualifier de « recherche-action » (Avison et al., 1999). A l'origine issu des sciences sociales, ce type de recherche vise à transformer la réalité pour produire de la connaissance concernant ces transformations. Ici, nous cherchons à modifier le processus de conception d'un projet d'architecture en phase amont en utilisant de nouveaux outils mathématiques et informatiques afin de mieux comprendre leurs

atouts et leurs limites, mais surtout afin de mesurer leur compatibilité avec la méthode de conception des architectes.

Notre méthode de recherche s'est organisée autour de trois axes présentés en Figure 110. Il s'agit d'un processus itératif entre (1) un axe d'expérimentations des ENAPE dans la pratique, (2) un axe de veille et de développement d'outils d'analyse, et (3) un axe de recherche plus théorique.

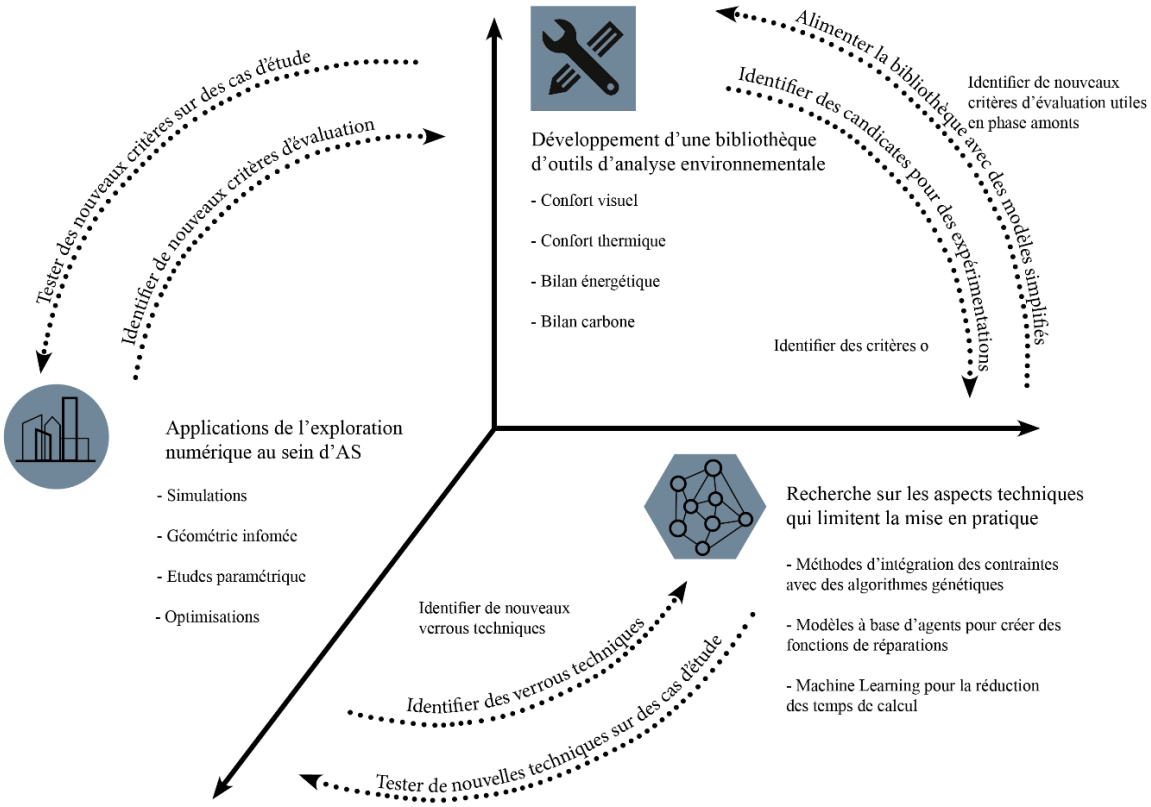


Figure 110 : Les trois axes de recherche de la thèse

Les expérimentations dans la pratique et la collaboration avec les équipes de projet (1) ont permis d'identifier le type et la nature des critères de performances ainsi que des contraintes rencontrées dans les problèmes de conception. Au fil des itérations, nous avons pu constituer la bibliothèque d'outils d'analyse (2) présentée à la partie 3 du chapitre 1 (p 116). Ces outils ont alors pu être testés lors de nouvelles expérimentations (1) et faire émerger des limites techniques (comme l'intégration des contraintes ou les temps de calcul) qui ont ensuite été étudiées sur les temps de recherche (3). Ces travaux ont permis d'identifier dans la littérature scientifique différentes solutions avant d'en faire l'objet d'études comparatives (3) présentées aux chapitres 3 et 4. Les solutions ont ainsi été testées sur des cas d'étude issus des expérimentations notables

présentés dans la section 2 de ce chapitre (p194) ou sur d'anciens projets de l'agence. Elles ont ensuite pu être de nouveau testées sur des projets réels (1).

Les objectifs de ces expérimentations n'étaient pas uniquement axés sur l'identification de verrous techniques. Nous souhaitions aussi comprendre pourquoi l'ENAPE semble être si difficile à intégrer dans le processus de conception des architectes, et évaluer si elle permet réellement d'aider à la décision. Il semblait aussi important de chercher à comprendre si ces méthodes peuvent être complémentaires aux interventions des bureaux d'études d'ingénieurs en environnement, et s'il est réaliste d'imaginer que des équipes de projet constituées d'architectes puissent un jour être autonomes sur ce type d'outils.

Organisation des expérimentations

Pour mettre en œuvre une ENAPE, il faut nécessairement être un développeur Grasshopper® confirmé, avoir été initié à l'optimisation multicritère et avoir des notions de conception bioclimatique. Même après avoir formé les quelques développeurs confirmés présents chez AS à l'optimisation, nous n'avons jamais réussi à faire expérimenter son usage par un membre d'une équipe de projet. Aussi, nous avons-nous même mis en œuvre l'ensemble des expérimentations.

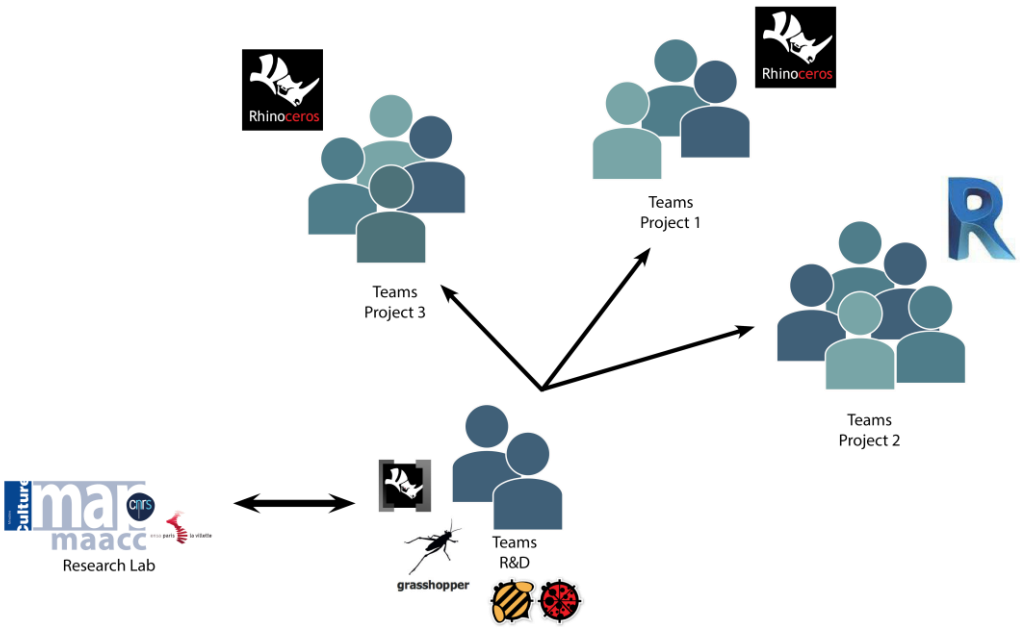


Figure 111 : Schéma d'organisation de la R&D d'Architecture Studio

Affiliés pendant quatre années au pôle R&D, nous avons eu chez AS un rôle transverse. Présentés aux équipes de projet comme experts de la conception paramétrique et

environnementale, nous étions à leur disposition pour intervenir lors des phases de conception et quelques fois même de chantier. L'avantage de ce mode d'organisation illustré en Figure 111, est que nos interventions ont toujours été réalisées en parallèle des projets pour ne pas gêner leur avancement, mais aussi que nous avons pu nous confronter à une grande quantité de problèmes de conception. Parmi les inconvénients nous avons observé que n'étant pas identifiés comme membres à part entière de l'équipe de projet, nous avons souvent été oubliés lors des réunions avec les ingénieurs des bureaux d'études, et qu'il était difficile d'obtenir une vision globale des projets. Ces deux sujets n'ont pas toujours facilité la formulation des problèmes d'optimisation.

Pour AS, l'accueil d'un doctorant était une première, aucun collaborateur de l'agence Architecture-Studio n'avait réellement travaillé avec un chercheur auparavant dans un tel cadre. Ainsi, les méthodes d'ENAPE ont fait l'objet de plusieurs présentations aux associés, aux partenaires et aux collaborateurs tout au long de la thèse pour expliquer les objectifs de cette recherche-action, afin que les équipes de projet puissent identifier sur quels problèmes nous solliciter.

Dès le départ, il a été convenu que nous serions sollicités prioritairement sur des problèmes de conception d'enveloppe pour plusieurs raisons :

(1) nous avons vu qu'il préexistait à ces travaux de recherche une culture de la modélisation paramétrique pour les façades chez AS,

(2) nous savons depuis la partie 1 du chapitre 1 (p41) que les enveloppes ont un fort impact sur la performance environnementale des bâtiments,

(3) nous savons depuis la partie 2 du chapitre 1 (p79) que ce type de problème est plus simple à modéliser que d'autres (génération de plans urbains et d'agencements des espaces) qui demandent des techniques génératives spécifiques,

(4) nous avons ressenti une réticence de la part des collaborateurs à déléguer l'exploration de la forme urbaine et l'agencement des espaces à un « *ordinateur* ».

Finalement, nous avons pu réaliser des explorations numériques pour la conception d'enveloppes pour tous les types de programmes (bureaux, logements, équipements), tous les types d'affaires (privé, concours internationaux, partenariats publics privés), pour des projets localisés dans tous les types de climats et à différentes phases du projet.

L'ensemble des documents produits sur chaque projet a été conservé pour pouvoir être réexploité pendant le temps consacré à la recherche plus théorique. Chaque mois, une à deux interventions sur les projets ont été réalisées en moyenne. Un cahier de recherche a été tenu et

alimenté mensuellement avec des descriptions des problèmes rencontrés, des méthodes testées, de la synthèse des résultats d'exploration et des notes sur les difficultés rencontrées qu'elles soient techniques ou autres. Lorsque la période consacrée aux expérimentations a pris fin, nous avons utilisé les archives de l'agence pour chaque projet ayant donné lieu à une expérimentation afin de pouvoir constater les éventuelles évolutions.

2.1.2 Enseignements généraux issues des expérimentations

De la conception traditionnelle à la conception computationnelle

Comme cela était attendu, nous n'avons pas pu expérimenter l'usage d'algorithmes d'optimisation multicritère de façon systématique à chaque sollicitation du jour au lendemain. Les architectes nous sollicitaient lorsqu'ils avaient un problème de conception, et selon la précision de la question posée, l'état d'avancement du projet, le temps imparti, les contraintes du projet, les méthodes de travail habituelles des collaborateurs et leurs méthodes de management, différentes méthodes ont été testées, soit dans le temps imparti du projet (axe 1), soit plus tard dans un temps moins contraint en laboratoire (axe 3). Nous présentons dans cette section une classification de ces méthodes de la plus traditionnelle à la plus « *computationnelle* ». Nous parlons ainsi de niveaux, il y en a 8 niveaux en tout, allant de 0 à 7. Plus le niveau est élevé plus il demande de compétences techniques.

<u>La méthode de conception traditionnelle</u>

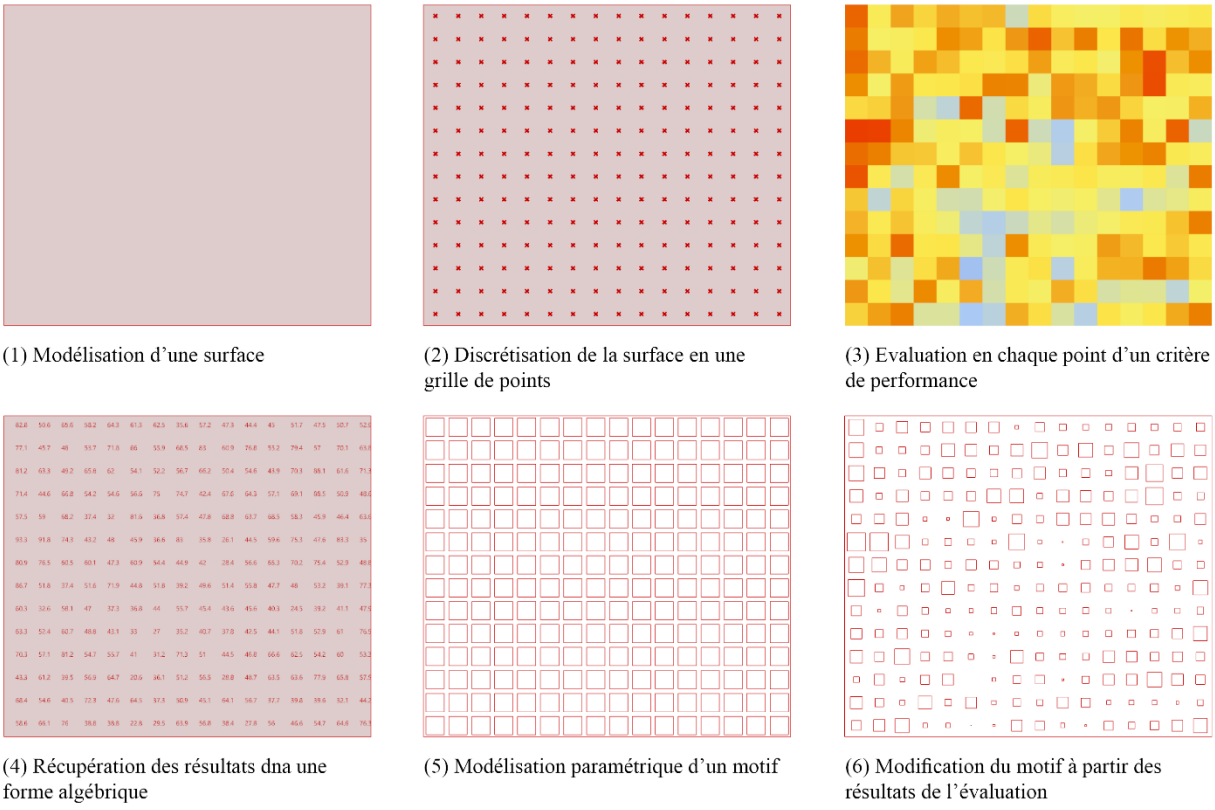
Le niveau 0 correspond au mode de conception traditionnel auquel sont habitués les collaborateurs de l'agence, c'est-à-dire où l'architecte produit la maquette 3D et une autre personne utilise la simulation pour évaluer l'objet modélisé (traditionnellement un ingénieur). Dans le cas des expérimentations, nous jouons le rôle de l'ingénieur. Au début des expérimentations, les habitudes étant difficiles à changer, nous étions principalement sollicités pour faire de l'analyse. Par la suite, une fois la définition d'une exploration numérique plus claire pour les équipes de projet, il arrivait tout de même que nous ne soyons pas en mesure de faire de l'exploration mais seulement une ou plusieurs analyses d'une même solution. C'était notamment le cas lorsque les équipes nous sollicitaient un peu trop tard dans le processus de conception, ou lorsqu'il était question d'une simulation pour vérifier le respect de la réglementation.

L’exploration « *manuelle* »

Le niveau 1 est une méthode d’exploration qualifiée de « *semi-automatique* » où les itérations se font encore « manuellement ». La modélisation et la simulation sont gérées sur la même machine et par le même utilisateur sur une interface unique. Les variantes du projet sont facilement générées à l'aide d'un modèle paramétrique couplé à un moteur d'évaluation. Le nombre d'itérations entre la modélisation et l'évaluation peut alors être beaucoup plus élevé que dans la méthode traditionnelle. Cette technique est appropriée lorsqu'il existe des contraintes esthétiques majeures difficiles à intégrer dans le modèle, et un vaste espace de solutions dans lequel le concepteur peut facilement évoluer de manière empirique. Elle peut aussi être utilisée pendant l’étape de formulation du problème où on cherche à déterminer les paramètres qui ont le plus d’impact sur les critères de performances étudiés. Cette méthode est accessible à tous les développeurs de scripts, y compris les débutants.

La méthode de la « *géométrie informée* »

Dans cette variante de l’exploration manuelle, une analyse précède la modélisation paramétrique. Cette technique identifiée grâce à la littérature scientifique présentée dans la section 2 du chapitre 1 se modélise en 6 étapes illustrées à l’aide de la Figure 112.



(1) Modélisation d’une surface

(2) Discrétisation de la surface en une grille de points

(3) Evaluation en chaque point d’un critère de performance

(4) Récupération des résultats dna une forme algébrique

(5) Modélisation paramétrique d’un motif

(6) Modification du motif à partir des résultats de l’évaluation

Figure 112 : Les étapes d'un modèle paramétrique avec de la géométrie informée

Une fois le modèle paramétrique achevé, il est toujours possible d'explorer ses différents paramètres en le reliant à un second moteur d'évaluation qui permettra de vérifier l'efficacité du motif généré. Cette méthode est très utile, notamment pour le contrôle des apports solaires lorsqu'il est le critère le plus contraignant d'un problème d'architecture et que l'exposition de la façade aux rayonnements solaires est particulièrement hétérogène. Cette méthode peut se suffire à elle-même ou être intégrée à un processus d'optimisation. C'est notamment ce qu'ont proposé les chercheurs Showkatbakhsh & Kaviani (2021) avec leur méthode générative « *homéostatique* » pour la rétroconception de deux tours Al-Bahr à Abou Dabi aux Emirats Arabes Unis.

L'étude paramétrique

Le niveau 3 n'implique toujours pas l'utilisation d'un algorithme d'optimisation, mais l'exploration est automatisée à l'aide d'une boucle de calcul (à l'aide du plugin TT Toolbox sur Grasshopper®). Soit l'espace de solution est suffisamment petit pour qu'on puisse analyser l'ensemble des solutions qui le compose, on parle alors d'optimisation par force brute, ou alors on utilise une fonction aléatoire pour ne sélectionner qu'une partie des solutions. Cette méthode est efficace sur des problèmes simples, avec un ou deux paramètres, comme l'orientation de lames ou le dimensionnement d'un détail de façade, seulement si les critères à évaluer peuvent être analysés rapidement. Elle peut aussi servir à faire une analyse de sensibilité pour faciliter la formulation de problèmes plus complexes. Enfin, cette méthode est utilisée pour générer des bases de données pour l'apprentissage d'algorithme de *Machine Learning* pour la création de modèle de substitution.

L'optimisation monocritère

Le niveau 4 est utilisé lorsque l'espace de solutions est très grand ou lorsque les simulations sont chronophages, comme dans le cas des STD ou de la CFD, et que l'on ne dispose pas de modèle de substitution adapté. Dans ce cas, il n'est pas possible d'analyser toutes les solutions, du moins avec la puissance des ordinateurs actuels et en respectant l'agenda imposé par les maîtrises d'ouvrage. Ainsi, les algorithmes d'optimisation permettent de trouver les meilleures solutions sans avoir à toutes les analyser. Les problèmes à critère unique sont rares dans la pratique, et les problèmes formulés par les architectes sont souvent faussement monocritère. Le dimensionnement d'un élément de façade pour la protection solaire par exemple, implique souvent au moins deux critères, soit les apports d'été vs les apports d'hiver, soit les apports annuelles vs la quantité de matière, sans quoi la réponse de l'algorithme risque

de ressembler à « *la solution la plus efficace est de fermer les volets* ». Cette méthode demande une bonne maîtrise de la programmation visuelle, elle est plutôt réservée aux développeurs confirmés, comme pour le niveau 2 et 3.

L'optimisation multicritère

Le niveau 5 est nécessaire lorsque plusieurs critères doivent être évalués simultanément, ce qui est récurrent dans la pratique. Dans ce cas, il est nécessaire d'utiliser un algorithme d'optimisation multicritère et des outils de visualisation spécifiques pour comparer les meilleures solutions trouvées. Il est plus pertinent d'utiliser des critères entrant en conflit, tels que la protection solaire et l'accès à la lumière naturelle, le confort d'été et la visibilité, les consommations énergétiques en été et les consommations énergétique en hiver. Il n'y a pas de limite sur le nombre de critères, mais plus il y a de critères, plus la lisibilité des résultats devient complexe. Dans la pratique, nous n'avons pas rencontré de problèmes où il était nécessaire de formuler plus de trois critères. Ce type de problème est accessible aux développeurs confirmés.

L'optimisation multicritère sous-contraintes

Le niveau 6 est nécessaire lorsque le problème est complexe (multicritères, façade courbe, irrégulières, contexte urbain hétérogène, etc.), ce qui est fréquent dans la pratique. Dans ce cas, il sera probablement nécessaire d'intégrer des contraintes dans le processus d'exploration et donc de choisir une méthode adaptée parmi les méthodes d'intégration existantes que nous verrons dans le chapitre suivant. Parmi elles, les fonctions de réparation développées à l'aide de techniques génératives comme les modèles à base d'agents apparaissent comme adaptées aux problèmes d'optimisation combinatoire que posent les projets d'architecture. A ce stade, il est nécessaire d'être un développeur expert pour modéliser ce type de problème car l'intégration des contraintes nécessite fréquemment l'usage de la programmation informatique.

L'optimisation multicritère avec modèles de substitutions

Enfin, pour le niveau 7 il faut ajouter l'usage d'algorithme de *Machine Learning* pour réduire les temps de calcul en créant des modèles de substitution. Pour le moment, ce niveau n'a pas été testé sur des cas réels en cours chez AS. Selon le type d'algorithme utilisé (présentés à la section 3 du chapitre 3), il sera nécessaire d'être un développeur confirmé, voir expert pour les utiliser.

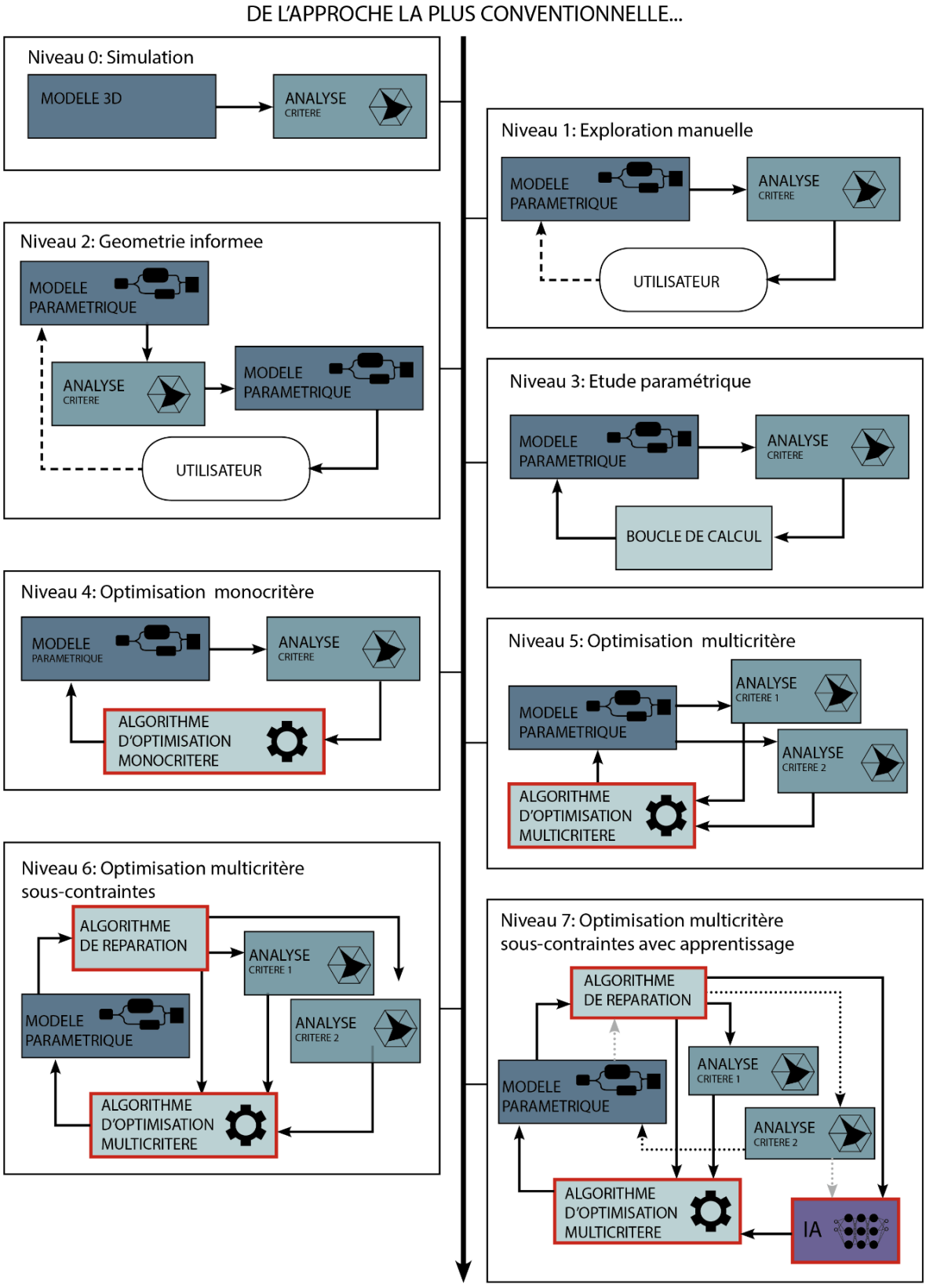


Figure 113: Les différentes méthodes d'exploration

Comme illustré en Figure 113, plus on est proche des premiers niveaux, moins on utilise de techniques génératives (encadrés en rouge). On ne peut alors parler que de « *design paramétrique* ». Plus on se rapproche du niveau 7, plus le nombre de techniques génératives nécessaires augmente (algorithme génétique, modèle à base d'agent pour l'intégration des contraintes, algorithme de Machine Learning pour les modèles de substitution), on peut alors parler de « *design génératif* ». Finalement, il n'y a pas une mais des méthodes informatiques pour faire de l'exploration numérique, plus ou moins difficiles à mettre en œuvre et plus ou moins adaptées selon les types de problèmes rencontrés dans la pratique.

Différentes configurations d'expérimentations

Commanditaires, motivations et usages de l'exploration

Au total 44 expérimentations ont été effectuées entre janvier 2019 et février 2023. Environ un tiers de ces expérimentations étaient de la conception traditionnelle. C'est la méthode la plus utilisée. Ces demandes venaient souvent d'une inquiétude d'un maître d'ouvrage, c'est notamment le cas sur les concours en PPP (partenariat public privé), d'un cahier des charges très exigeant, d'une petite affaire non accompagnée par un bureau d'étude, ou des architectes eux-mêmes qui souhaitent montrer les savoirs faire de l'agence lors d'un concours. Les outils d'analyse à destination des architectes intègrent des outils de visualisation qui permettent d'en faire de bons outils de communication, utiles pour rassurer la maîtrise d'ouvrage. Nous parlons alors d'un usage de « *communication* ». La demande venait parfois des associés ou chefs de projet qui souhaitaient s'assurer que leur projet respecte les exigences du programme avant de s'engager sur un projet pour un rendu de concours. Nous parlons alors d'un usage de « *validation* ». Ces deux usages que sont la communication et la validation ne sont pas réservés à la méthode traditionnelle.

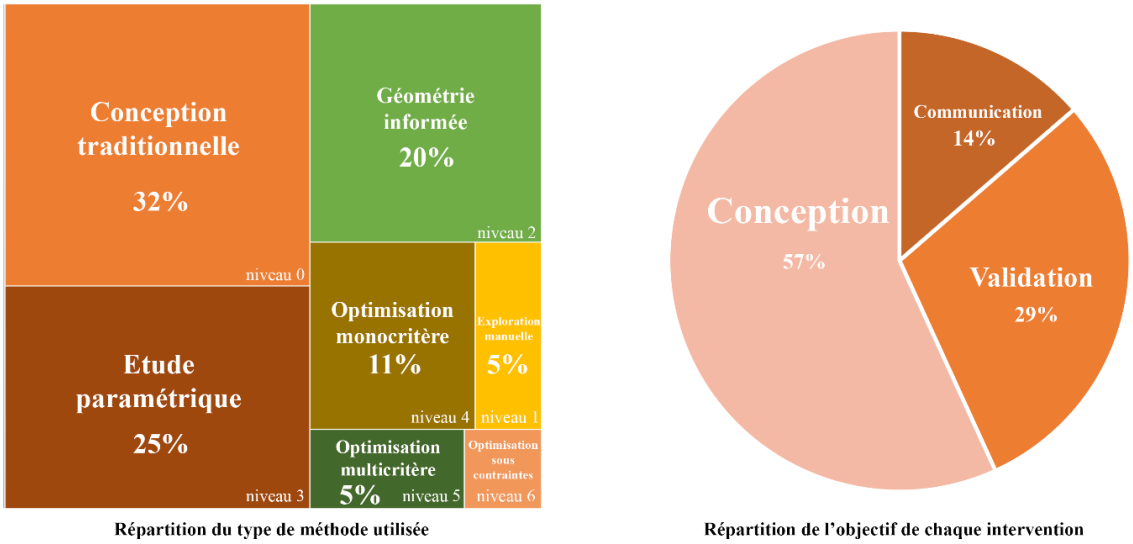


Figure 114: Répartitions des différentes méthodes et usages de l'exploration

Les boucles de calcul peuvent être utilisées pour produire des images animées grâce aux captures d'écran automatiques. On peut ainsi montrer l'évolution de la qualité de l'éclairage dans une pièce au cours d'une journée ou l'évolution de l'ensoleillement des espaces extérieurs tout au long de l'année. Au total 14 % des expérimentations étaient à des fins de communication, et 29 % à des fins de validation d'une solution. Le reste des expérimentations (57 %) étaient réellement consacré à l'aide à la conception.

Les études paramétriques et la géométrie informée sont, après la méthode traditionnelle, les méthodes que nous avons le plus utilisées, correspondant respectivement à 25 % et 20 % du total des expérimentations. Finalement, seulement 17 % des expérimentations ont abouti à l'usage d'algorithmes d'optimisation. Les 5 % qui restent étaient de l'exploration manuelle.
Nous avons synthétisé l'ensemble de ces expérimentations dans un tableau (voir Tableau 10) où les projets sont triés par ordre chronologique. Comme l'illustre le graphique en Figure 115, il n'y a pas eu au cours du temps une évolution particulière de la méthode utilisée. Nous avons continué au cours des quatre années à être régulièrement sollicités pour des interventions qui n'étaient pas de l'exploration mais seulement de la simulation. Cependant, au départ, les explorations étaient réalisées à notre propre initiative, sur les deux dernières années ce sont les équipes de projet qui étaient demandeuses de solutions optimisées. Avec le temps, les sollicitations ont eu tendance à se faire de plus en plus tôt dans le projet, avant qu'une solution ne soit figée.

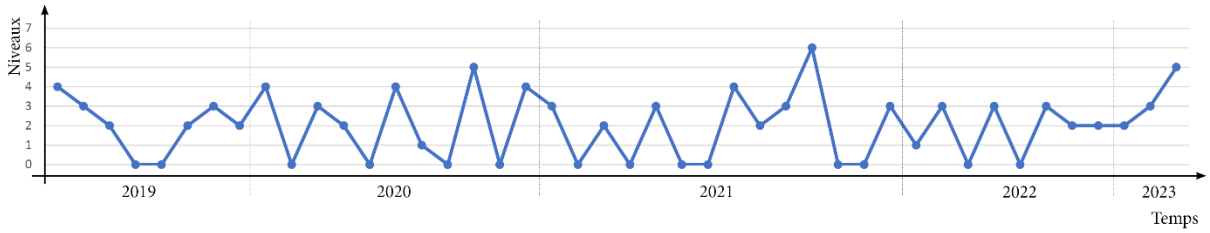


Figure 115 : Evolution du niveau des explorations au cours du temps

Il est important de noter que nos compétences en développement ont aussi beaucoup évolué au cours des expérimentations, ce qui peut expliquer pourquoi l'optimisation multicritère sous contrainte n'a été expérimentée lors des premières années. Au cours de l'année 2019 nous avons démarré en testant l'optimisation monocritère, en 2020 nous nous sommes intéressés à l'optimisation multicritère, en 2021 nous avons étudié les méthodes d'intégration des contraintes et en 2022 nous avons commencé à tester les algorithmes de *Machine Learning* pour les modèles de substitution.

Tableau 10: Liste des expérimentations

PROJET	LIEU	CLIMAT	DATE	PROGRAMME	PHASE	USAGE	CRITERES	NIV	METHODE
Institut Faire Face	Amiens, France	océanique	01/2019	laboratoires	développement	conception		4	
Campus ESSEC 2023	Cergy-Pontoise, France	océanique	02/2019	université	développement	conception		3	
Nouveau CHU de la Gualoupe	Pointe-à-Titre, France	tropical	03/2019	hospitalier	chantier	communication		2	
Immeuble de bureau	Buenos-Aires, Argentine	tropical	03/2019	résidentiel	développement	vérification		0	
ZAC du triangle de l'Echat à Créteil	Créteil, France	océanique	04/2019	urbanisme	développement	vérification		0	
Nouvelle salle de concert d'Ostrava	Ostrava, République Tchèque	montagnard	04/2019	salle de concert	esquisse	conception		2	
Bâtiment résidentiel à Buenos Aires	Buenos-Aires, Argentine	tropical	05/2019	résidentiel	développement	conception		3	
Immeuble de bureaux à la Toison d'Or	Dijon, France	océanique	09/2019	bureaux	esquisse	conception		2	
Hotel Akwaba	Abidjan, Côte d'Ivoire	équatorial	01/2020	hôtel	développement	conception		4	
Campus universitaire de Benguerir	Benguerir, Maroc	méditérannéen	02/2020	logements	développement	vérification		0	
Hôpital universitaire Grand Paris Nord	Saint Ouen-sur-Seine, France	océanique	03/2020	hospitalier	esquisse	conception		3	
Immeuble «Ame»	Montevideo, Uruguay	tropical	06/2020	logements	esquisse	conception		2	
Extansion hôpital d'Yverdon-les-bains	Yverdon-les-bains, Suisse	océanique	06/2020	hospitalier	esquisse	vérication		0	
Siège de l'organisation de la coopération islamique	Djeddah, Arabie Saoudite	aride	07/2020	bureaux	esquisse	conception		4	
Cité administrative - Caserne Thiry	Nancy, France	océanique	10/2020	bureaux	esquisse	conception		1	
Gare du Grand Paris Express	Nanterre-la-folie	océanique	10/2020	gare	esquisse	vérification		0	
Tribunal des affaires sociales de Wintherthur	Wintherthur, Suisse	océanique	10/2020	tribunal	esquisse	conception		5	
Cité scolaire internationale de Marseille	Marseille, France	méditérannéen	11/2020	école	esquisse	vérification		0	
Immeuble «Ame»	Montevideo, Uruguay	tropical	12/2020	logements	esquisse	conception		4	
Cité administrative de Toulouse	Toulouse, France	méditérannéen	01/2021	bureaux	esquisse	conception		3	
Résidence pour personne handicapée	Montigny le bretonneux	océanique	02/2021	logements	chantier	communication		0	
Tour Axian à Antananarivo	Antananarivo, Madagascar	tropical	02/2021	bureaux	esquisse	conception		2	

PROJET	LIEU	CLIMAT	DATE	PROGRAMME	PHASE	USAGE	CRITERES	NIV	METHODE
Bâtiment de bureau	Créteil, France	océanique	03/2021	bureaux	chantier	vérification		0	
Centre d'exploitation de Rosny-sous-Bois	Rosny-sous-Bois, France	océanique	03/2021	ateliers, gare	esquisse	conception		3	
Etablissement pénitentiaire des Grands-Marais	Orbe, France	océanique	03/2021	prison	esquisse	communication		0	
Transformation de l'immeuble TATI-Barbès	Bobigny, France	océanique	05/2021	gare	esquisse	conception		4	
Gare du Grand Paris Express Les Agnettes	Les Agnettes, France	océanique	09/2021	gare, logements	esquisse	vérification		0	
Cité du Ministère de la justice	St-Laurent-de-Maroni, Guyane	équatorial	09/2021	palais de justice	esquisse	conception		3	
Aménagement urbains du plateau Nord-Est	Chartres, France	océanique	10/2021	urbanisme	développement	communication		6	
Centre Pompidou à Massy	Massy, France	océanique	10/2021	musée archives	esquisse	communication		0	
Immeuble «Le Gallo»	Boulogne-Billancourt, France	océanique	11/2021	bureaux	esquisse	vérification		0	
Quartier Eurêka	Montpellier, France	méditérannéen	11/2021	bureaux, logements	esquisse	communication		3	
Prison de St-Laurent-de-maroni	St-Laurent-de-Maroni, Guyane	équatorial	12/2021	prison	esquisse	vérification		1	
Immeuble rue de l'université à Paris	Paris, France	océanique	01/2022	bureaux	esquisse	conception		3	
Immeubles de logements à Dakar	Dakar, Sénégal	aride	01/2022	logements	esquisse	vérication		0	
Résidence pour personne handicapée	Montigny le bretonneux, France	aridocéanique	03/2022	logements	chantier	conception		3	
Hôpital des enfants du HUG	Genève, Suisse	océanique	03/2022	hospitalier	esquisse	conception		0	
Opération d'aménagement de grand meridia à Nice	Nice, France	méditérannéen	04/2022	urbanisme	développemnt	communication		3	
Quartier média, innovation et civique à Riyad	Riyad, Arabie Saoudite	océanique	11/2022	bureaux, logements	esquisse	conception		2	
Learning center de l'ESTP	Cachan, France	méditérannéen	12/2022	université	esquisse	conception		2	
Ecole européenne et nouveau collège les renardières	Courbevoie, Paris	tropical	01/2023	écoles	esquisse	conception		2	
Le grand phare d'Issy	Issy-les-moulineaux, France	méditérannéen	01/2023	logements	développement	conception		3	
Cour d'appel de Bouaké	Abidjan, Côte d'Ivoire	océanique	02/2023	tribunal	esquisse	conception		5	

Comme le montre la Figure 116, les expérimentations ont été réalisées sur une grande variété de programmes : 43 % des interventions ont été réalisées sur des équipements publics, 21 % sur des bureaux, et 28 % sur des logements ; 6 % des interventions ont été réalisées à l'échelle urbaine mais seulement une expérimentation a abouti à l'usage d'algorithmes d'optimisation à cette échelle. De manière générale, ce sont les espaces de travail (bureaux, écoles, laboratoires, palais de justice) qui présentent le plus de problèmes d'optimisation multicritère. Le compromis entre la protection solaire, l'accès à la lumière naturelle et à la visibilité vers l'extérieur sont les critères les plus récurrents dans ce type de problèmes.

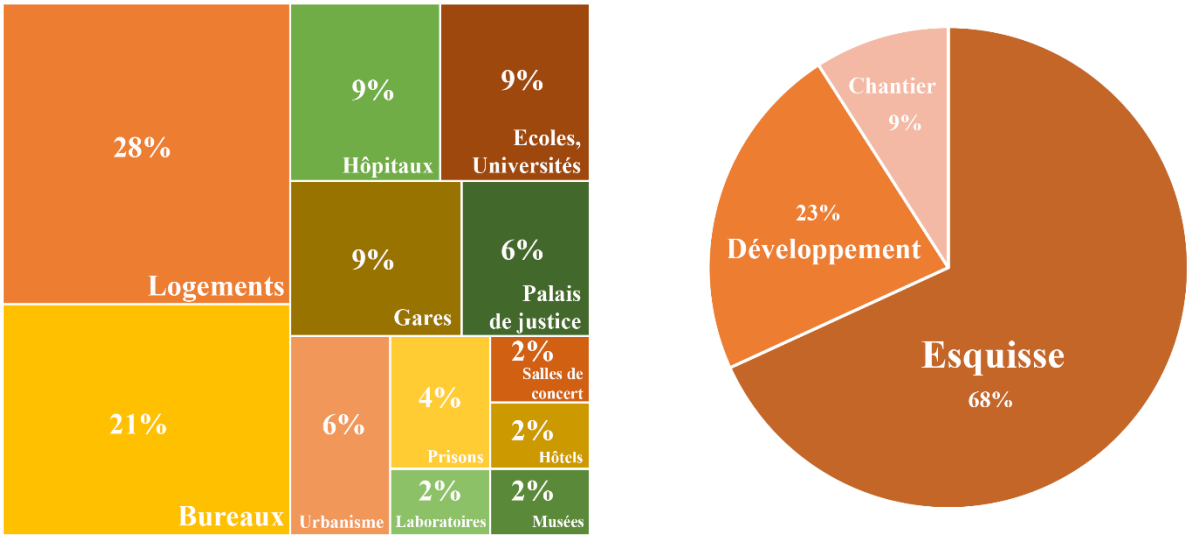


Figure 116 : Répartition des différents programmes (à gauche) et des phases (à droite)

Dans l'ensemble, ces trois critères sont les plus demandés par les équipes de projet (voir Figure 117). Finalement, nous n'avons jamais réussi à expérimenter de simulations énergétiques en phases amonts. Ce type de simulation est complexe à modéliser, plus spécifiquement à l'échelle du détail de l'enveloppe. Les architectes préféraient faire appel aux bureaux d'étude pour ce type d'analyse, et sont habitués à intégrer la STD dans les phases avales du développement de projet. La grande majorité des interventions ont eu lieu en phase esquisse (pour 68 % des cas). Dans la plupart des cas, il s'agissait de concours. Nous avons pu expérimenter de l'optimisation dans des phases plus avales car l'enveloppe des bâtiments est un élément qui peut encore beaucoup évoluer après les concours, même lors des phases chantier où il est encore possible (voire parfois nécessaire) de rationnaliser le processus de fabrication. Un sujet comme l'orientation des éléments d'un système de protection solaire peut encore être discuté en phase de chantier.

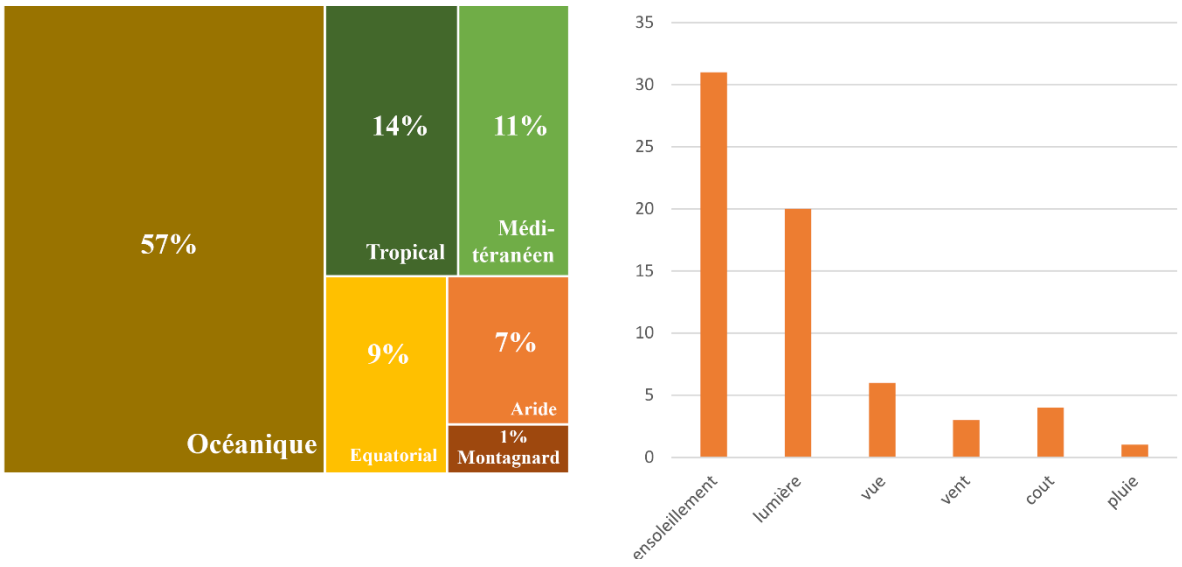


Figure 117 : Répartition des climats (à gauche, et des critères étudiés (à droite)

Pour la plupart des expérimentations réalisées, nous avons rencontré deux grandes configurations en termes de typologie de climat, soit un climat tempéré avec une opposition entre saison chaude et saison froide, soit des climats chauds où l'enjeu est essentiellement de se protéger du soleil toute l'année.

Les difficultés rencontrées pour la mise en place de l'optimisation.

Avant de rentrer dans le détail de certaines expérimentations dans la section suivante de ce chapitre, nous formulons ici une première synthèse des difficultés générales rencontrées dans la pratique. Si nous avons pu observer certaines limites précédemment exposées dans la littérature scientifique que nous avons classée en 4 catégories (pédagogie, interface, robustesse, technique), nous avons aussi observé des difficultés d'un autre ordre que nous qualifions de « *structurel* ».

Les limites « *structurelles* »

Il s'agit des limites liées aux méthodes de travail et d'organisation des architectes en agence. Ces dernières sont particulières, bien distinctes des méthodes utilisées par les ingénieurs. Pour élaborer notre théorie, nous nous appuyons sur nos observations personnelles sur le terrain et sur les écrits de Bryan Lawson, chercheur britannique, architecte et psychologue qui a décrit le processus de conception et les méthodes de travail des designers et plus spécifiquement des architectes dans son ouvrage *How Designers Think* (Lawson, 2006).

La pensée des architectes s'accompagne souvent de philosophies de conception que Lawson appelle « *design philosophies* » : *« Rather, designers have their own motivations,*

reasons for wanting to design, sets of beliefs, values and attitudes ». Toujours d'après Lawson, celle-ci peut avoir un impact très fort sur leurs méthodes de travail, comme chez AS où la philosophie pour une conception collaborative est incarnée dans la méthode du tracé rouge. Plus qu'une méthode, selon nous, les ENAPE peuvent être considérées comme une philosophie de conception, et celle-ci ne convient pas à tous les architectes. Dans la pratique, nous avons pu observer que certains architectes d'Architecture-Studio adhèrent complètement à cette philosophie à tel point qu'ils peuvent insister pour utiliser des algorithmes d'optimisation sur des problèmes qui ne s'y prêtent pas. A l'inverse, nous avons cherché à optimiser certaines solutions proposées par des équipes qui ne se sont jamais intéressées aux résultats des explorations.

Ensuite, nous avons pu observer que ces phases amonts où les ENAPE auraient le plus d'impact sur la performance des bâtiments, sont aussi celles où les collaborateurs doivent faire face à fort niveau de pressions. Les chefs de projet, même adhérants à la philosophie, ne sont pas toujours sereins à l'idée d'innover dans l'urgence. Lors de nos expérimentations en phases concours, souvent le projet avait complètement changé de direction avant même d'avoir exploiter les résultats de l'optimisation, ou alors une solution avait fini par être figée par l'équipe avant même de recevoir les résultats de l'analyse.

Changer de méthode de travail, quelles qu'elles soient, représente un coût et un risque, donc un effort pour une entreprise qu'elle fera difficilement si elle n'y voit pas un intérêt clair. Finalement, c'est lorsque les architectes y étaient contraints par une demande de la maîtrise d'ouvrage (soit au cours du développement du projet, soit via une programmation particulièrement exigeante) que nous avons été le plus sollicités.

Pour mettre en place ces méthodes, nous l'avons vu, il faut être *a minima* un développeur confirmé. Ce type de compétences relève de la spécialisation (Boissieu, 2020), elles sont alors rares chez les chefs de projet et les directeurs d'agence. Ainsi, lorsqu'un architecte-développeur met en place ce type d'approche, il le fait sous la direction d'un chef de projet et d'un directeur d'agence. Il doit ainsi jongler entre deux types de managements qui ne sont pas toujours bien adapté à l'utilisation d'ENAPE.

Le chef de projet pratique une gestion du projet et de son équipe similaire à ce qui se trouve dans d'autres domaines. Il est souvent le seul à avoir la vision globale du projet, de l'ensemble de ses contraintes, du planning, des différents intervenants et des différentes tâches à réaliser. Ainsi, pour gagner du temps, il transmet à chaque collaborateur exclusivement les informations qu'il juge nécessaires à la bonne exécution de la tâche qui lui est confiée. Ce

dernier est accoutumé, lorsqu'il s'agit des performances environnementales, à transmettre au bon interlocuteur (qui parfois diffère selon le critère à étudier) la maquette 3D et la liste des matériaux dont il a besoin. L'interlocuteur est souvent chargé de réaliser des simulations où les critères sont traités séparément, ce qui n'est pas le cas lorsqu'on fait de l'optimisation. Pour faire des ENAPE, l'architecte-développeur a besoin de beaucoup plus d'informations sur le projet : quels sont les critères de performance à prendre en compte ? certains critères sont-ils plus contraignants que d'autres ? quels sont les paramètres géométriques et les matériaux qu'il est possible de faire varier ? sur quel ensemble de valeurs ? Y-a-t-il des contraintes rendant certaines solutions inexploitables (structurelle, économique, normative) ? Répondre à toutes ces questions demande un certain temps d'échange et parfois même plusieurs allers-retours pour pouvoir formuler correctement un problème avant de lancer des calculs. Lors de nos expérimentations, les cas où le chef de projet cherchait à restreindre les échanges à des transferts de maquette n'ont jamais pu aboutir à de réelles explorations.

Les méthodes de management des directeurs d'agence d'architecture sont moins conventionnelles que celles des chefs de projet. Elles s'inscrivent dans une continuité des pratiques des écoles d'architecture où les étudiants produisent et les professeurs corrigent. La métaphore utilisée par Bryan Lawson en parlant du management de l'architecte Michael Wilford nous semble particulièrement adaptée pour décrire la façon dont le processus de conception est souvent contrôlé chez AS. Les collaborateurs peuvent être apparentés à des journalistes et le directeur d'agence à un éditeur de journaux à qui ils présenteraient leurs articles pour qu'il puisse suggérer des modifications. L'usage de l'optimisation multicritère est déstabilisant avec ce type de fonctionnement, car il ne s'agit pas de corriger une solution, ou bien même quelques variantes, ou encore la pluralité de solutions situées sur le front de Pareto. Il s'agirait soit d'exploiter les résultats et choisir une solution, soit remettre en question la formulation du problème, notamment la qualité de l'espace de solution de départ. Dans les deux cas, cela implique des connaissances spécifiques comme la maîtrise des outils de data visualisation ou la notion même d'espace de solutions et sa modélisation. Lors de nos premières interventions dans la pratique chez AS, nous pensions pouvoir faire de la base de données servant d'entrée à l'application web DesignExplorer® le livrable officiel de nos expérimentations. Les rendus incluant plus d'une dizaine de variantes n'ayant fait l'objet d'aucun retour de la part des équipes, nous avons rapidement abandonné l'idée. Finalement, nous produisions des rapports contenant seulement quelques variantes situées sur le front Pareto. Les équipes étaient ensuite libres de présenter ou non l'intégralité ou une partie de son contenu aux associés lors des revues de projet. Si cette organisation a le mérite de ne pas trop

bouleverser l'organisation traditionnelle, elle entraîne selon nous un appauvrissement de la démarche. Pour que l'on puisse réellement parler d'aide à la décision, il faudrait qu'une partie au moins du processus, notamment la phase finale d'exploration de l'espace de solutions généré, soit réalisée par un décideur, soit le chef de projet, soit l'architecte en chef.

Un autre sujet qui peut expliquer le gap qui existe entre théorie et pratique pour l'exploration numérique est la méconnaissance que l'on peut avoir du rôle du dessin chez les architectes. L'idée qu'utiliser l'ordinateur pour dessiner à la place de l'architecte lui permettrait systématiquement de gagner du temps est partiellement faussée. Le dessin n'est pas uniquement un outil de représentation. Il s'agit aussi et surtout d'un outil de conception que Bryan Lawson appelle le « *design drawing* ». Lors de la conception d'un projet, l'architecte rencontre de multiples problèmes. C'est par le dessin, en cherchant des solutions que ce dernier va pouvoir les résoudre mais aussi en soulever de nouveaux. Ainsi, les problèmes d'architecture ne peuvent être énoncés de manière exhaustive car il est impossible d'être certain d'avoir fait apparaitre tous les problèmes que soulèvent un projet. Ce simple fait explique pourquoi il est aussi difficile de formuler une bonne fois pour toute un problème de conception pour pouvoir le résoudre à l'aide d'algorithmes d'optimisation. Les solutions choisies au terme d'un processus d'exploration numérique sont donc fatalement appelées à être modifiées. Si l'architecte ne sait pas pourquoi cette solution est plus pertinente qu'une autre, les caractéristiques qui ont fait que l'algorithme l'a mise en avant risquent de disparaitre au fil des modifications successives.
Ainsi, pour être véritablement utiles aux architectes, ces explorations doivent permettre de mieux définir le problème autant que de présenter des solutions. Il est nécessaire de pouvoir déduire un apprentissage de ces explorations. Lors de nos expérimentations, nous avons observé que les conclusions sous la forme de principes de conceptions étaient parfois plus appréciées que le résultat géométrique lui-même. Inversement, il était courant que le résultat géométrique d'une exploration pour une enveloppe plaise aux architectes et soit conservé dans un rendu même si toute la logique paramétrique et performancielle derrière soit rendue caduque à la suite de modifications d'autres éléments comme la morphologie du bâtiment.

Finalement, il est complexe de dire clairement comment et quand intégrer les méthodes d'exploration numérique de façon systématique dans les projets d'architecture. D'après Lawson, il n'y a pas une méthode de conception, mais des méthodes qui diffèrent selon les concepteurs et les projets rendant le processus de conception difficile à définir. Cependant, il parait possible de distinguer deux usages de l'optimisation. Lawson s'appuie sur une

description que fait la Royal Institute of British Architects (RIBA) pour décrire le processus de conception commun à tous les types de design que nous utiliserons pour développer notre propos (voir en Figure 118). Celui-ci compte 4 étapes qui sont à distinguer des phases de réalisation d'un projet de construction (ESQ, APS, APD, PC…). Une première étape consiste à étudier toutes les informations, contraintes et données du problème, il s'agit de **l'étape d'assimilation**. Une seconde étape, nommée « **études générales** », correspond à l'étude de la nature du problème et des solutions possibles. Une troisième étape consiste à développer une des solutions identifiées à l'étape précédente, il s'agit du « **développement** ». Enfin la dernière étape permet de **communiquer** le projet.

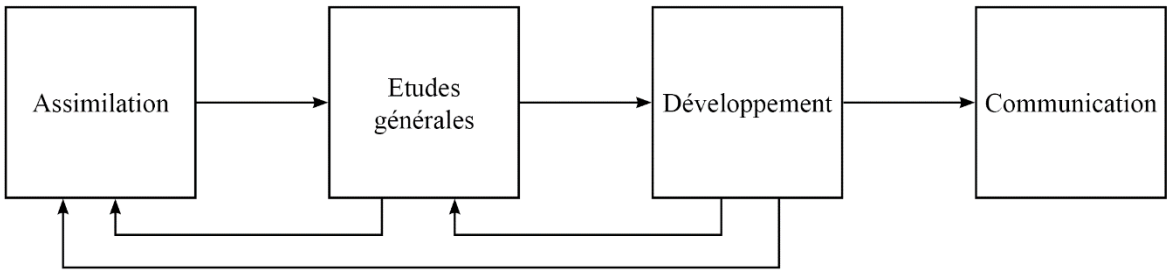


Figure 118 : Processus de conception selon RIBA (source: Lawson, 2006)

Lors de nos interventions sur des projets chez AS, nous avons exclusivement expérimenter l'usage de l'optimisation en phase de développement pour lequel les modèles paramétriques sont bien adaptés. Il s'agit alors plus d'optimiser une solution en lui appliquant des transformations à la marge que de faire une réelle exploration du champ des possibles. Ainsi, les expérimentations qui ont eu le plus de succès ont eu lieu lors des phases d'études d'avant-projet qui correspondent aux phases de développement d'une solution. Un autre usage de l'optimisation serait une utilisation lors de l'étape des études générales, avant d'avoir choisi une solution à développer. Cela implique d'être en mesure de proposer des espaces de solutions avec des formes beaucoup plus diverses, avec des variations topologiques difficiles à produire avec des modèles paramétriques. Ces modèles, beaucoup plus difficiles à mettre en œuvre seraient alors plus adaptés aux phases esquisses, mais impliqueraient la maîtrise d'autres techniques génératives nécessitant les compétences d'un développeur expert. La taille de l'espace de solutions serait aussi beaucoup plus importante imposant une augmentation des temps de calculs. Ainsi, certaines limites structurelles de ces méthodes innovantes nous renvoient à leurs limites techniques.

Les limites techniques

Nous avons soulevé deux limites techniques qui, selon nous freinent la mise en pratique de ces méthodes de conception innovantes. Il y a d'abord les temps de calcul qui réduisent la fluidité de la méthode et la difficulté d'intégrer des contraintes dans les modèles. Ces deux sujets feront l'objet d'études plus approfondies dans le chapitre suivant.

Les expérimentations concernant principalement les enveloppes, et les équipes continuant d'être accompagnées par des BE sur les questions d'énergie, nous avons été sollicités essentiellement sur des critères d'ensoleillement, de qualité de vue et de lumière naturelle. Ainsi, les temps de calculs n'ont jamais été rédhibitoire, mais bien souvent nous avons restreint la taille de l'espace de solutions pour pourvoir respecter les plannings. Les temps de calculs selon les projets varient d'une nuit à trois jours, alors effectués dans ce cas les week-ends, ils ne freinaient pas particulièrement l'avancement des projets. Avec le temps, nous avons de plus en plus été sollicités sur des sujets plus complexes (échelle urbaine, CFD) pour lesquels l'exploration d'une quantité importante de solutions est difficilement concevable sans l'usage de modèles de substitution réalisés à l'aide d'algorithmes de *Machine Learning*. Nous n'avons pas pu à ce jour les tester sur un cas pratique, mais avec le recul et les études comparatives présentées dans le chapitre 3, nous savons que plus qu'un gain de temps, ils nous auraient permis d'étudier beaucoup plus de solutions dans le même temps imparti, car ces algorithmes nécessitent d'être entrainés sur un nombre tout de même conséquent de solutions pour être fiables. Ils permettraient peut-être de prendre en compte d'autres critères comme les besoins énergétiques, ou l'ACV. Cependant, il restera toujours un temps de calcul incompressible pour générer des bases de données pour l'apprentissage qui nécessitent d'être reconduites dès qu'on change ne serait-ce qu'une seule variable du modèle paramétrique, ou un invariant (fichier météo, paramètres physiques des matériaux, contexte urbain). Ces temps incompressibles obligent à une certaine organisation, et implique surtout des connaissances à ajouter à une liste déjà bien longue.

Selon nous, la limite technique la plus rédhibitoire était la gestion des contraintes. Pour que l'exploration soit utile, les solutions générées sur le front de Pareto doivent être viables, constructibles et présentables à un client. Si seulement quelques solutions de l'espace préalablement défini ne le sont pas, cela reste facile à gérer, mais si la plupart des solutions sont caduques alors l'algorithme ne peut être démarré en l'état. Au cours de ces expérimentations, nous avons régulièrement été confrontés à ce problème. Ainsi, nous avons dû soit reformuler

les problèmes en réduisant drastiquement leur complexité et au passage la taille de l'espace de solutions, soit abandonner le projet faute d'avoir trouvé une méthode alternative acceptable.

Pour résoudre ce problème, nous verrons dans les chapitres suivants que les chercheurs en conception computationnelle, en particulier en optimisation appliquée à l'architecture, ont un véritable intérêt à s'intéresser à des méthodes hybrides qui mêlent les algorithmes génétiques à d'autres techniques génératives comme les MBA ou les AC.

Les autres limites (pédagogie, interface)

Finalement, il apparait que le manque de connaissance reste un frein majeur à la mise en pratique de ces méthodes. Si la présence dans l'agence d'un designer computationnel ayant été formé à ces approches est bien évidemment nécessaire, le besoin se situe, selon nous, aussi au niveau des autres collaborateurs. Il n'est pas nécessaire que tous les architectes connaissent parfaitement toute la chaine d'outils indispensables à la mise en place de ces méthodes. Cependant, nous avons observé que lorsque le commanditaire était un « *utilisateur* » de script, alors la formulation et la modélisation du problème était beaucoup plus fluide. Pour que la méthode aboutisse, il apparait comme essentiel que le commanditaire soit au moins un « *initié* » et ne perçoive pas ces outils d'intelligence numérique comme une baguette magique capable de rendre un projet avec un programme surcontraint faisable ou d'atteindre des objectifs de performance « *sans modifier l'architecture* » comme nous avons pu le rencontrer lors de certaines expérimentations.

Nous venons aussi de voir que pour profiter au mieux de l'apport de ces approches, les décideurs doivent avoir été formés à l'usage des outils de post-traitement des optimisations, aussi appelés les outils de data visualisation. La plupart des échanges avec les décideurs étant réalisés de manière informelle (souvent derrière un ordinateur), il serait plus fluide d'avoir une interface pour le post-traitement, à l'image de Design Explorer®, directement intégré dans les outils de programmation visuelle, qui permettrait d'analyser et de visualiser les premiers résultats de manière interactive alors que l'exploration est en cours.

Nous verrons aussi plus tard dans le chapitre 3 qu'il manque des interfaces pour utiliser dans Grasshopper® certains algorithmes de *Machine Learning* notamment pour la génération d'images. Ces algorithmes permettent de prédire non seulement la performance mais une visualisation graphique de cette dernière. La visualisation étant essentielle aux architectes, une telle interface favoriserait peut-être l'usage des modèles de substitution et donc la réduction des temps de calcul.

Pour faciliter l'intégrer des contraintes, il faut faciliter l'utilisation d'autres techniques génératives et concevoir ou implémenter des solveurs pour créer des MBA ou des AC, ce que nous avons commencé à faire dans le chapitre 4 de cette thèse.

Au terme de cette première section, il apparait que les limites pouvant expliquer l'écart entre la théorie et la pratique sont nombreuses. Elles peuvent être classées en différentes catégories : les limites liées au manque de pédagogie, au manque d'interface, à la remise en question de la robustesse de la méthode, les limites techniques et enfin les limites structurelles liées au méthode de travail des architectes et d'organisation des agences d'architecture. Pour illustrer ces difficultés rencontrées dans la pratique, nous proposons au lecteur de revenir dans la section suivante sur certaines des expérimentations conduites chez Architecture Studio.

2.2. Les expérimentations notables

Dans cette seconde partie du chapitre 2, nous revenons sur deux types d'expérimentations. Dans un premier temps, nous présentons certaines expérimentations ayant abouties. Nous avons sélectionné des expérimentations réalisées à différentes phases du projet pour essayer de comprendre si certaines méthodes sont plus adaptées que d'autres aux phases esquisses.

Dans un second temps, nous racontons les histoires telles que nous les avons vécues de quelques tentatives échouées d'expérimentations afin d'illustrer sur des cas concrets les limites techniques et structurelles que nous décrivions dans la section précédente.

2.2.1. L’enseignement des explorations ayant aboutis

Parmi l’ensemble des interventions réalisées sur des projets d’Architecture-Studio pendant ce doctorat, certaines ont pu aboutir à la mise en place de méthodes d’exploration numérique. Dans cette section, nous revenons sur les interventions les plus abouties pour la conception des enveloppes. Neuf expérimentations sont ainsi décrites : trois sur l’usage de l’optimisation en phase concours, trois autres sur l’usage de la méthode dite de la « *géométrie informée* » en phase concours et trois dernières sur l’usage de l’optimisation pendant les phases de développement de projet.

Exploration en phase concours

Le cas du siège social de l’organisation islamique à Djeddah

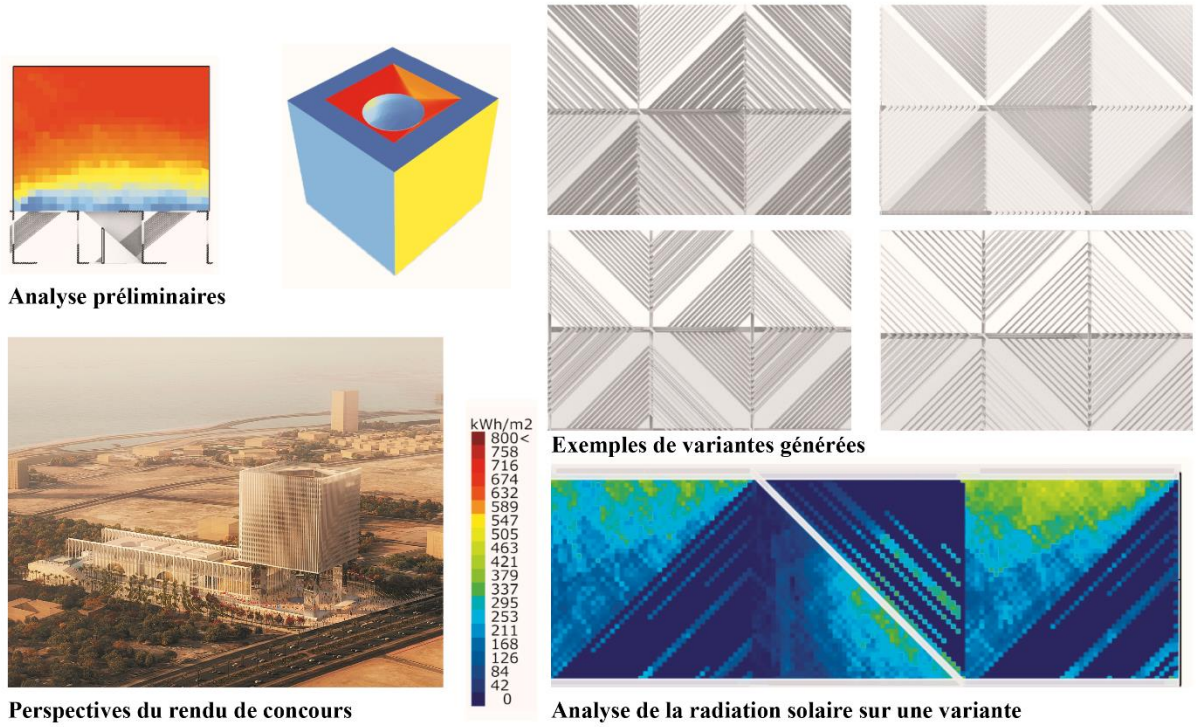


Figure 119 : Description du problème pour la façade du siège de l'organisation islamique

Lors du concours international pour le siège social de l’organisation islamique à Djeddah, les architectes d’Architecture-Studio ont proposé une tour cubique semblant presque en lévitation. Dans la première façade du concours, une première système de protection solaire avec un pattern en losange a été proposé pour cette tour que nous étions chargée d’optimiser. La demande provenait du chef de projet qui maîtrise la modélisation sur Rhinocéros et sait utiliser des scripts Grasshopper®. L’objectif était de produire un modèle paramétrique d’un détail pour pouvoir facilement générer des variantes pour la conception de l’enveloppe. Il s’agit

d’une enveloppe particulièrement épaisse, tramée avec des cubes pouvant accueillir des espaces extérieurs. Des lames en métal parcourent les différentes faces des cubes par leurs diagonales sans discontinuité de manière à créer un motif original présenté en Figure 119. Le modèle paramétrique contient donc cinq paramètres : l’épaisseur du motif, le nombre de lames, la largeur des lames, l’épaisseur des lames, et l’orientation des lames.

Il n’y avait pas de demande explicite de la part de l’équipe pour une optimisation. Le projet s’y prêtant et le modèle paramétrique étant existant, il n’était plus tellement coûteux de l’exploiter. Nous avons donc réalisé une optimisation de la radiation solaire annuelle avec l’algorithme génétique du solveur Galapagos pour essayer de minimiser les apports solaires. 83 générations de 50 individus chacune ont été calculées, soit 4292 évaluations, ce qui correspond à 0.26 % de l’espace de solutions qui compte 1 667 072 solutions.

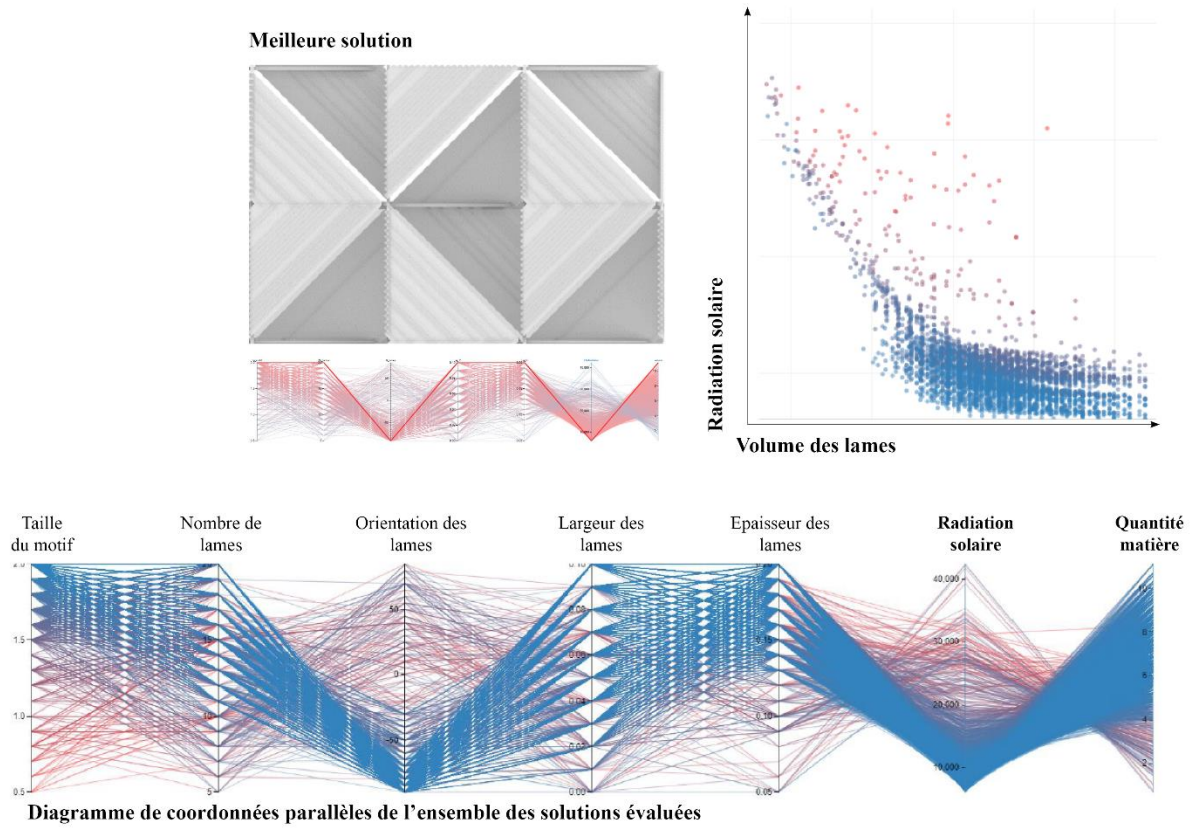


Figure 120 : Résultats de l'optimisation pour la façade du siège de l'organisation islamique

Cette expérimentation, l’une des premières, s’est révélée instructive. L’optimisation se faisant sur un unique critère, la solution optimale révèle une forme attendue où les lames créent un maximum d’opacité en utilisant systématiquement les extrêmes des intervalles de chaque paramètre géométrique, c’est-à-dire un nombre maximal de lames, d’une largeur maximale et une orientation verticale. Ainsi, les lames entrent en collisions au point de fusionner pour créer

un mur opaque. Dès lors, la solution optimale n'a pas grand intérêt en elle-même (voir Figure 120). Le problème, probablement trop rapidement formulé au départ, est un problème en coin (de type « *pour minimiser les apports solaires, il faut fermer les volets* »). Il aurait dû faire l'objet d'une optimisation multicritère en utilisant un critère qui entre en conflit avec les apports solaires (la visibilité, le coût, l'éclairage naturel). Nous avions testé cette approche avec le plugin Octopus en intégrant une évaluation du volume de matériau utilisé pour estimer le coût de construction, sans succès à cause de *crashs* successifs du solveur.

Pour exploiter les résultats, nous avons tout de même récolter les analyses de notre fonction de coûts (sans pour autant l'intégrer au processus d'optimisation), ce qui nous a permis d'intégrer ce critère dans l'analyse des résultats. Pour analyser l'ensemble des solutions évaluées lors du processus d'exploration, nous avons utilisé l'application web DesignExplorer2®. En conclusion, nous avons été en mesure de donner à l'équipe des intervalles restreints pour un dimensionnement efficace de ce système de protection solaire et nous avons proposé 5 variantes du détail de façades respectant ces prescriptions.

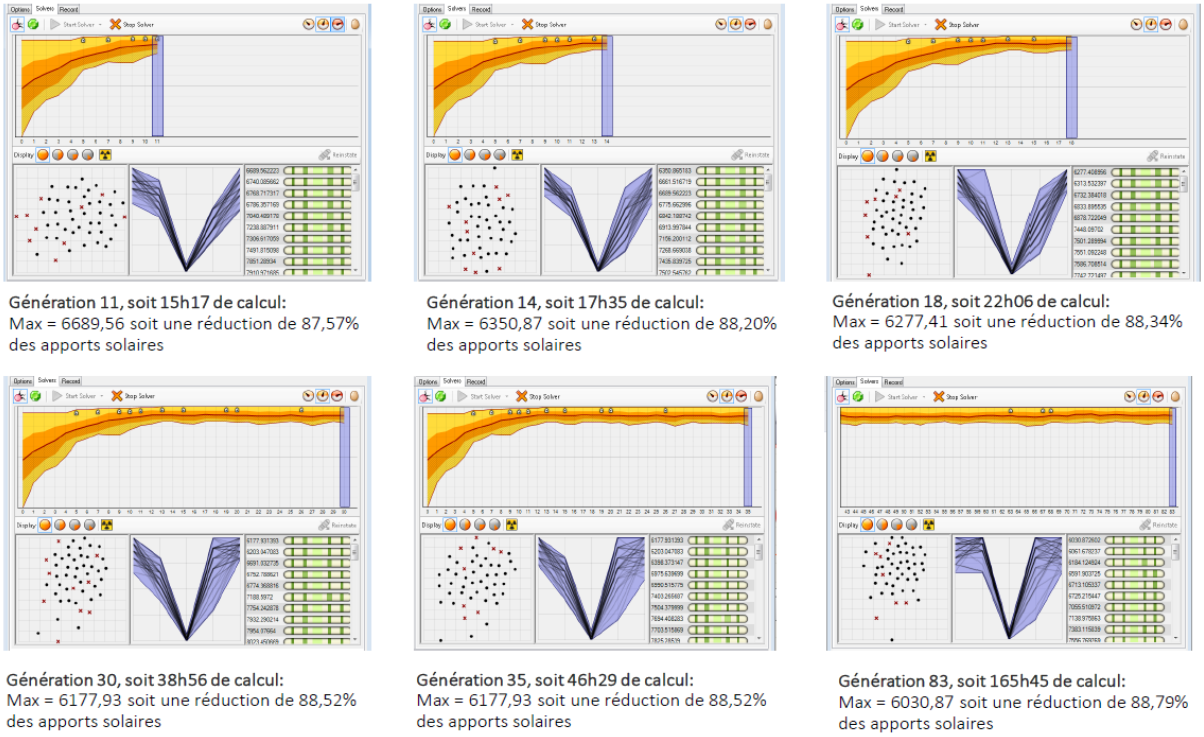


Figure 121 : Evolution de l'exploration au cours du temps

Bien qu'il s'agisse d'un concours, les temps de calculs n'ont pas été un problème. Nous avons laissé tourner l'algorithme plus de 165 h, ce qui correspond à plus de 6 jours. Ainsi, nous avons pu voir s'il y avait un véritable intérêt à attendre. Dans ce cas, la réponse était clairement

« *non* ». Comme le montre la Figure 121, les 150 dernières heures de calcul ont permis de réduire les apports solaires de 1,22 % seulement.

Finalement, l'équipe, très occupée par le concours, n'a pas pu prendre le temps d'analyser les résultats de l'optimisation, nous n'avons donc pas eu de retour de leur part. Les résultats n'ont pas pu être exploités par la suite car la façade n'était pas au goût du client et a dû être entièrement repensée pour la seconde phase du concours. La seconde proposition plus symbolique ne se prêtait pas à l'optimisation. Cette situation n'est pas restée un cas isolé, plusieurs cas d'expérimentation en phase concours sont restés inexploités par les équipes.

Le cas de la cour d'appel de Bouaké à Abidjan

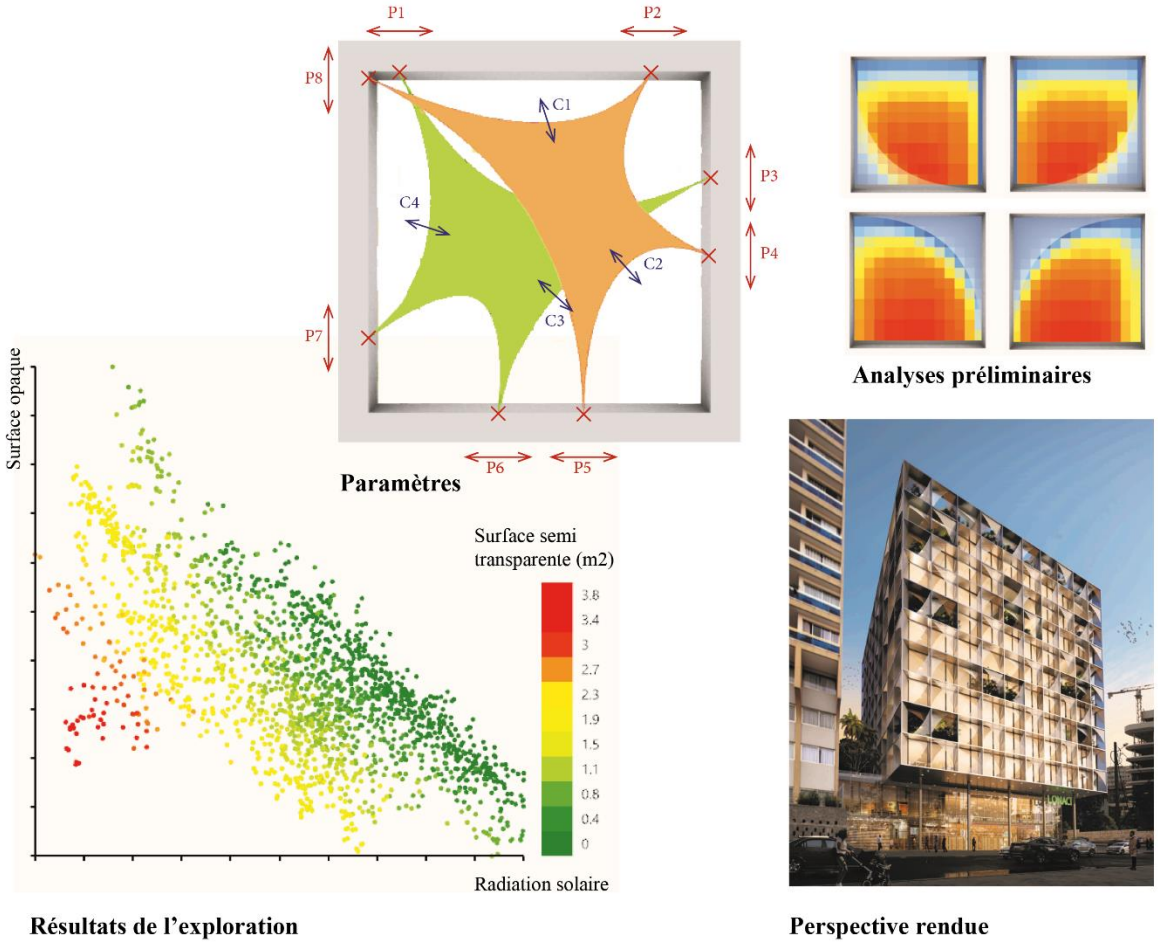


Figure 122 : Exploration pour la forme de toiles tendues

Pour ce concours pour une cour d'appel à Abidjan, les architectes d'Architecture Studio ont proposé un projet cubique jusqu'à son enveloppe, où chaque cube puisse acceuille une à deux grandes voiles apportant protection solaire et souplesse à la façade. Pour cette seconde expériementation, il s'est agi de nouveau de la conception d'un détail pour un système de

protection solaire. Une structure en béton avec une trame carrée permet de protéger de manière efficace trois façades sur quatre. La façade exposée à l'Ouest nécessitant d'être protégée d'avantage, l'architecte propose d'utiliser des toiles tendues, semi-opaques pour conserver une certaine qualité de vue. Nous avons été sollicités pour comprendre quelle forme serait la plus adaptée pour réduire efficacement les apports solaires. Une fois de plus, l'usage d'une approche générative ou paramétrique n'a pas été spécifiquement demandée. Des analyses préliminaires présentées dans la Figure 122, ont été réalisées avec une toile de forme simple. L'architecte souhaitait ensuite voir des propositions avec des formes plus complexes en utilisant deux toiles.

Un modèle paramétrique utilisant 12 paramètres (6 par toile) a été utilisé pour générer 2000 solutions dans un espace de 20 103 737 913 799 800 609 solutions (environ 10^{-15} %). Un algorithme d'optimisation multicritère (NSGAII) a été utilisé pour l'exploration sur 20 générations de 100 individus chacune. Trois critères ont été évalués : la radiation solaire sur le vitrage, la surface où les deux toiles se croisent, que l'on appelle « surface opaque » et la surface avec une seule toile (sans croisement des toiles). Les calculs ont duré 7 heures.

L'algorithme a permis de générer de nombreuses formes originales dont trois situées sur le front de Pareto (qui contient 88 solutions au total) sont présentées en Figure 123. Les temps de calculs n'ont pas été un obstacle en soi, mais nous n'avons pas eu le temps de présenter les résultats que l'architecte avait déjà fixé une solution à partir de l'analyse préliminaire (voir la perspective de rendue présentée en Figure 122). Dans ce type d'exploration où les formes générées peuvent fortement impacter l'image de la façade, proposer quelques variantes parmi les centaines générées font perdre beaucoup de sens à la démarche. L'usage par l'équipe de concepteurs des outils de post-traitement des résultats comme Designer Explorer2® aurait été particulièrement adapté au contexte de cette expérimentation.

L'ENAPE est particulièrement adaptée à la conception d'un détail de façade pour des critères comme l'ensoleillement, la lumière ou la vue. Les temps de calculs sont adaptés à la phase de concours même sans utiliser de modèles de substitution. D'après notre expérience, s'il persiste un blocage pour ce type d'application précise, il n'est pas technique, mais plus structurel. Comme nous l'avons vu, certains architectes ne partagent pas cette philosophie de conception. Ils n'ont pas l'habitude de pratiquer le projet d'architecture de cette façon, et peuvent avoir le sentiment de « *perdre le contrôle sur la forme* ». Les outils de post-traitement, comme Design Explorer® pourraient réduire ce sentiment de perte de contrôle à condition que les architectes aient été préalablement formés.

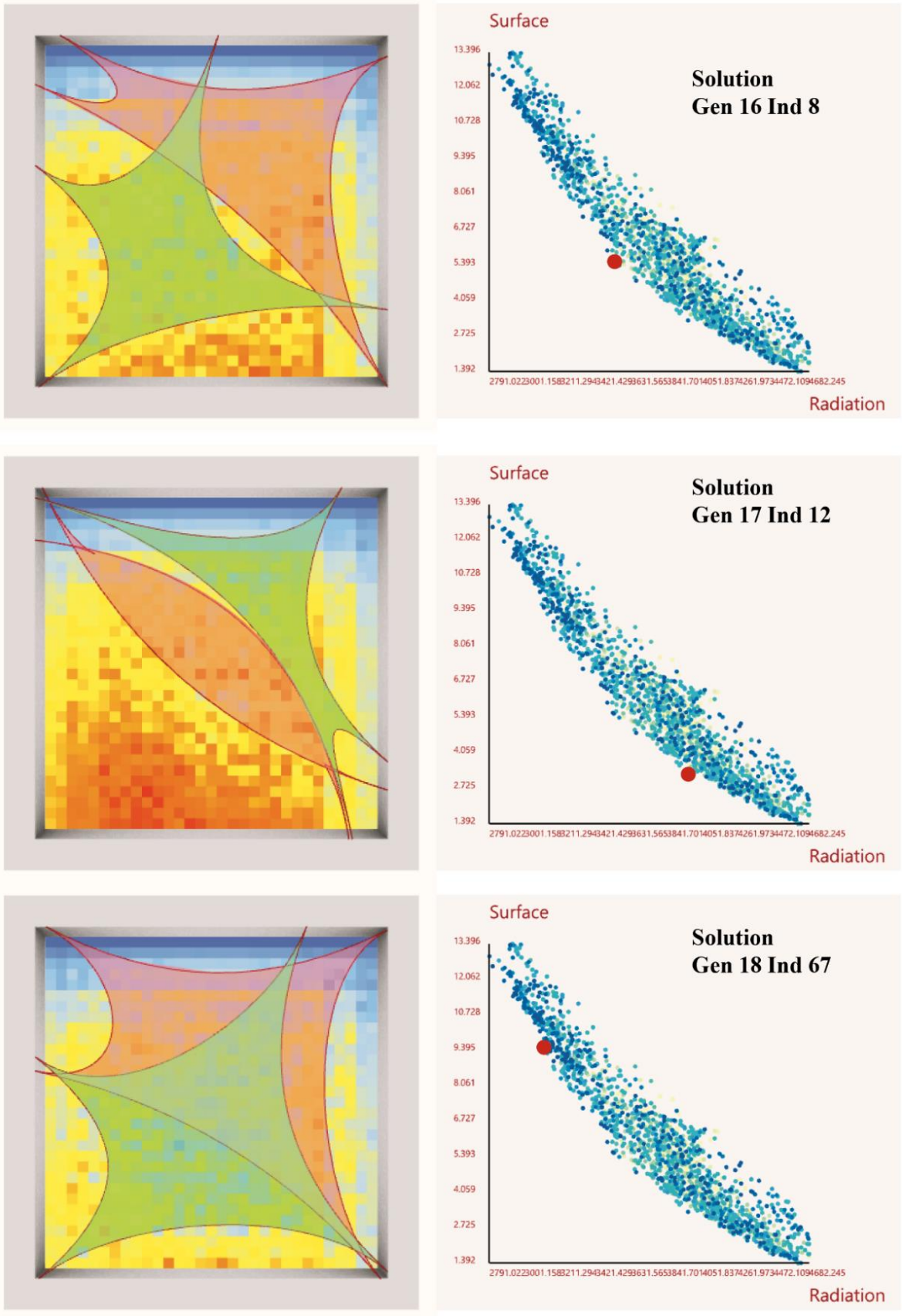


Figure 123: Trois solutions sur le front de Pareto

Le cas du Tribunal des affaires sociales de Winterthur

Architecture Studio a proposé pour un concours pour la conception d'un tribunal un bâtiment cylindrique entièrement vitré ouvert vers l'extérieur avec un système de protection solaire adapté au climat local. Dans le cas de cette troisième expérimentation, il ne s'agit plus d'optimiser un détail de façade, mais de traiter la façade dans sa globalité. Le bâtiment proposé étant courbe et le contexte urbain hétérogène, les apports solaires sur le vitrage sont hétérogènes, la démarche ne peut être appliquée uniquement sur un détail. Winterthur est situé en Suisse, dans un climat tempéré où l'été est chaud, et l'hiver est froid. L'équipe souhaite proposer une façade paramétrique avec une casquette horizontale dont la profondeur diffère selon l'orientation et où des lames verticales varient elles aussi en profondeur (voir Figure 124).

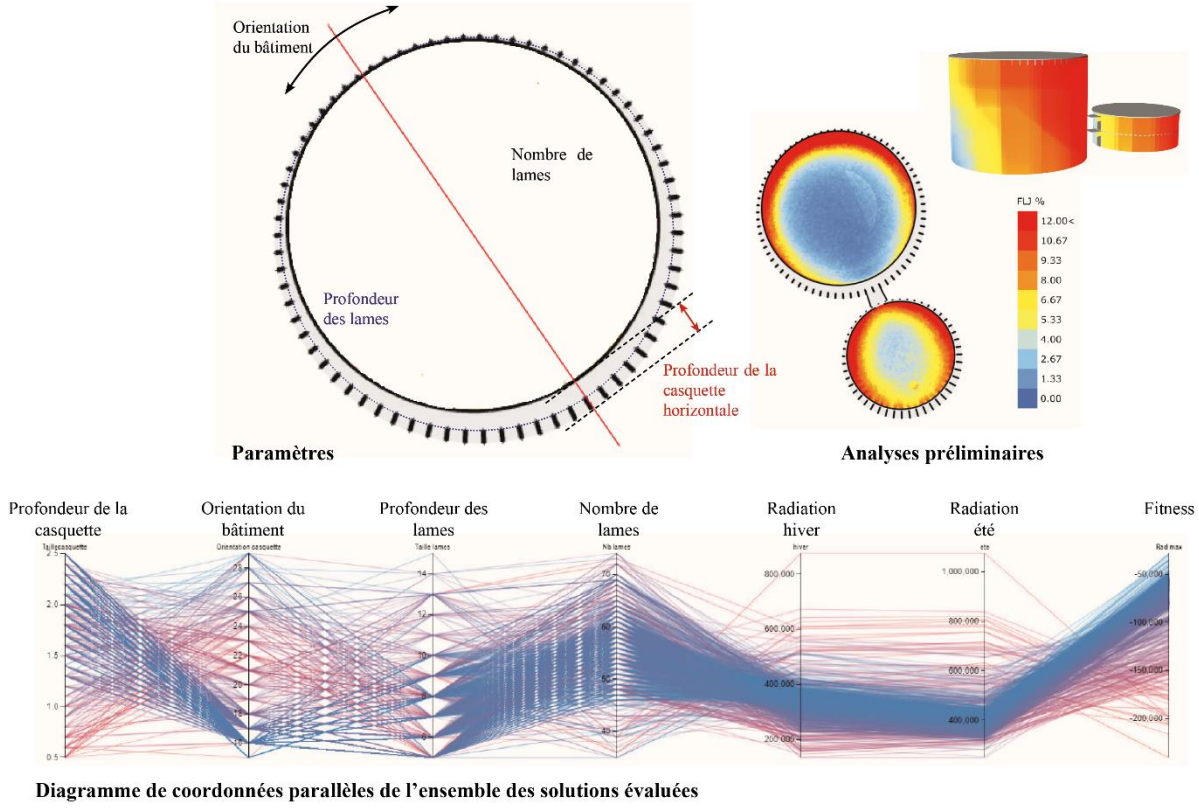


Figure 124 : Description des paramètres et de l'ensemble de solutions explorées

Nous avons été sollicités pour aider l'équipe à déterminer les paramètres qui permettraient de protéger correctement le vitrage du rayonnement solaire. Bien que l'usage de l'optimisation n'ait pas été explicitement demandé, nous avons réalisé une optimisation multicritère où les deux critères ont été agrégés : heures d'ensoleillement en hiver – heures d'ensoleillement en été. Nous n'avons pas pu accéder aux données météo de la ville de Winterthur, donc seul le rayonnement direct a pu être étudié. 1050 solutions au total ont été

évaluées pour un espace de 151 536 solutions (soit 0,7 % de l'espace de solution). A l'aide du solveur Galapagos, 10 générations de 50 individus (avec un boost x2 sur la première génération) ont été calculées avec l'algorithme génétique en 35 heures.

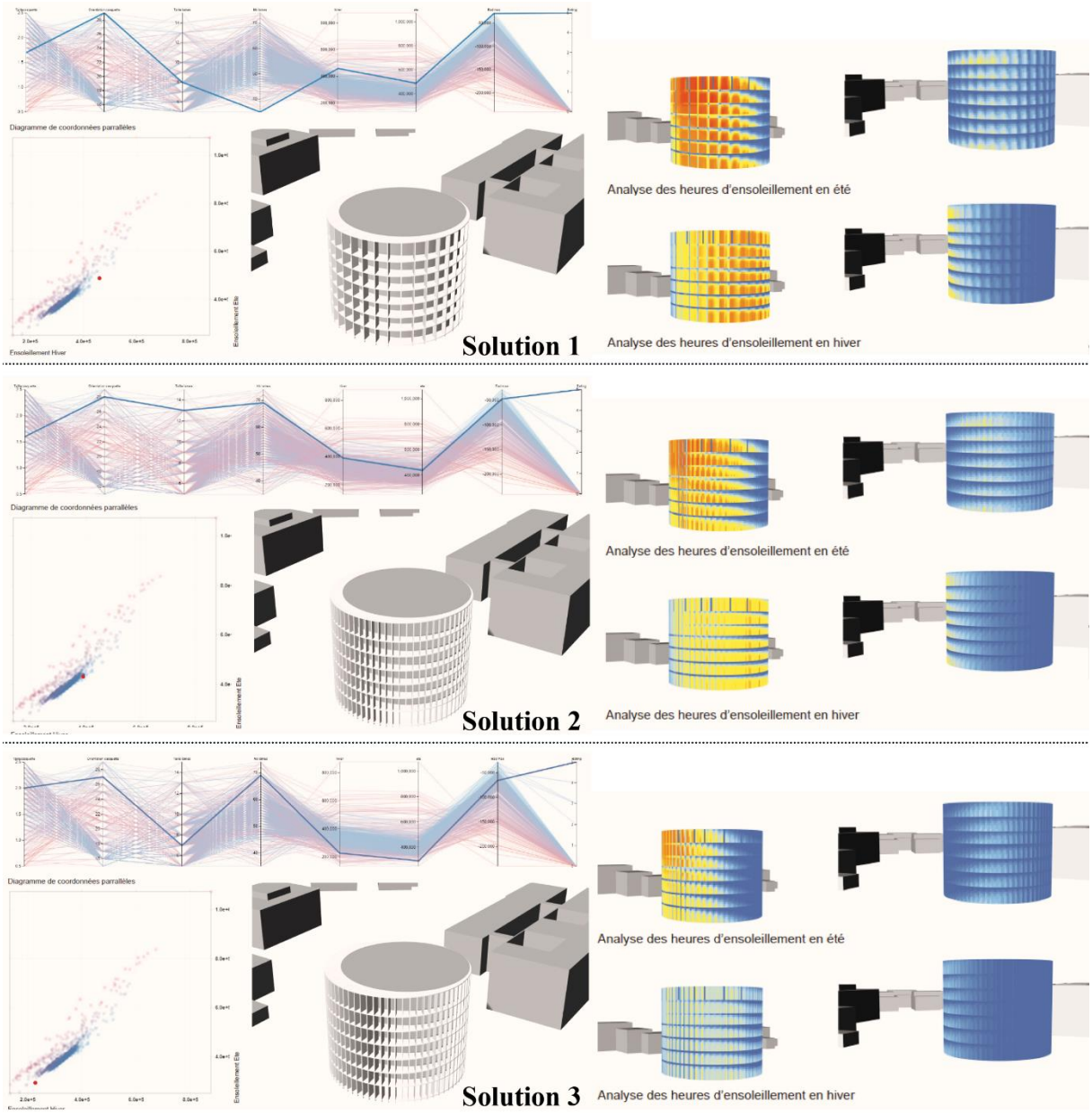


Figure 125: Trois solutions issues du front de Pareto présentées aux architectes

Plusieurs solutions sur le front de Pareto présentées en Figure 125 ont été sélectionnées, exportées sur Rhinocéros® et transmises aux architectes avec le résultat de leurs évaluations. Cependant, toutes ces façades avaient le défaut d'avoir une répartition de leur apports solaires très hétérogènes. Avec ce système, il n'y avait pas suffisamment de protection au Nord à moins

d’utiliser une très grande quantité de lames. Le problème a donc été formulé un peu trop rapidement et aurait mérité des analyses préliminaires plus poussées.

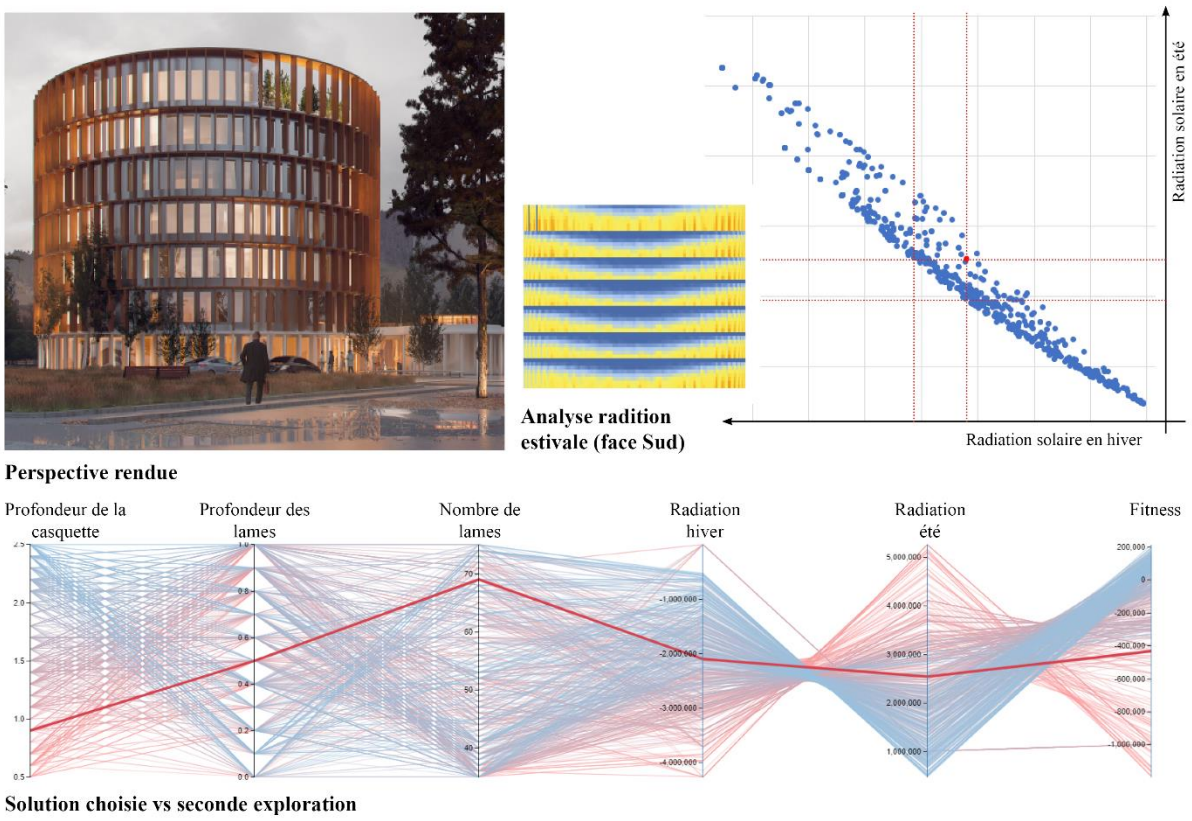


Figure 126 : Solution rendue et exploration post-concours

Nous n’avons pas eu de retours de l’équipe sur les résultats de cette exploration. Plusieurs semaines plus tard, nous avons été sollicités juste avant le rendu, pour faire des analyses d’ensoleillement et de lumière de la solution qui a été choisie pour le rendu du concours. Le système avait été adapté par les architectes avec une casquette au Nord et des lames avec des tailles qui restent inchangées, ce qui permet de résoudre le problème de l’exposition au Nord. Bien après le rendu du concours, nous avons relancé un algorithme en adaptant le modèle paramétrique à la solution finalement proposée pour mesurer son efficacité, c’est-à-dire voir s’il existe une solution qui protège aussi bien du soleil en été sans perdre des apports en hiver. On observe sans surprise sur le graphique présenté en Figure 126 que la solution choisie (point rouge) est assez éloignée du front de Pareto.

Pour explorer une façade dans sa globalité, les temps de calculs sont plus importants, mais ils restent raisonnables pour des critères comme l'ensoleillement. A cette échelle-là, il est difficile d'intégrer des critères comme la lumière en phase concours sans utiliser de modèles de substitution. Nous n'avons d'ailleurs jamais pu réaliser une telle expérimentation dans un contexte de pratique professionnelle.

Finalement, la question de l'utilité des algorithmes d'optimisation dès la phase de concours pour la conception des enveloppes se pose. Si une enveloppe peut être optimisée, elle pourra toujours l'être dans les phases ultérieures, comme nous le verrons plus tard. Ce qui est important à cette phase est de s'assurer que l'image de la façade présentée au client peut respecter les exigences du programme même si des adaptations à la marge seront nécessaires. Contrairement à la morphologie du bâtiment et son orientation, les enveloppes peuvent encore évoluées après le concours.

Il semble préférable pour les questions de conception de l'enveloppe uniquement, de déployer la méthode sur plusieurs phases. Les analyses préliminaires, le modèle paramétrique et les études de sensibilité des paramètres peuvent être réalisés en phase esquisse. L'exploration à proprement parler avec les algorithmes génétiques et le post-traitement des résultats peuvent être conduits dans les premières phases de développement du projet.

L'ENAPE pourrait aussi être utilisée avant la recherche de solution architecturale, lors de l'étape d'études générales du processus de conception, en utilisant des paramètres génériques de l'enveloppe (ratio de surface vitrée, orientation des protections solaires, matériaux) afin de traduire les exigences numériques du programme en contraintes géométriques. N'ayant jamais été sollicités si tôt dans le processus de conception, nous n'avons pas pu faire ce type d'expérimentations. Cependant, nous avons été souvent sollicités sur des problèmes où l'usage de l'optimisation n'est pas la méthode la plus efficace pour trouver une solution performante rapidement. C'est notamment le cas des enveloppes où les apports solaires sont répartis de façon hétérogène sur les éléments vitrés du bâtiment et où l'architecte souhaite utiliser des effets de texture particulier comme des effets aléatoires ou des effets de dégradés.

La préférence pour la géométrie informée

La méthode de la géométrie informée est rapidement apparue comme particulièrement adaptée pour répondre à ce type de problème en phase de concours.

Le Learning center de l'ESTP

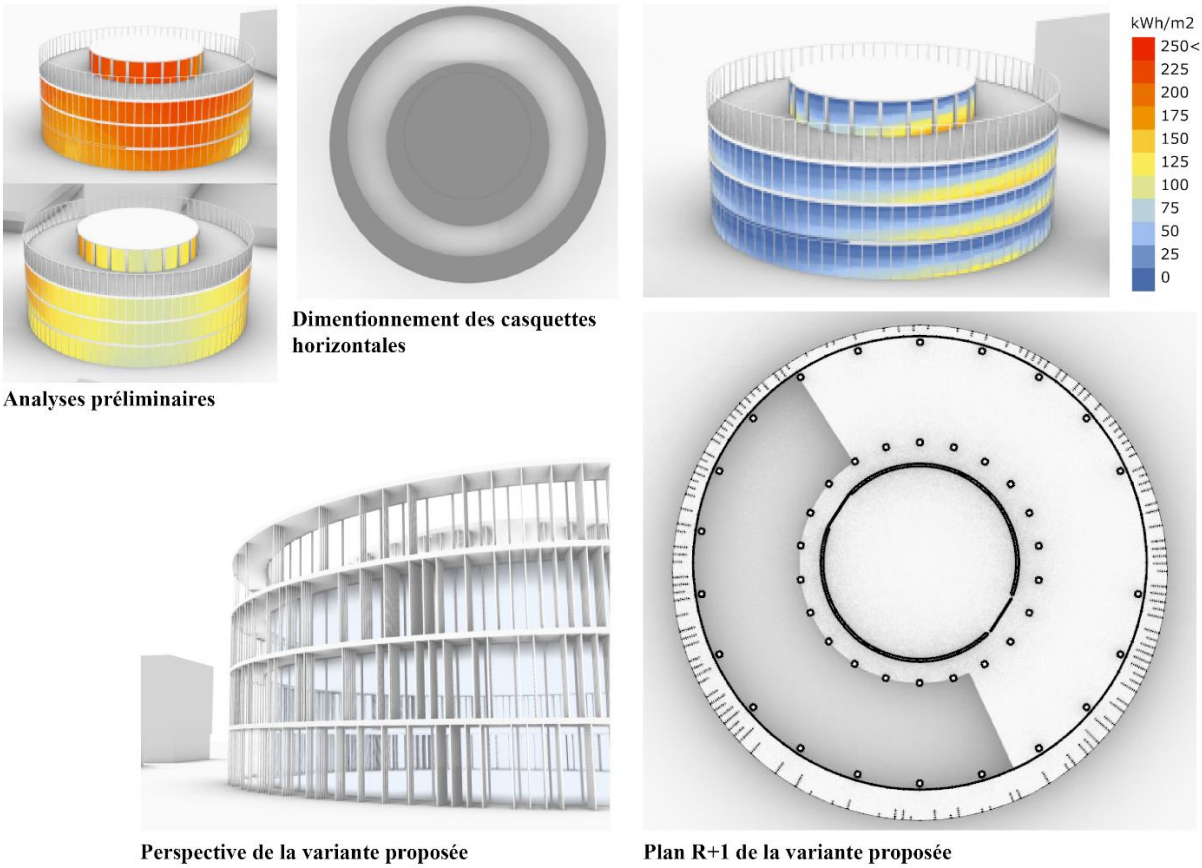


Figure 127 : Capture d'écran des différentes analyses et formes générées

De nouveau pour le concours du campus de L'ESTP, Architecture Studio a proposé pour le learning center un bâtiment cylindrique complètement vitré plomgeant les espaces de travail dans la végétation. Pour ce projet, nous avons rencontré un problème faussement similaire à celui du tribunal des affaires sociales de Winterthur. Ici, les architectes souhaitaient créer un effet aléatoire (dans la disposition et la dimension des lames). Cette exigence rend difficile l'usage de l'optimisation, car elle implique soit d'avoir deux paramètres par lames (un pour définir la présence ou non de la lame, et un second pour définir sa largeur, ce qui ferait exploser la combinatoire) ; soit d'utiliser seulement deux paramètres qui seraient des graines d'aléatoire (un pour une fonction qui supprime aléatoirement des lames, et un pour une autre fonction qui change l'ordre des valeurs d'une liste de largeur pour les lames). Cependant, les fonctions aléatoires empêchent les algorithmes d'optimisation (fondés sur des logiques statistiques) de trouver des corrélations entre les paramètres et les critères, ce qui empêche les algorithmes de converger. Une autre méthode a donc dû être utilisée.

Le problème a été traité en deux étapes. La première étape a consisté à réaliser une petite étude paramétrique pour le dimensionnement de la casquette (une dizaine d'évaluation a été

réalisée). Les apports solaires sur le vitrage restant hétérogènes, une seconde étape où la méthode de la géométrie informée a été utilisée pour l'allocation des lames et leur dimension. Un algorithme pseudo-aléatoire avec un coefficient de chance de disposition de la lame et de sa taille a été développé. Plus ce coefficient est important, plus il y a de chance qu'une lame soit générée et que cette lame soit large. Le coefficient est directement fonction de la radiation solaire estivale moyenne reçue par le vitrage placé juste derrière la lame ($Mrad_l$). Ainsi pour chaque lame, un coefficient de chance (C_l) différent est attribué, il est défini comme suit : $C_l = \beta \times Mrad_l$ où β est un paramètre qui permet de faire varier les coefficients de toutes les lames de façon proportionnelle. Plusieurs évaluations sont ensuite effectuées en faisant varier le paramètre β jusqu'à obtenir un niveau de protection solaire satisfaisant. Plusieurs variantes sont ensuite générées en faisant varier la graine d'aléatoire de l'algorithme, elles sont ensuite exportées et présentées à l'équipe d'architectes.

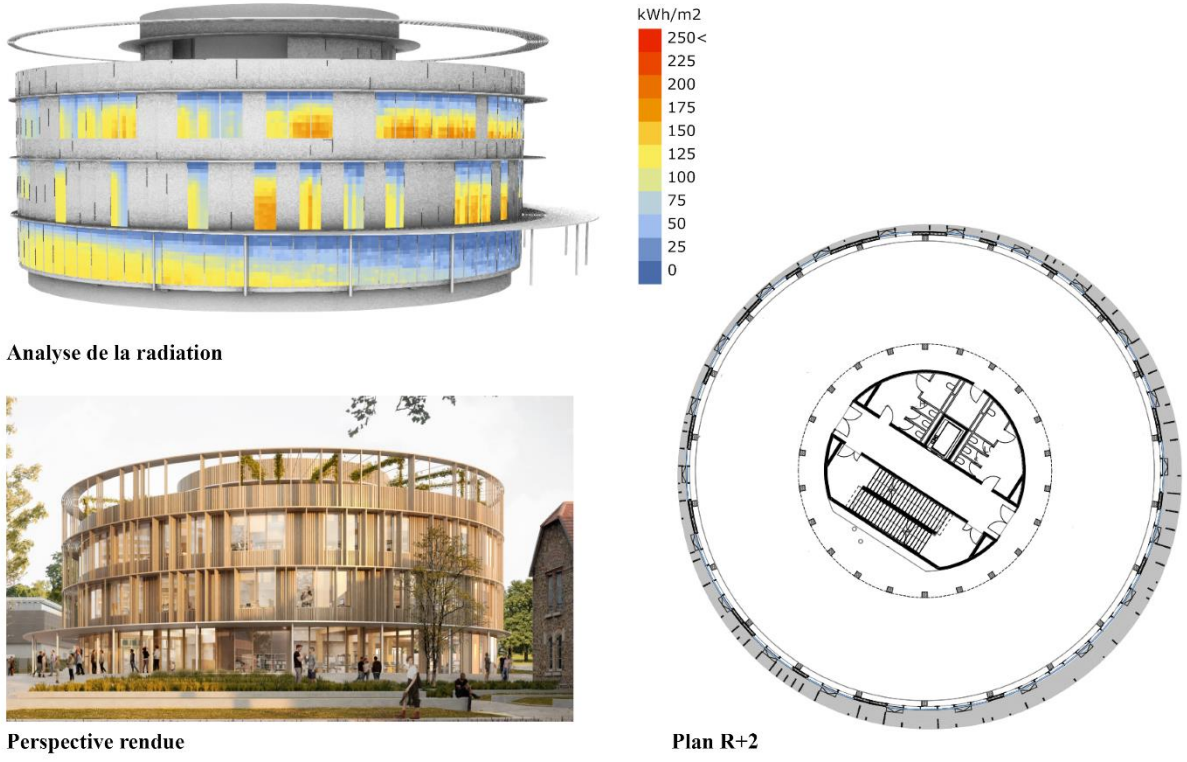


Figure 128 : Solution choisie pour le rendu de concours

Par la suite, pour répondre à des critères économiques, la façade qui devait être complètement vitrée a été en partie opacifiée. Le système de protection solaire a pu être allégé, des lames ont été supprimées manuellement faute de temps. Comme on peut l'observer avec les représentations graphiques de la Figure 128, les dimensions de la casquette horizontale ont été conservées, la quantité de lames a nettement été diminuée et celles-ci servent finalement

plus à l'intégration des éléments opaques qu'à la protection contre les rayonnements solaires des éléments vitrés.

Dès lors que les solutions générées sont modifiées, elles ne peuvent plus être considérées comme des solutions optimisées. Si les conditions du problème évoluent, sans véritable compréhension de ce qui permettait la performance (ce qui est quasiment impossible lorsqu'on travaille sur des effets aléatoires), les calculs doivent nécessairement être reconduits. A cette phase, l'objectif reste de remporter le concours. Bien souvent, le coût du projet et la qualité des perspectives sont des critères qui passent avant la performance à cette étape.

La Tour Axian à Antanarivo

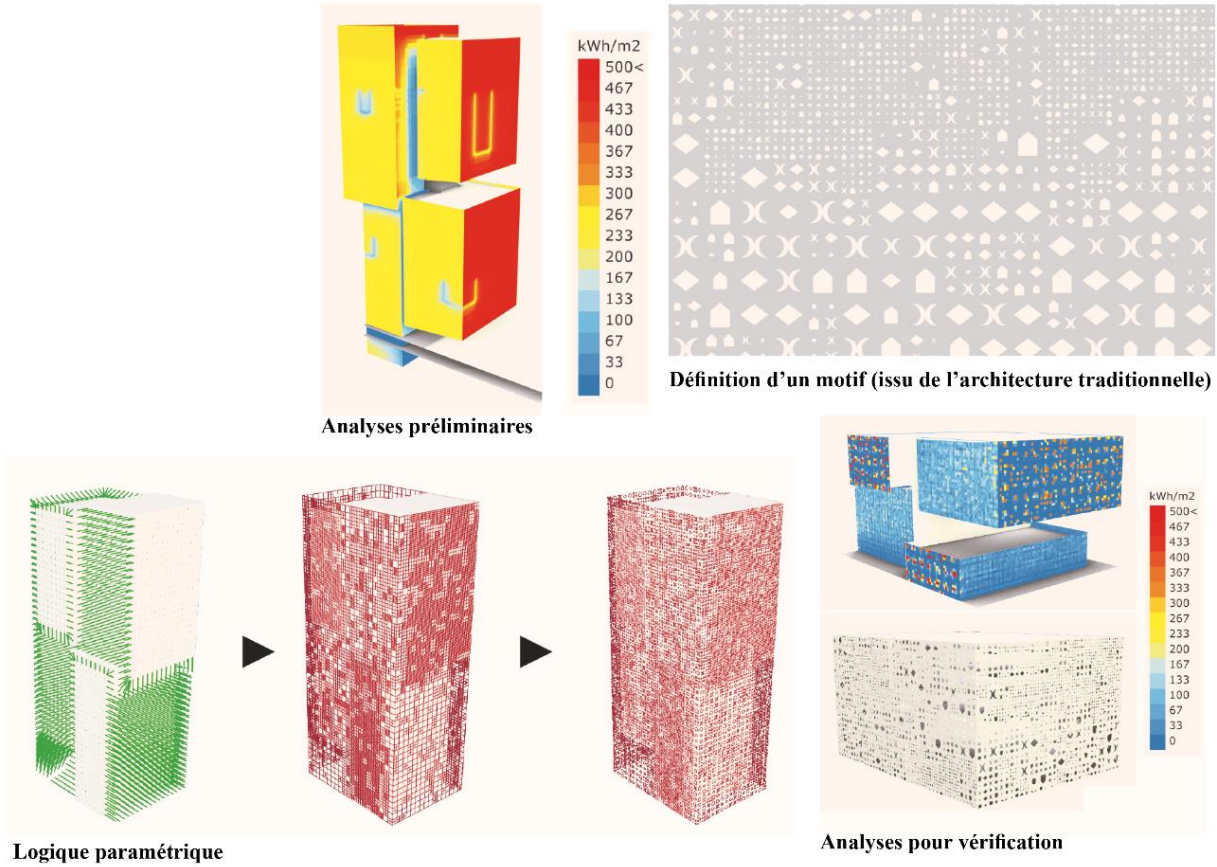


Figure 129 : Description du système de protection solaire proposé

Lors du concours pour une tour de bureaux à Antanarivo, les architectes d'Architecture Studio se sont inspirés de patterns rencontrés dans l'architecture vernaculaire de l'île de Madagascar pour concevoir une enveloppe adaptée au climat local. Pour ce second exemple, des allers-retours ont été réalisés jusqu'au rendu du concours. La première idée était de proposer une peau percée à l'aide de ces différents motifs pour protéger une tour composée d'une agrégation de blocs de dimensions différentes. Nous avons proposé d'utiliser la diversité créée

par les blocs pour faire varier la taille des trous en façade. Pour éviter un ensoleillement direct et protéger des surchauffes estivales, les protections solaires situées à proximités de la tour devaient être composées de petits trous, à l'inverse, les parties plus éloignées pouvaient être percées avec des éléments plus larges. La méthode de la géométrique informée a été utilisée. La distance entre les deux peaux a été calculée tous les mètres carrés. A partir de ces informations, une discrétisation adaptative de la seconde peau a été générée et différents motifs ont été appliqués sur les peaux de tailles différentes à l'aide d'une fonction aléatoire. L'efficacité du système de protection solaire a ensuite été vérifié à l'aide d'une analyse de la radiation et d'une étude de l'autonomie en lumière du jour dans les espaces intérieurs pour vérifier la qualité de la lumière dans les bureaux. L'ensemble des résultats de l'étude ont été transmis aux architectes à l'aide des images présentées en Figure 129.

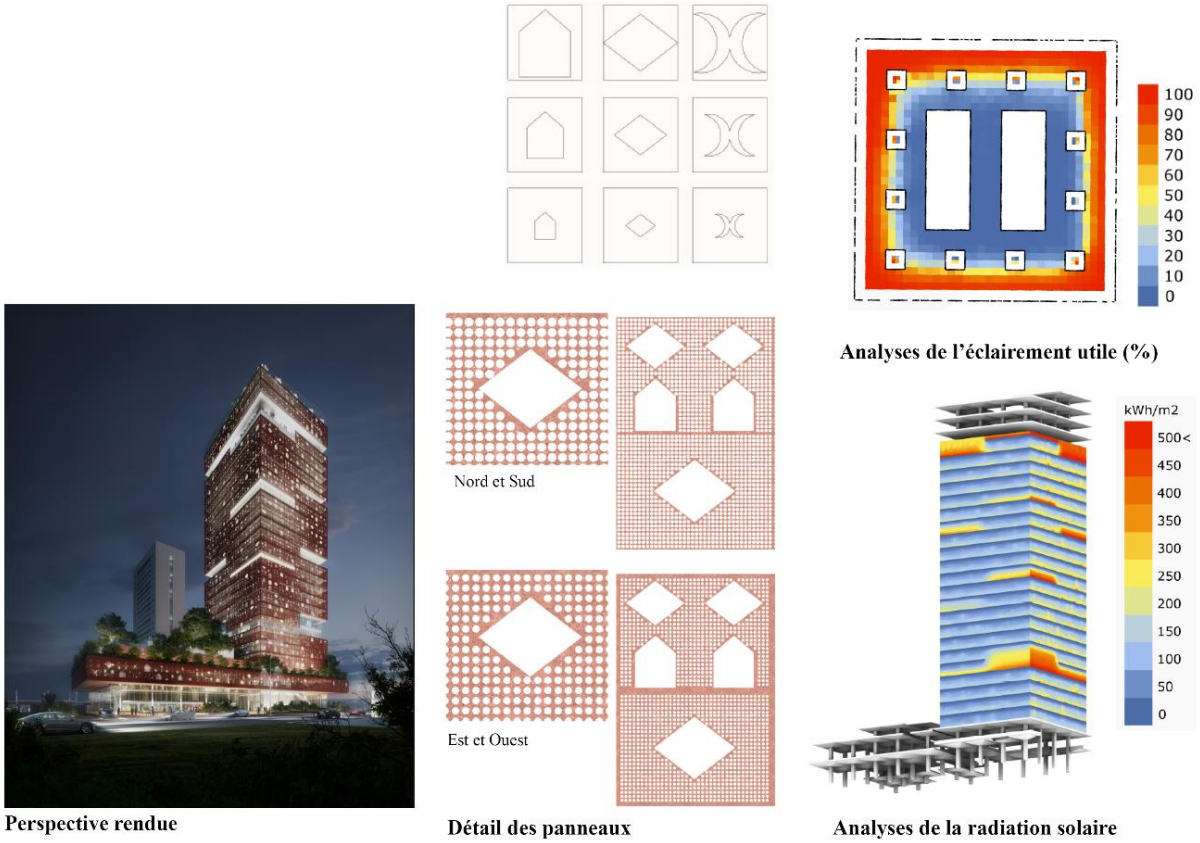


Figure 130 : Description de la solution choisie pour le rendu

Comme on peut le voir sur la perspective de rendu du concours présentée Figure 130, la façade résultante de ce processus de conception numérique a plu aux architectes, le système des blocs a évolué faisant disparaitre avec lui toute la logique générative derrière la façade. Ce phénomène est très courant dans le processus de conception. Certains éléments apparaissent au

cours du projet pour certaines raisons et sont ensuite conservés pour d'autres raisons. Pour réduire les apports solaires, des protections horizontales ont été ajoutées, et pour augmenter l'autonomie en lumière du jour, des petites perforations sont appliquées sur toutes la seconde peau.

Le quartier Média, innovation et civique à Riyad

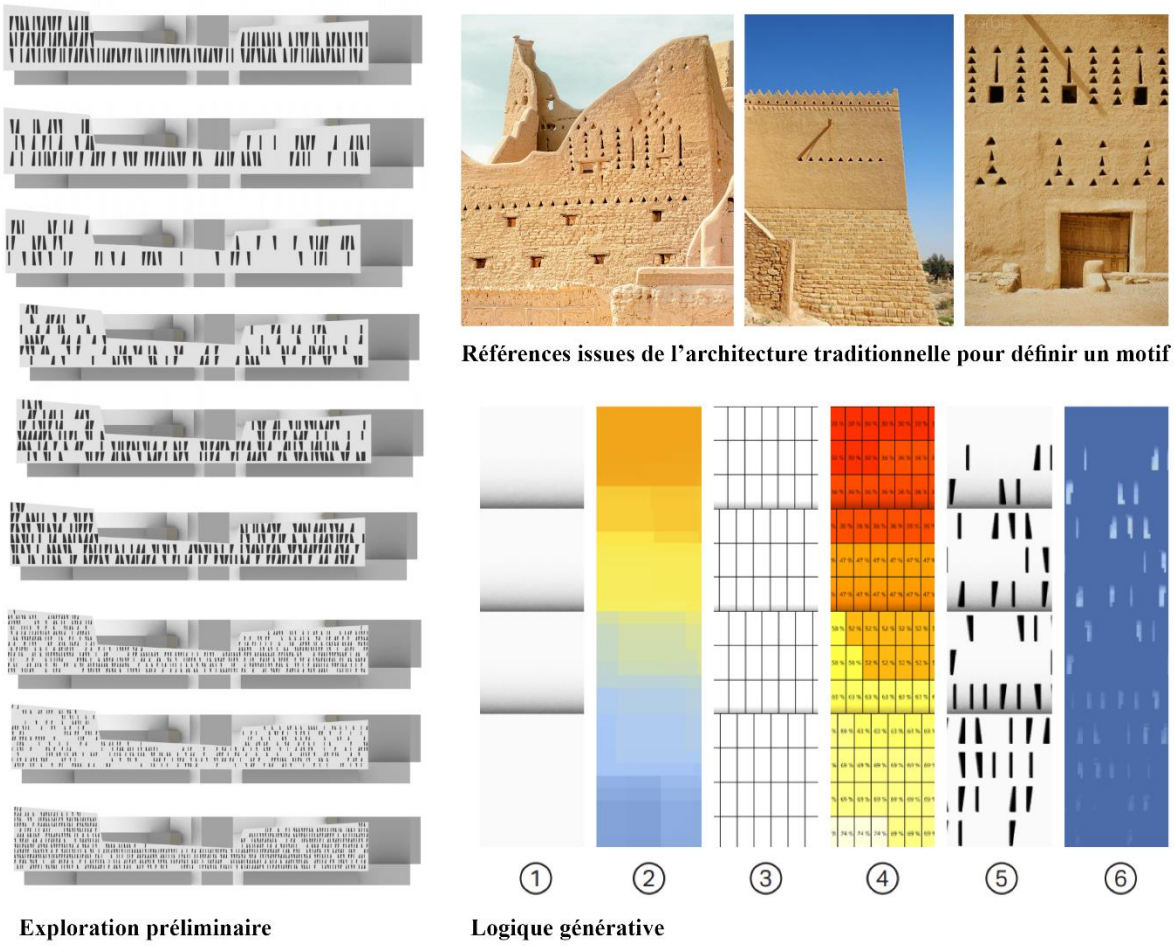


Figure 131 : Description du processus de conception

Pour le concours pour la conception d'un quartier entier à Riyad en Arabie Saoudite, les architectes d'Architecture Studio ont de nouveaux utilisés des motifs identifiés dans l'architecture vernaculaire locale pour trouver une grammaire formelle commune à l'ensemble des bâtiments qui composent le quartier qui puissent être adaptée selon les différentes orientations et expositions des enveloppes. Ces bâtiments doivent être conçus pour accueillir des logements, des bureaux et des commerces. Une première étape a consisté à utiliser un modèle paramétrique pour définir un langage formel qui pourrait servir à l'ensemble des bâtiments du quartier. Différentes propositions sont testées sur un immeuble de bureaux en

Figure 131. Ensuite, la morphologie des bâtiments en cascade a permis de créer des ombres rendant hétérogènes les apports solaires sur la façade. Ainsi, nous avons pu relier le modèle paramétrique des motifs aux résultats d'une analyse de la radiation solaire et à une fonction aléatoire avec le même principe que celui présenté pour le Learning Center de l'ESTP.

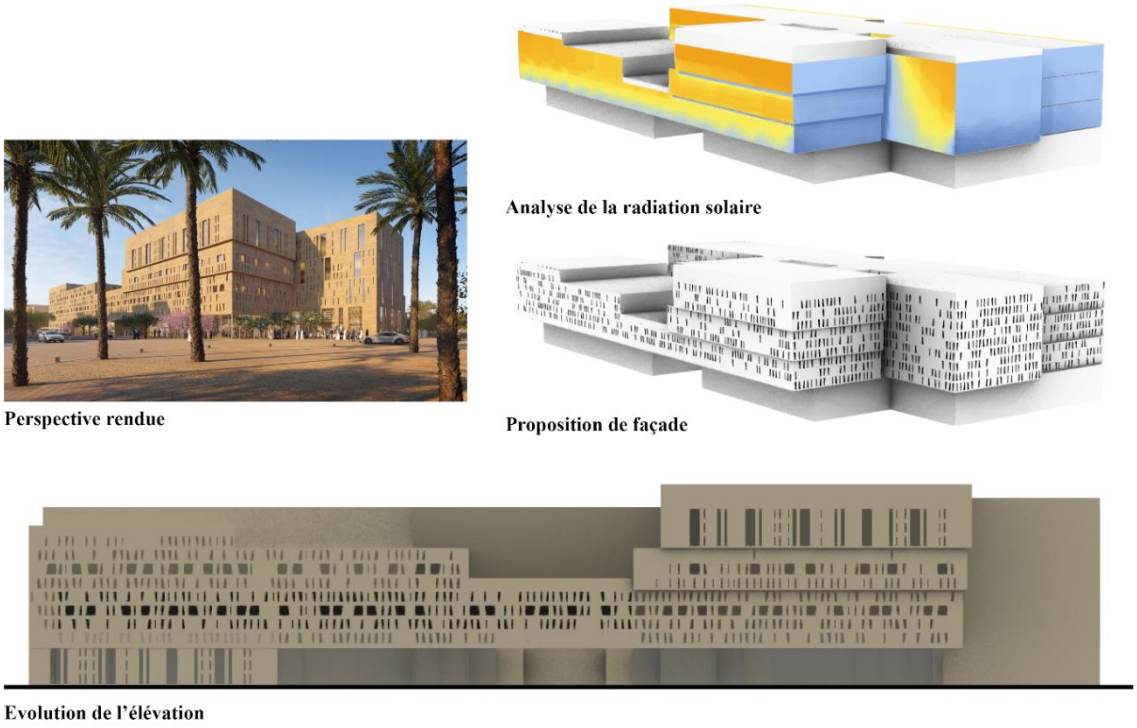


Figure 132 : Description de la solution proposée pour le rendu

Les propositions résultantes ont été transmises à l'équipe de projet. Pour plus de lumière naturelle, certains percements ont été fusionnés manuellement pour créer des ouvertures plus imposantes (voir Figure 132). Pour les autres bâtiments, leur morphologie ne se prêtait pas à la méthode de la géométrie informée mais le modèle paramétrique des motifs a été utilisé par les membres de l'équipe sur l'ensemble des façades du projet.

Finalement, lorsqu'il s'agit uniquement d'étudier le critère de la protection solaire, et si les apports solaires sont hétérogènes, la méthode de la géométrie informée est plus rapide à mettre en œuvre et plus efficace en phase esquisse lorsqu'il s'agit de proposer une façade hétérogène. Cependant, l'utilisation de cette méthode demande de la rigueur est nécessite d'être conduite jusqu'au rendu pour qu'elle est un impact sur les performances du projet. Cette méthode a d'autres avantages que la performance, les solutions générées plaisent souvent aux architectes. A noter que cette méthode ne remplace pas nécessairement l'optimisation. Lorsqu'un concours est remporté, il est possible de poursuivre l'étude avec de l'optimisation

couplée à de la géométrie informée, notamment pour intégrer d'autres critères comme la visibilité et la lumière naturelle.

Exploration en phase de développement

Plus rarement, nous intervenons en phase de développement de projet. Lors de ces phases, le concept général de l'enveloppe est déjà figé, il s'agit de définir un espace de solutions afin d'optimiser ces performances environnementales sans changer l'architecture de l'enveloppe. En phase d'avant-projet, les architectes ont souvent plus de temps, la tâche est donc plus accessible que lors du concours. Cependant, de nouveaux critères apparaissent comme la réduction des coûts et donc l'optimisation de la fabrication.

Le cas de l'Institut de recherche Faire Face d'Amiens

Pour les laboratoires de l'institut Faire Face à Amiens., Architecture Studio a proposé d'utiliser de grandes lames en aluminium aux formes triangulaires et à l'orientation variables pour pouvoir éclairer ces espaces avec de la lumière indirecte en toute saison et pour toutes les orientations. Ces grandes lames verticale à la géométrie variable dessinant des courbes en façade protègent les laboratoires de l'ensoleillement direct et favoriser la réflexion de la lumière pour un éclairage naturel indirect (voir Figure 133).

Dans un premier temps, nous avons cherché à optimiser l'orientation des lames. Nous souhaitions pouvoir interroger l'orientation de chacune des lames, soit 240 paramètres, en tenant compte de deux critères : l'ensoleillement et la lumière. Cependant, la grande majorité des solutions de l'espace de solutions contenait des lames entrant en collision et bien souvent le dessin de la courbe générée par la variation géométrique de chaque lame n'était plus visible. A défaut de trouver une méthode efficace pour gérer ces contraintes, nous avons simplifié le problème.

Nous avons restreint notre optimisation à quatre paramètres, une orientation de lames par façade. Une analyse de sensibilité a été réalisée pour la lumière naturelle. Seulement 5 angles ont été testés par façade (-60°, -30°, 0°, +30°, +60°) afin d'évaluer l'éclairement aux solstices et équinoxes à 12 h. Si l'orientation des lames a un impact sur la quantité de lumière naturelle, celle-ci reste modérée lorsque les lames sont toutes dans le même sens. Ainsi, seul l'ensoleillement direct a été pris en compte dans notre optimisation réalisée avec l'algorithme génétique du solveur Galapagos. Celui-ci a rapidement convergé vers une solution.

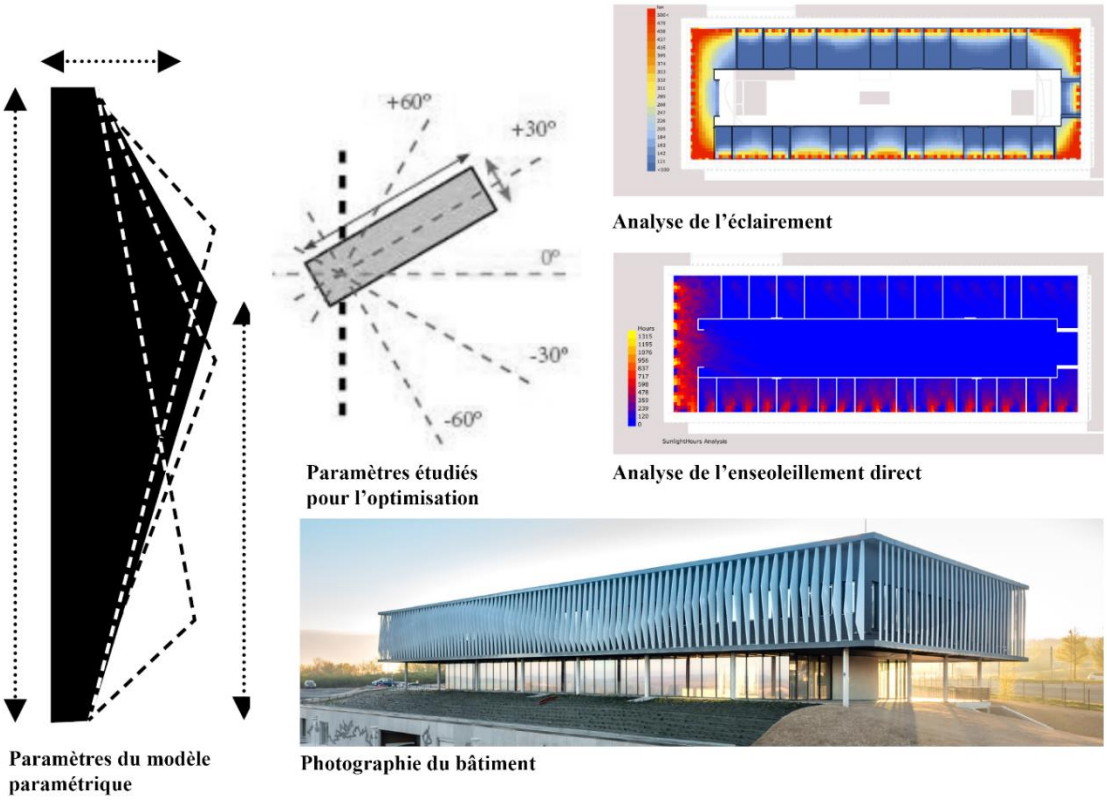


Figure 133 : Optimisation de l'orientation des lames

Dans un second temps, nous avons dû de nouveau modifier le système de protection solaire pour restreindre le nombre de types de lames pour réduire les coûts de fabrication de la façade. Pour cela, nous avons élaboré une méthode paramétrique (voir Figure 134) dans laquelle la morphologie des lames s'adapte en fonction du nombre de types maximum. Cet outil a permis de faire un compromis entre le coût de fabrication et la visibilité de la courbe dessinée par les lames.

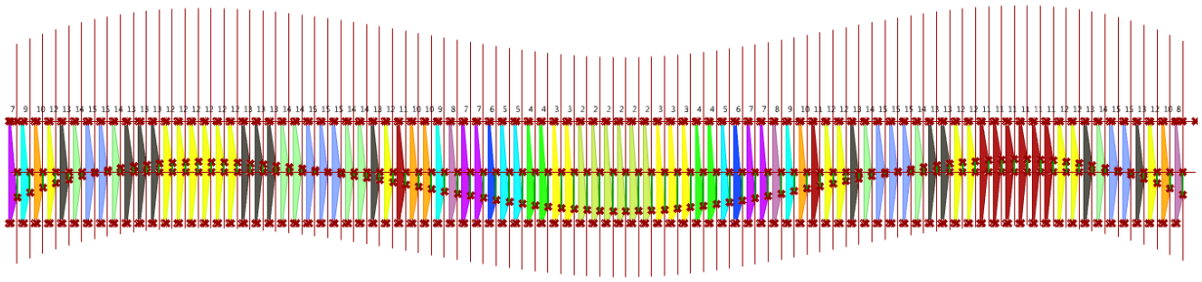


Figure 134 : Méthode paramétrique pour la réduction du nombre de type de lame

Finalement, dans cet exemple, les trois critères que sont la lumière naturelle, l'ensoleillement direct et le coût de fabrication ont pu être traités séparément avec des méthodes

numériques différentes puisqu'ils n'entraient pas ou peu en conflit les uns avec les autres compte tenu des paramètres qu'il était possible de faire varier à ce moment précis du projet.

L'immeuble Ame à Montevideo

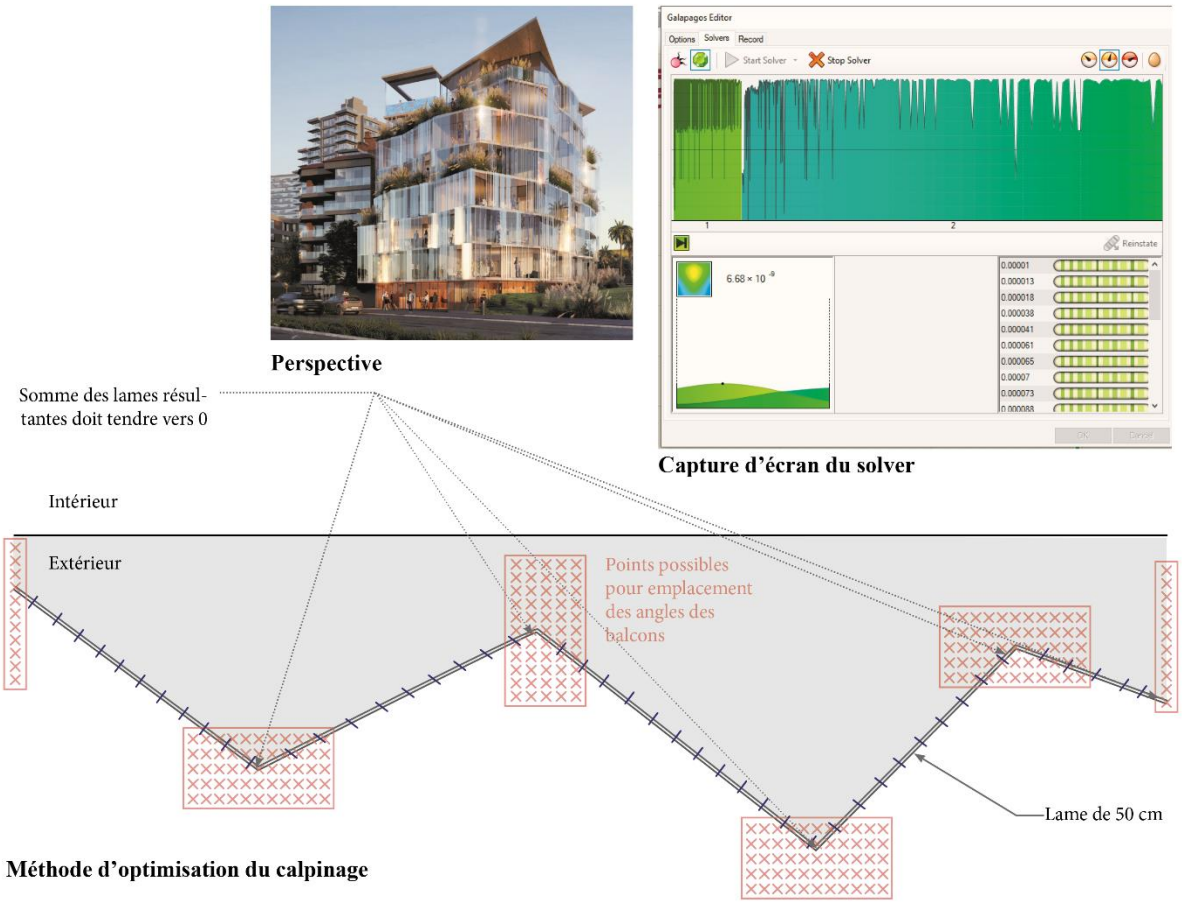


Figure 135 : Exploration de la morphologie des balcons pour l'optimisation du calepinage des lames

L'immeuble de logement Ame conçu par Architecture Studio est idéalement situé à Montevideo en Uruguay. Sa façade entièrement vitrée offre des vues exceptionnelles à ses habitants mais nécessite une protection solaire adaptée. Des lames en verre sérigraphiées sont utilisées pour protéger du rayonnement solaire et servir de garde-corps sur les balcons à géométrie variable. Dans un premier temps la géométrie des bacons a été optimisée afin de pouvoir recouvrir l'ensemble de la façade avec un seul type de lame. Pour cela l'algorithme de recuit simulé du solveur Galapagos (qui s'est révélé plus efficace dans ce cas) a été utilisé. Le processus d'optimisation est décrit en Figure 135. Les sommets des triangles qui forment les balcons sont les paramètres. Ils peuvent varier en *x* et en *y* sur une zone restreinte dessinée en rouge. La fonction d'évaluation est définie par la somme des largeurs des petites lames

résultantes à la fin de chaque segment constituant la limite des balcons. Pour obtenir une solution adaptée, cette somme doit être nulle.

Dans un second temps, une approche paramétrique proche de la méthode de la géométrie informée a été utilisée pour la sérigraphie des lames (voir Figure 136). En sérigraphiant les lames vitrées, il est possible de les rendre plus ou moins opaque. Ainsi, cinq niveaux d'opacité ont été définis (85%, 70%, 55%, 40%, 25%). Une fois la morphologie des balcons fixée, une analyse de la radiation solaire en façade a permis d'identifier les zones les plus exposées aux rayonnements solaires. Les lames ont pu ensuite être réparties depuis le logiciel Rhinocéros® sur différents calques directement reliés au script Grasshopper® pour l'analyse du rayonnement solaire, ce qui a permis d'itérer le processus jusqu'à obtenir une enveloppe suffisamment protégée. Cette méthode a notamment permis de conserver le contrôle sur l'effet aléatoire produit en façade.

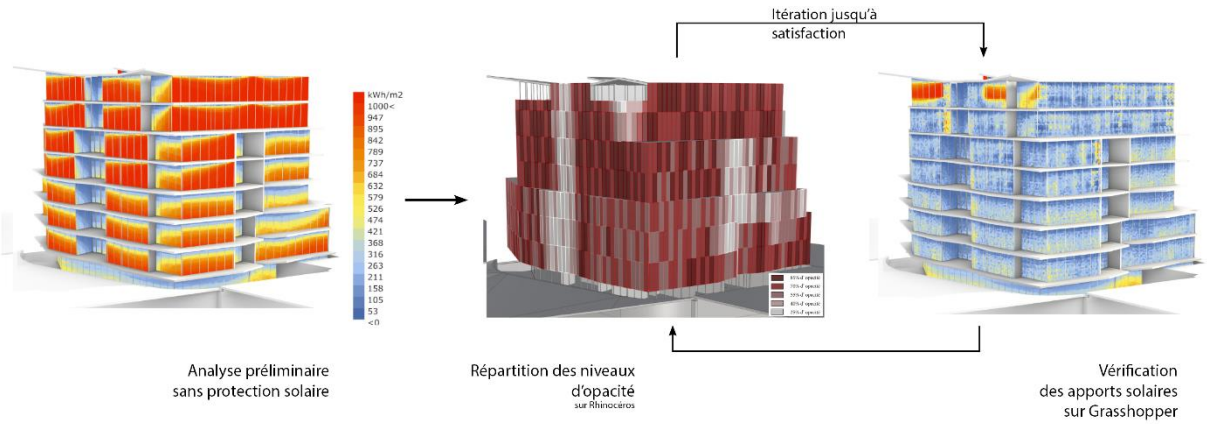


Figure 136 : Méthode paramétrique pour la répartition semi-aléatoire des différents types de lames

A l'inverse de l'exemple précédent, l'usage de l'optimisation était ici approprié pour le critère de fabrication des éléments architecturaux, tandis qu'une méthode paramétrique été plus adaptée pour l'intégration du critère d'ensoleillement. Le critère de fabrication a du nécessairement être traité avant celui de l'ensoleillement, ce qui n'était pas le cas pour les laboratoires de l'Institut Faire Face.

Le grand phare d'Issy

L'immeuble de logements à Issy-les-Moulineaux conçu par Architecture Studio s'élève comme une figure de proue au dessus de la gare du grand Paris Express d'Issy-RER. Des lames verticales colorées sont utilisées pour protéger sa façade du rayonnement solaire et donner une identité graphique au bâtiment. Compte tenu du caractère résidentiel de ce projet, il est

important que ce dispositif n'impacte pas trop la visibilité depuis les logements. Ainsi, une étude paramétrique a été réalisée pour définir des règles sur la disposition et l'orientation des lames situées devant les ouvertures pour réduire au maximum leur impact sur la visibilité tout en conservant leur rôle de protection contre les rayonnements solaires estivaux. Dans un premier temps, des optimisations multicritères pour interroger l'emplacement et l'orientation d'une lame ont été réalisées pour différentes orientations du bâtiment et différentes typologies de fenêtres. Ces explorations ont permis de déterminer que pour certaines orientations du bâtiment, les solutions se trouvant sur le front de Pareto étaient toutes situées dans la même zone, comme le montre le graphique en nuage de points présenté en Figure 137.

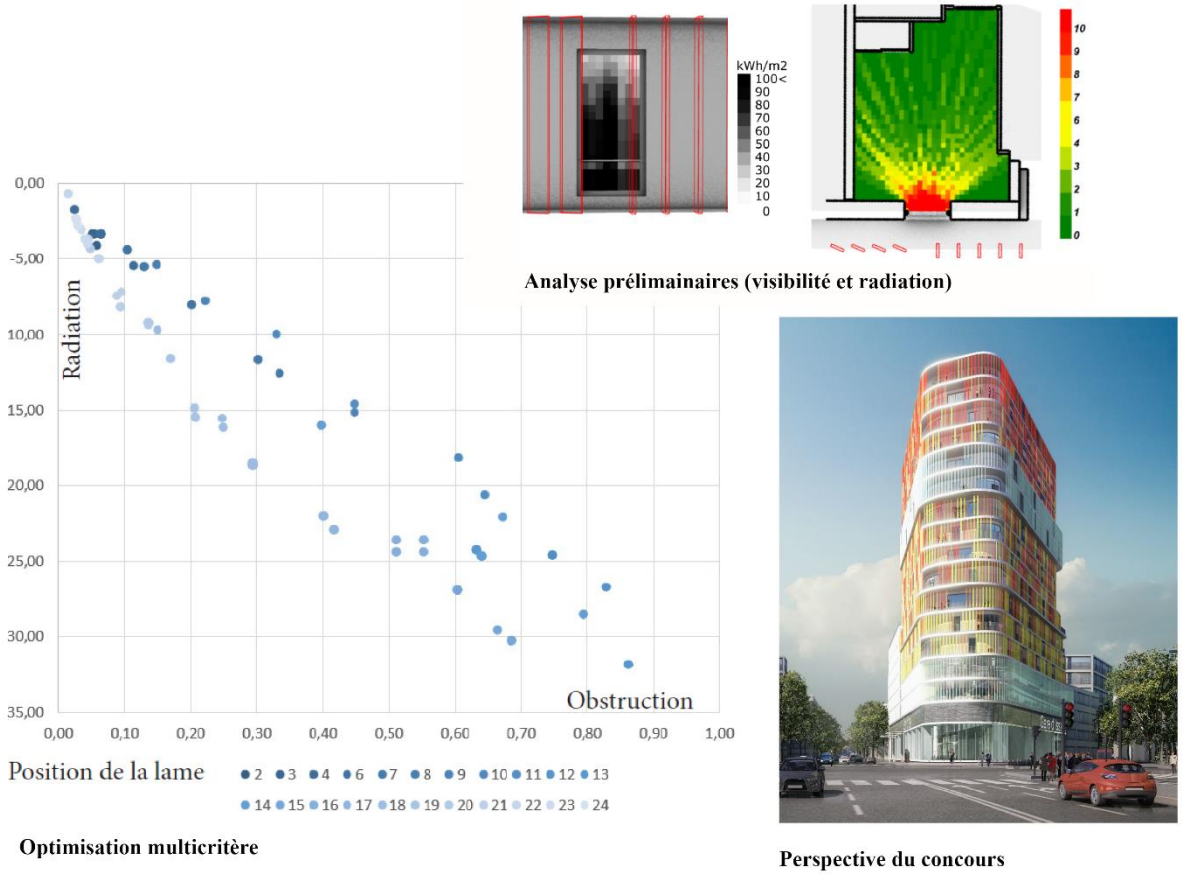


Figure 137 : Optimisation multicritère pour identifier l'emplacement idéal des lames

Dans un second temps, des études paramétriques pour analyser plus finement l'impact de l'orientation des lames lorsqu'elles sont positionnées aux emplacements identifiés lors de la première étape ont été réalisées pour chaque orientation du bâtiment. Quatre grandes conclusions illustrées en Figure 138 ont pu être tirées de cette étude :

(1) L'emplacement de la lame devant les ouvertures dépend de la forme de la pièce. Si la protection solaire n'est pas indispensable, il est préférable de disposer la lame selon la forme de la pièce.

(2) Si la protection solaire n'est pas indispensable, l'orientation de la lame optimale pour réduire l'obstruction dépend de son emplacement. Si elle est disposée au centre de la fenêtre, il est préférable de l'orienter perpendiculaire au vitrage. Plus on l'éloigne du centre, plus il convient de la faire tourner légèrement (environ 30°). Les lames qui ont un impact sur la vue qui ne sont pas située devant l'ouverture doivent être orientée au-delà de 30°.

(3) Pour les ouvertures orientées au Sud-Est non protégées, les lames disposées à droite de la fenêtre sont plus efficaces que celles situées à gauche pour protéger du soleil. A l'Ouest, il n'y a pas de différence significative entre les deux positions, il est donc préférable de la disposer selon la géométrie de la pièce.

(4) L'orientation des lames qui permettent de mieux protéger du soleil à l'Ouest comme au Sud-Est est aussi celle qui impacte le plus la qualité de la vue.

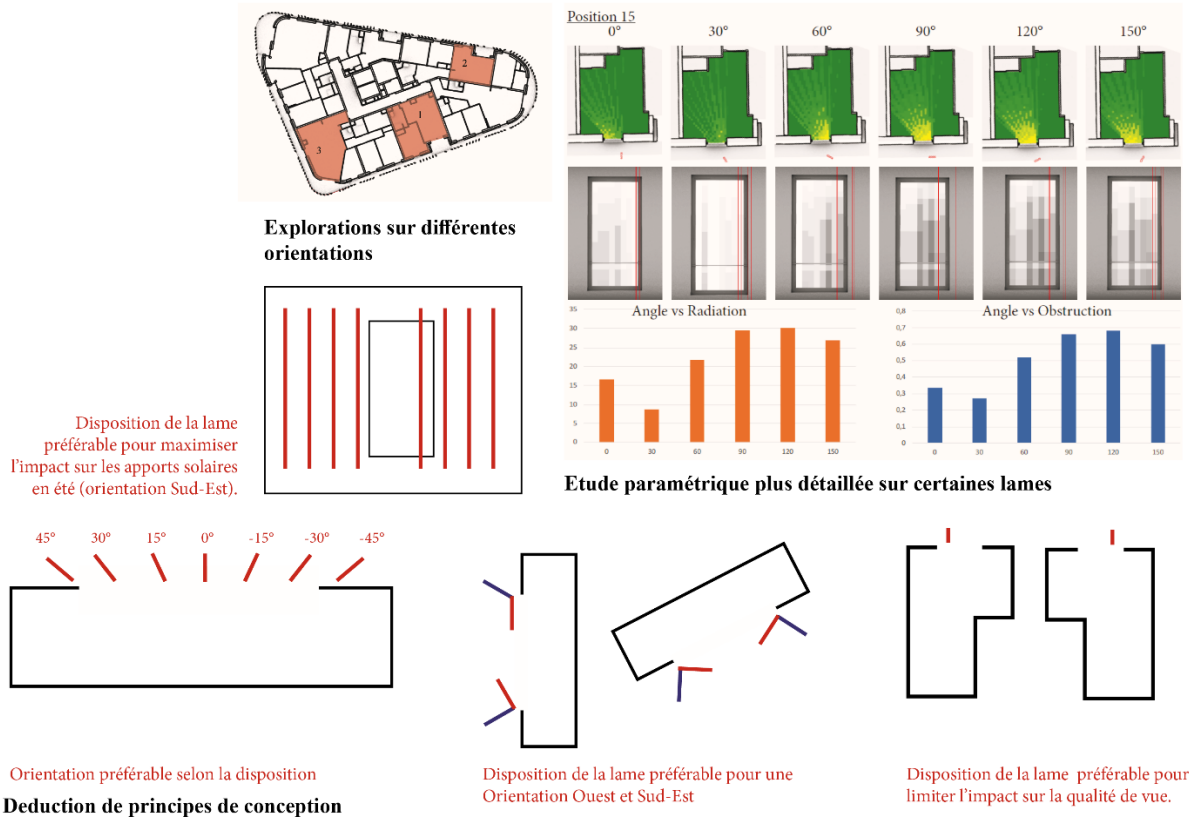


Figure 138 : Conclusions de l'étude paramétrique

Plutôt que de chercher à optimiser l'ensemble du dispositif en une unique exploration, il est souvent plus efficace pour les enveloppes de réaliser plusieurs explorations, une pour chaque cas de figure (orientations, masques urbains, détails de fenêtres) pour éviter l'explosion combinatoire. Cependant, démultiplier le nombre de calculs à réaliser finit par être très chronophage, ce qui explique pourquoi ce n'est pas réalisable dans les phases de concours. Pour exploiter les résultats, deux solutions sont possibles : soit on identifie plusieurs solutions pour chaque situation étudiée afin de pouvoir reconstituer une façade hétérogène, soit on définit des règles de conception à partir des résultats des analyses comme pour ce dernier exemple.

2.2.2. L'enseignement des expérimentations n'ayant pu aboutir

Dans cette section, nous faisons la description de certaines des interventions sur des projets qui, à première vue, semblaient être de parfaits candidats pour une expérimentation avec des algorithmes d'optimisation, et qui ne se sont finalement pas révélés adaptés. Nous cherchons ici à comprendre pourquoi et comment remédier à ces blocages.

Savoir identifier un problème (d'optimisation)

Si certains architectes sont réticents à l'idée d'utiliser des algorithmes d'optimisation, d'autres sont très convaincus par la conception générative. Ils y voient un outil puissant car capable à la fois d'assister la créativité et de servir de « *caution scientifique* » pour justifier des décisions auprès de la maîtrise d'ouvrage. Cependant le design génératif ne se prête pas à toutes les situations.

Le campus de l'ESSEC 2023, Research Tower (CGY1)

Ce fut notamment le cas pour la rénovation d'une enveloppe pour la Research Tower du campus de l'ESSEC pour laquelle nous sommes intervenu en phase d'avant-projet sommaire. Il s'agit de la conception d'une enveloppe double peau. La peau extérieure est faite de lames amovibles horizontales vitrées en partie sérigraphiées pour le contrôle du rayonnement solaire. La peau intérieure est constituée de panneaux pleins et de panneaux vitrés, de 2 tailles différentes, petite et grande. La trame du bâtiment impliquait de regrouper les panneaux en modules, chacun d'entre eux étant constitué de 2 grands panneaux et de 2 petits. Pour assurer la flexibilité de la façade, 3 des 4 panneaux devaient être vitrés (2 grands et 1 petit), ce qui laisse 12 possibilités de modules (voir Figure 139).

L'architecte souhaitait pouvoir jouer sur la disposition de ces modules de façon que leur organisation paraisse aléatoire mais permette une répartition de l'éclairage naturel optimale. La

quantité, la forme et la fréquence des panneaux vitrés étant constantes avec ce système et, en l'absence de masques solaires, les apports en rayonnements lumineux étant homogènes sur la façade, la disposition des modules n'a, ici, pas d'impact sur la répartition de l'éclairage naturel. Dans le cas de ce projet, l'enjeu se trouvait surtout sur la peau extérieure pour éviter les surchauffes estivales tout en conservant un maximum d'éclairement naturel. Celle-ci étant entièrement vitrée, l'enjeu portait surtout sur l'optimisation des propriétés du vitrage (qui n'ont que très peu d'impact sur le rendu architectural) pour que celui-ci laisse passer essentiellement le rayonnement visible. Le sujet a d'ailleurs été traité par optimisation par un bureau d'étude plus tard dans le projet.

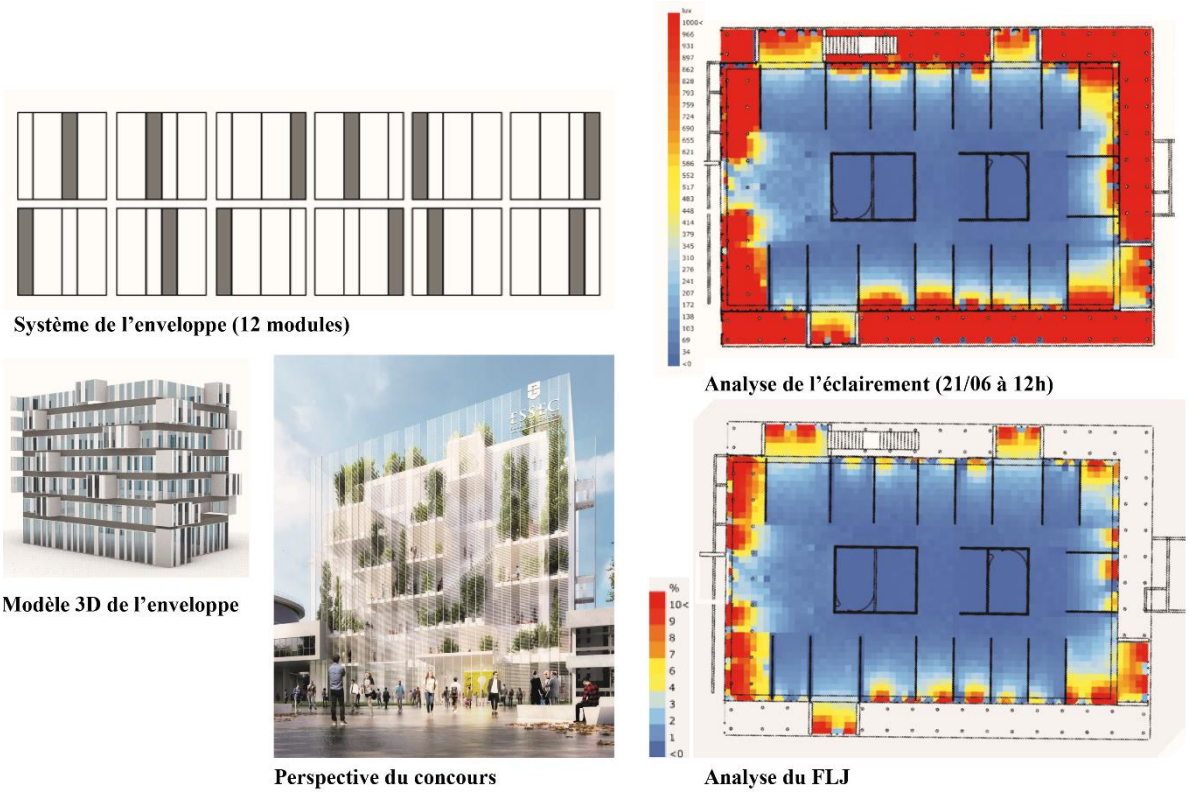


Figure 139 : Description en images de l'étude pour la Research Tower de l'ESSEC

Pour faire de la conception générative et performancielle, il est nécessaire que les utilisateurs, que ce soit le designer computationnel, le chef de projet ou l'architecte du projet, connaissent les paramètres qui affectent les critères environnementaux qu'ils souhaitent intégrer. Dans certains cas, il est difficile de savoir si un paramètre impacte un critère ou non, on peut alors réaliser une étude de sensibilité. Il s'agit d'une étude paramétrique où quelques solutions de l'espace de solutions prises au hasard sont évaluées afin de vérifier s'il existe une corrélation entre les paramètres et les critères d'évaluation.

Le cas du centre d'exploitation de Rosny-sous-Bois

Dans le cadre d'un concours, nous avons été sollicités par un jeune collaborateur architecte, adepte de la conception paramétrique, avant même qu'il n'ait réellement commencé à dessiner. Le projet consistait à réhabiliter la couverture d'un centre d'exploitation et de ses ateliers qui assurent la maintenance des trains et de l'infrastructure. L'idée de l'architecte était de créer des boîtes de formes différentes et de les disposer en toiture selon les besoins en éclairage, en apports solaires et en ventilation. Certaines boites pouvaient servir de puits de lumière, de tour à vent ou de cheminée solaires. Leurs dispositions et leurs morphologies respectives devaient être disposées de façon optimale en fonction de la vitesse des vents locaux et des effets venturi générés par les boîtes elles-mêmes, en fonction du rayonnement visible pour assurer un éclairage naturel suffisant pour travailler et en fonction du rayonnement solaire global pour éviter les surchauffes et favoriser le fonctionnement des cheminées solaires.

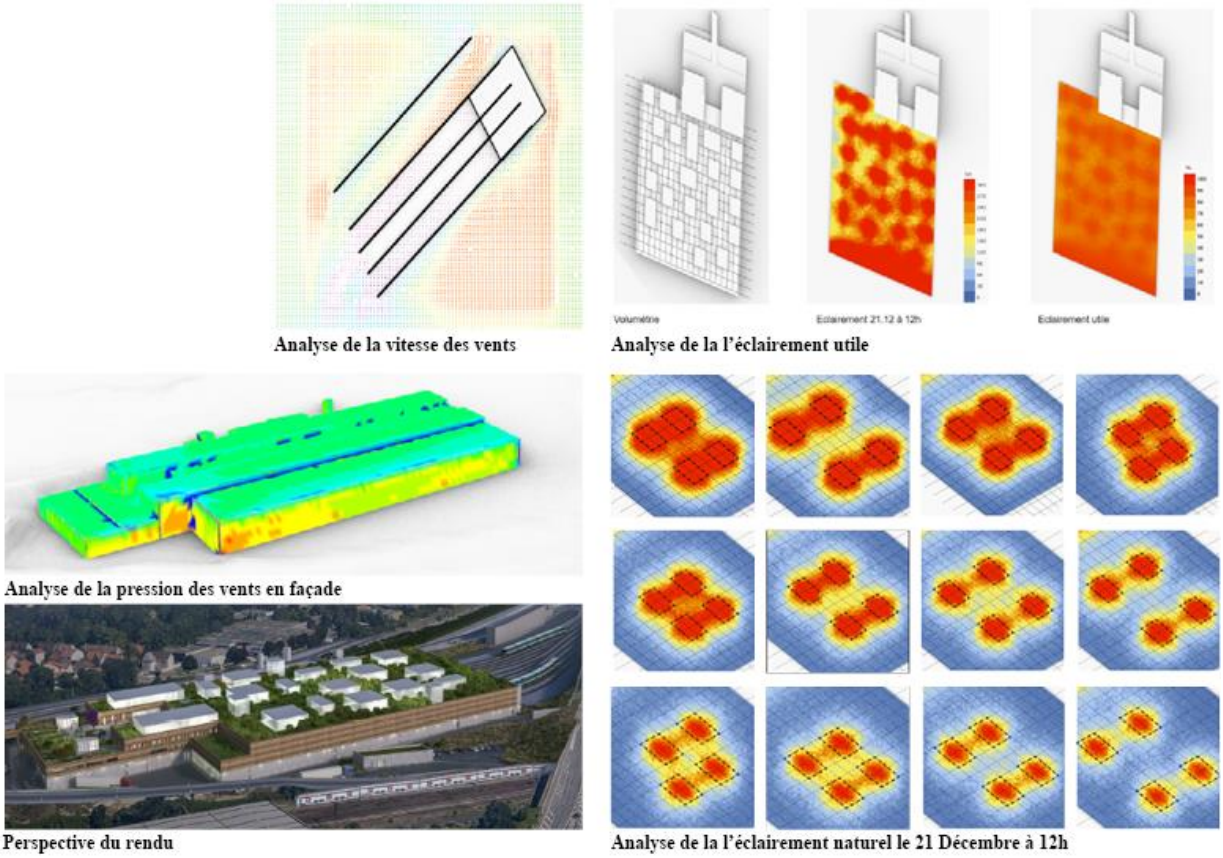


Figure 140 : Description en images de l'étude pour le centre d'exploitation de Rosny-sous-Bois

Compte tenu de l'ambition que nous avions pour cette expérimentation, nous avons sollicité l'aide du bureau d'étude environnemental qui accompagnait les architectes sur ce projet afin de pouvoir faciliter la formulation du problème et vérifier la faisabilité de nos choix en

matière de dispositif bioclimatique. Rapidement, le système des tours à vents nous a été décrit comme non adapté à ce climat, les ingénieurs du bureau d'étude lui préféraient l'usage des cheminées solaires pour la ventilation naturelle. Paradoxalement, nous étions satisfaits puisqu'en parallèle nous prenions conscience de la difficulté de réaliser une optimisation avec de la CFD à cause des temps de calculs. Dans un second temps, nous avons réalisé que les principales contraintes pour le positionnement des cheminées solaires venaient de l'intérieur du bâtiment, et non des données environnementales. Au terme des échanges, il ne restait plus que le compromis entre l'éclairage naturel et le confort thermique à condition que les temps de calculs restent raisonnables, ce qui n'était pas le cas pour le confort thermique.

Enfin, définir un espace de solutions était complexe pour ce cas. Le nombre de boîtes devait être variable, leurs dimensions aussi, mais elles devaient être positionnées sur la trame structurelle et ne devaient, bien entendu, pas entrer en collision. Faute de connaissances et d'outils sur l'intégration des contraintes à ce moment-là de la thèse, nous n'avons pu réaliser un modèle paramétrique simple qui respecte toute ces contraintes. Finalement, nous avons réalisé une étude paramétrique avec seulement quelques boîtes pour pouvoir définir des règles de positionnement pour assurer la qualité de l'éclairage naturel.

Nous retenons de cette expérience que si l'exploration de solutions demande de multiples compétences, la formulation du problème est un travail d'équipe et les designers computationnels en agence peuvent aussi se reposer sur les experts avec qui ils ont l'habitude de travailler. Ainsi, solliciter l'aide des bureaux d'étude pour leur connaissance empirique de la conception environnementale peut constituer une alternative à l'étude de sensibilité.

Pouvoir et savoir formuler le problème

Les consignes transmises par le directeur d'agence ou le chef de projet ne sont parfois pas suffisamment précis pour pouvoir formuler le problème correctement. Nous l'avons vu précédemment, cela peut être dû à des habitudes de management parfois incompatibles avec l'optimisation multicritère.

<u>Le cas de la cité du ministère de la justice de St Laurent de Maroni</u>

Pour cette expérimentation, nous avons été sollicités en phase concours après un premier rendu intermédiaire pour lequel deux variantes avaient été proposées. Le client souhaitait conserver la volumétrie et l'agencement des espaces de la première variante et voulait y appliquer le système d'enveloppe de la seconde variante. Cette façade, entièrement vitrée dans

le projet d'origine est désormais composée d'une première peau en terre (murs très épais) percée de fenêtres très verticales à la manière de meurtrières et d'une seconde peau faite de lames en bois. Cette dernière doit protéger des rayons du soleil, des intempéries, des intrusions et des projectiles en cas de manifestations. Dessinée à trois mètres de la peau intérieure pour laisser la place pour une circulation depuis la salle des pas perdu, elle doit remplir toutes ces fonctions protectrices tout en laissant passer la lumière pour éclairer naturellement des bureaux dont les exigences du programme en autonomie en lumière du jour sont très élevées. Comme le concept de l'enveloppe avait déjà été validé par le client, les modifications ne pouvaient se faire qu'à la marge (largeur, épaisseur, orientation et quantité de lames).

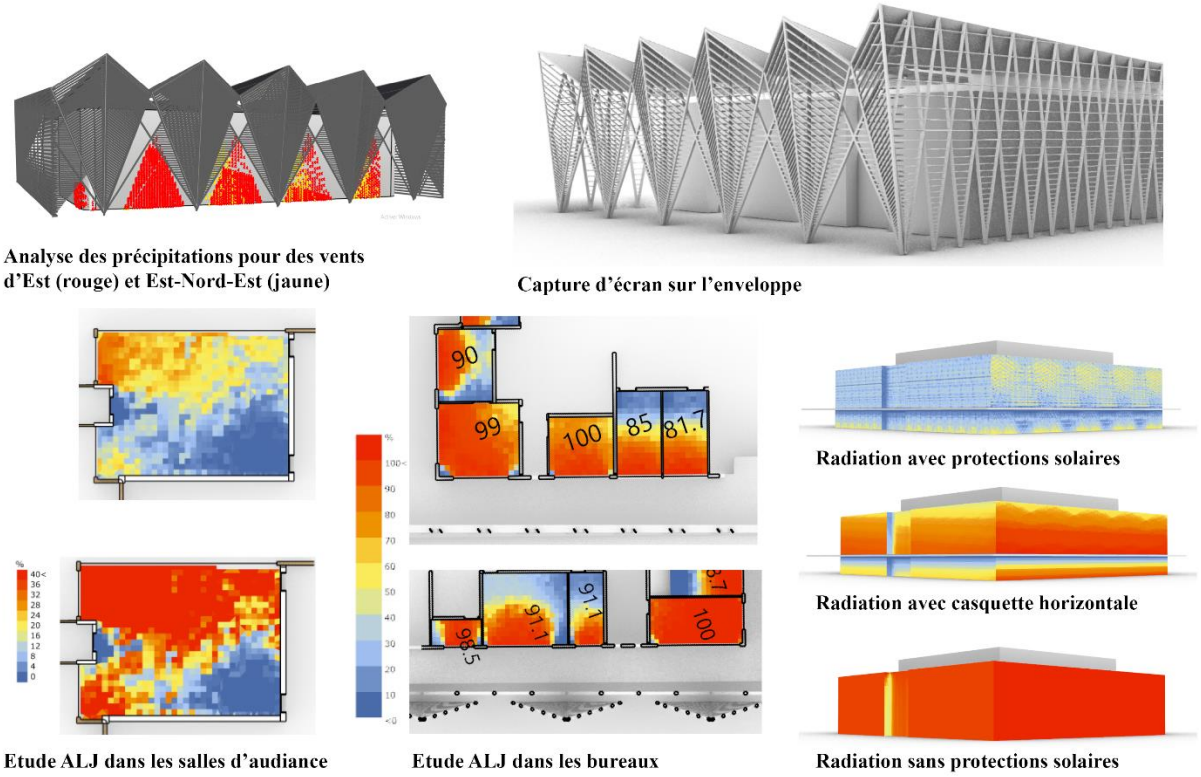


Figure 141 : Description du projet pour le palais de justice à St Laurent de Maroni

Si nous pouvons formuler le problème aujourd'hui, au moment de l'expérimentation ces informations nous sont parvenues progressivement au fil d'échanges avec différents membres de l'équipe. Il n'était pas clair au départ que la protection contre la pluie était une contrainte et pas un critère à optimiser, car la solution proposée ne doit pas protéger « *au mieux* » contre la pluie, la solution proposée ne doit pas laisser passer une seule goutte de pluie. De même pour la largeur de l'espace de circulation, ce paramètre devait rester inchangé, mais compte tenu des résultats des analyses d'éclairage, les architectes ont été contraints de céder. Cette situation s'est

d'ailleurs répétée pour presque tous les paramètres géométriques de la première et la seconde peau.

Les architectes désireux de conserver l'image architecturale de leur façade ont cherché à garder au maximum le contrôle sur la géométrie. Ainsi, nous avions trop peu de paramètres pour définir un espace de solutions susceptible d'apporter une réponse au problème. Nous avons dû réaliser de multiples analyses pour évaluer la performance de l'enveloppe sur les différents critères durant plusieurs semaines. Tous les matins, après présentation des résultats des simulations réalisées la veille, un nouveau paramètre était rendu variable, d'abord sur un domaine très restreint, puis en fin de journée, à cause des nouveaux résultats, sur un domaine plus large.

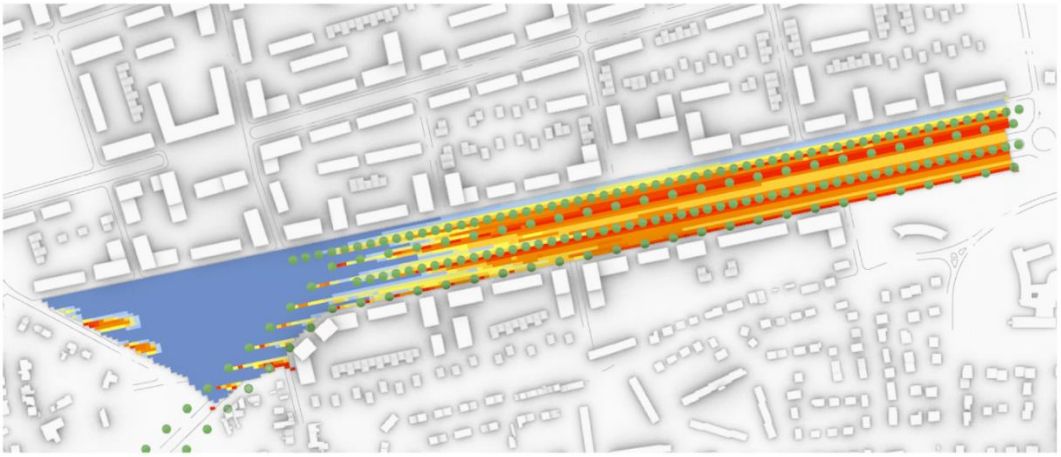
Au terme de ces itérations, il est apparu que le problème était surcontraint car même les solutions situées aux extrémums (par exemple une façade nord entièrement vitrée sans lames en bois) ne permettaient pas d'atteindre les exigences de la maîtrise d'ouvrage, mais aussi que l'utilisation d'un algorithme génétique n'aurait rien pu changer à cela. Avec le recul, nous conseillons pour trouver une issue lors de ces situations extrêmes où le problème est surcontraint à la fois par un programme très exigeant et une solution architecturale qui plait, de chercher à savoir s'il existe une solution faisable avant de chercher à en trouver une. Pour cela une étude de sensibilité avec une quantité importante de paramètres variants sur quelques valeurs aurait sans doute permis de faire la démonstration de l'absence de solution et d'arriver plus rapidement à la conclusion qu'il fallait soit changer la façade, soit revoir à la baisse les exigences du programme (ce qui a finalement été convenu). Ainsi, parfois la bonne formulation d'un problème de conception compte bien plus de variables que ce que les architectes souhaitent de faire varier.

Finalement, nous retenons de cette expérience que les outils d'intelligence numérique ne peuvent pas faire de miracle et qu'un problème surcontraint restera toujours sans solution. Le micro-management n'est pas adapté pour mettre en œuvre de l'optimisation et les designeurs computationnels ont besoin d'avoir la vision la plus complète du problème dès les études préliminaires et surtout éviter d'étudier les critères en silo.

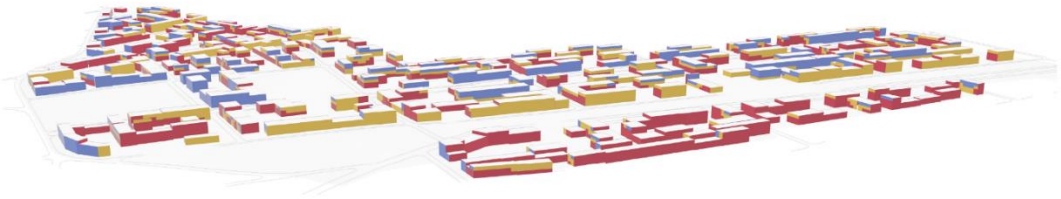
Aménagements urbains du plateau Nord-Est

Dans cette seconde expérimentation appliquée pour la première fois à l'échelle urbaine, la formulation du problème est restée floue tout au long de l'expérimentation. Même si les

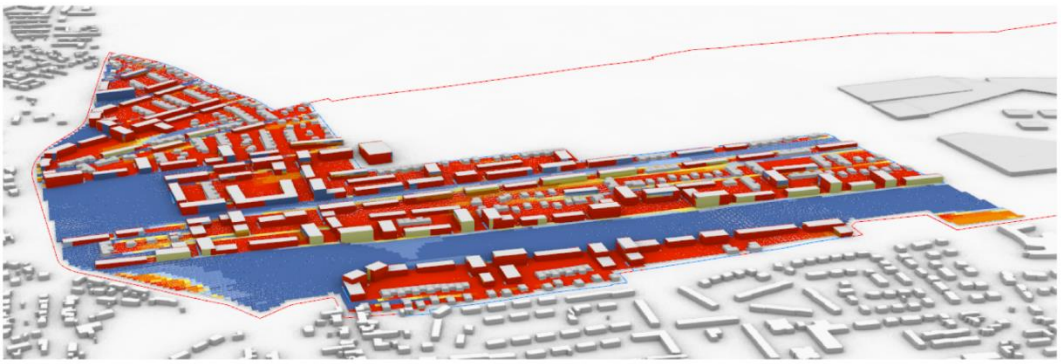
urbanistes commanditaires de cette étude étaient convaincus que l'approche par optimisation pouvait avoir du potentiel à cette échelle, ils ne savaient pas exactement ce qu'ils en attendaient, alors un management de type production-correction s'est installé.



Analyse de l'impact de la végétation sur l'accès à la vue sur la cathédrale depuis les voiries



Analyse de l'accès à la vue sur la cathédrale depuis les logements



Analyse de l'accès à la vue sur la cathédrale depuis les espaces publics

Figure 142 : Evaluation de la visibilité de la cathédrale de Chartres depuis les espaces publics et privés

L'unique certitude pour ce projet d'aménagement urbain pour la ville de Chartres était qu'un critère facilement quantifiable pouvait être étudié : la visibilité sur la cathédrale. Plusieurs idées d'études sont évoquées par l'équipe d'urbanistes d'AS : l'étude de la qualité de vue depuis les logements, l'analyse de l'impact de la végétation sur la vue depuis les espaces publics, ou encore de l'étude de la qualité de vue depuis les espaces extérieurs privés. Un plan masse composé d'îlots urbains ouverts mixtes avec trois typologies de logements (maisons individuels, maisons en bande, et petits immeubles de logements collectifs alignés sur la voirie)

a déjà été dessiné, mais il nous est proposé de le réinterroger en jouant sur la disposition des bâtiments, leurs hauteurs et la disposition de la végétation. Le quartier comptant plus d'une trentaine d'îlots, la combinatoire est gigantesque.

Dans un premier temps, nous proposons de diviser le problème en sous-problèmes et de se concentrer uniquement sur les îlots et la visibilité depuis les logements. Ainsi, nous développons un outil d'évaluation spécifique, notamment pour réduire les temps de calculs, présenté dans la section 3 du chapitre 1. Il permet d'évaluer la visibilité sur un élément remarquable. Nous l'avons testé sur les logements, les espaces extérieurs publics et privés. Les résultats sont présentés en Figure 142.

Dans un deuxième temps, nous avons développé en python un générateur d'îlots urbains (ouverts mixtes pour du logements) qui permet de générer des îlots similaires à ceux du plan masse originel. Après plusieurs jours de programmation, nous organisons un nouvel échange avec les urbanistes. A la vue des îlots générés, la commande évolue. Finalement, les urbanistes ne souhaitent plus que leur plan soit remis en question, mais préféreraient une étude à partir de la génération de formes plus « *abstraites* » pour pouvoir faire des prescriptions générales. Si la nature de ces prescriptions n'a pas été précisée, les urbanistes ont exprimé la volonté d'interroger la densité.

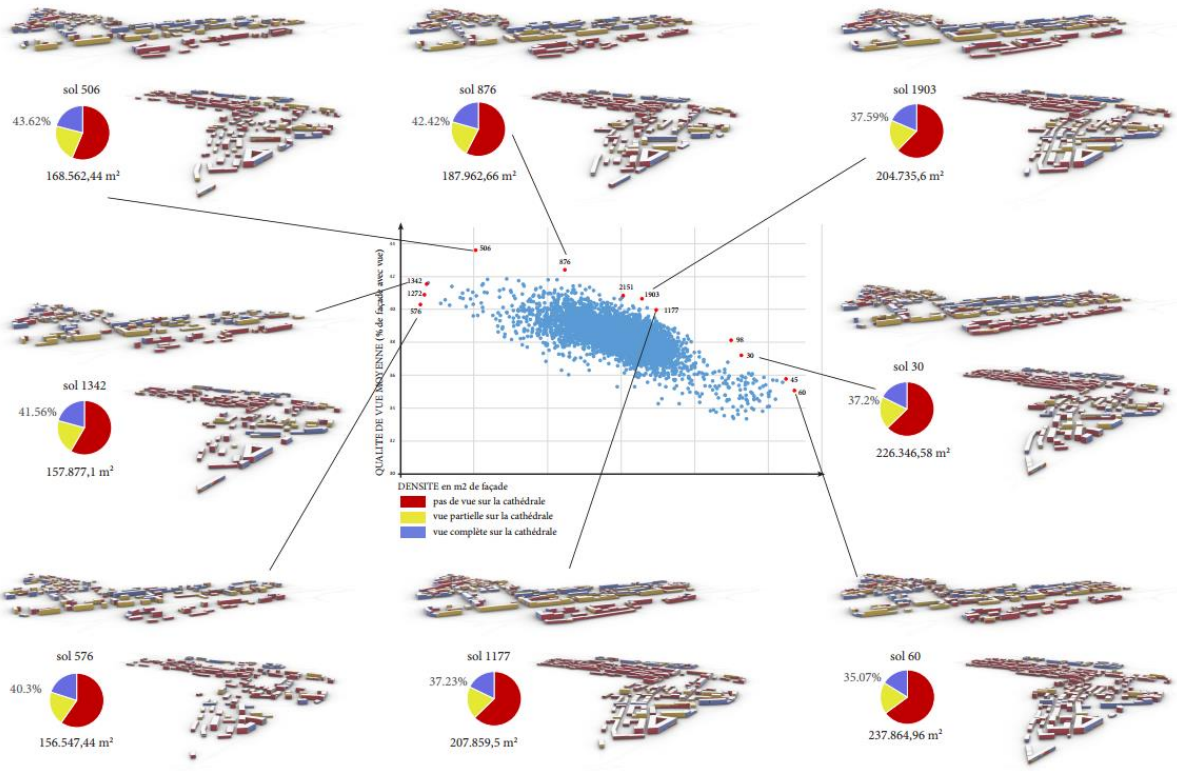


Figure 143 : Résultats de l'optimisation sous contraintes (exploration 2)

Dans un troisième temps, nous avons créé un nouveau générateur d'îlots très simple pour qu'il ne génère que des boîtes alignées aux voiries. Chaque polyligne de chaque îlot est divisée en plusieurs segments délimitant les surfaces pour les bâtiments. Les paramètres du modèle servent uniquement à faire varier les hauteurs. Deux explorations sont lancées avec l'algorithme génétique NSGAII pour l'optimisation de deux critères : la visibilité sur la cathédrale et la densité. La première optimisation a été réalisée sans intégrer de contrainte. Elle a permis de vérifier que la densité avait bien un impact sur la visibilité moyenne sur la cathédrale depuis les logements. Pour réduire la taille de l'espace de solutions et ne générer que des quartiers avec des densités réalistes, nous avons relancé les calculs en intégrant des contraintes avec une fonction de réparation pour que la densité de chaque îlot généré soit équivalente à plus ou moins 25 % de la densité observée dans le plan originel. Ainsi, nous avons pu conclure que l'agencement des bâtiments et la répartition des étages peuvent entrainer un différentiel de 7 % de quantité de façade ayant une vue sur la cathédrale, et que les meilleures solutions placent les bâtiments les plus hauts sur les espaces les plus éloignés de la cathédrale (ce qui était prévisible). On observe sur le graphique en Figure 143 que la relation entre densité et visibilité moyenne sur la cathédrale est parfaitement linéaire.

Pour finir, nous transmettons les résultats graphiques, algébriques et géométriques aux urbanismes avec les modèles 3D de quelques solutions situées sur le front de Pareto. Il semblerait que les urbanistes n'aient finalement pas exploité les résultats de cette étude.

Au terme de cette étude, nous concluons que l'usage de l'optimisation lorsque l'on ne sait pas précisément ce que l'on cherche n'est pas pertinent. En architecture, la production par le dessin permet de faire avancer le projet, produire sans forcément savoir où l'on va est un réflexe payant en conception traditionnelle, mais qui s'avère plus laborieux en conception computationnelle en particulier avec des processus d'optimisation.

Savoir modéliser le problème

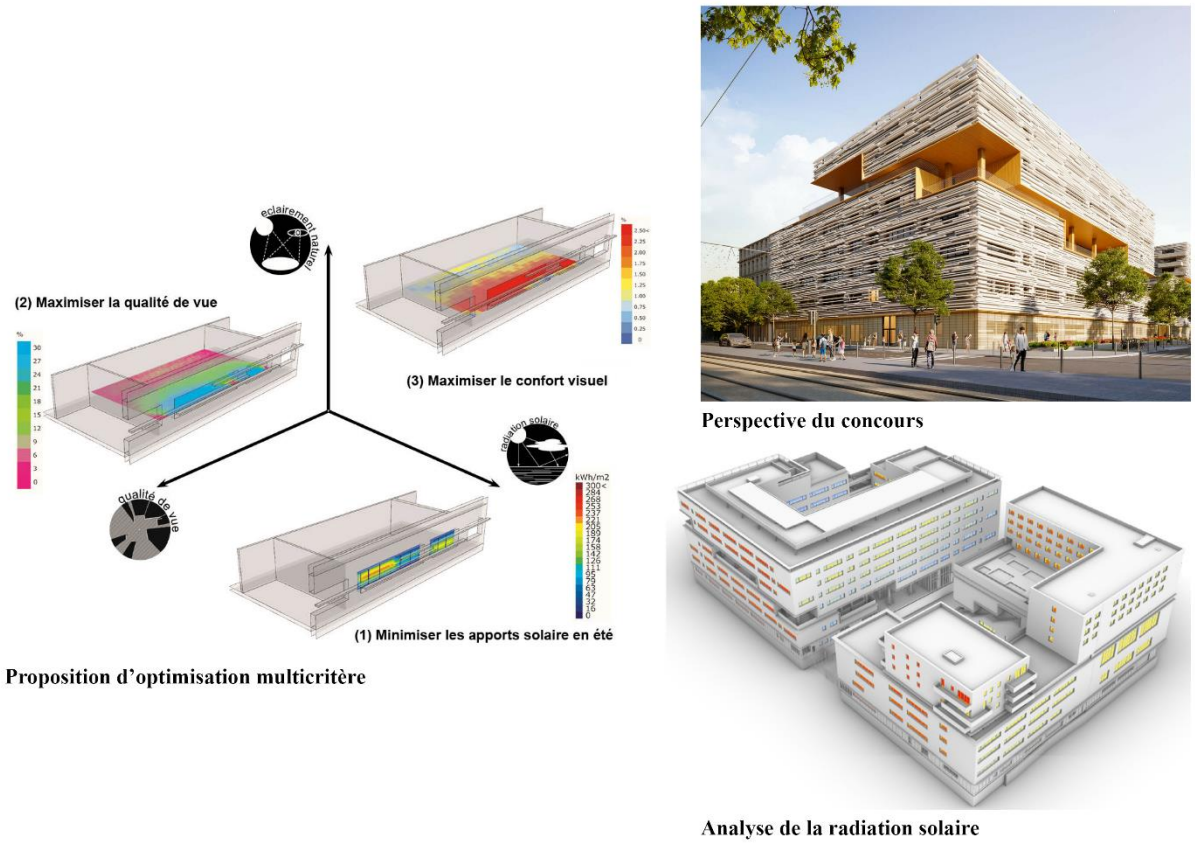
Il arrive aussi que pour certains problèmes, la formulation ne représente pas un défi comparé à leur modélisation. C'est notamment le cas pour les deux tentatives d'expérimentations que nous présentons ici.

Cité scolaire internationale de Marseille

Dans le premier cas, nous avons été sollicités sur un concours réalisé en PPP pour l'optimisation d'un système de protection solaire pour une école à Marseille. La demande venait

à l'origine d'un commentaire de la maîtrise d'ouvrage qui, *« quitte à proposer une façade paramétrique »*, suggérait qu'elle soit optimisée. Nous avons donc cherché à créer un modèle pour ce projet qui a entrainé de nombreux questionnement.

Il s'agissait d'optimiser la disposition de lames horizontales pour trouver un compromis entre trois critères pour les salles de classes : la réduction des apports solaires en été, l'accessibilité à la lumière naturelle et la visibilité vers l'extérieur. L'ensemble des lames devait être réparti de façon homogène et créer un effet aléatoire. L'objectif est donc de faire une optimisation à la marge du système d'enveloppe sans remettre en question l'architecture générale de la façade, en essayant si possible d'optimiser son coût.



Proposition d'optimisation multicritère

Figure 144 : Description du problème pour la cité scolaire internationale de Marseille

L'esquisse dessinée par l'équipe d'architectes étant recouverte de plusieurs centaines de lames (voir Figure 144), il apparait rapidement que le risque d'explosion combinatoire est élevé. Deux approches sont alors possibles :

(1) La première consiste à optimiser chaque façade par orientation, soit 4 explorations par bâtiments. Un seul modèle serait réalisé et les calculs serait ensuite lancés à huit

reprises. Mais même pour une seule façade, la combinatoire reste grande et des contraintes doivent être intégrées au modèle pour éviter que certaines lames ne se retrouvent en lévitation devant les baies, que des lames ne rentrent en collisions, qu'elles soient uniformément réparties sur l'ensemble de la façade et qu'il n'y ait[2] pas d'alignements de lames.

(2) La seconde consiste à faire un modèle à l'échelle de la salle de classe et à réaliser autant d'explorations qu'il y a de configurations de salles de classe dans l'école. Cela permettrait de restreindre le modèle à quelques paramètres sur un nombre raisonnable de lames. Les architectes pourraient ensuite reconstituer le puzzle de chaque façade à partir des résultats des différentes configurations.

La seconde solution s'est avérée irréaliste compte tenu du nombre de configurations. Pour que deux salles de classe soit considérées comme des configurations similaires, elles doivent avoir la même dimension, la même orientation, la même quantité d'apports solaires (sachant que le contexte urbain est particulièrement hétérogène), elles doivent aussi accueillir des enfants du même âge car la hauteur des yeux pour l'évaluation de la visibilité doit être la même. Cela reviendrait à relancer plusieurs dizaines de fois l'algorithme.

Finalement, il semble moins chronophage de traiter le problème façade par façade. Cependant, cette première solution risque d'aboutir à des résultats où les scores sont très hétérogènes selon les classes, avec certaines pièces très performantes et d'autres très mauvaises. Après plusieurs tentatives, nous n'avons pas réussi à trouver un modèle paramétrique qui permette d'intégrer toutes les contraintes précédemment citées. Cette tentative d'expérimentation a finalement dû être abandonnée, mais nous servira plus tard de cas d'étude pour notre recherche sur les méthodes d'intégration des contraintes.

Réhabilitation de la Caserne Thiry à Nancy

Pour ce dernier exemple d'application, nous avons été sollicités sur un concours pour une réhabilitation d'une caserne de pompier en immeuble de bureaux. L'enjeu de ce projet est la conservation en l'état d'une façade classée qui laisse peu entrer la lumière naturelle, rendant le respect des exigences en matière d'éclairement naturel dans des espaces de travail très difficile. Nous sommes donc intervenus dès l'étape des études générales lorsqu'aucune solution n'avaient encore été figée.

Au départ, l'équipe d'architectes souhaitait que nous comparions la qualité de l'éclairement de différentes variantes présentées dans la Figure 145 en effectuant des

évaluations selon les exigences du programme basé sur la norme HQE (haute qualité environnementale). Il s'agit d'une analyse du FLJ effectuée tous les 50 cm^2 sur l'ensemble des locaux, car le respect de la règle se juge au pourcentage de la surface du local atteignant un certain seuil. La norme exigeait aussi de prendre en compte le contexte urbain, et la végétation. L'ensemble des dispositifs architecturaux proposés jouant sur la réflexion de la lumière, les analyses devaient être effectuées en haute qualité. Les temps de calculs étaient donc très conséquents (jusqu'à 24h pour une seule solution), rendant l'usage de l'optimisation rédhibitoire.

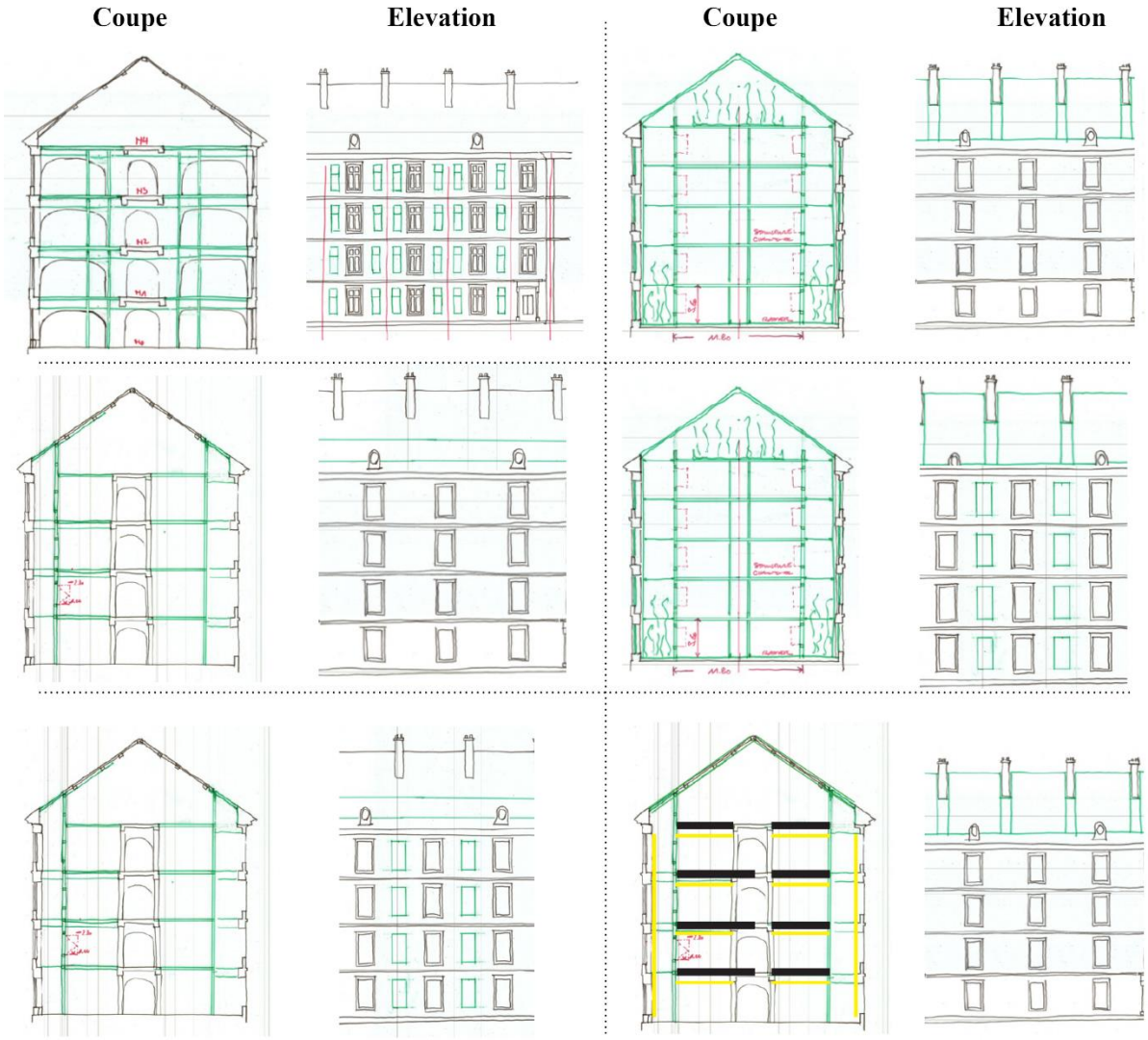


Figure 145 : Les différentes variantes de projet testées

Au cours de l'étude, voyant que les objectifs seraient difficiles à atteindre, les architectes ont commencé à démultiplier les paramètres sur lesquels nous pouvions jouer pour essayer de trouver une solution adéquate. Parmi eux, ont été ouverts le dimensionnement et la localisation du vitrage en toiture, le dimension et la localisation des puits de lumière dans les circulations,

le dimensionnement et la disposition de dispositifs réfléchissants, le dimensionnement et la localisation de nouvelles fenêtres en façade, le dimensionnement et la localisation de parois vitrées intérieures, le coefficient de transmission lumineuse du vitrage, le détail des fenêtres, le dimensionnement et la disposition des faux plafonds.

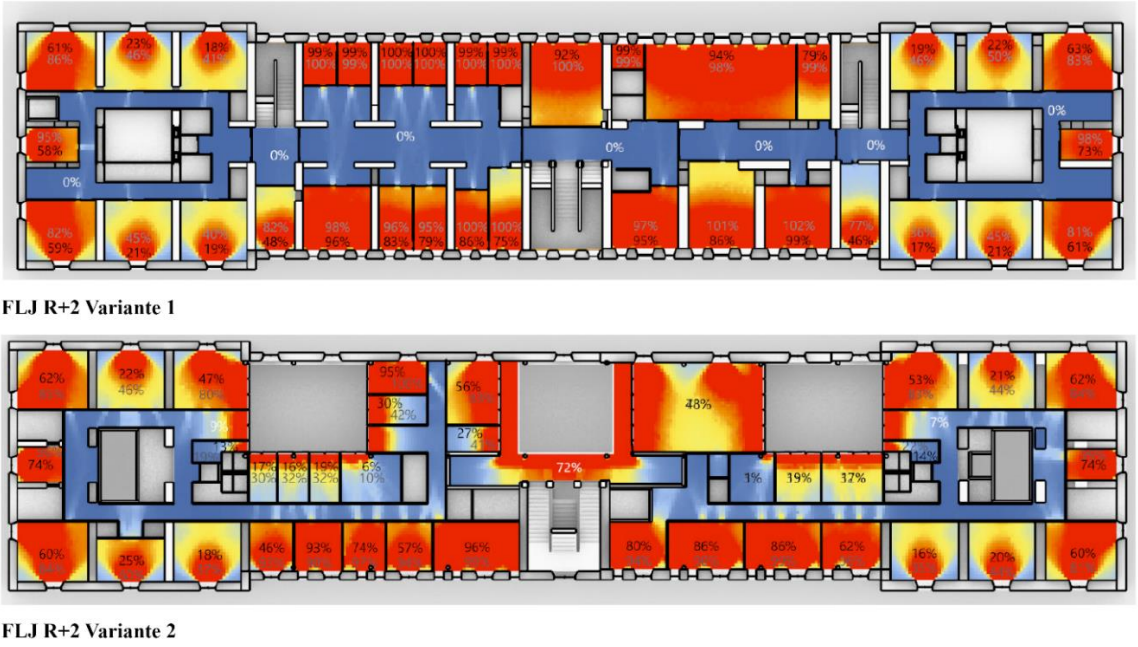


Figure 146: Analyse du FLJ pour deux variantes

Deux échelles étaient donc interrogées pour chercher à optimiser l'accès à la lumière naturelle, l'échelle globale du bâtiment où les architectes cherchaient quel dispositif (patios intérieurs, toiture vitrée…) permettrait d'éclairer un maximum de locaux (voir Figure 146), et l'échelle du détail où l'on étudiait quel détail de fenêtre pourrait permettre d'augmenter la surface du local avec un FLJ suffisant (voir Figure 147).

Abstraction faite des temps de calculs pour lesquels même l'entraînement d'algorithme de ML pour créer des modèles de substitution ne semblait pas réaliste, l'usage de l'optimisation restait complexe et peut-être même non pertinent. Nous aurions pu effectuer des évaluations avec une méthode d'analyse simplifiée comme un éclairement au 21 décembre à midi avec une précision en moyenne qualité. Cependant, que ce soit pour l'échelle du bâtiment ou du détail de fenêtre, la définition d'un modèle paramétrique était très difficile pour plusieurs raisons. D'abord parce que, comme déjà évoqué, le management à la tâche où les paramètres changent tous les jours ne s'y prête pas, et ensuite parce que les solutions que nous devions comparer étaient souvent très éloignées typologiquement les unes des autres.

Avec un modèle paramétrique nous pouvons interroger la dimension et l'emplacement des patios, ou interroger la quantité de fenêtres en façade, ou la quantité de surface vitrée en

toiture (ce qui a été fait), mais c'est un exercice différent de comparer les trois ou même plus de dispositifs. Aussi, certains paramètres, comme l'emplacement des patios, a un impact sur l'agencement des bureaux, il aurait donc fallu modéliser un allocateur spatial, ce qui ne s'improvise pas.

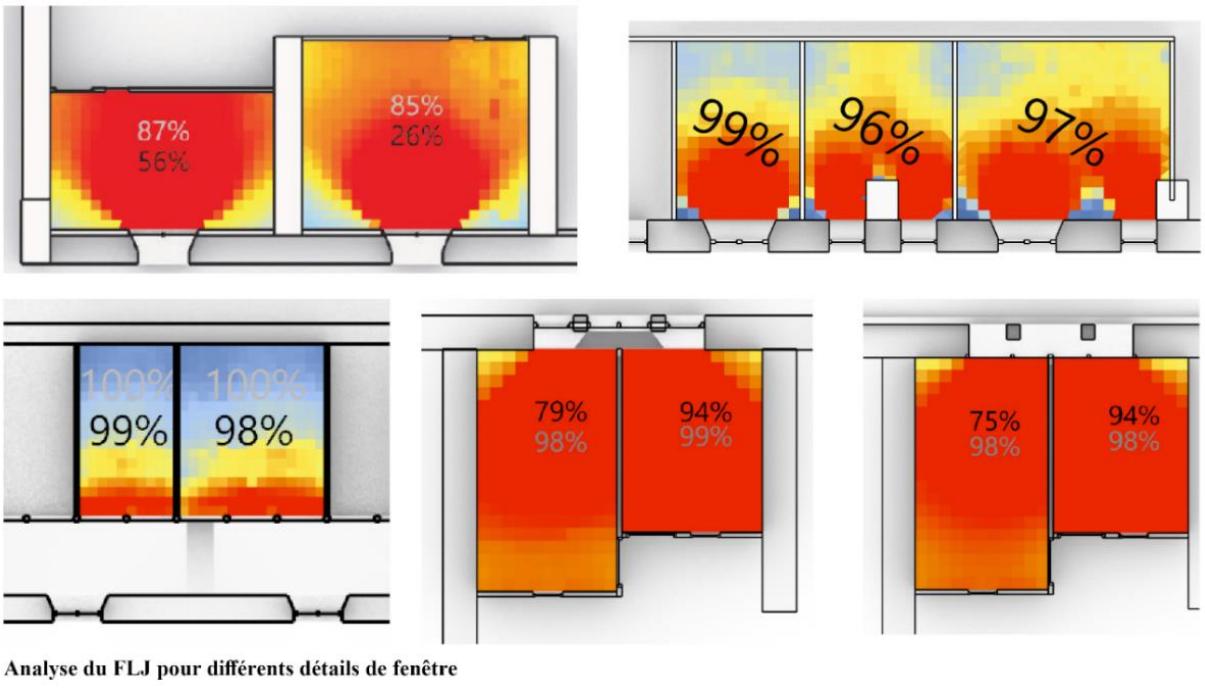


Figure 147 : Exploration pour le détail des fenêtres

Il en va de même pour le détail des fenêtres. Dans la solution retenue par les architectes, qui sera finalement construite, AS ayant remporté le concours, des parois sont disposées sur l'axe des fenêtres, ainsi certaines éclairent deux bureaux à la fois. Pour favoriser la diffusion des rayons lumineux, un partie de la paroi (celle proche de la fenêtre) est vitrée. Une telle solution ne peut être inventée par un algorithme d'optimisation. Ces derniers permettent de comparer des dispositifs architecturaux, d'optimiser un dispositif architectural mais pas d'en inventer un.

Au terme de cette seconde partie du chapitre 2, quinze exemples d'applications sur des cas réels en agence d'architecture nous ont permis d'illustrer les différentes difficultés, notamment les limites d'ordre structurel, susceptibles d'être rencontrées dans la pratique. Ces dernières, selon nous, constituent l'une des raisons majeure de l'écart qui continue d'exister entre théorie et pratique en matière de conception générative et performancielle. Nous retenons aussi que les expérimentations qui ont abouti ont toutes été développées lors de l'étape de développement et non lors des études générales. Ainsi, il s'agissait essentiellement d'optimiser une première proposition déjà dessiner par un architecte.

Il apparait surtout que les problèmes rencontrés sur certains projets étaient très clairs, faciles à formuler, mais que leur modélisation était trop complexe pour être réalisée dans le temps qui pouvait être consacré à l'étude. Cette difficulté peut notamment s'expliquer par la limite des modèles paramétriques qui ne permettent pas toujours de facilement intégrer des contraintes ou de définir les espaces de solutions attendus.

Enfin, concernant les phases de concours, les temps de calcul sont souvent trop longs pour qu'une exploration numérique avec utilisation d'algorithmes d'optimisation puissent être envisagée à ce stade du projet. Ainsi, nous proposons de terminer ce chapitre par de nouvelles expérimentations incluant l'usage d'algorithme d'intelligence artificielle réputé pour leur capacité réduire les temps de calcul.

2.3. Réduire les temps de calcul avec des modèles de substitution

Dans cette troisième partie du chapitre 2, nous nous intéressons aux algorithmes de *Machine Learning* qui permettent de créer des modèles pouvant se substituer aux simulations et ainsi considérablement réduire les temps de calcul des ENAPE.

Dans un premier temps, nous nous concentrons sur les méthodes permettant de prédire les valeurs numériques des fonctions objectifs, puis dans un second temps, nous nous intéressons à des algorithmes plus récents qui permettent de générer des images, donc de prédire la visualisation des résultats des simulations.

Nous effectuons pour les deux méthodes des tests sur des cas inspirés des problèmes rencontrés en agence chez Architecture-Studio afin de mieux appréhender l'applicabilité de ces techniques dans la pratique.

2.3.1. Prédire des résultats numériques

Pour réduire les temps de calcul, une méthode bien connue consiste à entraîner des algorithmes d'apprentissage automatique, ou ML pour « *machine learning* » en anglais à prédire les résultats des logiciels de simulations.

Etat de l'art des méthodes

Les différents types d'algorithmes de Machine Learning

Les algorithmes de ML sont classés dans une sous-catégorie de l'intelligence artificielle. Il s'agit de systèmes qui ont la capacité d'apprendre à partir de données et de s'améliorer avec l'expérience. Ces dernières années, et plus particulièrement depuis 2019, un véritable engouement pour ces sujets s'est développé dans le champ de la conception computationnelle (Sari & Berawi, 2021). Derrière le terme « machine learning » se cache différentes techniques. Selon les auteurs, l'appartenance à la catégorie ML est discutable. Par exemple, dans leur revue qualitative Sari et Berawi considèrent les algorithmes génétiques et les réseaux bayésiens comme des techniques de ML, contrairement à David Newton qui considère qu'il ne s'agit pas d'intelligence artificielle (Newton, 2019).

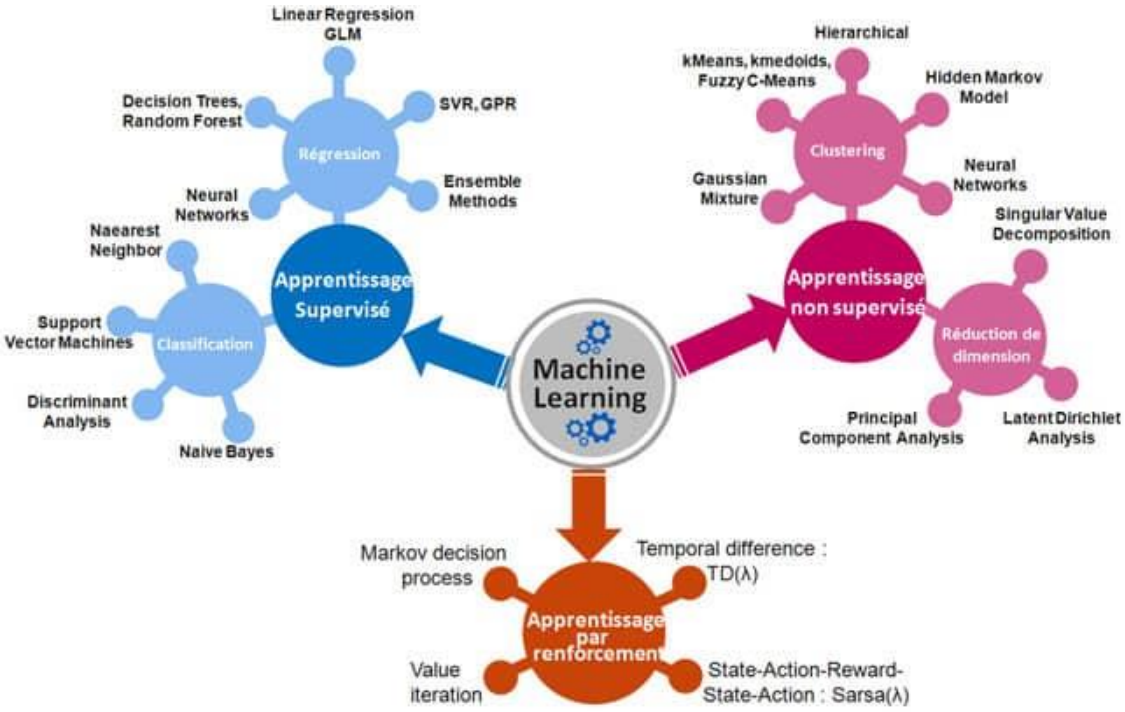


Figure 148 : Les différents types d'algorithmes de ML (source: https://www.groupe-hli.com/machine-learning-dans-industrie/)

Comme le montre la Figure 148, il existe différents types d'apprentissage automatique dont les trois principaux sont l'apprentissage supervisé, non-supervisé et l'apprentissage par renforcement. Ces algorithmes permettent de réaliser quatre types de tâches différentes :

(1) des régressions, c'est-à-dire approcher une variable à partir des valeurs d'autres variables qui lui sont corrélées. On parle aussi d'ajustement de courbe.

(2) de la classification, c'est-à-dire à catégoriser des objets, comme par exemple identifier si un email est un spam ou non, ou classer des images.

(3) du regroupement des données (en anglais « *clustering* »), *c'est-à-dire à* diviser un ensemble de données en paquets homogènes. Cela peut, par exemple, être utilis é pour segmenter une image.

(4) la réduction de dimensions, qui sert à réduire la dimension de l'espace de calcul en supprimant des variables pour pouvoir entraîner plus rapidement des algorithmes d'intelligence artificielle.

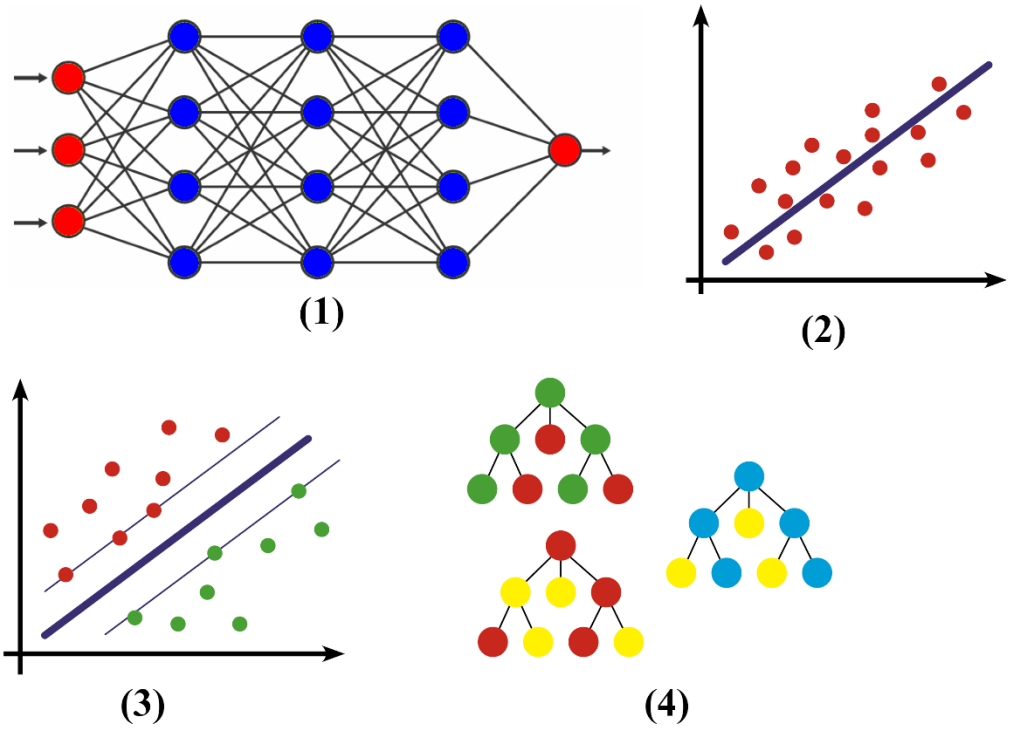


Figure 149 : Les algorithmes d'apprentissage supervisé

Un modèle de substitution cherche à faire une régression. Les algorithmes utilisés pour faire de la régression sont des algorithmes d'apprentissage supervisé. Comme l'illustre la Figure 149Figure 148, il existe cinq grandes familles d'algorithmes d'apprentissage automatique qui permettent de faire de l'approximation de courbe :

(1) les réseaux neuronaux,
(2) les régressions (ainsi le terme peut faire référence soit à la tâche, soit à un type d'algorithme),
(3) les machines à vecteur de support ou SVM (pour « *Support Vector Machines* » en anglais),
(4) les arbres de décisions ou forêts aléatoires (ou « *random forest* » en anglais) et
(5) les méthodes ensemblistes qui consistent à utiliser plusieurs algorithmes différents pour augmenter la précision de la précision.

Les modèles de substitution pour l'architecture

Au chapitre 1, nous avons pu constater que pour les applications intégrant des préoccupations architecturales, rares sont les auteurs qui intègrent des modèles de substitution à leur processus d'optimisation. Cependant, de plus en plus de publications sur l'optimisation énergétique des bâtiments intègrent des modèles de substitution, mais il s'agit essentiellement d'étudier des paramètres relevant de l'expertise des bureaux d'étude (Araújo et al., 2023; Aydın et al., 2015; Bamdad et al., 2020; Chegari et al., 2022). Les modèles de substitution peuvent aussi être utilisés à d'autres fins que l'optimisation. Ils servent en phases amonts, pour faire des analyses de sensibilité et des analyses d'incertitude, ou encore des analyses en temps réel (inférieur à 0.1s).

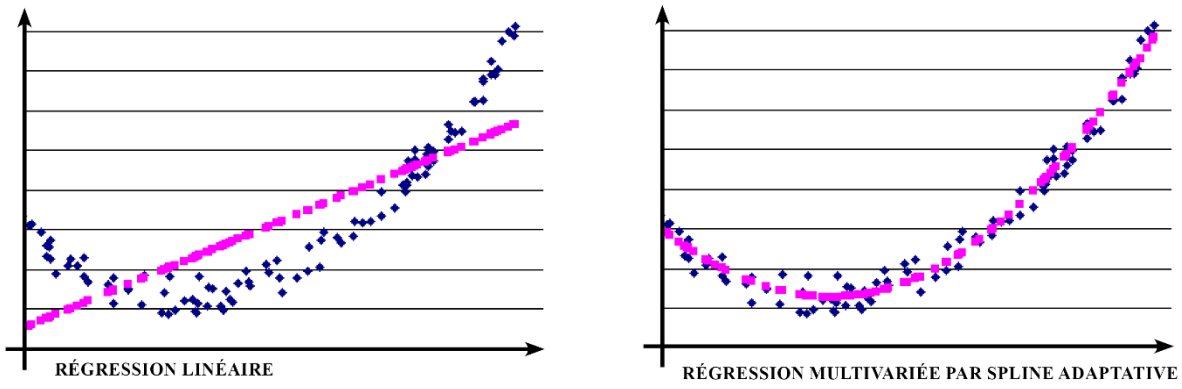


Figure 150 : régression linéaire vs MARS

Il y a désormais suffisamment de publications sur le sujet pour que certains chercheurs puissent réaliser des études comparatives des modèles de substitution en s'appuyant sur des revues de littérature exhaustives. Parue notamment en 2019, une étude s'appuient sur l'analyse de 57 publications sur les modèles de substitution pour la conception de bâtiment durable dans le but de proposer un guide pratique à la réalisation de ce type de modèle (Westermann & Evins,

2019). Les modèles de substitution cités dans cette revue son principalement utilisés pour la prédiction des consommations d'énergie, mais aussi pour l'éclairage naturel, le confort thermique (PMV) et l'ACV.

Ainsi les auteurs identifient 8 techniques d'apprentissage principalement utilisées : les réseaux neuronaux, qui se trouvent dans 13 % des articles, les fonctions de base radiale , ou RBF pour « *radial basis function* » en anglais (7 %) qui sont un type de réseau neuronal, les processus gaussiens qui sont des processus d'apprentissage bayésiens non paramétriques de modélisation contrairement aux régressions (13 %), les régressions linéaires 33 %, les régressions multivariées par spline adaptative (en anglais MARS pour « Multivariate adaptive regression splines ») qui correspondent à 10 % des papiers (voir Figure 150), les SVM (10 %), des forêts aléatoires 7 %, les méthodes ensemblistes (5 %), et d'autres techniques (2 %).

Les auteurs concluent que les prédictions énergétiques peuvent atteindre 95 % de précisions, mais que les résultats sont plus faibles pour les autres critères. Le choix de l'algorithme dépend du type d'analyse (approfondie, rapide). La performance des algorithmes dépend de la base de données mais aussi de leur paramétrage (qui varie selon les problèmes), ainsi le choix de l'algorithme dépend du niveau d'expertise de l'utilisateur. Les processus gaussiens seraient efficaces avec de petits échantillons (par opposition aux RBF ou MARS), et serait plus adapté aux non-experts pour obtenir des prédictions précises.

Les étapes de fabrication d'un modèle de substitution

Pour créer un modèle de substitution, il y a trois grandes phases : (1) une étape préliminaire qui est l'échantillonnage, (2) puis l'étape d'apprentissage, (3) et enfin l'étape de test de l'algorithme (voir Figure 151).

Pour la première étape, il faut générer les solutions qui constitueront la base de données et les évaluer avec la méthode que l'algorithme d'apprentissage cherchera à imiter. C'est une étape cruciale pour la performance du modèle. Il existe deux types d'échantillon : statique et adaptatif. Il semblerait que les échantillons adaptatifs permettent d'obtenir des résultats précis plus rapidement (avec un échantillon plus petit).

Pour la seconde étape, il faut choisir un algorithme d'apprentissage adapté au problème, le paramétrer et lancer les calculs qui peuvent être plus ou moins longs en fonction du type d'algorithme, de son paramétrage, des caractéristiques du problème (nombre de variables) et de la taille de l'échantillon.

Pour la dernière étape, il faut tester l'algorithme en générant de nouvelles solutions qui devront aussi être évaluées pour pouvoir comparer les résultats des simulations et des prédictions afin de mesurer la précision du modèle de substitution.

Toute la difficulté de l'élaboration d'un modèle de substitution réside dans l'étape de l'échantillonnage, c'est-à-dire la constitution de la base de données qui servira à nourrir l'algorithme d'apprentissage.

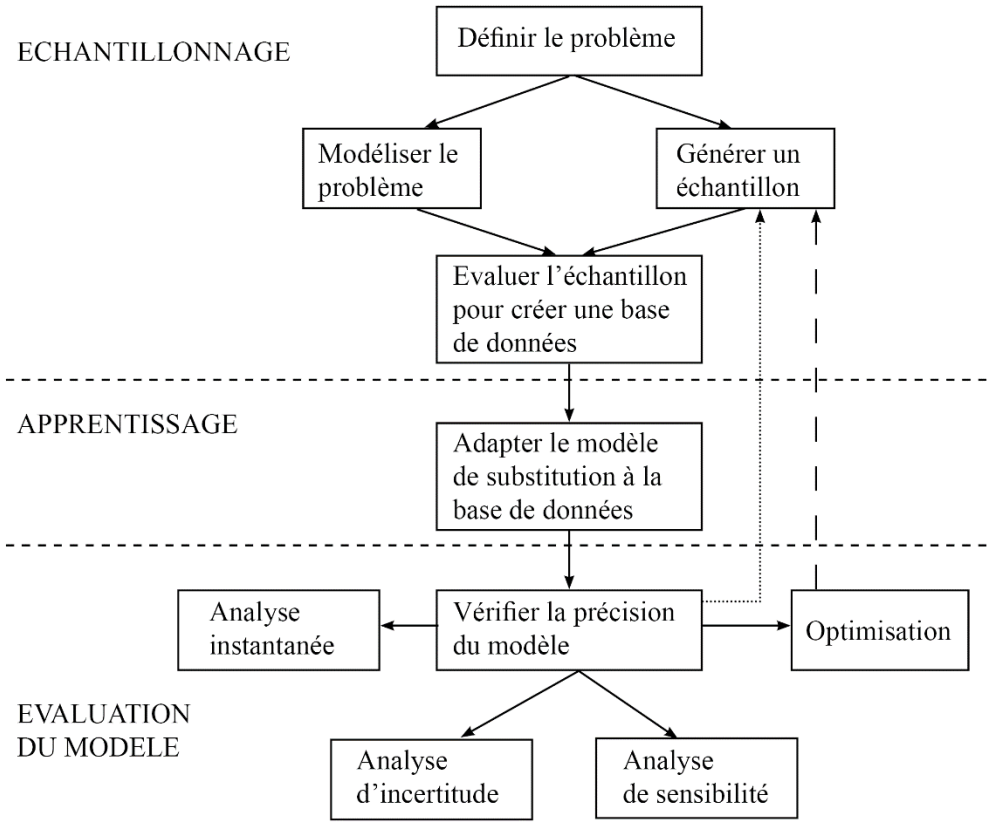


Figure 151 : Les étapes de création d'un modèle de substitution (source: (Westermann & Evins, 2019))

Les plugins Grasshopper® pour faire de l'apprentissage

De nombreux algorithmes de ML sont à la disposition des architectes, notamment via Grasshopper ®. Nous avons identifié plusieurs plugins qui permettent d'utiliser des algorithmes d'apprentissage pour faire des régressions en utilisant la plateforme food4rhino.com.

Le plugin **LunchBox®** contient divers types d'outils, notamment des algorithmes de ML. Il est développé par Nathan Miller depuis décembre 2012 et est régulièrement mis à jour depuis (dernière version avril 2023). Il compte 17 composants pour le ML. On y trouve des composants pour utiliser des réseaux neuronaux, différents types de régressions (linéaire, non linéaire, multivariables et logique). Ce plugin propose aussi d'autres algorithmes notamment

pour faire de la classification (SVM, K-means, bayésien naïf) et du clustering (K-means, mixture Gaussienne). L'utilisation de ces outils est facile d'accès, même pour des non-experts.

Le plugin **Owl®** compte 15 composants uniquement destinés au ML. Il est développé par Mateusz Zwierzycki depuis novembre 2018 et est régulièrement mis-à-jour depuis. Les trois types d'apprentissages (supervisé, non-supervisé, par renforcement) sont inclus dans cet outil. En apprentissage supervisé, trois composants permettent de faire uniquement des réseaux de neurones.

Pug® est un plugin spécialisé dans le ML développé par Diego Appelaniz depuis janvier 2023. Il compte 28 composants qui s'appuient sur les librairies de Tensorflow.net. Il permet de faire de la régression avec des réseaux de neurones. Pour cela, il est nécessaire d'utiliser de nombreux composants, ce qui permet plus de flexibilité dans son paramétrage, mais rend le plugin moins accessible aux non-experts que LunchBoxML, par exemple.

Dodo® est un plugin qui contient une collection d'outils, notamment quelques-uns pour le ML. Il est développé par Lorenzo Greco depuis novembre 2015 et n'a pas été mis à jour depuis février 2019. Trois composants permettent de créer des réseaux de neurones.

Le plugin d'optimisation **Octopus®** intègre depuis sa version 0.4 (décembre 2018) quelques composants pour faire du ML, notamment de l'apprentissage supervisé (7 composants). Ils permettent d'utiliser des réseaux de neurones et des SVM, et à l'image de ceux de LunchBox®, ils sont faciles à utiliser.

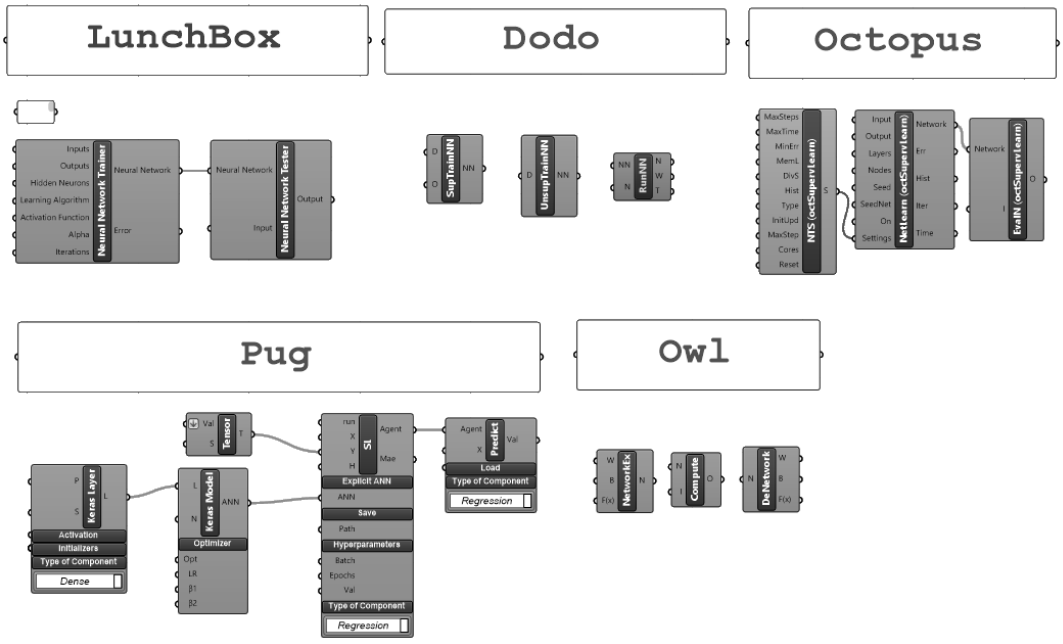


Figure 152 : Les composants Gh pour faire de l'apprentissage avec des réseaux de neurones

Le plugin **Crow®** est une extension de NeuronDotNet.dll développé par Vijeth Dinesha. Il est accessible sur Grasshopper® depuis mai 2016. Il contient 15 composants pour faire des réseaux de neurones pour de l'apprentissage non-supervisé et supervisé, mais essentiellement pour faire de la classification (et non pas de la régression).

Finalement, les algorithmes de ML déjà implémentés sur Grasshopper® pour faire de la régression pouvant être intégrés à des processus d'optimisation sont essentiellement des réseaux de neurones (voir Figure 258), quelques régressions (LunchBox®) et une SVM (Octopus).

Les expérimentations des algorithmes de ML

Méthode d'évaluation des modèles de prédiction

Afin de pouvoir un jour utiliser du ML lors de nos expérimentations sur des cas réels en agence, nous avons testé ces différents composants sur deux problèmes distincts (un détail d'enveloppe et une morphologie d'îlots) et pour plusieurs critères de performance (rayonnement solaire, éclairage naturel, confort thermique et consommations énergétiques). Notre objectif est d'identifier quel algorithme serait le mieux adapté pour quel critère pour une utilisation par des non-experts de la data science, c'est-à-dire en utilisant les paramètres par défaut des différents composants.

Nous avons sélectionné 7 algorithmes à comparer parmi les plugins les plus "*user-friendly*" de notre état de l'art, soit 5 réseaux de neurones identifiés sur Octopus® et LunchBox® (Octopus Network, Resilient Backpropagation, Backpropagation, Evolutionary, levenberg Marquardt), une régression multivariable (LunchBox®) et une SVM (Octopus®).

Pour analyser les résultats et comparer les algorithmes entre eux, nous avons dû choisir un indicateur adapté pour comparer les modèles. Il existe de nombreux critères d'évaluation des modèles de prédiction (Caruana & Niculescu-Mizil, 2004), et il ne semble pas y avoir de consensus scientifique sur une mesure qui surpasserait les autres. Les trois indicateurs les plus utilisés dans la littérature scientifique depuis 25 ans pour l'évaluation des modèles de régression sont les suivants (Botchkarev, 2019) :

(1) l'erreur quadratique moyenne (MSE pour « *mean square error* » en anglais). Pour la calculer, il faut élever au carré les erreurs individuelles (valeur réelle – valeur prédite), sommer ces erreurs, puis diviser le résultat par le nombre d'erreurs obtenues.

(2) l'erreur absolue moyenne (MAE pour « mean absolute error » en anglais. Pour la calculer, il faut additionner toutes les erreurs absolues et les diviser par le nombre d'erreurs, et

(3) l'erreur moyenne en pourcentage absolu (MAPE pour « *mean absolute percentage error* » en anglais) (Botchkarev, 2019). Elle se calcule en divisant d'abord l'erreur individuelle par la valeur réelle individuelle, puis en faisant la somme des valeurs absolues de ces valeurs, le résultat est ensuite divisé par le nombre d'erreurs. Aujourd'hui, cet indicateur est le plus populaire des trois.

Les deux premières méthodes sont dépendantes de l'échelle (dites « dimensionnées »), c'est-à-dire que leur valeur s'exprime dans la même unité que les données analysées. Elles ne permettent donc pas de savoir, par exemple, si la SVM est plus performante pour prédire l'éclairage naturel ou le confort thermique, puisque ces mesures utilisent des unités de mesure différentes. Le MAPE est un indicateur dit « sans dimension », son unité est le pourcentage. Nous l'avons donc choisi pour notre étude comparative.

Expérimentations pour un détail d'enveloppe

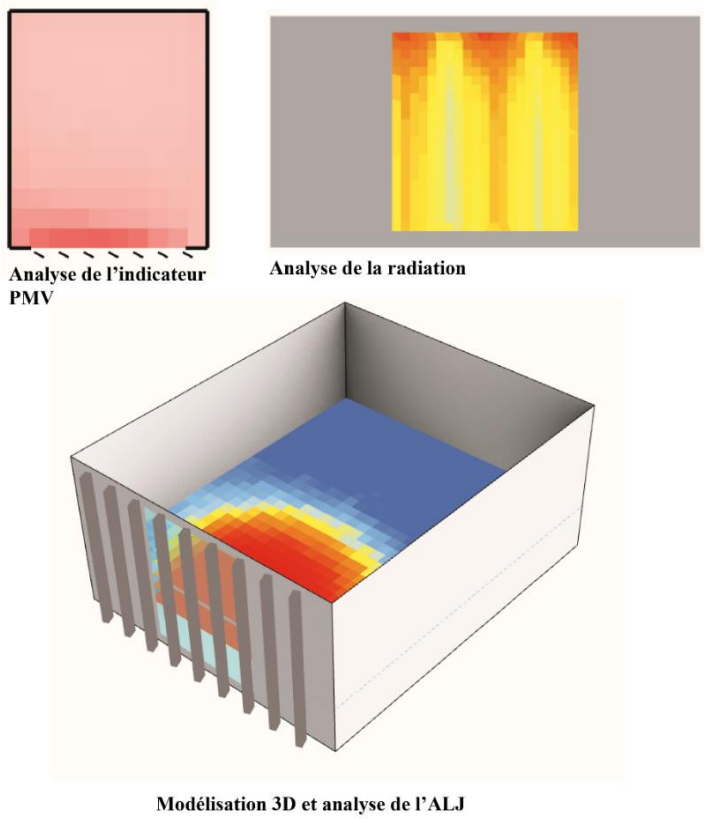


Figure 153 : Modélisation 3D de la pièce et visualisation des critères

Figure 154 : Extrait de l'échantillon pour le problème d'optimisation de détail de façade

Nous avons élaboré un modèle de substitution pour prédire la performance de critères de confort d'un bureau en fonction du dimensionnement de sa fenêtre et de son système de protection solaire. La pièce mesure 6m de profondeur et 5m de largeur. Trois indicateurs sont calculés à l'aide des outils Ladybug® et Honeybee® et des logiciels de simulations Energy Plus® et Radiance® : un indicateur du confort thermique (PMV), une analyse des apports solaires en été et une analyse de l'éclairement annuel (ALJ). Une représentation 3D de la pièce et des évaluations est illustrée en Figure 153.

Un modèle paramétrique est utilisé pour générer 550 solutions. Un extrait de cet ensemble de solution est présenté en Figure 154. Le modèle paramétrique compte 6 paramètres : la largeur et la longueur de la fenêtre, le nombre de lames, l'orientation des lames, la largeur des lames et l'épaisseur des lames. Un programme est utilisé pour générer les solutions au hasard et exporter les résultats dans un fichier csv.

Après avoir été entraînés, les algorithmes sont testés sur 50 solutions. La MAPE est ensuite calculée directement dans Grasshopper® pour les 7 algorithmes et une méthode ensembliste, les 3 critères et 12 tailles d'échantillon différentes (5, 15, 25, 50, 100,150, 200, 300, 400, 425, 450 et 500 solutions). Les résultats numériques sont présentés dans le tableau en Figure 155.

Pour les trois critères, l'algorithme le plus performant est la SVM du plugin Octopus® qui atteint une MAPE inférieure à 0,1 % lorsque la taille de l'échantillon atteint 450 solutions.

Comme le montrent les graphiques présentés en Figure 156, en dehors du cas de la SVM, les prédictions sont très imprécises notamment pour l'ALJ et la radiation solaire. Après la SVM, c'est la régression multi-variables qui obtient les meilleurs résultats et, dans ce cas, la précision augmente avec la taille de l'échantillon. Cependant, même avec 500 solutions, la régression atteint une MAPE d'environ 15,5 % pour la PMV, 27 % pour la radiation, et 44 % pour l'ALJ. Les résultats sont trop imprécis avec cette méthode pour être utilisés avec de l'optimisation.

Dans notre cas, la méthode ensembliste qui consiste à faire une moyenne des prédictions de plusieurs méthodes différentes n'est jamais meilleure que le meilleur des 7 algorithmes testés séparément.

		Octopus Network	Octopus SVM	Resilient Backpropagation	Backpropagation	Evolutionary	Levenberg Marquardt	Regresssion multivariable	Ensemble
500	ALI	83,299059	0,7612	68,577231	68,877827	68,15981	76,640301	43,967313	39,174894
	RAD	104,663196	0,014126	59,472215	60,55886	53,140631	80,767416	27,03768	31,339738
	PMV	29,055647	0,520512	30,726155	35,841283	18,596564	27,8721	15,524922	17,659726
450	ALI	206,189439	0,761481	73,23635	68,000633	75,259841	76,640301	44,372351	59,769498
	RAD	172,446665	0,014127	55,692232	60,735384	72,411202	80,767416	27,112199	43,374242
	PMV	40,470864	0,520512	27,905488	33,526556	26,377337	25,594332	15,416645	15,133661
425	ALI	85,315413	38,581714	73,690607	68,518583	70,243456	76,6403	44,429934	46,875928
	RAD	105,707639	29,400407	60,031342	59,224662	51,465884	80,767416	27,040096	38,986572
	PMV	36,772653	8,518235	16,089305	35,553037	24,231259	26,938179	15,700997	19,366914
400	ALI	59,432606	76,993621	75,386205	67,747894	70,875973	76,640295	45,347447	47,662168
	RAD	53,127324	55,556319	59,850776	60,18969	50,23819	80,767416	27,249622	34,725827
	PMV	35,319998	17,04576	24,156575	36,958877	30,165017	25,246816	16,097237	22,129133
300	ALI	106,82963	88,100387	71,604744	67,484766	75,767514	65,112136	43,839506	58,602354
	RAD	53,828664	63,941168	58,714254	61,08685	55,271014	55,274848	26,861584	36,522458
	PMV	39,567334	18,421581	16,289122	29,096756	33,303508	28,741852	16,324381	19,52619
200	ALI	108,673619	85,431724	68,662267	64,804199	59,746092	65,765369	45,198798	52,794369
	RAD	86,75529	61,375479	67,872015	59,635171	52,724177	58,455068	27,652068	36,691039
	PMV	32,884793	18,108924	23,448613	20,981628	30,269341	19,651239	16,114188	18,95408
150	ALI	107,216873	83,031438	72,425982	62,811629	60,154971	76,639017	45,564753	60,584506
	RAD	60,081866	59,124923	62,418427	57,861484	68,502152	80,750533	27,633795	36,959251
	PMV	40,573634	18,033225	26,547337	18,211779	31,871713	22,547567	16,209589	20,135836
100	ALI	131,989784	83,577267	69,981277	63,005861	71,984688	58,647182	46,921282	57,601893
	RAD	58,949269	59,466435	119,843174	57,447624	51,192116	51,749307	29,522326	39,668586
	PMV	35,472334	18,252544	23,078426	18,063669	31,217745	19,252921	16,66859	19,236594
50	ALI	78,527197	81,603754	70,316226	63,326438	73,063927	57,148259	42,359827	53,315398
	RAD	50,938855	58,547424	51,880422	55,572449	50,071359	50,404461	26,330406	38,485533
	PMV	29,149168	18,276976	23,03239	17,904264	28,317505	27,644086	16,353648	16,320849
25	ALI	102,467379	79,257531	72,089581	73,217425	62,364402	59,95229	45,818998	63,093379
	RAD	52,823999	54,86315	50,289416	50,013184	51,685571	50,575948	26,608848	40,540714
	PMV	43,918761	18,226689	24,483517	21,653276	25,020509	27,35922	17,682463	21,134783
15	ALI	186,754271	82,566374	69,483455	63,947741	66,239848	71,260164	58,78629	66,131289
	RAD	114,746743	56,61544	69,555195	52,309484	50,379829	55,341963	31,19234	38,461944
	PMV	31,4059	18,500173	27,877012	24,857096	38,726009	28,259328	18,691903	19,955166
5	ALI	171,399142	62,639755	151,915127	74,500446	73,166602	67,626658	66,32148	78,562865
	RAD	137,326222	50,101505	69,712685	57,558551	54,414418	52,01846	47,589109	43,986645
	PMV	43,918761	18,612055	43,918761	19,190226	38,006659	23,783824	40,794989	25,579696

Figure 155 : Ensemble des résultats numériques pour l'évaluation des modèles de prédiction pour l'évaluation d'un espace de bureau (MAPE en %)

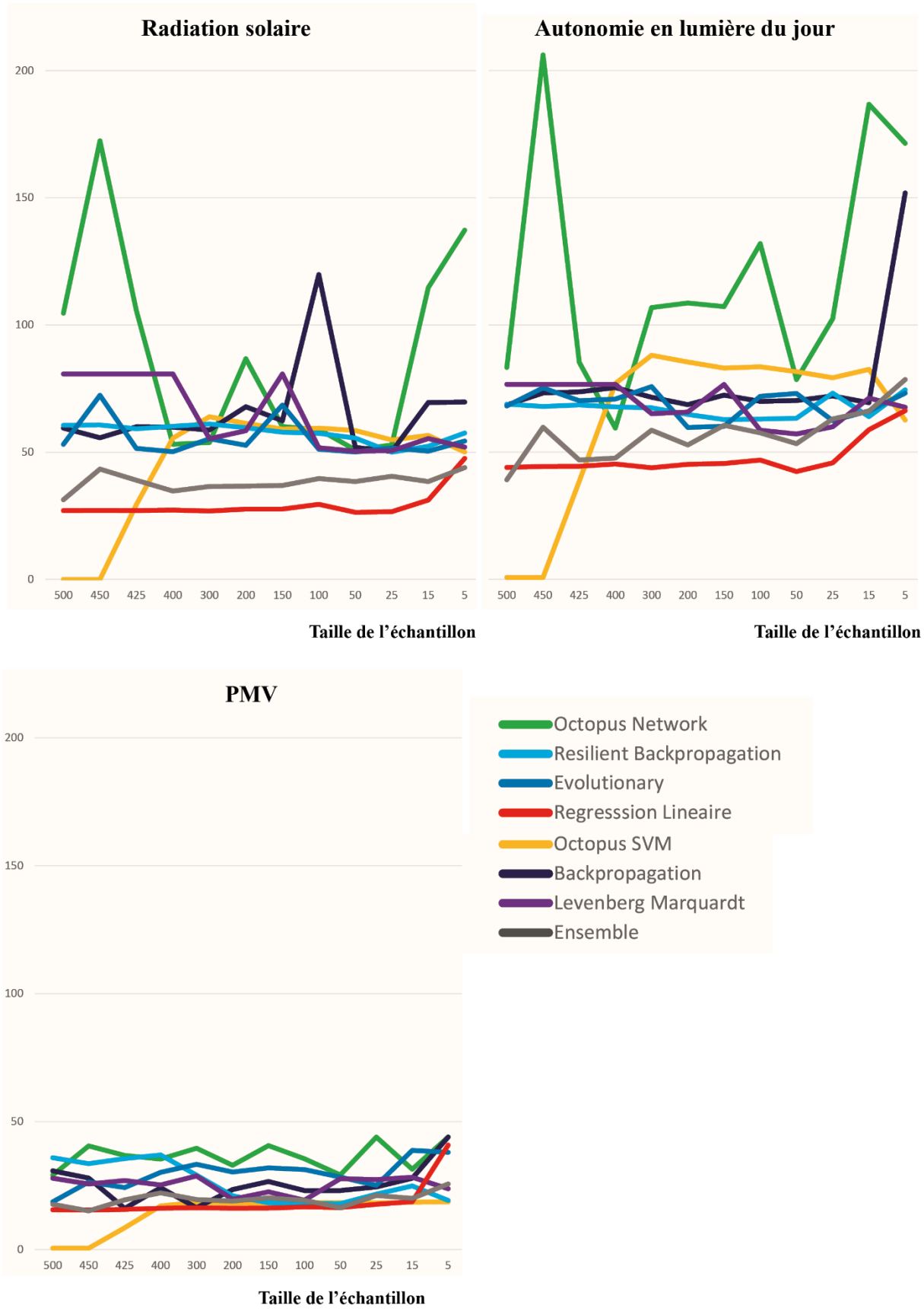
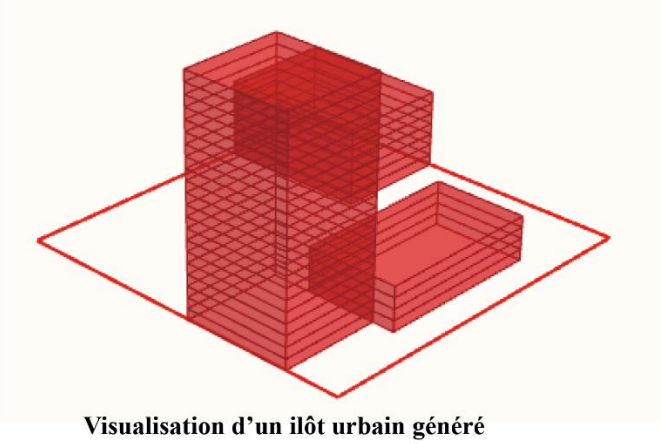


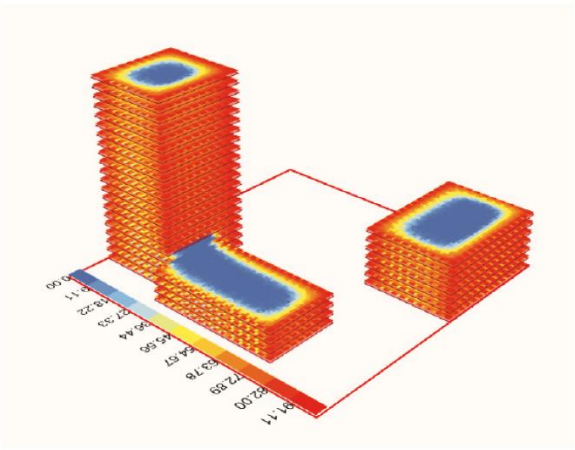
Figure 156 : La MAPE en fonction de la taille de l'échantillon pour les différents critères

Expérimentations pour la génération d'îlot

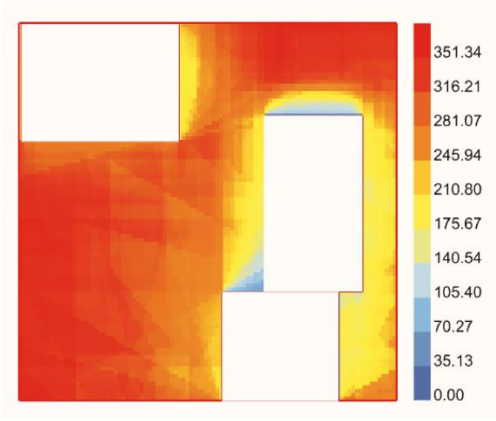
Un second modèle de substitution a été réalisé pour un problème de génération d'îlot. Trois bâtiments de hauteur, largeur, et longueur variables sont générés dans un terrain carré de 70 mètres de côté. Les coordonnées des 3 bâtiments varient, ainsi que leur orientation qui peut être Nord-Sud ou Est-Ouest. Une fonction de réparation est utilisée pour que les bâtiments n'entrent pas en collision mais peuvent avoir des murs mitoyens. Un exemple d'îlot est modélisé en 3D en Figure 157. Quatre critères de performances sont évalués : les consommations en chauffage, en refroidissement, à l'aide de Honeybee® et Energy Plus®, la radiation des espaces extérieurs avec Ladybug® et l'ALJ avec Honeybee® et Radiance®.



Visualisation d'un ilôt urbain généré



Visualisation de l'analyse d'ALJ



Visualisation de l'analyse de la radiation solaire des espaces extérieurs

Figure 157 : Modélisation 3D d'un îlot et visualisation des critères

A l'aide d'un modèle paramétrique 550 solutions sont générées. Il aura fallu plus de 180 heures de calcul au total, un peu plus d'une semaine, pour élaborer cette base de données, dont un extrait est présenté en Figure 158.

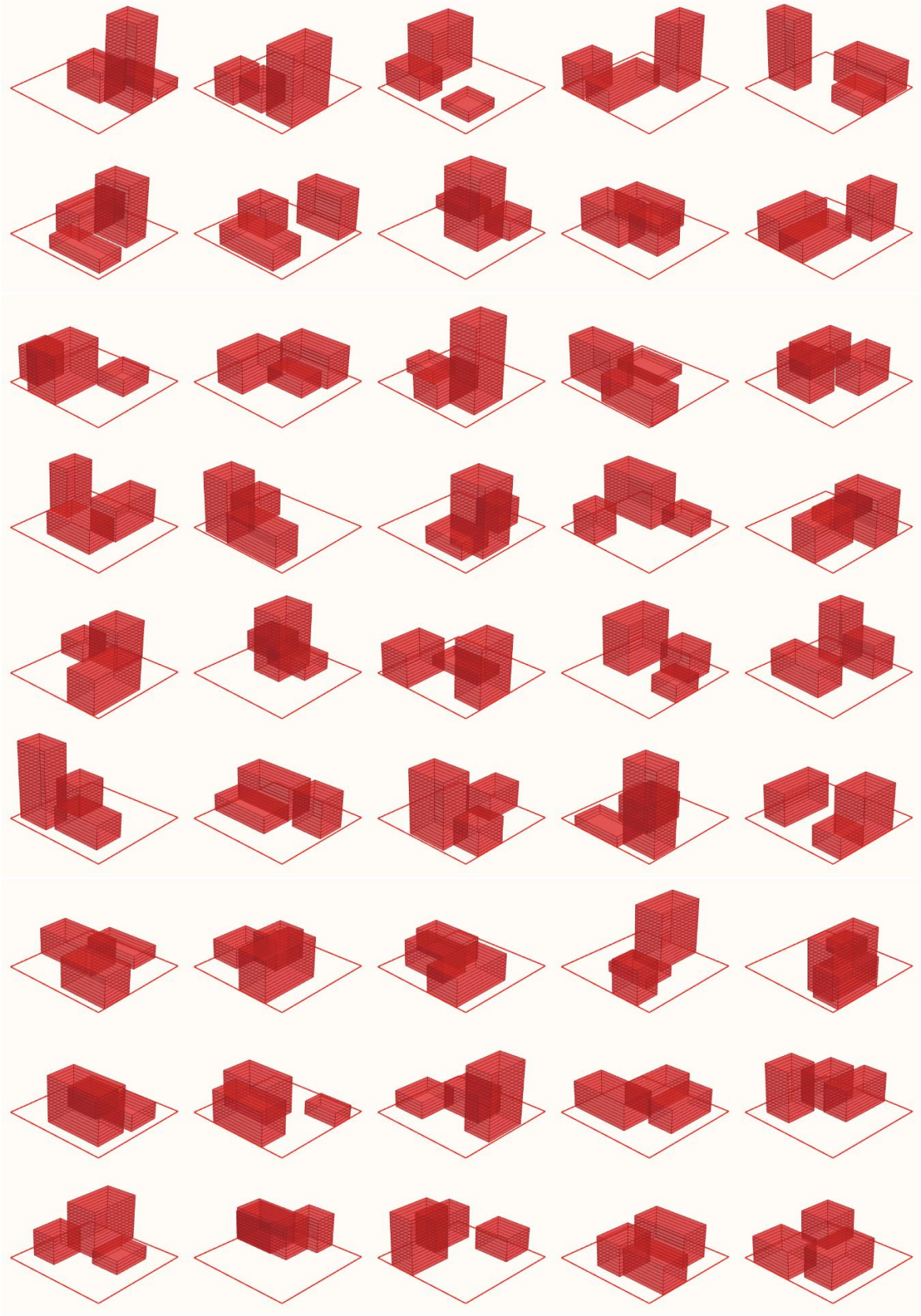


Figure 158 : Extrait de l'échantillon pour le problème d'optimisation de la morphologie d'un îlot

		Octopus Network	Octopus SVM	Resilient Backpropagation	Backpropagation	Evolutionary	Levenberg Marquardt	Regresssion multivariable	Ensemble
500	ALU	13,749945	17,796154	22,969291	22,070962	22,073907	25,805191	7,416471	12,355885
	RAD	7,895304	10,091758	17,36743	17,739121	17,237884	17,79819	5,629776	8,932301
	FROID	2,206159	3,384602	3,578263	3,538593	3,541462	3,791176	1,33128	2,088997
	CHAUD	4,382908	7,283792	10,596227	10,300861	11,161178	10,919767	2,47183	5,258117
300	ALU	12,665682	17,596556	25,07961	25,232834	26,415923	27,540309	7,789403	13,151247
	RAD	7,984666	9,830848	17,035599	17,559316	17,292753	18,177137	5,465267	8,55042
	FROID	1,893994	3,368653	3,66689	3,586298	3,559021	4,009461	1,313133	1,948416
	CHAUD	3,324052	7,194736	11,196438	11,068378	11,445081	11,162896	2,480044	4,787223
200	ALU	11,58128	17,407227	19,883474	19,412338	19,207005	21,540671	7,591799	11,787808
	RAD	8,33958	9,840781	17,342578	17,955291	17,247624	18,649543	5,939284	8,745322
	FROID	2,037226	3,356771	3,653827	3,48833	3,494926	3,722821	1,348694	1,877795
	CHAUD	3,949864	7,151807	8,321513	8,113066	8,95253	8,79961	2,558534	4,511085
100	ALU	12,609699	17,097798	15,171908	15,116864	15,706137	16,214693	8,99232	11,783045
	RAD	8,472267	9,635637	17,971626	18,303208	17,40129	18,496801	6,310879	8,599936
	FROID	2,080092	3,353002	2,919858	2,879867	2,892936	3,204696	1,397689	1,915021
	CHAUD	3,540327	7,046924	8,774378	8,82633	8,821947	9,14704	2,724281	4,384752
50	ALU	14,239074	17,022145	15,209842	15,505809	16,690074	16,2155	10,554942	12,258495
	RAD	12,725456	9,675532	17,026275	14,059567	13,204424	14,479421	9,802996	9,896155
	FROID	1,790998	3,37152	2,849781	2,987973	3,124676	3,06563	1,412582	1,753753
	CHAUD	4,071222	7,177163	9,551123	9,51629	9,508039	9,356617	3,068318	4,876777
25	ALU	21,002682	16,95127	15,625988	15,616367	16,359494	17,377587	12,065466	13,895865
	RAD	15,479953	12,114143	16,279855	16,927629	18,43539	17,408887	17,617047	11,88806
	FROID	2,812587	3,354996	2,927978	2,885981	3,512517	3,431249	1,692591	1,793681
	CHAUD	4,911748	7,322129	9,390509	9,413076	9,129753	9,325764	3,177545	5,292067
20	ALU	28,187834	17,186491	17,037646	15,66022	14,906034	16,126775	21,138928	15,588296
	RAD	12,207292	13,319036	16,926514	17,355507	17,535302	17,052945	23,520569	13,418534
	FROID	3,568048	3,390813	3,121586	2,948588	3,345469	3,130673	2,528293	1,729794
	CHAUD	4,144164	7,579607	9,115503	9,44982	9,776306	9,546607	4,682916	5,436729
15	ALU	25,59556	18,098093	17,103686	15,909269	17,106668	17,762726	57,862344	24,07436
	RAD	14,817108	12,307166	17,094956	14,457872	13,069615	16,942796	25,326932	9,505086
	FROID	4,355192	3,408989	3,386095	3,094835	3,844928	3,300127	17,295153	5,24602
	CHAUD	7,002493	7,488005	9,603748	9,60292	10,444629	9,242384	24,58979	8,687153
5	ALU	37,97092	22,03266	21,507449	20,000423	22,274274	21,26621	30,952593	21,446749
	RAD	24,556254	11,784263	14,377696	13,648851	14,275098	13,674005	21,627518	12,827505
	FROID	6,053791	3,421818	3,539323	3,177272	3,444138	3,566768	25,200527	6,791447
	CHAUD	14,242841	8,48934	9,796118	9,366127	10,76768	9,265961	21,343048	8,625271

Figure 159 : Ensemble des résultats numériques pour l'évaluation des modèles de prédiction pour des îlots (MAPE en %)

Le modèle paramétrique compte 17 paramètres, soit 6 par bâtiment (hauteur, longueur, largeur, orientation, coordonnée X, coordonnée Y), moins un paramètre. La hauteur du troisième bâtiment n'est pas paramétrable car elle est la résultante d'une contrainte de volume à bâtir. Ainsi, tous les îlots générés ont sensiblement le même volume.

Après avoir été entraîné, les algorithmes sont testés sur 50 solutions. La MAPE est ensuite calculée directement dans Grasshopper® pour les 7 algorithmes et un assemblage de 4 algorithmes (SVM Octopus, Network Octopus, Backpropagation, Regression multivariable), les 4 critères et 9 tailles d'échantillon différentes (5, 15, 20, 25, 50, 100, 200, 300 et 500 solutions). Les tableaux des résultats sont présentés en Figure 159.

Comme le montre les graphiques en Figure 160 et Figure 161, la performance des algorithmes dépend de la taille des échantillons et du critère évalué. Pour la prédiction des consommations en refroidissement, la MAPE descend en dessous des 5 % (avec un échantillon de 5 solutions seulement) avec les réseaux de neurones de LunchBox® et avec la SVM d'Octopus®. Pour la prédiction des consommations de chauffage, avec 25 solutions, la MAPE est inférieure à 5 % avec le réseau de neurone d'Octopus® et atteint 3,2 % avec une régression multivariable. Pour la prédiction des consommations énergétiques, lorsque l'échantillon est supérieur ou égal à 25 solutions, la régression multivariable est systématiquement la méthode la plus précise.

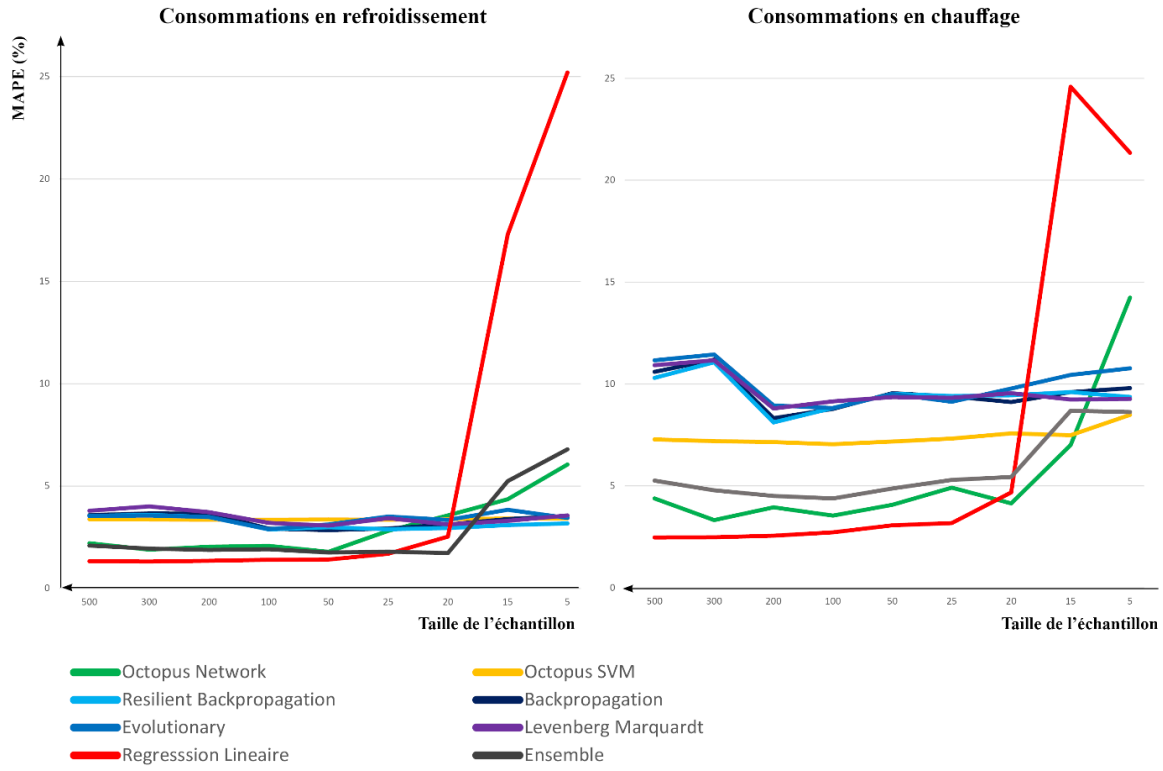


Figure 160 : La MAPE en fonction de la taille de l'échantillon pour les consommations énergétiques.

Pour la radiation solaire, aucun algorithme pour aucune taille d'échantillon n'atteint ce seuil des 5 %. La meilleure performance (5,5 %) est atteinte avec une régression multivariable avec un échantillon de 300 solutions. Pour l'ALJ, c'est aussi avec la régression multivariable avec 500 solutions, pour une MAJE de 7,4 %. A partir d'un échantillon de 50 solutions, la régression multivariable est toujours la méthode la plus performante quel que soit le critère évalué sur ce problème.

Certains algorithmes semblent plus robustes que d'autres, c'est le cas notamment de la SVM du plugin Octopus®, dont la MAPE ne s'envole pas pour de petits échantillons comme pour la régression multilinéaire et le réseau de neurone d'Octopus®. Les réseaux neuronaux proposés par LunchBox® tendent à être moins performants lorsque les échantillons dépassent 100 solutions. Sur ce problème, l'algorithme de réseau de neurones le plus performant est celui d'Octopus®.

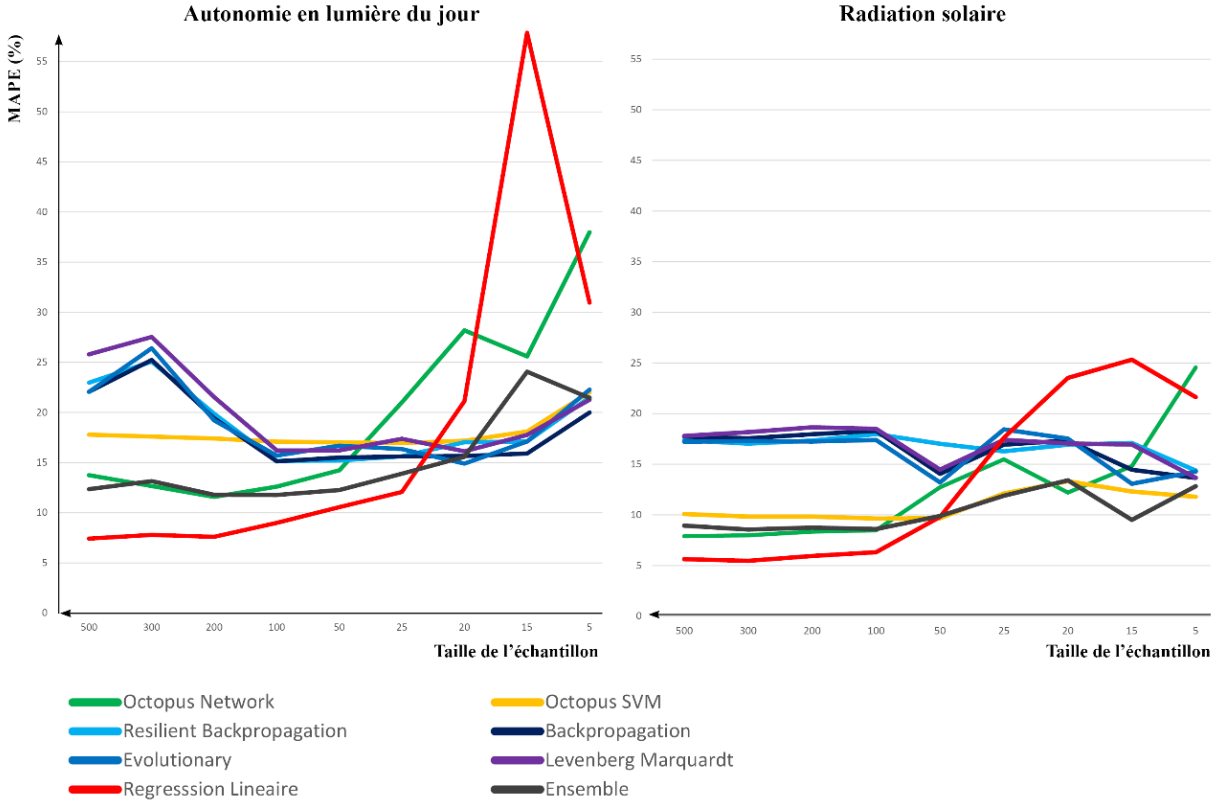


Figure 161 : La MAPE en fonction de la taille de l'échantillon pour l'ALJ et la radiation solaire

Quelle que soit la taille de l'échantillon et quel que soit l'algorithme utilisé, les prédictions sont toujours plus précises pour les critères de consommations énergétiques que pour la radiation solaire et l'ALJ (voir Figure 162). Avec la régression multivariable, plus la taille de l'échantillon augmente, plus la précision augmente. Cependant, plus l'échantillon est grand, plus augmenter sa taille a un faible impact, voire très faible, sur la précision des

prédictions, notamment pour les consommations énergétiques. Pour les consommations de chauffage, utiliser un échantillon de 500 solutions, plutôt que 25, demande plus de 170 heures de calcul en plus, pour une augmentation du MAPE de 0,7 %. Pour l'éclairage naturel et la radiation solaire, il faut un minimum de 100 solutions pour obtenir de bon résultats sur ce problème.

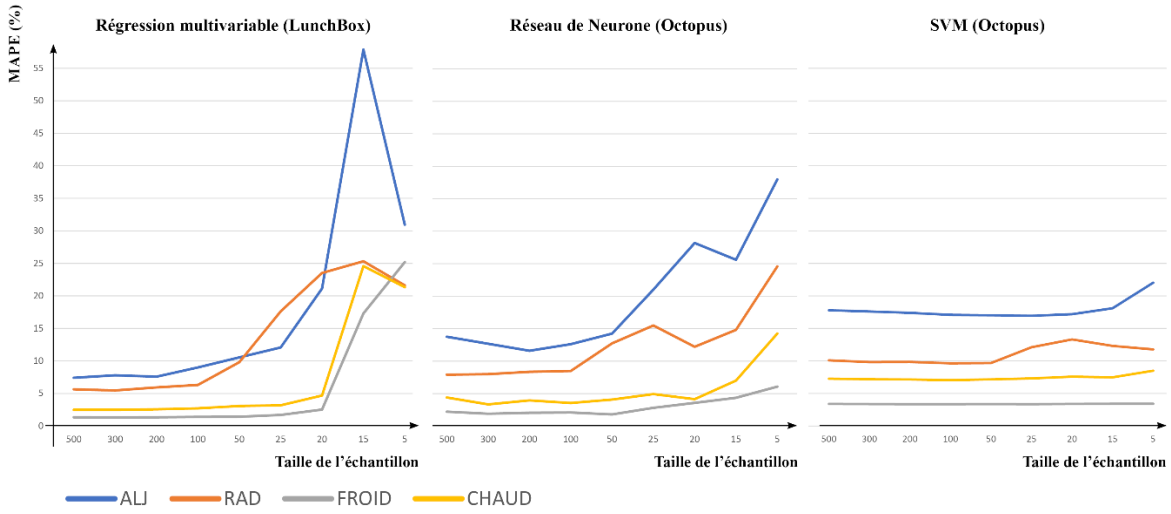


Figure 162 : La MAPE en fonction de la taille de l'échantillon pour les 3 algorithmes les plus performants

Sur ce problème, la méthode des ensembles, qui consiste à assembler les prédictions de plusieurs algorithmes, est parfois plus précise que les 7 algorithmes testés. C'est le cas notamment pour la prédiction des consommations en refroidissement avec un échantillon de 20 solutions, ou pour la radiation solaire avec des échantillons de 15 ou 25 solutions. Cette méthode peut aussi être intéressante dans le cas où on ne peut évaluer la performance des algorithmes de ML, par exemple si on souhaite développer un solveur d'optimisation en boîte noire, qui évaluerait des solutions lors de la première génération avec de la simulation, puis utiliserait des modèles de substitution entraînés sur cette population pour évaluer les solutions des générations suivantes. Sans un tel outils, dans la pratique, il n'est pas très chronophage, comparé au temps nécessaire pour créer une base de données de tester plusieurs composants avant de choisir lequel utiliser pour lancer une optimisation.

S'il n'est pas possible de tirer des conclusions générales sur la base de nos deux cas d'études, ces derniers permettent de nous éclairer sur la faisabilité des modèles de substitution dans un contexte professionnel. En effet, pour être utile en pratique pour faire de l'optimisation, un modèle de substitution entrainé pour un problème spécifique doit pouvoir être précis avec de petits échantillons. Il semble qu'un tel modèle puisse être efficace pour la prédiction des

besoins énergétiques, notamment à l'échelle de l'îlot où les simulations sont coûteuses en temps de calcul. Nous avons pu élaborer des modèles précis avec une base de données de 20 solutions (7 heures de calcul environ pour un ilot) avec une régression multivariable.

Cependant, il semble que ces algorithmes sont moins adaptés à la prédiction des indicateurs de confort comme l'éclairage naturel, la radiation solaire ou le confort thermique. Ils ne sont pas suffisamment précis avec des échantillons de petites tailles. Le cas particulier de l'algorithme de SVM d'Octopus a permis de créer des modèles très précis à l'échelle de la pièce, cependant il faudrait vérifier que cet algorithme peut atteindre un tel niveau de performance sur d'autres problèmes pour pouvoir recommander une utilisation systématique. Aussi, cet algorithme a nécessité un échantillon de 450 solution, ce qui pour la pratique reste particulièrement conséquent, on ne peut parler d'un réel gain de temps.

Ainsi, nous avons poursuivi nos recherches et identifié et testé des algorithmes et techniques plus récentes pour la création de modèles de substitution pour les indicateurs de confort visuel et thermique.

2.3.2. Prédire des résultats graphiques

Avec certains algorithmes d'apprentissage automatique, il est possible de prédire des images, c'est le cas des GAN pour « *generative adversal network* » en anglais, traduit « *réseaux antagonistes génératifs* ». Il s'agit d'un type particulier de réseaux de neurones permettant de faire de l'apprentissage semi-supervisé. Cette technique d'intelligence artificielle intéresse particulièrement les architectes aujourd'hui pour ces capacités génératives.

Réseaux de neurones antagonistes

Un algorithme de *Deep Learning*

Il semble important de distinguer les techniques de « *machine learning* » au sens large, du « *deep learning* » (« *apprentissage profond* » en français) qui est une sous-catégorie du « *machine learning* ». Le *deep learning* est capable de traiter des données non-structurées. Aujourd'hui, lorsqu'on parle d'algorithmes d'intelligence artificielle, il est sous-entendu qu'on parle plutôt des techniques récentes de *deep learning* comme les réseaux de neurones artificiels, et non des techniques plus anciennes comme les algorithmes génétiques ou les modèles de régressions.

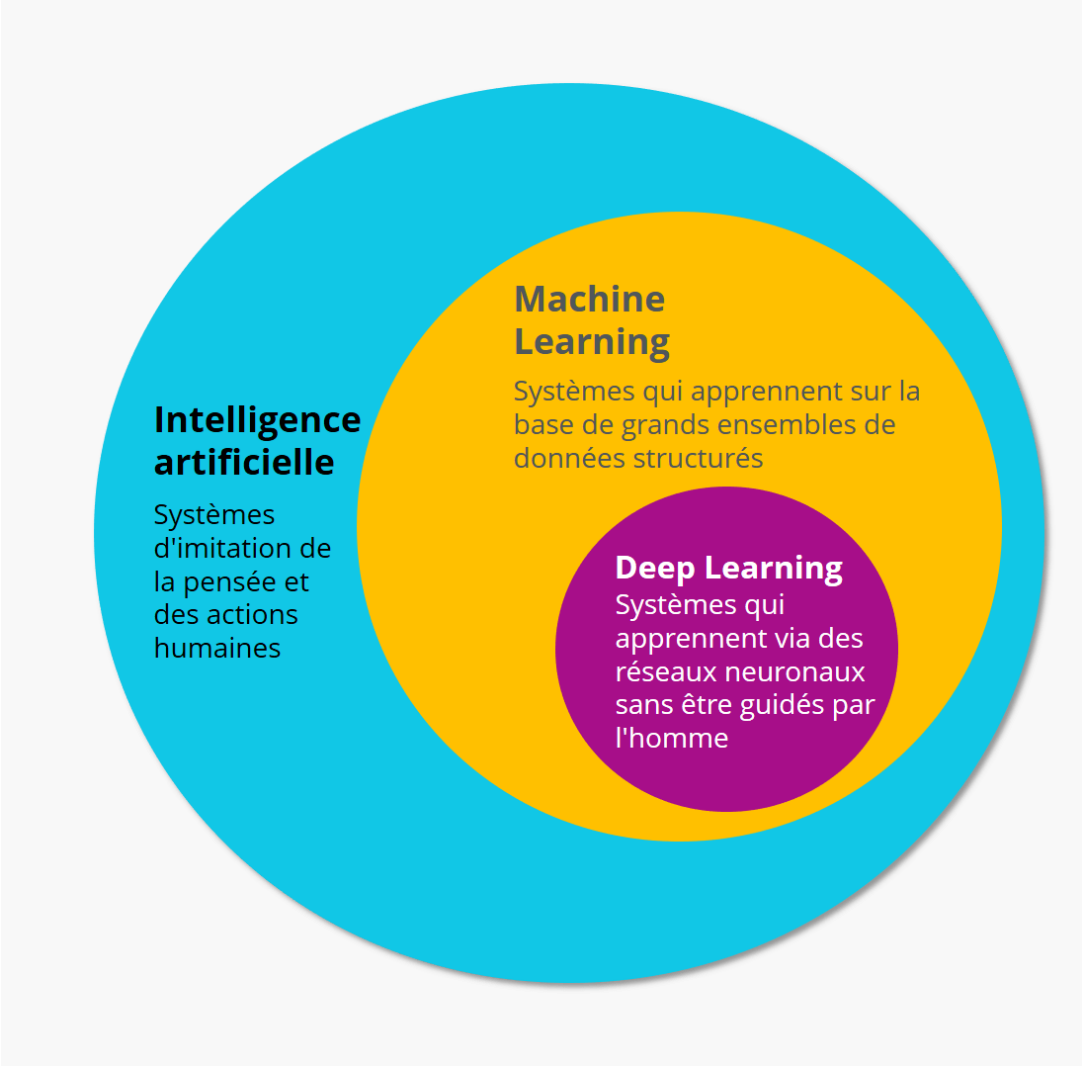


Figure 163 : Machine learning vs Deep Learning

Les GAN sont des algorithmes d'apprentissage non supervisés relativement récents appartenant à la catégorie *Deep Learning*. Ce sont les algorithmes utilisés entre autres pour générer de l'art car ils sont capables de générer des images particulièrement réalistes. Un GAN met en compétition deux réseaux de neurones. « *Le premier réseau est le générateur, il génère un échantillon (ex. une image), tandis que son adversaire, le discriminateur essaie de détecter si un échantillon est réel ou bien s'il est le résultat du générateur* » (Goodfellow et al., 2016). A chaque fois que le discriminateur surprend le générateur avec un échantillon non réel, ce dernier s'améliore jusqu'à être en mesure de duper le discriminateur.

Applications en Architecture

D'après D. Newton, les GAN sont les plus adaptés à la conception architecturale (Newton, 2019). Il existe encore peu de recherche sur le sujet en architecture et en urbanisme. Cette méthode ne nécessite pas d'expliciter des paramètres puisque l'algorithme apprend à partir d'images, ce qui en fait un outil puissant. David Newton utilise cette technique pour générer des plans à partir de références d'architectes connus, mais aussi pour générer des façades à partir d'un style architectural et pour des formes de bâtiments (Newton, 2019). Stanislas Chaillou utilise cette technique pour l'agencement spatial intérieur (Chaillou, 2020). Les GAN peuvent aussi être utilisés pour générer des îlots urbains à partir de l'analyse de tissus urbains existants (Fedorova, 2021).

Figure 164 : Génération de façades gothiques avec un GAN (source: (Newton, 2019))

Cette technique est parfois utilisée pour faire de la prédiction de la performance. Au lieu de prédire les valeurs des résultats de simulation, les GAN prédisent le rendu graphique de « *heat map* », les traductions visuels des simulations.

Applications pour la prédiction des performances

Jeffrey Landes, data scientist chez Spacemaker Ai explique dans l'ouvrage de Stanislas Chaillou sur l'intelligence artificielle comment il utilise les GAN pour réduire les temps de calcul pour la prédiction de l'écoulement des vents, ou l'ensoleillement à l'aide de GAN (Chaillou, 2021).

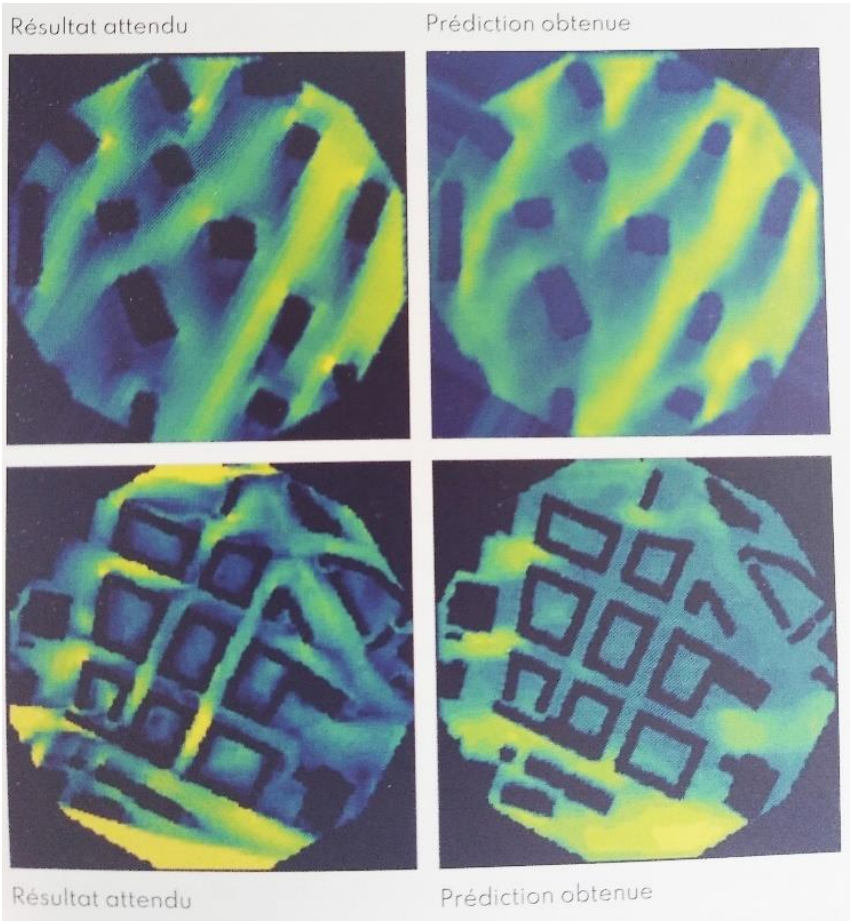


Figure 165 : Exemples prédiction de l'écoulement des vents (source:(Chaillou, 2021))

Une méthode pour réaliser des prédictions de l'écoulement des vents avec un GAN nommé pix2pix (Isola et al., 2017) pour étudier les flux au niveau des piétons a fait l'objet d'une publication (Mokhtar et al., 2020). Duering et ses co-auteurs ont utilisé cette technique pour prédire le micro climat à l'échelle d'un îlot urbain (Duering et al., 2020). Deux indicateurs sont utilisés : la radiation solaire et la vitesse des vents. Les données servant à entrainer l'algorithme ont été générées avec Grasshopper®, Ladybug® pour la radiation et OpenFOAM® pour les vents. Les auteurs précisent que les entraînements ont duré plusieurs mois.

Pix2pix a aussi été utilisé pour élaborer une méthode de prédiction de l'éclairement naturel d'un espace intérieur (Jia, 2021). La méthode est destinée aux phase amonts de conception et implique une étape de voxelisation (voxel dont la taille est paramétrable), le modèle 3D est ensuite transformé en image 2D. L'algorithme utilisé pour la prédiction est nommé « *Open Neural Network Exchange* » (ONNX model). La méthode a été implémenté dans un outil Grasshopper® nommé ArchiGAN pour le moment inaccessible en opensource.Pix2pix a aussi été testé pour la prédiction de la connectivité visuelle et spatiale (Tarabishy et al., 2020).

Les GAN sur Grasshopper®

Il existe très peu d'outils finalisés sur Grasshopper® pour démocratiser l'utilisation des GAN. A notre connaissance, seul le plugin Pug® intègre deux composants pour utiliser des GAN. Un exemple est donné par son développeur, il consiste à utiliser l'algorithme pour déchiffrer des chiffres et des lettres écrits à la main (Apellániz et al., 2023). Cependant, ces composants ne permettent pas de reproduire un pix2pix.

Sur Github, il est possible de trouver des projets d'implémentation de GAN pour Grasshopper®, notamment SmokingGAN (https://github.com/enmerk4r/SmokingGAN), ou encore Daylight Gan (https://github.com/mitevpi/daylight-gan) qui utilise pix2pix. Il s'agit encore de projet, pas complètement aboutis ne pouvant être utilisé sans usage d'un langage informatique.

Ainsi, pour pouvoir expérimenter les GAN pour la prédiction de critères de performance comme l'éclairement naturelle ou le rayonnement solaire, nous avons dû créer un script python à partir d'un tutoriel Tensorflow Core (*pix2pix,* 2023).

Fonctionnement de pix2pix

Figure 166 : Génération de photos de façades avec pix2pix (source: https://www.tensorflow.org/tutorials/generative/pix2pix?hl=fr)

Pix2pix, en français « image-à-image » cherche à translater une image d'un domaine source vers un domaine cible à partir de paires existantes. Il peut être utilisé pour un grand nombre de tâches différentes comme transformer des croquis en photo, transformer des vues aériennes en plan Google Maps® ou encore générer des photos de façades à partir d'un schéma (voir Figure 166). Dans notre cas, le domaine source est celui de la représentation géométrique en plan de solution générée avec un modèle paramétrique, et le domaine cible est le même plan avec une représentation 2D d'une simulation environnementale.

Comme tous les GAN, pix2pix est composé de deux réseaux de neurones qui s'entraînent ensemble. Le premier réseau, appelé « générateur » s'entraîne à générer des fausses images (des prédictions). Le second réseau, appelé « discriminateur » s'entraîne à distinguer les fausses images (celles du générateur) des vraies images (celles issues des simulations). Le générateur de pix2pix est un U-net modifié qui permet de faire de la segmentation sémantique, c'est-à-dire à faire de la classification de pixels. Il se compose d'un encodeur, qui va traiter les données d'entrée afin d'en donner une nouvelle représentation, et d'un décodeur, qui va chercher à reconstruire les données de départ à partir des données « encodées » par l'encodeur.

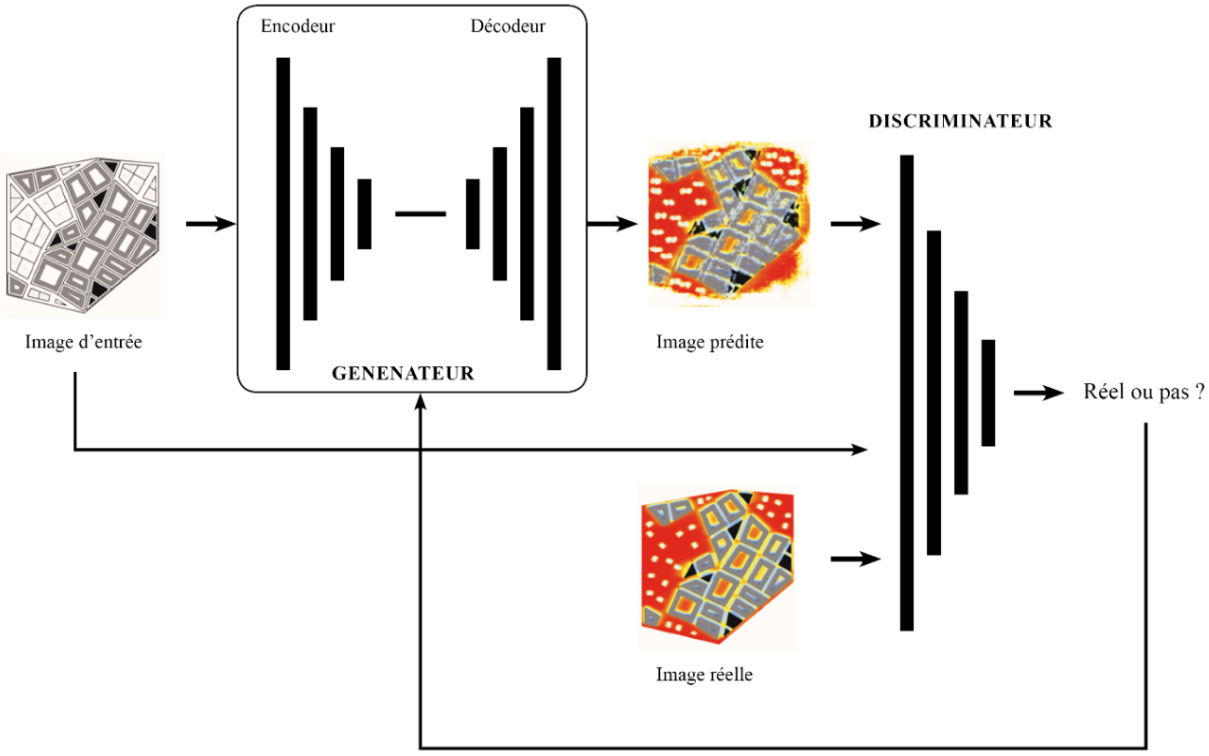


Figure 167 : Architecture de pix2pix

Le discriminateur de pix2pix est un classificateur convolutif PatchGAN (Isola et al., 2017). Il produit des cartes de prédictions « réel/faux ». Chaque pixel de cette carte est une région de l'image d'entrée du discriminateur. Les données de sorties du générateur permettent d'entraîner le discriminateur et inversement. L'architecture générale de pix2pix est schématisée en Figure 167. Un cycle complet du jeu de données d'entraînement s'appelle une « époque », il faut plusieurs époques pour obtenir des résultats satisfaisants.

Expérimentations des GAN

Nous avons cherché à expérimenter l'utilisation des GAN pour la prédiction de la performance sur des problèmes théoriques comparables à ceux qu'on peut rencontrer dans la

pratique. L'objectif est avant tout d'appréhender la faisabilité de cette méthode pour des applications en agence (temps de calcul, taille de l'échantillon), et de voir si la prédiction des résultats graphiques est suffisamment précise pour être utilisée dans un processus d'optimisation, voire plus précise qu'avec les algorithmes de ML d'Octopus® ou LunchBox®.

Ainsi, nous avons testé pix2pix sur trois problèmes différents : un détail d'enveloppe, un problème de génération d'îlots et un problème de génération de quartier. Les deux premiers problèmes sont similaires à ceux utilisés pour les expérimentations sur la prédiction des résultats numériques.

Expérimentations pour un détail d'enveloppe

Nous avons commencé par tester pix2pix pour la prédiction de la radiation solaire et de l'ALJ sur notre problème de dimensionnement de fenêtre et de protection solaire pour un bureau. Le modèle paramétrique utilisé est le même que pour l'expérimentation des algorithmes de ML pour la prédiction des valeurs numériques, seulement ici, une capture d'écran est réalisée en plan et en élévation après chaque évaluation.

Une fois les 500 évaluations effectuées, les 4 captures d'écran par solution sont assemblées sur la même image avec à gauche le plan et l'élévation sans les évaluations, et à droite le plan et l'élévation avec les résultats de l'étude de radiation et de lumière.

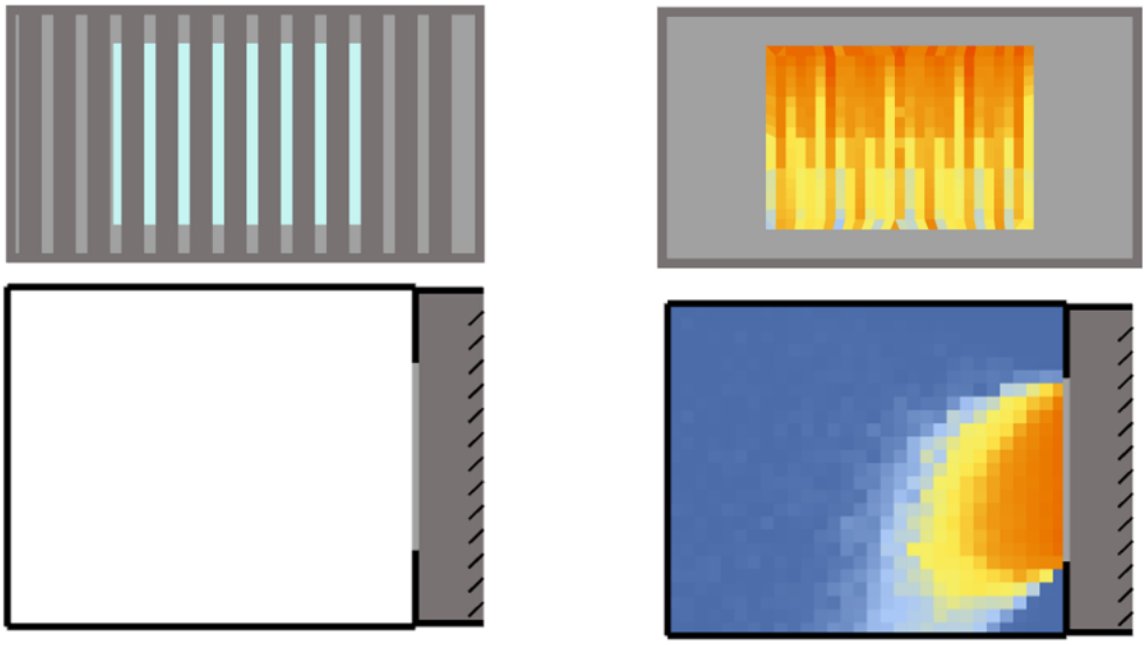


Figure 168 : Paire d'images issues de la base de données d'entraînement

Plusieurs tests sont ensuite effectués, avec des tailles d'échantillons différentes (50, 100, 250, 500 paires d'images), et avec un temps d'apprentissage différents (100, 500, 1 000, 2 000, 10 000 époques). Ainsi, avec un ordinateur 16 CPU Intel Xeon at 2.4 GHz, les temps de calcul varient selon la taille de l'échantillon, mais surtout selon la quantité d'époques : entre 3 et 4 minutes pour 100 époques, 13 et 16 minutes pour 500 époques, 26 et 31 minutes pour 1 000 époques, 52 et 1h02 pour 2 000 époques, entre 4h17 et 4h52 pour 10 000 époques.

Nous présentons les résultats des tests pour 3 solutions (Figure 170, Figure 171, Figure 172), dont les résultats des simulations sont présentés sur la figure ci-dessous :

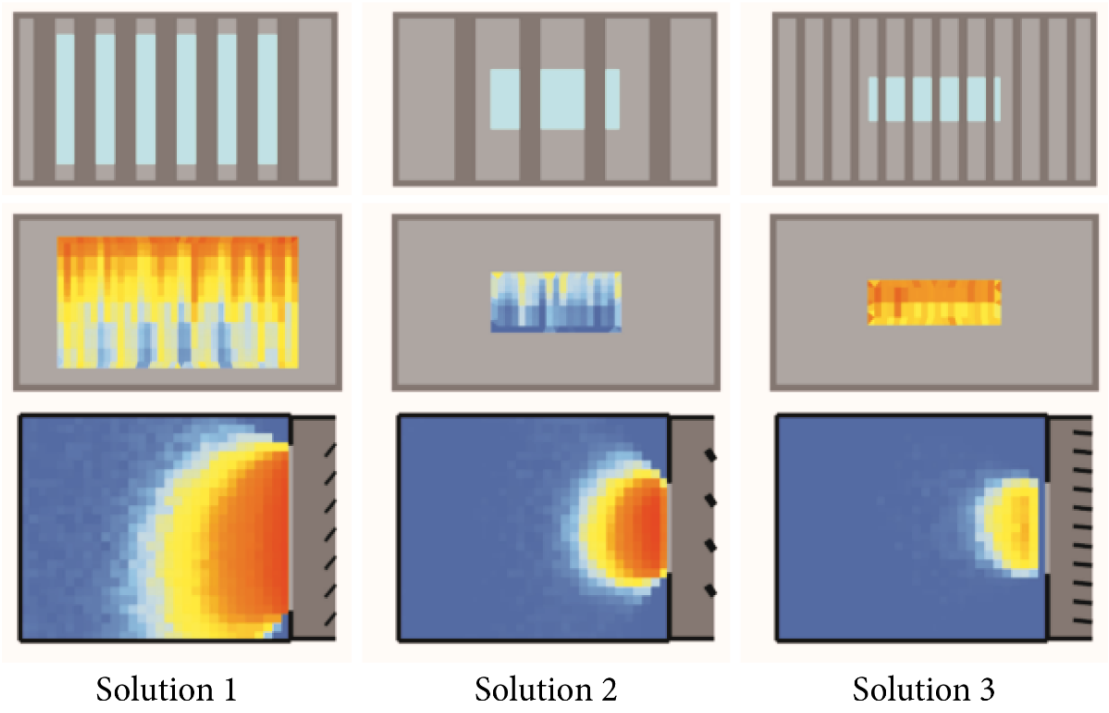


Figure 169 : Images réelles de 3 solutions

Plus la taille de l'échantillon et le nombre d'époques augmentent, plus la prédiction semble précise à l'œil. Nous constatons que pour ce problème, il faut une base de données de minimum 100 paires d'images et 2000 époques que ce soit pour la radiation solaire ou l'ALJ pour obtenir des résultats acceptables. Nous avons testé l'algorithme sur une vingtaine de configurations et les images prédites finissent toujours par se rapprocher très fortement des images réelles. Les GAN semblent être une solution prometteuse comme substitut aux régressions pour la prédiction de l'éclairement et la radiation, même s'il serait intéressant de chercher à retrouver des valeurs numériques sur plusieurs centaines de cas prédits pour mesurer la précision et la robustesse d'un modèle de substitution basé sur l'usage de GAN.

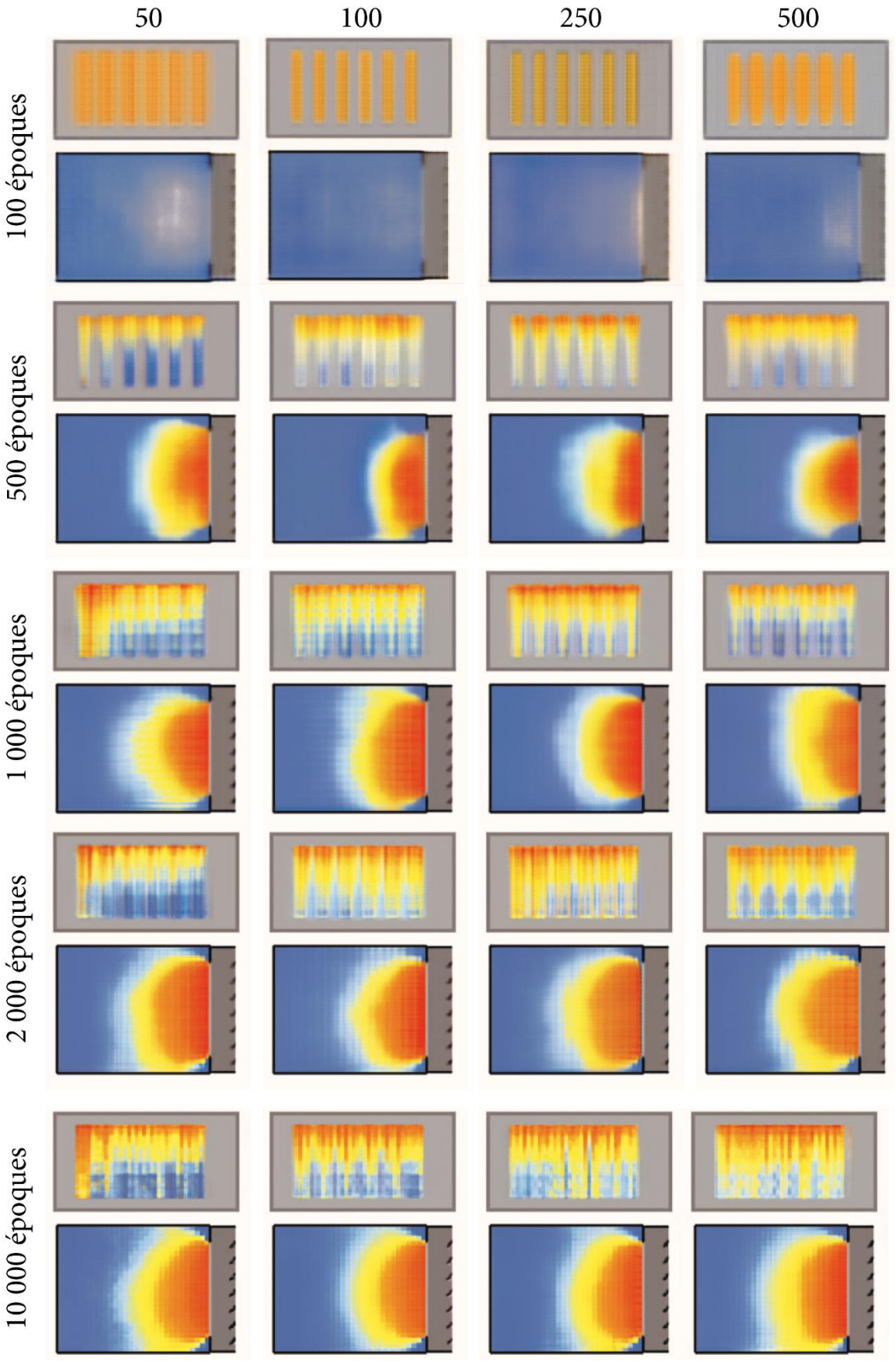


Figure 170 : Résultats graphiques solution 1

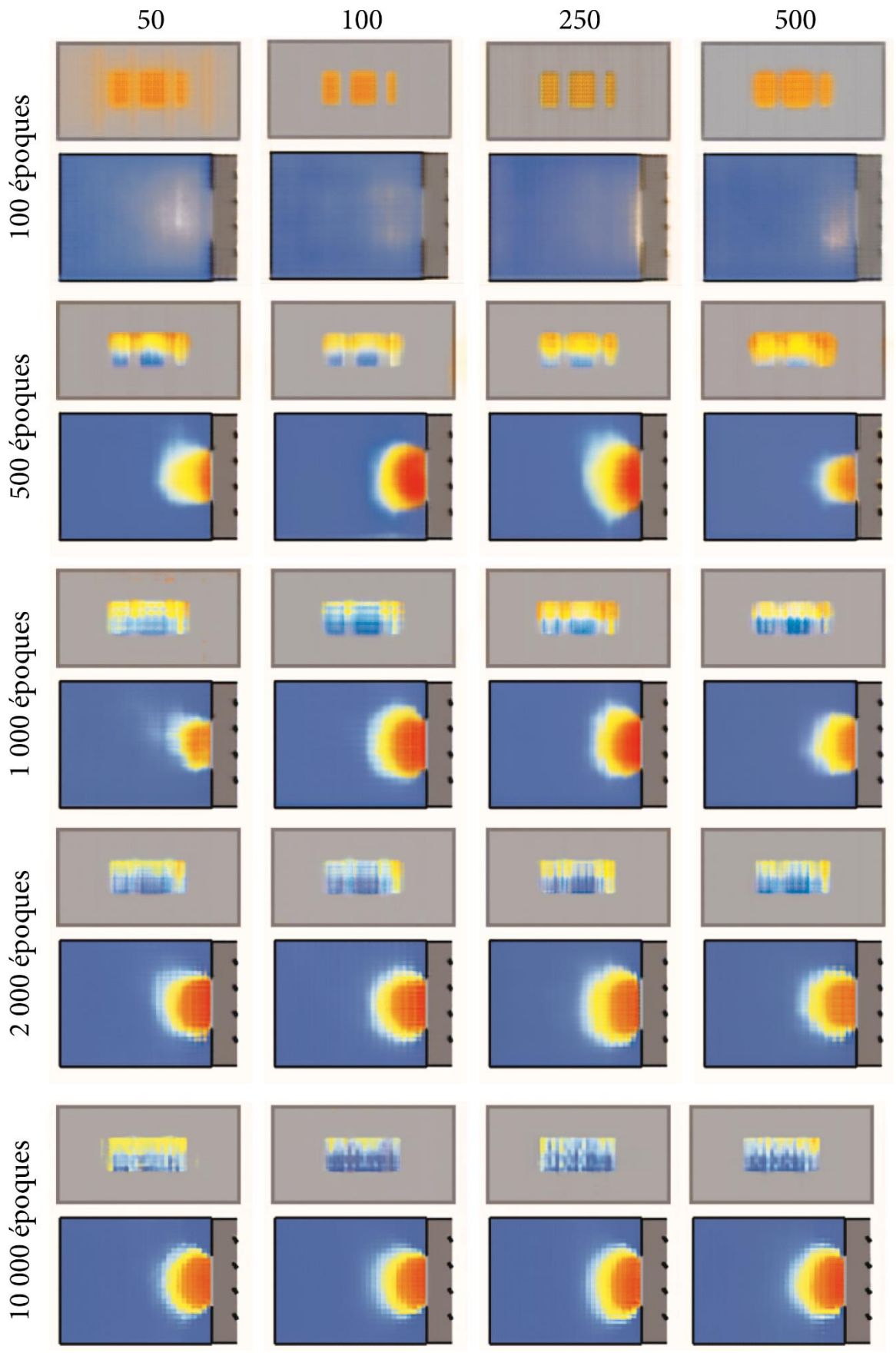


Figure 171 : Résultats graphiques solution 2

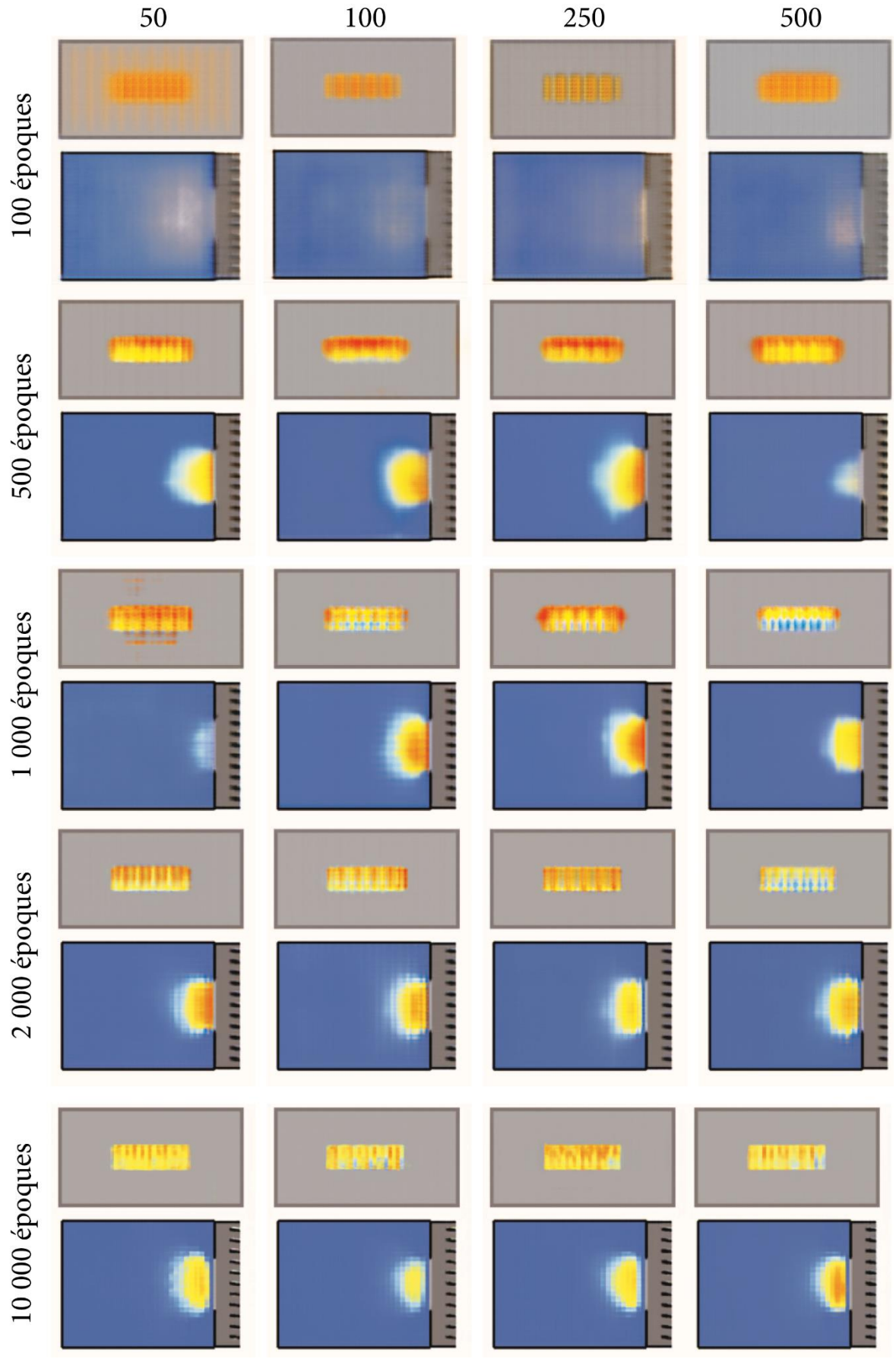


Figure 172 : Résultats graphiques solution 3

Expérimentations pour la génération d'îlots

Pour cette seconde expérimentation, nous réutilisons le problème déjà étudié de la génération d'un îlot sur un terrain carré. L'objectif de cette seconde expérimentation est d'étudier la robustesse de l'algorithme lorsqu'il est testé sur des formes de bâtiments topologiquement très différentes des formes utilisées pour son entraînement. Pour intégrer le paramètre de hauteur des bâtiments dans l'image d'entrée, nous colorions les masses en niveau de gris en fonction de leur hauteur, le noir correspondant à la hauteur maximale.

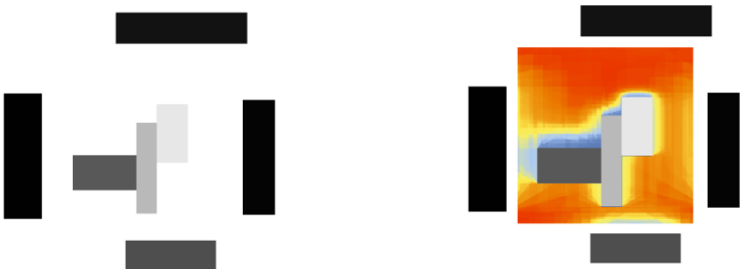


Figure 173 : Paire d'images issues de la base de données d'entraînement

Ainsi, nous avons généré une base de données constituée de 1 000 paires d'images d'îlots urbains constitués de trois bâtiments parallélépipédiques, où la radiation solaire des espaces extérieurs est évaluée avec un contexte urbain qui est le même pour toutes les solutions générées (voir Figure 173). L'algorithme a ensuite été entrainé sur 5 000 époques (4h37 minutes). Puis nous avons modifié le modèle paramétrique à plusieurs reprises pour générer d'autres images pour tester l'algorithme : des îlots avec 5 bâtiments, des bâtiments cylindriques, des bâtiments triangulaires ou en ellipse.

Comme attendu, l'algorithme arrive très bien à prédire la radiation sur les formes avec lesquels il a été entraîné comme pour l'exemple de la Figure 174. Contre toute attente, pour les autres formes, la précision est telle qu'il est difficile de distinguer à l'œil nu une image réelle d'une image prédite (voir Figure 175 et Figure 176).

Figure 174 : A gauche image réelle, à droite image prédite

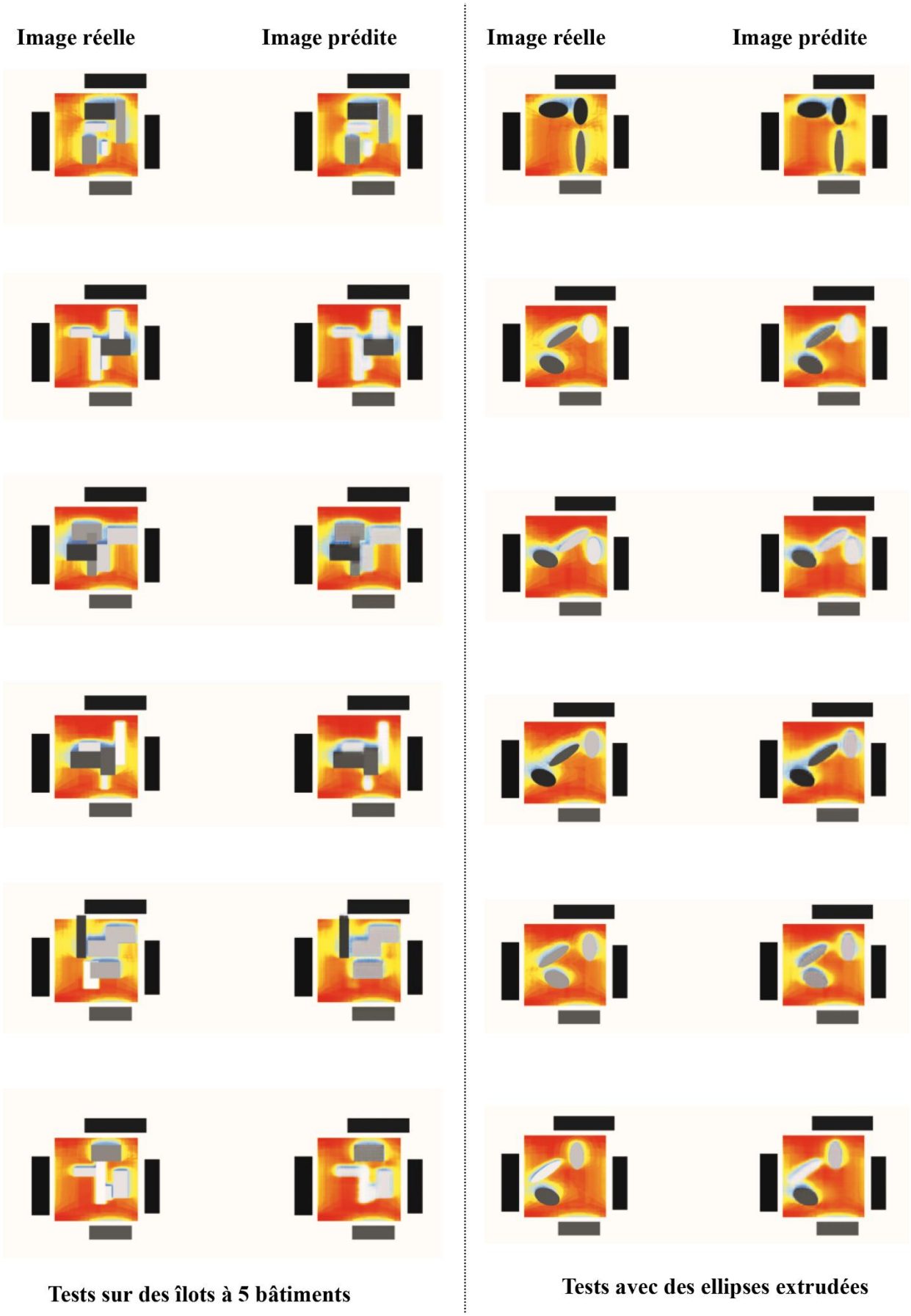


Figure 175 : Tests avec 5 bâtiments et des ellipses extrudées

Figure 176 : Tests avec des cylindres et des triangles extrudés

Nous avons ensuite réalisé une seconde expérimentation à l'échelle de l'îlot en créant une nouvelle base de données de 1 000 solutions avec un contexte urbain identique pour toute la base de données mais qui diffère de la première base.

L'algorithme a été entrainé sur 5 000 époques (2h53min02s). Le modèle paramétrique ayant servi à l'élaboration de la base de données a ensuite été modifié pour générer des îlots avec des contextes urbains différents. Les résultats, en partie présenté en Figure 177, montrent des prédictions un peu moins précises qu'avec les variations des formes. Il est possible de distinguer les images à l'œil nu, mais ces différences semblent suffisamment faibles pour impacter la prise de décision d'un architecte.

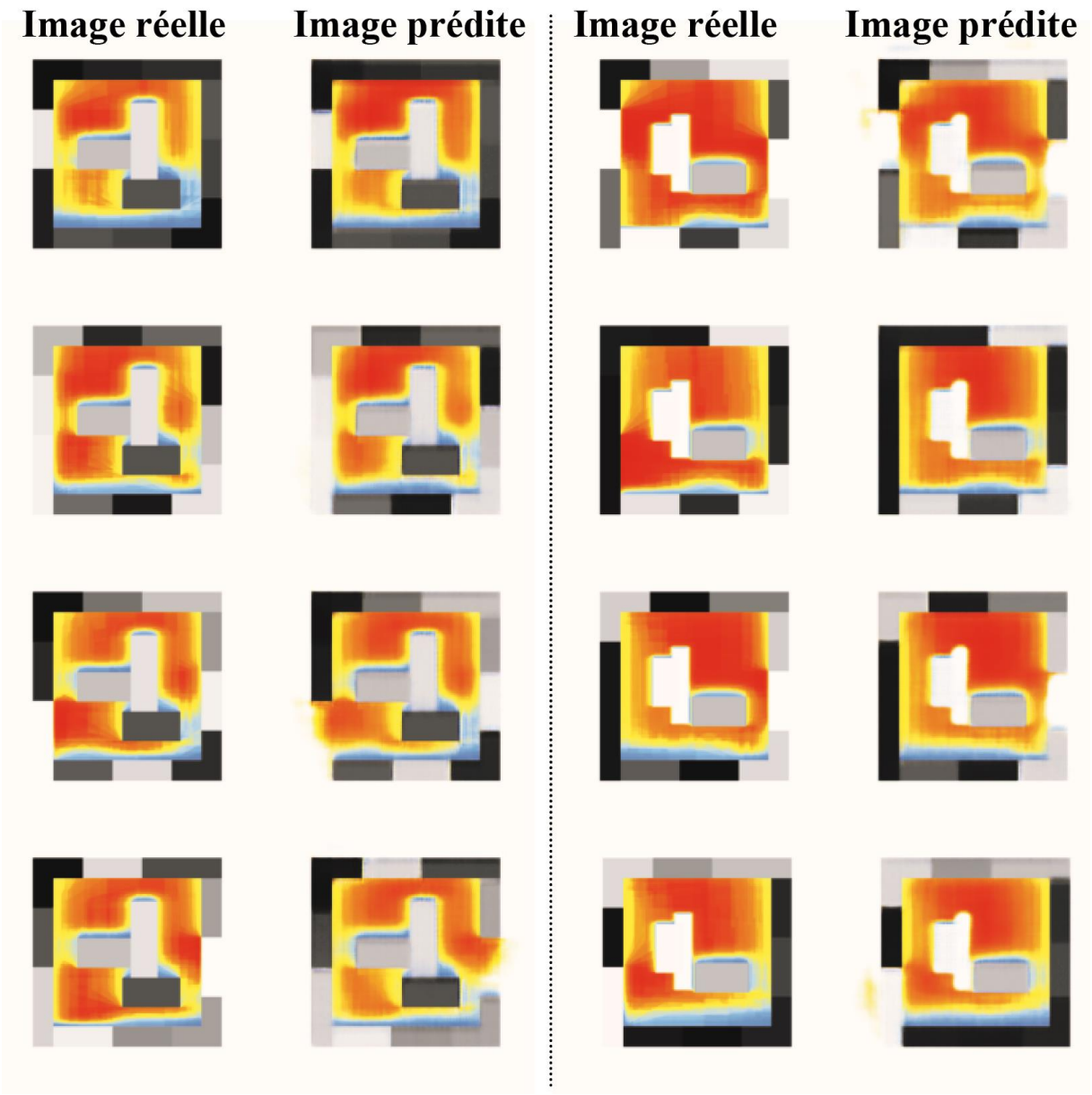


Figure 177 : Résultats avec une variation du contexte

Enfin, nous avons réitérer l'expérience des variations des formes d'îlots avec un autre critère : l'écoulement des vents. L'analyse des vents a été réalisée avec le plugin Butterfly® et le logiciel de simulation OpenFoam®. Deux représentations différentes des résultats ont été capturées. Une représentation qui ne tient compte que de la vitesse des vents, et une représentation qui prend en compte la vitesse et l'orientation des vents. L'analyse de l'écoulement des vents étant très chronophage (beaucoup plus que la radiation solaire), les entraînements (un pour chaque type de représentation) ont été réalisés avec une base de données de 570 solutions seulement, sur 5 000 époques (2h02min22s).

Les premiers résultats où l'algorithme a été testé avec les mêmes formes que celles utilisées pour l'entraînement montrent que si l'algorithme est fiable pour la prédiction de la vitesse des vents, il a plus de mal à prédire leur orientation (voir Figure 178).

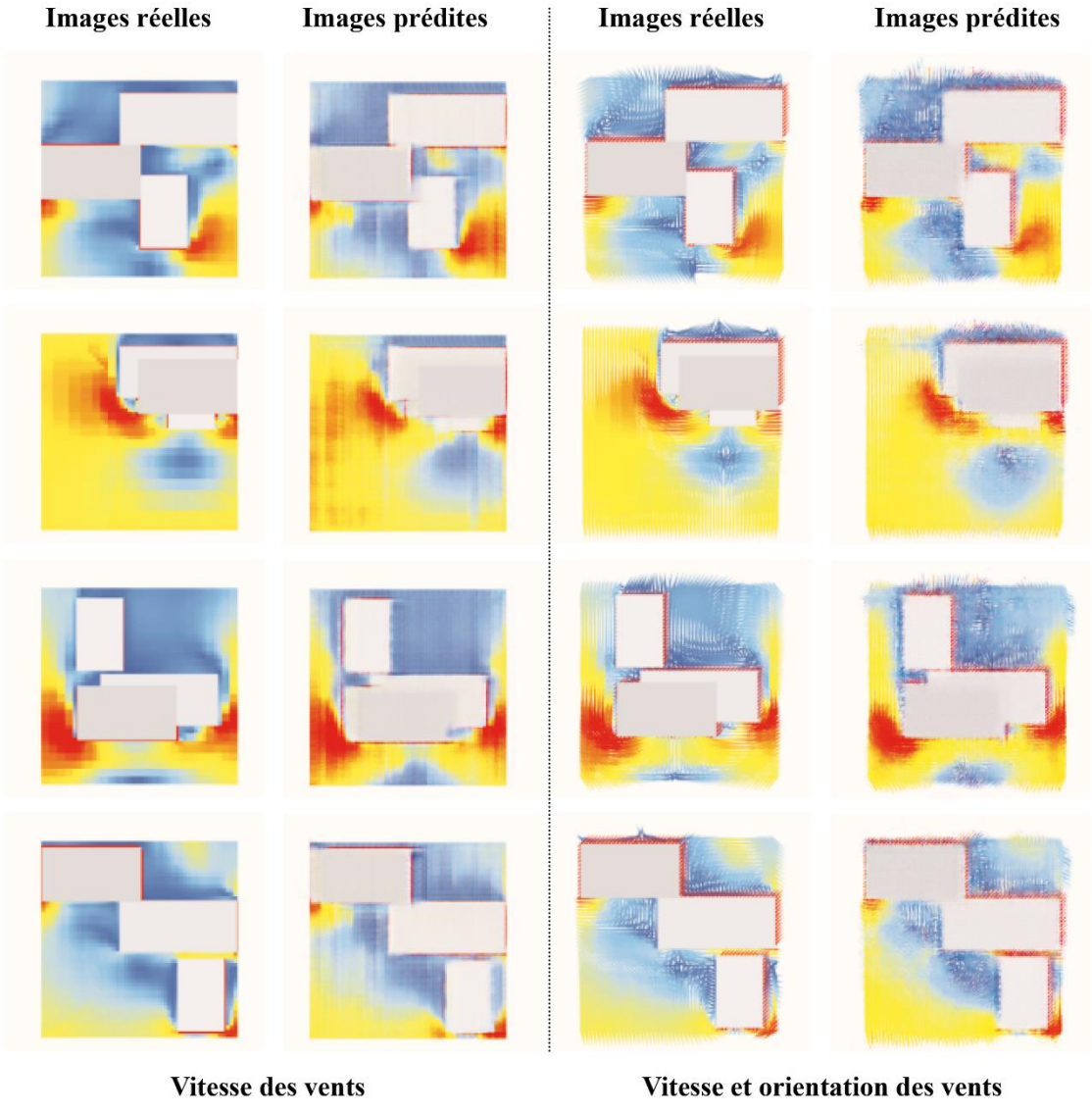


Figure 178 : Prédiction à gauche de la vitesse des vents, à droite la vitesse et l'orientation des vents

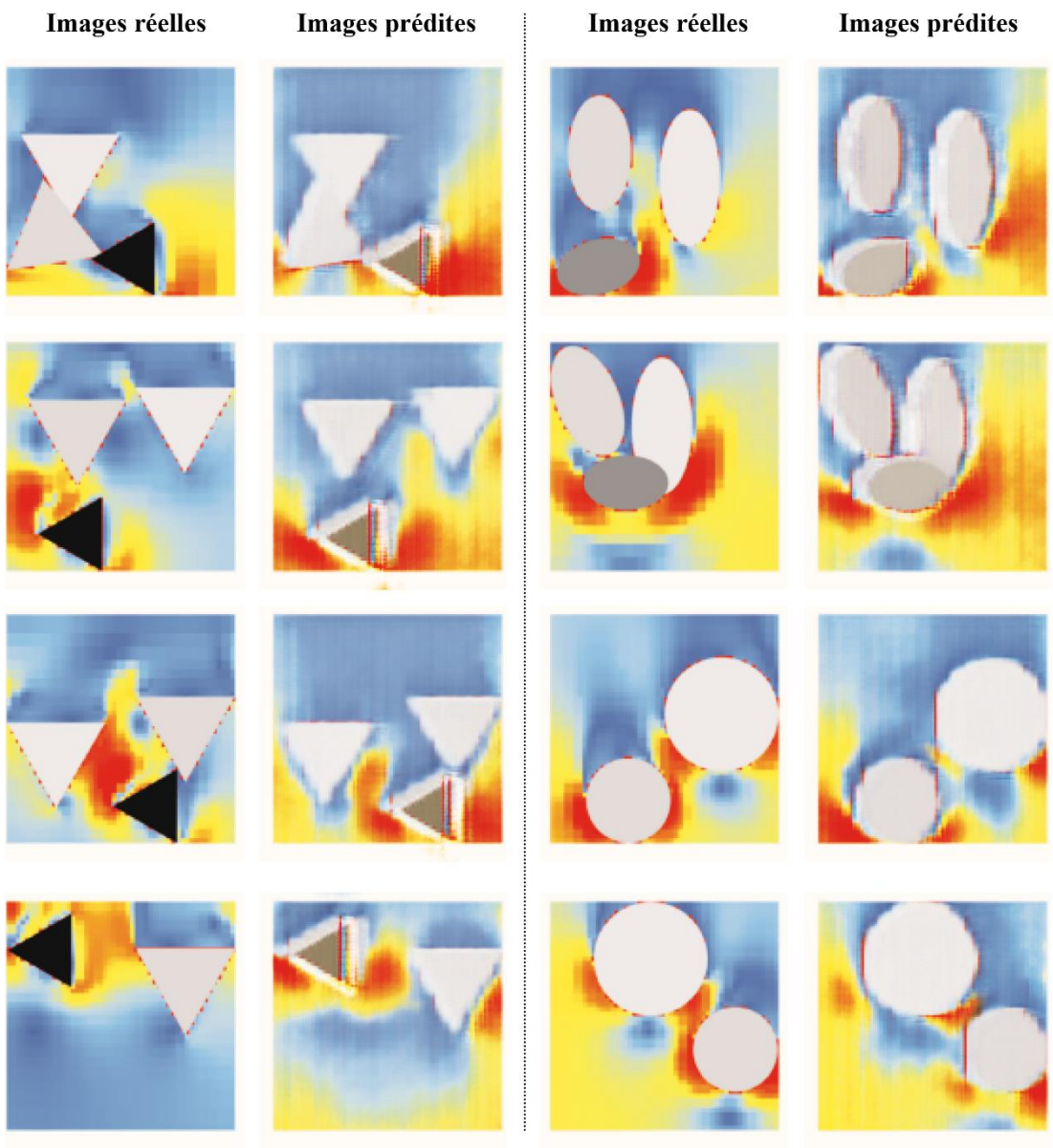


Figure 179 : Prédiction de l'écoulement des vents avec des formes différentes

Nous avons ensuite testé l'algorithme pour la prédiction de la vitesse des vents uniquement sur des morphologies différentes (le triangle extrudé, les ellipses extrudées et les cylindres). Contrairement à l'expérience précédente avec la radiation solaire, les images prédites sont très éloignées des simulations comme le montrent les résultats présentés en Figure 179. Ainsi, une base de données de qualité doit être adaptée au critère à prédire. Une certaine quantité et un certain niveau de diversité de formes pourront être suffisants pour créer un modèle de substitution pour un certain critère, et insuffisant pour un autre. L'étude de l'écoulement des vents semblent être un exemple particulièrement complexe, nécessitant une base de données très importante pour créer un modèle de substitution qui puisse être utilisé sur différents projets.

Expérimentation à l'échelle urbaine

Nous avons continué à interroger, mais cette fois-ci à l'échelle urbaine, la robustesse des modèles de substitution réalisés avec des GAN lorsqu'ils sont confrontés à des caractéristiques, notamment des morphologies sur lesquelles ils n'avaient pas été entrainés. A cette échelle, les temps de calcul des simulations sont décuplés, il est intéressant d'interroger la faisabilité de ce type de méthodes à cette échelle, d'autant qu'un modèle entraîné pour une mission d'urbanisme peut plus facilement être réutilisé sur un autre projet (à condition que celui-ci se situe dans la même zone géographique).

Pour cette dernière expérimentation sur les algorithmes de ML, un modèle paramétrique a été développé à l'aide des composants du plugin Grasshopper® DecodingSpaces Toolbox® que nous avions présentés dans le chapitre 1 (p103). Ainsi, nous avons pu générer, sur un terrain carré de 400 m de côté, des réseaux viaires irréguliers et 3 typologies d'îlots différentes : îlots fermés (en R+3), des tours, et des maisons individuelles. Nous avons de nouveau évalué la radiation solaire des espaces extérieurs. La surface évaluée est discrétisée en une grille de Xm de côté. L'image du domaine d'entrée contient le tracé du parcellaire et les masses en niveau de gris pour intégrer la hauteur des bâtiments. La base de données est constituée de 1 000 paires d'images de 2x 512 pixels de côté semblables à celle présentée dans la figure ci-dessous.

Figure 180 : Paire d'images issues de la base de données d'entraînement

L'algorithme a été entraîné sur 5 000 époques (4h23min51s), puis testé avec 4 types d'images différentes : (1) des images générées avec le même modèle paramétrique que celui ayant servi à la constitution de la base de données, (2) des images avec un modèle où les hauteurs ont été modifiées, (3) des images avec un modèle où les paramètres du réseau viaire ont été modifiés, (4) des images avec un modèle où la forme du quartier est variable.

Les résultats sont plus que convaincants avec le premier type d'image, comme on peut le voir pour les 3 tests présentés en Figure 181.

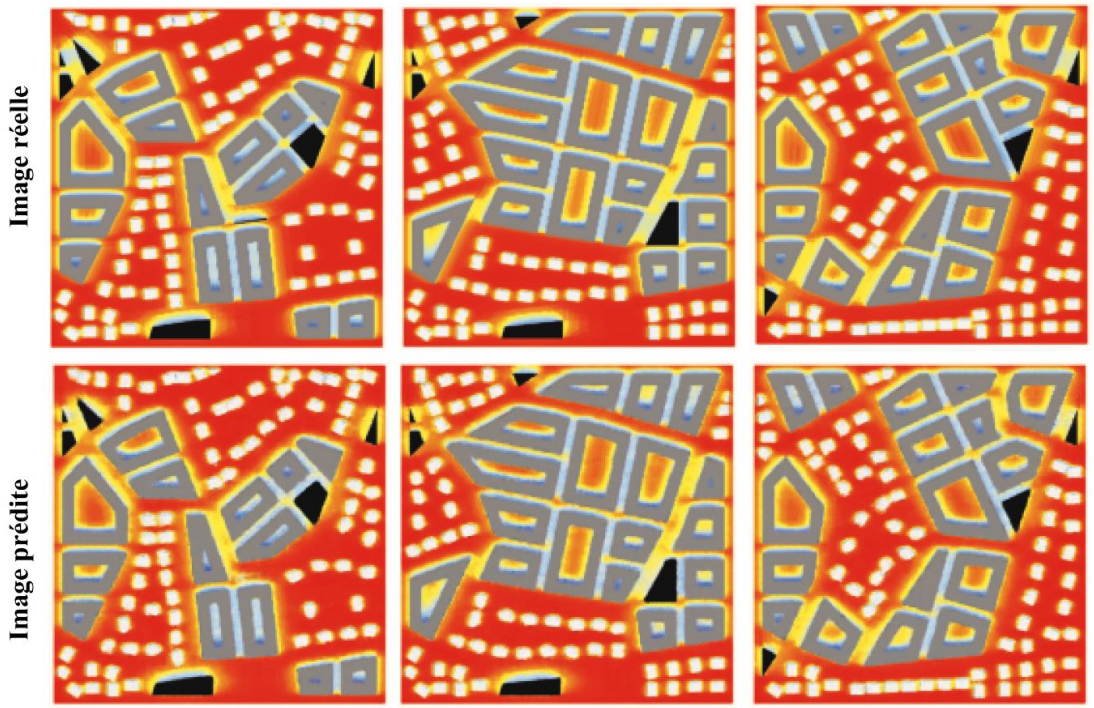


Figure 181 : Tests à partir d'images de la base de données originales

Comme on peut le voir sur les Figure 182 et Figure 183, contrairement à l'échelle de l'îlot, l'algorithme est beaucoup moins résilient avec les images de types (2) et (3).

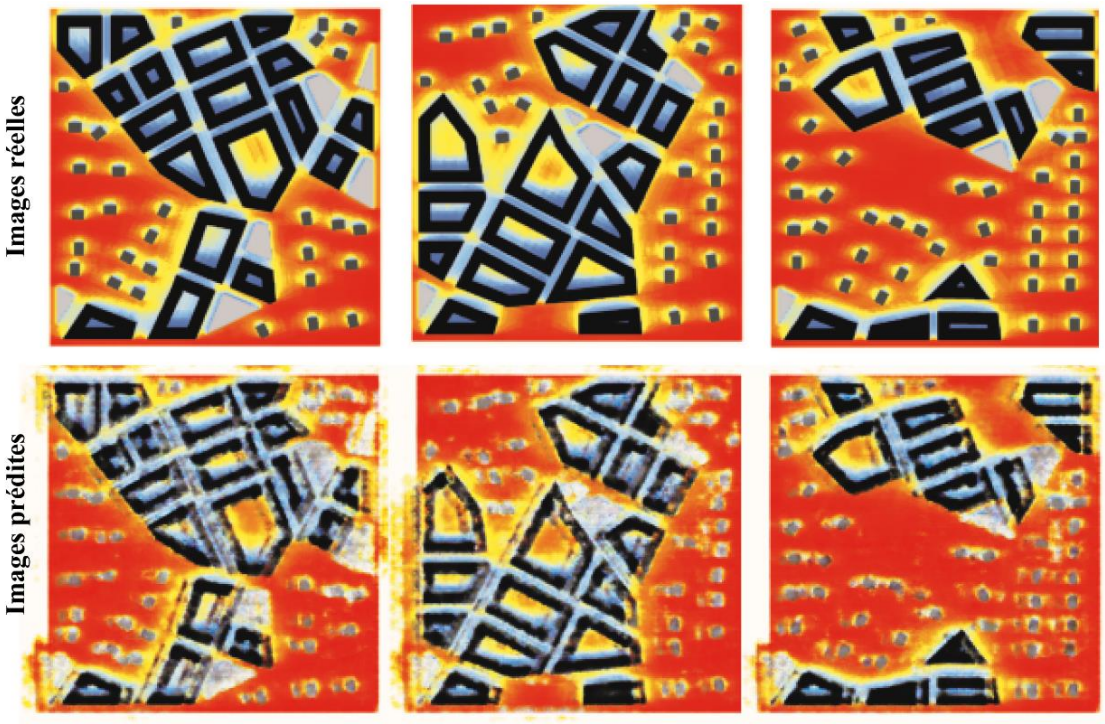


Figure 182 : Tests à partir des images où les hauteurs ont été modifiées

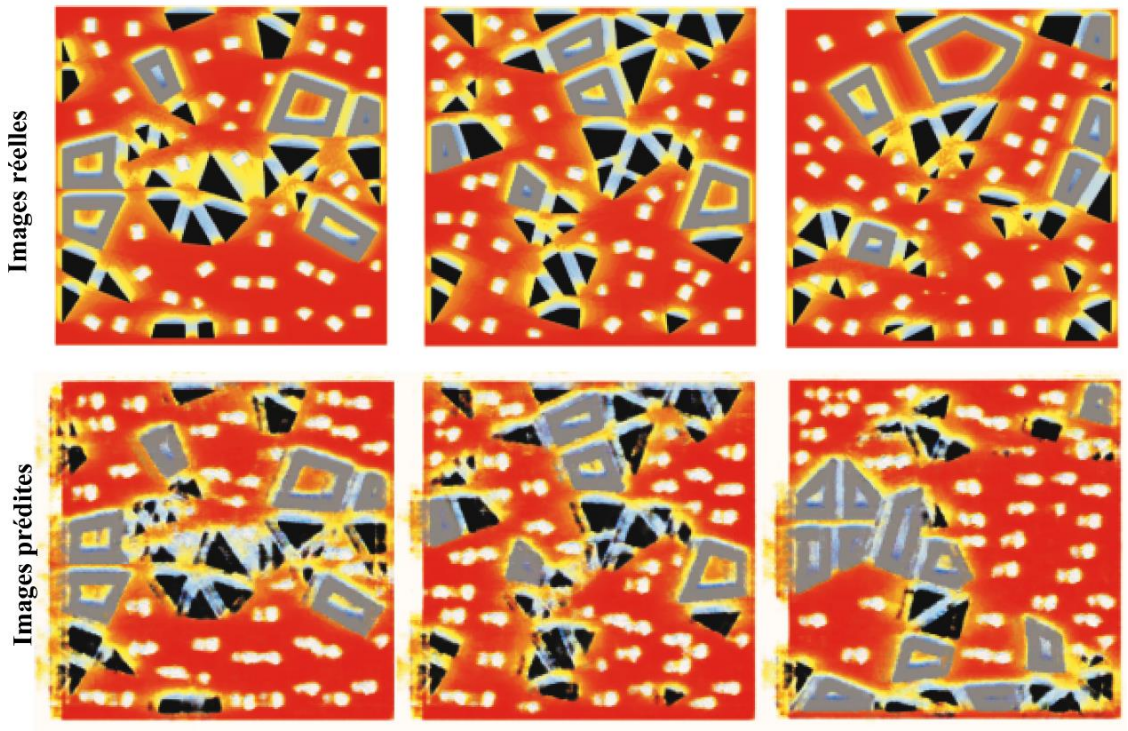


Figure 183 : Tests à partir d'images où les paramètres du réseau viaire ont été modifiés

Enfin, les images de type 4 ne sont pas exploitables (voir Figure 184). Pour qu'un modèle de substitution à l'échelle urbaine puisse être exploitable sur différents projets, il faudrait qu'il soit entraîné avec une grande base de données contenant une grande variété de typologies de bâtiments et de réseaux viaires, mais aussi de morphologies de périmètres d'intervention.

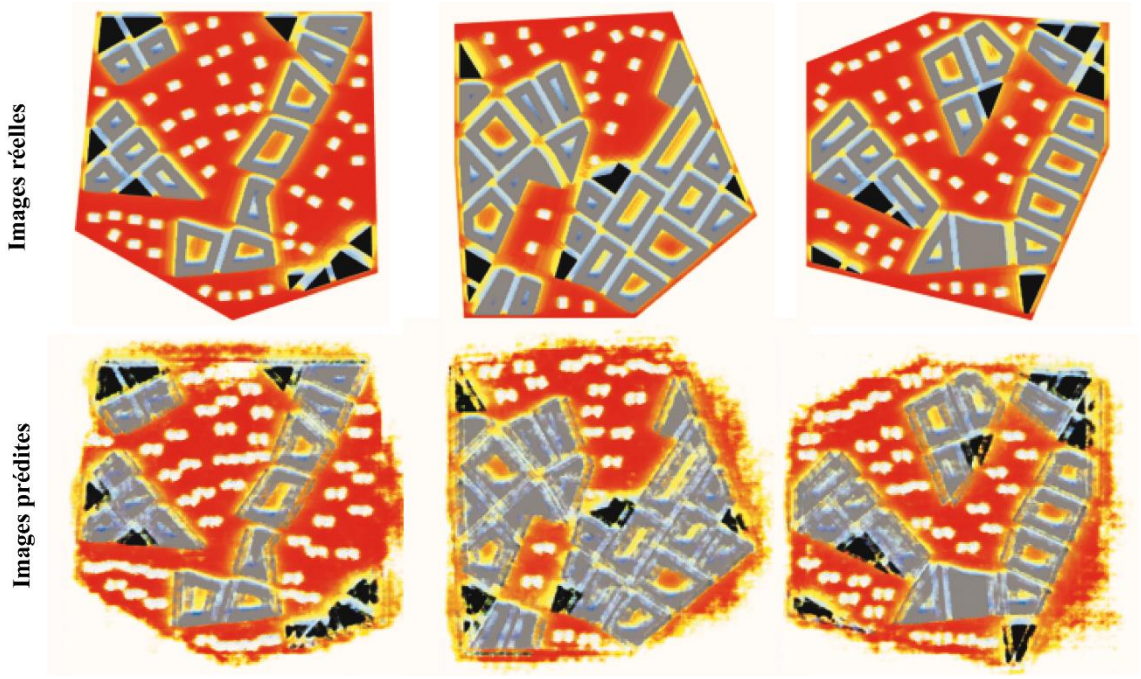


Figure 184 : Tests à partir d'images où la forme du quartier est variable

Finalement, les GAN apparaissent comme une solution prometteuse pour la prédiction du confort visuel et thermique pour faire de l'ENAPE, car ils permettent à la fois d'obtenir une visualisation graphique des évaluations et une prédiction précise (qui reste à mesurer numériquement). Cependant, pour que cette méthode puisse être utilisée par les architectes, elle doit être interopérable avec leurs outils de conception, notamment Grasshopper® qui compte déjà beaucoup d'outils essentiels pour faire du design génératif.

L'autre avantage des GAN est qu'ils peuvent travailler avec des données non-structurées, et ainsi s'adapter à différents modèles paramétriques contrairement aux régressions. Il semble que concevoir des modèles réplicables à différents projets situés dans une même zone géographique puisse être concevable avec des GAN, notamment pour l'étude de la morphologie d'un îlot, et peut-être aussi à l'échelle urbaine avec des échantillons de grandes tailles. Pour des problèmes d'architecture, notamment pour l'enveloppe, les modèles sont plus difficilement réplicables, il reste intéressant d'utiliser des GAN car les échantillons peuvent être plus petits qu'avec une régression linéaire ou une SVM, et semblent potentiellement plus précis.

Au terme de cette troisième partie du chapitre 2, il semble que les temps de calcul, décrits dans la littérature scientifique comme la limite technique la plus contraignante pour expliquer le manque de popularité des ENAPE dans la pratique chez les maîtres d'œuvre, puissent être en grande partie, et relativement facilement avec les outils existants, réduits grâce à l'utilisation d'algorithmes d'apprentissage automatique. Il semble que les simulations très couteuses en temps comme les consommations énergétiques puissent être facilement prédites en temps réel avec des régressions multivariables, et que de nouvelles techniques comme les GAN puissent être adaptées pour la prédiction des indicateurs de confort visuel et thermique. Si certains algorithmes sont d'ores et déjà accessibles et utilisables par des non-experts sur Grasshopper®, les GAN restent encore difficiles d'accès.

Au terme de ce deuxième chapitre il apparait que les limites que nous avons identifiées dans la pratique chez Architecture Studio diffèrent quelque peu de ce qui est décrit dans la littérature scientifique. Nous avons relevé que toute une catégorie de facteurs limitants a été oubliée par les chercheurs. Il s'agit des limites liées aux méthodes de travail des architectes. Le processus de conception n'est pas une démarche systématique, l'exploration de solutions traditionnelles sert à soulever de nouveaux problèmes. Les solutions optimisées par les processus d'exploration numérique sont fatalement destinées à être modifiées. Les architectes

doivent pouvoir apprendre des méthodes d'exploration et comprendre quelles sont les caractéristiques qui font qu'une solution est plus performante qu'une autre. Aussi, il reste nécessaire de former des architectes pour démocratiser l'usage de ce type d'outils innovants. Bien entendu, la formation de jeunes designer computationnels maitrisant la programmation visuelle avec des connaissances en conception bioclimatique et en optimisation est essentielle, mais aussi la formation à l'usage des outils de post-traitements qui servent à analyser les résultats pour les architectes plus expérimentés qui les encadrent.

Ensuite, si les temps de calcul peuvent être longs, il ne se sont pas rédhibitoires pour la mise en œuvre de ces approches, notamment pour les critères d'ensoleillement, de lumière et de vue, même si dans certains cas nous recommandons, en phase esquisse, plutôt l'usage de la méthode dite de la « *géométrie informée* » , ou de modèles de substitution. Ces derniers créés à l'aide d'algorithmes d'intelligence artificielle sont connus et utilisés pour remplacer la simulation et réduire les temps de calcul. Des techniques comme les régressions multivariables ou les réseaux neuronaux sont particulièrement efficaces pour la prédiction des consommations énergétiques. Ils le sont moins pour la prédiction de l'éclairage naturel ou des apports solaires. Ils nécessitent d'être entraînés avec des données structurées et doivent être développés pour des problèmes spécifiques. Des algorithmes plus récents comme les réseaux de neurones antagonistes permettent de prédire de façon précise des indicateurs de confort comme l'ALJ, la vitesse du vent, ou la radiation solaire à différentes échelles en générant des images. L'entrainement de ces algorithmes peut être long, demande des bases de données un peu plus conséquente mais permet de conserver la visualisation dans l'espace des résultats de simulation et n'a pas besoin de données structurées.

Finalement, la limite technique qui apparaît comme la plus bloquante est l'intégration des contraintes au processus d'optimisation. Les problèmes rencontrées dans la pratique sont souvent multicritères et doivent intégrer de multiples contraintes. Evaluer des solutions non constructibles est sans intérêt pour les architectes et représentent une perte de temps, sans compter que dans certains cas l'algorithme n'est pas en mesure de générer une seule solution faisable. Ainsi, dans la seconde partie de cette thèse, nous concentrons nos efforts sur les limites techniques que sont l'intégration des contraintes et les temps de calculs et nous cherchons à identifier dans la littérature scientifique comment remédier à ces problèmes.

Au terme de la première partie de cette thèse, nous savons désormais que l'enveloppe des bâtiments, leur morphologie, voir même l'agencement des espaces intérieurs peuvent avoir un fort impact sur la performance énergétique. L'exploration numérique est une méthode innovante, basée sur l'usage de techniques génératives, ayant fait l'objet de nombreuses publications et permettant d'aider les architectes à améliorer la performance des bâtiments qu'ils conçoivent. De nombreux types de problèmes peuvent être traités avec ces approches, mais leur modélisation est plus ou moins accessible aux non-experts de la programmation. Les problèmes de conception qui intègrent des préoccupations architecturales sont souvent plus complexes à modéliser que les problèmes d'ingénierie environnementale de la construction. De nombreux outils numériques sont aujourd'hui accessibles, souvent gratuitement, pour faire de l'exploration numérique, principalement en passant par le logiciel de modélisation Rhinocéros et son langage de programmation visuel Grasshopper. Les outils d'analyse notamment sont de plus en plus nombreux. Il est désormais possible de réaliser des analyses de l'ensoleillement, de la lumière, de la qualité de vue, du confort au vent, du confort thermique, de l'énergie, et du bilan carbone directement depuis le modeleur 3D.

L'expérimentation de ces approches dans un cadre professionnel nous a permis de comprendre qu'il n'existait pas une méthode d'exploration mais plusieurs, plus ou moins automatisées et intégrant plus ou moins de techniques génératives. La mise en place de ces approches sur des cas réels s'est révélée être une tâche complexe. Pour débuter, nous recommandons de traiter des problèmes de conception d'enveloppes qui sont plus accessibles. Les raisons historiquement décrites dans la littérature pour expliquer la difficile démocratisation de l'exploration numérique chez les architectes ont évolué. Les interfaces entre logiciels et les formations se sont beaucoup développées. Les raisons que nous évoquons aujourd'hui sont d'ordre structurelles, liées aux méthodes de travail et de management des architectes. Concernant l'argument majeur des temps de calcul, il existe aujourd'hui des solutions directement applicables grâce aux algorithmes de Machine Learning.

Pour conclure, la limite majeure de ces approches réside dans la difficulté à modéliser des problèmes complexes devant intégrer de nombreuses contraintes comme c'est le cas pour la plupart des problèmes que rencontrent le architectes. Ainsi, dans la seconde partie de cette thèse nous nous concentrons sur les solutions techniques qui permettraient de faciliter la modélisation de ce type de problèmes et la rendre accessible à des non-experts de la programmation informatique.

Partie 2

Les techniques génératives pour l’optimisation multicritère sous-contraintes appliquée à des problèmes de conception architecturale

Dans cette seconde partie de la thèse, nous cherchons à savoir comment gérer aux mieux les contraintes dans les problèmes d'optimisation rencontrés par les architectes. Pour cela, nous commencerons dans un premier chapitre par étudier plus en détails les algorithmes d'optimisation. Nous verrons qu'il en existe différents types qui permettent de traiter différents types de problèmes. Nous verrons quelle est la catégorie d'algorithme la plus adaptée pour traiter les problèmes de conception architecturale. Nous aborderons plus en détails les algorithmes multicritères, plus particulièrement les algorithmes évolutionnaires comme NSGAII que nous utiliserons à plusieurs reprises lors d'expérimentations et d'études comparatives. Nous listerons et testerons les différents outils d'optimisation actuellement à disposition des architectes.

Ensuite, nous aborderons la question des contraintes en commençant par une définition. Nous présenterons différentes méthodes d'intégration des contraintes compatibles avec les algorithmes évolutionnaires issues de la littérature scientifique des mathématiciens, et de celle des designers computationnels. Ces méthodes seront ensuite testées sur un problème issu de la pratique professionnelle et comparées. Parmi ces méthodes, une se distingue des autres, il s'agit de la méthode des fonctions de réparation. Le second chapitre sera donc dédié à cette méthode spécifique qui sera testée sur d'autres cas d'études.

Nous verrons alors que pour être utilisée, la méthode des fonctions de réparation nécessite souvent l'usage de techniques génératives spécifiques comme les automates cellulaires ou d'autres modèles à base d'agents. Ainsi, nous reviendrons sur le fonctionnement de ces différentes techniques et sur les outils existants qui facilitent leur mise en œuvre. Nous constaterons qu'il reste difficile d'élaborer des fonctions de réparation sans passer par la programmation informatique, ainsi nous proposerons notre propre librairie de programmes développés pour Grasshopper®. Pour finir, le plugin « Urchin » résultant de ce développement sera présenté en détail et testé sur différents types de problème traitants de la morphologie des bâtiments, de la conception urbaine, ou encore d'agencement des espaces intérieurs.

Chapitre 3 : Méthodes de gestion des contraintes pour les problèmes d'optimisation rencontrés en conception architecturale

Dans ce troisième chapitre, qui entre plus avant dans les préoccupations techniques, nous cherchons des solutions dans la littérature scientifique pour tenter de résoudre la principale limites techniques qui pourraient justifier un manque d'intérêt des architectes pour les ENAPE.

Pour cela, nous présentons dans une première partie les connaissances accumulées sur les algorithmes d'optimisation pour la conception architecturale, notamment les différentes classes d'algorithmes, leurs fonctionnements et les différents solveurs d'optimisation à la disposition des architectes. Dans un second temps, nous nous intéressons aux différentes méthodes d'intégration des contraintes par le truchement de ces algorithmes. Elles sont ensuite testées sur un cas d'étude issu de la pratique.

3.1. Les algorithmes d'optimisation en architecture

Dans cette première partie du chapitre 3, nous revenons sur les différents algorithmes d'optimisation utilisés en architecture. Dans un premier temps, afin de mieux cerner la nature des problèmes de conception, nous revenons sur les grandes définitions du champ de l'optimisation, avant de présenter les différentes méthodes de résolution qui catégorisent les algorithmes d'optimisation.

Dans un second temps, nous nous intéressons au cas particulier de l'optimisation multicritère, notamment la méthode dite de Pareto, avant d'exposer le détail du fonctionnement de l'algorithme bien connu NSGAII.

Dans un troisième temps, nous exposons les différents solveurs d'optimisation à la disposition des architectes pour les problèmes monocritères et multicritères. Nous discutons de la question du choix de l'algorithme et du paramétrage des algorithmes génétiques.

3.1.2. Les différents types d'algorithmes d'optimisation

Définition d'un problème d'optimisation

De manière générale, « optimiser » c'est, d'après le dictionnaire Larousse, chercher à rendre meilleur, « *donner à quelque chose ; à une machine, à une entreprise, etc., le rendement optimal en créant les conditions les plus favorables ou en tirant le meilleur parti possible* ». « Optimiser » revient à rechercher un « optimum », c'est-à-dire « *un état, degré de développement de quelque chose jugé le plus favorable au regard de circonstances données* » (définition du Larousse).

Pour être résolu mathématiquement, un problème d'optimisation doit d'abord faire l'objet d'une ***modélisation mathématique***. Celle-ci est réalisée en 3 étapes : (1) l'identification des ***variables de décision***, c'est-à-dire les paramètres du système qui peuvent influencer son fonctionnement, (2) la description de la méthode d'évaluation de l'état du système appelée « critère » ou « ***fonction objectif »***, (3) une traduction mathématique des circonstances, c'est-à-dire des ***contraintes*** qui définissent les valeurs que les variables peuvent prendre (Bierlaire, 2006). Optimiser revient alors à trouver l'état du système le plus favorable, ***l'optimum*** de la fonction objectif en maximisant ou minimisant cette fonction sur son ensemble (voir Figure 185).

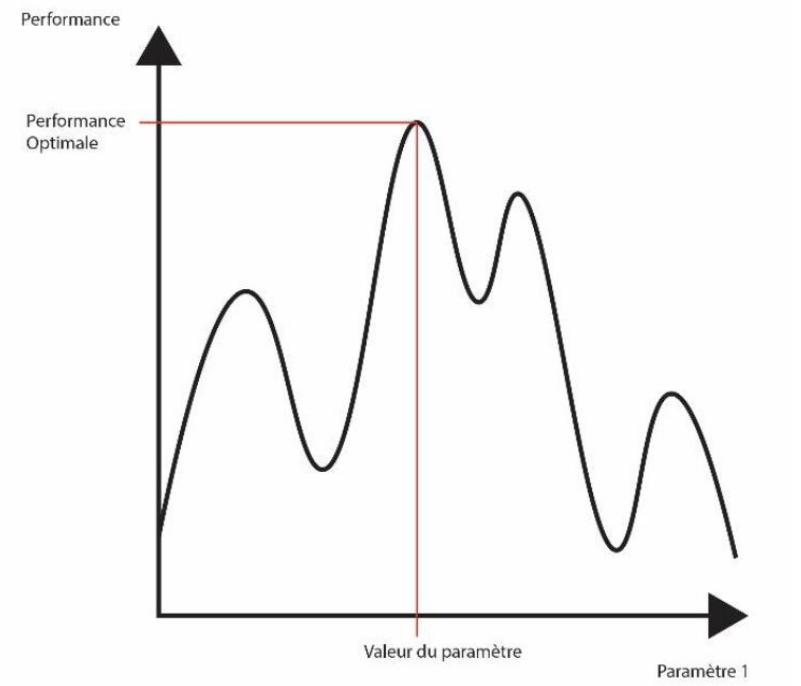


Figure 185 : Recherche d'un optimum (source: (Dissaux, 2018))

Suivant la forme de la fonction objectif et des contraintes, il est possible de distinguer des classes de problèmes engendrant des sous-disciplines de l'optimisation. Nous ne reviendrons pas sur l'ensemble de ces classes ici mais cherchons à comprendre à quelles classes appartiennent les problèmes d'architecture que nous rencontrons dans la pratique.

En architecture, nous rencontrons des problèmes d'optimisation en ***dimension finie*** car les inconnues sont des nombres entiers ou réels, on parle aussi d'optimisation paramétrique, (par opposition à l'optimisation en dimension infinie où les inconnues sont des fonctions et où l'on parle d'optimisation fonctionnelle). L'ensemble des valeurs que prennent nos paramètres sont toujours discontinues avec les outils de programmation visuel que nous utilisons (Grasshopper®, Dynamo®). Nous faisons donc de l'***optimisation combinatoire***, encore appelée optimisation discrète (par opposition à l'optimisation continue). Nous parlons aussi d'***optimisation globale*** (voir Figure 186), car la recherche du minimum ou du maximum se fait sur l'ensemble des données (par opposition à l'optimisation locale qui se contente de trouver des optimums locaux). Il s'agit toujours d'***optimisation statique***, car la fonction objectif et les contraintes ne changent pas au cours du temps (par opposition à l'optimisation dynamique).

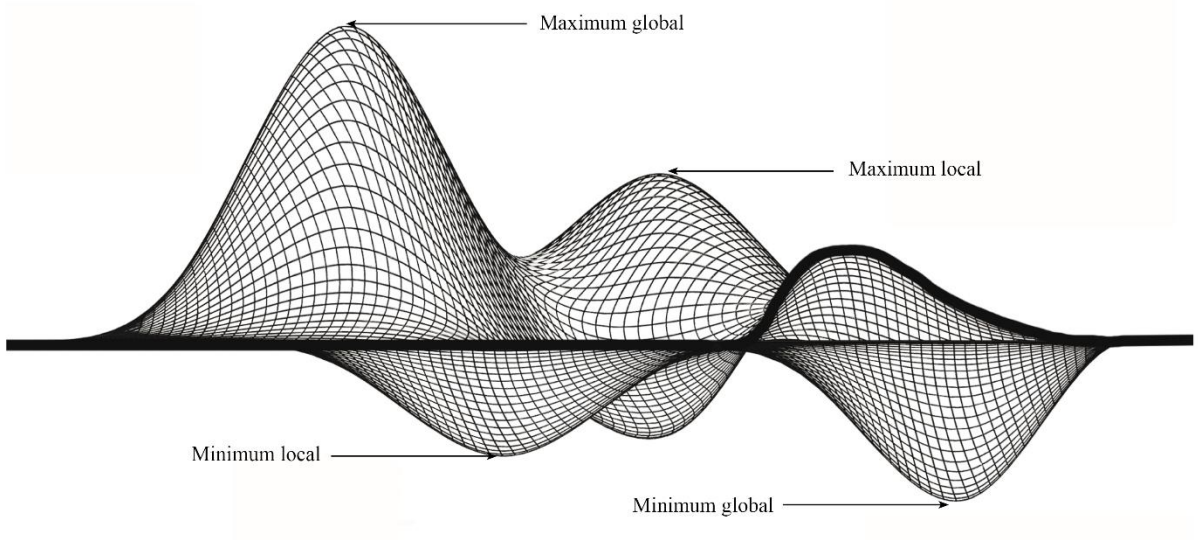


Figure 186 : Distinction entre optimums locaux et globaux

Les problèmes d'optimisation rencontrés en architecture sont des problèmes complexes où la combinatoire est souvent exponentielle. En conception performancielle, les fonctions objectifs sont calculées par des logiciels de simulations complexes intégrant des fonctions aléatoires ou des modèles de substitution basés sur les données produites par ces logiciels utilisés pour réduire les temps de calcul (voir définition donnée au chapitre 1 p.57). Il s'agit de problèmes d'***optimisation stochastique*** car les modèles ont un caractère aléatoire,

contrairement aux problèmes d'optimisation déterministe où les données du modèle sont connues avec précision. Les fonctions objectifs utilisées pour la modélisation des problèmes d'architecture ne sont jamais formulées mathématiquement, on parle de fonction objectif en ***boîte noire***, on ne peut donc pas les considérer comme linéaires ou dérivables. Les problèmes rencontrés dans la pratique sont ainsi des problèmes d'***optimisation non-linéaire*** avec des ***fonctions non différentiables***.

Aussi, les problèmes rencontrés en architecture sont presque toujours des problèmes d'***optimisation multicritère*** (par opposition à l'optimisation monocritère), car ils prennent en compte plusieurs fonctions objectifs. Il s'agit aussi souvent d'***optimisation avec contraintes,*** car les valeurs que peuvent prendre les variables sont restreintes (par opposition à l'optimisation sans contrainte). Enfin et surtout, les problèmes rencontrés en conception architecturale diffèrent pour chaque projet.

Les différentes méthodes de résolutions

Il existe de nombreuses familles d'algorithmes d'optimisation. Le choix de l'algorithme dépend de la nature du problème, de la qualité des solutions souhaitée et du temps disponible pour implémenter l'algorithme. Compte tenu de la nature des problèmes rencontrés en conception architecturale, seules quelques catégories d'algorithmes nous intéressent.

Les méthodes d'optimisation sans gradient

Ces méthodes d'optimisation fonctionnent uniquement avec les valeurs de la fonction objectif et des contraintes, elles ne nécessitent pas d'information sur les dérivées des fonctions objectifs, on parle aussi d'algorithmes « gradient free » (par opposition aux algorithmes «gradient based»).

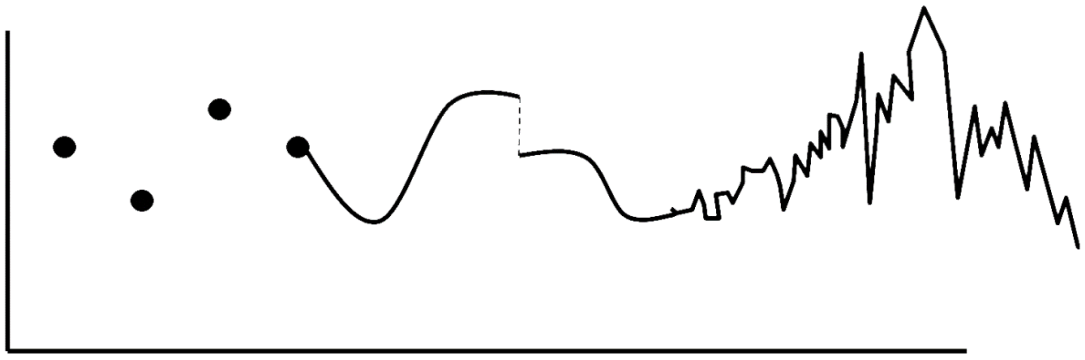


Figure 187 : Illustration d'une fonction discrète, discontinue et avec du bruit

Les méthodes d'optimisation sans gradient permettent de résoudre des problèmes pour lesquels la fonction représentant l'espace de solutions est discrète, discontinue ou avec beaucoup de bruit comme illustrée dans la Figure 187. Elles sont aussi utilisées lorsque la fonction objectif est une « boîte noire ». Parmi les méthodes sans gradient, on peut trouver des méthodes exactes, encore appelées méthodes déterministes ou des méthodes stochastiques.

Les méthodes déterministes

Les méthodes déterministes, aussi appelées méthodes « exactes » ne contiennent aucun caractère aléatoire. Une méthode d'optimisation déterministe parcourt toujours l'espace de solution de la même façon, tandis qu'un algorithme stochastique, parcourt l'espace de solutions de façon aléatoire. Ainsi, ces méthodes aboutissent toujours aux mêmes résultats si l'on utilise toujours les mêmes données d'entrée. La plupart de ces méthodes d'optimisation sont des méthodes avec gradient, mais il existe aussi des algorithmes d'optimisation déterministes sans gradients comme l'algorithme DIRECT (« DIviding RECTangles ») qui est une méthode d'optimisation déterministe globale ou l'algorithme Nelder-Mead qui est une méthode d'exploration déterministe locale (Cerf, 2022). Selon Duvigneau, il existe quatre catégories d'approches déterministes : les méthodes de type simplexe, les méthodes de recherche dans une famille de directions (par motifs), les méthodes à surface de réponse, les méthodes d'interpolation (Duvigneau, 2006).

Les métaheuristiques

Les métaheuristiques forment une catégorie de méthode stochastiques. Elles peuvent s'appliquer à des problèmes discrets ou continus, et être de nature globale ou locale (Cerf, 2018). Ces algorithmes permettent de résoudre des problèmes complexes à haut niveau d'abstraction pouvant être adaptés à de nombreux problèmes. « Méta » signifie « au-delà », ainsi les « métaheuristiques » sont à un niveau (d'abstraction) au-dessus des « heuristiques ». Une heuristique est une méthode empirique qui est spécifique à un problème particulier, ainsi, une métaheuristique est une méthode qui permet de résoudre différents problèmes. Une métaheuristique ne permet pas de trouver la meilleure solution dans un temps de calcul raisonnable mais une solution approchée. **Ainsi, on ne peut parler de solution optimale, mais de solution optimisée**.

Ces méthodes sont préférées lorsque les temps de calcul de la fonction objectif sont importants, que celle-ci n'est pas dérivable et qu'elle contient de nombreux optimum locaux, ou encore lorsque le problème est discret ou qu'il contient des contraintes fortes. Elles sont

donc particulièrement adaptées aux problèmes rencontrés par les architectes. Elles sont faciles à programmer et à manipuler, ce qui les rend accessibles aux non-mathématiciens.

Ces algorithmes sont souvent inspirés de phénomènes naturels, physiques ou biologiques. Les métaheuristiques sont facile à implémenter, elles peuvent être adaptées à des problèmes linéaires ou non linéaires, convexes ou non, continus ou discrets. Cependant, les experts encouragent à faire preuve de prudence quant à l'usage de ce type de méthode, qui doivent être utilisées seulement en dernier recours. Contrairement aux approches déterministes, les métaheuristiques peuvent être utilisées avec tous les types de fonctions objectifs (Brownlee, 2011), y compris les boîtes noires.

Algorithmes à base de modèle

Les algorithmes d'optimisation à base de modèle (MBO, pour model-based optimization) sont des méthodes stochastiques qui se distinguent des algorithmes classiques, dit à base d'instance (Zlochin et al., 2004). Ces méthodes s'appuient sur des modèles probabilistes pour générer des solutions. Ces algorithmes génèrent une population de solutions à partir d'un modèle (d'une distribution) qui reflète les propriétés structurelles de la fonction objectif. Le modèle est adapté à chaque itération, en fonction de l'évaluation de chaque population, et oriente ainsi l'exploration vers les meilleures solutions.

Cette technique est considérée comme très efficace pour résoudre des problèmes d'optimisation couteux en temps de calcul (en particulier lorsqu'ils impliquent de la simulation). Il existe deux catégories de MBO, la *« distribution-based models optimization* » (DBO) et les « *surrogate model based optimization*» (SBO) (Bartz-Beielstein, 2016). Lorsque le problème est complexe et qu'il n'est pas possible de clairement le représenter, les SBO sont généralement utilisés (voir Figure 188). Ces méthodes utilisent des algorithmes de *Machine Learning* pour prédire la performance des solutions. Ils créent des modèles de substitution qu'ils utilisent pour trouver plus facilement de meilleures solutions, et qu'ils actualisent au cours du processus d'optimisation en utilisant les résultats des véritables simulations (T. Wortmann, 2017). Ces méthodes peuvent être utilisées avec des algorithmes déterministes, aléatoires ou des métaheuristiques.

Parmi les méthodes utilisées pour créer ces modèles de substitution, on trouve les régressions polynomiales, les réseaux de neurones artificiels ou encore les fonctions de base radiale (RBF). Bartz-Beielstein T. distingue 4 types de modèle de susbstitution : « *single*

surrogate based », « *multi-fidelity based* » lorsqu'on utilise plusieurs modèles d'un même système réel mais avec un niveau de détail différent, « *evolutionary surrogate based* » qui sont des algorithmes évolutionnaires intégrant un modèle de substitution pour réduire le temps d'évaluation, « *ensemble surrogate based* » lorsqu'on combine différents modèles (Bartz-Beielstein, 2016). Contrairement aux métaheuristiques, le nombre de variables est limité avec un algorithme SBO. Les méthodes RBF sont particulièrement efficaces pour les problèmes d'ingénierie impliquant de longs calculs de simulation. L'algorithme RBF fait une approximation de la relation entre les variables et le critère à optimiser à partir d'un petit nombre de simulations, ce qui permet une meilleure compréhension du problème et facilite le processus d'exploration de l'espace de solutions par le concepteur (Wortmann et al., 2015).

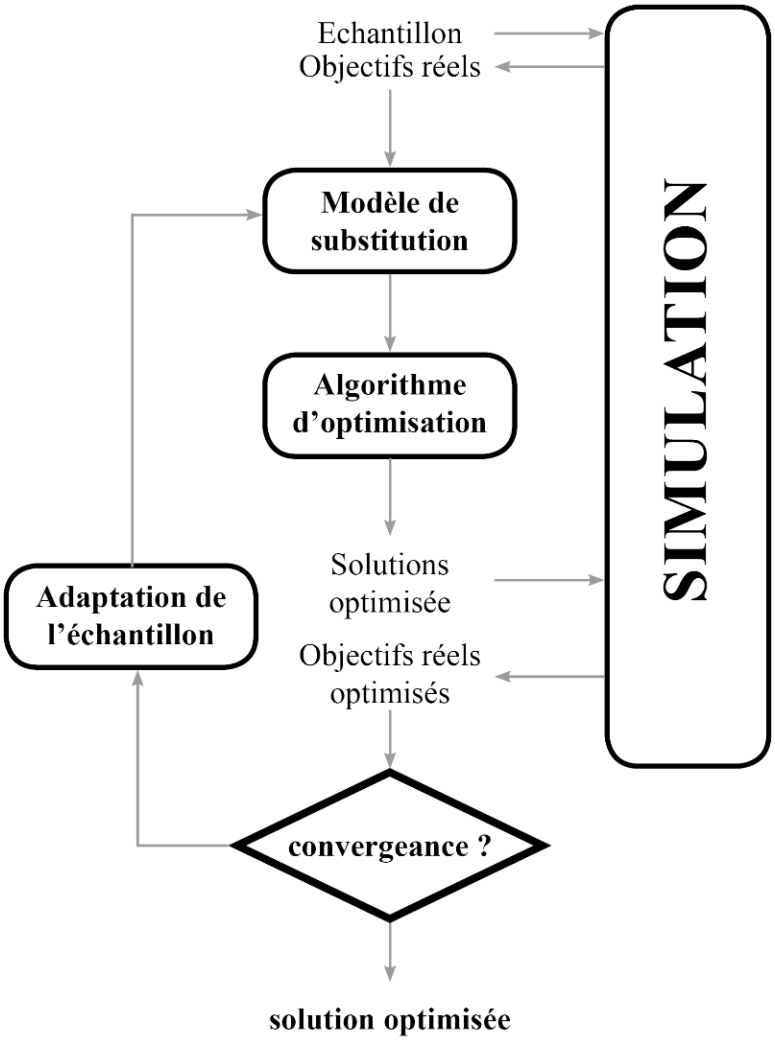


Figure 188 : Architecture d'une méthode d'optimisation à base de modèle de substitution (source : (Bevan et al., 2017))

Les algorithmes d'optimisation en architecture

Plusieurs chercheurs ont déjà listé les algorithmes d'optimisation les plus utilisés pour la conception des bâtiments (Evins, 2013; Nguyen et al., 2014). Leurs résultats, traduits de l'anglais, sont présentés en Figure 189. Les méthodes les plus employées sont des métaheuristiques comme les algorithmes génétiques, notamment NSGAII (Deb et al., 2002), l'optimisation par essaims particulaires, la méthode du recuit simulé et certaines méthodes déterministes comme l'algorithme Nelder-Mead, ou Hooke-Jeeves.

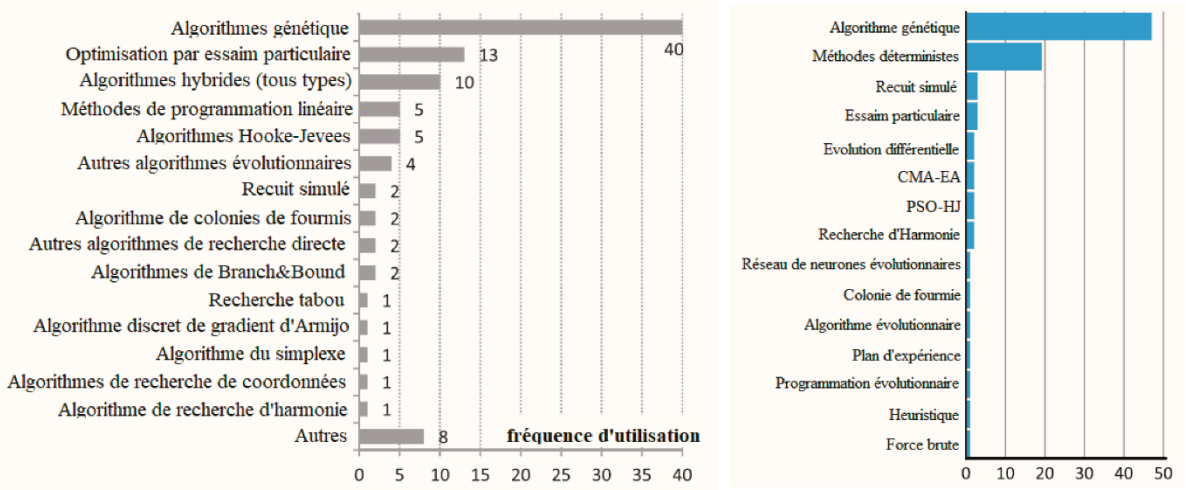


Figure 189 : Les algorithmes d'optimisation les plus utilisés pour la conception des bâtiments (source : à gauche (Nguyen et al., 2014), à droite (Evins, 2013))

<u>Les algorithmes génétiques</u>

Les algorithmes génétiques (Mirjalili, 2019) sont connus pour être facilement accessibles à des non-mathématiciens, c'est pourquoi ils sont si populaires. Pour utiliser un algorithme génétique il faut connaître quelques termes de biologie génétique. Ces algorithmes s'inspirent de la théorie de l'évolution de Darwin (Darwin, 2004), c'est-à-dire de l'idée que les espèces évoluent et survivent grâce à la sélection naturelle. Pour qu'un ensemble d'individus d'une même espèce, appelé « population », survive, des couples sont formés et se reproduisent. Les gènes contenus dans les chromosomes de chaque parent se croisent. On parle alors du « *croisement génétique* » - et quelquefois mutent, il s'agit de la « *mutation génétique* » donnant naissance à de nouveaux individus : les enfants, soit une nouvelle génération. Les gènes d'un individu, son « *génotype* », lui donnent son apparence et ses caractéristiques propres, il s'agit de son « *phénotype* ».

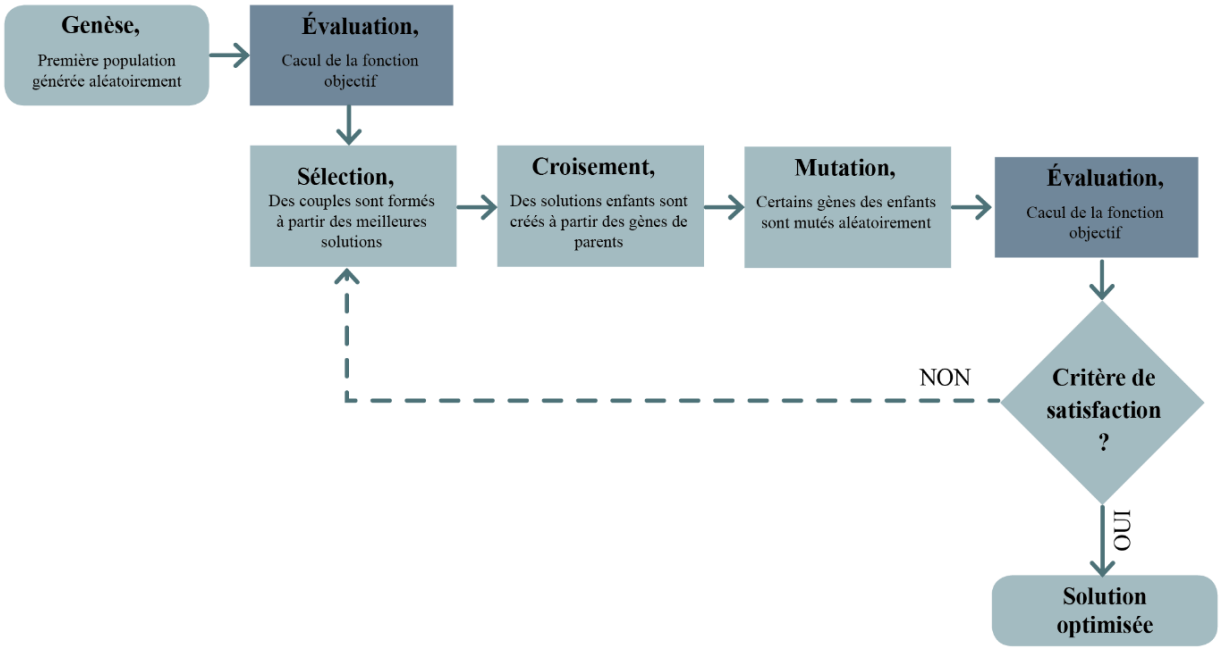


Figure 190 : Fonctionnement général d'un algorithme génétique

Lorsqu'on applique cette logique à un problème de conception, les variables (qui peuvent être la hauteur d'un vitrage, la largeur d'une lame, le nombre d'étages d'un bâtiment ou encore les paramètres physiques des matériaux) constituent le génotype d'une solution. Aussi, les scores obtenus après évaluation de la solution (qui peuvent être le niveau de facteur de lumière du jour, la quantité d'apports solaires ou le coût) constituent le phénotype. Il existe aujourd'hui de nombreux algorithmes génétiques à disposition décrits dans la littérature scientifique mais leur fonctionnement général suit toujours le même schéma logique présenté en Figure 190. Une première population peut être générée aléatoirement (ou définie par l'utilisateur selon le type d'algorithme et de solveur utilisé), puis chaque individu parent est évalué à l'aide de la ou des fonctions objectifs. Ensuite, des paires sont formées. Une méthode de croisement (aussi appelée enjambement ou recombinaison) est alors appliquée (voir Figure 191) avant que certains gènes de certains individus enfants ne soient mutés. L'ensemble de la population enfant est évaluée et devient la nouvelle population de parent. La boucle continue ainsi jusqu'à satisfaction d'un critère choisi par l'utilisateur (un nombre de générations, un temps de calcul limite, une quantité de génération stagnante, etc.).

Il existe différentes méthodes de sélection des individus (par rang, tournoi, sélection uniforme), et différentes méthodes de croisement (uniforme, semi-uniforme, ordonnée, multi-points…) et de mutation (uniforme, non-uniforme, Gaussien, supervisée…). C'est la combinaison de ces trois algorithmes qui permet de définir un algorithme génétique, ce qui explique leur grand nombre.

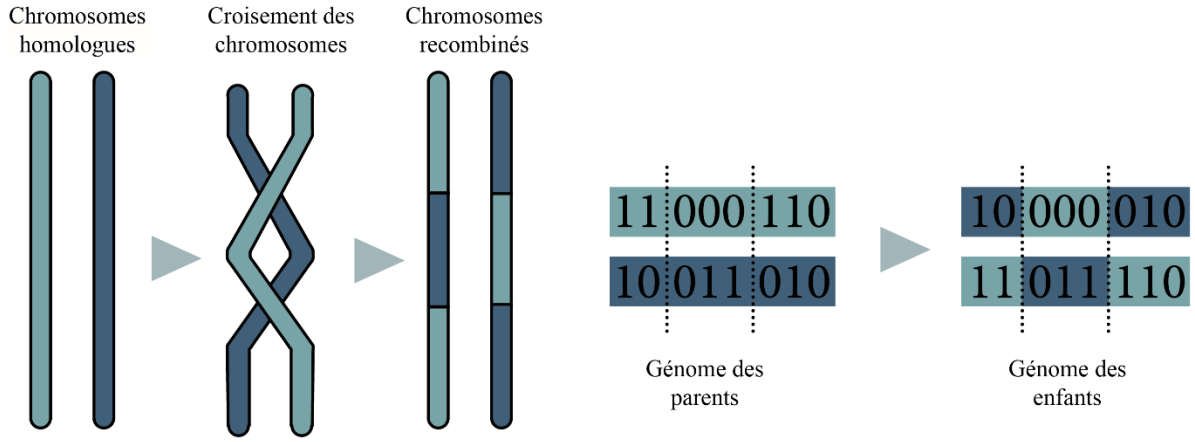


Figure 191 : Illustration du croisement en biologie et en programmation génétique

Les algorithmes par essaims particulaires

Les méthodes d'optimisation par essaims particulaires (PSO pour *particle swarm optimisation* en anglais) sont relativement récentes. Il s'agit d'une invention de Eberhart et Kennedy présentée en 1995 (Kennedy & Eberhart, 1995). Elle trouve son inspiration dans les programmes de type MBA cherchant à imiter le déplacement des groupes d'oiseaux et les bancs de poissons. Ces méthodes cherchent à reproduire des comportements sociaux (Clerc, 2005). Comme les algorithmes génétiques, elles utilisent des populations d'individus (de solutions), soit un ensemble de particules (d'agents) appelé « *essaim* ». Ces particules sont en mouvement dans l'espace de solution et coopèrent pour permettre à l'algorithme de converger.

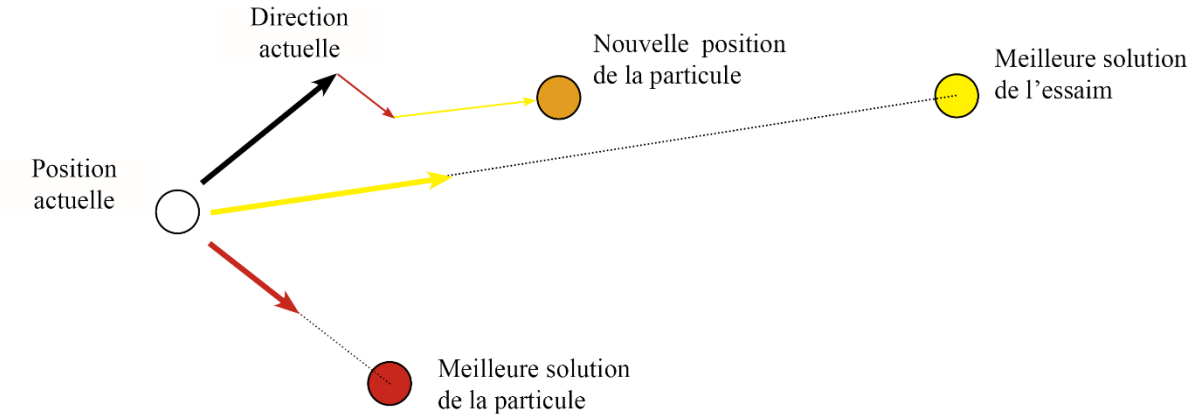


Figure 192 : Le mouvement des particules en optimisation par essaims particulaires

A chaque itération, le mouvement de chaque particule, composé d'une direction et d'une vélocité (vitesse) change. Chacune des particules de l'essaim dispose d'une mémoire, celle de leur meilleure performance et peut communiquer avec les particules voisines. Comme illustré

en Figure 192, la trajectoire d'une particule est guidée à chaque itération par sa dernière trajectoire, par le souvenir de sa meilleure performance et par le comportement de ses voisins eux-mêmes guidés par leurs meilleurs performances. Ainsi, l'essaim finit par converger vers une région où il y a beaucoup de bonnes solutions, et peut-être la meilleure solution (Cichocka, 2017).

Méthode du recuit simulé

La méthode du recuit simulé proposée en 1953 par Metropolis (Metropolis et al., 1953) est d'abord un processus métallurgique appelé le « recuit » durant lequel le métal est alternativement chauffé jusqu'à devenir liquide puis refroidi très lentement jusqu'à devenir solide et atteindre un état d'équilibre où l'énergie est minimale pour obtenir une structure parfaite proche d'un Crystal. Le système est alors « *gelé* ».

L'algorithme d'optimisation du même nom, qui date du début des années 80, s'inspire de cette méthode. Il cherche à simuler le comportement de la matière et à atteindre cet état d'équilibre thermodynamique. La fonction objectif représente l'énergie du système et un paramètre fictif est ajouté, la température.

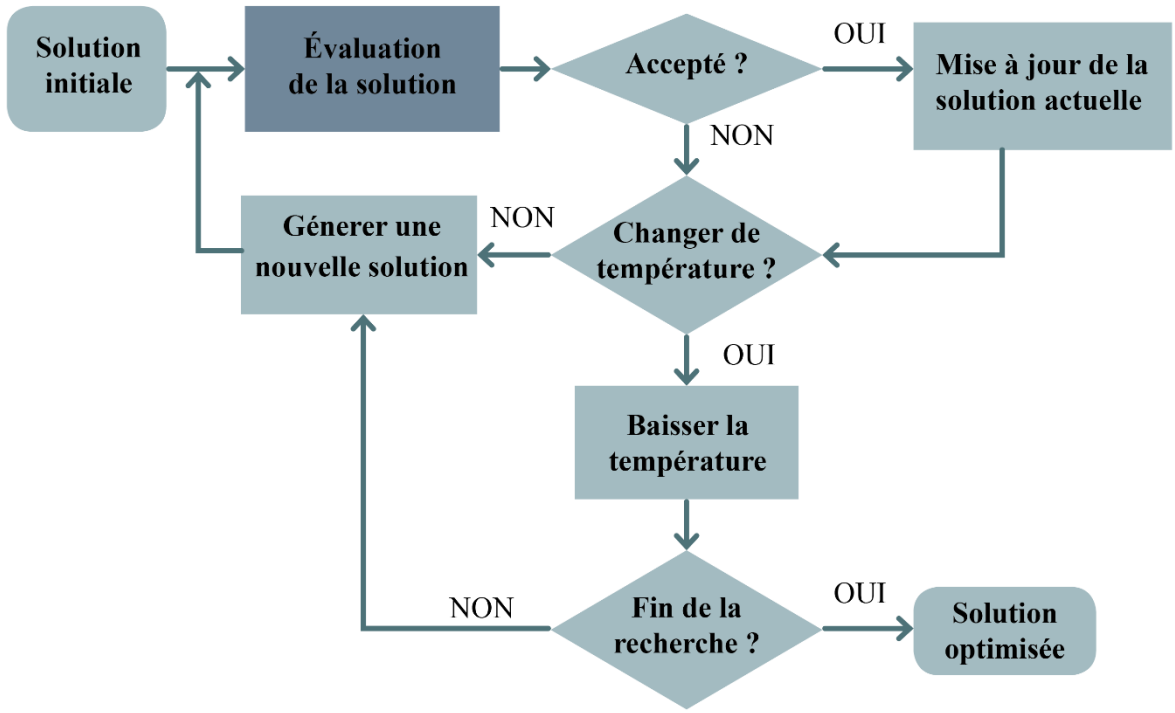


Figure 193 : Fonctionnement de l'algorithme du recuit simulé (source: (Pham & Karaboga, 2012))

Le fonctionnement de l'algorithme est présenté en Figure 193 (Pham & Karaboga, 2012). Une température initiale, souvent élevée, et une solution initiale sont d'abord choisies. Une seconde solution est générée au hasard dans le voisinage de la première avant d'être évaluée

puis comparée à cette dernière. L'opération est répétée jusqu'à atteindre un équilibre statistique. La température est ensuite abaissée et l'opération de nouveau répétée. L'algorithme s'arrête lorsque le système est gelé.

Une température élevée accepte toutes les solutions, ce qui équivaut à une exploration aléatoire. Une température moyenne laisse passer des solutions dégradées pour éviter les minimum locaux. Enfin, une température nulle équivaut à une recherche locale.

Au départ, certaines mauvaises solutions sont conservées afin d'éviter les optimums locaux, puis, petit à petit, uniquement les solutions meilleures que la précédente sont acceptées afin de tendre vers un optimum.

Méthode de Nelder-Mead

La méthode de Nelder-Mead créée en 1965 par John Nelder et Roger Mead (Nelder & Mead, 1965) est une méthode d'optimisation locale autrement nommée méthode du simplexe, ou *« downhill simplex method »*. Elle est basée sur l'utilisation du simplexe qui est, pour simplifier, une généralisation du triangle à un nombre de dimension quelconque.

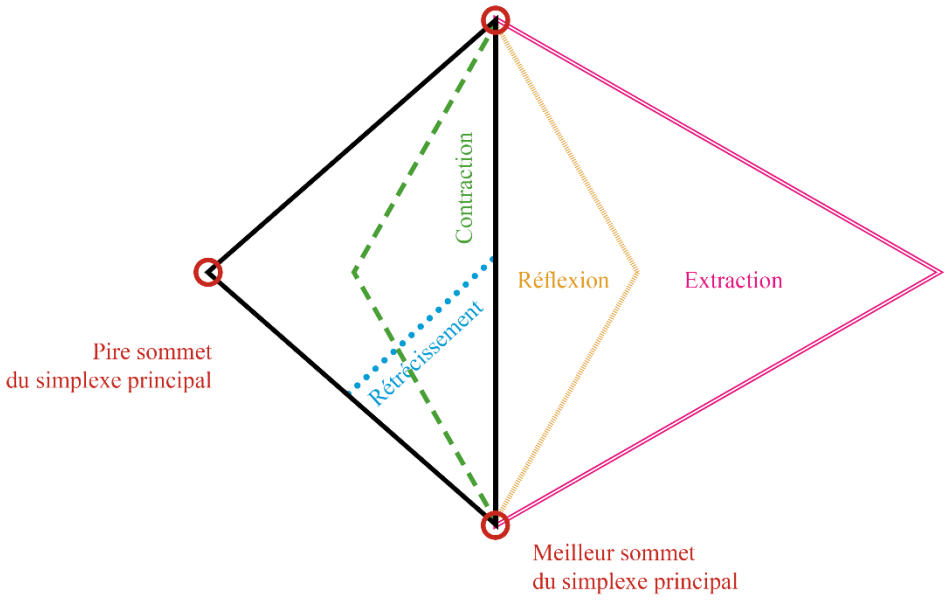


Figure 194 : Les opérations basiques de la méthode du simplexe (source : (Jalaeian-F, 2012))

Le simplexe effectue des transformations (réflexion, extension, contraction et rétrécissement) dans l'espace de solutions jusqu'à converger éventuellement vers une solution optimale. Les différentes étapes de l'algorithme sont précisées en Figure 195. Il s'agit d'une stratégie d'optimisation locale, mais elle peut être couplée à d'autres méthodes comme la méthode du recuit simulé pour réaliser une exploration globale.

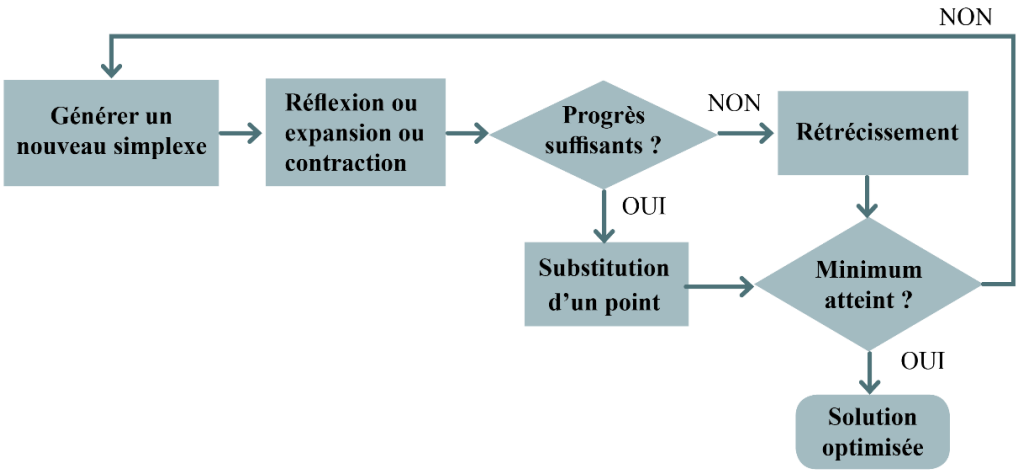


Figure 195 : Algorithme de Nelder-Mead (source:(Barati, 2014))

Méthode de Hooke Jeeves

Créée en 1961 par Hooke et Jeeves, cet algorithme est une méthode d'optimisation par motif, locale, et déterministe qui ne nécessite pas de gradient. Ce type d'algorithme se déplace dans l'ensemble de solution à l'aide d'un motif (un ensemble de directions permettant d'identifier d'autres solutions voisines) placé sur la solution courante. Si celui-ci détecte une amélioration (une solution voisine plus performante), le motif se déplace sur elle. L'algorithme de Hooke Jeeves comporte deux types de mouvement : le mouvement exploratoire et celui du motif, tous deux illustrés en Figure 196. A chaque mouvement exploratoire, la fonction objectif est calculée. Les directions sont parcourues les unes après les autres (Kramer et al., 2011).

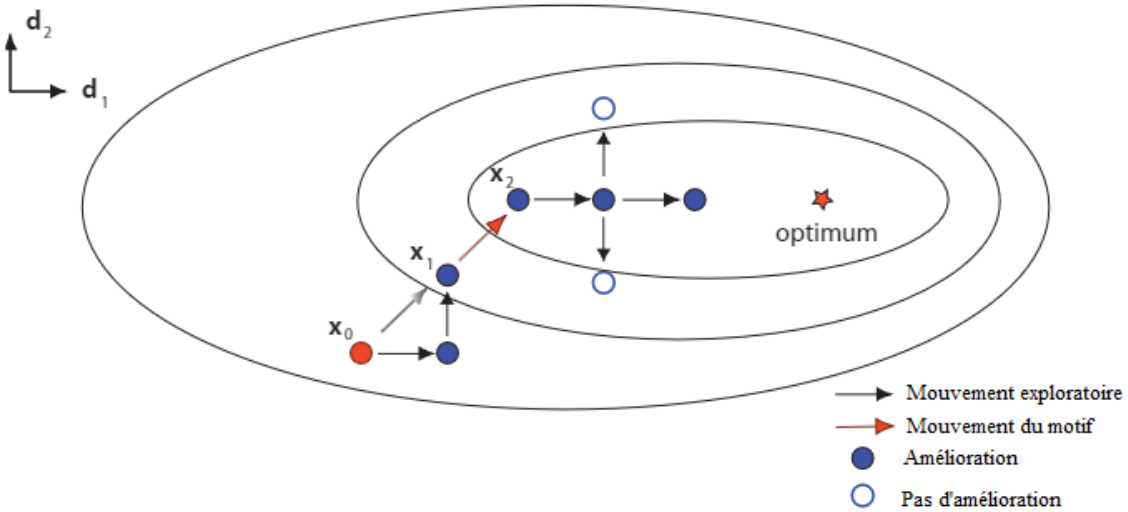


Figure 196 : Méthode d'exploration de l'algorithme de Hooke Jeeves (source:(Kramer et al., 2011)

3.1.2. L'optimisation multicritère

Les différentes méthodes d'optimisation multicritère

Nous l'avons vu dans les chapitres précédents ; que ce soit dans les exemples issus de la littérature scientifique ou dans les cas rencontrés dans la pratique en agence, les problèmes de conception rencontrés par les architectes sont presque toujours multicritères. C'est-à-dire que l'optimisation doit être réalisée sur plusieurs fonctions objectifs, on parle aussi de problèmes multiobjectifs. Pour résoudre ce type de problèmes d'optimisation, plusieurs types d'approches existes (Gunantara, 2018):

(1) La **méthode lexicographique** qui consiste à lancer un processus d'optimisation monocritère pour chacun des objectifs du problème en commençant par le critère le plus important. Lors de la seconde optimisation, des contraintes issues de la première exploration sont intégrées au processus, et ainsi de suite jusqu'au dernier objectif.

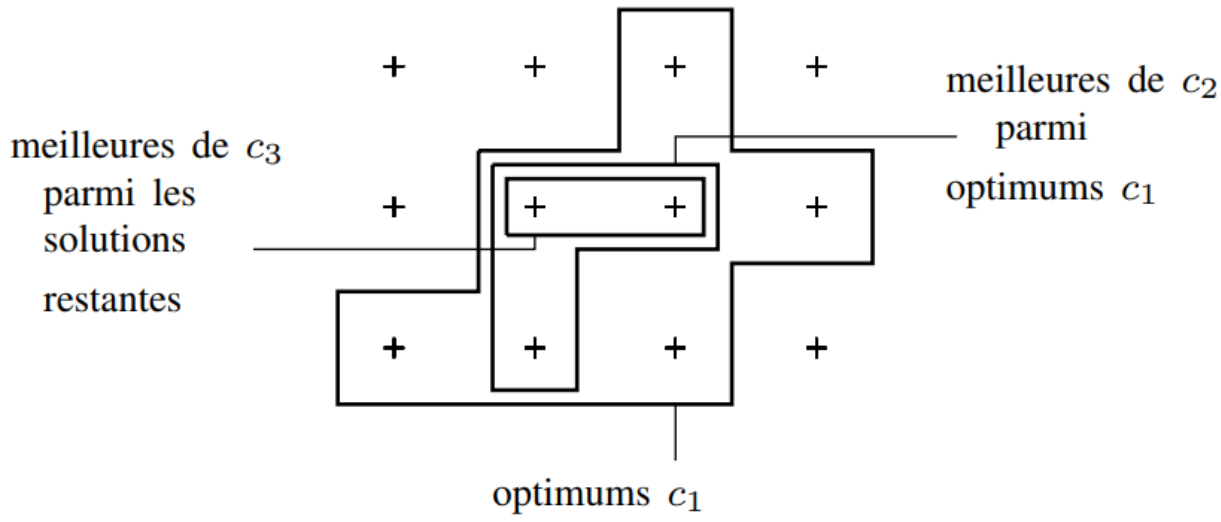


Figure 197 : optimisation lexicographique sur 3 critères (source:(Le Berre et al., 2012))

(2) La **scalarisation** qui consiste à réduire les fonctions objectifs en une unique fonction et ainsi résoudre le problème avec un algorithme d'optimisation monoobjectif,

(3) La **méthode de Pareto**, qui permet de trouver un ensemble de solutions optimisées à l'aide d'algorithmes d'optimisation multicritère. Elle permet de faire un arbitrage entre plusieurs critères. Les solutions situées sur le front de Pareto correspondent aux meilleurs combinaisons possibles trouvées par l'algorithme.

Dans la pratique, la (1) serait trop chronophage, certaines méthodes de la catégories (2) sont envisageables sur certains problèmes, et la méthode (3) est, selon nous, la plus adaptée aux problèmes de conception rencontrés par les architectes.

La scalarisation

La somme pondérée

La méthode de la somme pondérée est la méthode de scalarisation la plus connue, et la plus facile à implémenter. Cette méthode nécessite de connaître à l'avance la hiérarchie entre les objectifs. L'approche consiste à faire une somme pondérée des différentes fonctions objectifs. La pondération permet de hiérarchiser les critères d'évaluation, elle s'exprime à l'aide de « poids » et sert à différencier l'importance d'un objectif par rapport à un autre. Pour une quantité n d'objectifs, la fonction objectif F prend alors la forme suivante :

$$F(x) = w_1.f_1(x)+ w_2 .f_2(x)+\ldots+ w_n.f_n(x),$$

Où $f_i(x)$ sont les fonctions objectifs respectives de chaque critères i et w_i sont les coefficients de pondération associés. L'avantage de cette méthode est qu'elle est de même complexité et demande le même temps de calcul qu'une version similaire du problème à un objectif (Ehrgott, 2005).

Selon (Gunantara, 2018) il existe plusieurs méthodes pour définir les coefficients de pondération, notamment (1) la méthode des poids égaux, (2) la méthode de classement des poids des centroïdes ou encore (3) la méthode de la somme des rangs .

Dans la pratique, il est souvent complexe de correctement mettre en œuvre cette méthode. En effet, avant de pouvoir pondérer les fonctions objectifs, elles doivent d'abord être normalisées. Selon le problème, et surtout le critère évalué, les plages de valeurs que peuvent prendre certains objectifs sont très variables. Une fonction dont les valeurs oscillent entre 0 et 1, ne peut être sommées avec une fonction dont les valeurs oscillent entre 10 000 et 100 000 ; il est nécessaire de rendre ces valeurs comparables. Cela implique pour chaque objectif de connaître les valeurs minimum et maximum que peuvent prendre la fonction. Or, ces informations ne sont pas toujours connues sans un échantillon préalablement calculé de l'ensemble de solutions.

Dans certains cas, cette approche peut être pertinente, notamment lorsque les objectifs peuvent être évalués avec la même unité, par exemple pour étudier le compromis entre les apports solaires en hiver et en été (kWh/m^2), ou lorsqu'on exprime des critères de confort en surface, comme la surface d'un bâtiment atteignant un certain niveau de FLJ, ou un certain degré de visibilité vers l'extérieur, ou encore un certain seuil de l'indicateur PMV. Dans ces cas, il est possible d'agréger des fonctions qui étudient la lumière, la vue et le confort thermique.

Les autres méthodes de scalarisation

Il existe d'autres méthodes de scalarisation (Ehrgott, 2005; Miettinen & Mäkelä, 2002). Elles sont plus complexes à formuler, et beaucoup plus difficiles à implémenter avec un solveur d'optimisation boîte noire comme ceux à la disposition des architectes.

La **méthode de Ɛ-contrainte** consiste à choisir une seule fonction objectif et de formuler les autres comme des contraintes. Cette méthode nécessite d'avoir la main sur l'algorithme d'optimisation, ce qui n'est pas le cas avec les solveurs boîtes noires qu'utilisent les architectes.

La **méthode de Benson** est d'une méthode hybride entre la méthode de la somme pondérée et la méthode Ɛ-contrainte. Elle est utilisée pour vérifier si une solution faisable est efficace ou non.

La **méthode de Tchebychev** pondérée augmentée est une méthode peu connue. Elle a pour objectif de minimiser le pire des objectifs pondérés.

Les **méthodes par points de références** incluent différentes méthodes de scalarisation. Elles utilisent toutes une solution faisable ou infaisable de l'espace de solution. Il s'agit soit d'un vecteur de valeurs de critères souhaitables (niveau d'aspiration) ou un vecteur représentant des résultats disponibles ou à éviter (niveau de réservation).

La méthode de Pareto

Dans la littérature, il est parfois question de méthodes « *a posteriori* » (Emmerich & Deutz, 2018). Ces méthodes ne nécessitent pas de hiérarchisation préalable des fonctions objectives, elles visent à trouver toutes les solutions optimales situées sur le « *front de Pareto* » (voir Figure 198). Il existe trois classes de techniques qui permettent de faire cela : la

programmation mathématique, les algorithmes évolutionnaires et les méthodes de *Deep Learning*. Nous nous concentrerons ici sur la technique à base d'algorithmes évolutionnaires qui est l'approche la plus populaire en conception générative et performative.

Le front de Pareto

Le concept de front de Pareto est essentiel en optimisation multicritère car il permet de comparer deux fonctions objectifs sans avoir à normaliser les résultats des solutions et sans avoir à hiérarchiser les fonctions. Cette méthode permet d'ordonner l'espace de solutions à l'aide du concept de dominance.

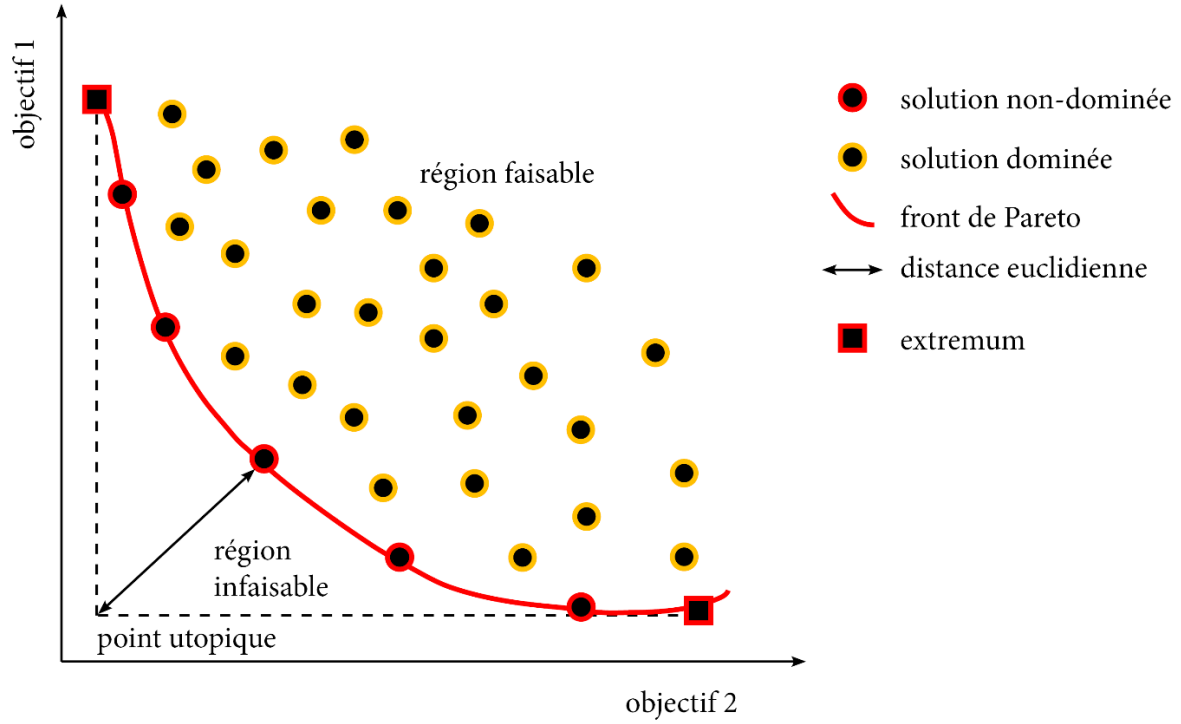


Figure 198 : Concept du front de Pareto

Comme le montre la Figure 198, dans l'espace de solutions, certaines solutions sont « *dominées* » et d'autres « *non dominées* ». Une solution est un optimum de Pareto (« *non-dominée* ») lorsque la valeur d'un critère ne peut être améliorée sans dégrader la valeur d'un autre critère. Il y a d'autres notions comme le « *point utopique* », la « *distance euclidienne* » nécessaire pour trouver la solution optimale. La plus courte distance euclidienne à partir du point utopique permet de trouver la solution optimale, le meilleur compromis entre les deux fonctions objectifs (critères à optimiser).

Le tri non-dominé

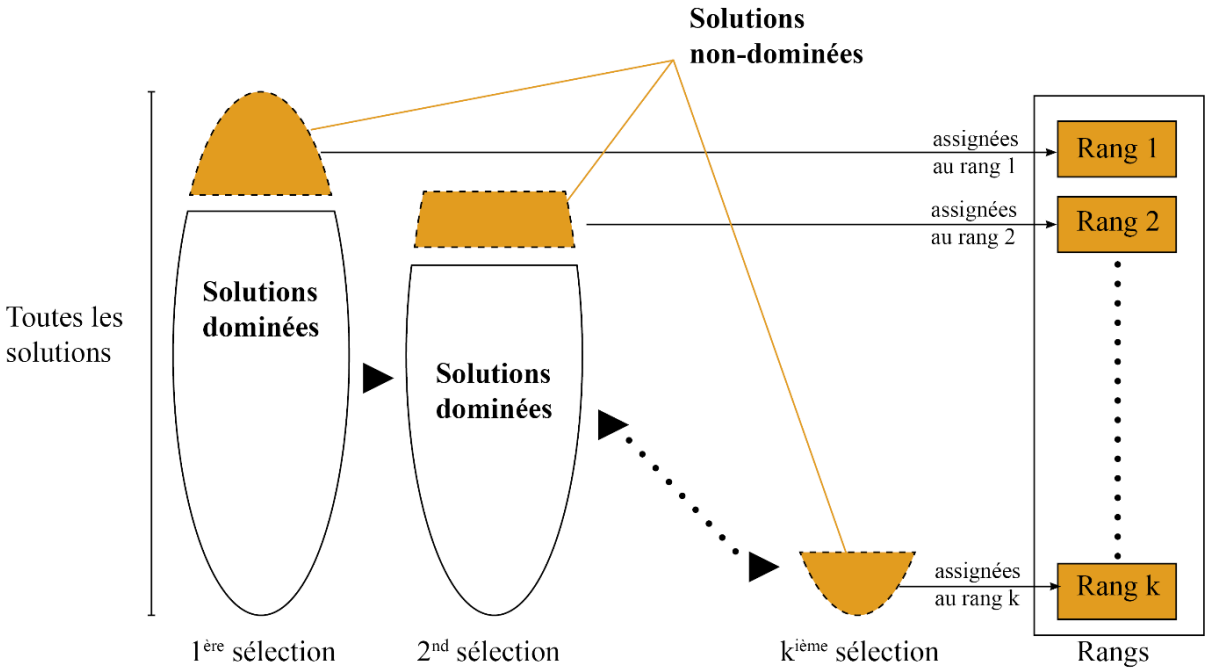


Figure 199 : Le tri non-dominé

Le tri non-dominé utilise le principe de dominance de Pareto pour trier les solutions d'une population. De nombreux algorithmes évolutionnaires multicritères utilisent cette méthode pour sélectionner les meilleurs individus d'une génération à l'autre. Il existe différents algorithmes pour réaliser cette opération qui suivent tous le même schéma présenté en Figure 199. Les meilleures solutions de l'ensemble de solutions sont calculées, elles constituent le front de Pareto, ou le premier rang de solutions. L'algorithme est de nouveau lancé avec l'ensemble des solutions restantes. Les meilleures solutions de cet ensemble constituent alors les solutions de rang 2. L'algorithme s'arrête lorsque toutes les solutions ont été assignées à un rang.

Quelle que soit la méthode utilisée, le résultat obtenu peut être représenté comme sur la Figure 200 présentée ci-dessous.

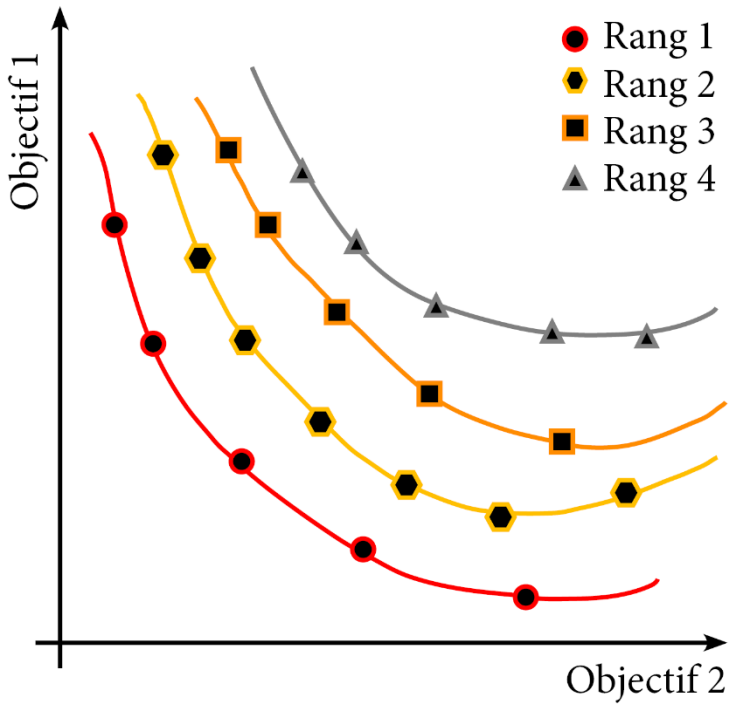


Figure 200 : Ensemble de solutions trié selon deux objectifs

Les algorithmes évolutionnaires multicritères

Parmi les algorithmes qui permettent de faire de l'optimisation multiobjectifs notamment en utilisant la méthode de Pareto, les algorithmes évolutionnaires ont la capacité de traiter ce type de problèmes dans leur totalité. En anglais, la littérature parle de « *multi-objective evolutionary algorithms* » (MOEA). Il existe de nombreux MOEA comme par exemple NSGA II, SPEA2, PESA, NPGA, PAES, MPGA, CAEP, MOPSO (Kunkle, 2005). Nous ne détaillerons pas ici le fonctionnement de tous ces algorithmes, mais seulement celui de NSGAII que nous utilisons régulièrement en pratique.

L'algorithme NSGAII

NSGA II, « *Non Dominated Sorting Genetic Algorithm* » (Deb et al., 2002), en français algorithme génétique élitiste de tri non dominé, est l'algorithme le plus utilisé dans le champ de la construction (Costa-Carrapiço et al., 2020). C'est l'algorithme que nous avons choisi d'utiliser pour nos différentes études comparatives et pour le développement de notre solveur d'optimisation multicritère sous contraintes.

C'est l'algorithme de tri et de sélection, appelée « *fast non dominated sorting* », qui fait la particularité de cet algorithme génétique. Dans la publication originale, il est testé avec une méthode de croissement binaire simulé (Deb & Agrawal, 1995) et une méthode de croisement polynomiale. En Figure 201 est présentée l'architecture générale de NSGAII telle que nous l'avons utilisée dans cette thèse.

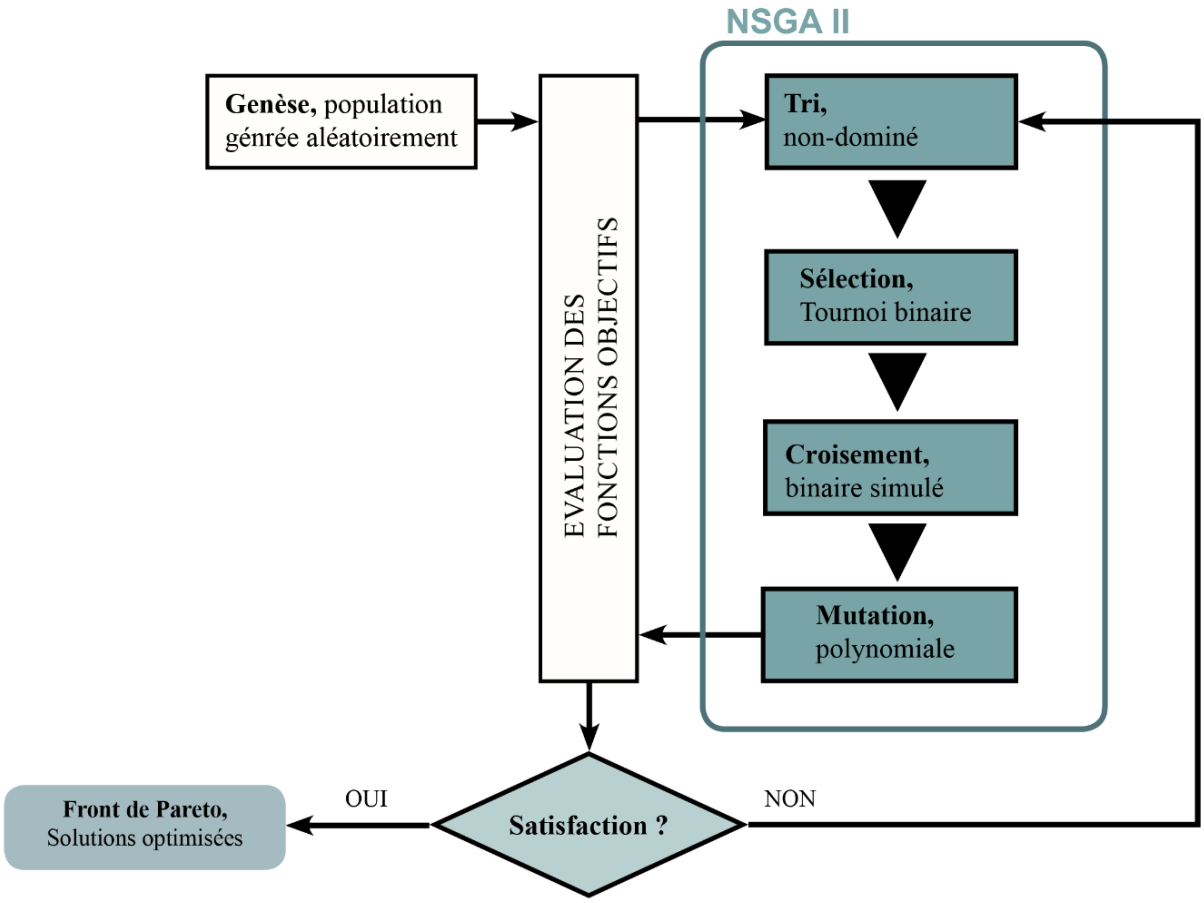


Figure 201 : Fonctionnement de NSGAII

Fast non-dominated sorting

Dans NSGAII les solutions sélectionnées sont choisies en fonction de leur performance, calculée à l'aide de la méthode de tri non-dominée. Elle permet de classer les solutions par rang et à l'aide de la distance d'encombrement qui permet de maintenir une bonne répartition des solutions dans la population comme le montre la Figure 202 (Deb et al., 2002).

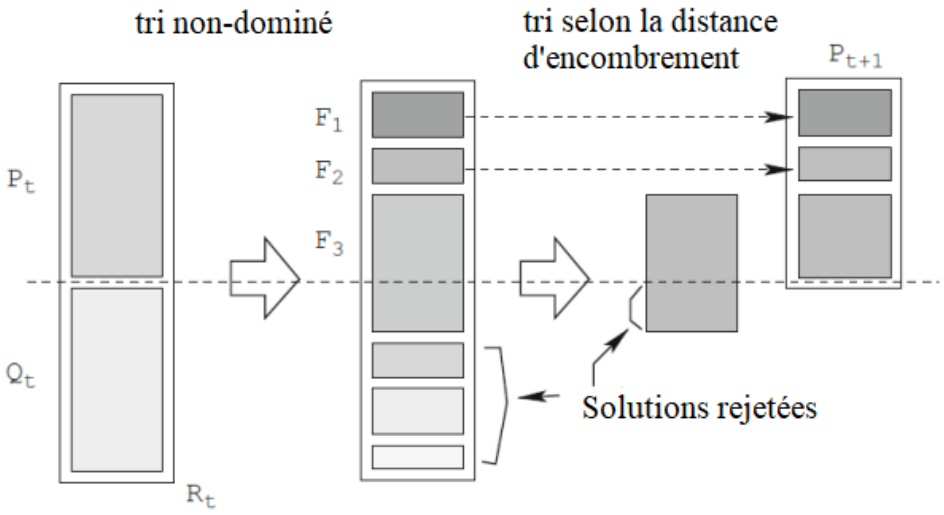


Figure 202 : Tri des solutions avec NSGAII (source:(Deb et al., 2002))

La distance d'encombrement mesure la proximité d'une solution par rapport aux solutions voisines (voir Figure 203), elle permet ainsi de faire une estimation de la densité de solutions. Plus cette mesure est petite, plus cette solution est encombrée par d'autres solutions. Le calcul de la distance d'encombrement intervient juste après le tri des solutions non-dominées qui donne le rang d'une solution. Avec ces deux informations (distance et rang), l'algorithme va pouvoir sélectionner des solutions à l'aide d'un « *opérateur de comparaison de la foule* » (de l'anglais « *crowded-comparaison operator* »). Si deux solutions ont le même rang, c'est la distance d'encombrement qui permettra de distinguer les solutions. La solution avec une distance d'encombrement supérieure sera considérée comme plus performante.

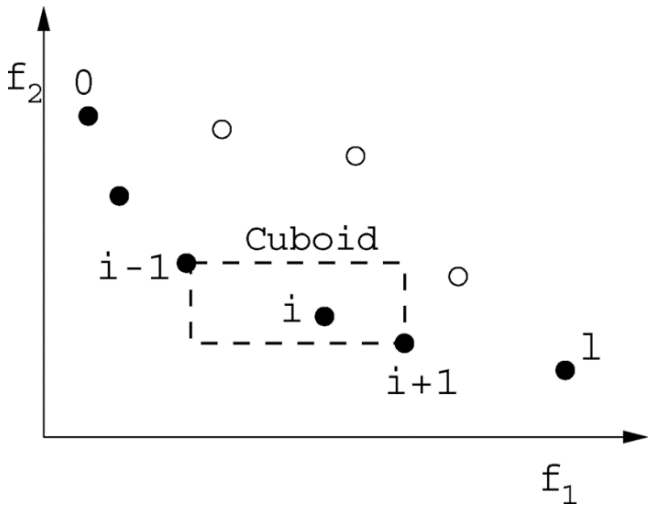


Figure 203 : Méthode de tri par la distance d'encombrement

La sélection par tournoi binaire

En programmation génétique, la méthode de sélection permet de sélectionner les meilleurs individus pour la reproduction des solutions. C'est une étape essentielle de l'algorithme. Selon la méthode utilisée, la pression de sélection, qui mesure au combien les meilleures solutions sont favorisées, peut être plus ou moins élevée. Si elle est trop faible, l'algorithme risque d'être lent. A l'inverse, si la pression est trop forte, l'algorithme risque de converger trop vite vers un optimum local (Miller & Goldberg, 1995).

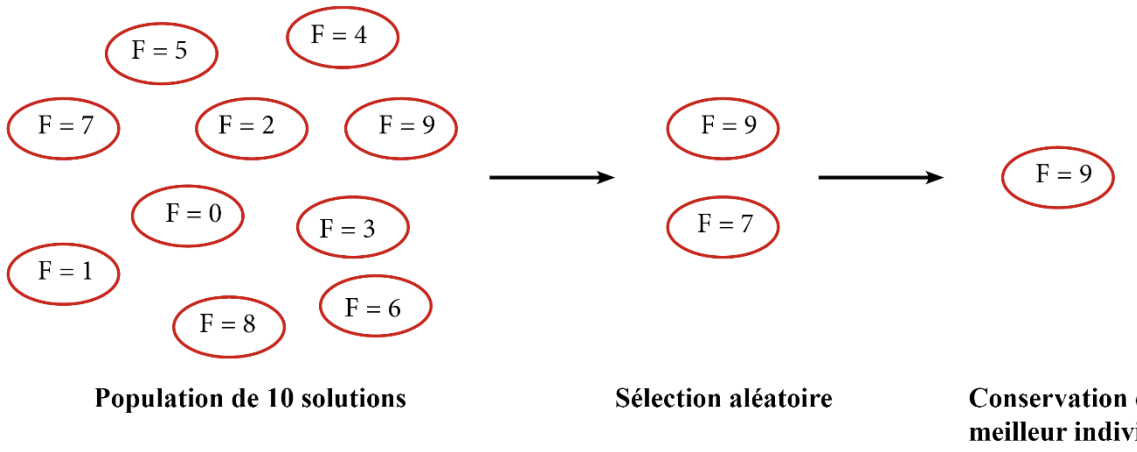


Figure 204 : Sélection par tournoi binaire

Dans une sélection par tournoi, une certaine quantité (appelée « *taille du tournoi* ») de solutions sont sélectionnées aléatoirement. La meilleure des solutions sélectionnées sera retenue pour l'étape du croisement. Plusieurs tournois sont réalisés pour pouvoir constituer une nouvelle population. Un tournoi binaire est un tournoi constitué de deux individus.

Le croisement binaire simulé

Le croisement binaire simulé, en anglais Simulated Binary Crossover (SBX) cherche à simuler le mécanisme du croisement à point unique utilisé dans la plupart des algorithmes génétique binaire dans des problèmes d'optimisation continus. Dans ce cas, la valeur d'une variable d'une solution est représentée par une chaîne de bits (composée de chiffre 0 ou 1). Le croisement à point unique choisit un point au hasard sur la longueur de la chaîne. Il suffit ensuite de croiser les bits situés sur chaque côté du point pour créer les chaînes des solutions enfants (voir Figure 205).

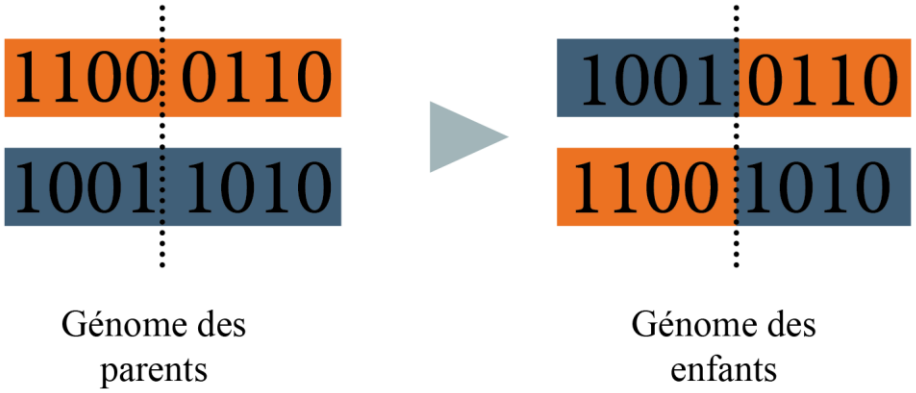


Figure 205 : Croisement binaire à point unique

La capacité de l'algorithme à choisir un point arbitrairement sur la chaîne (nommée « *pouvoir de recherche* ») est altéré lorsqu'il s'agit de variables continues. SBX utilise alors une méthode pour obtenir un niveau de pouvoir de recherche équivalent à celui du croisement à point unique utilisé pour l'optimisation discrète. A partir de deux parents X et Y, ce croisement génère deux enfants, p et q selon la formule suivante :

$$p_i = \frac{[\,(1-\beta)\,x_i + (1+\beta)\,y_i\,]}{2}$$

$$q_i = \frac{[\,(1+\beta)\,x_i + (1-\beta)\,y_i\,]}{2}$$

Où β est une variable (un facteur de dispersion) dont la valeur est comprise dans l'intervalle [0, ∞]. SBX adapte ce système à des problèmes multivariables en s'inspirant d'un mécanisme emprunté à la méthode de croisement uniforme. Lorsqu'il s'agit d'un problème à plusieurs variables, la longueur des chaînes diffère selon les variables, le point peut donc tomber sur les bits d'une variable et perturber cette dernière.

La mutation polynomiale

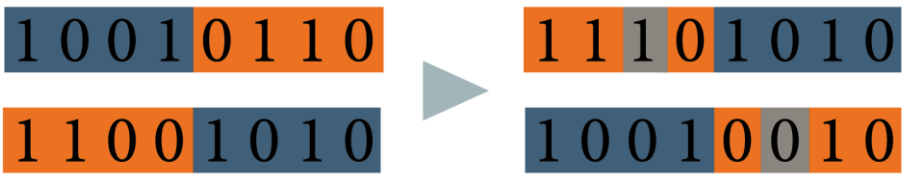


Figure 206 : Mutation binaire

Avec un système de mutation binaire, chaque bit de chaque solution enfant a une probabilité de changer (de passer de 0 à 1 ou de 1 à 0). NSGAII utilise la mutation polynomiale. Pour chaque solution enfant, chaque variable de décision a une probabilité de muter. Deux paramètre sont nécessaires pour définir l'opérateur : l'index de distribution (I) qui peut prendre n'importe quelle valeur positive et la probabilité de mutation (P) comprise entre 0 et 1. Dans la littérature, il est commun d'utiliser pour le paramètre P la valeur 1/n où n correspond au nombre de variables de décisions et la valeur 20 pour le paramètre I (Hamdan, 2012). Nous avons utilisé ces valeurs pour le paramétrage de notre solveur d'optimisation.

3.1.3. Les algorithmes à disposition des architectes

S'il existe de nombreux algorithmes d'optimisation, ils ne sont pas tous facilement accessibles aux architectes qui ne sont pas formés à la programmation informatique.

Les solveurs d'optimisation intégrés aux modeleurs 3D

Très peu de modeleurs 3D utilisés par les architectes intègrent des moteurs d'optimisation. A notre connaissance, seul Grasshopper® et Dynamo® donnent accès à des algorithmes d'optimisation. Sur Dynamo®, un seul algorithme peut être utilisé avec le package Optimo®. Il s'agit d'un solveur d'optimisation multicritère qui permet d'utiliser NSGAII. Dans un environnement comme celui de Grasshopper®, il existe actuellement huit plugins d'optimisation (sans compter les plugins d'optimisation spécifiques comme l'optimisation topologique), 11 solveurs au total (illustrés dans la Figure 207), permettant d'accéder à plus d'une vingtaine d'algorithmes d'optimisation. Trois solveurs seulement permettent de faire de l'optimisation multicritère.

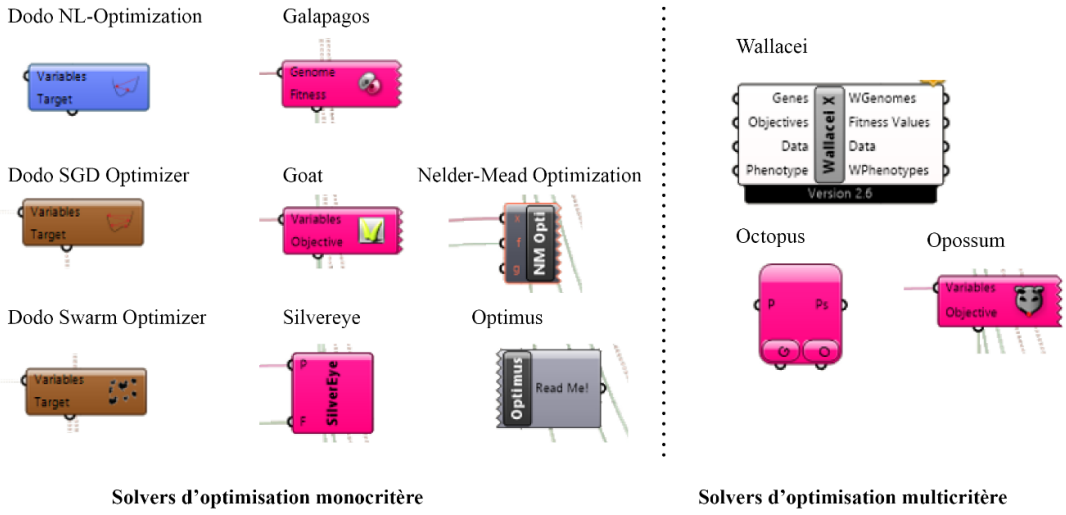


Figure 207 : Solveurs d'optimisation sur Grasshopper

Les solveurs d'optimisation monocritère

Nous avons déjà présenté le plugin Galapagos® dans le premier chapitre de cette thèse. Il est le solveur d'optimisation le plus connu chez les architectes et est désormais intégré à Grasshopper®. Il contient deux algorithmes monocritères, un algorithme génétique et un algorithme de recuit simulé.

- **Le plugin Dodo®**

Dodo® est une collection d'outils développés par Lorenzo Greco pour faire du *machine learning*, de l'optimisation et de la manipulation géométrique. Ce plugin met à disposition des algorithmes d'optimisation non linéaires connus. Les algorithmes proviennent da la librairie python NLOpt library. Son notamment implémenté dans ce plugin l'algorithme DIRECT, COBYLA (« *Constrained optimization by linear approximation* »), Subplex (« *SUBspace-searching simPLEX method* »). Deux solveurs sont disponibles dans la dernière version du plugin, notamment un plugin contenant un algorithme à essaims particulaires. Ils fonctionnent comme le composant du solveur Galapagos. Ils contiennent des petites interfaces très simples qui permettent de suivre la convergence des algorithmes à l'aide de graphiques. Il n'y a pas eu de mise à jour du plugin depuis le début de l'année 2019.

- **Le plugin Goat®**

Goat® est un composant d'optimisation développé par Simon Flöry, Heinz Schmiedhofer et Martin Reis. Ce composant fonctionne comme celui du solveur Galapagos. L'utilisateur peut choisir entre 5 algorithmes différents : DIRECT, COBYLA, Subplex, BOBYQA (« *Bound Optimization BY Quadratic Approximation* »), CRS2 (« *Controlled Random Search 2* »). Une petite interface permet de paramétrer l'algorithme, mais aucun tableau de bord ne permet de suivre le processus d'optimisation en cours de calcul. Le calcul s'arrête tout seul lorsque le temps de calcul limite est atteint. Aucun composant ne permet d'analyser les résultats ou de les récupérer, sauf si le composant est couplé avec les composants du plugin Colibri.

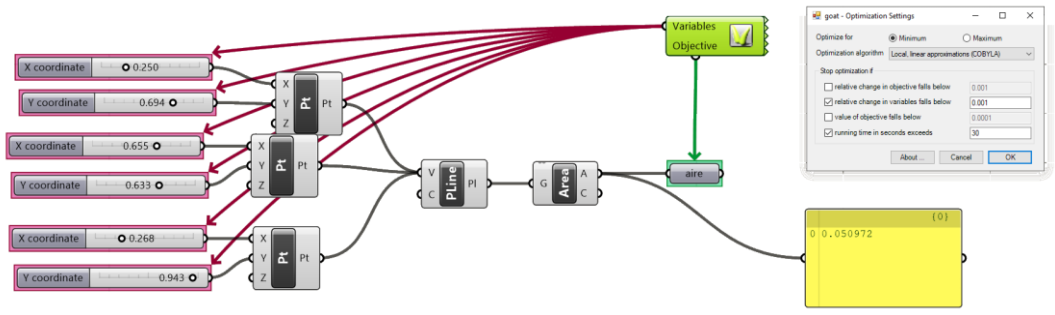


Figure 208 : Interface du plugin Goat

- **Le plugin Nelder-Mead Optimization®**

Ce plugin est composé d'un composant d'optimisation développé par Eckersley O'Callaghan's Digital Design Group. Un seul algorithme est implémenté dans ce composant, il est basé sur la méthode Nealder-Mead qui permet de faire de la recherche locale. Il intègre un système pour gérer des contraintes et une interface pour la saisie des paramètres et pour suivre le processus d'optimisation. Le composant n'a pas été mis à jour depuis aout 2018.

- **Le plugin Silvereye®**

Silvereye® est un composant d'optimisation qui utilise un algorithme d'optimisation par essaim particulaire (Cichocka et al., 2017). Il fonctionne comme Galapagos et est doté d'une petite interface graphique pour initialiser l'algorithme et suivre le processus d'optimisation. Une fonction permet d'exporter les résultats de la recherche en format csv. Le composant n'a pas été mis à jour depuis janvier 2018.

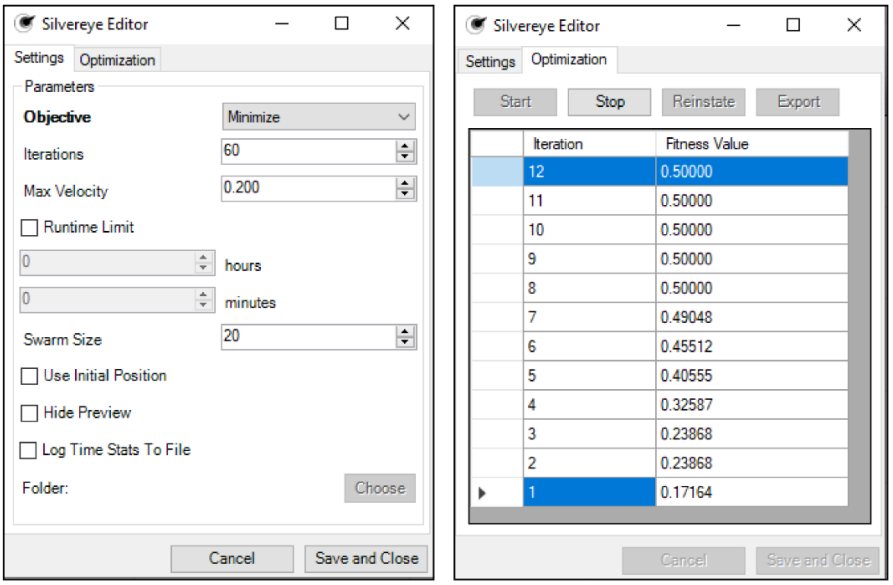


Figure 209: Interfaces du composant Silvereye

- **Le plugin Optimus®**

Optimus contient 9 composants nécessaires pour mettre en place un processus d'optimisation avec un algorithme évolutionnaire à évolution différentielle appelé « *Self-Adaptive Differential Evolution with Ensemble of Mutation Strategies* » (Cubukcuoglu et al., 2019). Comparé aux autres solveurs monocritère, Optimus est plus difficile d'accès mais donne plus de liberté, notamment pour le paramétrage de l'algorithme. Il ne contient pas d'interface, ni pour la saisie des paramètres, ni pour le suivi de l'exploration, et ne contient pas de composant pour la visualisation. La dernière mise à jour de ce plugin date de juin 2020.

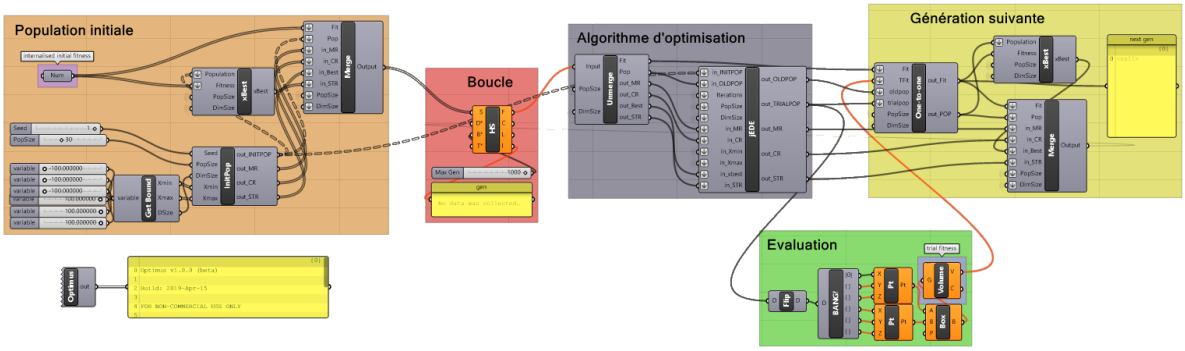


Figure 210: Script pour un processus d'optimisation avec Optimus

Les solveurs d'optimisation multicritère

Il y a désormais trois plugins Grasshopper® qui permettent de faire de l'optimisation multicritère.

- **Le plugin Octopus®**

Ce premier plugin est composé de 39 composants tous destinés à l'optimisation multicritère. Il est développé à l'université d'arts appliquées de Vienne et chez Bollinger+Grohmann Engineers. Deux algorithmes évolutionnaires sont implémentés dans son solveur : SPEA2 (« *Strength Pareto Evolutionary Algorithm* 2 ») et HypE (« *Hypervolume-based search algorithm* »). L'utilisateur a aussi le choix entre différents types d'opérateurs pour la mutation. Octopus est doté d'une interface graphique parmi les plus abouties sur Grasshopper® pour l'optimisation et permet de saisir les paramètres de l'algorithme. Il contient notamment un tableau de bord avec outil interactif représentant l'espace de solutions (voir Figure 211).

Octopus® est un plugin très complet. Il donne accès à des algorithmes de *Machine Learning* pour utiliser des modèles de substitution plutôt que de la simulation, mais aussi des composants pour faire des boucles compatibles avec le solveur, et des composants pour faire de la programmation génétique. Il ne contient pas de composants pour la visualisation des résultats, mais des graphiques interactifs sont intégrés au solver. Il est possible d'exporter les données accumulées au cours de l'exploration depuis l'interface du solveur. Cependant, selon notre expérience personnelle, le solveur Octopus® a tendance à faire *crasher* facilement le logiciel Rhinocéros, il est alors impossible de mettre la main sur les données produites avant le *crash*. Sa dernière mise à jour date de décembre 2018.

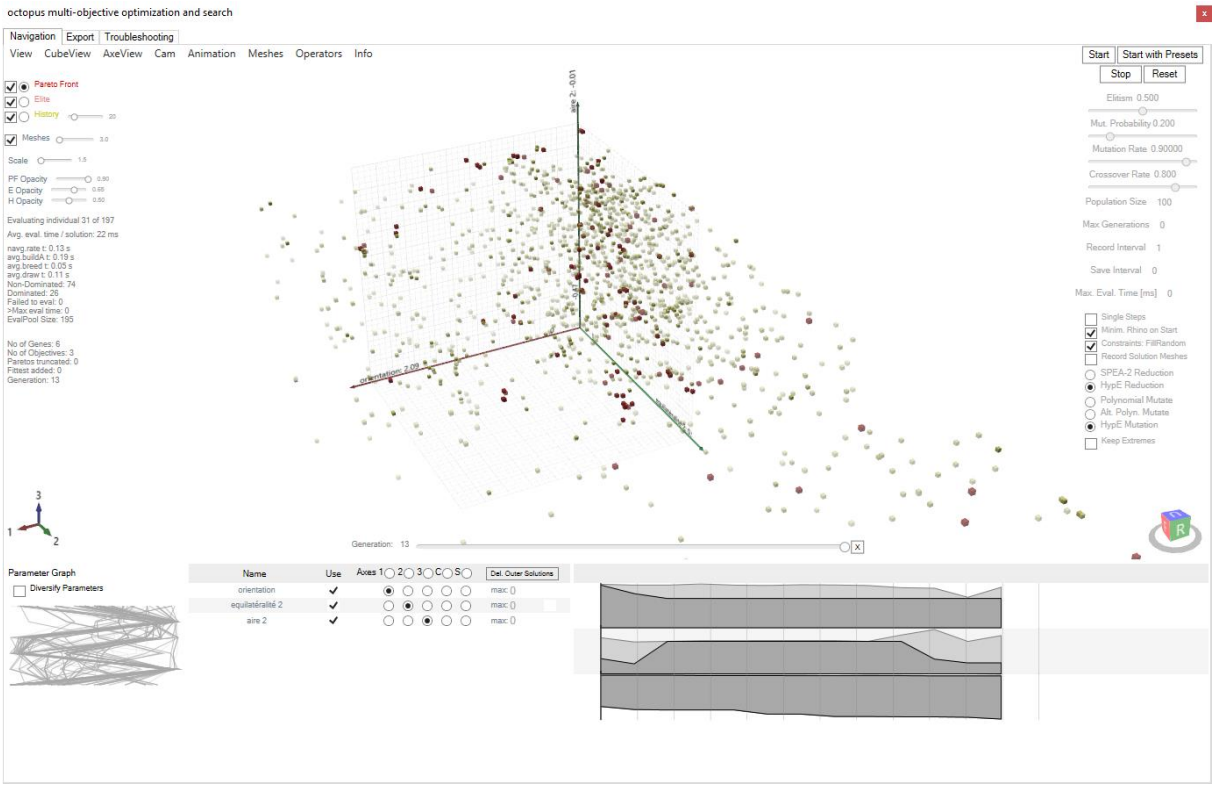


Figure 211 : Interface du solveur Octopus

- **Le plugin Wallacei®**

Wallacei® est un plugin d'optimisation développé à l'origine par Mohammed Makki dans le cadre d'un travail de doctorat. Il permet d'utiliser un algorithme évolutionnaire multi-objectif, NSGAII et utilise les bibliothèques python JMetal, LiveCharts et HelixToolkit. Il contient un composant servant de solveur nommé « *wallacei X* » doté d'une interface proche de celle d'Octopus, et des composants (« *Wallacei Analytics* ») pour l'analyse et la visualisation des résultats directement dans Grasshopper® (diagramme de coordonnées parallèles, graphique de l'écart-type, graphique en diamant, etc.). Ce solveur donne accès, à la fin de l'exploration, à l'historique des données accumulées au cours du processus qui peuvent être exploitées directement dans Grasshopper®.

Avant de développer notre propre plugin, nous avons régulièrement utilisé Wallacei® lors de nos expérimentations ou pour nos études comparatives car il est bien documenté, relativement stable (en comparaison d'Octopus), y compris avec le plugin Colibri® qui permet de récupérer les données sous forme de tableurs au cours de l'exploration et de faire des captures d'écran automatiques de chaque solution générée. Aussi, le plugin continue d'être régulièrement mis à jour, la dernière version sur Rhino 7 date d'avril 2022.

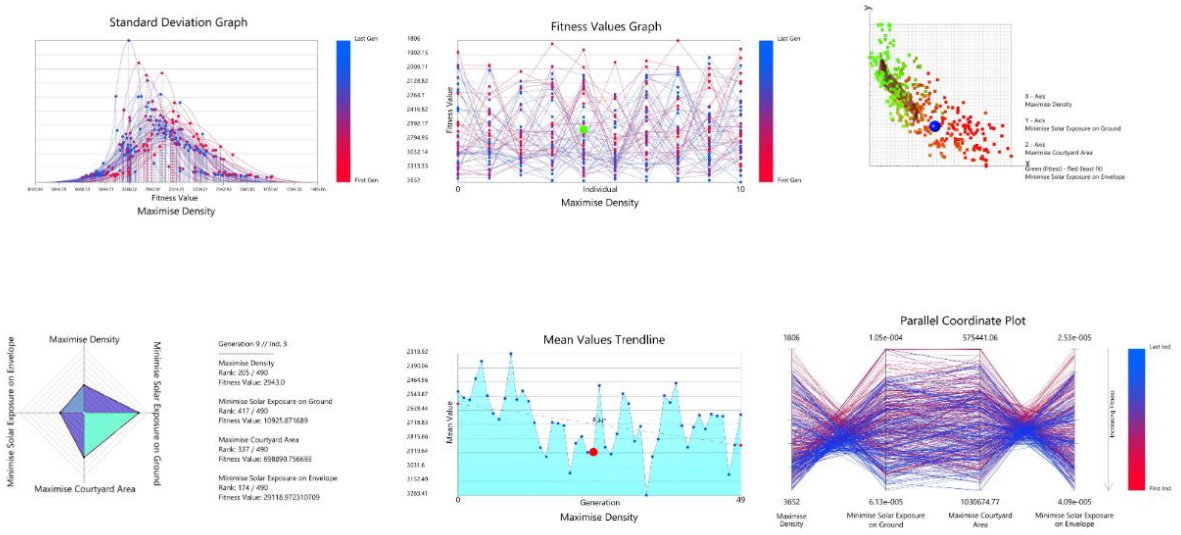


Figure 212: Outils d'analyse du plugin Wallacei (source: https://www.food4rhino.com/en/app/wallacei)

- **Le plugin Opossum®**

Opossum® est un composant d'optimisation développé par Thomas Wortmann qui s'appuie sur la librairie python RBFOpt (« *Radial Basis Function Optimization* »). Ce solveur contient deux algorithmes mono-objectifs : RBFOpt à base de modèles et CMA-ES (« *Covariance matrix adaptation evolution strategy* ») et 5 algorithmes multi-objectifs : RBFMOpt (la version multi-objectifs de RBFOpt), MACO (« *Modified Ant Colony Optimization* »), MOEA/D (« *Multiobjective Evolutionary Algorithm Based on Decomposition* »), NSGA-II et NSPSO (« *Non-dominated Sorting Particle Swarm Optimization ») i*ssus de la bibliothèque Pygmo 2. La dernière version d'opossum (3) inclut un outil visuel interactif pour la visualisation mais uniquement de l'optimisation mono-objectif.

Classification des algorithmes d'optimisation

Au début de ce chapitre, nous avons passé en revue les différents types d'algorithmes d'optimisation. Nous avons également listé les différents algorithmes d'optimisation à la disposition des architectes dans certains modeleurs 3D. En guise de synthèse, nous proposons en Figure 213 une classification des algorithmes d'optimisation pour des applications en conception architecturale où nous distinguons les classes d'algorithmes adaptés aux problèmes les plus fréquemment rencontrés des classes d'algorithmes destinés à des problèmes plus rares. Nous précisions en rouge dans l'arborescence les noms des algorithmes aujourd'hui implémentés dans Grasshopper®.

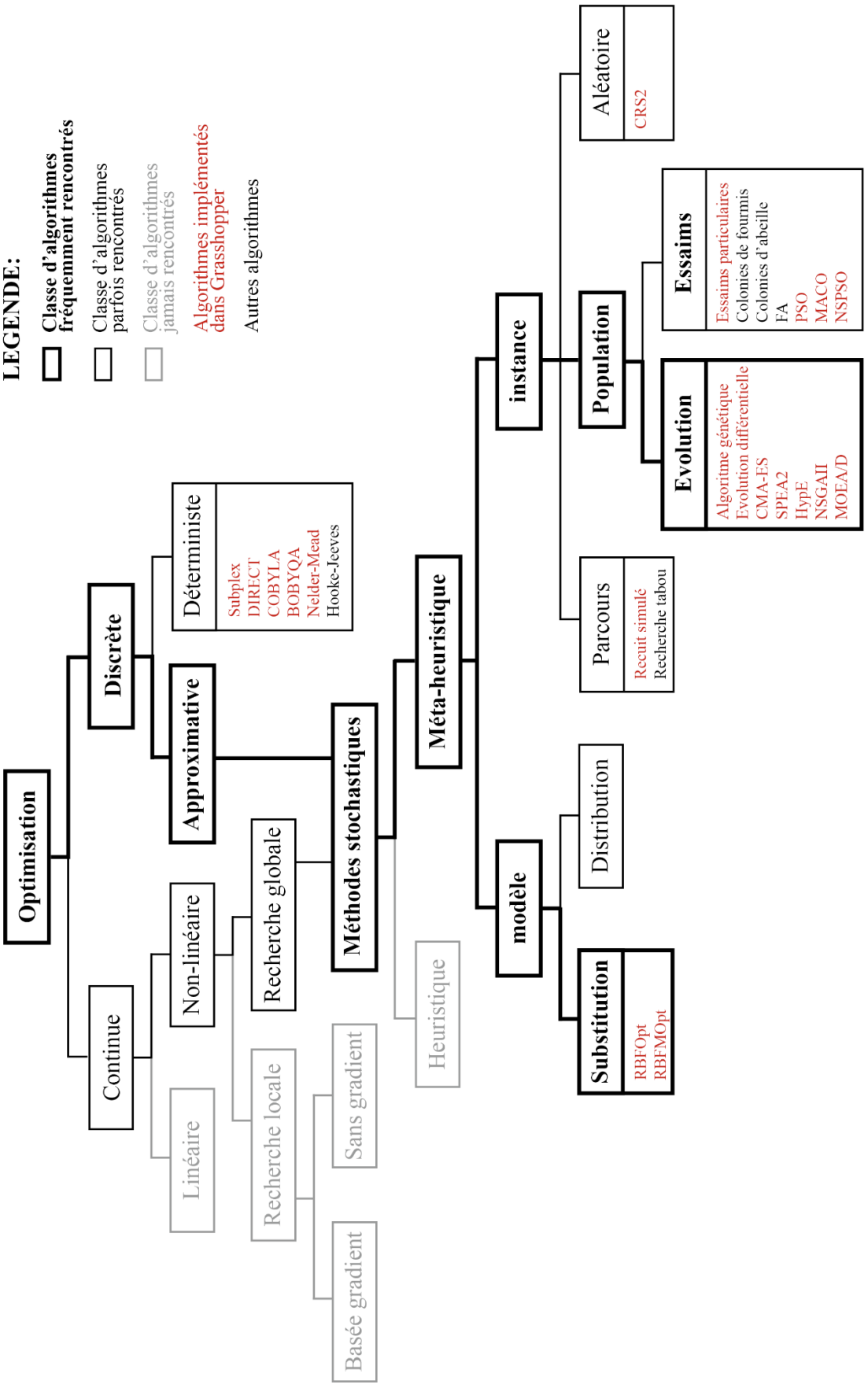


Figure 213 : Classification des algorithmes d'optimisation pour la conception architecturale

Le choix de l'algorithme

Aujourd'hui, les architectes ont à leur disposition un grand nombre d'algorithmes d'optimisation notamment via le logiciel Rhinocéros® et Grasshopper® son langage de programmation visuelle. Alors, quel solveur et quel algorithme choisir ?

Le théorème « *no free lunch* »

Il n'existe pas, à l'heure actuelle, un algorithme d'optimisation suffisamment efficace pour résoudre tous les types de problèmes. Ce phénomène porte un nom, le « *no free lunch theorem* » (Wolpert & Macready, 1997). Le choix de l'algorithme est donc une étape essentielle dans la recherche de solutions optimales.

Il y a plusieurs critères pour choisir les algorithmes. Nous l'avons vu au début de ce chapitre, la nature du problème (global/local, linéaire/non-linéaire, continu/discret, avec ou sans gradient, mono-objectif/multi-objectif) a un impact très fort sur le type d'algorithme à choisir. Une fois la nature du problème connue, le nombre d'algorithmes à disposition est plus restreint mais reste important.

Pour faire un choix, des critères qualitatifs peuvent être pris en compte par les architectes comme la complexité de prise en main du solveur, son degré d'interactivité, sa stabilité, mais surtout des critères quantitatifs comme la vitesse de convergence de l'algorithme (considéré comme le critère le plus important), mais aussi la stabilité de l'algorithme (sa fiabilité) et les temps de calcul (T. Wortmann & Nannicini, 2017). Il est difficile de faire des généralités sur la performance des algorithmes d'optimisation, car celle-ci dépend des caractéristiques du problème sur lequel ils sont testés.

L'optimisation pour la performance du bâtiment est un cas particulier puisque, bien souvent, les temps de calcul de l'algorithme, à chaque itération, sont dérisoires comparés aux secondes, voire minutes nécessaires au calcul de la fonction objectif pour chaque simulation. Ainsi, les auteurs spécialisés sur ce type de problèmes utilisent le nombre d'évaluation effectuées comme référence, soit en évaluant la même quantité de solutions pour chaque algorithmes (Hamdy et al., 2016), soit en comparant le nombre d'évaluations exécutées plutôt que le temps écoulé (T. Wortmann & Nannicini, 2017).

Le choix d'un algorithme mono-objectif

Les algorithmes implémentés dans des solveurs sur Grasshopper pour faire de l'optimisation mono-objectif ont fait l'objet de plusieurs études comparatives. En 2015, il a été montré qu'utiliser l'algorithme SBO, nommé RBFOpt (« *radial basis function* ») et implémenté

dans Opossum® sur un problème d'optimisation d'enveloppe (sur 15 variables) pour l'éclairage naturel, était beaucoup plus efficace que d'utiliser l'algorithme génétique de Galapagos® (T. Wortmann et al., 2015). En 2016, cet algorithme est de nouveau comparé à celui de Galapagos® mais aussi à l'algorithme DIRECT implémenté dans Goat® sur quatre problème différents, deux portants sur l'éclairage naturel, et deux autres sur la structure. Pour les deux premiers, RBF est plus performant et l'AG de Galapagos® n'est jamais l'algorithme le plus efficace (T. and G. N. Wortmann, 2016).

En 2017, les algorithmes évolutionnaires de Galapagos® et de Octopus® ont été comparés à un algorithme d'optimisation d'essaims particulaires implémenté dans Silvereye®. L'étude a montré sur un problème de structure que, selon les paramètres utilisés, l'algorithme Silvereye® pouvait être plus performant que les algorithmes évolutionnaires (Cichocka et al., 2017).

En 2019, une étude compare 16 algorithmes d'optimisation sur 15 problèmes d'optimisation énergétique du bâtiment utilisant Energy Plus comme logiciel de simulation thermique (Waibel et al., 2019). Les algorithmes testés sont ceux de Galapagos®, Goat®, Opossum®, Silvereye® et quatre autres algorithmes issus d'une bibliothèque GitHub : un algorithme génétique simple (SGA), une stratégie d'évolution (ES), un algorithme d'essaim particulaire (PSO), et un algorithme d'essaims particulaires entièrement informé (FIPS). Ces algorithmes sont testés sur des bâtiments de bureaux, d'habitation, et une école. Le nombre de zones thermiques est variable selon les problèmes. Selon les problèmes, les variables considérées peuvent être l'orientation du bâtiment, le dimensionnement des fenêtres, les matériaux et les points de consigne. Seul un problème interroge la forme du bâtiment. Parfois, seul le fichier météo change entre deux problèmes. Aucun algorithme n'est systématiquement plus efficace que les autres sur les 15 problèmes traités. Finalement, l'auteur déconseille d'utiliser l'algorithme génétique de Galapagos®, le PSO de Silvereye®, les algorithmes Sbplex et CRS2 de Goat® car ils ne sont pas stables. Les algorithmes PSO et FIPS de la bibliothèque GitHub, l'algorithme de recuit simulé de Galapagos®, et les algorithmes CMA-ES et RBFOpt d'Opossum® sont recommandés par l'auteur. Les deux derniers sont parmi les meilleurs pour la plupart des problèmes sans avoir besoin d'utiliser des paramètres spécifiques au problème. CMA-Es est plus adapté lorsque les temps de calcul des simulations sont importants. L'algorithme génétique de Galapagos® est très souvent la moins bonne alternative.

Ainsi, comme le dit le théorème du « *no free lunch* », aucun algorithme n'est systématiquement plus performant qu'un autre, quel que soit le problème d'optimisation rencontré en conception architecturale et performancielle. Sans test préalable, il est difficile de savoir quel algorithme sera le plus performant pour un problème donné. Dans la pratique, il est impensable d'envisager de tester tous les algorithmes. Ainsi, nous recommandons pour l'optimisation mono-objectif intégrant des critères de performance environnementaux, d'utiliser les algorithmes du solveur Opossum® plutôt que d'utiliser l'algorithme génétique de Galapagos®.

Le choix d'un algorithme multi-objectifs

Lorsqu'il s'agit de comparer des algorithmes d'optimisation multi-objectifs, la tâche est encore plus complexe car il s'agit d'évaluer la qualité des fronts de Pareto générés, et de nombreux critères d'évaluation existent : la convergence vers le front de Pareto, la distribution de l'ensemble de Pareto (pour mesurer la diversité des solutions sur le front de Pareto), le nombre de solutions sur le front de Pareto, le temps d'exécution de l'algorithme. Malheureusement, un algorithme est rarement le plus performant sur tous ces critères (Weerasuriya et al., 2021). Plusieurs études ont déjà cherché à comparer différents algorithmes sur différents problèmes, mais aucune d'entre elles ne permet de conclure qu'un algorithme serait systématiquement plus performant qu'un autre.

En 2020, une étude comparait l'algorithme HypE de Octopus® et l'algorithme NSGAII de Wallacei® avec un algorithme SBO nommé RBFMOpt (*Radial Basis Function Multi-objective Optimization*) sur un problème d'optimisation de la morphologie d'un quartier résidentiel situé en Chine (T. Wortmann & Natanian, 2020). Les auteurs ont conclu qu'à l'exception du critère de robustesse, les trois algorithmes se sont comportés de manière similaire.

En 2021, les mêmes auteurs ont réalisé une seconde étude où 7 algorithmes multiobjectifs ont été comparés : les deux algorithmes d'Octopus® (HypE et SPEA2), les 5 algorithmes d'Opossum® (NSGAII, MOEAD, MACO, NSPSO et RBFMOpt) (T. Wortmann & Natanian, 2021). Les algorithmes ont été testés sur deux problèmes de morphologie d'un bâtiment, un premier avec une morphologie d'îlot fermé avec une cour intérieure, et un second avec une typologie de tour de grande hauteur. Les fonctions objectives étaient l'accès au soleil et la densité. Sur ces problèmes, RBFMOpt, HypE et SPEA2 ont eu les meilleures performances, suivis par NSGAII qui obtient des performances moyennes.

En 2021, une étude a comparé la performance de 4 algorithmes : NSGAII, PSO, GSA (« *Gravitational Search Algorithm* »), FA (« *Firefly Algorithm* ») sur un problème d'optimisation de la morphologie d'un bâtiment pour maximiser les zones de vent et le confort thermique (Weerasuriya et al., 2021). Les performances des algorithmes ont été comparées avec différentes tailles de population et pour une quantité variable de générations. Ainsi, l'algorithme NSGAII peut parfois être le premier contributeur au front de Pareto général (avec l'ensemble des solutions générées par les 4 algorithmes) par exemple au bout de 500 évaluations et avec une population de 16 et être l'avant dernier contributeur au bout de 3000 évaluations pour une population de 48.

Plus que pour l'optimisation mono-objectif, le choix de l'algorithme pour un problème d'optimisation multicritère n'est pas une tache évidente. D'après les études citées, les premiers algorithmes multi-objectifs implémentés dans Grasshopper (Hype, SPEA2 et NSGAII) semblent adaptés aux problèmes de conception impliquant des critères de performances environnementales ainsi que l'algorithme à base de modèle RBFMOpt récemment implémenté dans Opossum®.

Le paramétrage d'un algorithme génétique

Autant que le choix de l'algorithme, son paramétrage est une étape importante pour assurer sa performance, qu'il soit mono-objectif ou multi-objectifs. Mais le paramétrage d'un algorithme génétique est une tâche qui peut être complexe. Il existe heureusement dans la littérature des lignes directrices théoriques pour faciliter la mise en pratique.

Paramétrage de la population

Les premiers paramètres d'un algorithme génétique sont la taille de la population enfant et le nombre de générations. Tous les solveurs permettent la saisie de ces deux paramètres. La taille de la population peut être différente de la taille de la population de départ. Par exemple, Galapagos® propose un paramètre pour « *booster* » la taille population de départ à l'aide d'un coefficient multiplicateur de la taille de la population enfant. Le nombre de générations peut être directement fixé avec un entier ou résulter d'un temps de calcul limite ou d'un nombre de générations stagnantes (sans nouvel optimum) limite comme c'est le cas avec le solveur Galapagos®.

Nous venons de voir que la taille de la population et le nombre d'évaluations (donc le nombre de générations) peuvent avoir un fort impact sur les performances des algorithmes (Weerasuriya et al., 2021). Si la population est trop petite, l'algorithme risque de rapidement

converger vers un minimum local, mais si la population est trop grande, les temps de calcul risquent d'être trop importants. Plus la taille de la population augmente, plus la quantité de générations nécessaires pour que l'algorithme converge augmente (Gotshall & Rylander, 2002).

Alors quelle est la taille idéale de population ? Les chercheurs sont partagés sur ce point. Une analyse théorique de l'influence de la taille de la population sur un algorithme évolutionnaire réalisée en 2021 a montré que les populations de grandes tailles ne sont pas toujours utiles (T. Chen et al., 2012), mais les études basées sur des expériences informatiques tendent à montrer qu'une grande population permettrait d'augmenter la diversité et faciliterait le travail de l'algorithme. D'après une revue de littérature de 2019 sur les paramètres des algorithmes génétiques (Hassanat et al., 2019), certains suggèrent d'utiliser des populations contenant entre 50 et 100 individus, d'autres disent que plus la population est grande, plus l'algorithme a des facilités à converger. D'autres encore parlent d'une taille limite à partir de laquelle les solutions générées seront de moins bonne qualité. Il semblerait surtout que la taille optimale de la population enfant dépende des paramètres des opérateurs de mutation et de croisement.

Pour plus de performance, certains chercheurs étudient des populations dynamiques, c'est-à-dire des tailles de populations enfants qui évoluent au cours du processus d'optimisation (Tan et al., 2001).

Concernant le nombre de générations, il n'existe pas à notre connaissance de méthode pour connaitre le nombre exacte d'évaluations nécessaires pour maximiser la probabilité de convergence vers un optimum global. Certains auteurs utilisent une méthode empirique qui consiste à multiplier par 100 le nombre de paramètres lui-même multiplié par le nombre de fonctions objectifs (R. Koenig et al., 2020).

Paramétrage des opérateurs

D'autres paramètres ont un impact fort sur le fonctionnement d'un algorithme génétique ; ce sont les paramètres de probabilité des opérateurs de croisement et de mutation.

Le taux de croisement correspond à la probabilité qu'un croisement se produise entre les chromosomes de deux solutions parents. Un taux de croisement de 100 % signifie que toutes les solutions d'une génération sont issues d'un croisement. Un taux de croisement à 0 % signifie que les solutions d'une génération sont identiques aux solutions de la génération précédente. Dans le solveur Galapagos®, ce paramètre est remplacé par le taux de maintien (« *Maintain* »), qui équivaut à 100 – le taux de croisement.

Le taux de mutation correspond à la probabilité pour chaque solution enfant que son gène mute, ce paramètre contrôle la diversité des solutions dans la population. Un taux à 0 signifie qu'une mutation n'aura lieu pour aucune solution enfant. A l'inverse, un taux de 100 % signifie que toutes les solutions vont muter à chaque nouvelle génération. Dans le solveur Galapagos, il n'est pas possible de paramétrer le taux de mutation, mais un bouton permet d'augmenter ce taux au cours de l'exploration.

En 2019, Ahmad Hassanat et ses co-auteurs publient une revue de littérature sur les principaux travaux portant sur le choix du taux de mutation, du taux de croisement et de la taille de la population pour les algorithmes génétiques (Hassanat et al., 2019). Le Tableau 11 présente l'ensemble des paramètres utilisés dans les différents travaux de cette revue de littérature.

Tableau 11 : Paramètres (exprimés entre 0 et 1) utilisés dans différentes études sur le paramétrage des algorithmes génétiques (source: (Hassanat et al., 2019))

Taille de la population	Taux de croisement	Taux de mutation		Taille de la population	Taux de croisement	Taux de mutation
30	0.95	0.01		100	0.6	0.02
80	0.45	0.01		30	0.6	0.05
100	0.9	1		76	0.8	0.05
100	0.8	0.005		4000	0.5	0.001
50	0.9	0.03		30	0.6	0.001
50	1	0.03		50	0.8	0.2
50	0.8	0.01		50	0.9	0.1
15	0.7	0.05		30	0.9	0.1
100	1	0.003		20	0.8	0.02
30	0.8	0.07		50	0.9	0.01
100	0.8	0.01		20	0.9	0.3
500	0.8	0.2		330	0.5	0.5
40	0.6	0.1		100	0.8	0.1
100	0.8	0.3				

Parmi ces études, une a montré qu'utiliser un taux de mutation de 1/n où n correspond au nombre de variables est efficace. Le croisement est plus efficace sur des populations de

grande taille alors que la mutation est plus efficace sur les populations de petites tailles même s'il reste important de combiner les deux (Schlierkamp-Voosen, 1993). Comme pour la taille de la population, certains auteurs se sont intéressés aux opérateurs adaptatifs, qui ont des paramètres de probabilité qui varient au cours de l'exploration (Smith & Fogarty, 1997).

Finalement, suite à cet état des connaissances sur les mécanismes d'optimisation nous pouvons affirmer qu'il n'existe pas d'algorithme et de paramétrage idéal mais que de nombreuses études proposent des recommandations qui permettent de guider les praticiens de l'optimisation dans leur choix.

Au terme de cette première partie, nous avons identifié la nature des problèmes d'optimisation rencontrés en conception architecturale et identifié les différents types d'algorithmes qui permettent de résoudre cette catégorie de problème. Nous savons désormais qu'il s'agit principalement de problèmes d'optimisation globale, discontinue, sans gradient, multicritère qu'il convient de résoudre à l'aide de métaheuristiques.

Nous avons présenté ensuite les différentes méthodes d'optimisation multicritère existantes et recommandons l'utilisation de la méthode dite de Pareto pour les problèmes de conception architecturale. Nous connaissons désormais le fonctionnement d'un algorithme génétique, en particulier le fonctionnement de l'algorithme génétique NSGAII que nous utilisons régulièrement autant sur les cas d'étude issus de la pratique que sur des cas théoriques.

Enfin, nous avons identifié et testé l'ensemble des solveurs d'optimisation à la disposition des architectes principalement accessibles avec le langage de programmation visuelle du modeleur 3D Rhinocéros®. Bien que nous recommandions l'usage d'Opossum® pour les problèmes monocritères et Wallacei® pour les problèmes multicritères, nous avons proposé une classification des algorithmes qui permet aux architectes débutants en optimisation de s'orienter dans la diversité des algorithmes qui sont aujourd'hui à leur disposition.

Désormais, nous avons suffisamment de connaissances techniques sur l'optimisation pour nous intéresser au premier problème technique identifié lors de nos expérimentations sur des cas réels en agence : l'intégration des contraintes, notamment avec les algorithmes évolutionnaires.

3.2. Intégration des contraintes avec des algorithmes génétiques

Dans cette seconde partie du chapitre 3, nous cherchons à identifier les méthodes les plus adaptées pour l'intégration des contraintes dans des problèmes d'optimisation de conception architecturale. Dans un premier temps, nous exposons l'enjeu que représente les contraintes lorsqu'on fait de l'ENAPE, en revenant sur sa définition en architecture, qui diffère de sa définition en mathématiques, et sur le manque d'outils dédiés.

Dans un second temps, nous présentons 7 méthodes d'intégration des contraintes identifiées dans deux types de littératures scientifiques, celle du design génératif et performatif, et celle de l'optimisation. Nous illustrons comment implémenter ces méthodes en utilisant un cas d'étude issue de la pratique en agence.

Dans un troisième temps, nous présentons les résultats d'une étude comparative où ces 7 méthodes ont été testées sur un même problème d'optimisation. Les méthodes sont comparées à la fois sur leur capacité à générer des solutions qui respectent des contraintes, mais aussi à générer des solutions performantes et des solutions qui correspondent aux attentes des architectes en termes de design.

Cette étude comparative et la plupart des éléments qui constituent cette partie de la thèse ont déjà fait l'objet de publications (Duclos-Prévet et al., 2022b, 2021).

3.2.1. L'enjeu de l'intégration des contraintes

Définition d'une contrainte

Les contraintes en architecture

Le sens du terme « contrainte » n'est pas identique dans un contexte mathématique ou informatique et dans un contexte architectural. Les « contraintes de l'architecte » ont beaucoup été étudiées et leur définition ou classification varie selon les auteurs. Nous en donnons ici quelques exemples.

Dans son ouvrage *How designer think* (Lawson, 2006), Bryan Lawson explique qu'un designer doit faire face à 4 types de contraintes. Il y a les contraintes formelles (comme les proportions, la texture, les couleurs, la géométrie, l'organisation visuelle de l'objet), les contraintes symboliques, les contraintes radicales (au sens de fondamental, lié à la nature même de l'objet à concevoir) et les contraintes pratiques (les contraintes qui se réfèrent à la réalité de la mise en œuvre) que d'autres pourraient appeler contraintes techniques, catégorie dans laquelle on trouve les contraintes liées au climat, à la structure ou la durabilité du bâtiment. Il

distingue aussi deux domaines de contrainte, le domaine interne et le domaine externe qui concerne les relations de l'objet avec l'extérieur et précise qu'elles sont souvent plus difficiles à gérer. Les contraintes de l'architecte peuvent être générées par quatre profils de personnes différentes : le législatif, on parle de contrainte « réglementaire », le client, qui est à l'origine du programme, l'usager (le futur usager du bâtiment) qui est souvent différent du client, et l'architecte lui-même qui, à l'image d'un artiste, ne cherche pas uniquement à résoudre un problème, mais souhaite apporter sa propre vision à un projet.

Dominique Raynaud propose en 2004, une autre classification des contraintes (Raynaud, 2004). Il distingue les « contraintes sociales » définies comme rapport à autrui, des contraintes « physiques ». La première catégorie est donc définie par un tier comme le législateur, le client ect… Il distingue ensuite quatre types de contraintes : les contraintes simples (pouvant être une obligation ou une interdiction, pouvant être simplement ou doublement bornée), ceux sont les contraintes prises isolément par opposition aux contraintes composées qui sont la résultante d'une combinaison de contraintes simples. Il faut aussi distinguer les contraintes de disposition qui impose explicitement une forme ou une position contrairement aux contraintes de principes. Enfin, il y les contraintes latérales par opposition aux contraintes centrales. L'auteur explique aussi que la contrainte est négociable puisqu'il existe en droit français un « arsenal de dérogations prévues par le droit », que les contraintes sont soumises à interprétation, et que les architectes connaissent des méthodes de détournement de la contrainte.

D'après Philippe Boudon une contrainte est « une donnée que l'architecte est tenu de prendre en considération dans son projet », mais que sa prise en compte implique des choix. Il donne comme exemple *« un terrain sur lequel s'implante un projet comprend nécessairement des contraintes (nature du sol, vents dominants, ect.). Dès lors qu'il y a conception, ces contraintes sont interprétées, elles peuvent être prises en compte de diverses façons. L'architecte fait des choix, c'est lui qui décide du caractère positif ou négatif d'une contrainte en fonction du projet qui s'élabore. Il peut décider de tirer parti d'une contrainte ou au contraire la réduire autant que faire se peut. L'ensoleillement peut être recherché ou combattu ; il en va de même pour les vents. »* (Boudon et al., 1994). Il précise que comme un écrivain, l'architecte peut lui-même s'imposer des contraintes.

Ainsi, les architectes entendent par « contraintes », l'ensemble des règles physiques ou législative, et exigences des acteurs impliqués dans le projet qu'ils doivent prendre en compte lors de la conception. Cette définition de la contrainte n'est pas similaire à celle qu'on utilise en mathématique.

Les contraintes en optimisation

En mathématiques, la définition d'une contrainte est claire, et elle se distingue d'un critère d'optimisation. Il s'agit d'une condition inviolable, qui doit nécessairement être satisfaite à la différence d'un critère qui doit être satisfait au mieux. Les contraintes définissent l'espace des solutions possibles, appelées « région faisable » ou « ensemble admissible ». Une solution qui ne respecte pas les contraintes est dites « infaisable » ; et, dans la pratique, cette solution, bien souvent, a peu d'intérêt pour l'architecte. Par conséquent, évaluer les performances des solutions infaisables est une perte de temps, ce qui accentue le problème des temps de calcul souvent prohibitifs pour la pratique en agence d'architecture.

En architecture, rares sont les auteurs qui distinguent les critères des contraintes. Pendant longtemps, Brian Lawson n'a pas fait de distinction entre un critère et une contrainte. Dans la dernière version de son livre, l'auteur évoque cette distinction en citant une définition de (Portillo & Dohr, 1994) où le sens du mot « contrainte » se rapproche un peu plus de la définition des mathématiciens: « *Les critères font systématiquement référence à des fonctions et à des processus d'évaluation reposant sur un objectif, tandis que les contraintes se rapportent à des fonctions généralement caractérisées par leur caractère restrictif et plus étroitement alignées sur les exigences de solutions spécifiques*. (Traduit de l'anglais) ».

Les mathématiciens distinguent deux types de contraintes : les contraintes d'égalité et les contraintes en inégalité. Dans un modèle analytique (que l'on peut formuler mathématiquement), une contrainte est définie comme une fonction, par exemple $g(x)=0$ est une contrainte d'égalité, et $h(x) \geq 0$ est une contrainte d'inégalité. En pratique, un architecte qui fait de l'optimisation n'utilisent pas de méthodes de résolution analytique mais des méthodes de résolution numérique. Il y a alors peu d'intérêt, si contrainte il y a, de chercher à savoir de quel type de contrainte il s'agit, d'autant plus qu'une contrainte en égalité peut se formuler comme deux contraintes en inégalité.

Les contraintes en optimisation appliquées à l'architecture

Lorsque l'architecte fait de l'optimisation, c'est à lui de définir ce qui sera un critère et ce qui sera une contrainte en fonction de ce qu'il attend de son exploration numérique. En conception générative, ce qui est formulé comme une contrainte doit obligatoirement être respectée par le modèle, ce qui ne signifie pas pour autant qu'il s'agit d'une contrainte du projet.

Les contraintes de l'architecte issue du législateur, comme les règles d'urbanisme, les normes handicapées, peuvent être directement reformulées comme une contrainte inviolable.

Par exemple, l'exploration de la morphologie d'un bâtiment dans un contexte urbain va souvent être accompagné d'une contrainte d'inclusion dans un volume capable.

Les contraintes de l'architecte issues du client, comme les exigences du programme en termes de surface, peuvent selon le problème ou la phase du projet être une contrainte ou un critère d'optimisation. Par exemple, dans le cadre d'un concours pour une commande publique, si on cherche à optimiser la disposition de bâtiments sur une parcelle, le volume à bâtir qui découle des exigences du programme est une contrainte. Dans un autre contexte, comme une commande privée, l'objectif peut être de trouver un compromis entre confort des espaces intérieurs et volume à bâtir, alors ce dernier peut être formuler comme un critère d'optimisation.

Certaines contraintes de l'architecte, dites pratiques, comme les contraintes liées au climat, peuvent aussi être formulées différemment selon le contexte. L'architecte peut vouloir interroger sur un projet des dispositifs dans le but de trouver celui qui permettra de faire entrer le plus de lumière naturelle possible dans le bâtiment, il faudra alors formuler cette exigence comme un critère d'optimisation. A l'inverse, il peut chercher à optimiser un autre aspect comme la protection contre le rayonnement solaire pour le confort thermique d'été en s'assurant de respecter les normes concernant l'éclairage naturel. Ces dernières prenant souvent la forme de seuil minimum à atteindre, il s'agit d'un cas typique de contrainte en inégalité.

Certaines volonté de l'architecte peuvent être formulées comme des contraintes. Les considérations esthétiques, lorsqu'elles peuvent prendre une forme quantitative, en sont un bon exemple. Le degré de symétrie, la complexité géométrique, le degré de lissage d'une courbe, l'hétérogénéité des formes peuvent faire l'objet d'évaluation et être intégrée au processus d'optimisation comme des contraintes intangibles.

Ces contraintes esthétiques peuvent aussi avoir une utilité technique. La formulation d'une contrainte forte permet aussi de réduire la taille de l'espace de solutions, et peut dans certains cas éviter l'explosion combinatoire. Cela peut s'avérer utile lorsque l'on doit étudier une enveloppe dans sa globalité (voir Les différentes échelles de modélisation au chapitre 1 p92). Cependant, il ne faut pas trop en faire car l'inflation des contraintes peut entraver le déroulement prévu de l'algorithme en imposant trop de contraintes au problème à résoudre. Dans de telles circonstances, il peut être nécessaire de convertir certaines contraintes dures en critères d'optimisation.

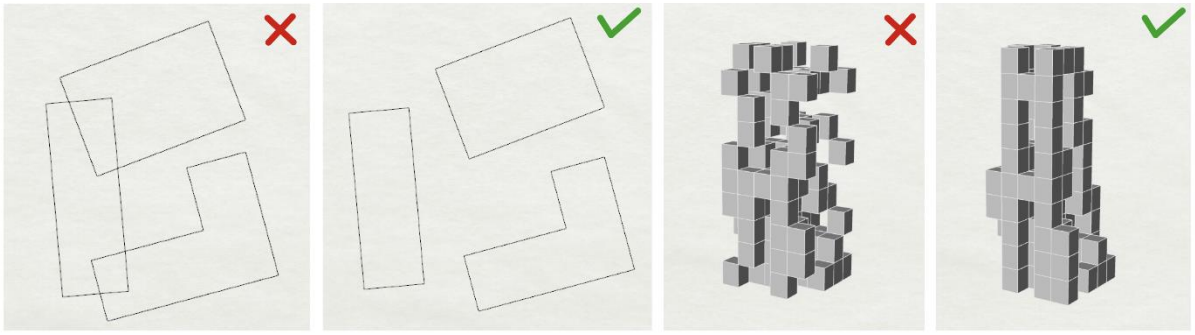


Figure 214 : Exemples de contraintes de type "aberration"

Il est aussi fréquent de devoir faire face à un type de contrainte que l'architecte ne soupçonne pas a priori, mais qui sont récurrentes dans la pratique de la conception computationnelle. Nous les avons appelées les « aberrations ». Il s'agit des cas où même le meilleur modèle paramétrique trouvé pour modéliser un problème de conception, définit un espace de solutions qui contient des aberrations.

Par exemple, interroger la disposition de trois bâtiments distincs sur une parcelle ne peut se faire difficilement autrement qu'en faisant varier les coordonnées d'un point fixé sur l'empreinte de chaque bâtiment et ainsi générer des solutions où les empreintes entrent en intersection. Un autre exemple parlant est celui des « cubes qui flottent » lorsqu'on fait de l'exploration de la morphologie du bâtiment avec des voxels (pixels en 3D), c'est-à-dire en discrétisant le volume capable en cube (voir Figure 214). On peut alors obtenir des solutions structurellement impossible à réaliser.

Un manque d'outils dédiés

Les outils orientés architecte

Parmi les 8 solveurs d'optimisation monocritère disponibles sur Grasshopper®, un seul permet d'intégrer des contraintes, il s'agit du plugin Nealder-Mead Optimisation® (voir Figure 215). Il utilise la méthode de Kreisselmeier-Steinhauser qui permet d'agréger des fonctions objectifs et plusieurs contraintes en une seule fonction objectif sans contrainte.

Les 3 solveurs d'optimisation multicritère disponibles sur Grasshopper® ne font pas beaucoup mieux. Octopus® et Wallacei® permettent d'utiliser la méthode bien particulière du « *death penalty* » avec un système de filtre. Cette méthode consiste à éliminer les solutions non faisables du processus pour en générer d'autres. Cette méthode de gestion des contraintes est déconseillée sauf si la quantité de solution infaisables est restreinte proportionnellement à la taille de l'espace de solutions, sinon la recherche risque de stagner (Coello Coello, 2002).

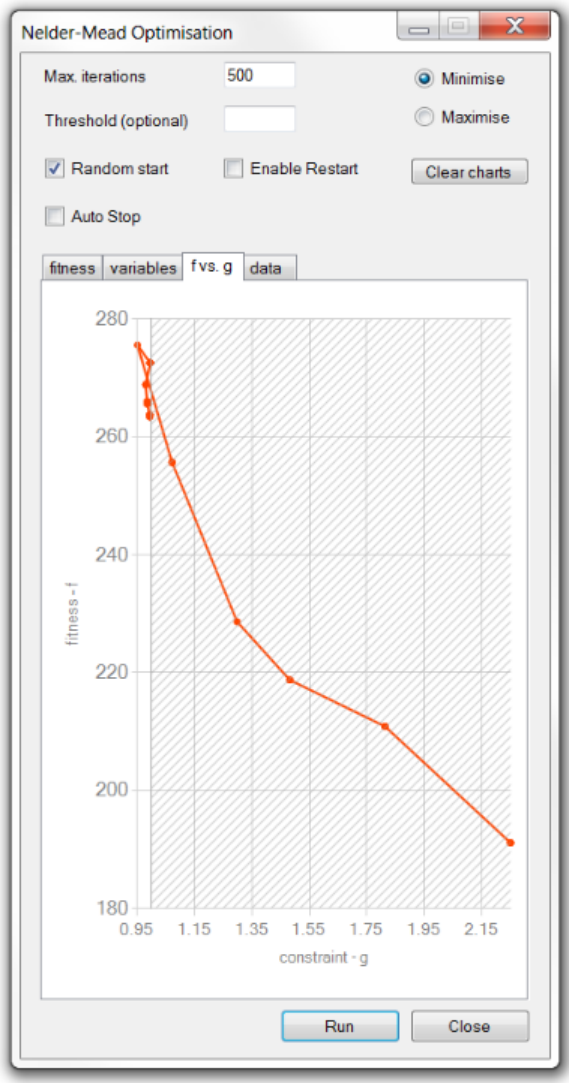


Figure 215 : Interface du plugin Nealder-Mead Optimization®

Les outils orientés ingénieurs

En regardant du côté des ingénieurs, il existe des logiciels d'optimisation pouvant être couplés à des moteurs de simulation environnementale utilisables pour la conception de bâtiments comme Matlab®, GenOpt® ou encore jEPlus+EA®. Ces trois exemples cités sont les plus utilisés dans le domaine de l'optimisation de la consommation énergétique des bâtiments (Kheiri, 2018) mais ne sont pas accessibles pour des architectes.

Matlab® propose des solutions d'optimisation conçues pour intégrer facilement les contraintes avec une différenciation automatique entre les objectifs et les contraintes pour plusieurs solveurs. Les contraintes sont ensuite gérées en fonction de l'algorithme d'optimisation sélectionné. Il s'agit de méthodes génériques pouvant être utilisées sur de nombreux types de problèmes.

GenOpt® (Generic Optimization Program), qui peut être couplés à EnergyPlus®, TRNSYS®, DOE-2® et d'autres, peut intégrer des contraintes à l'aide de fonctions de pénalité, une méthode générique que nous détaillerons plus tard dans cette partie du chapitre.

jEPlus+EA est un outil d'optimisation couplé à EnergyPlus® ou TRNSYS® avec une interface facile à utiliser (voir Figure 216) qui intègre NSGAII et qui permet de gérer des contraintes. S'il est possible de manipuler des paramètres géométriques avec cet outil, l'opération est particulièrement fastidieuse, même pour une simple boîte.

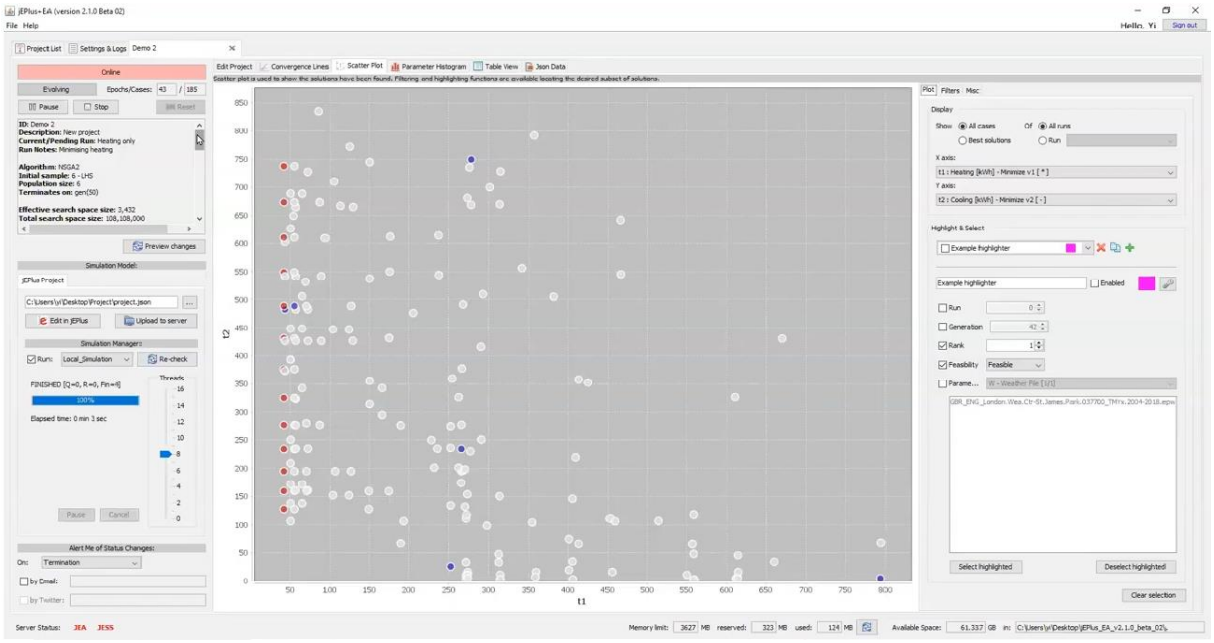


Figure 216 : Interface graphique de ijEPlus+EA v2

Ainsi, les trois outils d'optimisation les plus utilisés en thermique du bâtiment intègrent des méthodes de gestion des contraintes. Cependant, ces outils basés sur la modélisation numérique n'ont pas été conçus pour faire de l'optimisation de formes géométriques complexes. Nous savons depuis le chapitre 1 que les problématiques abordées par les ingénieurs diffèrent sensiblement de celles qui intéressent les architectes, à commencer par les variables d'intérêt. Ainsi, nous formulons l'hypothèse que les méthodes d'intégration traditionnellement utilisées par les ingénieurs ne sont pas nécessairement adaptées aux problèmes d'optimisation rencontrés par les architectes.

L'exemple de la façade de l'Institut Faire Face

Pour illustrer cet enjeu des contraintes, nous nous appuierons dans ce chapitre sur un exemple d'application issu de la pratique de l'agence Architecture-Studio que nous avons en partie présenté dans le chapitre précédant au sujet de l'Exploration en phase de développement (p212). Il s'agit d'un cas de système de protection solaire, celui des laboratoires de recherche

de l'Institut Faire Face à Amiens. Ce projet désormais construit présente une façade au R+1 composée de lames en aluminium déjà présente lors de la phase de concours comme on peut le voir sur la perspective de rendu présentée en Figure 217.

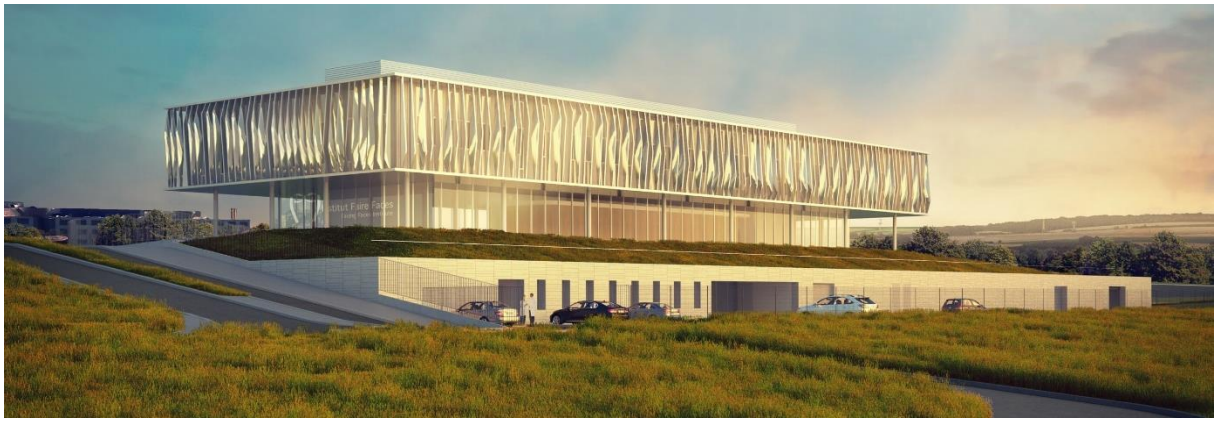


Figure 217 : Institut Faire Face, image de rendu de concours (source: architecture studio)

Ces grandes lames verticales en aluminium sont utilisées pour protéger la façade du rayonnement direct non désiré dans les laboratoires et favoriser un éclairage naturel indirect. Dès la phase d'avant-projet sommaire, nous souhaitions travailler à l'optimisation de l'orientation et de la forme de chacune des lames pour maximiser la lumière naturelle à l'intérieure et minimiser l'ensoleillement direct du vitrage de la façade des laboratoires. Cependant, interroger l'orientation des lames comme nous le souhaitions, c'est-à-dire en acceptant qu'elles puissent être toutes orientées différemment, notamment pour jouer sur la réflexion de la lumière (voir Figure 218), a entraîné plusieurs conséquences visibles sur la Figure 219.

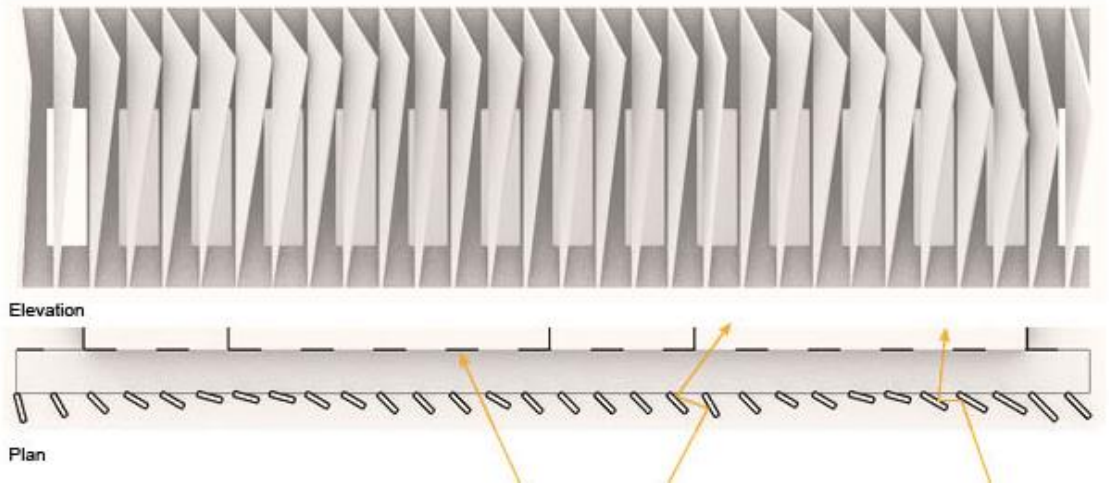


Figure 218 : Principe de fonctionnement de la façade en plan et en élévation

Pour un grand nombre de combinaisons de paramètres, les lames entrent en collision et, lorsque des lames voisines ont des formes trop différentes, cela crée un effet général en dents

de scie non désiré. Pour pouvoir réaliser l'exploration formulée de cette manière, il faut donc être en mesure d'intégrer deux contraintes : une contrainte de type « aberration » pour éviter les collisions, et une contrainte esthétique pour éviter l'effet en dents de scie. Pendant les phases de conception de ce projet, qui ont eu lieu quelques mois avant le démarrage officiel de cette thèse, nous n'étions pas en mesure de proposer une telle modélisation. Ainsi, en pratique, l'optimisation a été réalisée avec un paramètre d'orientation commun pour toutes les lames situées sur la même façade.

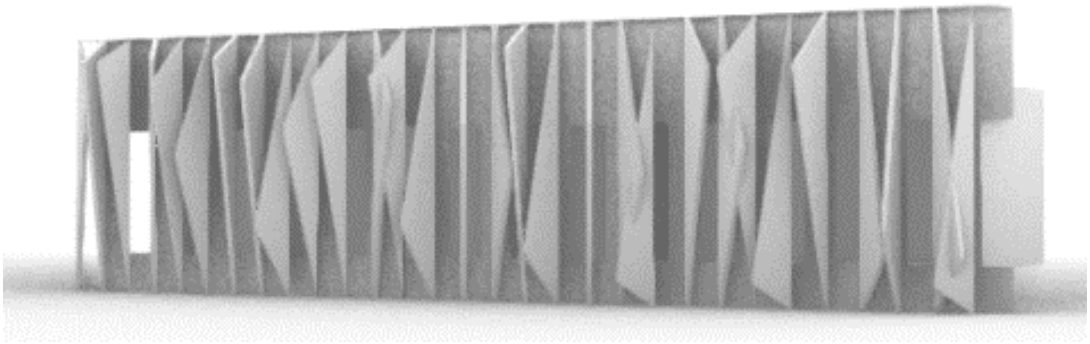


Figure 219 : Illustration de l'effet en dent de scie et des collisions de lames

Pour pouvoir gérer ce type de problème à l'avenir, nous avons souhaité reprendre ce cas d'étude et rechercher des solutions qui permettent de le résoudre sans avoir à réduire sa complexité. Ainsi, après avoir identifié des méthodes d'intégration des contraintes qui seront présentées dans la section suivante, nous les avons testées sur ce problème avec l'algorithme NSGAII (Deb et al., 2002).

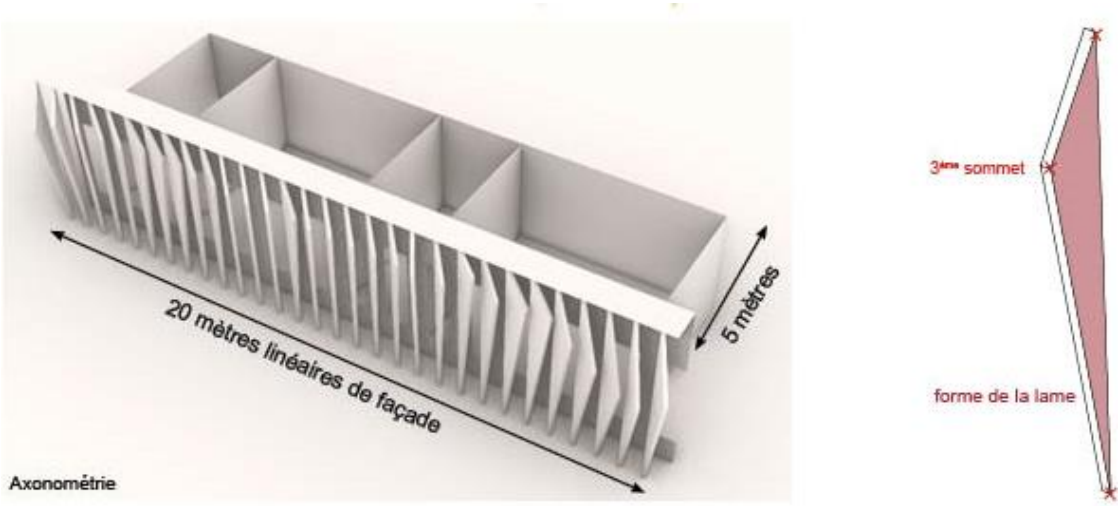


Figure 220 : Modélisation 3D et morphologie d'une lame

Chaque méthode a été testée sur un morceau de façade de 20 mètres linéaires exposé plein Sud composé de 30 lames. Pour chaque lame, un seul paramètre permet de modifier sa forme, il s'agit de la coordonnée en Z du troisième sommet du triangle (voir Figure 220). Les modèles paramétriques ont été réalisés avec des composants Grasshopper® et pour certaines méthodes impliquant l'usage de modèle génératif, les modèles ont était réalisés à l'aide de composants Python que nous avons-nous même développé.

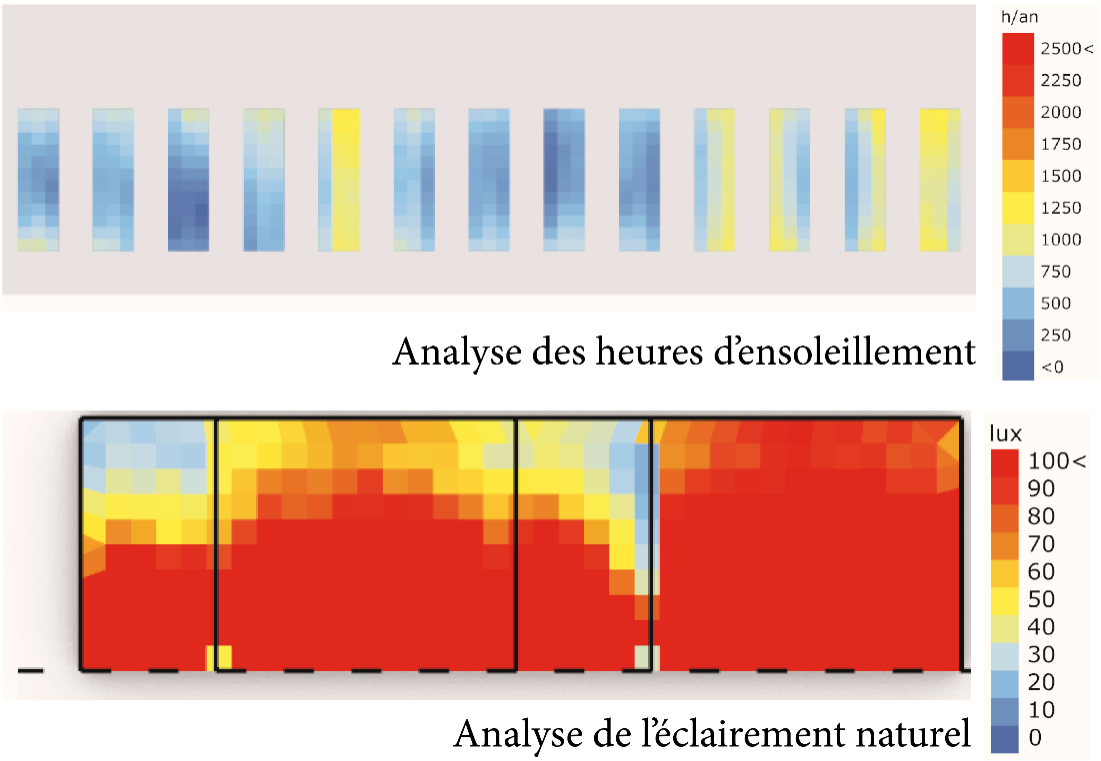


Figure 221 : Visualisation des analyses d'ensoleillement et d'éclairement

L'ensoleillement direct a été étudié à l'aide du plugin Ladybug® (Roudsari et al., 2013) et le logiciel de simulation de l'éclairage Radiance® a été utilisé pour l'éclairement naturel calculé le 21 décembre à 12h à l'aide du plugin Honeybee®. Les paramètres de réflexion des matériaux utilisés correspondent aux recommandations HQE pour les phases amonts de conception. La fonction d'évaluation pour la lumière correspond à l'éclairement moyen sur l'ensemble des surfaces des laboratoires. La fonction d'évaluation pour l'ensoleillement correspond au nombre d'heures d'ensoleillement total pour l'ensemble du vitrage en façade.

Pour chaque méthode d'intégration des contraintes, 1 000 solutions ont été générées avec des populations de 50 individus, soit 20 générations au total. Un groupe contrôle de 1 000 solutions a été généré aléatoirement. Chaque exploration a nécessité environ 10 heures de calcul avec un ordinateur de 15 CPU Intel Xeon at 2.4 GHz. Nous avons utilisé le plugin Colibri® pour stocker les résultats dans un fichier csv et effectuer des captures d'écran automatiques.

L'analyse des résultats et la visualisation des données ont ensuite été réalisées à l'aide Microsoft Excel® et le logiciel R®.

3.2.2. Les méthodes d'intégration des contraintes

Nous avons identifié, dans la littérature scientifique sur l'optimisation mathématique et dans la littérature sur le design génératif, différentes méthodes permettant de gérer les contraintes avec des algorithmes évolutionnaires. Nous en avons testé 7 au total sur notre cas d'étude. Trois méthodes sont génériques, elles peuvent être facilement répliquées sur d'autres problèmes, nous les avons appelées les « méthodes réplicables ». Quatre autres méthodes sont des heuristiques, elles ont nécessité la programmation d'algorithmes adaptés à ce problème spécifique, nous les avons appelées les « méthodes ad hoc ». Un groupe de 1000 solutions générées au hasard ont aussi été évaluées, il s'agit d'un groupe de contrôle.

Les méthodes réplicables

Les trois méthodes réplicables que sont (1) la fonction de pénalité, (2) l'ajout d'une fonction objectif et (3) la méthode de Constraint-NSGAII ont toutes trois été expérimentées avec le même modèle paramétrique simple présenté en Figure 222. Celui-ci compte 60 variables, 2 variables pour chaque lame. La première variable sert à paramétrer la forme triangulaire de la lame, il s'agit de la hauteur du troisième sommet du triangle. La seconde variable permet de faire varier son orientation. Au total, l'ensemble de solutions compte 2.25^{E}+55 possibilités.

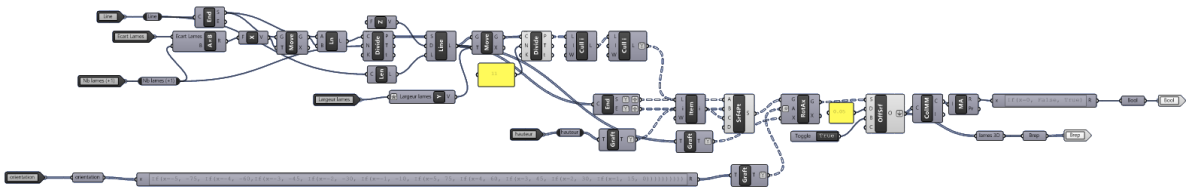


Figure 222: Script du modèle paramétrique simple

<u>Fonction de pénalité</u>

L'utilisation d'une fonction de pénalité est une technique bien connue en optimisation pour gérer les contraintes (Michalewicz & Schoenauer, 1996). Elle consiste à dégrader les valeurs des fonctions objectives en utilisant un système de malus, que l'on appelle le coefficient de pénalisation. Cette méthode est souvent efficace sauf si le sous-ensemble de solutions qui

respectent les contraintes (la « région faisable ») est proportionnellement très étroit comparé à l'ensemble de solution général (Coello Coello, 2016).

Il existe plusieurs types de fonctions de pénalisation : la « death penalty », les pénalités statiques, les pénalités dynamiques, les pénalités recuites, les pénalités adaptatives, les pénalités co-évolutionnaires (Coello Coello, 2002). Seules les deux premières peuvent être utilisées avec les solveurs « boîtes noires » présentés dans la section précédente. Les autres méthodes nécessitent d'adapter l'algorithme génétique et donc de modifier le programme informatique.

Dans un premier temps, nous avons commencé par tester la méthode de la « death penalty » en utilisant le solveur Wallacei®. Cette méthode consiste à supprimer toutes les solutions infaisables d'une population et à générer de nouvelles solutions jusqu'à obtenir une population constituée uniquement de solutions faisables. Cette méthode ne fonctionne pas sur ce cas d'étude très contraint. La taille de la région faisable est tellement étroite comparée à l'espace de solutions défini par le modèle paramétrique que l'algorithme n'a pas été en mesure de générer une génération.

Dans un second temps, nous avons testé la méthode (1), celle des pénalités dites « statiques », où les coefficients de pénalité sont fixes tout au long du processus d'optimisation contrairement aux pénalités adaptatives. Ce type de pénalité est simple à implémenter et aussi considéré comme plus robuste que d'autres pénalités plus sophistiquées (Michalewicz & Schoenauer, 1996). Il existe 3 façons de définir les pénalités :

(1) soit tous les individus infaisables sont pénalisés de la même manière,
(2) soit la valeur de la pénalité s'adapte au degrés d'infaisabilité de la solution,
(3) soit la pénalité s'adapte au niveau d'effort nécessaire pour rendre la solution faisable, ce qui s'appelle la « distance de faisabilité ».

Lorsqu'il y a peu de contraintes et peu de solutions faisables, comme c'est le cas sur ce problème, il semblerait qu'il soit plus efficace de calculer la distance de faisabilité plutôt que le nombre de contraintes violées (Richardson et al., 1989).

Ainsi, nous avons défini deux fonctions pour calculer le degré de violation des contraintes : l'aire totale des surfaces de lames entrant en collision, et l'amplitude des mouvements de la courbe dessinée par les lames en façade pour éviter l'effet en dents de scie. Les deux fonctions ont été facilement programmées avec des composants Grasshopper®, et leurs résultats sont ensuite agrégés afin d'obtenir la pénalité. Cependant, elle doit ensuite être pondérée de façon à pénaliser une fonction objective plus qu'une autre. Cette étape a nécessité d'évaluer un premier échantillon de l'ensemble de solutions pour connaître l'échelle des valeurs

que prennent les deux fonctions objectives. Ainsi, la principale difficulté de cette méthode réside dans la définition des coefficients de pénalité.

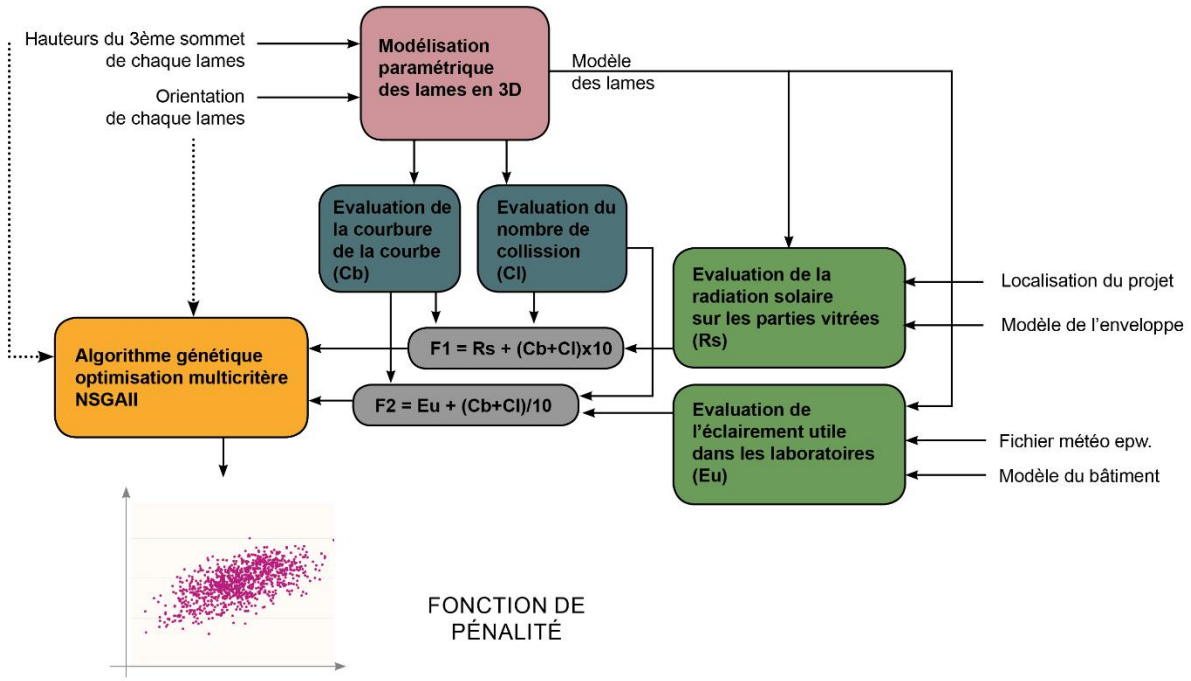


Figure 223 : Architecture du modèle général avec la méthode des fonctions de pénalisation appliquée au cas d'étude des laboratoires de l'Institut Faire Face

L'architecture du modèle général comprend le modèle paramétrique en rose, le modèle d'évaluation en vert, le modèle d'intégration des contraintes en bleu, les fonctions objectifs en gris, et l'algorithme d'optimisation en jaune est présenté en Figure 223.

Ajouter une fonction objectif

Une autre méthode, encore plus simple à implémenter consiste à formuler les contraintes comme des fonctions objectifs. Dans cette méthode (2), il est aussi nécessaire de mesurer le degré d'infaisabilité des solutions et donc de concevoir un modèle d'évaluation spécifique aux contraintes. Cette méthode est plus facile à implémenter que la méthode des fonctions de pénalité car elle ne nécessite pas l'étalonnage des coefficients de pénalité.

Pour cette seconde méthode (2), nous avons réutilisé les fonctions qui permettent de calculer l'aire totale des surfaces de lames entrant en collision, et l'amplitude des mouvements de la courbe. La fonction objectif à minimiser est définie comme l'agrégation de ces deux fonctions. Il aurait été possible d'ajouter deux fonctions au lieu d'une, une pour les collisions et une pour la courbure de la courbure, mais pour une question de lisibilité des résultats nous

avons préféré agréger ces deux critères. L'architecture du modèle général est présentée en Figure 224.

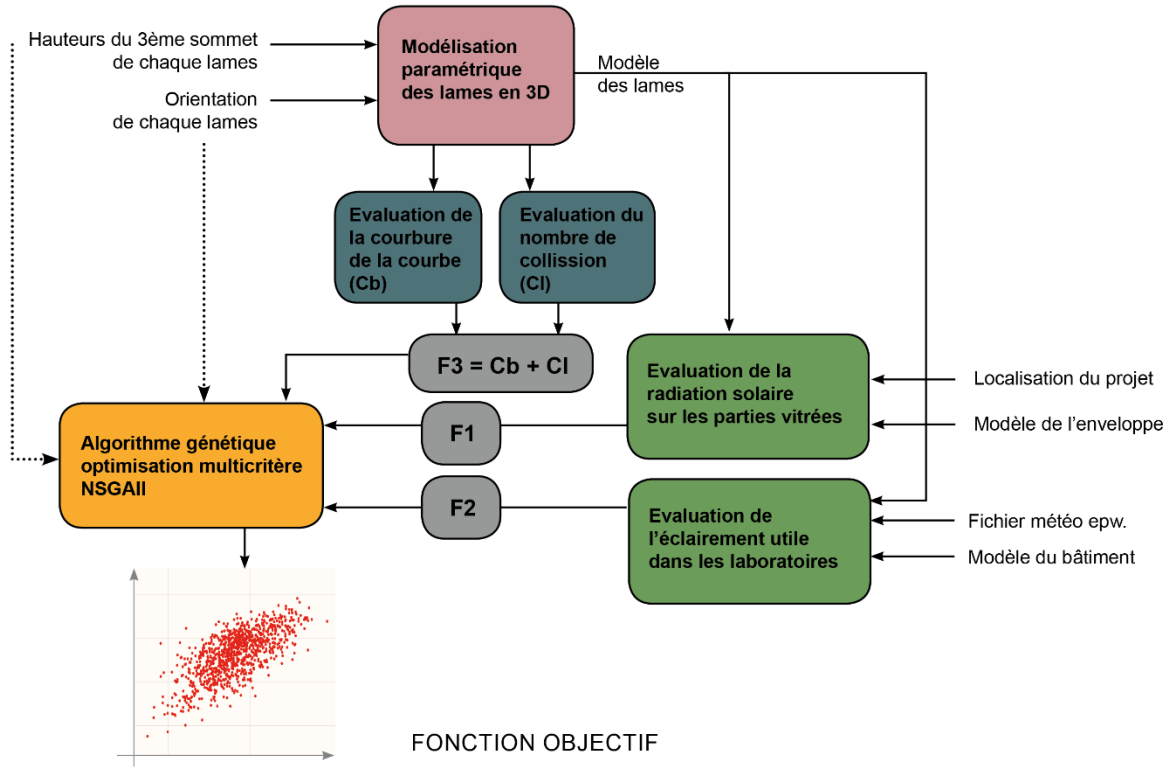


Figure 224 : Architecture du modèle général avec la méthode de l'ajout d'une fonction objectif appliquée au cas d'étude des laboratoires de l'Institut Faire Face

Constraint-NSGAII

Il existe de nombreuses autres méthodes d'intégration des contraintes facilement réplicables. A la différence des deux méthodes que nous venons de présenter, la plupart des méthodes sont intégrées dans les algorithmes, nous parlons de « méthode interne ».
D'après Michalewicz and Schoenauer (1996), il en existe 4 catégories de méthode :

(1) Les méthodes basées sur la préservation de la faisabilité des solutions,
(2) Les méthodes basées sur les fonctions de pénalité,
(3) Les méthodes qui font la distinction entre les solutions faisables et infaisables, et
(4) Et les méthodes hybrides.

Parmi ces méthodes internes on distingue la méthode « Constraint-NSGAII »(C-NSGAII) (Deb, 2000). Il s'agit d'une version de NSGAII qui intègre une méthode de gestion des contraintes basée sur une approche par fonction de pénalité mais qui n'implique pas d'utiliser

de coefficient. Cette méthode est un mélange entre les catégories 2 et 3. Elle implique de mesurer le degré de violation des contraintes pour pouvoir comparer deux solutions non faisables. L'intégration de la contrainte a lieu à l'étape de sélection par tournoi, lorsque deux solutions sont comparées. Les étapes suivantes sont alors appliquées :

(1) Une solution faisable est toujours préférée à une solution infaisable.

(2) Si les deux solutions sont faisables, la sélection se fait selon les fonctions objectifs avec les règles de NSGAII (meilleurs rangs, et distance d'encombrement) présentées dans la première partie de ce chapitre.

(3) Si les deux solutions sont infaisables, la solution avec le plus petit degré de violation des contraintes est retenue.

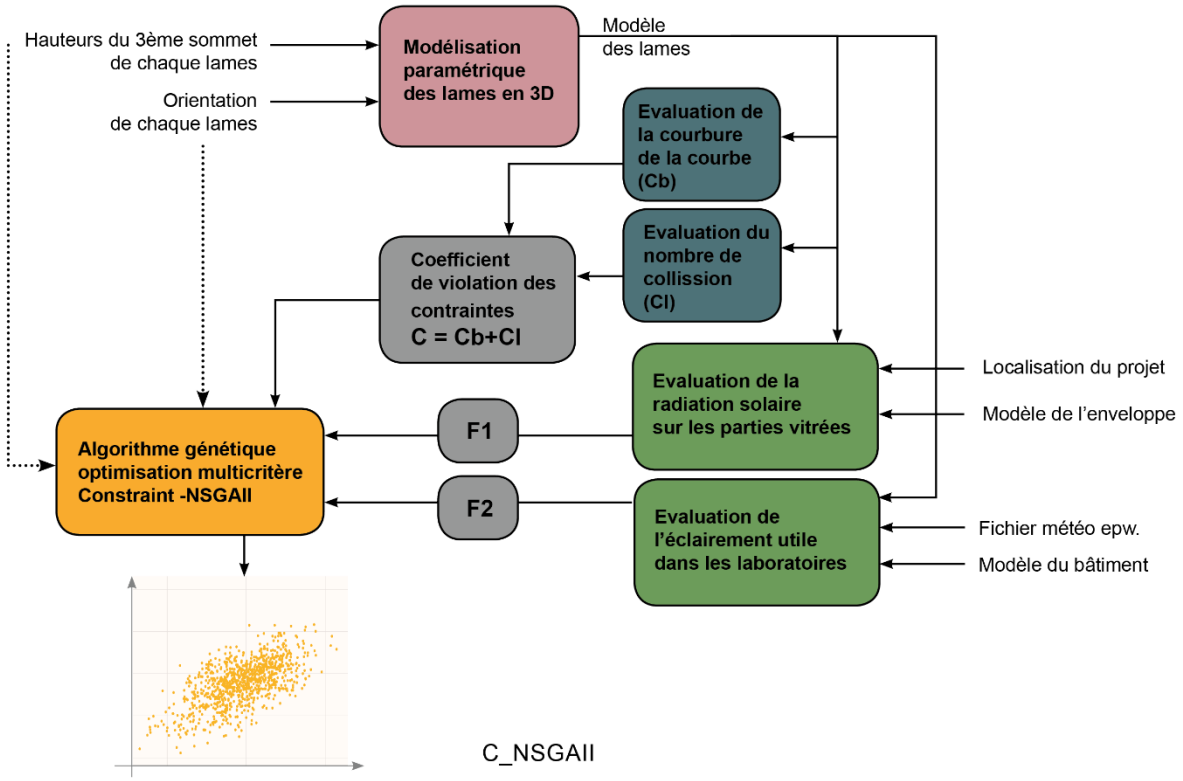


Figure 225 : Architecture du modèle général avec la méthode C- NSGAII appliquée au cas d'étude des laboratoires de l'Institut Faire Face

L'architecture générale du modèle lorsque les contraintes sont traitées avec C-NSGAII est présentée en Figure 225. Le coefficient de violation des contraintes est de nouveau défini avec les deux fonctions qui permettent d'évaluer les collisions et la courbure de la courbe. Finalement, l'implémentation de cette méthode demande exactement le même effort que pour la méthode de l'ajout d'une fonction objectif mais ne peut se faire qu'avec un solveur d'optimisation contenant C-NSGAII, ce qui n'est pas le cas de Wallacei® et d'Opossum®.

Les méthodes ad hoc

Quatre méthodes ad hoc ont été testées. Elles ont toutes nécessité la détermination d'une heuristique adaptée au problème spécifique rencontré.

Hyperparamètres

Une approche, radicale, consiste, quand cela est possible, à reformuler le problème de façon à supprimer les contraintes. Cela implique de repenser toute l'algorithmique du modèle paramétrique original de façon à supprimer les solutions infaisables et réduire la taille de l'espace de solutions.

Parmi les nombreux cas d'étude issus de la littérature scientifique sur les méthodes intégrées présentées au chapitre 1, nous avons identifié des exemples de méthodes où une géométrie intermédiaire, utilisant de nouveaux paramètres, est utilisée pour faire varier la géométrie à optimiser. Ces nouveaux paramètres sont parfois qualifiés d'hyper paramètres. Cette méthode (4) est la seule méthode ad hoc qui ne nécessite pas l'utilisation d'une technique générative.

Negendahl et Nielsen ont utilisé cette méthode pour intégrer une contrainte esthétique dans le cadre de l'optimisation d'une façade pliée pour obtenir un effet de dégradé. Ils utilisent 3 hyper paramètres : un pour l'amplitude des plis, un pour la disposition du pli et le dernier pour un « *blending effect* » (Negendahl & Nielsen, 2015). Y.H. Yi et H. Kim ont utilisés des points de contrôle pour l'optimisation d'îlots urbains afin d'éviter des problèmes de superposition de bâtiments (Y. K. Yi & Kim, 2015). Chatzikonstantinou et ses co-auteurs ont quant à eux utilisé des champs de force reliées à des points pour faire varier l'orientation d'éléments en façade dans une optique d'optimiser un système de protection solaire (Chatzikonstantinou et al., 2019).

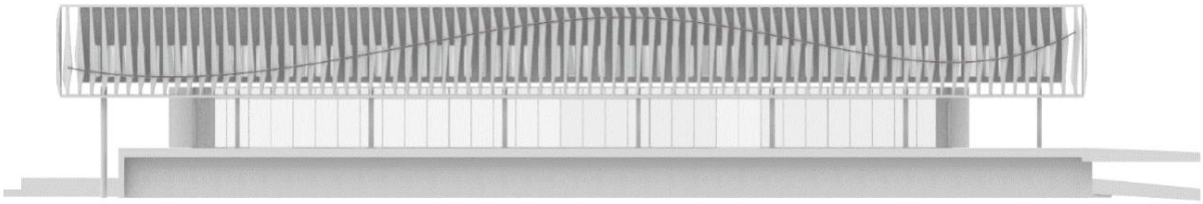


Figure 226 : Courbe de contrôle de la forme des lame en élévation

Nous pouvons appliquer cette méthode à notre problème en utilisant des courbes NURBS comme géométrie intermédiaire. Une première courbe est utilisée en élévation pour contrôler l'emplacement du troisième sommet de chaque lame et éviter un effet en dents de scie

(voir Figure 226). Les hyper-paramètres sont les coordonnées des points qui contrôlent la courbe. Une seconde courbe est utilisée en plan pour contrôler l'orientation des lames et ainsi éviter qu'elle ne rentre en collision. Le modèle paramétrique de base a été modifié ; les 60 variables ont été remplacées par 22, 6 points de contrôle par courbe.

Cette méthode a été relativement facile à mettre en œuvre puisqu'elle ne nécessite pas de connaissance en programmation informatique (autre que la programmation visuelle). L'architecture générale du modèle est présentée en Figure 227. Ce modèle ne génère que des solutions réalisables et réduit fortement l'espace des solutions qui compte désormais 4,9x1020 individus. Mais, en fixant le nombre de points de contrôle, l'éventail des possibilités dans la variété des courbes est réduit, ce qui peut potentiellement exclure de nombreuses solutions efficaces.

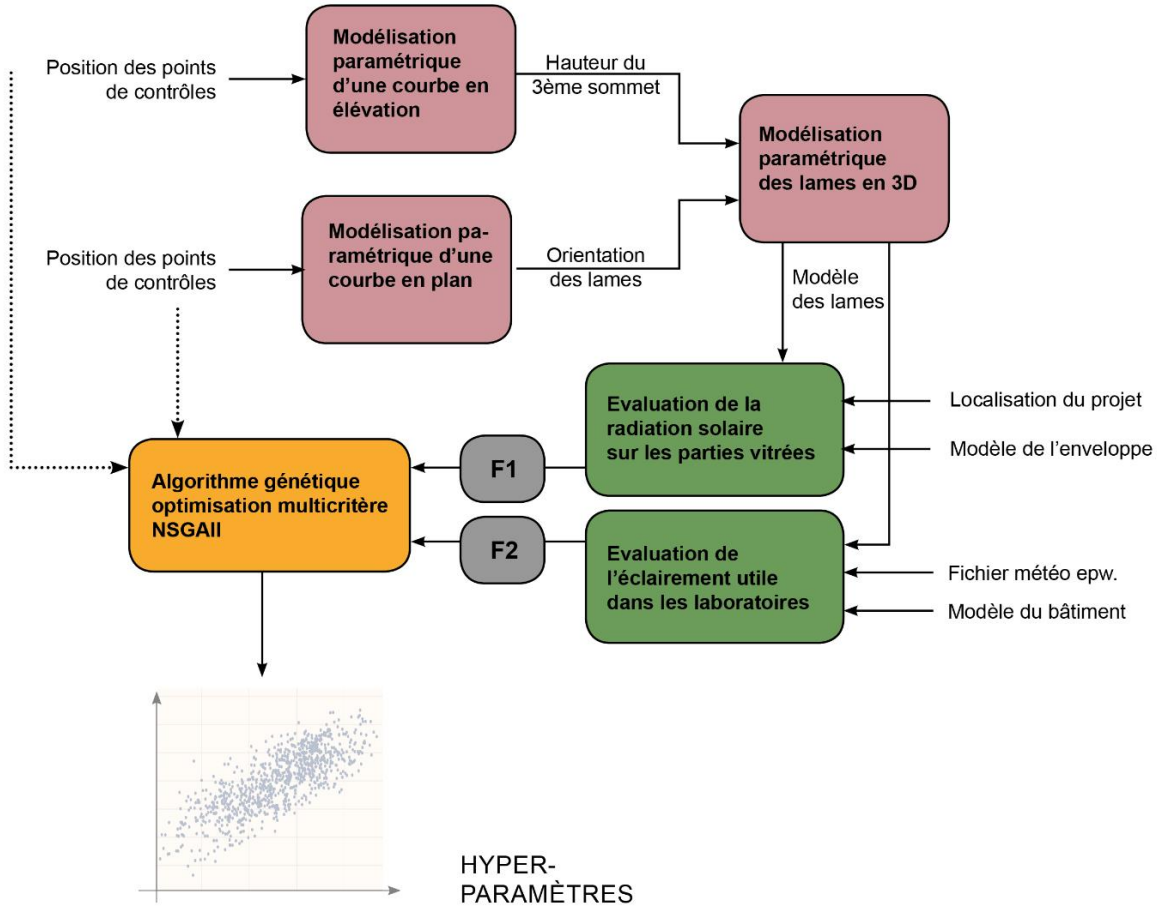


Figure 227 : Architecture du modèle général avec la méthode des hyper-paramètres appliquée au cas d'étude des laboratoires de l'Institut Faire Face

Modèles à base de règles

Il arrive qu'il ne soit pas possible de formuler le problème sous contraintes en utilisant uniquement la programmation linéaire. On peut alors se tourner vers une technique générative qui s'appuie sur l'utilisation de boucles pour générer des formes complexes à partir de règles simples (Caetano et al., 2020). Nous avons présenté dans le chapitre 1 les différentes techniques génératives utilisées en architecture (p83). Ces techniques peuvent être difficiles à implémenter car elles nécessitent souvent de connaitre un langage de programmation informatique.

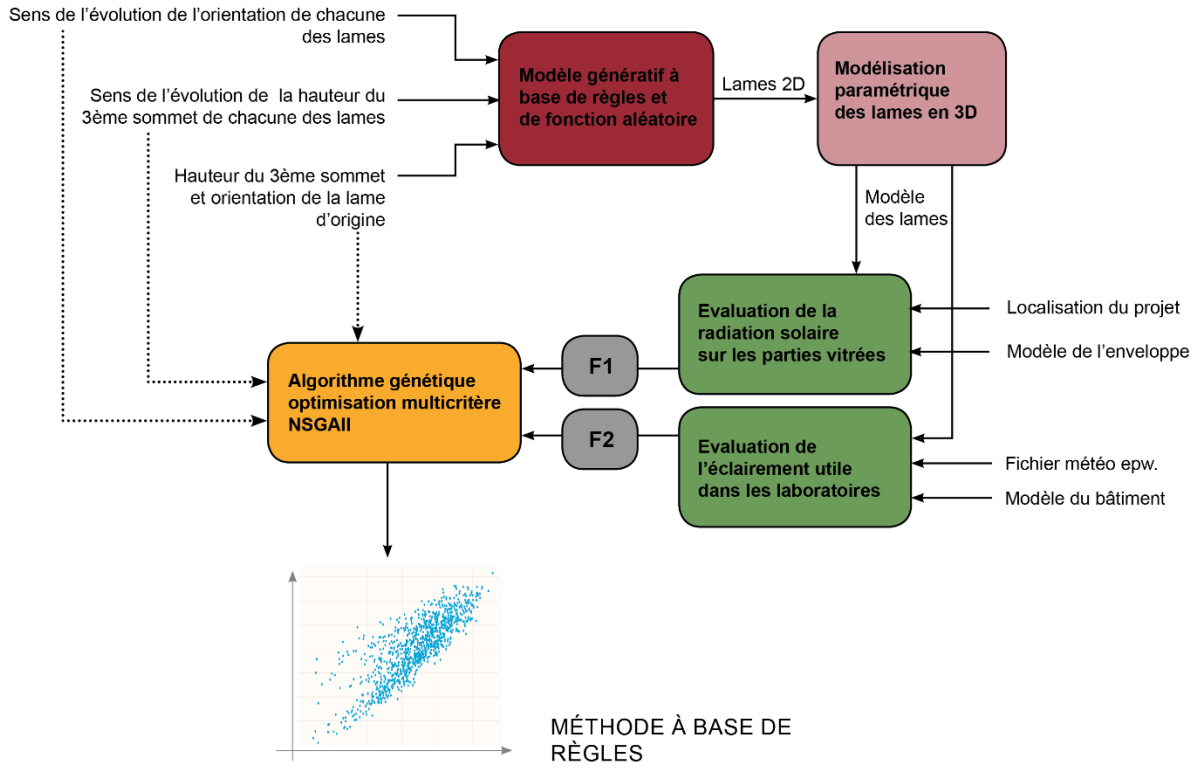


Figure 228 : Architecture du modèle général avec la méthode à base de règle appliquée au cas d'étude des laboratoires de l'Institut Faire Face

Pour notre cas d'étude, nous avons réalisé un programme en python qui permet de générer les 30 lames de l'échantillon de la façade les unes après les autres. La valeur des paramètres d'entrée indique au programme le sens des variations pour la forme et l'orientation de chaque lame en fonction des caractéristiques de la lame précédente. Pour la hauteur du troisième sommet, le paramètre peut prendre 3 valeurs : 0, alors la hauteur du point est identique à la lame précédente, 1, alors le point est un cran plus haut, 2, alors le point est un cran plus bas. Pour l'orientation, il peut prendre 5 valeurs : 0, la lame est orientée comme la lame précédente, 1 elle est orientée avec 60° de moins, 2, avec 3° de moins, 3 avec 60° de plus, 4

avec 30° de plus. L'architecture générale de l'ensemble de la modélisation intégrant un modèle génératif à base de règles est présentée en Figure 228.

Dans certains cas, il est nécessaire de modifier le paramètre, par exemple lorsque l'angle d'orientation atteint sa valeur maximale et ne peut plus varier qu'à la baisse ou reste stable. Pour pallier cela, nous avons intégré au programme une fonction aléatoire pour départager les valeurs possibles restantes. Cette dimension stochastique est à utiliser avec parcimonie car elle peut perturber l'algorithme dans son processus d'évolution.

Fonctions de réparation

En optimisation combinatoire, il semblerait que la méthode la mieux adaptée à la fois en termes de performance et en temps de calcul serait la fonction de réparation (Coello Coello, 2016). Une fonction de réparation consiste à « réparer » une solution non faisable pour la rendre faisable (voir Figure 229). Cela permet d'éviter d'évaluer des solutions qui ne sont pas exploitables pour l'architecte et donc de gagner un temps précieux. Cependant, les fonctions de réparation nécessitent de trouver une heuristique adaptée à chaque problème, ce qui n'est pas toujours possible selon le type de problème rencontré. Néanmoins, il est considéré que sur les problèmes d'optimisation combinatoire (c'est le cas pour les problèmes rencontrés dans la pratique en Architecture), il est souvent facile de définir un algorithme de réparation (Coello Coello, 2002).

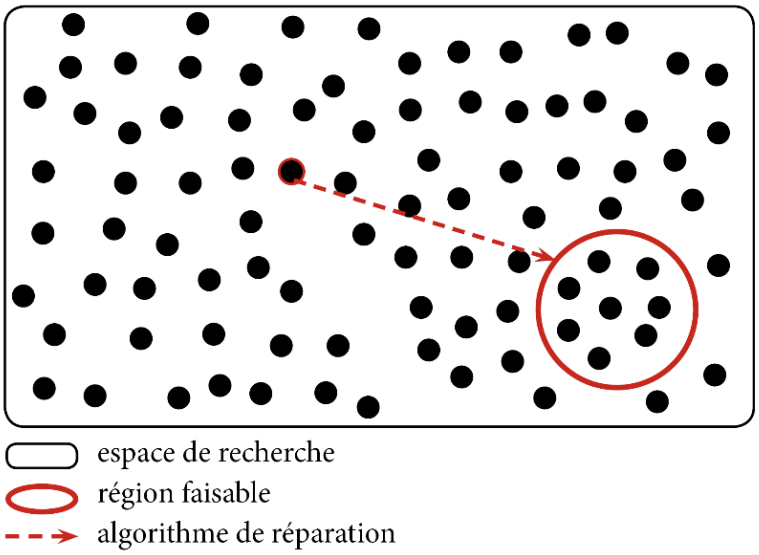


Figure 229 : Principe d'une fonction de réparation

Le principal reproche fait à cette méthode est qu'elle est problème-spécifique. Il est nécessaire de déterminer une nouvelle heuristique pour chaque nouveau problème. Salcedo-Sanz a identifié et classifié des types d'algorithmes de réparation issus de la littérature, mais il s'agit d'applications très éloignées de l'architecture (Salcedo-Sanz, 2009). Parfois, la réparation n'est pas évidente et l'algorithme devient complexe et implique un coup en temps de calcul important. Enfin, il peut aussi grandement transformer l'aspect des solutions d'une génération à l'autre ce qui peut perturber le processus d'évolution (Smith & Fogarty, 1997). Il existe deux manières d'utiliser une fonction de réparation avec un algorithme génétique :

(1) La méthode Baldwinienne où les gènes utilisés pour le croisement génétique correspond à ceux des solutions originales créées par le modèle paramétrique

(2) La méthode Lamarckienne où les gènes originaux sont, en partie ou en totalité, remplacés par ceux des solutions réparées (Salcedo-Sanz, 2009). Avec cette approche, il faut déterminer le bon taux de remplacement du génome, celui-ci varie selon les auteurs et le type de problème (Coello Coello, 2016). David Orvosh a montré que le taux de remplacement optimal pour des problèmes d'optimisation combinatoire était de 5 % (Orvosh & Davis, 1993).

Une fonction de réparation est un algorithme qui peut prendre la forme d'un modèle paramétrique, d'un modèle génératif spécifique comme un MBA, un AC ou un modèle génératif simple, similaire à celui utilisé avec la méthode précédente, de type modèle à base de règles.

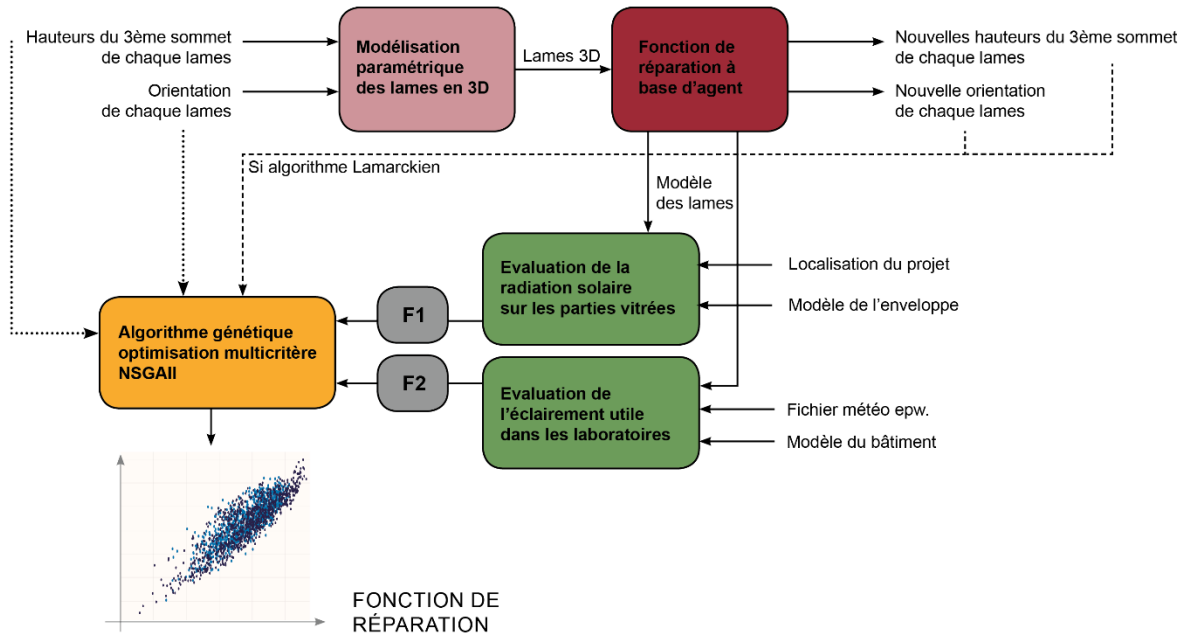


Figure 230 : Architecture du modèle général avec la méthode des fonctions de réparation appliquée au cas d'étude des laboratoires de l'Institut Faire Face

Nous avons appliqué cette méthode des fonctions de réparation sur notre cas d'étude en utilisant le modèle paramétrique simple utilisé avec les méthodes réplicables qui comprend 60 variables. Les solutions générées par ce modèle sont ensuite réparées à l'aide d'un modèle élémentaire à base d'agents pouvant être qualifié de système d'auto-organisation. L'architecture du modèle général avec la méthode des fonctions de réparation est présentée en Figure 230.

L'algorithme de réparation fonctionne ainsi : chaque lame constitue un agent doté de deux attributs (sa forme et son orientation) qui varient en fonction des attributs des agents voisins. A chaque itération, un algorithme interroge l'angle de l'orientation de la lame, puis un second interroge la hauteur du troisième sommet (voir Figure 231). Le programme réalise une boucle qui s'arrête lorsque chaque agent est satisfait (lorsqu'il respecte les contraintes de collisions et d'évitement de l'effet en dent de scie). Le système a alors atteint un état d'équilibre.

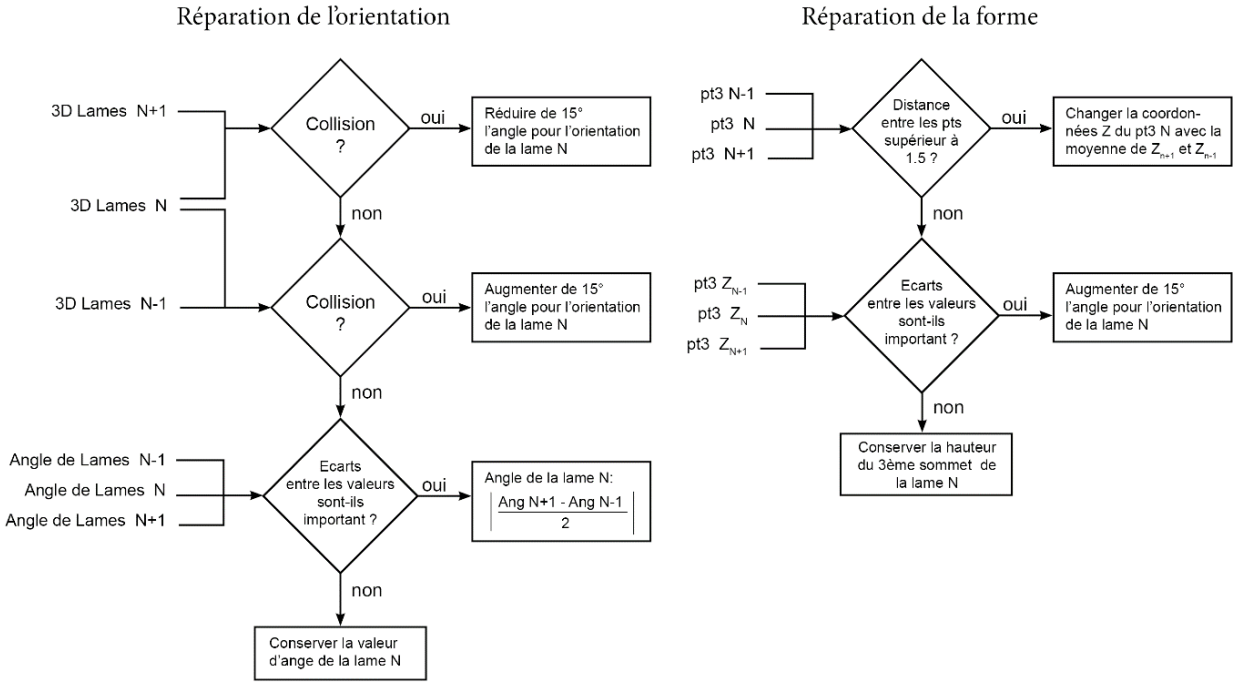


Figure 231 : Algorithmes de réparation utilisé sur le cas d'étude des laboratoires de l'Institut Faire Face

Cette méthode est testée une première fois avec l'approche baldwienne, puis dans un second temps avec une approche lamarckienne où pour 25 % des solutions réparées les gènes ont été remplacées. La première approche peut être effectuée avec n'importe quel solver d'optimisation. Pour tester l'approche lamarckienne, nous avons implémenté NSGAII dans Grasshopper® en développant notre propre solveur d'optimisation en Python, en utilisant la

librairie d'optimisation multiobjectifs Pymoo (Blank & Deb, 2020). L'architecture général du solveur et ses échanges avec Grasshopper® sont décrits en Figure 232.

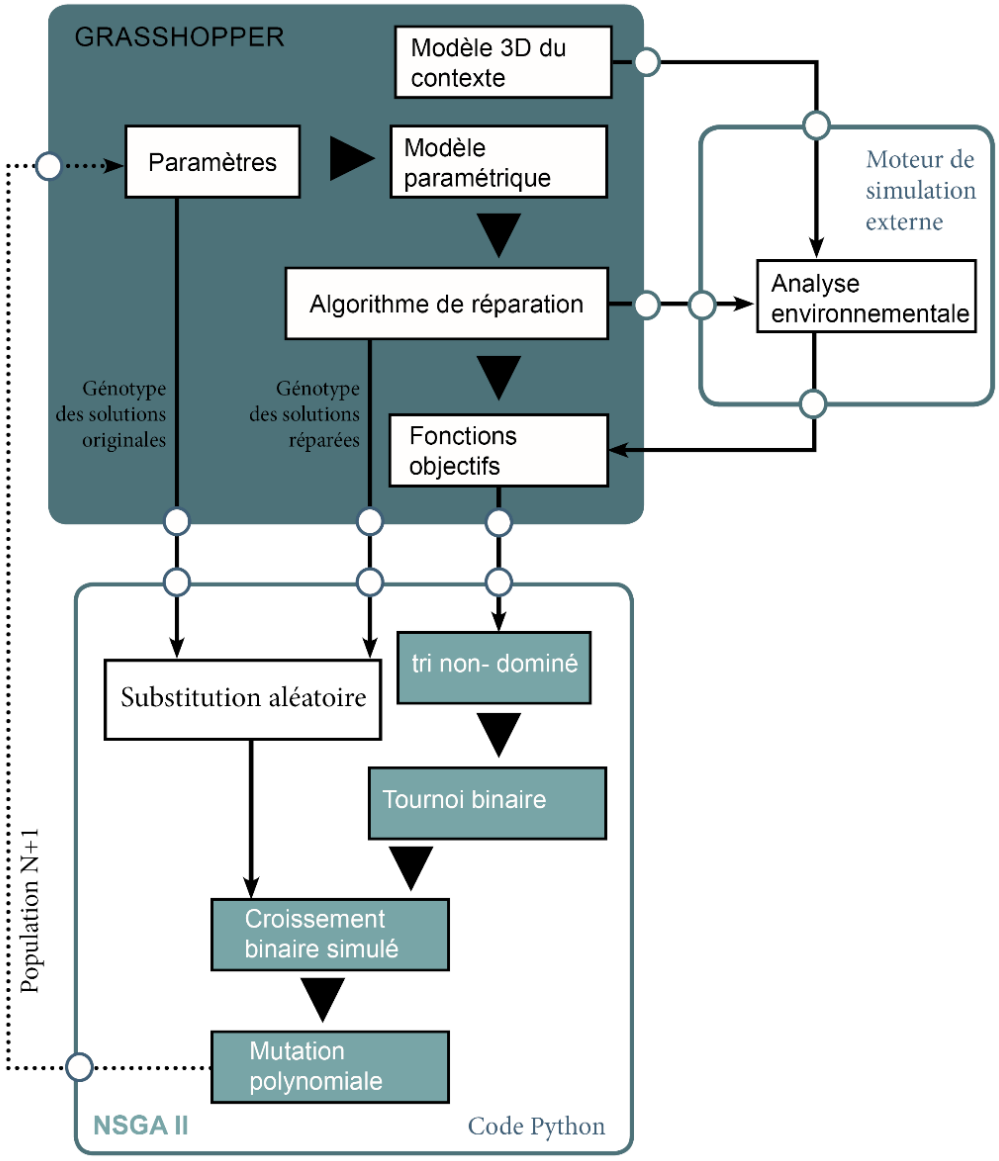


Figure 232 : Architecture du solver d'optimisation

3.2.3. Résultats de l'étude comparative

Après avoir modéliser les 7 approches avec les différentes méthodes de gestion des contraintes et la méthode aléatoire pour le groupe contrôle, nous avons lancé les calculs les uns après les autres avec environ 10 heures de calculs par méthodes. Nous présentons dans cette dernière sous-partie les résultats des méthodes réplicables dans un premier temps, puis ceux des méthodes ad hoc.

Résultats des méthodes réplicables

Fonction de pénalité

La méthode des fonctions de pénalités statique s'est révélée être très peu performante sur ce problème d'optimisation. Les résultats ne sont pas significativement différents du groupe contrôle représenté par les points noirs sur la Figure 233. Ainsi, les solutions générées avec cette méthode ne sont pas meilleures que des solutions générées au hasard. Le front de Pareto résultant de cette optimisation compte 10 solutions. Aucune d'entre elle n'appartient au front de Pareto global (en rouge sur la figure) des méthodes réplicables.

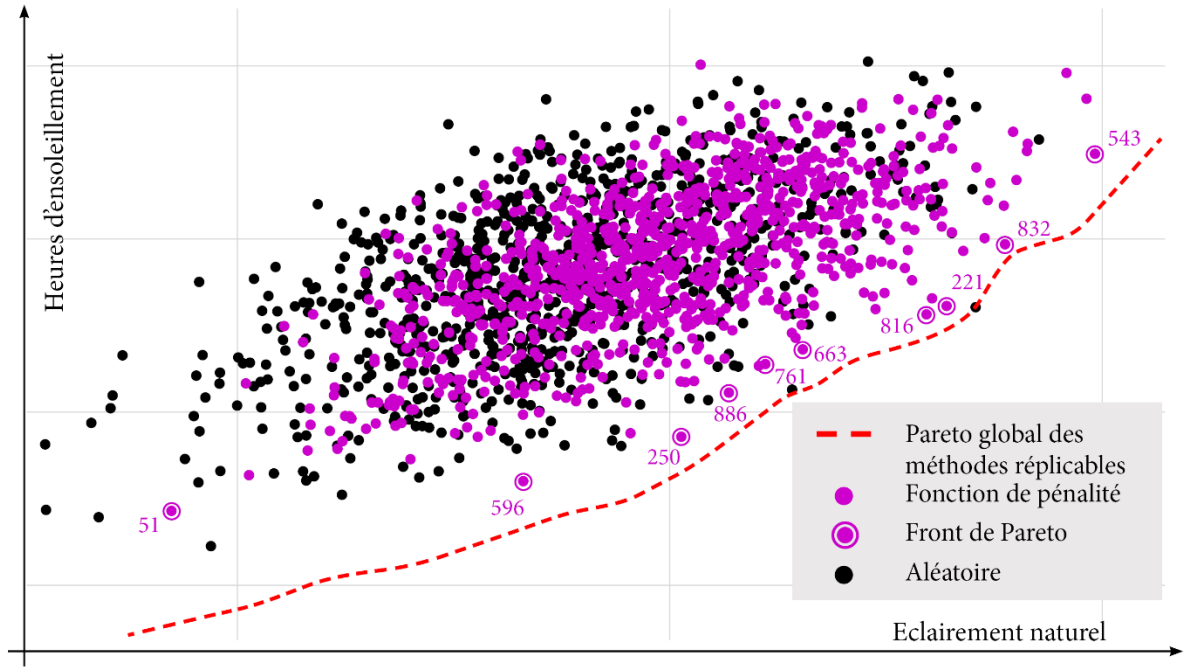


Figure 233 : Ensemble de solutions générée avec la méthode des pénalités statiques

Si l'on ne s'intéresse qu'à la question des contraintes, le résultat est encore plus décevant. Aucune des 1 000 solutions générées avec cette méthode ne respecte les contraintes. On observe sur les solutions du Front de Pareto représentées en Figure 234 et en Figure 235 que si certaines solutions arrivent à réduire un peu l'effet en dents de scie comme la solution 250 ou 816, le problème des collisions persiste sur toutes les solutions. De manière générale, les solutions générées par cette méthode ne sont pas esthétiquement satisfaisantes.

Enfin, nous avons utilisé les résultats de l'échantillon de contrôle pour étalonner les coefficients de pénalité, mais dans la pratique, cette méthode demande des temps de calcul en plus pour constituer un premier échantillon de solutions pour pouvoir réaliser cet étalonnage.

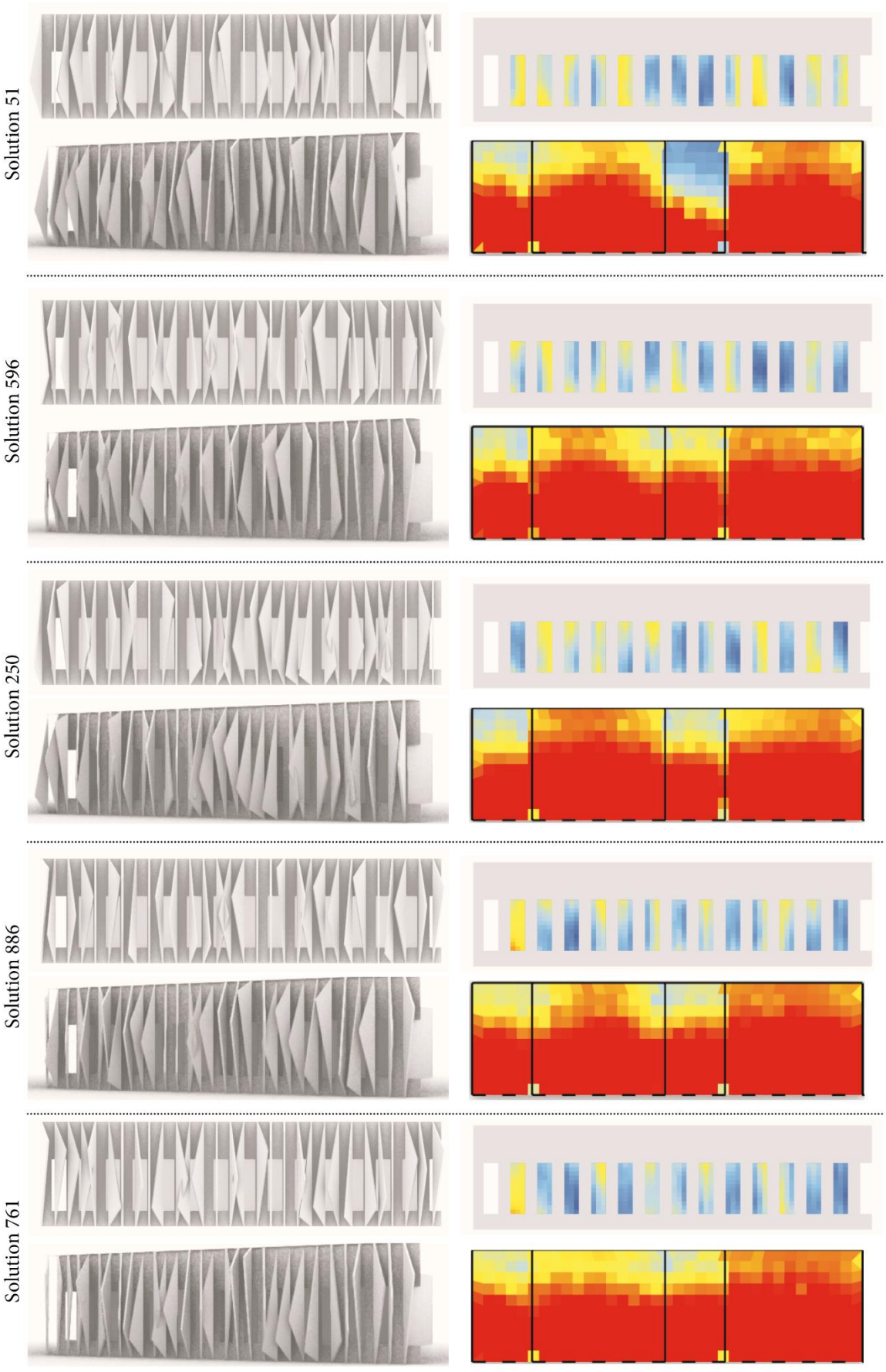


Figure 234 : Visualisation des solutions sur le front de Pareto de la méthode 1

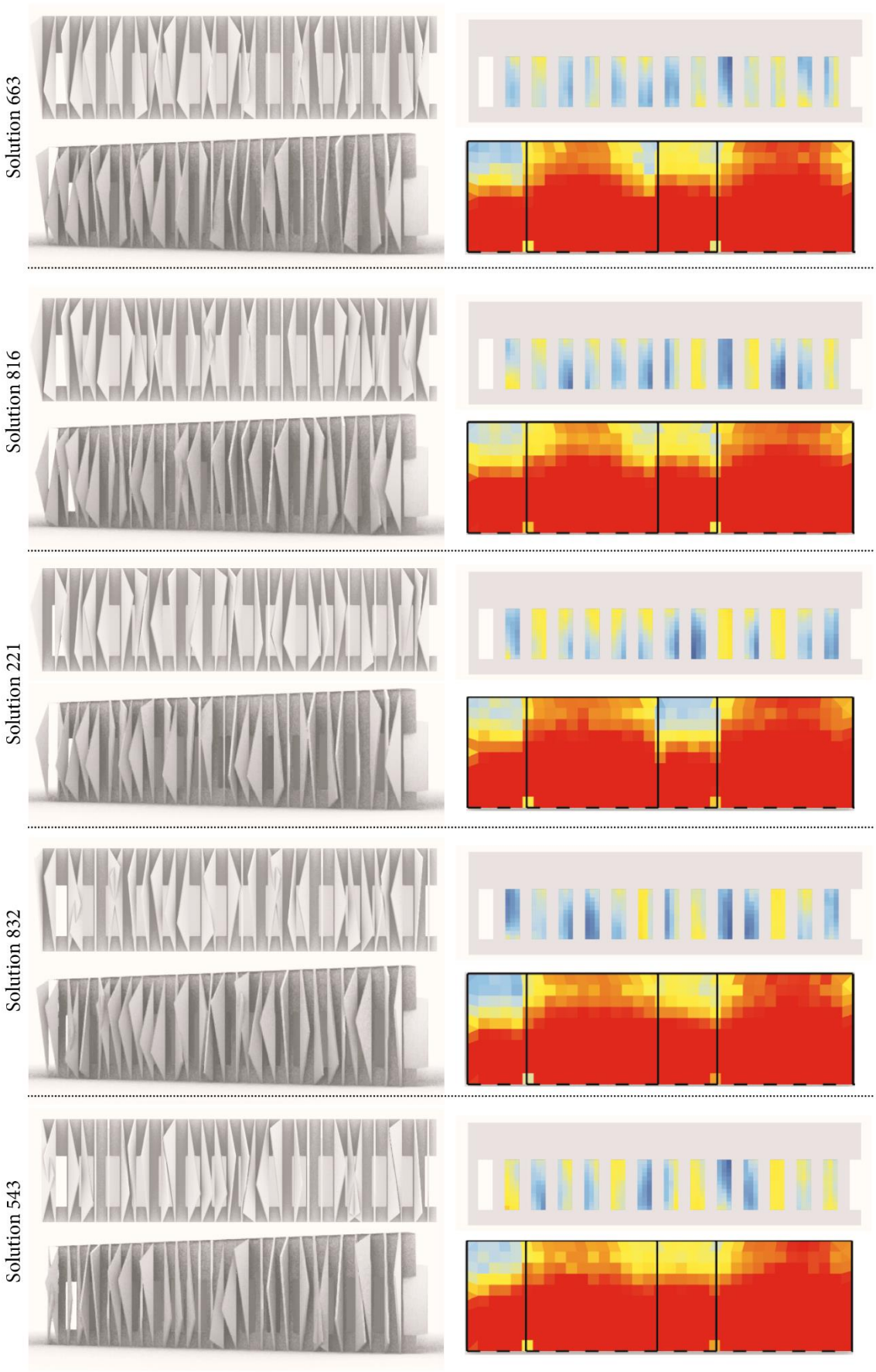


Figure 235 : Visualisation des solutions sur le front de Pareto de la méthode 1

Ajout d'une fonction objectif

Cette seconde méthode ne s'est pas révélée beaucoup plus efficace que la première. Comme le montre la Figure 236, quelques solutions, mais encore trop peu se distinguent de la zone où se trouvent les solutions du groupe contrôle. Le front de Pareto résultant de cette optimisation compte 11 solutions, dont 6 appartiennent au front de Pareto global des méthodes réplicables. Ainsi, concernant les critères environnementaux, cette méthode permet de générer de meilleurs solutions que la méthode des fonctions de pénalités.

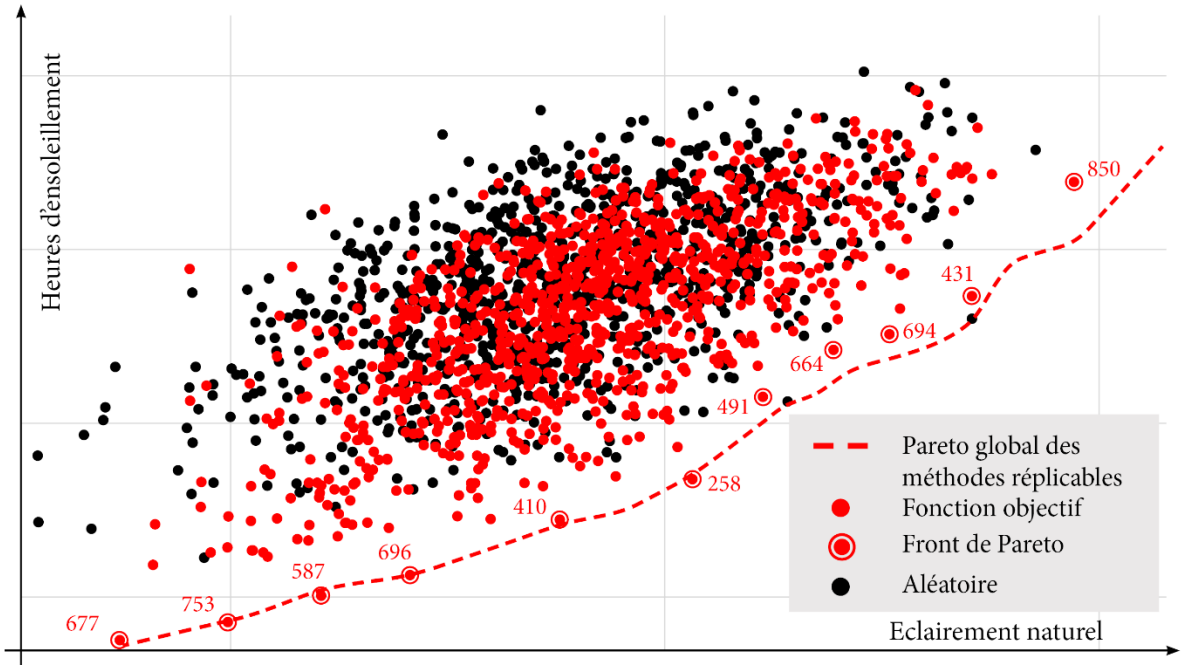


Figure 236 : Ensemble de solutions générée avec la méthode 2

Concernant les contraintes, le constat est le même que pour la précédente méthode. Aucune des 1 000 solutions générées ne respecte les contraintes et aucune solution générée n'est esthétiquement satisfaisante avec cette approche.

Cette technique reste très facile à implémenter et ne demande pas de réaliser des calculs au préalable. Elle peut être utile sur des problèmes où la région faisable est suffisamment large pour que la population de départ contienne quelques solutions faisables. Pour mesurer la taille de la région faisable, il faut générer 1 000 solutions au hasard, puis évaluer uniquement le degré de faisabilité des solutions (car les temps de calculs de ces fonctions est généralement moindre comparé aux simulations environnementales). Dans notre cas, le résultat est inférieur à 0,1 %, ce qui est bien trop peu pour une population de 50 individus.

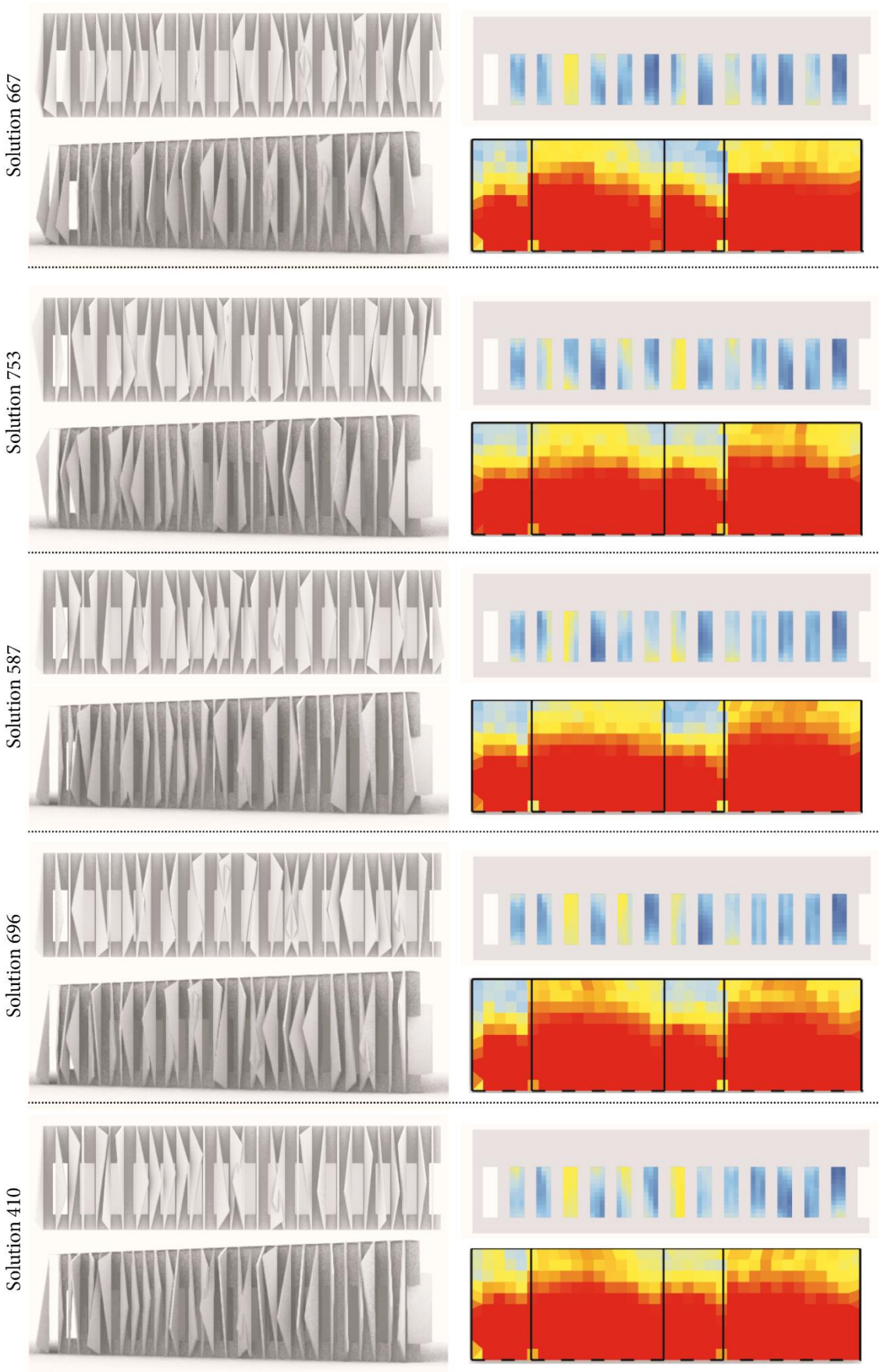


Figure 237 : Visualisation des solutions sur le front de Pareto de la méthode 2

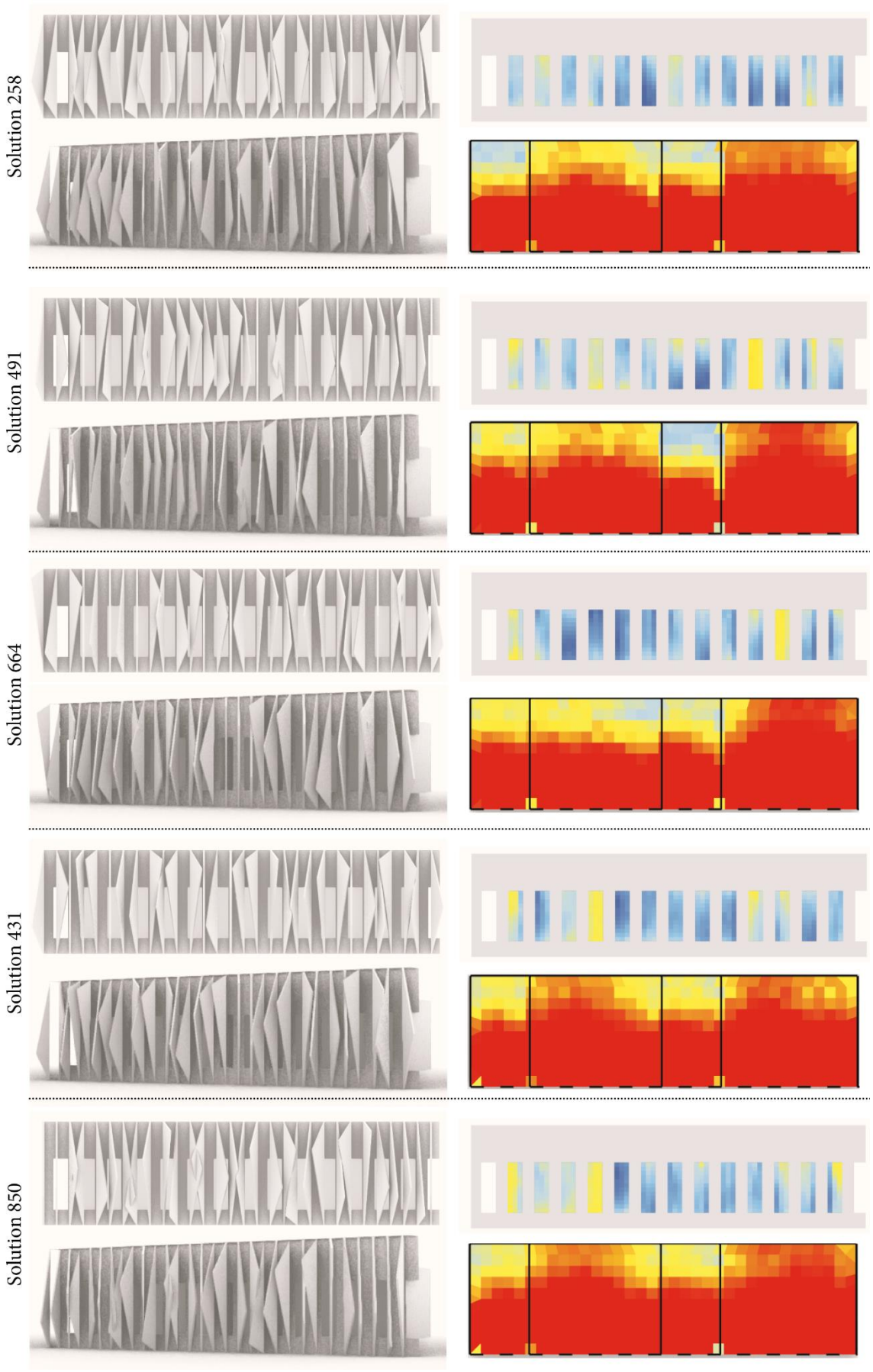


Figure 238 : Visualisation des solutions sur le front de Pareto de la méthode 2

Constraint-NSGAII

La dernière méthode réplicable obtient de meilleurs résultats que les deux premières mais reste aussi décevante compte tenu des objectifs. L'ensemble des solutions trouvées se détache un peu plus de la zone du groupe de contrôle, et le nombre de solutions sur le front de Pareto est plus important qu'avec les deux autres méthodes, avec 16 solutions au total dont 11 qui appartiennent au front de Pareto global des méthodes réplicables. Si elles sont plus nombreuses, elles sont aussi moins diversifiées qu'avec la méthode précédant.

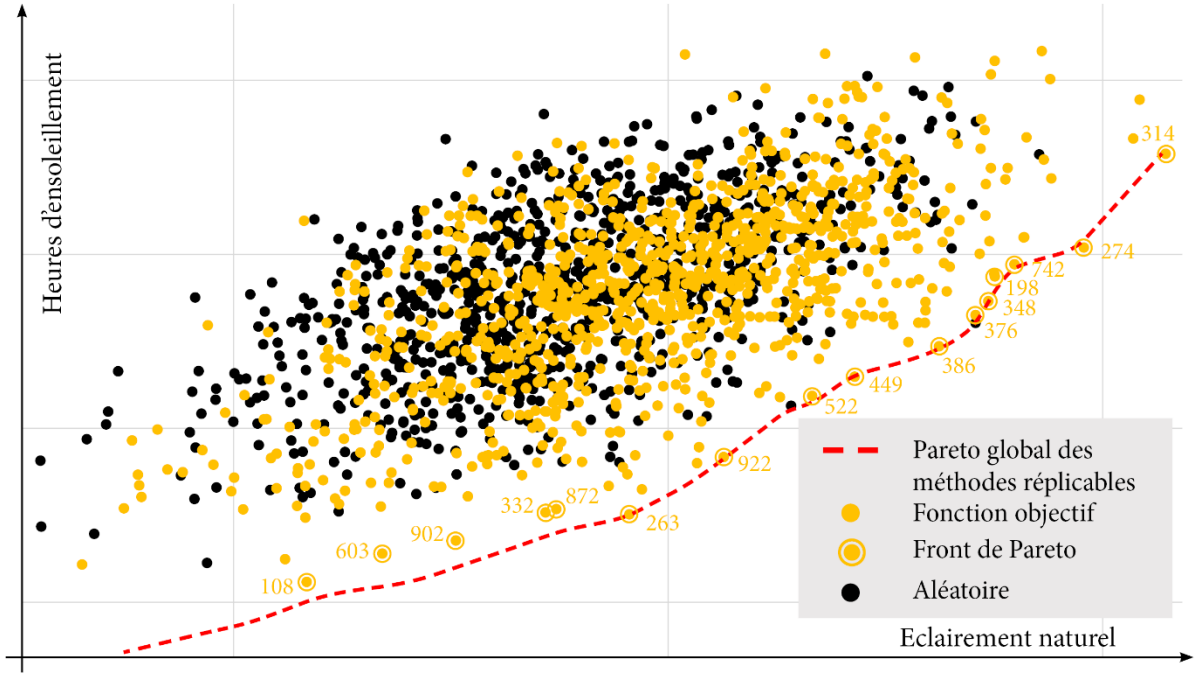


Figure 239 : Ensemble de solutions générée avec la méthode 3

Comme pour les autres méthodes, aucune des 1 000 solutions générées ne parvient à respecter les contraintes, même si on peut observer sur les élévations des solutions issues du front de Pareto présentées en Figure 240, Figure 241 et Figure 242 que certaines solutions comptent peu de lames entrant en collisions, comme la solution 263 et 449. Cependant, après avoir comparé les résultats des deux fonctions évaluant les deux contraintes (collisions et effet en dents de scie), aucunes des trois méthodes n'arrive à générer des solutions avec des scores de violation des contraintes significativement plus faibles que les autres. Et, ici aussi, les formes générées ne correspondent pas aux attentes des architectes en termes de design.

Cette troisième méthode a le mérite d'être aussi simple à implémenter que la méthode (2), et comme cette dernière, elle pourrait être utile sur des problèmes de conception architecturale un peu moins contraints.

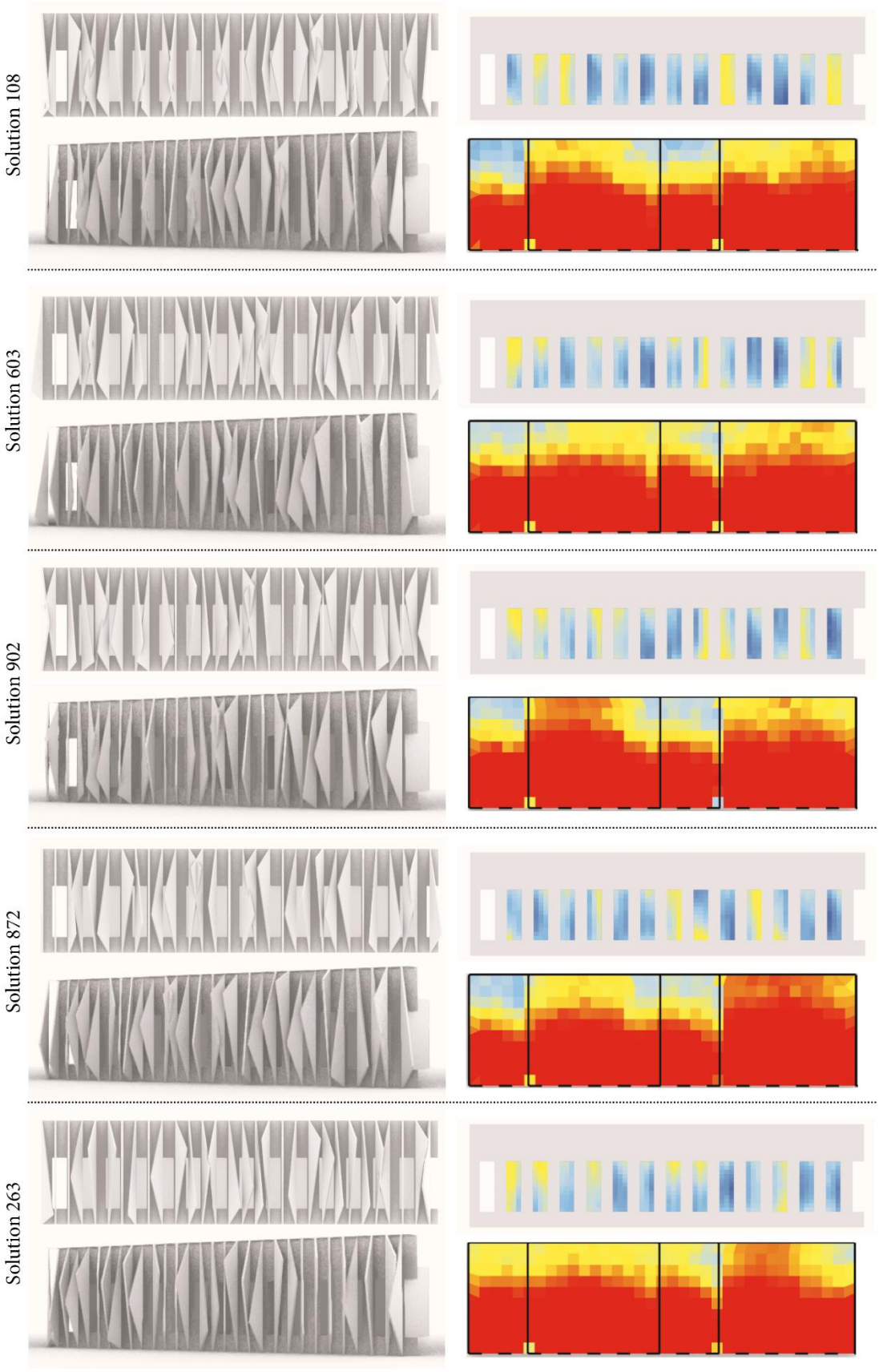


Figure 240 : Visualisation des solutions sur le front de Pareto de la méthode 3

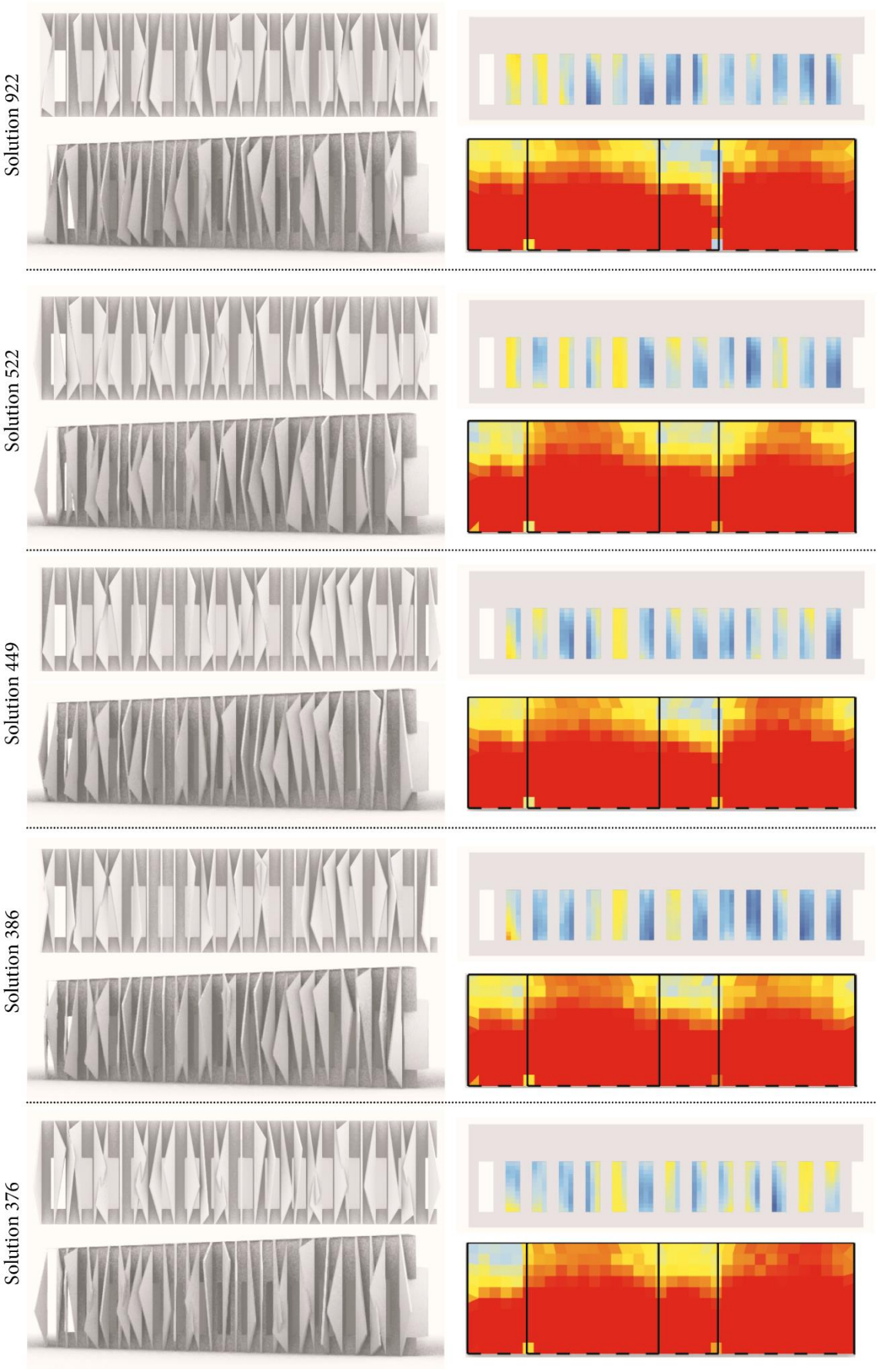


Figure 241 : Visualisation des solutions sur le front de Pareto de la méthode 3

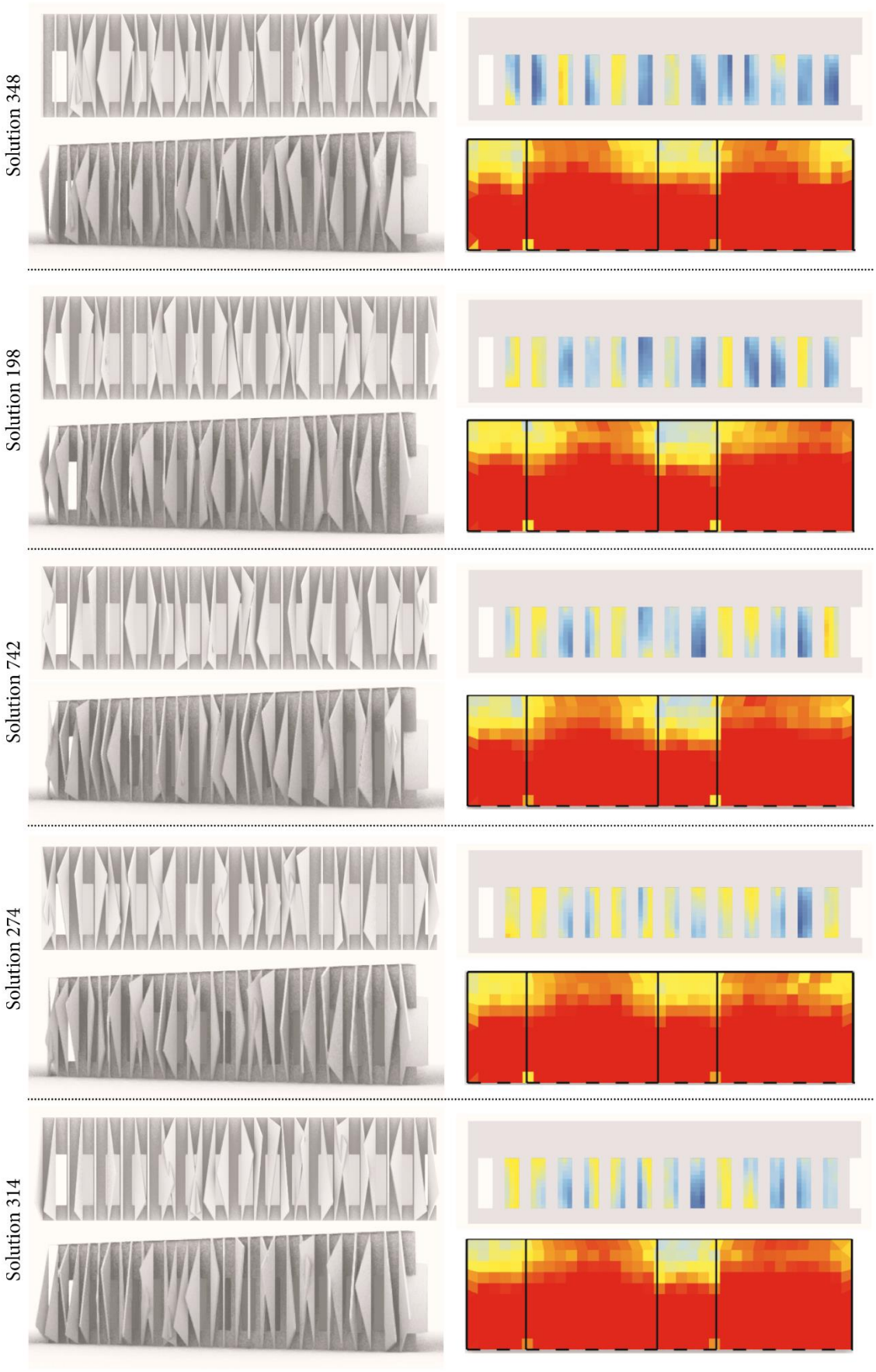


Figure 242 : Visualisation des solutions sur le front de Pareto de la méthode 3

Synthèse sur les méthodes réplicables

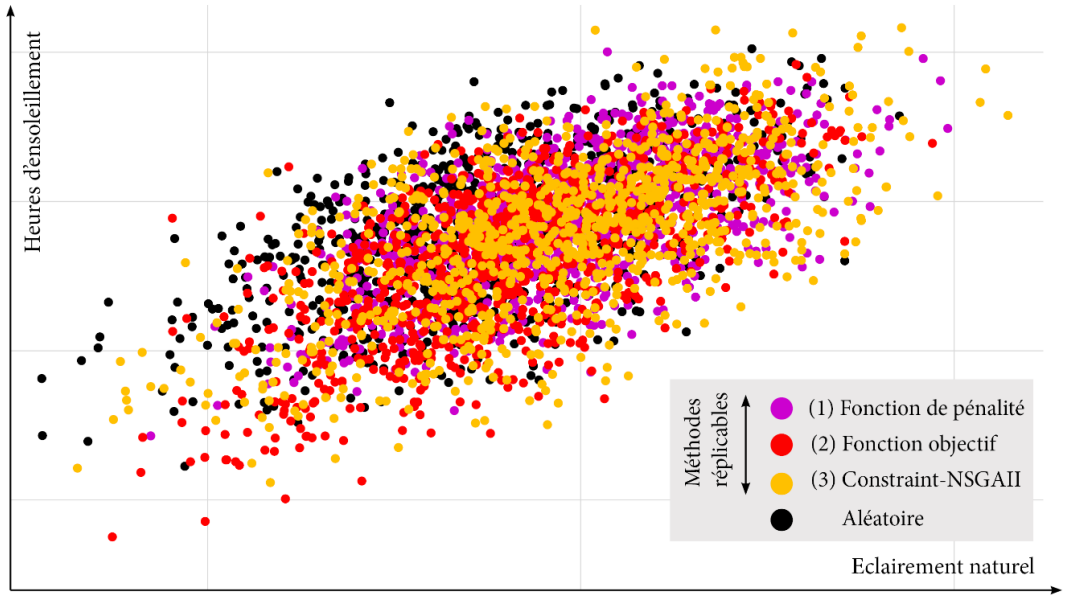


Figure 243 : Ensemble des solutions générées avec les méthodes réplicables

Dans le cas de ce problème de conception architectural, et peut-être aussi pour des problèmes de nature similaire, les méthodes génériques de gestion des contraintes ne parviennent pas à répondre au besoin de générer des solutions faisables. Lorsque le problème est très contraint et que la combinatoire est importante, comme c'est le cas dans cet exemple, NSGAII n'arrive pas avec ces méthodes à faire beaucoup mieux qu'une génération aléatoire.

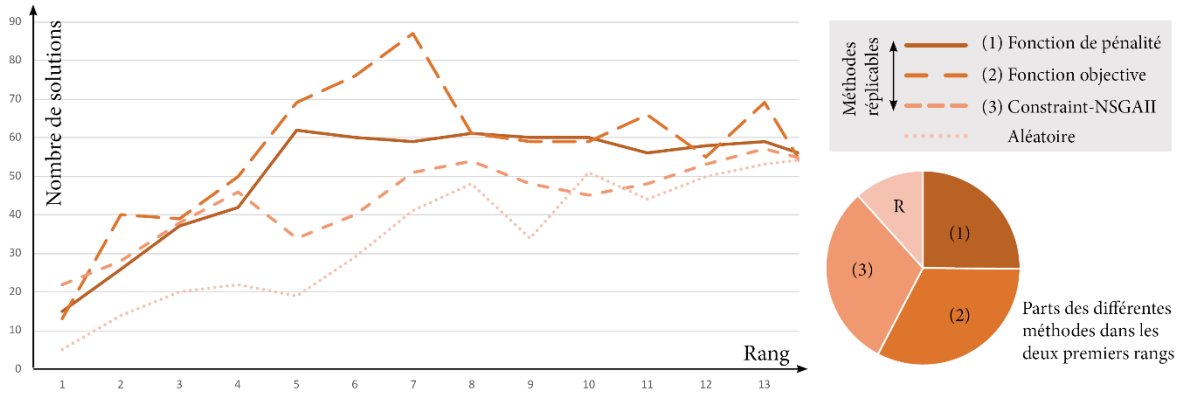


Figure 244 : Tri non-dominé de l'ensemble des solutions générées avec les méthodes réplicables

Afin de mieux comparer les différentes méthodes, nous avons considéré toutes les solutions générées par les trois méthodes et celles du groupe de contrôle, soit 4 000 individus au total. Nous avons ensuite appliqué la méthode de tri non-dominé à ce nouvel ensemble en s'appuyant sur les valeurs des fonctions objectifs. Les résultats sont présentés en Figure 244, le

rang 1. A la différence du front de Pareto global représenté en rouge pointillé dans les Figure 233, Figure 236 et Figure 239, trois critères ont été pris en compte pour ce tri : l'éclairement, les heures d'ensoleillement et les contraintes. Les résultats montrent que si la méthode (3) est le plus gros contributeur au front de Pareto. La méthode (2) est le plus mauvais contributeur au front de Pareto, mais aussi le plus gros contributeur pour les 6 meilleurs rangs qui suivent le front de Pareto. La méthode (1) est ici le plus mauvais contributeur en moyenne sur les 4 premiers rangs. Dans le cadre de cet exemple seulement, nous observons que la méthode basée sur une fonction de pénalité est moins efficace. Cela peut s'expliquer par la difficulté de trouver des coefficients de pénalité pertinents.

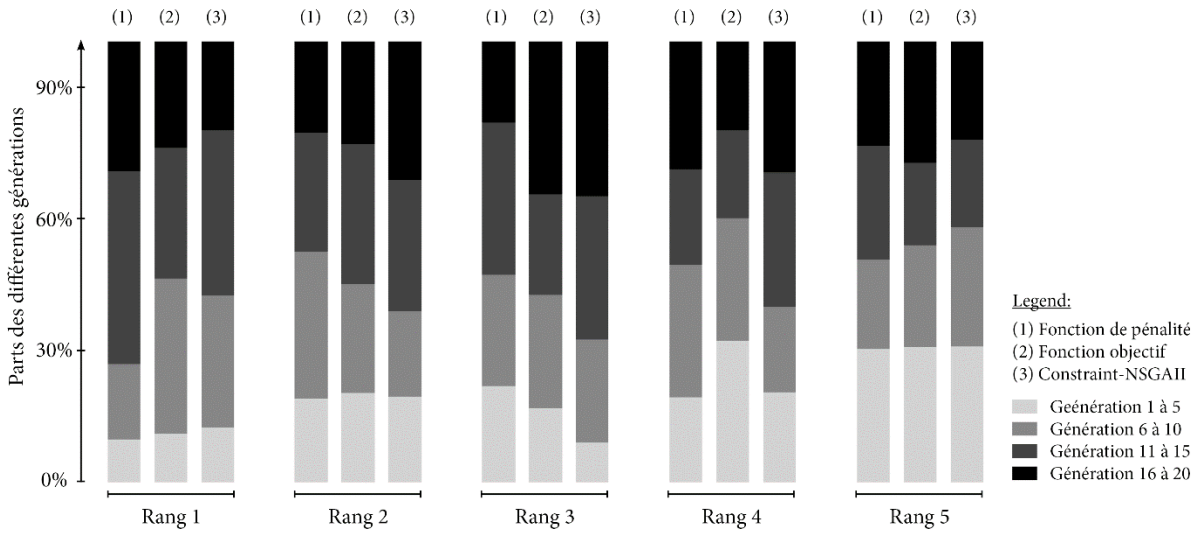


Figure 245 : Parts des différentes générations en fonction du rang pour les méthodes réplicables

Enfin, nous avons cherché à comprendre si le nombre de générations avait un impact sur la performance de ces méthodes. Ainsi, pour comparer la vitesse de convergence de chaque algorithme, nous avons réalisé une analyse, dont les résultats sont présentés en Figure 245, pour identifier la génération à partir de laquelle les meilleures solutions ont été obtenues. Pour chaque méthode, un tri des solutions non dominées a été effectué. Pour les trois méthodes, les premiers rangs présentent une part plus importante d'individus issus des dernières générations, la vitesse de convergence de NSGAII augmente au départ avant de ralentir au plus tard à partir de la génération 15. Les résultats obtenus ne permettent pas d'identifier une méthode qui générerait ses meilleures solutions de façon significativement plus rapide.

En résumé, même s'il n'est pas possible de conclure généralement sur la base d'une seule étude de cas, notre résultat tend à confirmer que les méthodes réplicables de gestion des contraintes peuvent se révéler inefficaces sur des problèmes de conception architecturale issus de la pratique.

Résultats des méthodes ad hoc

Hyper-paramètres

Cette méthode permet d'obtenir de bien meilleurs résultats que ceux du groupe contrôle et les méthodes génériques. Le front de Pareto résultant de cette expérimentation compte bien plus de solutions que pour les 3 méthodes précédentes, à savoir 21 individus. Comme le montre le graphique en Figure 246, tous dépassent les individus du front de Pareto global des méthodes réplicables. Ces solutions sont aussi bien plus diversifiées, les nouveaux extremums sont très éloignés de ceux des précédentes expérimentations. Cependant, les résultats sont bien loin de ceux du front de Pareto global de l'ensemble des méthodes ad hoc.

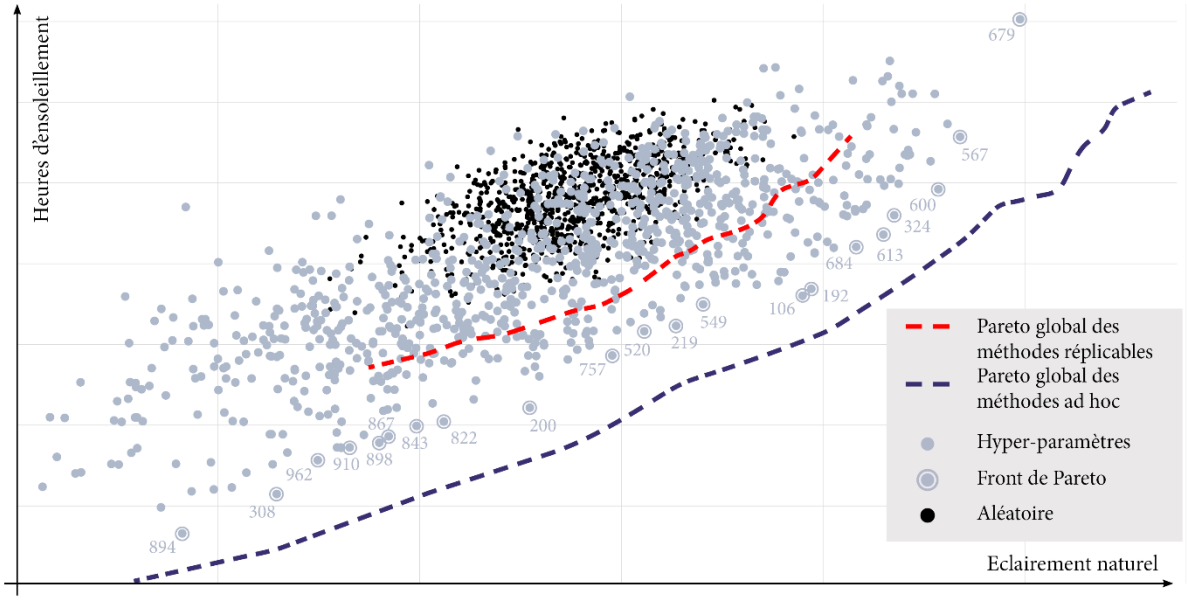


Figure 246 : Ensemble de solutions générée avec la méthode 4

L'heuristique a très bien fonctionnée puisque les 1 000 solutions générées respectaient les contraintes de collisions et ont permis d'éviter l'effet en dents de scie en générant des lames aux formes toutes différentes et avec des orientations toutes distinctes. En termes de design, ces solutions sont aussi beaucoup plus proches des attentes de l'équipe d'architecte que les solutions générées avec les méthodes génériques (voir Figure 247, Figure 248, Figure 249 et Figure 250).

Cependant, cette heuristique comporte un défaut. Certaines dispositions des points de contrôle créent des courbes pouvant sortir du cadre de l'élévation, entrainant des séries de lames aux formes identiques non souhaitées (lorsque le troisième sommet est à hauteur minimale ou maximale), comme pour les solutions 962, 910 ou 843.

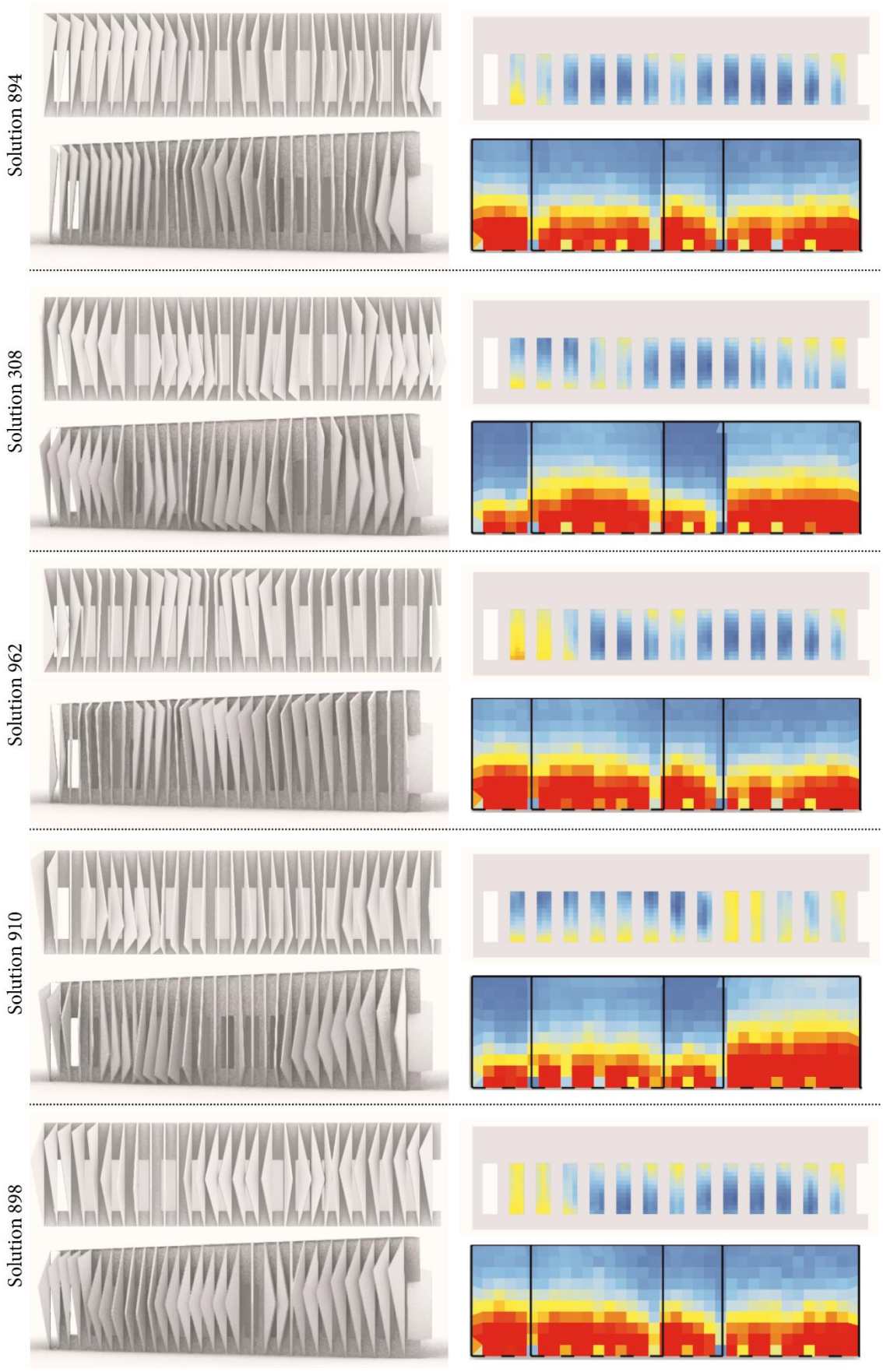


Figure 247 : Visualisation des solutions sur le front de Pareto de la méthode 4

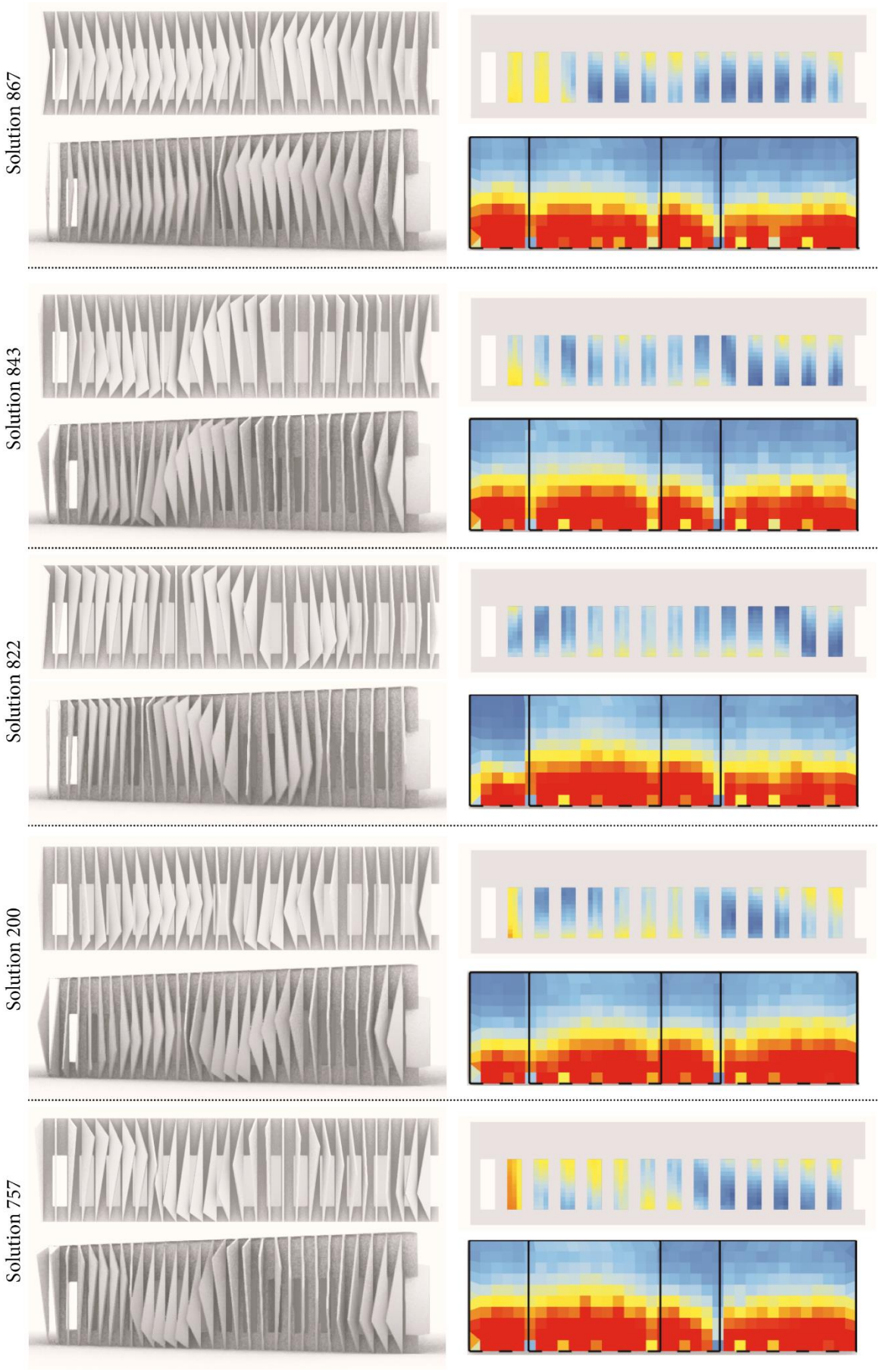


Figure 248 : Visualisation des solutions sur le front de Pareto de la méthode 4

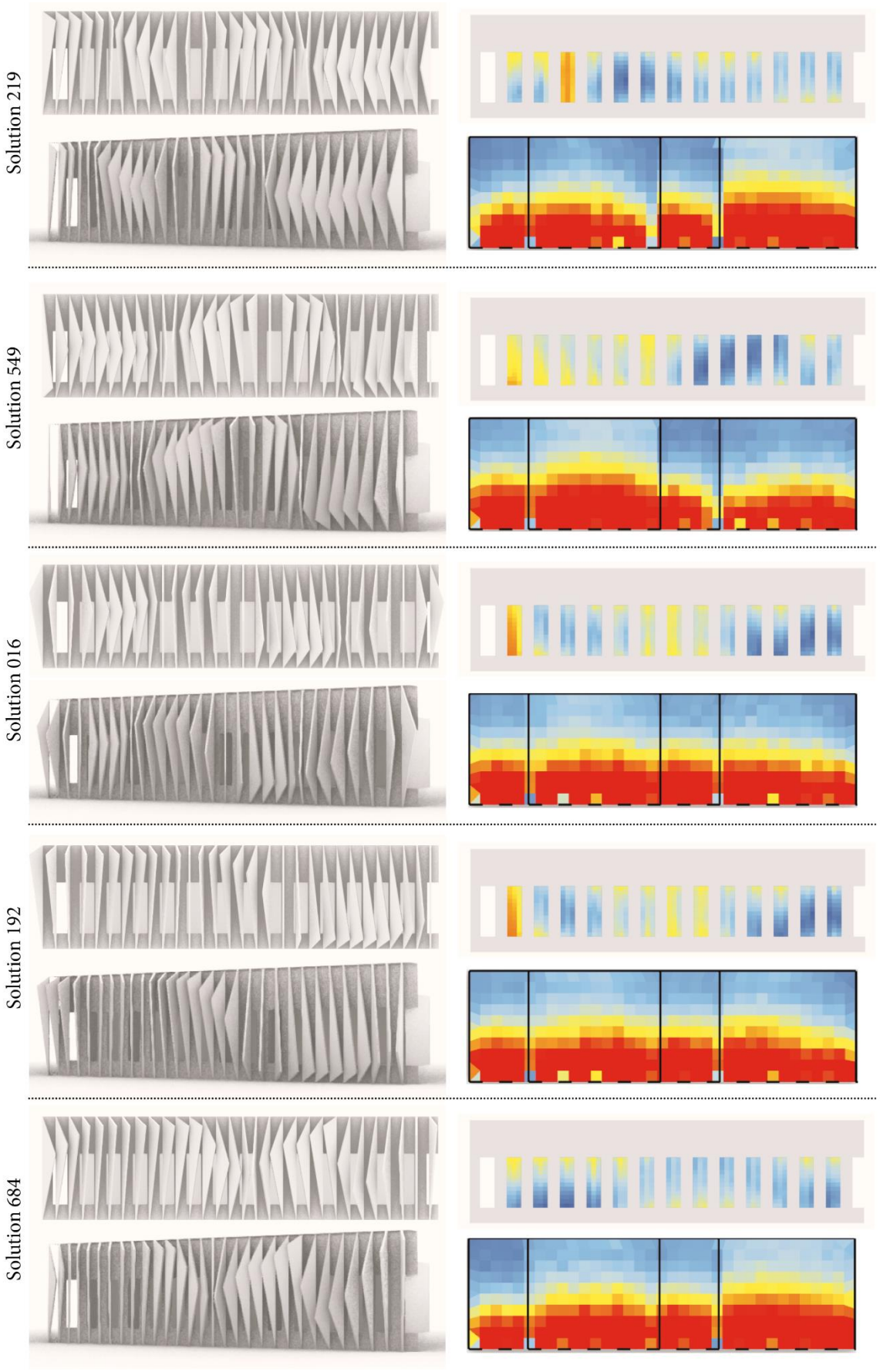


Figure 249 : Visualisation des solutions sur le front de Pareto de la méthode 4

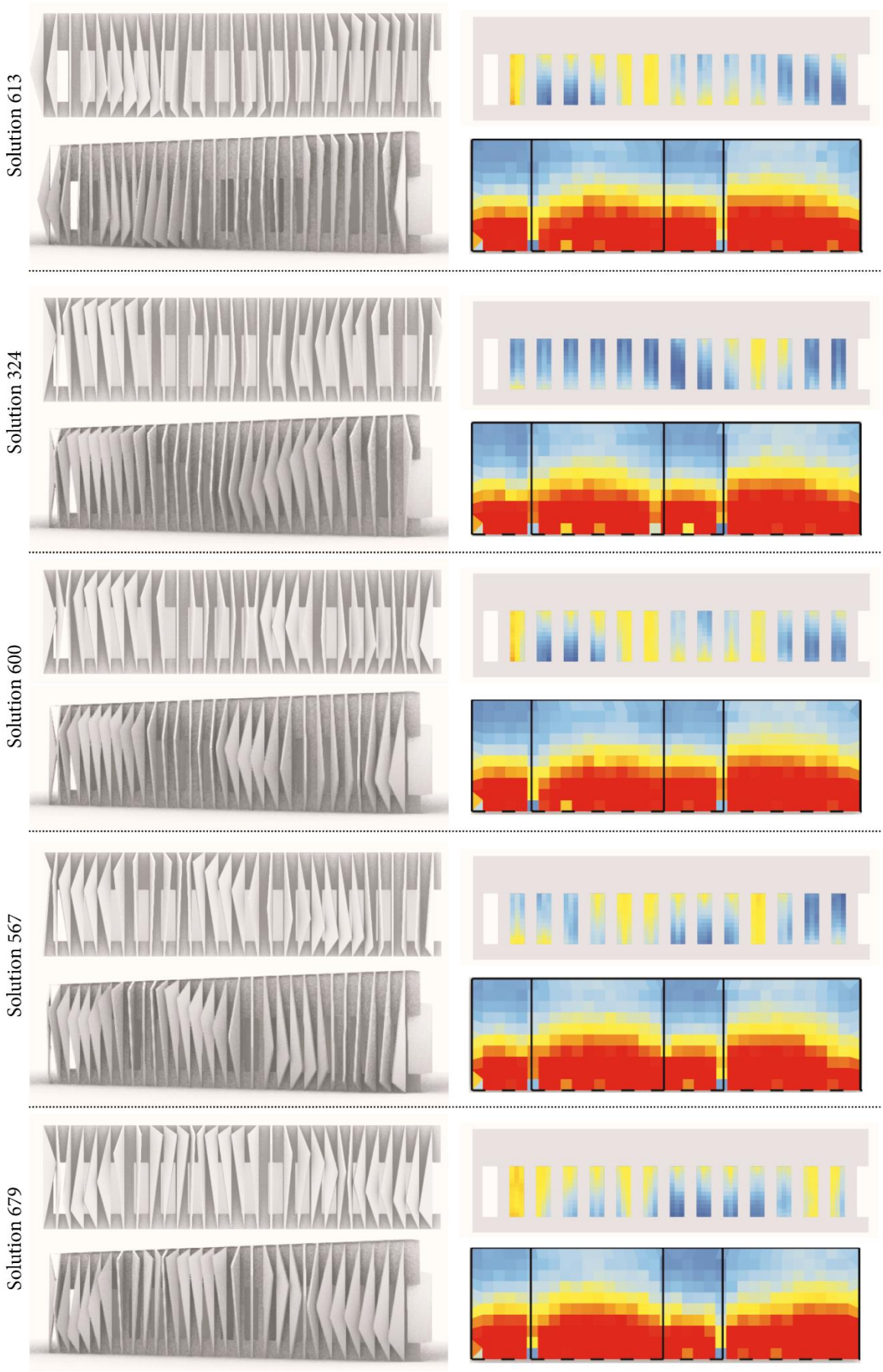


Figure 250 : Visualisation des solutions sur le front de Pareto de la méthode 4

Modèle à base de règles

Cette méthode est l'une des plus performantes parmi les 7 méthodes testées sur ce cas d'étude. Les résultats se distinguent bien du groupe contrôle. Elle compte 30 solutions sur le font de Pareto, dont 15 solutions contribuant au front de Pareto global des méthodes ad hoc. Ces solutions non-dominées sont aussi diversifiées, elles sont uniformément distribuées le long du front de Pareto. Cette méthode permet de générer des solutions bien plus performantes en termes de lumière et d'exposition au soleil direct que la méthode précédente.

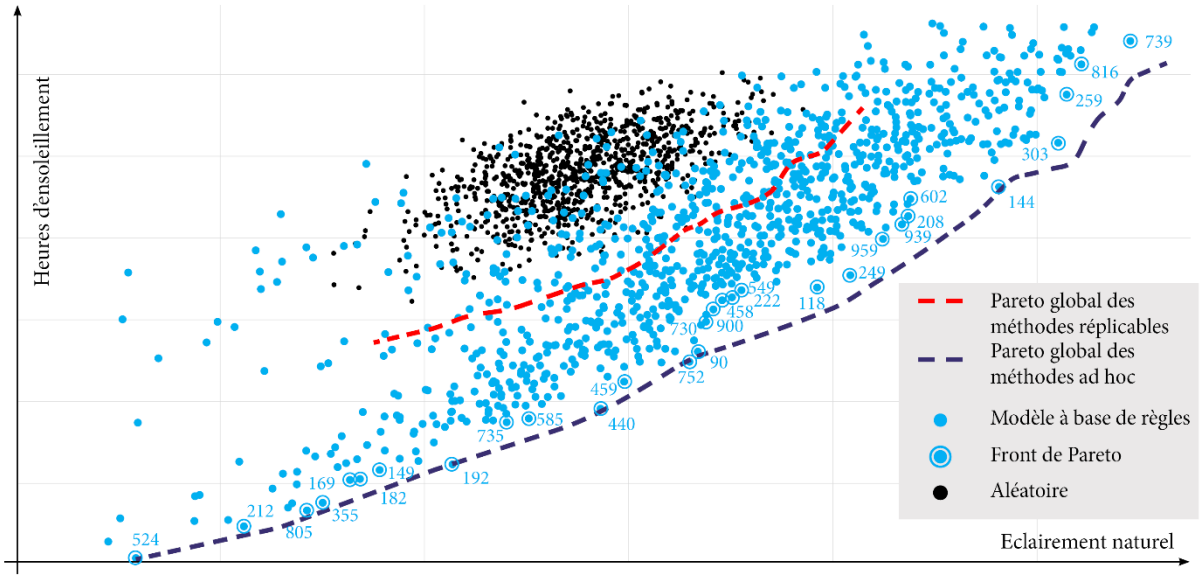


Figure 251 : Ensemble de solutions générée avec la méthode 5

Les 1 000 solutions générées avec cette méthode respectent les contraintes, aucune collision ou effet en dents de scie n'ont été détectés. Cette heuristique est donc particulièrement adaptée pour traiter ce type de contraintes. En plus d'être efficace, cette heuristique a été relativement facile à trouver et n'a pas demandé de nombreux tests, mais si elle demande des connaissances en programmation. Cependant, ce problème est simple puisque les lames ne concernent qu'un seul niveau. Une telle gestion des collisions (éléments après éléments) est plus complexe à réaliser sur plusieurs étages, notamment si les éléments peuvent entrer en collision avec des éléments des étages inférieurs et supérieurs.

Aussi, les résultats graphiques sont un peu décevants et ne répondent pas toujours aux attentes des équipes d'architectes. Comme on peut le voir sur les Figure 252 , Figure 253, Figure 254, Figure 255, Figure 256 et Figure 257, les solutions générées sont moins intéressantes qu'avec la méthode précédente. Les variations sont plus discrètes, générant parfois des façades presque homogènes comme pour les solutions 524, 149, 585 ou 752.

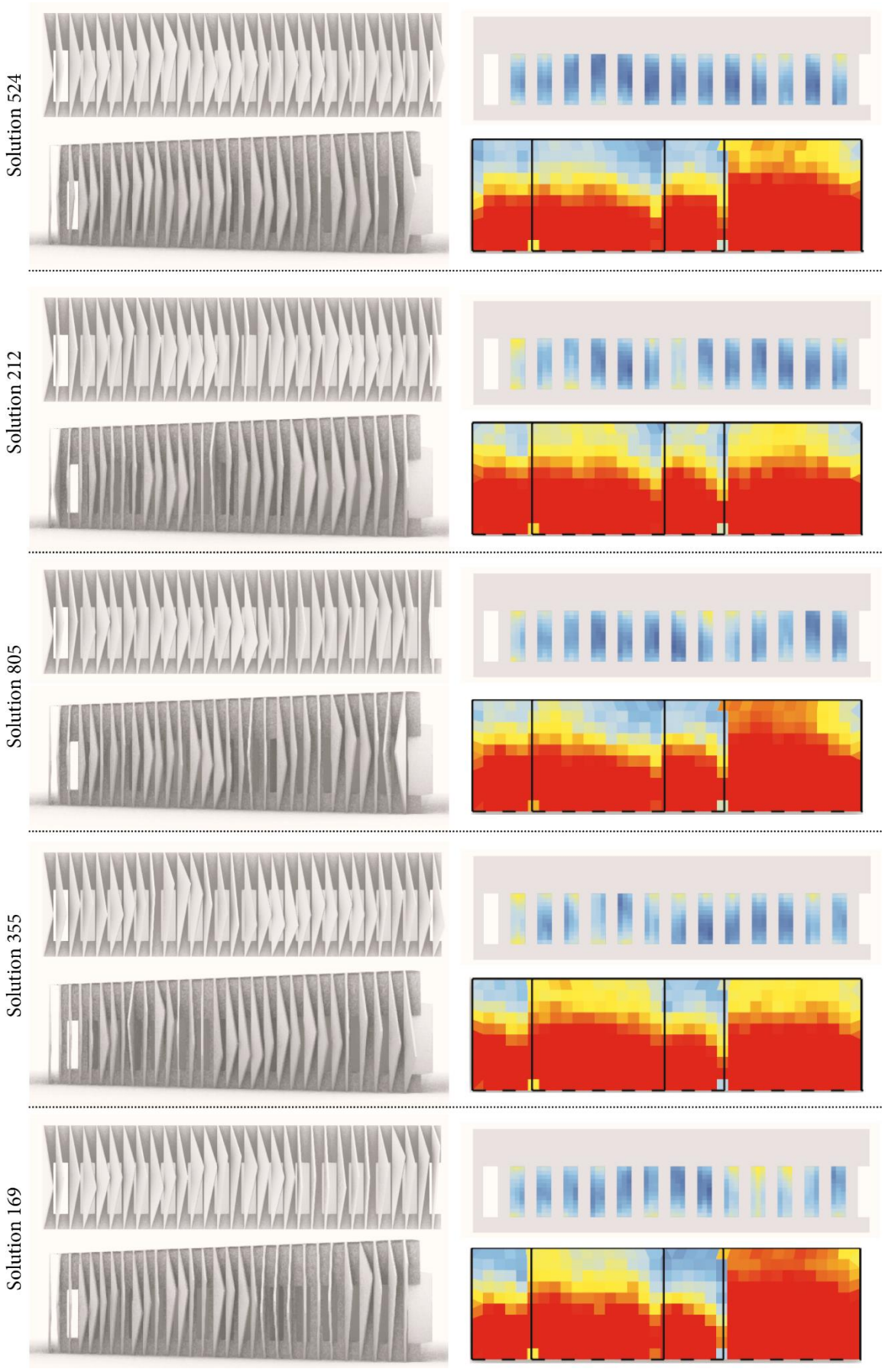


Figure 252 : Visualisation des solutions sur le front de Pareto de la méthode 5

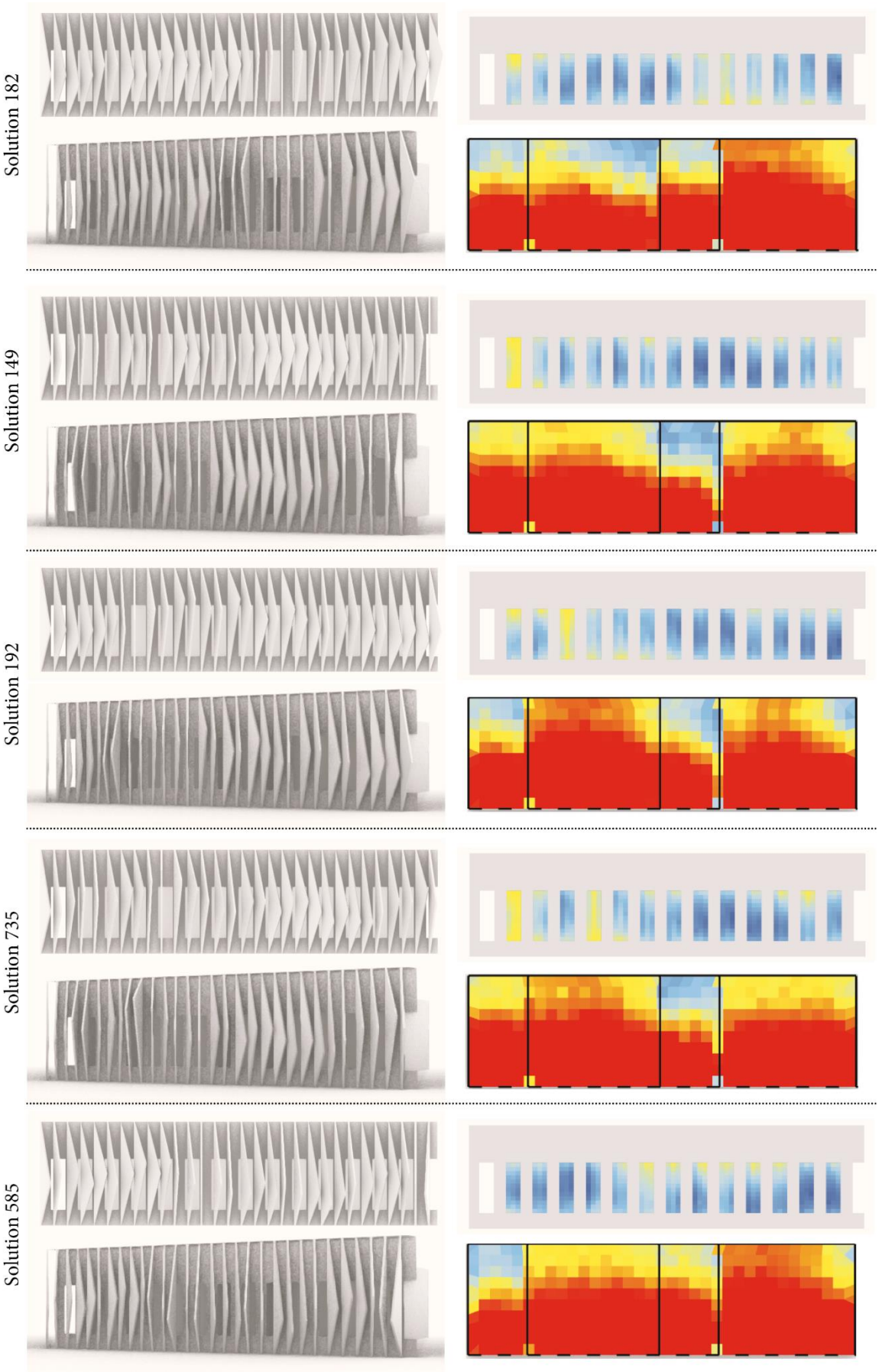


Figure 253 : Visualisation des solutions sur le front de Pareto de la méthode 5

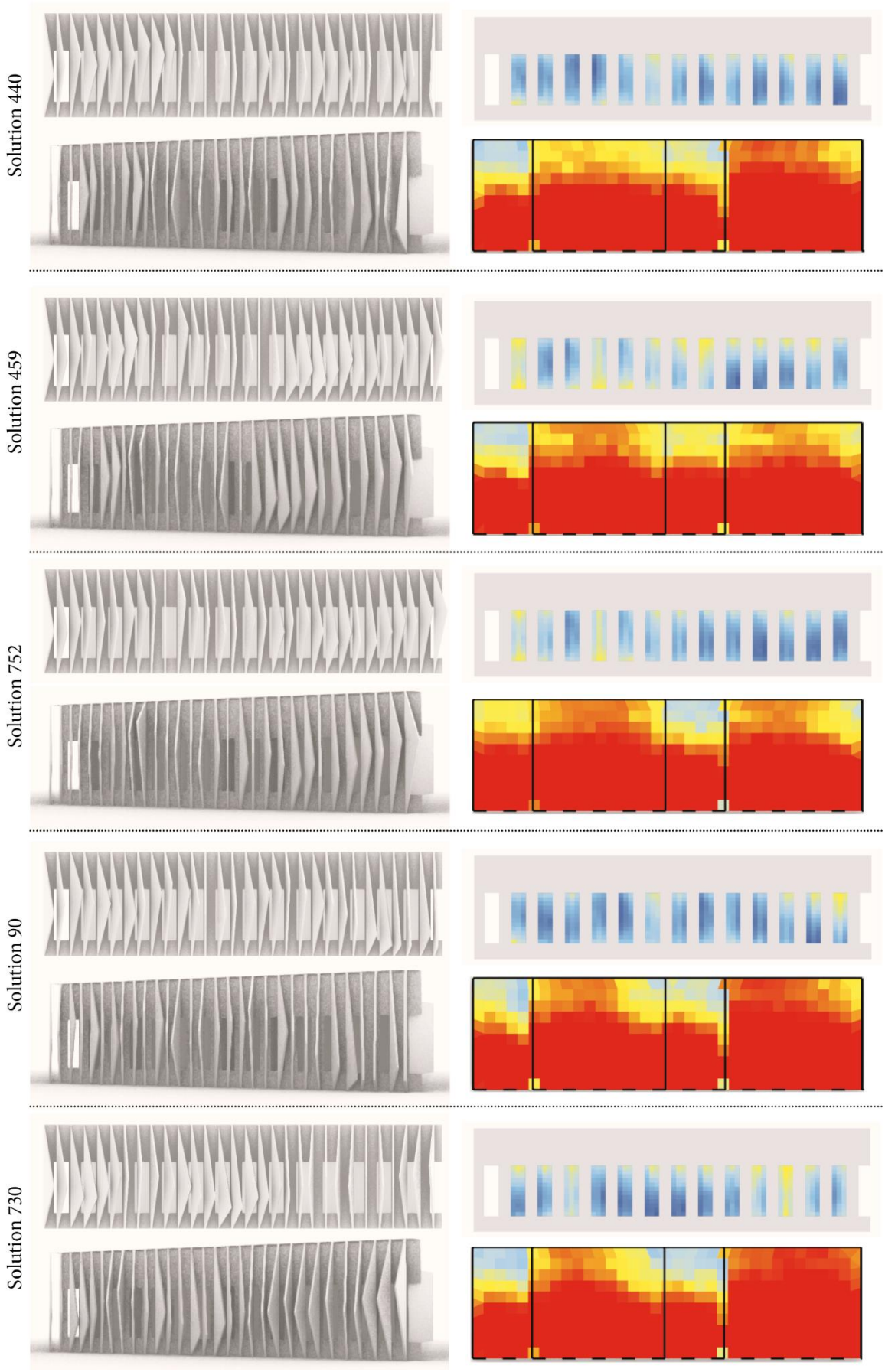


Figure 254 : Visualisation des solutions sur le front de Pareto de la méthode 5

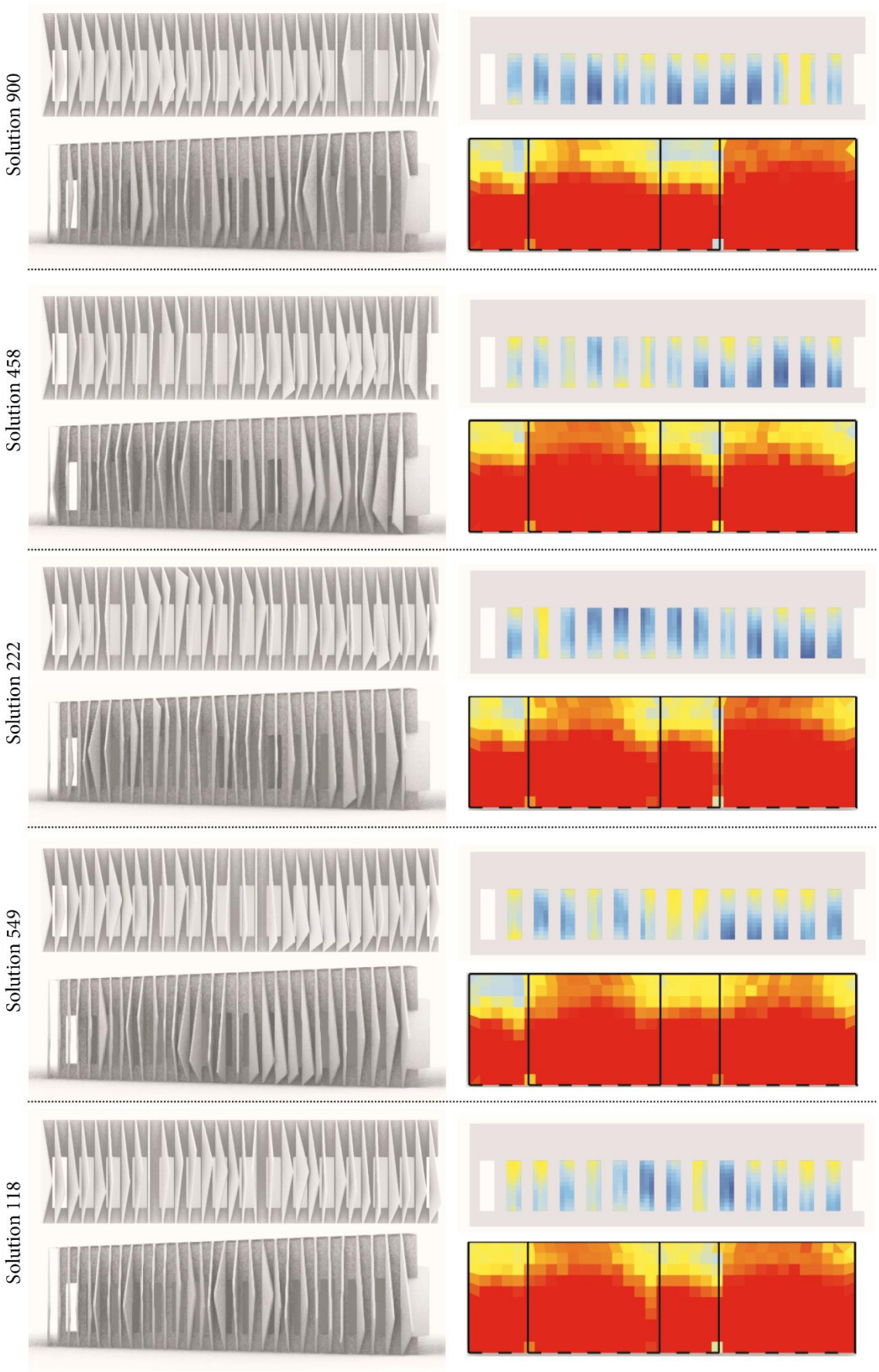


Figure 255 : Visualisation des solutions sur le front de Pareto de la méthode 5

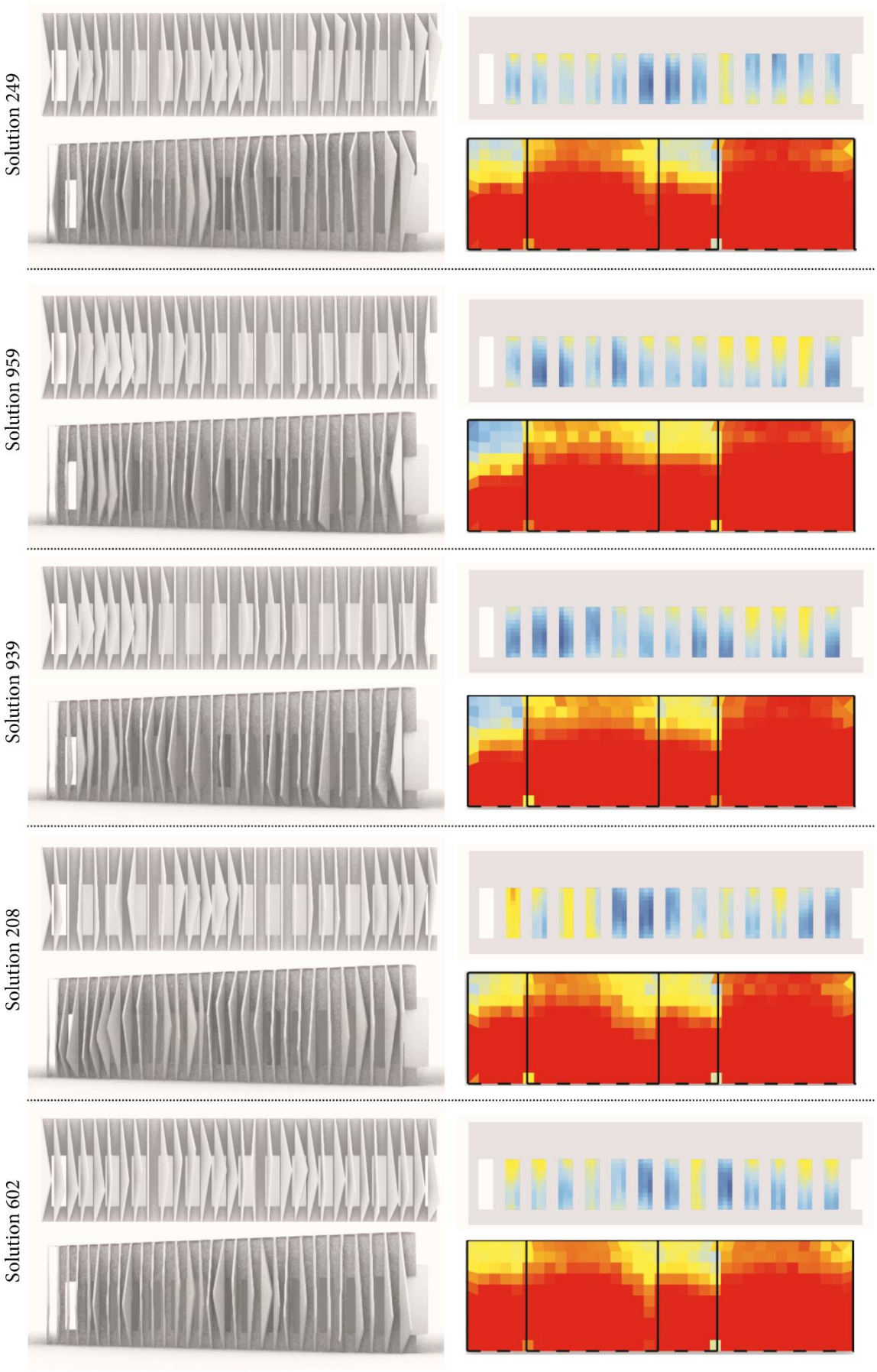


Figure 256 : Visualisation des solutions sur le front de Pareto de la méthode 5

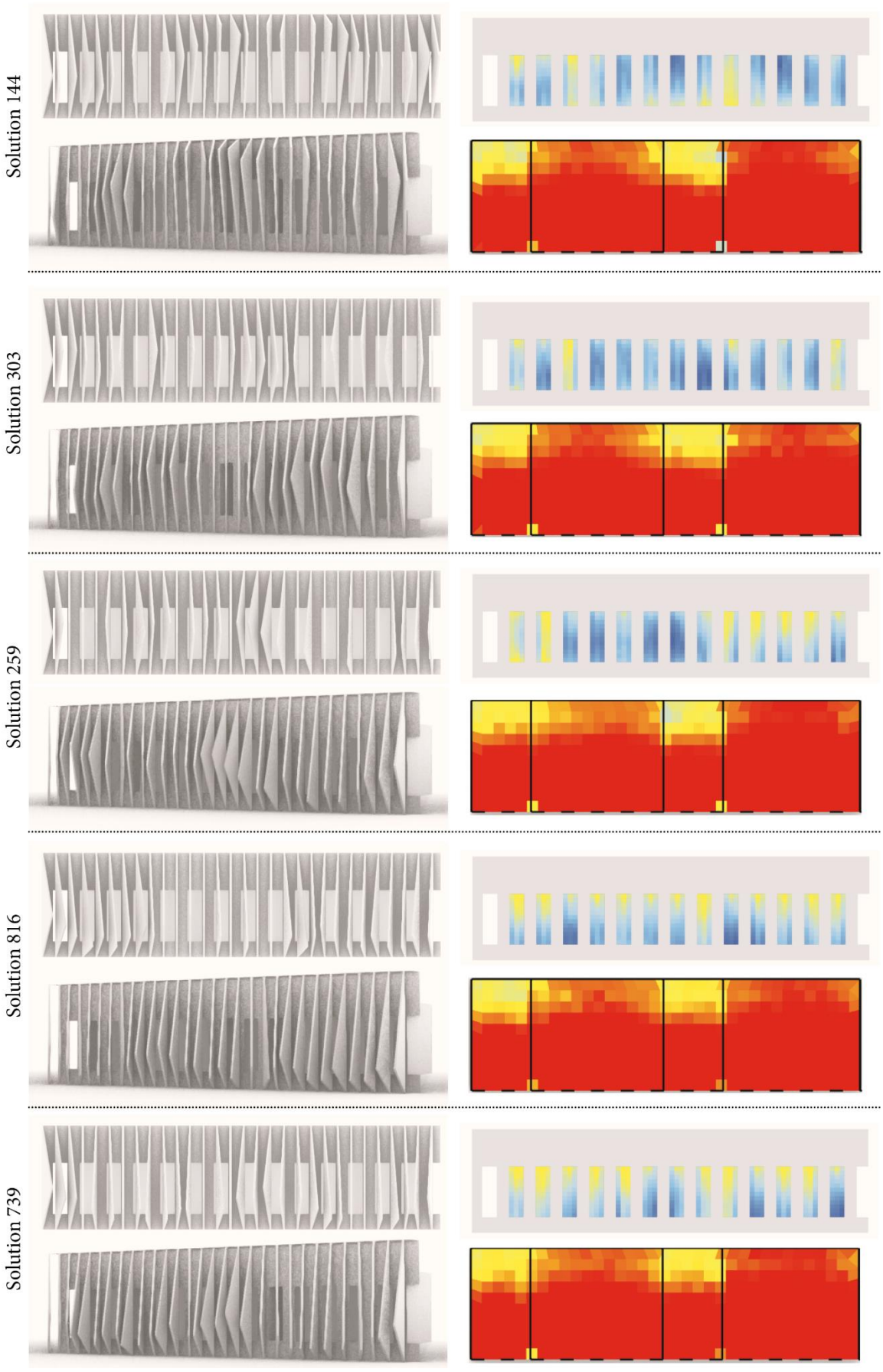


Figure 257 : Visualisation des solutions sur le front de Pareto de la méthode 5

Réparation baldwinienne

Cette méthode s'est révélée aussi très performante. Les solutions sont toutes très éloignées de la zone des solutions générées aléatoirement. Le front de Pareto compte une vingtaine d'individus, dont 4 contribuent au front de Pareto global des méthodes ad hoc.

Cependant, comme on peut l'observer sur le graphique présenté en Figure 258, les solutions sont moins diversifiées que pour les deux solutions précédentes. Sur ce cas précis, la réparation baldwienne obtient un front de Pareto moins bon que la méthode à base de règles.

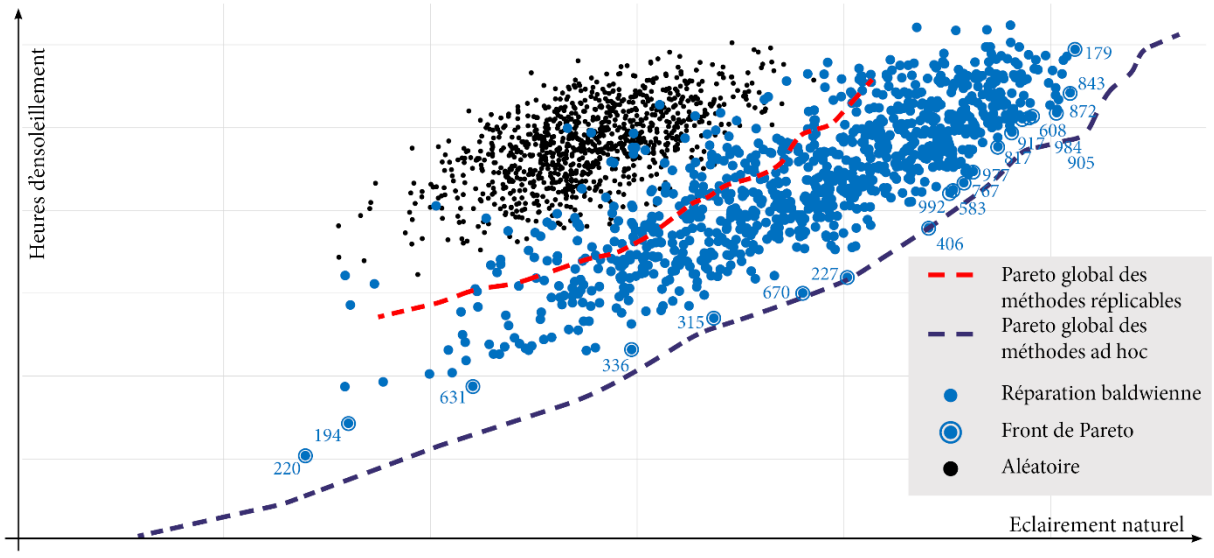


Figure 258 : Ensemble de solutions générée avec la méthode 6

Les 1 000 solutions générées respectent les contraintes. Si cette méthode de gestion des contraintes est très efficace, il est important de souligner que plus d'une dizaine de tests ont été nécessaires avant de trouver la bonne heuristique associée à un taux de réussite de 100 %. Pour chaque test réalisé, une heuristique a dû être imaginée et programmée. 1 000 solutions ont été générées et évaluées sur leurs degrés de violation des contraintes avec les fonctions développées pour les méthodes réplicables.

Pour ce cas spécifique, le temps passé à trouver la bonne heuristique fut utile car les solutions générées avec cette technique correspondent bien mieux aux attentes de l'équipe d'architecte en termes de design que toutes les autres méthodes déjà testées. Les solutions générées contiennent toutes des lames aux formes bien diversifiées et aux orientations diverses, comme on peut le voir sur les Figure 259, Figure 260, Figure 261et Figure 262.

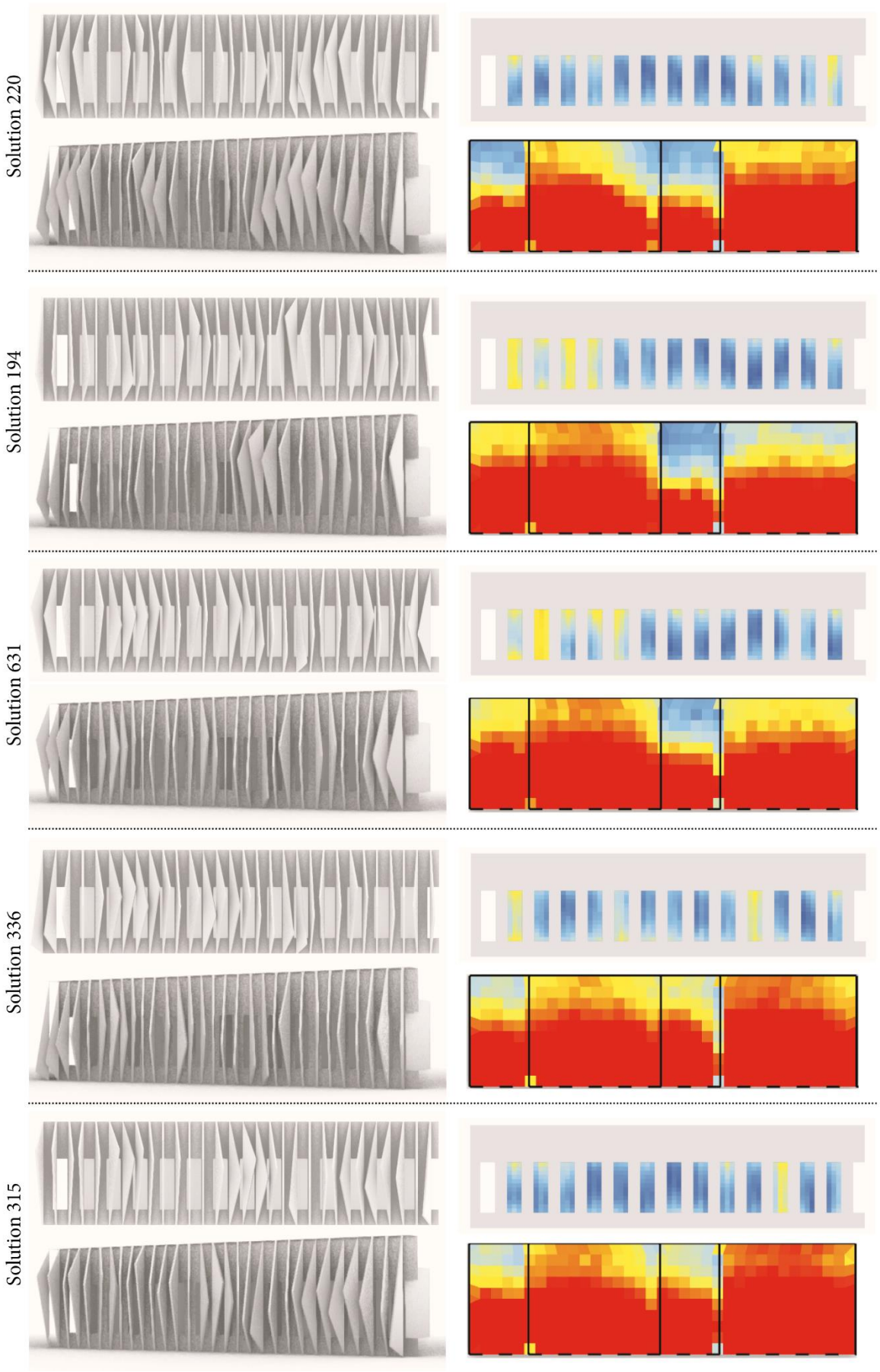


Figure 259 : Visualisation des solutions sur le front de Pareto de la méthode 6

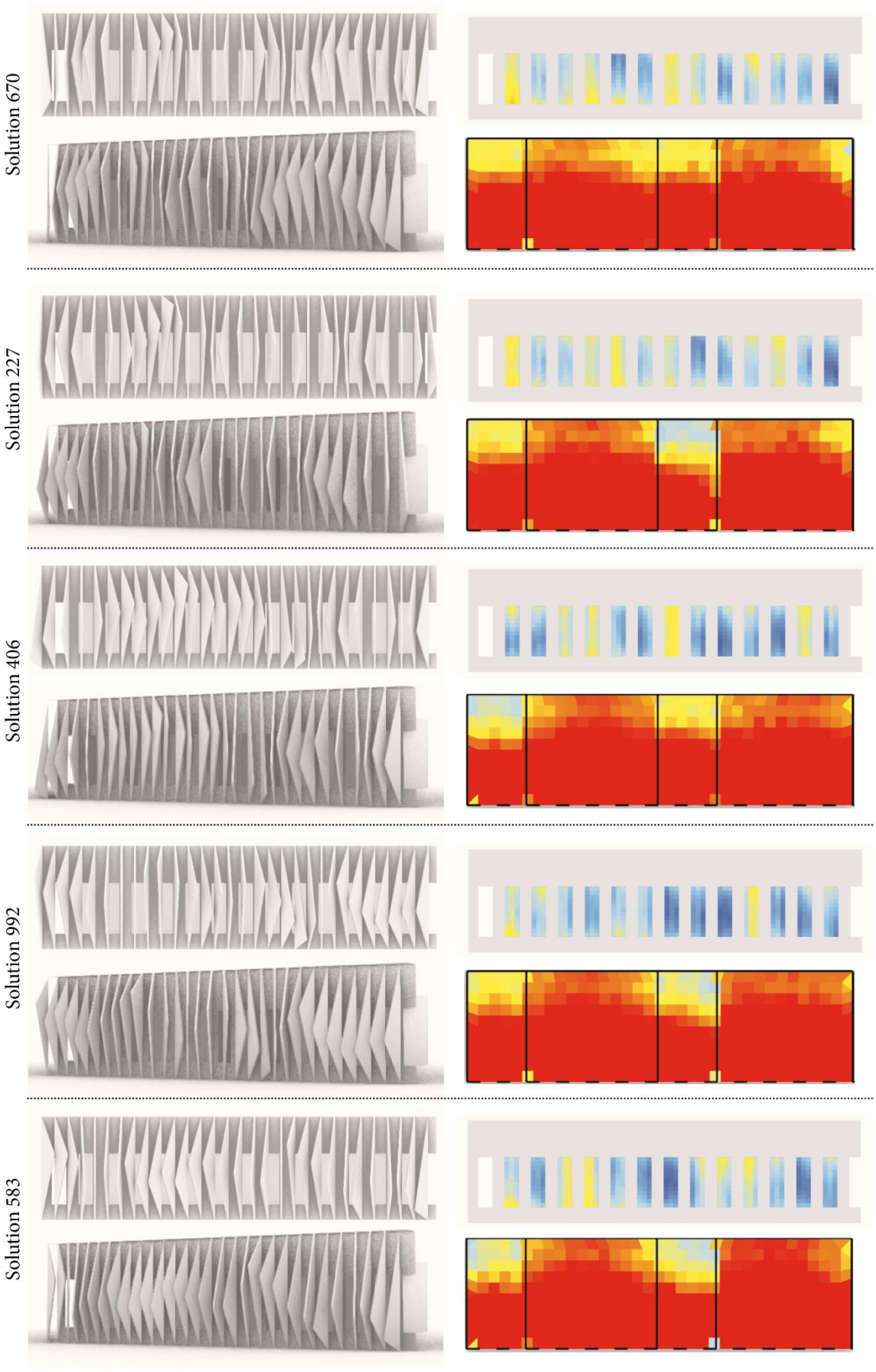


Figure 260 : Visualisation des solutions sur le front de Pareto de la méthode 6

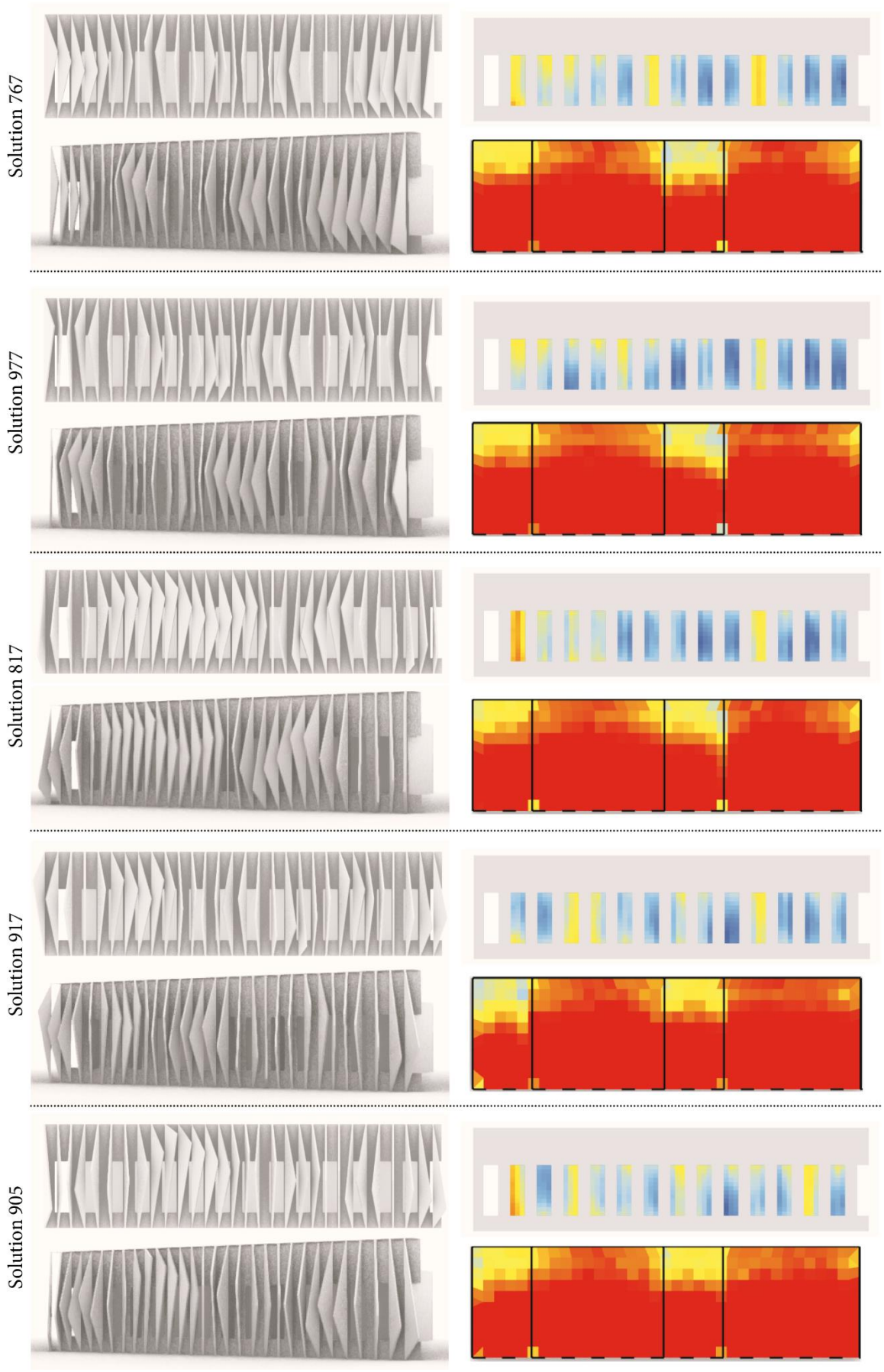


Figure 261 : Visualisation des solutions sur le front de Pareto de la méthode 6

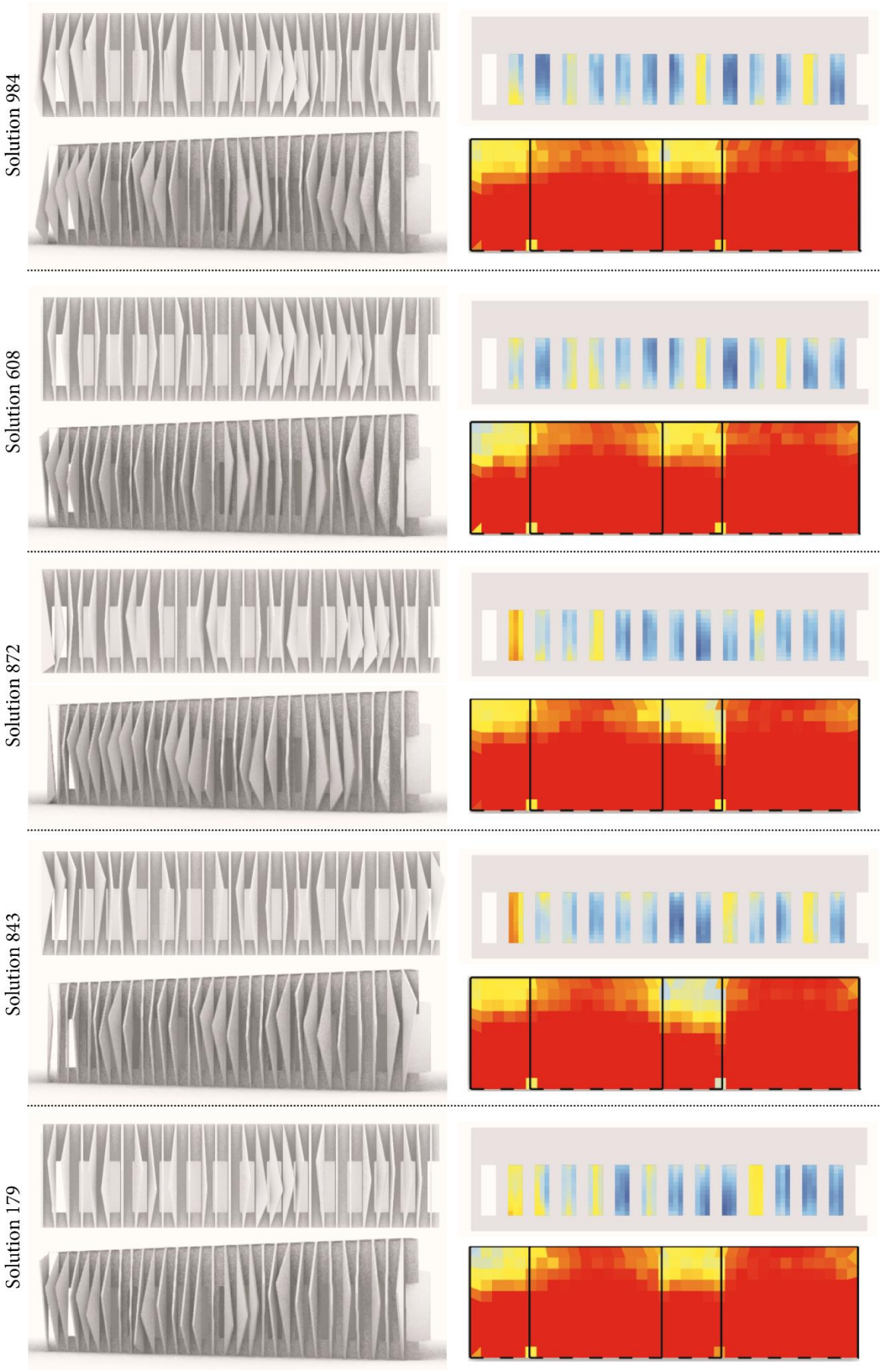


Figure 262 : Visualisation des solutions sur le front de Pareto de la méthode 6

Réparation lamarckienne

Cette dernière méthode s'est révélée particulièrement performante. Toutes les solutions générées se sont rapidement éloignées du groupe contrôle. Elle est la méthode qui compte le plus d'individus sur son front de Pareto avec 37 solutions. Avec l'approche lamarckienne, les solutions sont mieux réparties sur le front de Pareto qu'avec l'approche baldwinienne. Cette méthode est aussi la méthode qui contribue le plus au front de Pareto global des méthodes ad hoc avec 21 individus.

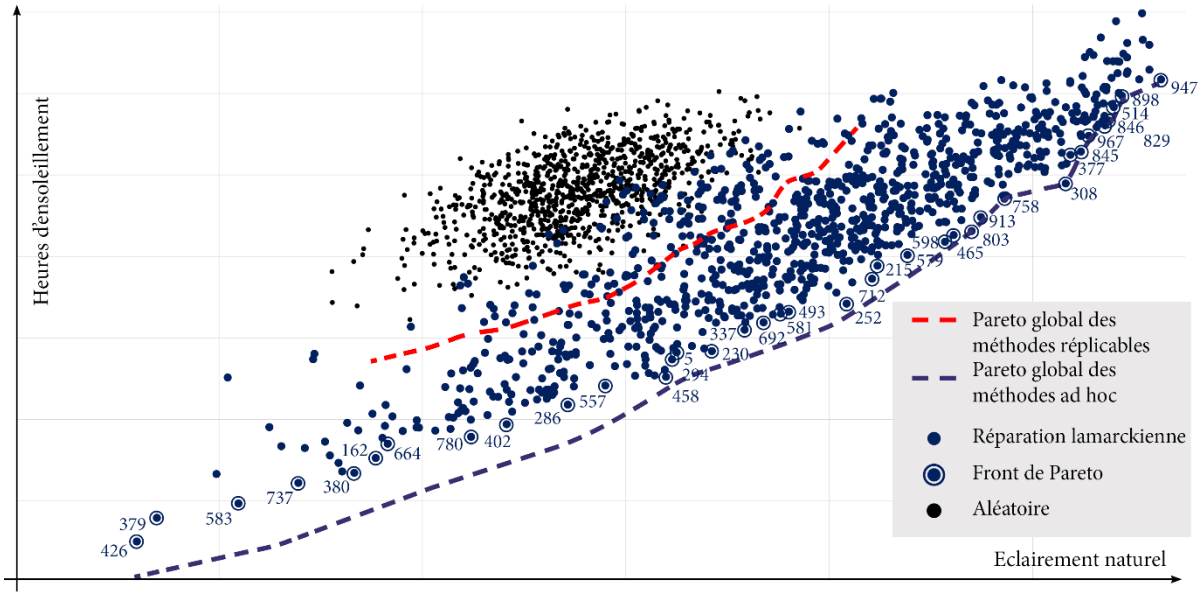


Figure 263 : Ensemble de solutions générées avec la méthode 7

Comme pour la réparation baldwienne, les solutions générées respectent les contraintes et sont esthétiquement satisfaisantes comparées aux solutions générées avec les autres heuristiques, comme on peut le voir sur les Figure 264, Figure 265, Figure 266, Figure 267, Figure 268 et Figure 269.

Cependant, cette triple performance (environnementale, intégration des contraintes, et esthétique) a un coût. Nous rappelons que trouver cette heuristique a pris du temps, et ajoutons que déterminer le bon taux de remplacement des solutions réparées peut aussi être chronophage. Avant d'utiliser un taux de remplacement de 25 %, nous avons lancé les calculs avec un taux à 5 % et à 15 %. Ces derniers présentaient de meilleurs résultats qu'avec 0 %, mais pas aussi bon qu'avec 25 %. Ainsi la recommandation des 5 % de solutions remplacées (Orvosh & Davis, 1993) ne fonctionne pas pour tous les problèmes.

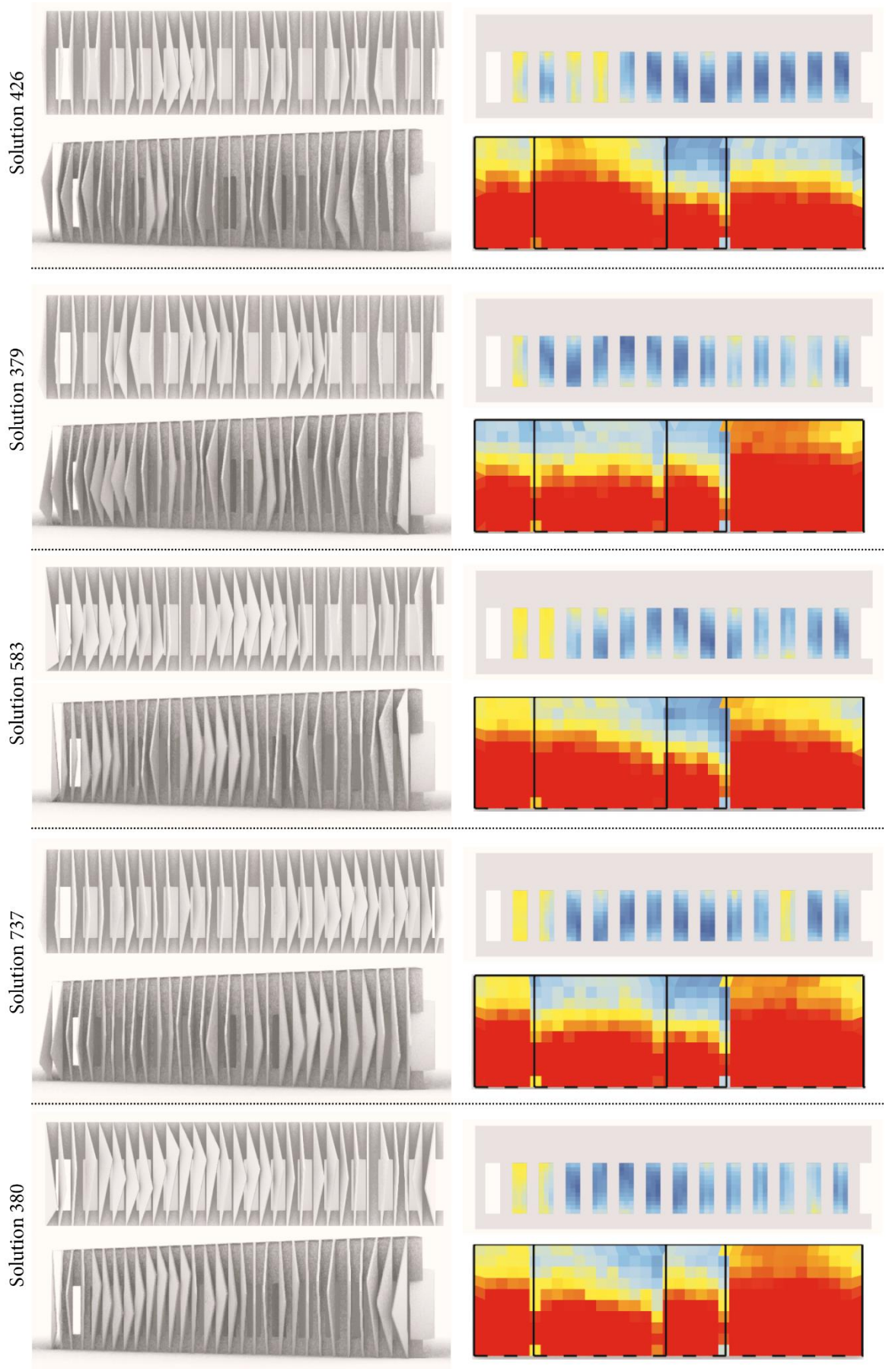


Figure 264 : Visualisation des solutions sur le front de Pareto de la méthode 7

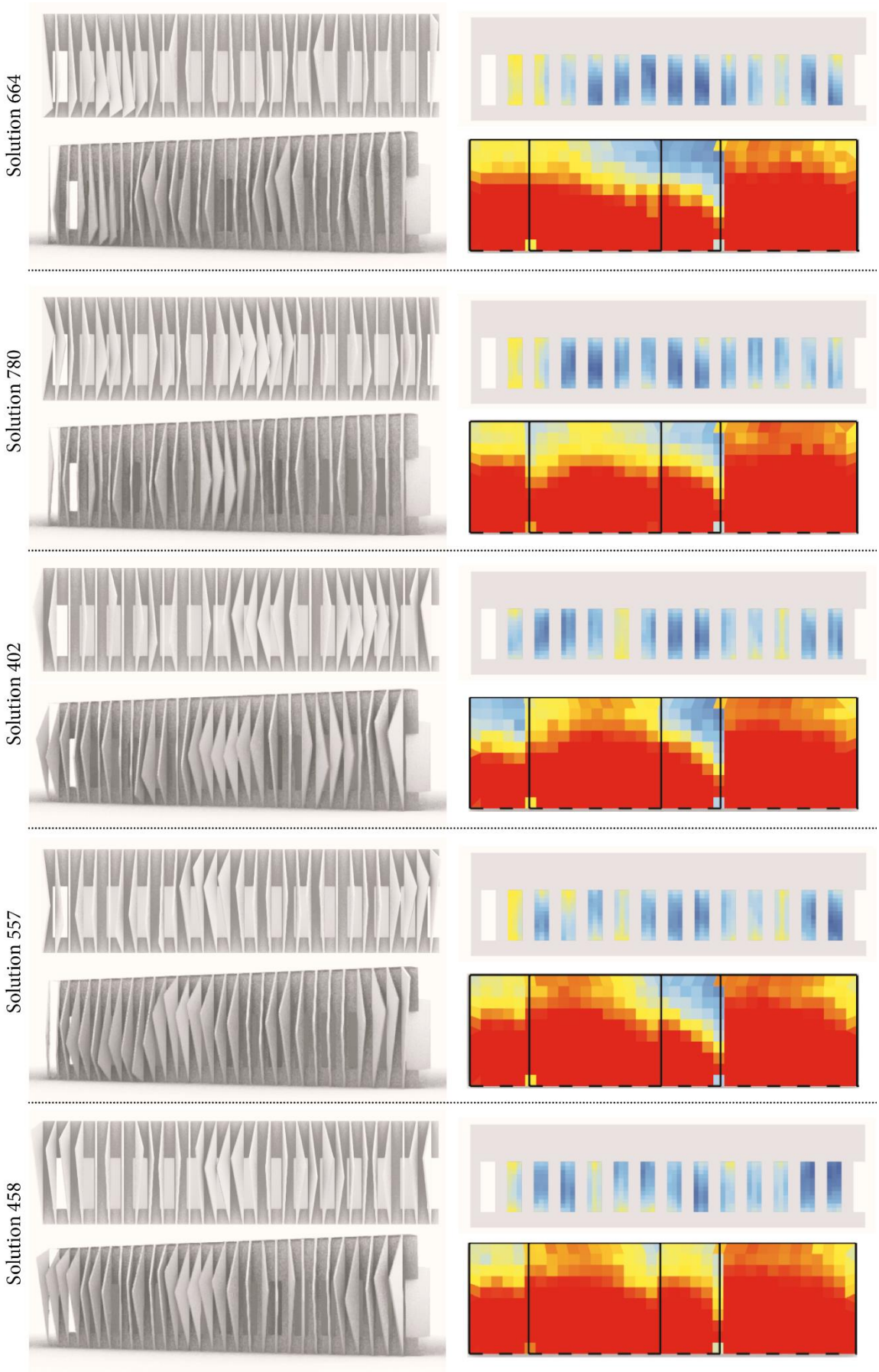


Figure 265 :Visualisation des solutions sur le front de Pareto de la méthode 7

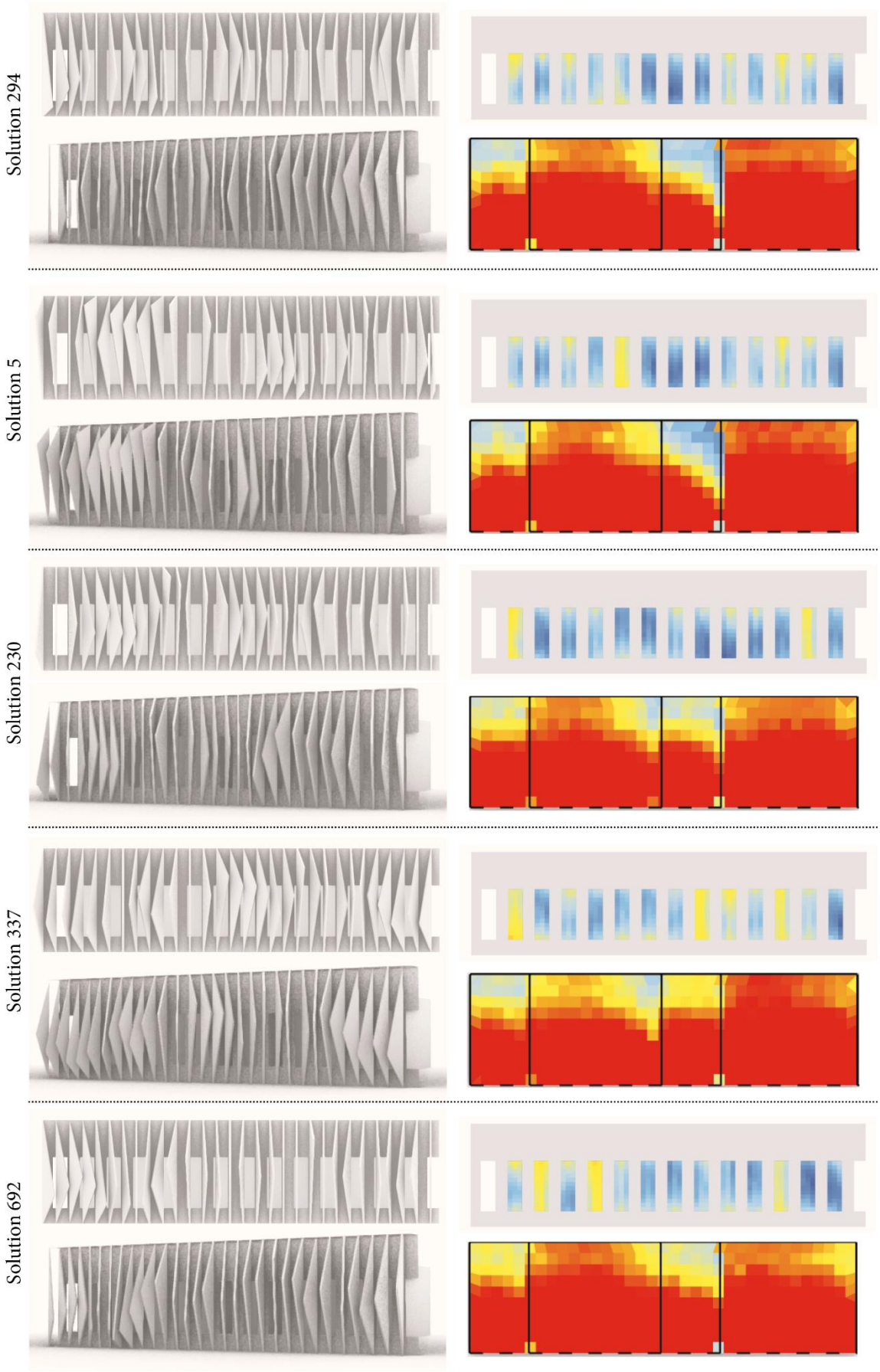


Figure 266 : Visualisation des solutions sur le front de Pareto de la méthode 7

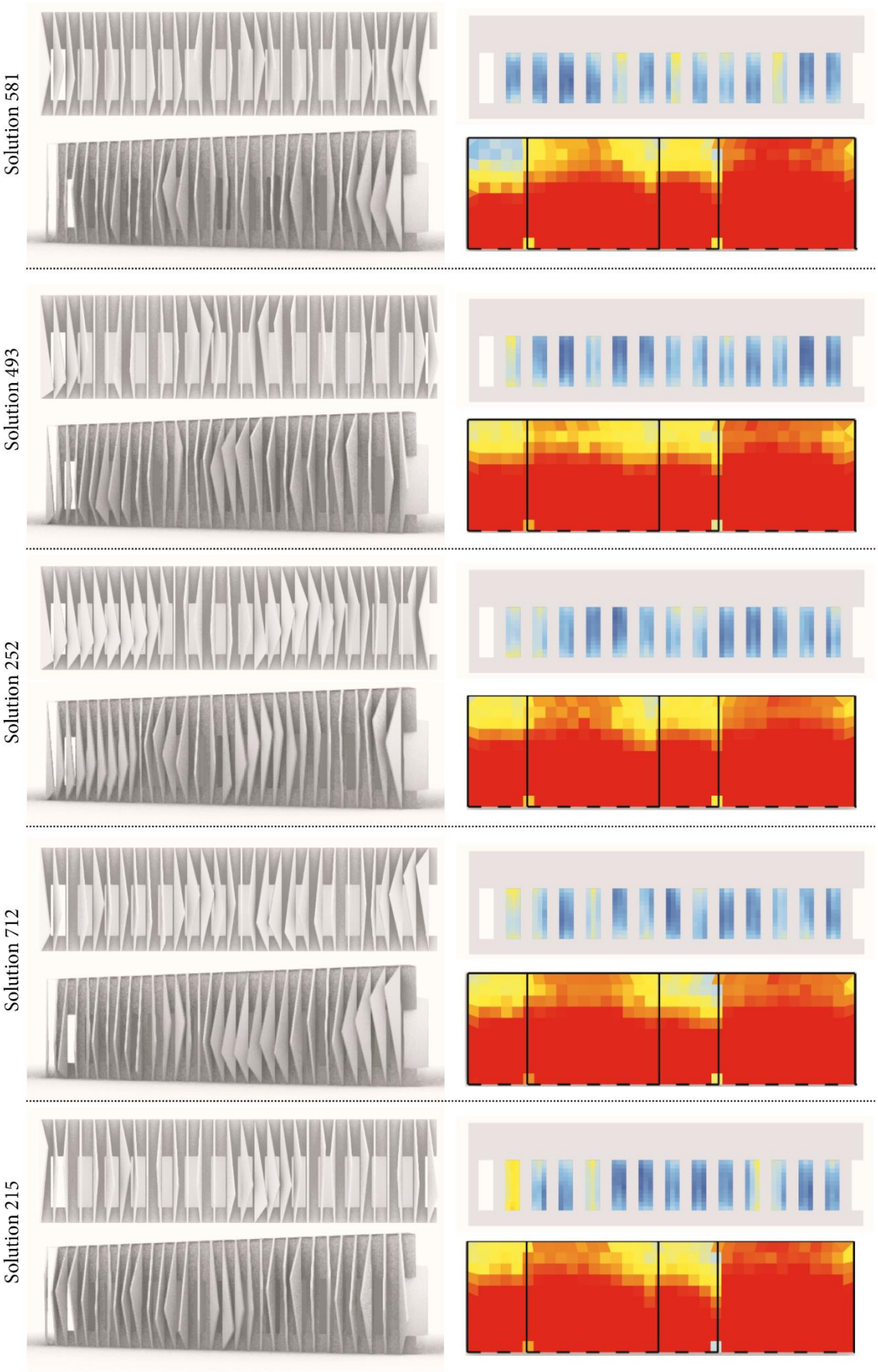


Figure 267 : Visualisation des solutions sur le front de Pareto de la méthode 7

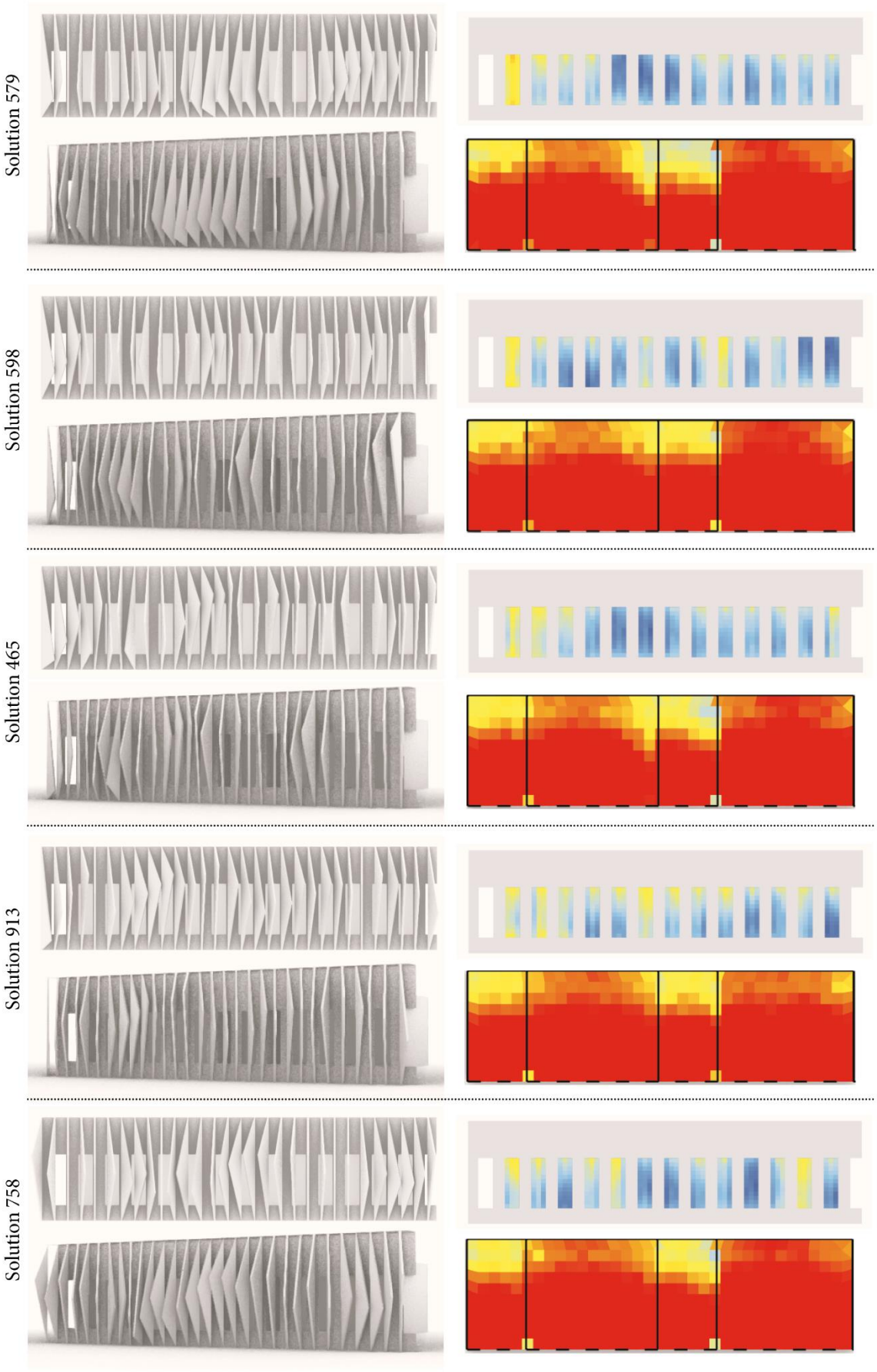


Figure 268 : Visualisation des solutions sur le front de Pareto de la méthode 7

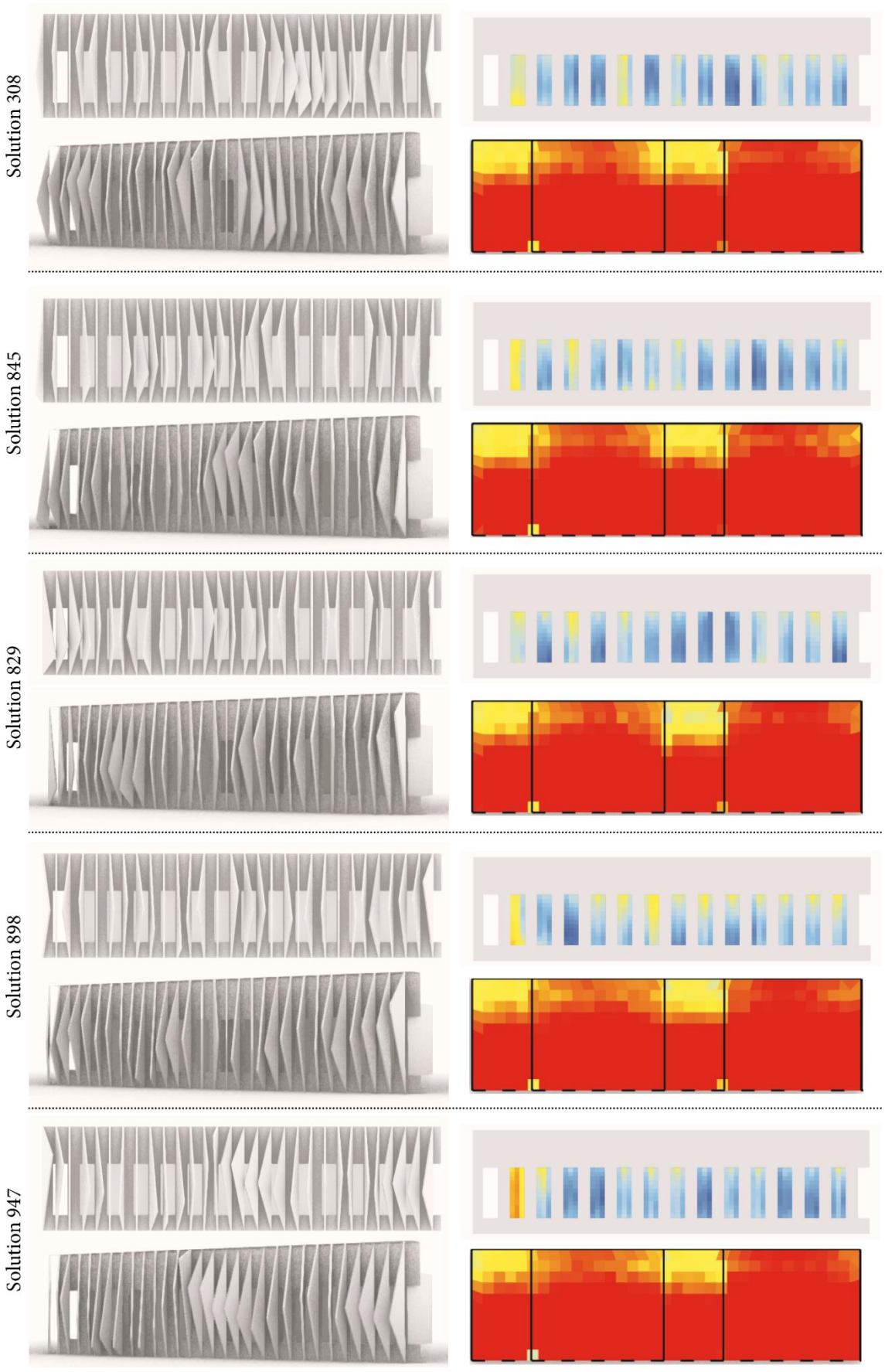


Figure 269 : Visualisation des solutions sur le front de Pareto de la méthode 7

Synthèse sur les méthodes ad hoc

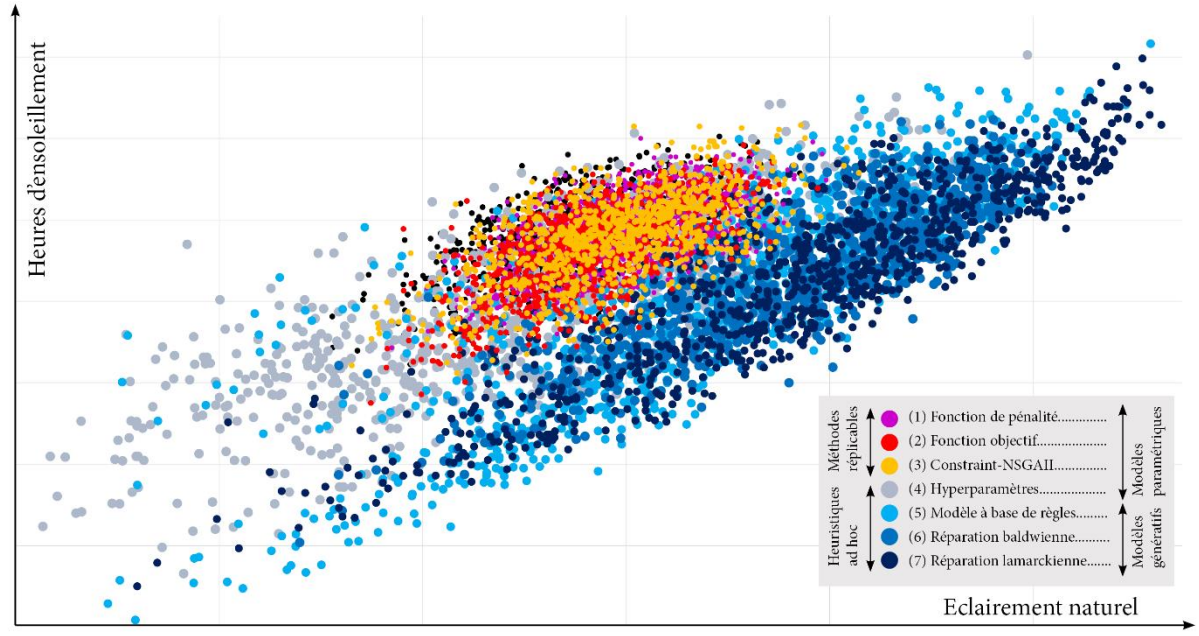


Figure 270 : Ensemble des solutions générées avec les 7 méthodes de gestion des contraintes

Comme nous pouvons le voir sur la Figure 270, il est plus facile de distinguer les résultats des méthodes ad hoc que ceux des méthodes génériques. La réparation Lamarckienne (7) compte beaucoup plus d'individus sur les premiers rangs. L'heuristique à base de règles (5) est aussi très performante. Le modèle paramétrique avec des hyperparamètres est ici beaucoup moins efficace avec seulement une solution dans les deux premiers rangs (voir Figure 271). Surtout, les méthodes réplicables sont beaucoup moins performantes que les méthodes ad hoc, et les méthodes utilisant des modèles génératifs (5, 6, 7) sont plus efficaces que les méthodes utilisant des modèles paramétriques (1,2,3,4).

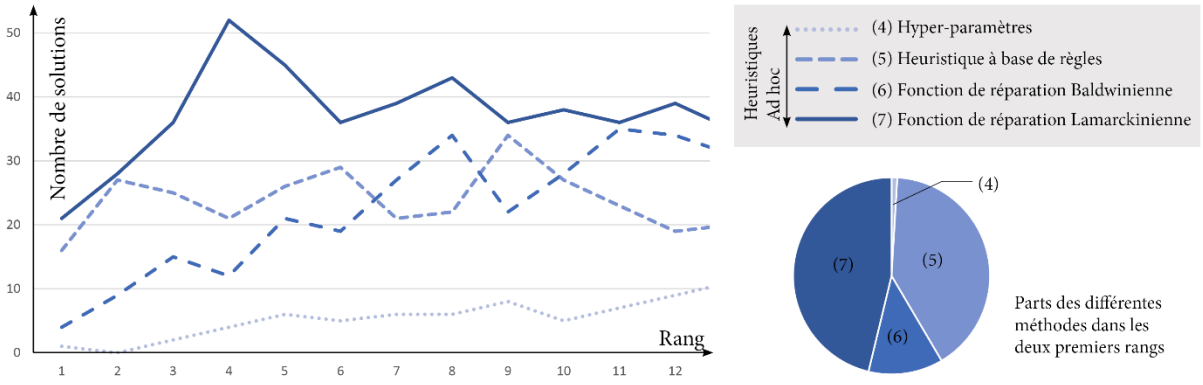


Figure 271 : Tri non-dominé de l'ensemble des solutions générées avec les méthodes ad hoc

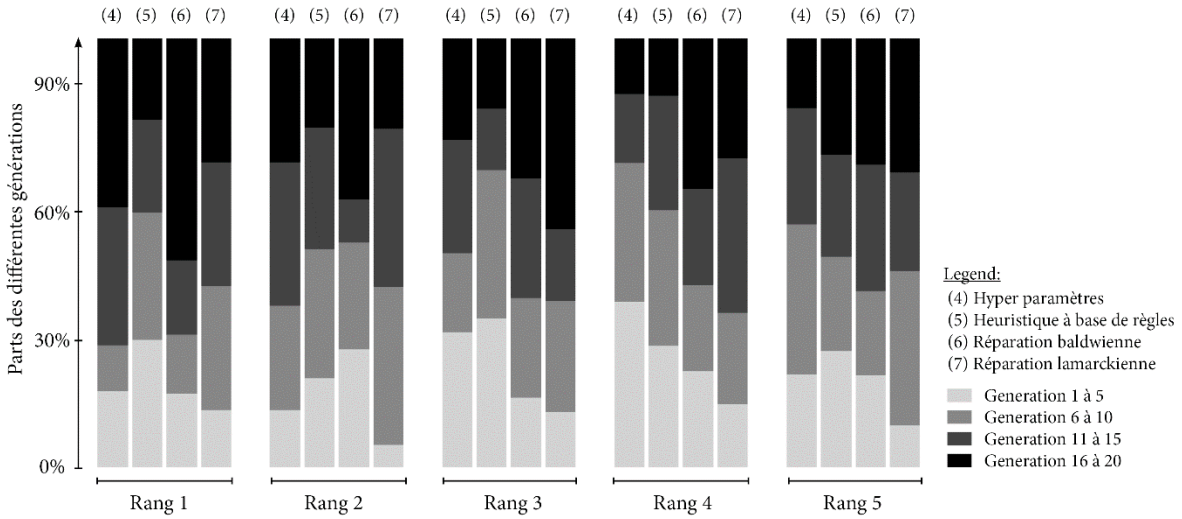


Figure 272 : Parts des différentes générations en fonction du rang pour les méthodes ad hoc

Aussi, les méthodes (5) et (7) ont générées plus rapidement leurs meilleures solutions, notamment pour les solutions des deux premiers rangs (voir Figure 272). La méthode des hyperparamètres contribue moins au front de Pareto global et la convergence de l'algorithme est moins rapide, 40 % des solutions du front de Pareto (rang 1) de la méthode (4) ont été trouvés lors des 5 dernières générations. Il y a un véritable intérêt dans ce cas d'étude à utiliser des méthodes génératives pour gérer les contraintes.

Au terme de cette deuxième partie, après avoir identifié, testé sur un cas d'étude issue de la pratique professionnelle et comparé les méthodes d'intégration des contraintes avec des algorithmes d'optimisation génétiques, nous proposons une classification des méthodes en Figure 273. S'il n'est pas possible de tirer des conclusions générales valides pour tous les types de projets, nous pouvons cependant, avec l'expérience acquise au fil de la pratique et de la lecture des experts, faire quelques recommandons aux praticiens.

Avant tout choix de méthode, il est important de faire une évaluation de la taille de la région faisable. Nous savons que les méthodes génériques peuvent se révéler inefficaces lorsque la région faisable est très étroite. Lorsque ce n'est pas le cas, nous recommandons d'utiliser des méthodes de gestion des contraintes internes comme Constraint-NSGAII pour gagner du temps en évitant des erreurs liées à une mauvaise définition des coefficients de pénalité. Si les solveurs à disposition n'intègrent pas de méthode interne, alors il est toujours possible de définir les contraintes comme une fonction objectif.

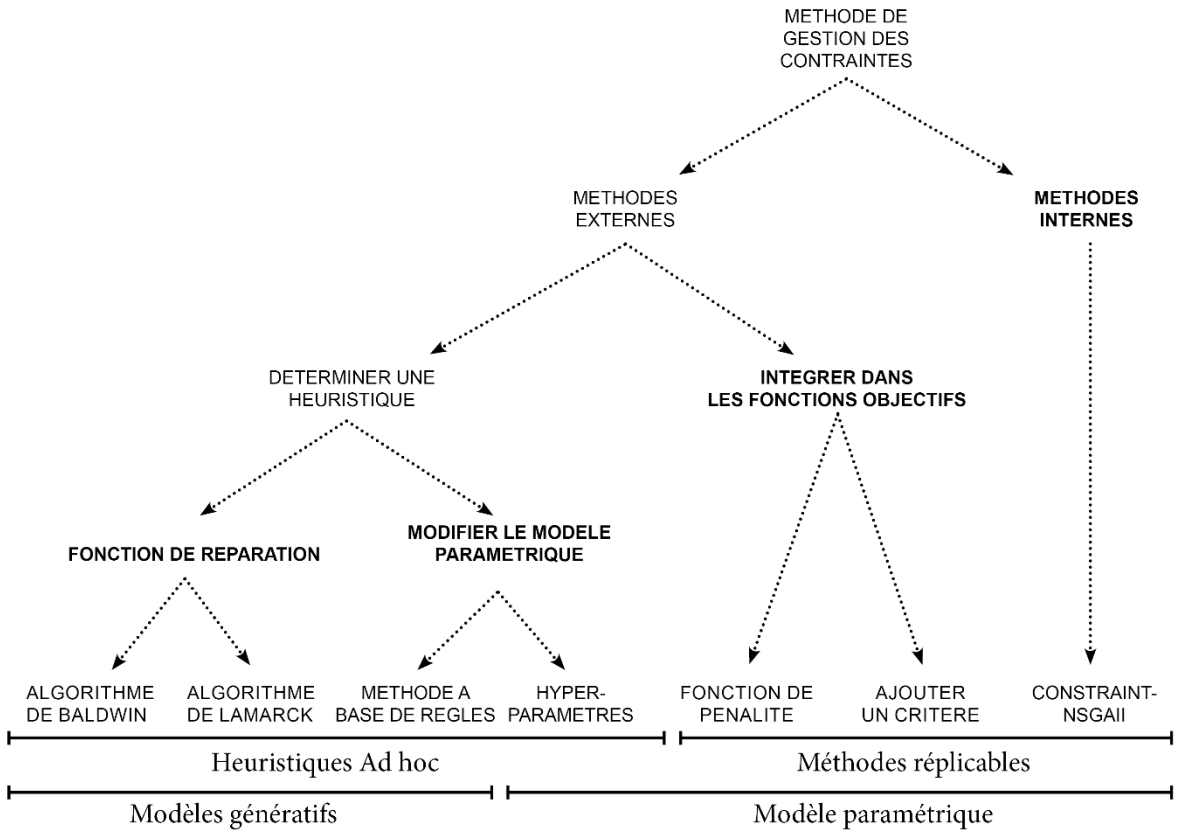


Figure 273 : Classification des méthodes de gestion des contraintes pour les problèmes de conception architecturale

Si la région faisable est étroite et qu'on ne maîtrise pas la modélisation générative, lorsqu'une heuristique peut être trouvée, la méthode des hyperparamètres est une bonne solution intermédiaire. Enfin, si la région faisable est étroite et que l'on maîtrise la modélisation générative, deux solutions sont possibles : soit trouver une heuristique qui permette de ne générer que des solutions faisables, soit trouver une heuristique qui permette de réparer les solutions infaisables.

Au terme du chapitre 3, nous savons désormais que les problèmes rencontrés par les architectes sont des problèmes d'optimisation globaux, discrets, et que les fonctions objectifs étudiées sont des boîtes noires. Ainsi, ils doivent donc être résolus avec des méthodes sans gradient. Les méthodes stochastiques comme les métaheuristiques sont donc particulièrement adaptées pour résoudre ce type de problèmes. Aussi, il s'agit la plupart du temps de problèmes multicritères, aux unités difficilement comparables, faisant des algorithmes évolutionnaires multiobjectifs, utilisant la méthode de Pareto, des outils particulièrement adaptés.

De plus, les problèmes des architectes sont souvent surcontraints. Les méthodes génériques de gestion des contraintes couramment utilisées avec ce type d'algorithmes se révèlent inadaptées. Nous avons identifié des méthodes pouvant s'avérer bien plus efficaces qui consistent à soit reformuler le problème pour supprimer la contrainte, soit réparer les solutions générées avant évaluation pour qu'elles deviennent faisables à l'aide d'une heuristique appelée fonction de réparation. Ces méthodes permettent de réduire les temps de calcul en réduisant la taille de l'espace de solution et en évitant d'évaluer des solutions inutiles. Cependant, elles impliquent pour chaque problème de trouver un algorithme génératif nécessitant des connaissances avancées en programmation que n'ont pas les architectes.

Ainsi, nous avons choisi de poursuivre nos recherches sur les fonctions de réparation et sur les différentes techniques génératives utiles à leur élaboration, afin de trouver des méthodes pour les rendre plus accessibles aux non-experts de la programmation informatique.

Chapitre 4 : Les techniques génératives comme fonction de réparation pour faire de l'optimisation sous-contraintes

Dans ce dernier chapitre, nous poursuivons nos recherches sur l'intégration des contraintes dans un processus d'optimisation en nous concentrant sur la méthode de réparation. Pour cela, nous commencerons par interroger la capacité des techniques génératives populaires en conception computationnelle (comme les grammaires de formes, les L-systèmes, les modèles à base d'agents comme l'intelligence en essain ou les automates cellulaires) à produire des fonctions de réparation. Ainsi, nous chercherons à approfondir nos connaissances sur ces techniques et à identifier les outils existants qui permettent de les modéliser facilement. Puis, nous expérimenterons de nouvelles fonctions de réparation sur des cas d'études issus de la pratique professionnelle.

Dans un deuxième temps, grâce aux conclusions tirées de ces expérimentations, et cette thèse devant aboutir à des résultats exploitable par nos collaborateurs architectes de l'agnecen, nous présenterons les développements réalisés pour la création d'un plugin Grasshopper® dédié à l'optimisation multicritère sous contraintes principalement axé sur la réparation de solution. Cet outil est le résultat de l'accumulation des scripts python codés à l'occassion des expérimentations sur des projets de l'agence Architecture Studio et des études comparatives présentées précédemment. Ce plugin nommé « *Urchin* » sera décrit en détail, dans une partie de ce chapitre s'apparentant à un guide d'utilisation.

Dans un troisième temps, nous présenterons des applications pour ce nouveau plugin issues de la littérature scientifique ou inspirée de la pratique. Cet outil est le fruit de recherches principalement réalisées sur des problèmes d'enveloppes. Nous savons que ce sujet est particulièrement accessible comparé à d'autres. Ainsi, cette dernière partie sera l'occassion de confronter nos solutions techniques à des problèmes de conception plus complexes.

4.1. Différentes techniques génératives pour différents types de contraintes

Différents types de techniques génératives pourraient faciliter l'intégration des contraintes dans un processus d'optimisation. Nous explorons dans cette première partie du chapitre 4 uniquement les plus connues en conception computationnelle (les grammaires de formes, les L-systèmes, l'intelligence en essaim et les automates cellulaires).

Dans un premier temps, nous revenons sur la définition de ces techniques et faisons un état des lieux des outils déjà à disposition des architectes pour manipuler ces techniques génératives. L'objectif de ce dernier est de voir s'il existe suffisamment d'outils pour pouvoir définir des fonctions de réparation sans passer par l'utilisation d'un langage de programmation, ou inversement, pour constater s'il existe un manque, et donc éventuellement un besoin en outils spécifiques.

Dans un second temps, les expérimentations n'ayant pu aboutir à la mise en œuvre de processus d'optimisation, faute d'incapacité à modéliser les contraintes, sont retravaillées et testées avec la méthode de réparation. Ces applications issues de la pratique professionnelle sont réalisées en dehors des calendriers des projets. Elles nous permettent d'affiner un cahier des charges pour un plugin Grasshopper destiné à démocratiser l'optimisation sous contraintes.

4.1.1. Les différentes techniques génératives

Dans cette section, nous proposons une description des techniques génératives les plus couramment utilisées en architecture, en plus des algorithmes génétiques. Nous avons déjà évoqué ces techniques au chapitre 1 (p. 83) et donné une première définition pour certaines d'entre elles ; nous souhaitons ici approfondir quelques notions.

La notion de boucle conditionnelle

Définition

Nous avons déjà donné une définition de la conception générative (CG) au début de cette thèse (p51). Nous rappelons qu'il convient de parler de CG lorsqu' « *un processus itératif où une instruction est exécutée en boucle et s'arrête lorsqu'un critère précis est satisfait* », ce qui, en informatique, se nomme une « boucle conditionnelle ». Ces boucles ne sont pas toujours faciles à utiliser lorsqu'on ne pratique pas de langage informatique car elle implique l'usage de la récursivité. Celle-ci n'est, à l'origine, pas inclue dans les langages de programmation visuelle comme Grasshopper® qui est « *plus un moteur de modélisation paramétrique que de modélisation algorithmique* » (De Boissieu & Guéna, 2012).

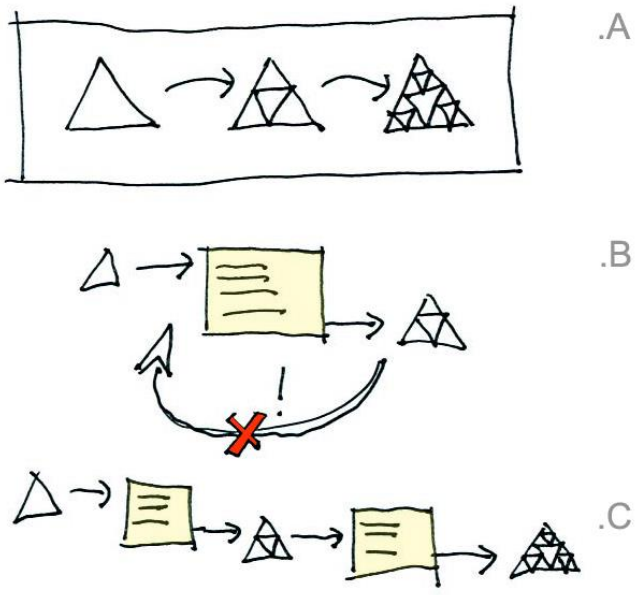


Figure 274 : Illustration du caractère acyclique de Grasshopper (source: (De Boissieu & Guéna, 2012))

En effet, les graphes sur Grasshopper sont dits acycliques. Pour pouvoir faire une simple boucle comme une fractale illustrée avec le schéma A sur la Figure 274, une même entrée de données ne peut être utilisée plusieurs fois (comme le montre le schéma B). Il est nécessaire de

copier la fonction (ici, qui dessine un triangle dans un triangle) plusieurs fois comme détaillé dans le schéma C. Cette caractéristique des graphes sur Grasshopper® pose problème lorsqu'on ne connait pas à l'avance le nombre d'itérations qui seront nécessaires à la satisfaction d'une condition, ou lorsque le nombre d'itération est très grand.

Pourtant certaines contraintes rencontrées sur des problèmes d'optimisation pourraient être résolues avec des boucles conditionnelles très simples. Prenons l'exemple de la conception d'un îlot fermé qui serait contraint d'avoir un cœur d'îlot en partie ensoleillé au moins 2 heures par jour, tous les jours de l'année. Nous pourrions créer une fonction de réparation qui, à chaque itération, agrandit de 5 % la taille de l'empreinte du cœur d'îlot ou, alternativement, réduit la hauteur des bâtiments et évalue les heures d'ensoleillement pour vérifier si la condition est satisfaite ou si une nouvelle itération est nécessaire. Dans ce cas, nous appellerons ce type d'algorithme un modèle génératif simple, ou « *à base de règles* », car il ne nécessite pas l'usage d'une technique générative spécifique. En effet, si certains cas peuvent être réglés avec une boucle conditionnelle simple, d'autres nécessitent l'usage d'algorithmes plus complexes.

Les techniques génératives plus complexes

Les chercheurs V. Singh et N. Gu ont identifiés et comparés les approches computationnelles les plus couramment utilisées dans le monde de l'architecture (Singh & Gu, 2012). En dehors des algorithmes génétiques (AG), ils identifient quatre techniques de design génératifs. Ces techniques sont usuellement classées en deux catégories (Oxman, 2006) : les méthodes évolutionnaires, (inspirées par des phénomènes naturels) que sont les automates cellulaires (AC) (Von Neumann & Burks, 1966) et l'intelligence distribuée, utilisée dans les MBA ou systèmes multi-agents (Ferber, 1997) ; et les méthodes linguistiques reposant essentiellement sur les grammaires de formes (Stiny, 1980) ainsi que les L-system (LS) (Lindenmayer, 1968).

V. Singh et N. Gu ne sont pas les seuls à intégrer ces techniques dans la famille des processus génératifs. En 2020, Caetano et ses co-auteurs ont publié une taxonomie pour un ensemble de termes clés de la conception computationnelle à partir d'une littérature exhaustive. Ils y précisent notamment à propos de la définition du design génératif que « *plusieurs auteurs considèrent des approches telles que la génération algorithmique, les automates cellulaires, les méthodes évolutionnaires, les grammaires génératives et de forme, les L-systèmes, l'auto-organisation, les modèles basés sur les agents et les systèmes en essaim, comme faisant partie de la conception générative* » (Caetano et al., 2020).

Avec les recherches présentées dans la troisième partie du chapitre 2, nous pouvons désormais ajouter à cette liste l'usage des techniques de Machine Learning. Les réseaux de neurones antagonistes peuvent être utilisés pour faire du design génératif, avec des applications principalement en 2D puisqu'il s'agit de génération d'images.

Faire des boucles avec un langage de programmation visuelle

Il existe plusieurs plugins Grasshopper® qui permettent de faire des boucles conditionnelles comme Hoopsnacke® disponible depuis 2012 ou encore Anemone® mis en ligne en 2013. Cependant, ces outils ne peuvent être utilisés pour faire de la réparation de solution car ils n'ont pas été programmés pour fonctionner avec les solveurs d'optimisation actuels qui n'attendent pas que la boucle conditionnelle soit terminée pour récupérer les informations de la solution réparée.

Ainsi, le plugin Octopus®, qui héberge un solveur du même nom, contient un composant permettant de définir des boucles conditionnelles appelé « octopusLoop ». Il permet de faire le calcul en arrière-plan et ne transmet les données de sortie que lorsque la boucle est terminée. Ainsi, ce composant peut être utilisé avec des solveurs comme Galapagos®, Octopus® ou autres. Toutefois, ces solveurs ne permettent pas de faire de la réparation lamarckienne (voir définition au chapitre 3 p. 335).

Les méthodes linguistiques

Les grammaires de formes

Pour rappel, une grammaire de forme (GF) est une méthode qui consiste à faire émerger des compositions en appliquant successivement un ensemble de règles de transformation à une forme de départ, appelée forme initiale (Knight, 2003). Knight et Stiny ont illustré l'applicabilité de ce processus à l'architecture à partir d'une forme et de règles très simples : le rectangle, et sa rotation à 90°. Les formes générées résultantes sont présentées en Figure 275.

Cette technique n'a pas toujours été assistée par un ordinateur. Elle était à l'origine utilisée à des fins pédagogiques pour faire de l'analyse historique des styles en architecture avant de servir à la production architecturale ou, encore, comme un outil d'aide à la conception (Tepavčević & Stojaković, 2012).

Une récente revue de littérature sur les GF appliquée à l'ingénierie et à l'architecture recense 83 publications sur 50 années de recherche (Haakonsen et al., 2023). Les auteurs ont identifié 5 types d'applications : le détail du bâtiment, la façade, le plan d'étage, la forme globale du bâtiment ; et les autres. La plupart des études cherchent davantage à développer des

GF plutôt que d'analyser l'architecture existante, et l'application la plus répandue dans cette revue de littérature est la génération de plans. Elles sont aussi très utilisées pour la génération procédurale de ville notamment avec le logiciel CityEngine (Ghorbania & Shariatpour, 2021) ou la reconstruction 3D (Mathias et al., 2011)La plupart des publications datent d'après l'année 2010 et quelques publications intègrent de l'optimisation notamment depuis 2016 (avec 7 références sur les sujets d'architecture), ce qui s'expliquerait par l'augmentation de l'intégration des interpréteurs (nom donné aux outils pour faire de la GF).

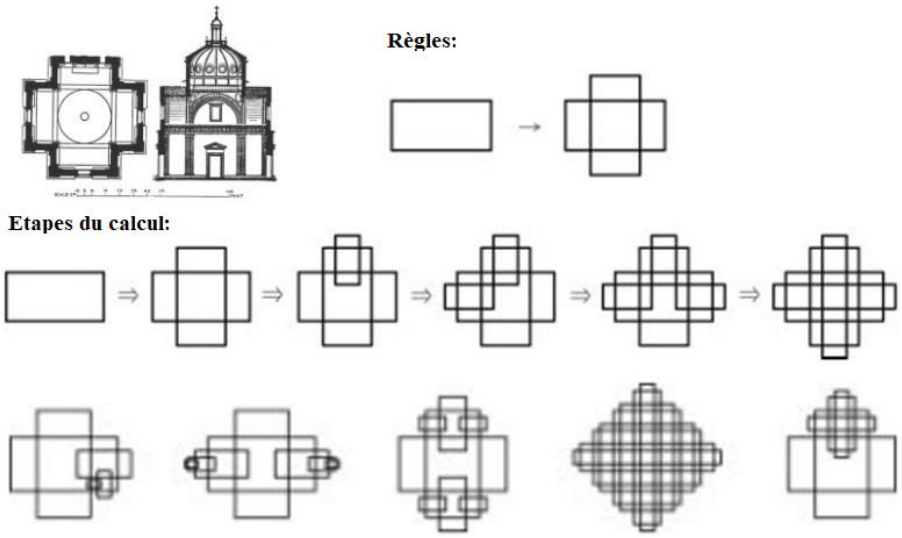


Figure 275 : Exemple d'une grammaire de forme (source : (Knight & Stiny, 2001))

Ainsi, bien que nous n'ayons pas à ce jour eu l'occasion d'utiliser les GF pour créer de fonction de réparation, ou pour créer des espaces de solutions sans solutions infaisables, nous savons que les GF permettent de créer des espaces de solutions pour des problèmes aussi complexes que l'allocation spatiale (voir Figure 276).

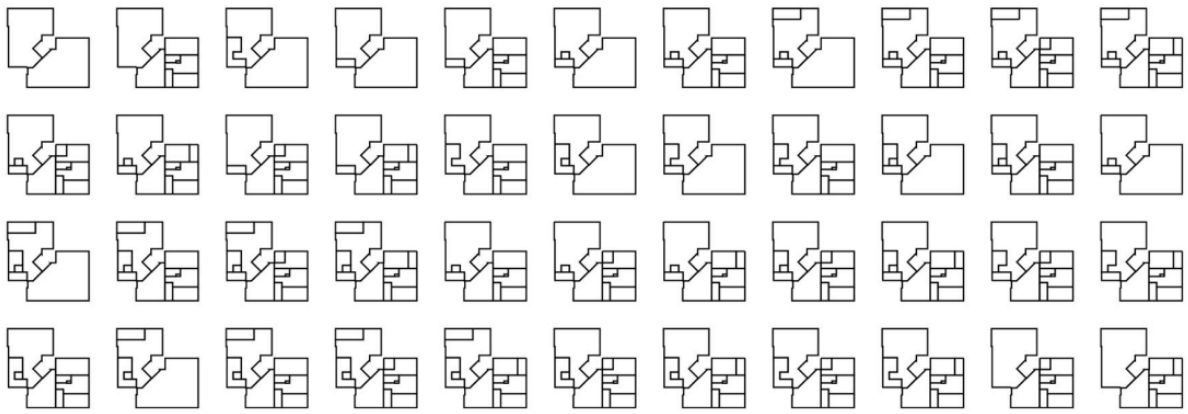


Figure 276 : 44 plans générés avec une GF (source : (Veloso et al., 2018))

Dans la littérature scientifique, on trouve différents exemples d'utilisation de GF pour la performance environnementale, mais elles peuvent aussi être utilisées uniquement comme processus génératif, comme la génération de variantes de façade (Bao et al., 2013). Elles sont aussi utilisées pour la conception de systèmes de protection solaire intelligents (Ceranic & Nguyen, 2018; Kotsopoulos et al., 2012; Tomasowa & Sjarifudin, 2017). Cette technique a été utilisée par les ingénieurs pour résoudre des problèmes de dimensionnement, notamment pour la ventilation d'une façade double peaux (Ashrafi, 2017), ou l'allocation des ouvertures en façade en fonction de l'éclairage naturel (Alshoubaki et al., 2016). Certains utilisent des GF pour générer des maisons individuelles selon un style d'architecture et trouver les solutions les plus efficaces d'un point de vue thermique (Caldas, 2008; Granadeiro et al., 2013). Enfin, en 2017, M. Muehlbauer, J. Burry et A. Song ont proposé d'utiliser une *grammaire évolutionnaire*, pour automatiser la génération de formes sans la séparer de l'exploration de solution. Cette approche mêlant GF et algorithmes génétiques permet de générer des ensembles de solutions pré-optimisés (Muehlbauer et al., 2017).

Les L-systèmes

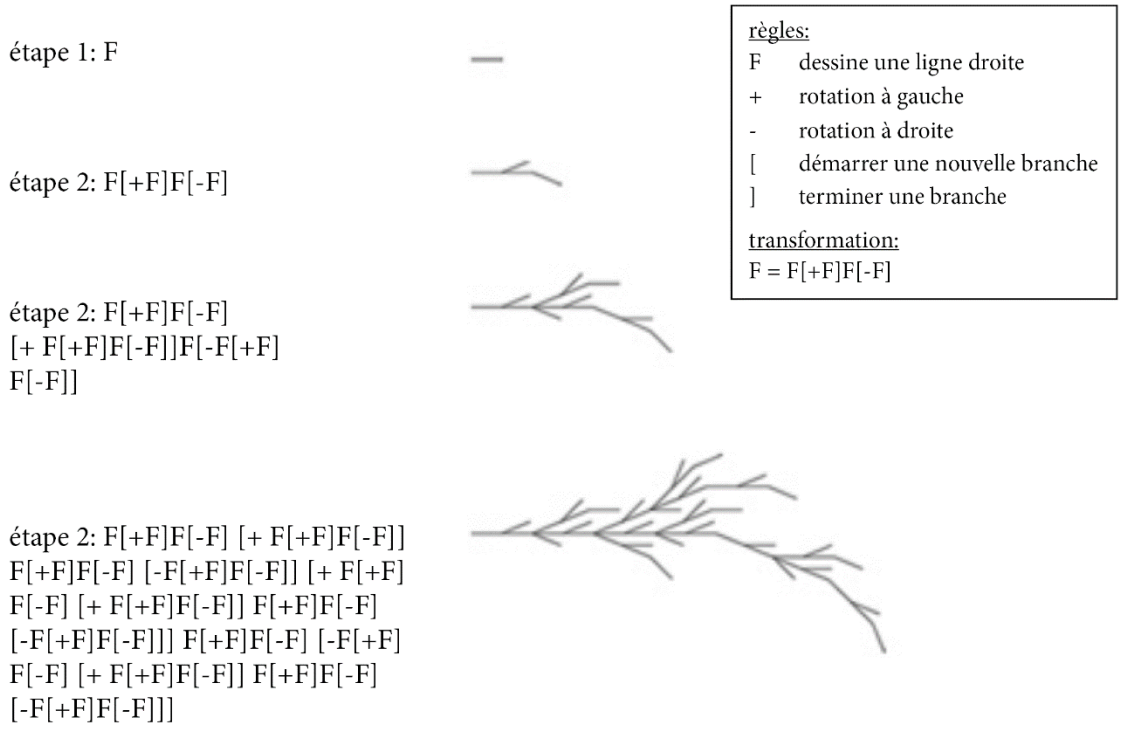


Figure 277 : Exemple de L-système (source: https://stackoverflow.com/questions/29793849/how-to-create-a-simple-l-system-in-processing)

La seconde méthode linguistique la plus répandue en architecture computationnelle est le L-système (LS) (Lindenmayer, 1968). Il s'agit d'une grammaire formelle (méthode informatique à différencier de la grammaire de forme) inventée par un biologiste qui lui a donné son nom : Lindenmayer. Il s'agit d'un système de réécriture, c'est-à-dire un modèle qui cherche à transformer une syntaxe en appliquant des règles précises. Cette grammaire particulière permet de modéliser le processus de croissance des plantes, comme dans l'exemple présenté en Figure 277.

Ce type de processus génératifs est principalement utilisé à l'échelle urbaine pour la génération de réseaux viaires, comme nous l'avons vu au chapitre 1 (p 103). Nous n'avons pas, pour le moment, utilisé de LS pour gérer des contraintes sur un sujet issu de la pratique, mais nous n'éliminons pas l'idée qu'ils puissent servir à la définition d'une fonction de réparation, notamment pour la génération de réseau viaire.

Plugins Grasshopper® pour implémenter des méthodes linguistiques

Plusieurs outils sont disponibles gratuitement pour implémenter des GF ou des LS sur Grasshopper®. Ils peuvent être utilisés facilement par les architectes pour faire du design génératif. Rupa® est un plugin Grasshopper développé par Alva Sondakh pendant sa thèse de doctorat. Il s'agit d'un unique composant, conçu pour être un assistant à la conception de grammaires de forme. Il permet de contraindre des formes paramétriques en 3D (inclusion dans la parcelle, collisions, mitoyenneté). Avec cet outil, il est possible d'explorer la morphologie d'un bâtiment, ou de plusieurs, à la géométrie relativement complexe (pas uniquement des parallélépipèdes) dans une parcelle deforme complexe.

Figure 278 : Exemple de forme générer avec trois rectangles

SortalGI est un interpréteur de grammaires de forme qui a été en partie implémenté sur Grasshopper® en 2022. Il permet de traiter des lignes, courbes, des couleurs en 2D ou 3D. Il est composé de 74 composants permettant d'imaginer de nombreuses applications aux grammaires de formes.

Si des LS peuvent être facilement reproduits avec les plugins permettant de faire des boucles conditionnelles, il existe des plugins qui permettent de faciliter la programmation de LS dans Grasshopper®. Le plugin Leaf® lancé en 2022 permet avec 5 composants de générer très facilement des formes très complexes. La géométrie initiale peut être n'importe quelle géométrie (ligne, courbe, surface, ou volume). Nous avons testé le plugin avec une pyramide pentagonale dont les résultats et le script sont présentés en Figure 279.

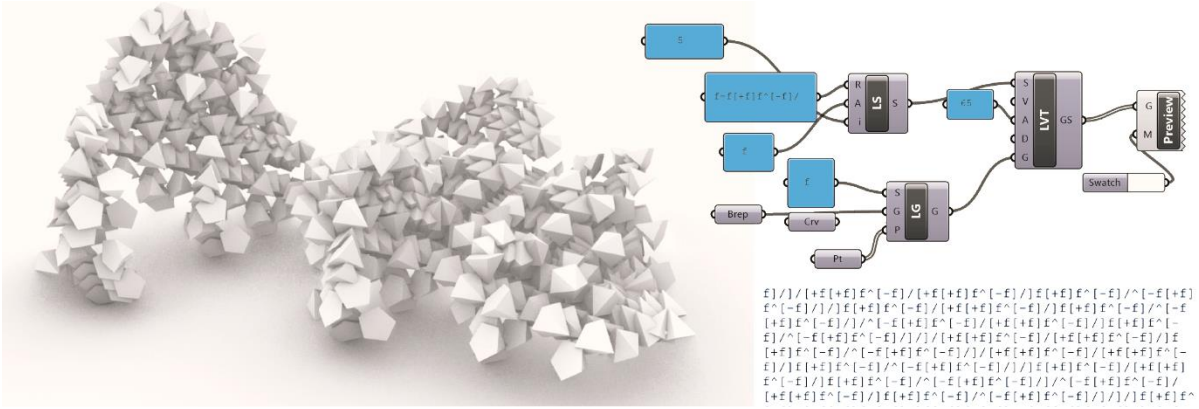


Figure 279 : Test du plugin Leaf® pour la génération avec des LS avec une pyramide pentagonale

Le plugin Dragonturtle® développé depuis 2022 est un plugin pour faire du design génératif qui contient notamment des composants permettant de modeler des fractales. Il contient aussi 4 composants permettant d'utiliser des LS.

Les méthodes à base d'agents

Les modèles à base d'agents mobiles

Il existe de nombreux termes qui font référence aux modèles à base d'agents (MBA) : « *système d'auto-organisation* », « *modélisation à base d'agents* », « *simulations à base d'agents* », « *systèmes multi-agents* ». Selon C.M. Macal, tous ces termes renvoient à des méthodes inspirées de l'intelligence artificielle distribuée (Macal, 2016). Le terme MBA est utilisé dans de nombreuses disciplines et sa définition peut varier. Un MBA est un processus qui permet de résoudre des systèmes complexes constitués de multiples entités en interaction appelées agents. En exécutant des règles simples en fonction d'informations locales, les agents

peuvent atteindre un objectif global en coopérant entre eux et en interagissant avec leur environnement. Les agents doivent être des individus modulaires identifiables, autonomes, sociables, puisqu'ils interagissent entre eux ; et qui ont un état qui peut varier au cours du temps (Macal & North, 2005).

Figure 280 : Vols d'oiseau, bancs de poissons et colonies de fourmis (source : (Lecheval et al., s. d.), www.nationalgeographic.com)

L'intelligence en essaim (IE), les automates cellulaires (AC) peuvent être perçus comme des MBA particuliers disposant de caractéristiques distinctes. L'IE est une notion utilisée pour la première fois par Gerardo Beni (Beni, 2004) pour décrire des MBA fortement bio-inspirés où les caractéristiques des agents rappellent celles des insectes (et autres animaux sociaux, voir Figure 280) et où, à la différence de l'AC, le phénomène central réside dans le déplacement collectif d'agents simples et quasi-identiques.

L'IE peut être employée de façon très littérale en s'appuyant sur des comportements d'animaux sociaux comme les oiseaux, les fourmis ou les bancs de poissons. Il est aussi possible d'imaginer ses propres algorithmes avec des règles définies pour atteindre des objectifs précis ; comme, par exemple, pour gérer des contraintes de collisions entre les agents qui seraient des empreintes de bâtiments.

C'est ce qu'ont fait les architectes de l'agence Morphosis pour la conception de façade de la cour du Emerson College Los Angeles Center où des panneaux en aluminium sont plus ou moins pliés de façon à obtenir une protection solaire hétérogène adaptée aux besoins du bâtiments (voir Figure 281). Le modèle à base d'agents est utilisé pour éviter les collisions entre les panneaux et assurer un effet de continuité dans leur disposition. Du côté des chercheurs, les MBA sont parfois utilisés en conception générative et performancielle, notamment pour l'optimisation des systèmes de couverture (Pantazis & Gerber, 2020; Parascho, 2013), des systèmes de protections solaires (Gerber et al., 2017), ou de production électrique en façade (Zarrabi et al., s. d.). A une plus grande échelle, les MBA ont été utilisés pour dessiner les

circulations dans un parc (Vegas et al., s. d.) ou la disposition de plusieurs bâtiments (Y. K. Yi & Kim, 2015).

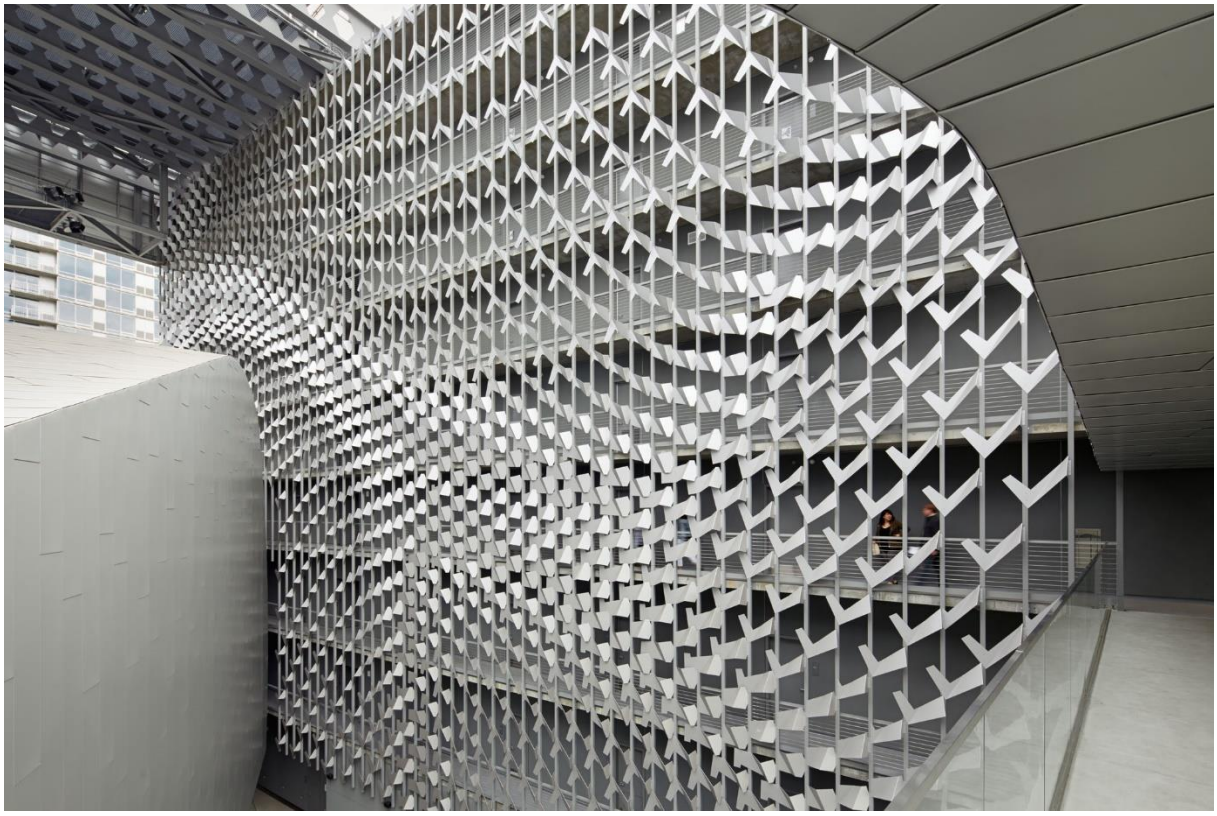


Figure 281 : Façade intérieure de l'Emerson College Los Angeles Center de Morphogenesis (source: https://www.brucedamonte.com/projects/emerson-college/)

Les automates cellulaires

Les AC sont des MBA particuliers où les agents sont disposés sur une trame orthogonale dont la dimension (D) peut être de 2 ou 3. Les agents d'AC, appelés « *cellule*s » ne sont pas préalablement définis (Von Neumann & Burks, 1966). L'évolution de l'état d'une cellule impacte l'état des cellules voisines situées dans un rayon (R). A partir des paramètres D, K et R, on définit un automate cellulaire pour lequel il existe un nombre fini de règles applicables.

Le jeu de la vie, crée par John Horton Conway en 1970 est un exemple connu d'AC. Il se joue sur une grille à deux dimensions. Les cellules peuvent prendre deux états : vivantes (cellules blanches), ou mortes (cellules noires). La taille du rayon est d'une cellule, c'est-à-dire que les huit cellules voisines peuvent impacter l'état de l'agent. Le jeu de la vie est constitué de deux règles simples : (i) à chaque itération une cellule morte possédant exactement trois cellules voisines vivantes devient vivante, (ii) une cellule vivante possédant deux ou trois cellules

voisines reste vivante, sinon elle meurt. Le jeu de la vie permet notamment de générer des motifs comme ceux présentés dans la Figure 282, identifiés à l'aide d'un algorithme génétique (Alfaro, 2009).

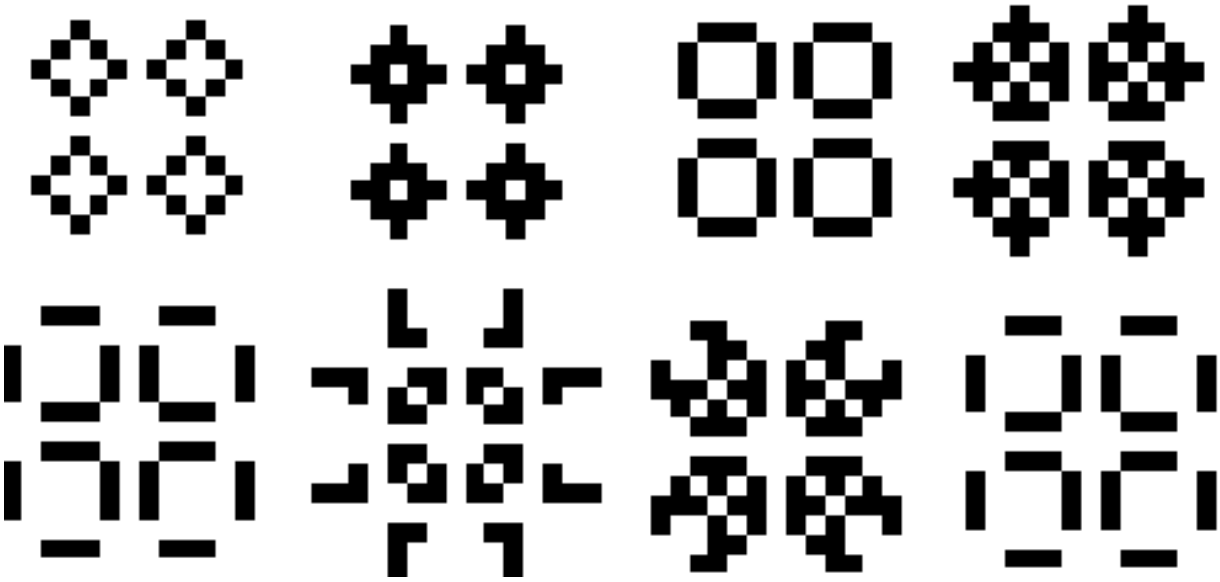


Figure 282 : Exemples de patterns créés avec le jeu de la vie trouvés à l'aide d'un algorithme génétique (source : (Alfaro, 2009))

Les AC sont utilisés par des artistes pour générer des objets 3D. L'architecte computationnel Michel Hansmeyer a notamment utilisé les AC pour concevoir des composants de structures pour des volières, dont les résultats sont présentés en Figure 283.

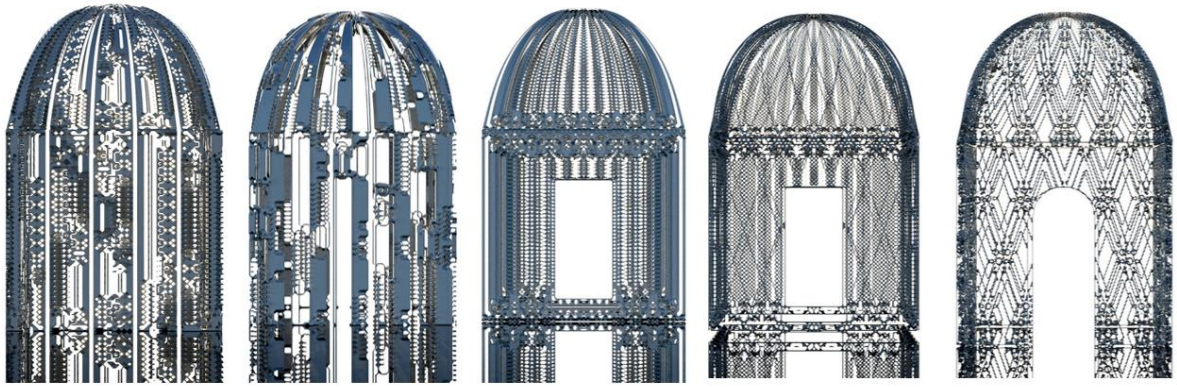


Figure 283 : Voxel-based volières, de Michael Hansmeyer (source: https://www.michael-hansmeyer.com/voxels)

Les AC ont été largement utilisés en conception générative et environnementale. Cependant, la plupart des auteurs les utilisent pour leur faculté d'émergence et pour l'esthétique que ces méthodes sont capables de générer plutôt que pour intégrer des objectifs

environnementaux. Ainsi, M. Zawidzki propose en 2009 une façade adaptative à partir d'un AC où des films polarisants permettent de contrôler les apports solaires (Zawidzki, 2009). A plusieurs reprises, les AC ont été utilisés pour la conception de façade, notamment pour contrôler l'éclairage naturel (Alshoubaki et al., 2016; F. Fathy et al., 2015; Kim, 2015).

Plugins Grasshopper® pour implémenter des modèles à base d'agents

Les architectes ont aujourd'hui de nombreux outils à disposition pour faire de la modélisation à base d'agents, notamment des plugins Grasshopper® (bien qu'il existe aussi Toxiclibs pour Processing® et Flower Power pour Rhinocéros®). La majorité d'entre eux ont été créés pour faire de la *swarm intelligence* (mots anglais pour IE). Ces plugins ont, pour la plupart, un fonctionnement similaire (Boid®, Zebra®, Faerie®, Physarealm®, Quelea®, Termite_ABMS_toolkit®, Culebra®, Nusery®, ABxM®) ; ils permettent de faire de la simulation particulaire, où les agents sont des particules, soit des points 3D en géométrie Grasshopper®.

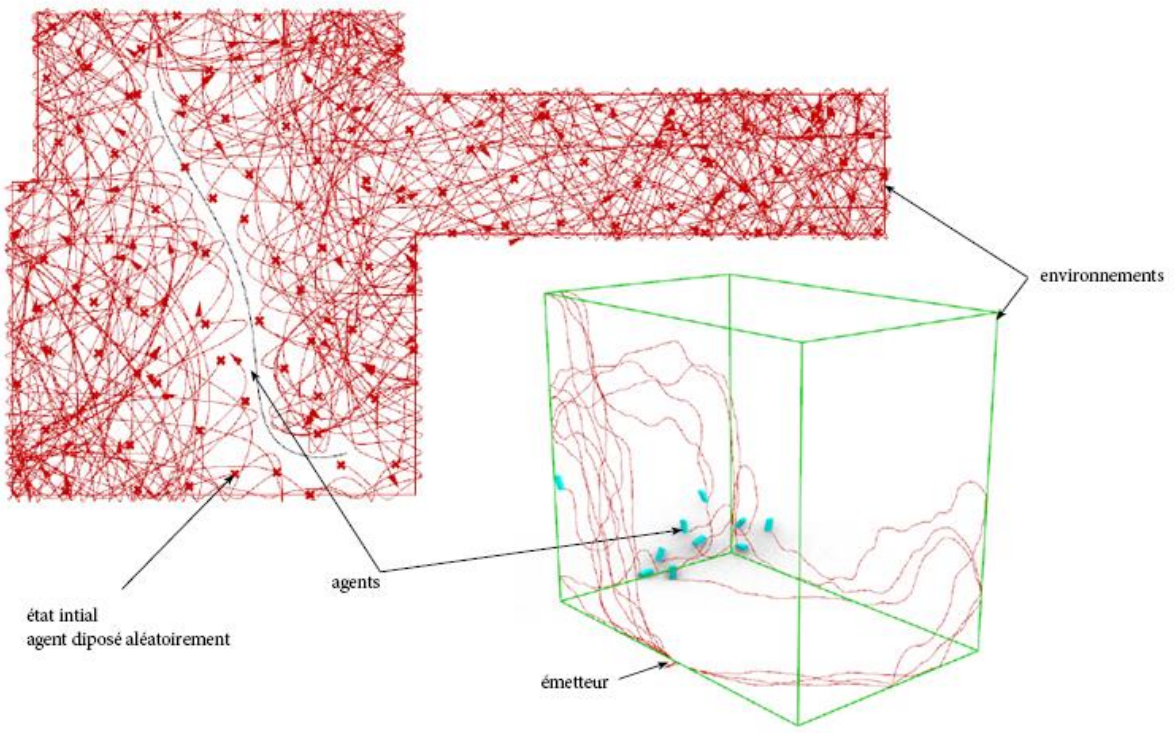


Figure 284 : MBA crée avec Quelea à droite et avec Zebra à gauche

Quatre types de composants sont nécessaires pour créer un MBA avec ce type de plugins :

(1) Les composants qui permettent de définir l'état initial des agents (des points), qui peuvent être disposés soit manuellement, soit aléatoirement, soit en utilisant des

émetteurs (qui peuvent être des points, des courbes, ou des surfaces) qui génèrent des particules.

(2) Les composants qui permettent de définir les règles d'interactions entre agents, aussi appelés *behaviors*, soit les comportements en anglais. Ces composants varient selon les plugins. Les comportements les plus courants sont l'attraction, la répulsion ou encore l'alignement.

(3) Les composants qui permettent de définir des règles d'interaction avec l'environnement. Les particules peuvent être cantonnées à une géométrie (un cube, une surface, une polysurface fermée), ce que certains plugins appellent *environnement*. Elles peuvent aussi interagir avec des objets géométriques placés dans cet environnement avec des règles comme l'évitement des collisions, l'attraction à ces objets, ou la révolution autour de ces objets.

(4) Les composants qui permettent de lancer la simulation (le solveur) et d'en exploiter les résultats, par exemple en conservant les traces du passage des particules.

Certains plugins sont plus aboutis que d'autres. Boid® par exemple n'a pas de solveur, il faut utiliser le plugin Anemone® pour lancer les simulations. Seuls les plugins Physarealm® et Quelea® ont des émetteurs. Faerie® permet de faire de la simulation de pluie. Culebra® est le plus abouti en ce qui concerne la visualisation (voir Figure 285), et plus adapté à un usage créatif de la MBA plutôt qu'à l'optimisation ou l'évaluation.

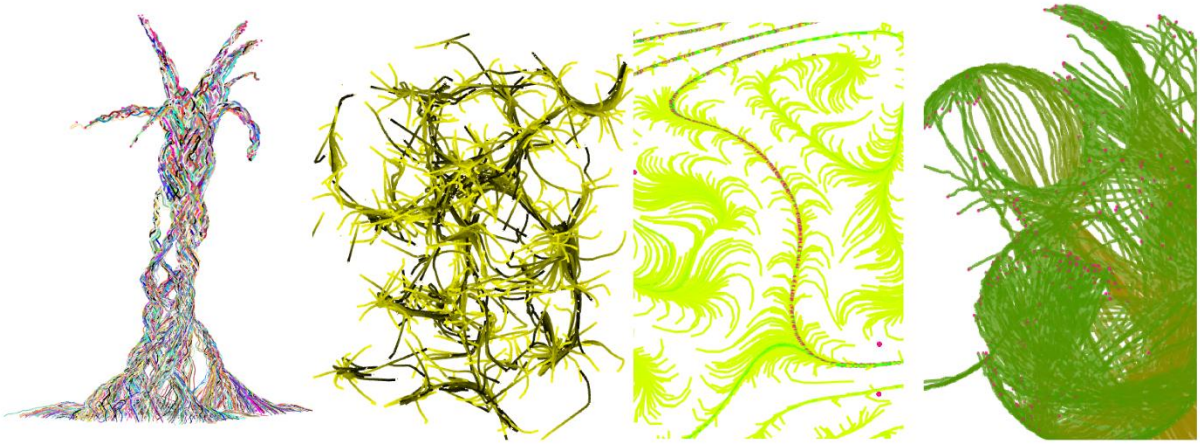


Figure 285 : Géométrie générée avec le plugin Culebra

Si ces outils permettent bien de faire de la modélisation à base d'agents, ils ne sont pas particulièrement utiles pour programmer des fonctions de réparation. En effet, si on reprend

l'exemple des contraintes de collision pour l'allocation spatiale de bâtiments sur une parcelle, mettre en interaction des particules n'est pas la méthode adaptée. Dans ce cas, nous avons besoin de mettre en interactions des courbes fermées représentant les empreintes des bâtiments ou des volumes fermés représentant les bâtiments eux-mêmes. Seul le plugin Nusery® permet de mettre en interaction différents types de géométries. Toutes les géométries Grasshopper® peuvent être définies comme un agent (voir Figure 286). Ce plugin compte 77 composants dédiés à la MBA, dont 28 pour définir des *behaviors*. Lorsque plusieurs *behaviors* sont utilisés, ils peuvent être hiérarchisés à l'aide de coefficients de pondération. Nusery® est, à l'heure actuelle, le plugin le plus abouti sur Grasshopper® pour faire de la modélisation à base d'agents.

Cependant, les *behaviors* de Nursery® ne peuvent être utilisés sur tous les types de géométrie, notamment les règles d'attraction et de répulsion ne fonctionnent qu'avec les particules. Ainsi, il est mal adapté à nos problèmes de réparation. Il ne permet pas, par exemple, de modéliser facilement le problème de collision des lames présenté au chapitre précédent.

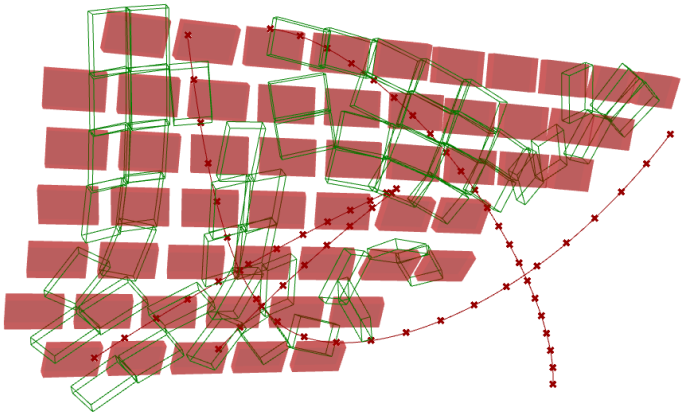


Figure 286 : MBA pour aligner des briques le longs de courbes 3D avec le plugin Nusery

Kangaroo Physics® est un plugin pour faire de la modélisation physique, bien qu'il ne soit pas présenté comme tel, il peut aussi être utilisé pour créer des MBA puisqu'il s'agit de modélisation particulaires. Il contient un solveur spécial (Kangaroo Zombie) qui peut être utilisé avec des plugins d'optimisation.

Le plugin le plus récent est ABxM®, il date de juillet 2022 et compte 25 composants. Il a été développé par des chercheurs de l'université de Stuttgart pour des applications à des problèmes de structure et de robotique pour la construction (Groenewolt et al., 2018), comme le montrent les images présentées en Figure 287 issues de leur site internet. Ce plugin est particulièrement adapté à l'optimisation pour la discrétisation de surfaces courbes. Deux types de modèles peuvent être utilisés, soit avec des particules (*boid system)*), soit avec des cellules (*matrix system*), ce qui permet de modéliser des AC.

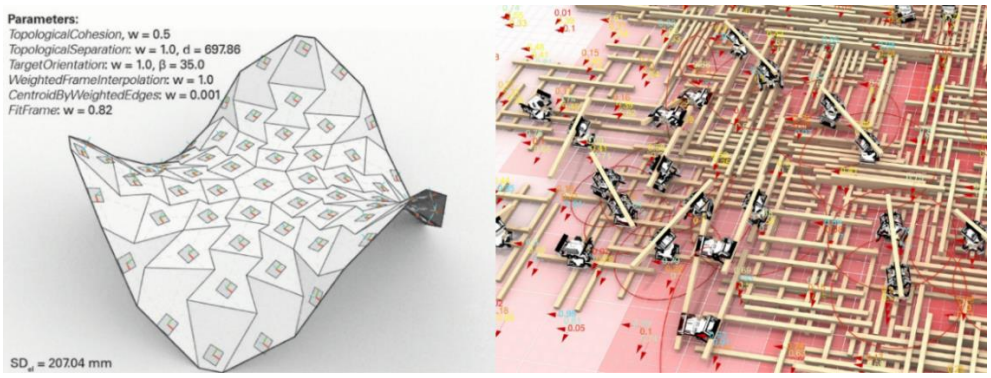


Figure 287 : Exemples d'applications du plugin ABxM (source: https://www.icd.uni-stuttgart.de/research/research-tools/abxm-framework/)

Quatre autres plugins Grasshopper permettent de créer des AC (Rabbit®, Peacock Mantis Shrimp®, Kthulhucat® et Bee®), à une, deux ou trois dimensions (voir Figure 288). Seulement ils ne supportent que deux états possibles (cellule vivante ou morte), avec un rayon de 1 uniquement (soit 2 voisins en 1D, 8 en 2D, et 26 en 3D). Nous le verrons dans la suite de ce chapitre, ces AC ne sont pas suffisamment complexes pour permettre de définir des fonctions de réparation utiles aux problèmes de conception architecturale.

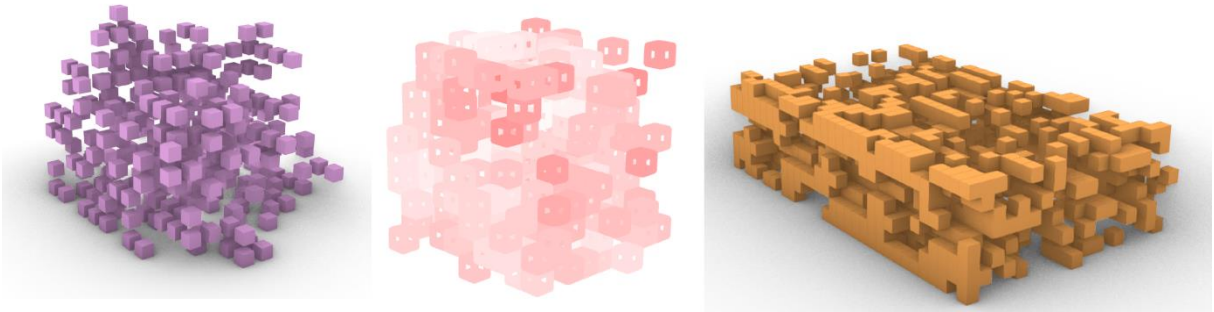


Figure 288 : AC générés via Kthulhucat®, Peacock Mantis Shrimp® et Bee®

Finalement, s'il existe de nombreux outils à la disposition des architectes pour faire de l'IE ou pour la modélisation d'AC, ils n'ont pas été développée pour permettre de programmer des algorithmes de réparation, et ne sont donc pas adaptés à cette tâche spécifique. Nous pouvons désormais confirmer qu'il manque un outil pour faciliter la programmation de modèles génératifs pour l'intégration des contraintes avec des algorithmes évolutionnaires multicritères. Avant de pouvoir en définir les contours, il semble nécessaire de nous confronter à de nouveaux problèmes d'optimisation sous contraintes issus de la pratique, notamment pour identifier les différentes formes que peuvent prendre ces algorithmes de réparation.

4.1.2. Applications issues de la pratique

Afin de vérifier la pertinence de la méthode d'intégration des contraintes qui consiste à utiliser des fonctions de réparation, nous avons testé notre méthode sur d'autres cas d'étude. Parmi les cas présentés dans le chapitre 2, certains n'avaient pas pu aboutir notamment parce qu'ils nécessitaient d'intégrer des contraintes. Nous avons de nouveau tenté de résoudre ces problèmes en utilisant des fonctions de réparation. Nous avons aussi cherché des problèmes d'optimisation sous contraintes parmi les anciens projets d'AS. Au total, quatre cas d'étude ont fait l'objet de nouvelles expérimentations.

Pour chaque cas d'étude nous avons lancé les explorations avec différents taux de remplacement du génome original afin de mesurer l'impact de ce paramètre sur la performance de l'algorithme. L'objectif principal de cette étude comparative était d'appréhender la robustesse de la méthode avec des contraintes de différentes natures, avec des fonctions de réparation inspirées de technique génératives différentes (simple boucle conditionnelle, automate cellulaire, swarm intelligence).

Boucle conditionnelle pour intégrer une contrainte environnementale

Description du cas d'étude

Dans la pratique, certains critères environnementaux peuvent apparaître comme prioritaires par rapport à d'autres. Les solutions n'atteignant pas un certain seuil (pour respecter une norme par exemple) seront alors considérées comme inexploitables par l'architecte. Il est alors préférable de considérer ce critère comme une contrainte et de s'assurer que toutes les solutions qui seront étudiées dans les processus évolutionnaires respectent bien cette dernière.

Cela peut s'apparenter à une pré optimisation qui permettrait d'assurer que le critère environnemental jugé le plus critique atteigne un seuil minimum. Showkatbakhsh and Kaviani ont d'ores et déjà décrit une méthode basée sur la performance se déroulant en 2 étapes : un premier processus paramétrique permettant d'assurer la protection solaire puis le processus génératif évolutionnaire avec des critères élargis (Showkatbakhsh & Kaviani, 2021).

Dans le cadre du concours, aujourd'hui gagné, pour la conception de la cité du ministère de la justice de Saint Laurent de Maroni, un système de protection de l'enveloppe nécessitait une optimisation pour réduire les apports solaires tout au long de l'année tout en assurant un éclairement naturel optimal dans les salles d'audience et les espaces de travail. Cette seconde peau devait aussi absolument protéger les bâtiments contre les intempéries, l'espace tampon

crée entre les deux peaux étant un espace de circulation. Un critère d'analyse de la quantité de matière a aussi été pris en compte dans le processus d'optimisation pour tenter de réduire les coûts de construction.

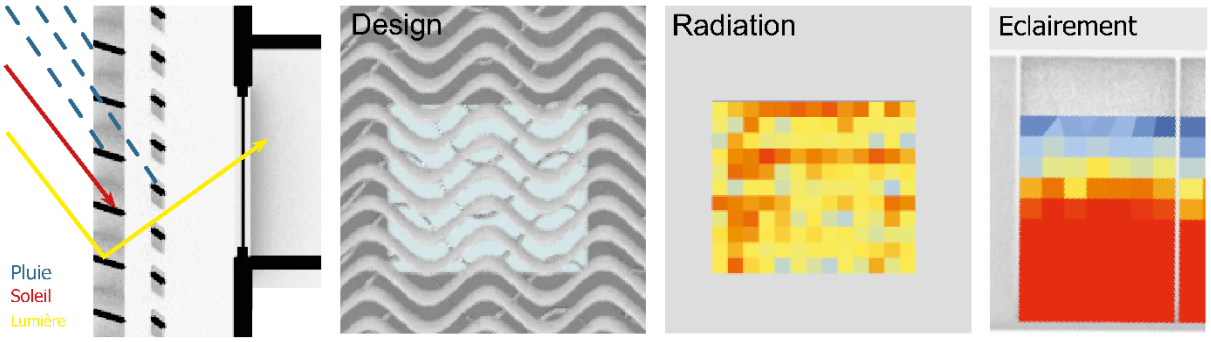


Figure 289 : Coupe, élévation, analyse de la radiation sur le vitrage et analyse de l'éclairement des espaces intérieurs

Pour cette première application, nous nous sommes inspirés de ce problème. Dans notre cas, les lames ne sont pas en bois mais en aluminium pour maximiser les chances de réflexion de la lumière (voir Figure 289). Les lames sont aussi courbes pour ajouter de la complexité et démultiplier les directions des rayons lumineux réfléchis.

Modélisation du problème

Le système de protection est composé de deux couches de lames en aluminium courbées formant des vagues favorisant la réflexion de la lumière. La distance entre les deux couches, le nombre de lames, le nombre de vagues par lames, leur amplitude, l'inclinaison et la largeur des lames sont les variables du modèle paramétrique, soit 11 variables au total, pour un espace de solutions de 54.6×10^{13} solutions. L'éclairement naturel a été calculé pour les pires conditions, le 21 décembre, à l'aide du plugin Honeybee® et du logiciel de simulation d'éclairage Radiance. La radiation solaire a été calculée à l'aide du plugin Ladybug® et les précipitations ont été simulées avec un système de lancer de rayons avec une inclinaison déduite à partir de la vitesse et de l'orientation des vents dominants locaux.

De nombreuses solutions créées par le modèle paramétrique ne protègent pas suffisamment la façade des précipitations. Nous avons donc conçu une fonction de réparation qui permet de modifier les quatre gènes qui ont le plus d'impact sur ce critère :

1. La distance entre deux lames de la couche 1
2. La distance entre deux lames de la couche 2

3. La largeur des lames de la couche 1

4. La largeur des lames de la couche 2

La fonction de réparation se déroule comme suit. Lors de la première itération, la valeur du gène (1) est déplacée vers le haut, puis une analyse de précipitation est effectuée avec cette nouvelle solution. Si la solution est satisfaisante, la réparation est jugée suffisante. Si la solution n'est pas satisfaisante, une deuxième itération a lieu et le gène (2) est modifié et une analyse de précipitation est effectuée, et ainsi de suite jusqu'au gène (4). Si aucune des solutions trouvées n'est satisfaisante, le processus recommence au gène (1). Le processus s'arrête lorsque la simulation vérifie que la façade est protégée contre la pluie. Pour plus de détail, le code python de la fonction de réparation est présenté en annexe 2.

L'architecture générale du modèle est décrite dans la figure ci-dessous :

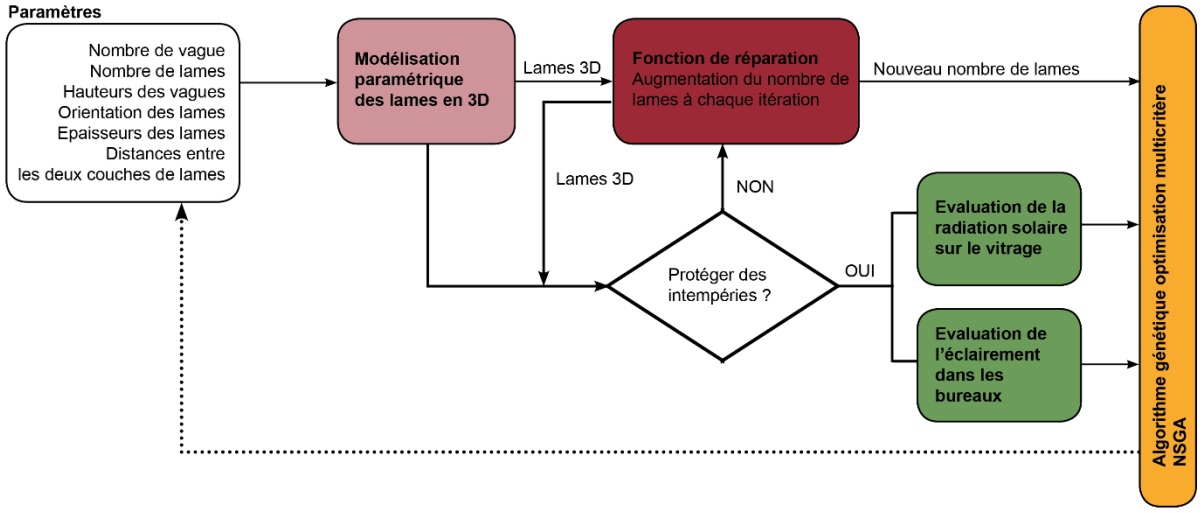


Figure 290 : Architecture générale du modèle pour le cas d'étude de St Laurent de Maroni

Résultats

Six explorations ont été calculées, dont une exploration complètement aléatoire pour servir de groupe contrôle. Les 5 autres ont été effectuées avec des taux de remplacement (du génome original avec le génome des solutions réparées) variable (100, 50, 15, 5 et 0%). Pour chaque optimisation, 10 générations de 30 individus chacune ont été calculées. Ainsi, pour chaque explorations, 300 solutions ont été générées, soit 1800 solutions. Quelques une de ces solutions sont illustrées en Figure 291.

Les résultats concernant la performance des explorations sont présentés dans le graphique en Figure 292.

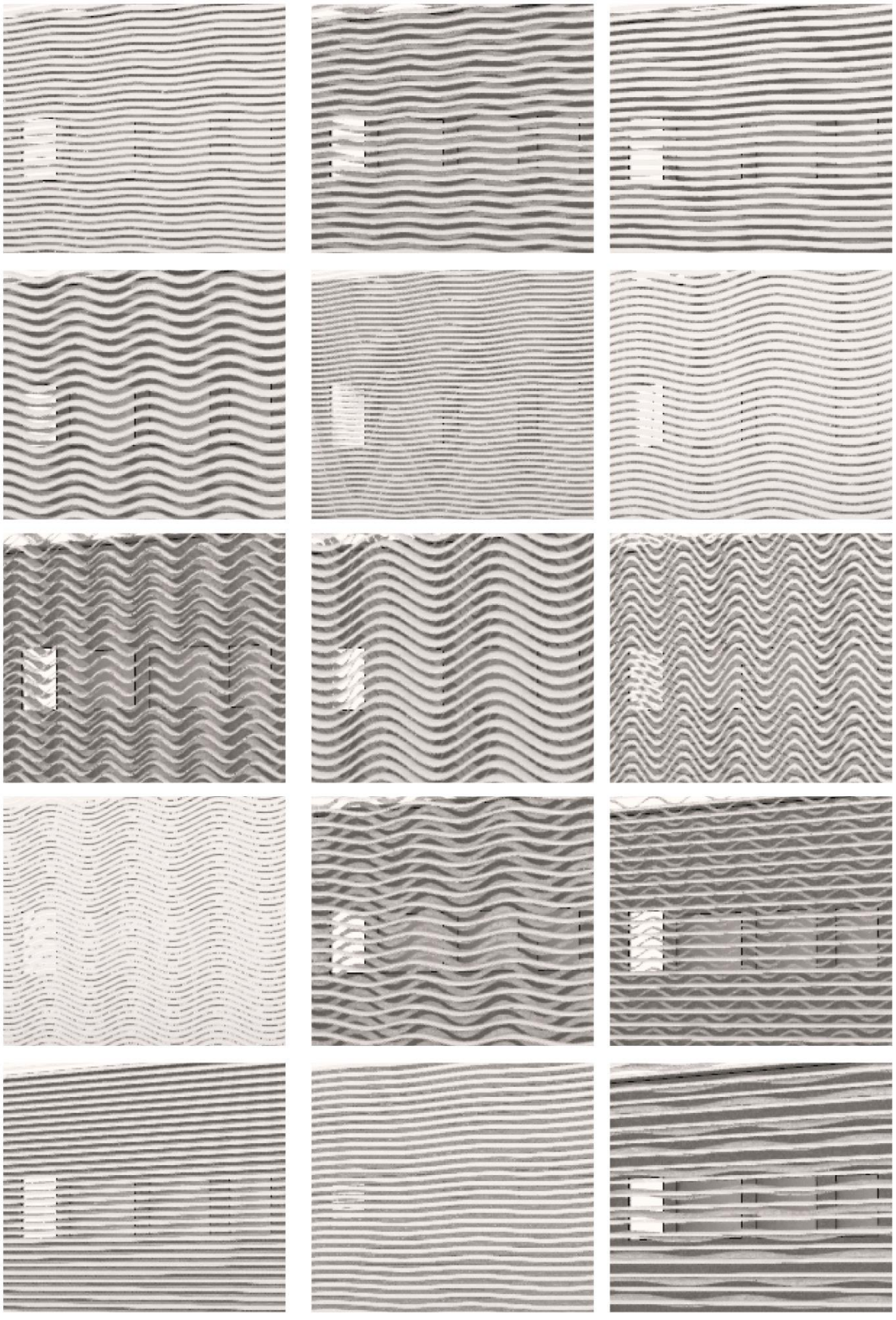


Figure 291 : Extrait de l'ensemble de solution pour le cas d’étude de Saint Laurent de Maroni

Pour comparer les approches et déterminer le taux de remplacement approprié pour trouver les meilleurs solutions (non-dominées), nous avons, comme pour le cas d'étude présenté au chapitre précédent, agrégé l'ensemble des solutions trouvées lors des 6 explorations et effectués un tri non-dominés similaire à celui effectué par NSGAII. Ainsi, sur la Figure 292, le rang 1 correspond aux solutions situées sur le front de Pareto, le rang 2 est constitué des solutions légèrement inférieures, etc.

Ce problème n'est pas très contraint puisqu'un peu plus de 36 % des solutions générées ne nécessitent pas d'être réparées. Quel que soit le taux de génotype originaux remplacés par les génotypes des solutions réparées, l'algorithme génétique arrive à converger puisque les meilleurs rangs comptent peu de solutions générées aléatoirement. Bien qu'il s'agisse d'optimisation combinatoire, c'est un remplacement de 100 % des génotypes qui est ici l'option la plus performante. On observe même que plus le taux de remplacement est important, plus le nombre d'individus dans les premiers rangs est élevé.

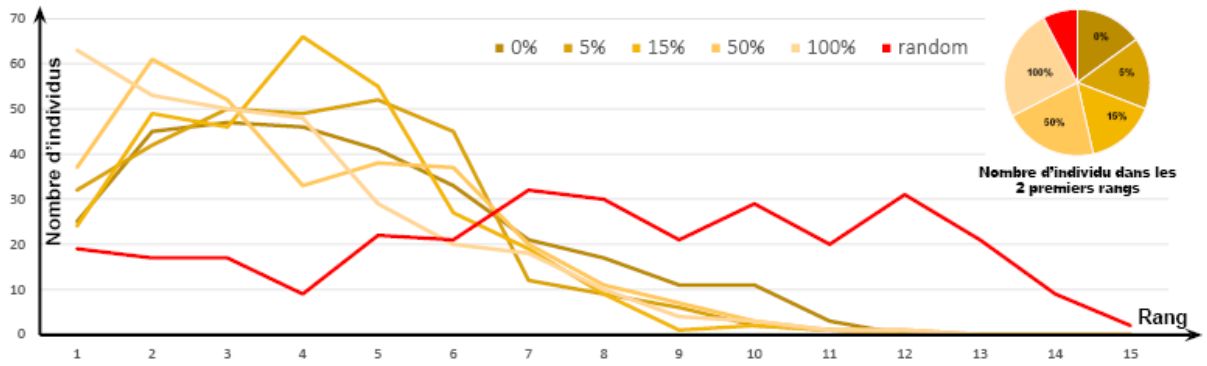


Figure 292 : Nombre d'individus par rangs pour les 6 explorations du cas d'étude de St Laurent de Maroni

Finalement, même pour une fonction de réparation très simple comme celle utilisée pour ce cas d'étude, nous avons dû utiliser un langage de programmation (python) faute de pouvoir faire fonctionner les plugins existants pour faire des boucles conditionnelles avec le solveur d'optimisation que nous avons développé pour pouvoir faire de la réparation avec NSGAII.

Automate cellulaire pour supprimer des aberrations

Description du cas d'étude

Ce second cas d'étude est issu de la pratique professionnelle de l'agence Architecture Studio. Il a déjà fait l'objet d'une publication suite à une conférence de recherche (Duclos-Prévet et al., 2022a). Il s'agit de la conception d'une enveloppe, en phase esquisse, pour la tour de bureaux Axian située à Madagascar. Au début du concours, deux variantes ont été étudiées

pour la façade. La première, celle qui a été retenue, a déjà été présentée au chapitre 2, la seconde est présentée ici.

Cette enveloppe doit permettre de contrôler les apports solaires (non nécessaires quelle que soit la période de l'année) pour éviter les surchauffes et ainsi réduire au mieux les consommations en énergie pour le refroidissement du bâtiment. Cependant, il faut éviter qu'atteindre cette protection nuise au confort visuel. Nous avons cherché dans cette variante à conserver des vues non obstruées vers l'extérieur pour chaque bureau. Cette problématique où un compromis entre visibilité et confort thermique est recherché a déjà fait l'objet d'optimisation avec des algorithmes génétiques dans la littérature scientifique (Showkatbakhsh & Kaviani, 2021).

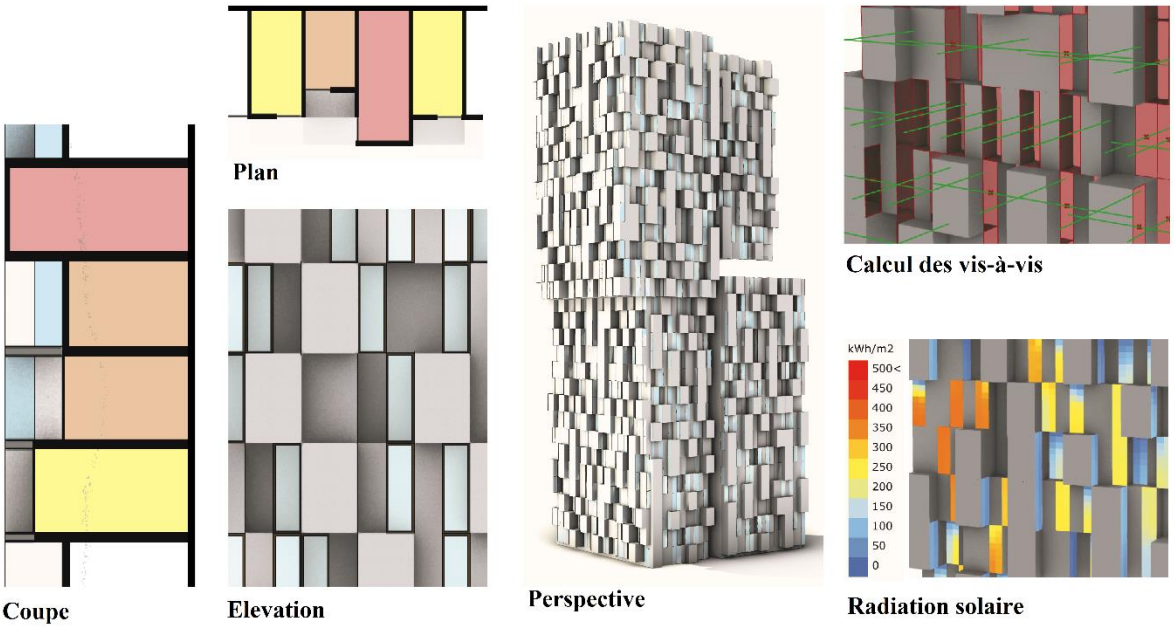


Figure 293 : Ensemble des dessins représentant le concept architectural et les analyses environnementales pour la tour Axian à Madagascar (source: (Duclos-Prévet et al., 2022a)

Pour cette variante, l'équipe d'architectes s'était inspiré de l'art vernaculaire local pour concevoir une enveloppe avec un système de protection solaire constituer d'une trame orthogonale constituée de cellules parfois vides et parfois opaques semblant être disposées de façon aléatoire, rappelant ainsi les motifs générés par des AC. L'équipe souhaitait par ailleurs donner du sens à ces motifs en les disposant non pas aléatoirement mais en fonction de l'ensoleillement. Ainsi, nous avons proposé de fusionner l'enveloppe principale et les cellules protectrices pour créer une façade poreuse avec des profondeurs de bureaux variables. Le principe de cette enveloppe est illustré en Figure 293. Cette porosité permet à l'enveloppe de créer des ombres propres afin de s'auto-protéger des rayons du soleil sans nécessiter un système

de protection supplémentaire qui réduirait drastiquement la visibilité depuis les bureaux vers l'extérieur.

Ainsi, nous avons discrétisé les façades selon la trame des bureaux en modules rectangulaires. Trois dispositions (formes) pour les bureaux sont possibles. Certains bureaux sont ainsi disposés en saillie (A), d'autres restent en l'état (B), et certains sont en retrait (C). Pour chaque bureau, une cellule doit être vitrée. Selon la forme du bureau, le vitrage peut être placé différemment. Finalement, il y a 10 variantes de bureaux possibles présentées en Figure 294 sous la forme d'un tableau.

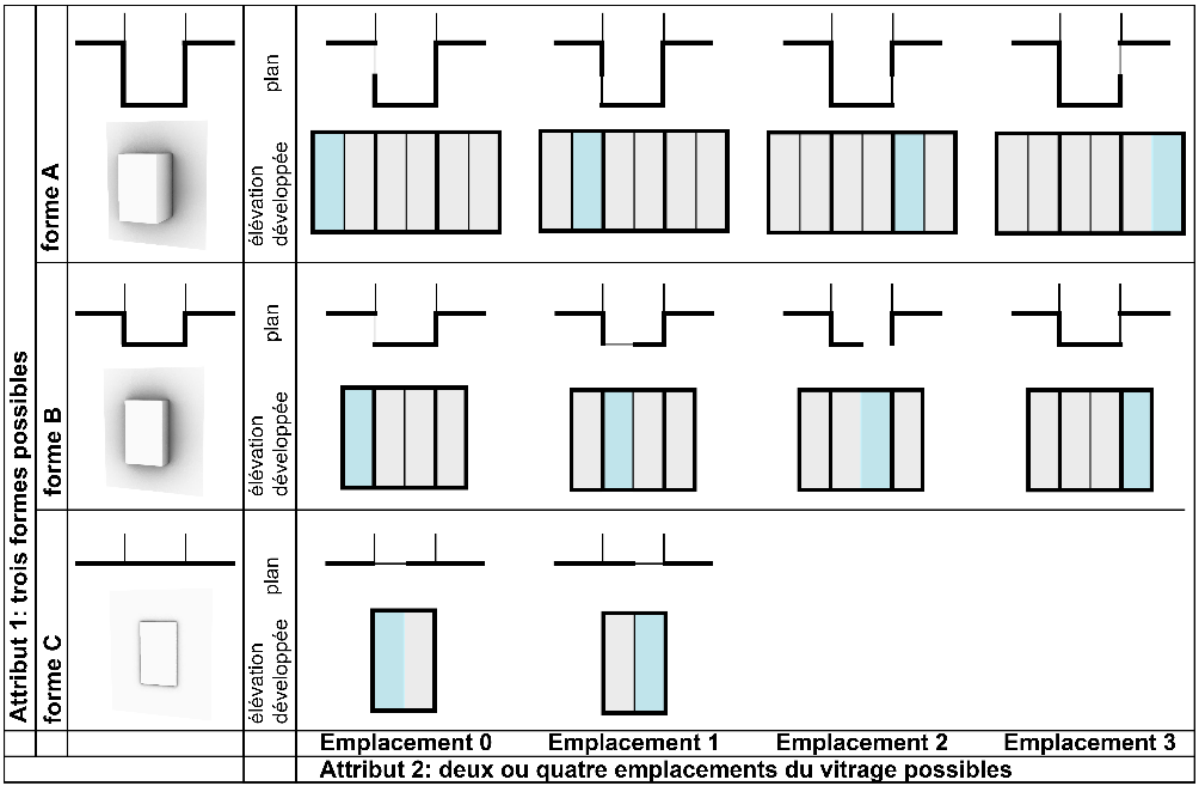


Figure 294 : Représentations des 10 modules classés selon les attributs « forme » (A, B, C) et « emplacement du vitrage » (0, 1, 2, 3) (source : (Duclos-Prévet et al., 2022a)

Modélisation du problème

L'expérimentation a été conduite sur un morceau de façade orienté plein Sud et composé de 150 bureaux dans un contexte urbain hétérogène. Ainsi, le modèle compte 150 variables, soit un ensemble de de 1 x 10^{150} solutions au total. Pour l'évaluation annuelle de la radiation, le plugin Ladybug® a été utilisé ainsi qu'un fichier météo de la station de l'aéroport d'Antananarivo (https://www.ladybug.tools/epwmap/). Pour évaluer la visibilité vers l'extérieur, nous avons fait un petit programme pour compter le nombre de vis-à-vis entre les éléments vitrés de la façade avec un système de lancers de rayons.

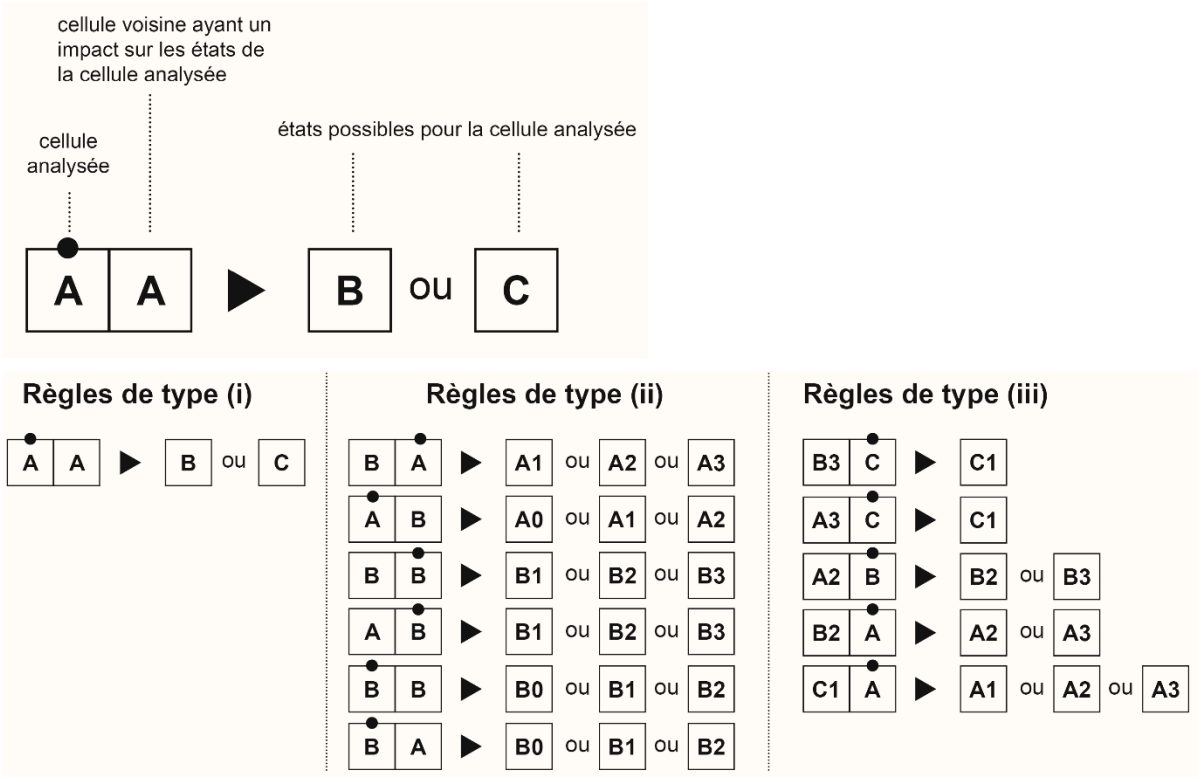


Figure 295 : Ensemble des règles de l'automate cellulaire permettant de supprimer les aberrations (source : Duclos-Prévet et al., 2022a)

Bien que cela ne soit pas forcément visible *a priori*, ce problème est soumis à de nombreuses contraintes. Lorsque les 150 variables sont définies aléatoirement, il arrive systématiquement que le vitrage de certains bureaux donne sur le bureau voisin plutôt que sur l'extérieur, nous parlons donc d'aberrations (car il ne viendrait pas à l'esprit d'un architecte de dessiner une telle enveloppe). Il y a des incompatibilités de voisinage entre les différents modules de bureaux. Pour résoudre ce problème nous avons programmé une fonction de réparation fonctionnant comme un AC, seulement au lieu de compter une caractéristique comprenant deux états (vivant ou mort), il compte deux caractéristiques (forme et disposition du vitrage) comprenant 3 états pour la forme (A, B ou C) et 4 ou 2 états pour la disposition du vitrage (0,1 ou 0,1,2,3). Aussi, une autre subtilité pour intégrer nos contraintes a trait au besoin de spécifier la position exacte des voisins impactant l'état de la cellule plutôt que de définir un rayon. Pour supprimer les aberrations, favoriser des motifs verticaux (pour correspondre aux références de l'architecture vernaculaire) et favoriser la production d'ombres propres, trois types de règles ont été définies :

(1) La forme des cellules voisines impose une modification de la forme de la cellule.

(2) La forme des cellules voisines impose une modification de l'emplacement du vitrage de la cellule.

(3) La forme et l'emplacement du vitrage des cellules voisines imposent une modification du vitrage de la cellule.

L'ensemble de ces règles est présenté en Figure 295.

Résultats

Six explorations ont été calculées, dont une exploration complètement aléatoire pour servir de groupe contrôle. Les 5 autres ont été effectuées avec des taux de remplacement (du génome original avec le génome des solutions réparées) variable (100, 50, 15, 5 et 0%). Pour chaque optimisation, 30 générations de 50 individus chacune ont été calculées, soit 1500 solutions par explorations, 9000 solutions au total dont quelques-unes sont présentées en Figure 297.

Ce problème est particulièrement contraint, 100 % des solutions générées ne respectent pas les contraintes. Les performances de l'algorithme génétique avec les solutions réparées est bien meilleur que pour l'exploration aléatoire qui ne contient aucune solution dans les 80 premiers rangs. Pour ce problème, le taux de remplacement le plus performant est de loin celui de 15 %. Il est bien meilleur que les 4 autres sur les deux premiers rangs comme le montre les résultats présentés en Figure 296.

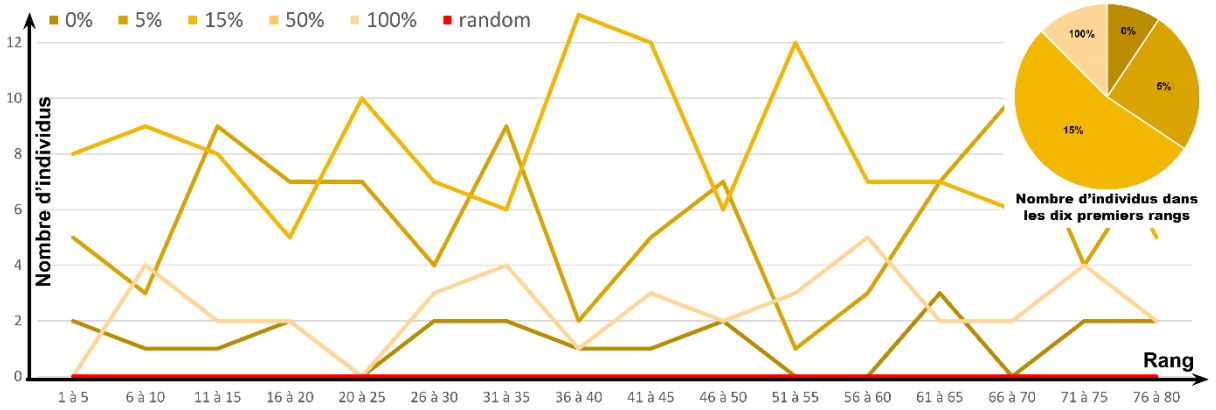


Figure 296 : Nombre d'individus par rang pour les 6 explorations du cas d'étude de Madagascar

Au terme de ce second cas d'étude, il apparait que l'utilisation d'un AC soit une méthode particulièrement efficace pour intégrer des contraintes dans un système tramé. Cependant, il ne s'agit pas ici d'un AC classique, mais d'un d'AC complexe avec un paramétrage est un peu inhabituel. Pour cette expérimentation, le paramétrage des règles fut laborieux, notamment pour préciser l'emplacement des voisins impactant la cellule. Des outils pour faciliter la programmation de ce type d'algorithmes avec une IHM pour faciliter la saisie des règles restent à imaginer.

Figure 297 : Extrait de l'ensemble de solutions générées pour le cas d’étude de Madagascar

Auto-organisation pour supprimer des collisions

Description du cas d'étude

Le problème des collisions ou des superpositions est une contrainte récurrente dans différents types de problème de conception computationnelle : conception d'enveloppe (Parasho et al., 2013, Duclos-Prévet et al., 2021), superposition des fenêtres, génération de plans (Veloso et al., 2019), échelle urbaine (Koenig & Schmitt, 2016).

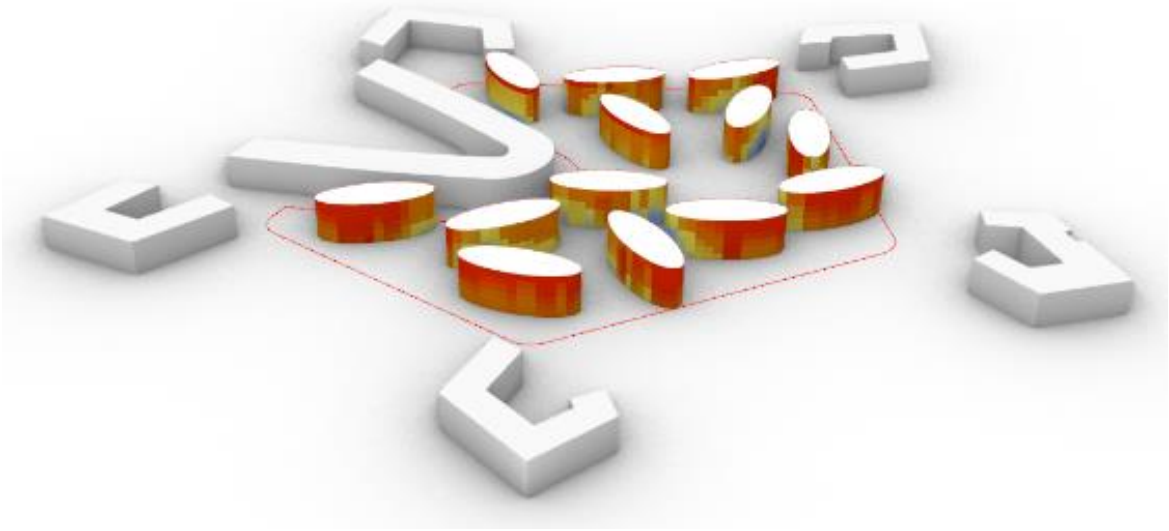


Figure 298 : Modélisation 3D du projet du fort d'Aubervilliers sur Rhinocéros

Pour ce troisième cas d'étude, il s'agit d'une rétroconception d'un projet d'Architecture Studio. Un ensemble de 13 bâtiments de logements à la forme elliptique construits à Aubervilliers en région parisienne sont disposés au cœur d'une ancienne forteresse militaire (voir Figure 298). L'objectif de l'exploration est d'optimiser la disposition des empreintes des bâtiments de façon à réduire les apports solaires estivaux pour le confort thermique, de maximiser les apports en hiver pour réduire les coûts de chauffage tout en évitant au mieux les vis-à-vis.

Modélisation du problème

Pour réaliser cette exploration, un modèle paramétrique est utilisé pour faire varier les coordonnées en x et y des empreintes de chaque bâtiment ainsi que leur orientation. Le problème compte 39 variables, et un espace de 1.1×10^{97} solutions. Les apports d'été et d'hiver ont été calculés à l'aide du plugin Ladybug®, et les vis-à-vis ont été mesurés à l'aide de lancers de rayon. La méthode utilisée pour évaluer ces vis-à-vis a été présentée au chapitre 1 (p**Erreur ! Signet non défini.**). Les résultats sont visualisés en façade comme sur la Figure 299.

Figure 299 : Les trois fonctions objectifs du cas d'étude du fort d'Aubervilliers

Pour la grande majorité des solutions créées avec ce modèle paramétrique, deux contraintes ne sont pas respectées : (i) les bâtiments se superposent et (ii) sortent des limites de la zone constructible. Pour résoudre le problème de collision nous avons dû élaborer un algorithme de répulsion adapté à la forme elliptique. Il s'agit d'un système d'auto-organisation (McCormack et al., 2004; Singh & Gu, 2012). Dans ce MBA, chaque empreinte est un agent qui se déplace en fonction de sa proximité avec ses voisins et avec son environnement qui est ici la courbe fermée délimitant la zone constructible. Dans notre algorithme, le comportement des agents varie selon le nombre de bâtiments avec lequel ils rentrent en collision, selon la nature de la collision, et selon la proximité avec le périmètre de la zone constructible. Le détail du fonctionnement de cet algorithme de répulsion est présenté dans la prochaine partie de ce chapitre.

Résultats

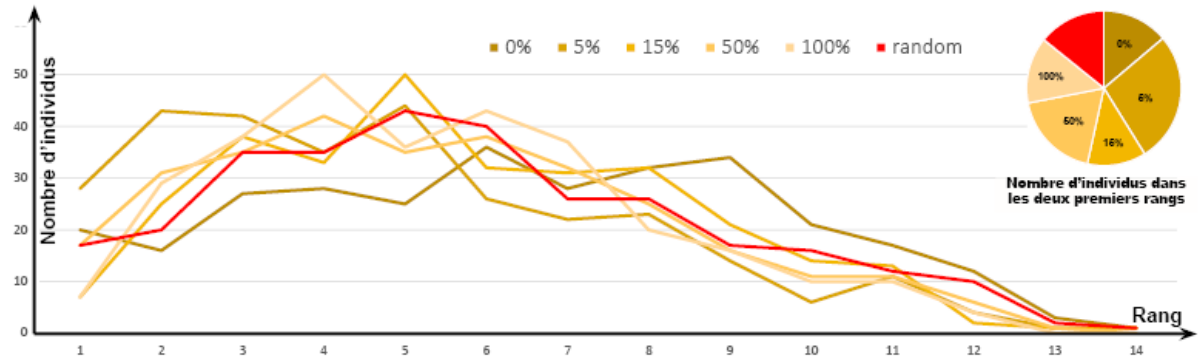


Figure 300 : Nombre d'individus par rang pour les 6 explorations du cas du fort d'Aubervilliers

Ce problème est très contraint, la région faisable est très étroite. La quasi-totalité des solutions générées ont nécessité d'être réparées puisqu'au moins deux bâtiments se superposent systématiquement. Pour chaque exploration, 10 générations de 30 individus chacune ont été calculées, soit 300 solutions par explorations, et1800 solutions générées au total dont quelques-unes sont présentées en Figure 301.

Figure 301 : Extraits de l’ensemble de solutions générées pour le cas d’étude du fort d’Aubervilliers

Concernant la performance de l'exploration, les résultats sont présentés en Figure 300. L'algorithme génétique semble rencontrer des difficultés pour converger. Il ne fait pas beaucoup mieux qu'une exploration aléatoire. Le taux de remplacement le plus adapté est de 5 % des solutions réparées comme il est recommandé pour les problèmes d'optimisation combinatoire (Orvosh & Davis, 1993).

De multiples boucles pour de multiples contraintes

Description du cas d'étude

Pour certains problèmes, les contraintes sont multiples et peuvent être de différentes natures, comme des contraintes constructives ou des contraintes esthétiques. On peut alors tenter d'élaborer un MBA qui permettrait d'intégrer de façon pondérée toutes les contraintes. Cependant, il n'est pas toujours possible de définir une règle qui permette de résoudre tous les problèmes simultanément. Si les contraintes ne rentrent pas en concurrence, on peut alors tenter de les traiter séparément et successivement avec différentes boucles conditionnelles simples. Ainsi, on peut être amené à utiliser plusieurs fonctions de réparation pour un même problème.

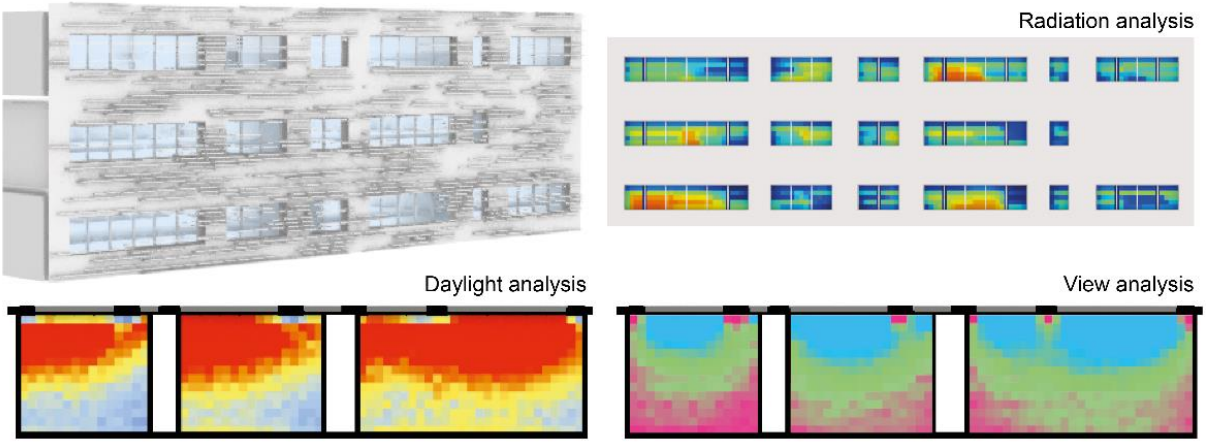


Figure 302 : Modélisation 3D et résultats d'analyse pour le cas d'étude de Marseille

Dans le cadre d'un concours pour un complexe scolaire à Marseille (déjà présenté au chapitre 2), nous avons cherché à optimiser un système de protection solaire pour éviter les surchauffes en été sans porter atteinte au confort visuel (qui comprend à la fois l'éclairement naturel et la qualité de vue vers l'extérieur). La radiation solaire et la qualité de vue vers l'extérieur ont été analysées avec des indicateurs du plugin Ladybug®. L'éclairement naturel mesuré au 21 Décembre à 12h est simulé avec Radiance.

L'enveloppe du bâtiment est protégée avec des dizaines de lames horizontales aux longueurs et largeurs variables pour créer un effet aléatoire (voir Figure 302). Cependant, les

lames ne peuvent être véritablement disposées de manière aléatoire. Il est nécessaire de prendre en compte 3 types de contraintes différentes :

1. Les contraintes constructives pour la fixation des lames (elles ne peuvent flotter devant les fenêtres).
2. Les contraintes de coût avec l'industrialisation des lames (réduction du nombre de types) et limiter la quantité totale de lames (le nombre de mètres linéaires).
3. Les contraintes esthétiques, notamment la recherche d'un effet aléatoire et d'une répartition homogène des lames sur la façade.

Modélisation du problème

Nous avons dans un premier temps tenté d'élaborer un unique algorithme pour gérer toutes ces contraintes. Pour cela, un système à base d'agents est particulièrement adapté. Chaque lame horizontale est un agent qui peut se déplacer dans l'axe des Y et des Z.

Le nombre de lames et leurs dimensions respectives sont fixés dès le départ, ce qui permet de gérer les contraintes de coût. Les autres contraintes sont gérées par des règles permettant de mettre en mouvement les agents. Les limites de l'enveloppe et le contour des fenêtres définissent l'environnement.

A chaque itération une translation est appliquée à l'agent dont le vecteur est la somme pondérée des vecteurs résultants pour chaque contrainte. Avec 252 lames, il faut un nombre important d'itérations pour équilibrer le système. Le processus devient alors un processus d'optimisation à part entière bien trop chronophage pour constituer une fonction de réparation.

Une autre méthode, moins chronophage, a donc dû être imaginée. Elle a nécessité d'adapter le modèle paramétrique permettant une première allocation de 252 lames. La hauteur du bâtiment est divisée par 63 pour créer une trame horizontale. Huit points sont disposés sur chaque ligne de la trame. Ils servent de début et de fin pour 4 lames par ligne, soit 504 variables (une par point), auxquelles il faut ajouter 252 variables pour définir la largeur de chacune des lames. La taille de l'ensemble de solutions est alors de $2{,}6.10^{1130}$. Trois fonctions de réparation sont ensuite utilisées, les unes à la suite des autres :

(1) Une première étape permet d'assurer l'homogénéité de la répartition des lames. La façade est divisée en 119 zones carrées. Le nombre de lames par zone est ensuite analysé. Les zones les moins denses récupèrent ensuite une partie des lames des zones denses à proximité.

(2) Une seconde étape permet d'ajuster la disposition des lames passant devant les baies pour éviter qu'elles ne se retrouvent en lévitation ou en porte à faux. Ainsi, selon sa relation à la baie, la lame peut être déplacée ou modifiée.

(3) Enfin, une dernière étape permet d'ajuster la longueur de chacune des lames afin de réduire le nombre de types de lame et plafonner le mètre linéaire de lames.

Résultats

Ce problème est lui aussi très contraint. Toutes les solutions générées nécessitent réparation. Comme le montre la Figure 303, NSGAII arrive à converger puisque le nombre d'individus dans les deux premiers rangs est plus important que pour le groupe contrôle quel que soit le taux de remplacement utilisé. Pour chacune des 5 explorations réalisées avec NSGAII, 10 générations de 30 individus chacune ont été calculées, 300 solutions par explorations et 1800 solutions générées au total. Un extrait de ces solutions est présenté en Figure 304. Les taux de remplacement de 50 et 100 % donnent de loin les meilleurs résultats.

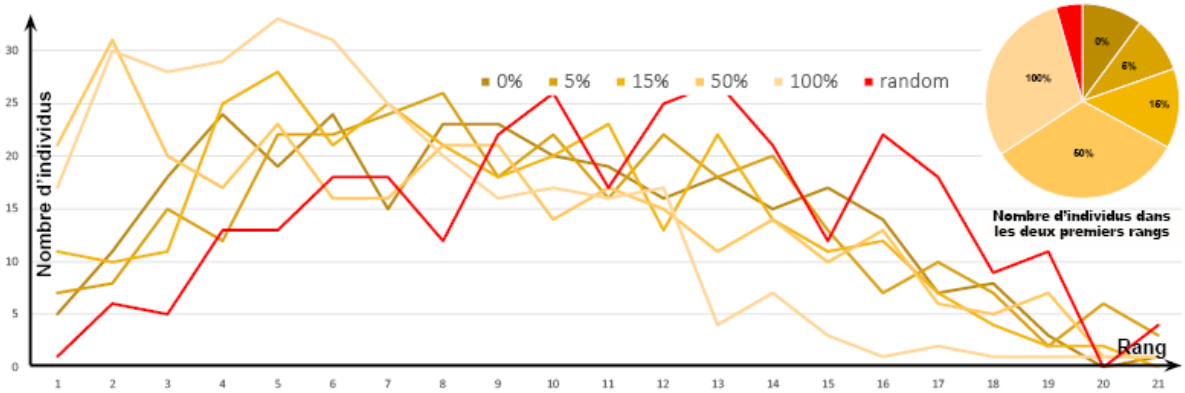


Figure 303 : Nombre d'individus par rang pour les six explorations du cas de Marseille

Une fois de plus, ce cas d'étude a nécessité l'usage d'un langage de programmation (python) faute de pouvoir faire des boucles conditionnelles fonctionnant avec notre solveur d'optimisation sous contraintes. Il nous a permis d'observer que les MBA où les agents sont mobiles (*swarm intelligence*, auto-organisation) sont adaptés pour faire de la réparation seulement si le nombre d'agents n'est pas trop important, au risque de voir exploser les temps de calcul.

Les expérimentations présentées ici ne suffisent pas pour tirer des conclusions définitives sur la meilleure façon d'implémenter une fonction de réparation. Cela constitue une première adaptation d'une méthode ayant fait ses preuves sur des problèmes très éloignés de la conception architecturale.

Figure 304 : Extrait de l’ensemble de solutions générées pour le cas d’étude de Marseille

Nos expérimentations sur quatre cas d'études (5 en comptant celle du chapitre précédent) permettent de confirmer que l'approche lamarckienne donne de meilleurs résultats que l'approche baldwinienne (lorsque le taux de remplacement est de 0 %). Cela justifie le besoin d'un solveur adapté permettant le remplacement des génotypes des solutions originales par ceux des solutions réparées. Aussi, il apparaît que la règles du taux de remplacement à 5 % pour les problèmes discrets (Orvosh, 1993) ne soit pas systématiquement vérifiée. Pour les 4 problèmes, le taux de remplacement optimal n'est jamais le même (100 %, 15 %, 5 %, 50 %), il est donc difficile de faire des préconisations à ce sujet.

Pour le troisième cas d'étude, l'exploration conduite par l'algorithme génétique semble perturbée. Seul un taux de remplacement de 5 % se distingue du groupe contrôle en quantité de solutions performantes trouvées. C'est aussi le problème pour lequel les solutions réparées s'éloignent le plus des solutions originales en termes de génotype, ce qui correspond aux recommandations de Smith and Coit, (1997) qui disent qu'un génome trop modifié peut altérer le fonctionnement de l'algorithme.

Finalement, les fonctions de réparation peuvent être utilisées sur différents types de problèmes et pour des contraintes de différentes natures : constructives, esthétiques, environnementales, de coûts de construction, ou des aberrations. Pour cela un algorithme spécifique doit être développé pour chaque problème. Nous savons désormais que des techniques génératives comme de simples boucles conditionnelles, ou encore des MBA élémentaires comme des algorithmes d'auto-organisation ou des automates cellulaires complexes, sont pertinents pour créer des fonctions de réparation. Nous n'avons pas à ce jour eu besoin d'utiliser les autres techniques génératives (LS et GF) présentées au début de ce chapitre pour définir les fonctions de réparation nécessaires pour résoudre les problèmes d'optimisation que nous avons rencontrés dans la pratique.

L'ensemble de ces recherches nous ont permis de définir un cahier des charges pour une première bibliothèque d'outils utiles pour la modélisation de fonction de réparation. Nous savons désormais qu'il est nécessaire de (1) pouvoir faire des boucles conditionnelles qui fonctionnent avec un solveur d'optimisation, (2) pouvoir modéliser des automates cellulaires complexes avec des IHM pour la saisie des règles, (3) pouvoir modéliser des systèmes d'auto-organisation avec des éléments qui ne sont pas des particules comme des courbes fermés.

Au terme de cette première partie du chapitre 4, il apparait que s'il existe des outils pour manipuler la plupart des techniques génératives connues en conception computationnelle, ces derniers ne sont pas adaptés pour la création de fonctions de réparation. Dans les plugins Grasshopper disponibles aujourd'hui, les SMA sont notamment restreints à la simulation de particules et les AC à seulement deux états possibles.

Aussi, quatre nouveaux cas d'étude issus de la pratique et des anciens projets d'AS nous ont permis de confirmer que la méthode lamarckienne était plus efficace que la méthode baldwinienne pour faire de la réparation. Toutefois, les solveurs d'optimisation existants ne se sont pas montrés adaptés pour faire de la réparation lamarckienne ; cela confirme qu'il existe un besoin pour un nouveau solveur d'optimisation. Ces études ont aussi permis de noter qu'un taux de remplacement de 5 % des solutions originales par les solutions réparées n'était pas forcément recommandé pour les problèmes rencontrés par les architectes.

Ces expérimentations nous ont surtout permis de confirmer que les SMA avec quelques agents mobiles, les AC ou autres boucles conditionnelles simples peuvent se révéler très utiles pour faire de la réparation de solution. Cependant, ces techniques nécessitent à l'heure actuelle des connaissances en programmation informatique pour être implémentées dans un processus d'exploration. En modélisant ces différents problèmes issus de la pratique, nous avons pu établir une première liste d'outils nécessaires pour faciliter la programmation de fonctions de réparation. Le développement de certains d'entre eux ont été entamés à l'occasion de ces modélisations, avant d'être améliorer afin de pouvoir généraliser leur usage sur des problèmes divers.

Le résultat final de ces différentes phases de développement informatique est un premier plugin Grasshopper® destiné à l'optimisation multicritère sous contrainte notamment avec des fonctions de réparations développées à l'aide de techniques génératives. Nous en faisons une description détaillée dans la partie suivante de ce chapitre.

4.2. Une bibliothèque d'outils Grasshopper® pour la gestion des contraintes

Cette thèse de doctorat faisant l'objet d'une convention CIFRE, il est attendu que les résultats de notre recherche puissent être exploité rapidement par les collaborateurs de l'agence Architecture Studio. Ainsi, nous savons que les techniques génératives à base d'agents sont adaptées pour l'intégration des contraintes dans un processus d'optimisation en utilisant la méthode de la fonction de réparation, mais les collaborateurs n'ont, pour le moment, pas d'outils pour exploiter cette connaissance. Ainsi, l'ensemble des scripts développés en python pour les expérimentations sur les projets de l'agence ou pour les études comparatives ont été rassemblés pour créer un plugin Grasshopper dédié. Certains scripts ont été modifiés et généralisés à différents types de problème. Dans cette deuxième partie du chapitre 4, nous faisons une description détaillée du plugin « *Urchin* », qui est le premier plugin Grasshopper® pour l'optimisation multicritère pensé pour faciliter l'intégration des contraintes rencontrés dans les problèmes de conception des architectes. Il est composé de 34 composants au total répartis en trois catégories : 20 pour la réparation, 4 pour l'optimisation et 10 pour la visualisation. Cette partie est écrite comme un guide d'utilisation du plugin où chaque composant est passé en revue, son fonctionnement expliqué, contextualisé et illustré.

4.2.1. Les outils pour créer des automates cellulaires

Pour pouvoir créer des fonctions de réparation avec des automates cellulaires, nous avons dû créer une douzaine de composants (voir Figure 305). Avant de préciser leur fonctionnement et leur utilité, nous souhaitons revenir sur la définition de nos automates cellulaires qui ne correspond pas entièrement à celle donnée au début du chapitre (p.390).

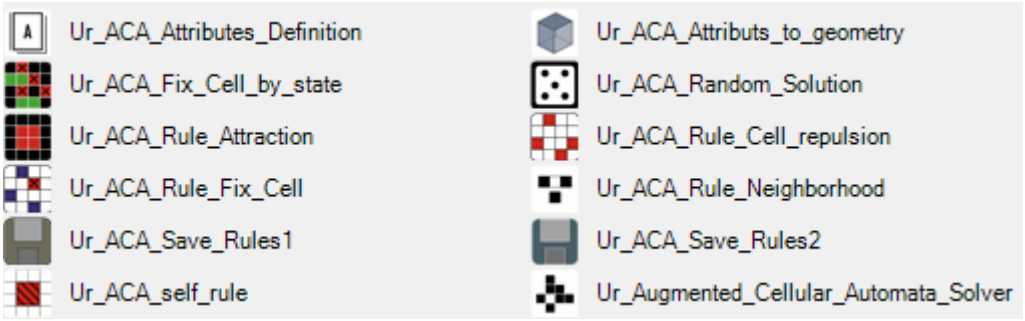


Figure 305 : Liste des composants pour créer des AC

Automate cellulaire augmenté

Les outils développés pour *Urchin* permettent de définir des algorithmes un peu plus complexes qu'un automate cellulaire classique ; ainsi nous parlerons d'automate cellulaire « augmenté » (ACA). Tout au long de cette partie, nous illustrerons le principe d'un ACA à l'aide d'un problème simple de collision de lames verticales. Une façade composée de 38 lames réparties sur 3 niveaux compte 3 types de lames de largeurs différentes (0.5, 0.75, et 1 mètre) pouvant être orientées selon 5 angles différents (-45, -25, 0, 25 ou 45 degrès).

Les caractéristiques spécifiques de l'ACA

Ces algorithmes diffèrent des automate cellulaire classique, comme le jeu de la vie, selon trois aspects :

(i) Les cellules voisines ayant un impact sur la cellule concernée peuvent être choisies individuellement au moment de la définition d'une règle.

Plutôt que de définir un rayon de cellules voisines ayant un impact à chaque itération sur l'état de la cellule considérée, l'utilisateur choisit d'abord un rayon, pour limiter une première fois les cellules voisines pouvant être impactantes, puis il sélectionne parmi ces cellules lesquelles sont réellement prises en comptent dans la règles (voir Figure 306).

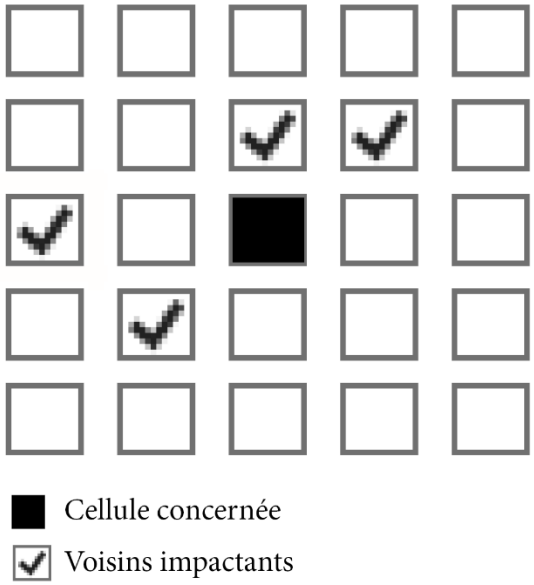


Figure 306 : Sélection des cellules impactantes (avec un rayon de 2).

(ii) Les cellules peuvent avoir différentes caractéristiques que nous appelons un « attribut ».

Les AC classiques, comme le jeu de la vie, ne comptent qu'un attribut avec deux états possibles pour les cellules (« morte » ou « vivante »). Pour appliquer ces algorithmes à des problèmes d'architecture, on peut imaginer utiliser des états de nature différentes et en plus grande quantité (par exemple « grande lame », « moyenne lame », « petite lame »). Si cela est déjà plus utile, avec un seul attribut, on ne peut optimiser qu'un seul paramètre (ici la taille de la lame). Comme pour le problème présenté dans la partie précédente de ce chapitre (p.400), les problèmes rencontrés par les architectes comptent souvent plusieurs paramètres. Dans notre exemple, les lames varient en largeur et en orientation, et ces deux paramètres ont un impact sur les collisions. Pour intégrer aussi l'orientation des lames, il faut ajouter à l'attribut « largeur » un second attribut « angle » qui compte 5 états possibles pour les 5 angles que peuvent prendre les lames.

Avec les composants du plugin Urchin, le nombre d'attributs est illimité, ainsi que le nombre d'états possibles pour chaque attribut. Pour définir les attributs et les états, il faut utiliser le composant « Ur_ACA_Attributes_Definition » et l'utiliser avec des composants de chaîne de caractères comme sur la Figure 307 (« angle » et « largeur »). Le second composant présenté sur la figure (Ur_ACA_Random_Solution) permet de générer une solution originale aléatoirement (il n'est pas nécessaire lorsqu'on souhaite utiliser un ACA pour faire de la

réparation). Pour générer une solution, ce composant a besoin de connaitre l'arborescence des cellules, le paramètre d'entrée G est un arbre structuré comme dans la figure ci-dessous.

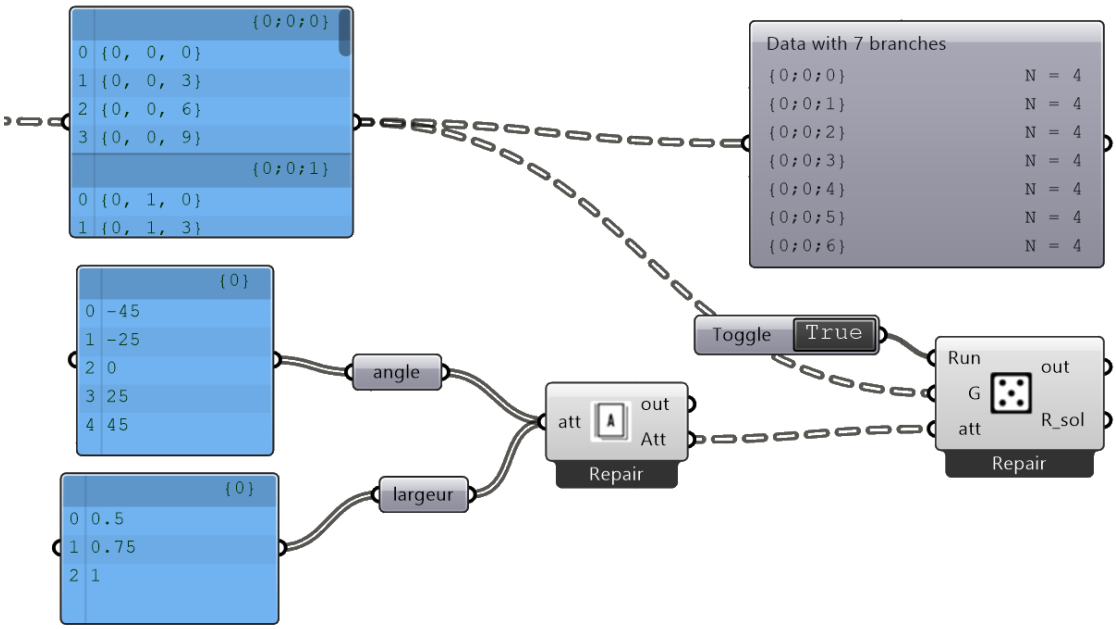
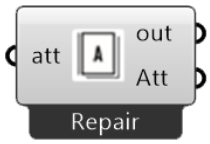
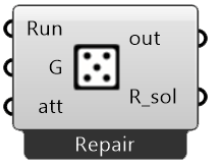


Figure 307 : Composants d'initialisation (Ur_ACA_Attributes_Definition et Ur_ACA_Random_Solution)

Dans le tableau ci-dessus sont indiqués les différentes données d'entrée et sortie des composants d'initialisation d'ACA.

Tableau 12 : Composants Urchin d'initialisation d'un ACA

Composants	Données entrées	Données sorties
Ur_ACA_ Attributes_Definition 	**att** : Liste d'états possibles organisés par attribut (nombres, nombres entiers, booléen ou chaîne de caractère). L'ordre des attributs compte. Un attribut ne peut impacter un attribut placé avant dans la liste.	**Att** : Arbre de données contenant toutes les informations sur les attributs et leurs états possibles.
Ur_ACA_ Random_Solution 	**Run** : « True » pour générer une solution aléatoire **G** : Arbre de données contenant une grille de points, rectangles, ou surfaces. **att** : Arbre de données contenant toutes les informations sur les attributs et leurs états possibles.	**R_sol** : Arbre de données contenant pour chaque cellule les états possibles pour chaque attribut sélectionné aléatoirement.

(iii) Les cellules peuvent échanger leur état.

Une dernière caractéristique spécifique de ces ACA est la capacité à conserver la quantité de cellules pour un ou plusieurs états choisis par l'utilisateur. Dans les problèmes rencontrés en conception architecturale, il est souvent nécessaire de fixer des quantités (matériaux, programme… ect). Cette option est possible avec le solver pour ACA d'Urchin. Il est possible de fixer le nombre de cellules d'un état spécifique d'un attribut choisi, ou de fixer la quantité de cellules de tous les états d'un attribut choisi ou de fixer la quantité de cellules de tous les états de chaque attribut.

Pour notre problème, on pourrait par exemple fixer la quantité de cellules à grandes, moyennes et petites, ce qui revient à fixer l'attribut « largeur ». Ainsi, lorsqu'une cellule doit changer de largeur pour respecter une règle, elle va d'abord chercher parmi les cellules avoisinantes une cellule avec qui elle peut échanger sa largeur en s'assurant que cette dernière respecte toujours les règles, malgré sa nouvelle largeur.

Architecture générale du script pour définir un ACA

Pour créer un ACA, quatre étapes sont nécessaires. La première que nous venons de présentée est l'initialisation avec la définition des attributs et des états, et la génération d'une solution originale. La seconde est la définition et l'enregistrement des règles. La simulation correspond à la troisième étape. La dernière étape consiste à transformer les résultats numériques de la simulation en géométrie. Les différents composants nécessaires à l'élaboration de ces 4 étapes sont illustrés en Figure 308.

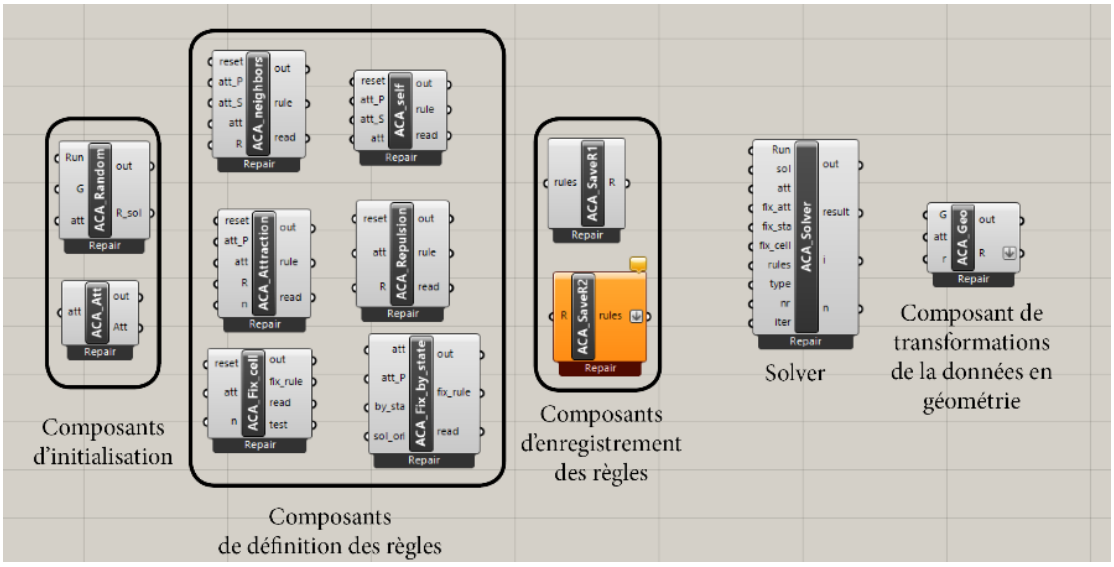


Figure 308 : Les composants nécessaires à la création d'un ACA

Pour générer une solution originale, il faut d'abord définir une grille 2D ainsi que les différents paramètres avec des composants « sliders » ou des « Genepool » en correspondance avec les attributs et états définis. L'architecture générale d'un ACA défini avec le plugin Urchin est présentée en Figure 309.

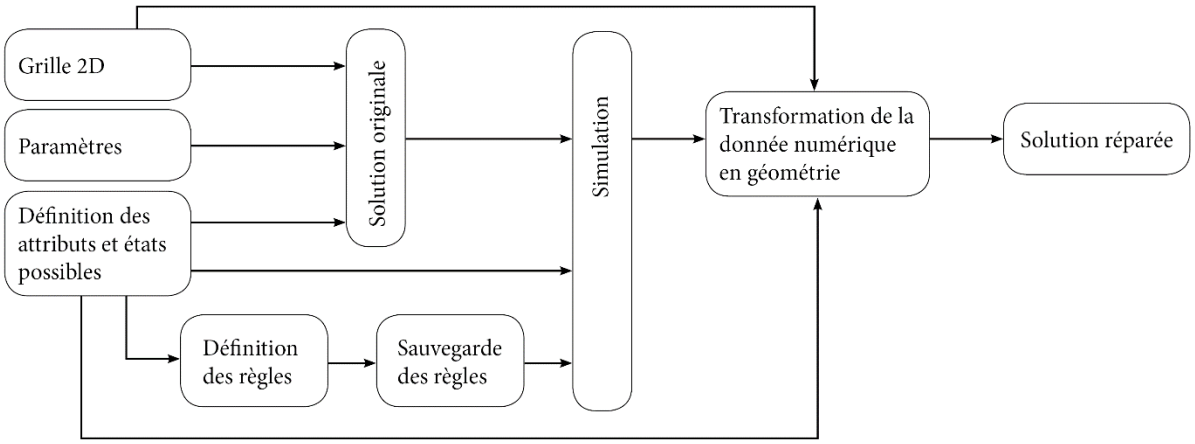


Figure 309 : Architecture générale d'un ACA avec le plugin Urchin

En Figure 311 est présenté un script d'ACA (avec ces différentes étapes) appliqué à notre exemple de collision de lames. Les résultats de l'exécution de ce script sur une solution sont présentés en Figure 310.

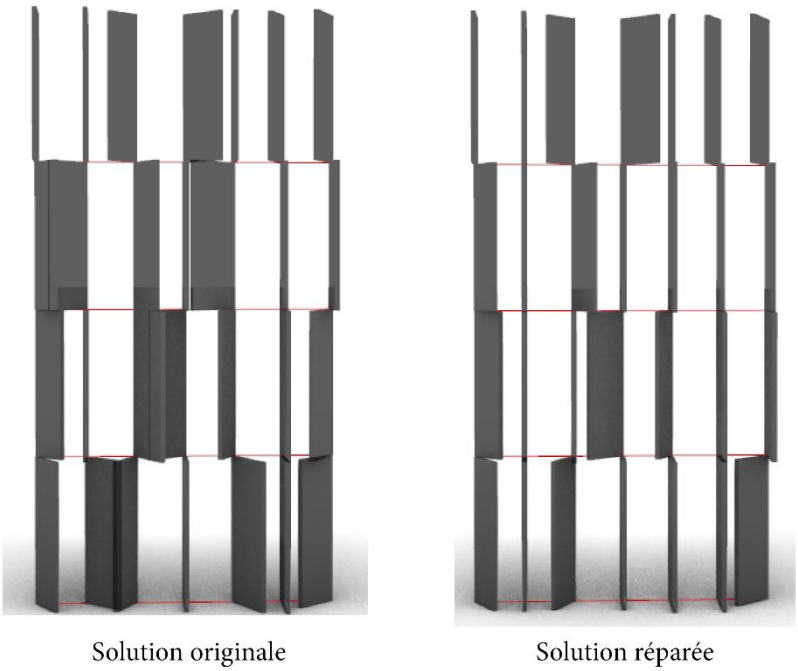


Figure 310 : Exemple d'une réparation de solution

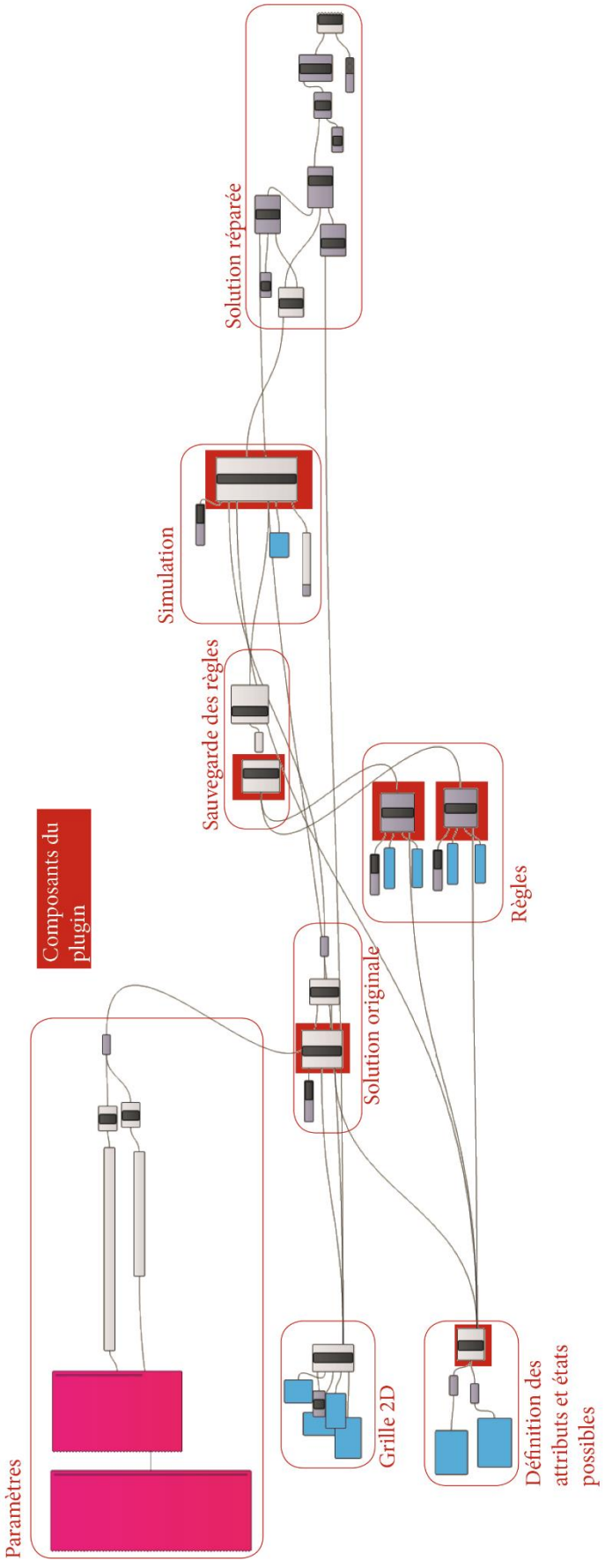


Figure 311 : Architecture d'un ACA pour la réparation de lames en collision

Les composants nécessaires à la définition des règles

Liste des composants pour la définition des règles

Huit composants peuvent être utilisés pour définir des règles, dont deux servent à l'enregistrement des règles (pour éviter de devoir les ressaisir à chaque réouverture du fichier Grasshopper) :

(1) Le composant « Ur_ACA_Save_Rules1 » permet de formater toutes les règles crées pour pouvoir les internaliser dans un composant gh conteneur de Data.

(2) Le composant « Ur_ACA_Save_Rules2 » permet de reformater les règles sous forme de listes Python pour pouvoir les donner en entrée du solveur ACA.

Parmi les 6 composants restants, deux permettent de geler les états de certains cellules sélectionnées :

1) Le composant « Ur_ACA_Rule_Fix_Cell » permet de geler l'état d'un ou de plusieurs attributs d'une ou plusieurs cellules en saisissant manuellement l'index de la colonne et de la ligne de chaque cellule que l'utilisateur souhaite geler.

2) Le composant « Ur_ACA_Rule_Fix_by_state » permet de geler l'état de plusieurs cellules en saisissant le nom de l'attribut et de l'état que l'on souhaite geler. Ainsi, toutes les cellules présentant cet état dans la solution originale conserveront cet état de cet attribut spécifique dans la solution réparée.

Enfin, il reste 4 composants pour définir les règles proprement dites de l'automate. Il est possible d'utiliser plusieurs fois le même composant pour définir différentes règles. Le nombre de règles pouvant être utilisées sur une même simulation n'est pas limité. Cependant, un excès de règles peut empêcher le solveur d'atteindre l'état d'équilibre. Nos composants permettent de définir des règles de 4 types différents :

(1) Le composant « Ur_ACA_Self_Rule » permet à une cellule de se contraindre elle-même. Ce composant n'est utile que si le problème compte au moins deux attributs différents. Alors, il est possible de contraindre l'état du premier attribut en fonction de l'état du second. Par exemple, une « grande lame » ne peut pas être orientée à 45° ou à 45°.

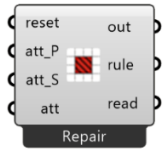
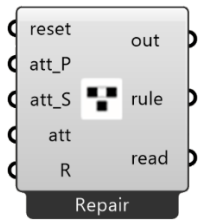
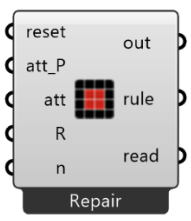
(2) Le composant « Ur_ACA_Rule_Neighborhood » permet d'adapter l'état d'un attribut de la cellule en fonction de l'état des attributs des cellules voisines. Par exemple, si les lames voisines situées à gauche ou à droite d'une grande lame sont grandes ou moyennes et orientées à -45° ou 45°, alors la lame de la cellule doit être à un angle de 0°.

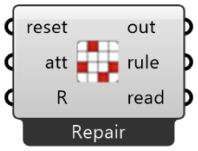
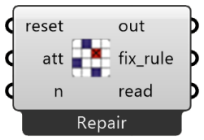
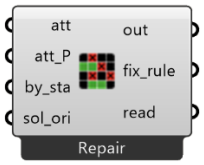
(3) Le composant « Ur_ACA_Rule_Attraction » permet d'attirer des cellules entre elles. Avec ce composant, il est possible d'imposer que les cellules, étant dans un certain état, aient un nombre minimum de cellules voisines (situées dans un rayon donné) avec des caractéristiques particulières. Ainsi, il est possible d'attirer des cellules avec des caractéristiques identiques. Par exemple, si une lame est petite, elle doit avoir au moins deux lames voisines situées en haut, en bas, à gauche et à droite et qui soient elles aussi petites.

(4) Le composant « Ur_ACA_Rule_Répulsion » permet de repousser des cellules entre elles. A l'inverse du composant précédent, celui-ci permet d'interdire certaines caractéristiques à des cellules voisines. Par exemple, si une lame est grande, alors les lames des cellules voisines situées à gauche et à droite ne peuvent avoir des angles de -45° ou de 45°.

Liste des données d'entrée et de sortie pour chaque composant

Tableau 13: Composants Urchin de définition des règles pour un ACA

Composants	Données entrées	Données sorties
Ur_ACA_Self_Rule 	**reset** : Bouton pour faire apparaître la fenêtre de saisie. **att_P** : Liste des noms des attributs impactants. **att_S** : Liste des noms des attributs impactés. **att** : Arbre de données contenant toutes les informations sur les attributs.	**rule** : Liste de listes python à sauvegarder. **read** : Description de la règle saisie.
Ur_ACA_ Rule_Neighborhood 	**reset** : Bouton pour faire apparaître la fenêtre de saisie. **att_P** : Liste des noms des attributs impactants. **att_S** : Liste des noms des attributs impactés. **att** : Arbre de données contenant toutes les informations sur les attributs. **R** : Un entier pour définir la taille du rayon (quantité de cellules voisines affectées par la règle).	**rule** : Liste de listes python à sauvegarder. **read** : Description de la règle saisie.
Ur_ACA_ Rule_Attraction 	**reset** : Bouton pour faire apparaître la fenêtre de saisie. **att_P** : Liste des noms des attributs impactants. **att** : Arbre de données contenant toutes les informations sur les attributs. **R** : Un entier pour définir la taille du rayon (quantité de cellules voisines affectées par la règle). **n** : Nombre de voisins nécessaires	**rule** : Liste de listes python à sauvegarder. **read** : Description de la règle saisie.
Ur_ACA_ Rule_Repulsion	**reset** : Bouton pour faire apparaître la fenêtre de saisie.	**rule** : Liste de listes python à sauvegarder.

	**att** : Arbre de données contenant toutes les informations sur les attributs. **R** : Un entier pour définir la taille du rayon (quantité de cellules voisines affectées par la règle).	**read** : Description de la règle saisie.
Ur_ACA_ Rule_Fix_Cell 	**reset** : Bouton pour faire apparaître la fenêtre de saisie. **att** : Arbre de données contenant toutes les informations sur les attributs. **n**: Nombre de cellules à fixer	**Fix_rule** : Liste de listes python à connecter directement au solveur (input « fix_cell »). **read** : Description de la règle saisie.
Ur_ACA_ Rule_Fix_by_state 	**att** : Arbre de données contenant toutes les informations sur les attributs. **att_P** : Nom de l'attribut concerné. **by_state** : Liste des « états spéciaux ». Les cellules ayant cet état sur la cellule d'origine seront gelées pour l'attribut concerné. **sol_ori** : Arbre de données contenant les états de la solution originale.	**Fix_rule** : Liste de listes python à connecter directement au solveur (input « fix_cell »). **read** : Description de la règle saisie.
Ur_ACA_ Save_Rules1 	**rules**: Liste de listes python réalisées à l'aide des différents composants de définition de règles (Self_Rule, Rule_Neighborhood, Rule_Attraction, Rule_Repulsion)	**R :** Arbre de données à internaliser dans GH pour sauvegarder les règles pour la prochaine ouverture du fichier gh.
Ur_ACA_ Save_Rules2 	**R :** Arbre de données internalisé dans gh.	**Rules :** Liste de listes python à connecter directement au solveur (input « rules »).

Interface de certains composants

Parmi les huit composants qui servent à définir les règles, cinq contiennent une interface graphique (IHM) crée à l'aide de la librairie python opensource Eto forms disponible depuis Rhino 6 pour créer des interfaces dans Grasshopper (Davidson, 2018).

Pour le composant « Ur_ACA_Self_Rule », deux entrées sont saisies à l'aide d'une IHM : (1) l'état spécifique de l'attribut impactant (att_P), et (2) les états interdits de l'attribut impacté (att_S). En Figure 312 est illustrée la saisie de l'exemple « une « grande lame » ne peut pas être orientée à -45° ou à 45° ».

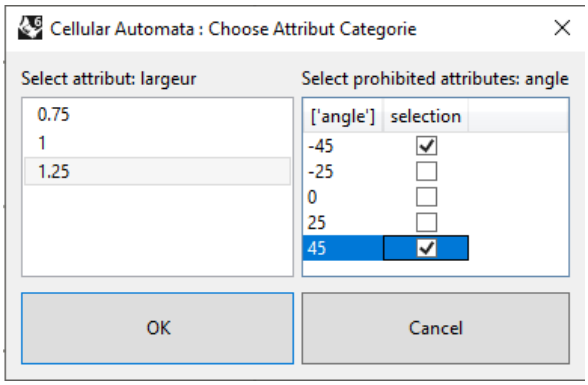


Figure 312 : Interface du composant « Ur_ACA_Self_Rule »

Pour le composant « Ur_ACA_ Rule_Neighborhood », quatre entrées sont saisies à l'aide d'une IHM : (1) l'état spécifique de l'attribut impactant (att_P), (2) les états des différents attributs des cellules voisines, (3) l'emplacement de ces cellules voisines et (4) les états interdits de l'attribut impacté (att_S). En Figure 313 est illustrée la saisie de l'exemple « si les lames voisines situées à gauche ou à droite d'une grande lame sont grandes ou moyennes et orientées à -45° ou 45°, alors la lame de la cellule doit être à un angle de 0° ».

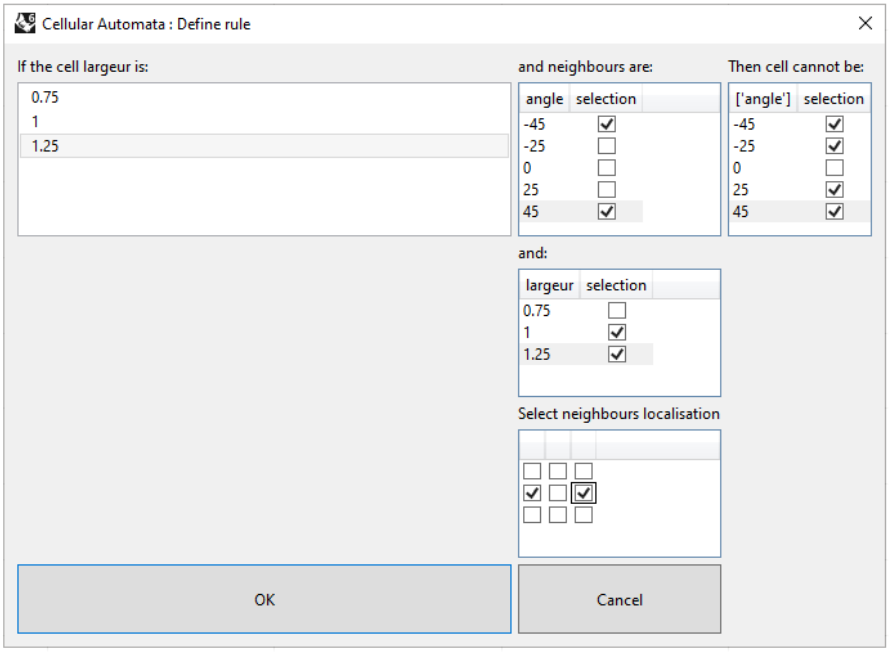


Figure 313 : Interface du composant « Ur_ACA_ Rule_Neighborhood »

Pour le composant « Ur_ACA_Rule_Attraction », trois entrées sont saisies à l'aide d'une IHM : (1) l'état spécifique de l'attribut impactant (att_P), (2) les états des différents attributs des cellules voisines à attirer, (3) l'emplacement de ces cellules voisines. En Figure 314 est illustrée la saisie de l'exemple « si une lame est petite, elle doit avoir au moins deux lames voisines situées en haut, en bas, à gauche, à droite qui soient elles aussi petites ».

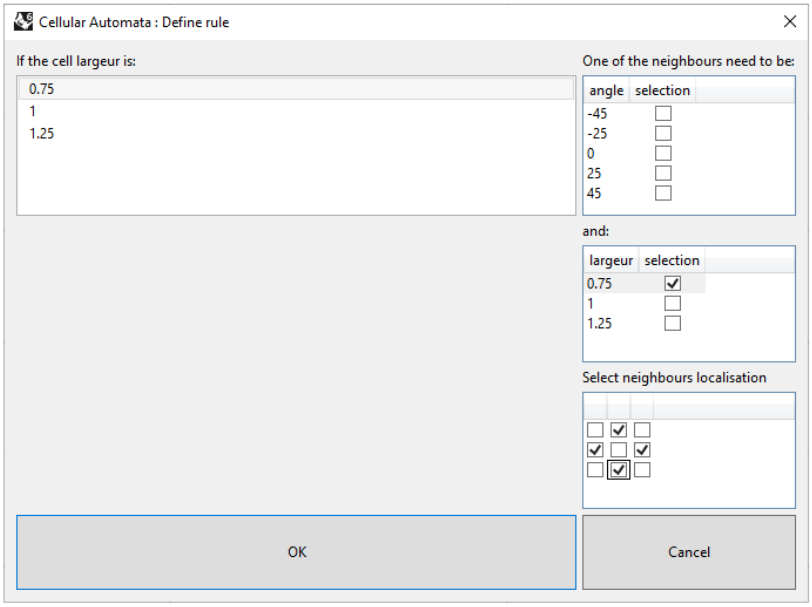


Figure 314: Interface du composant « Ur_ACA_Rule_Attraction »

Pour le composant « Ur_ACA_Rule_Repulsion », quatre entrées sont saisies à l'aide d'une IHM : (1) les états des différents attributs des cellules voisines, (2) l'emplacement de ces cellules voisines, (3) préciser si la règle est vraie si au moins un voisin présente les états précédemment saisies (« OR ») ou si tous les voisins doivent présenter ces états (« AND »), (4) les états des différents attributs des cellules voisines à repousser. En Figure 315 est illustrée la saisie de l'exemple « si une lame est grande, alors les lames des cellules voisines situées à gauche et à droite ne peuvent avoir des angles de -45° ou de 45°».

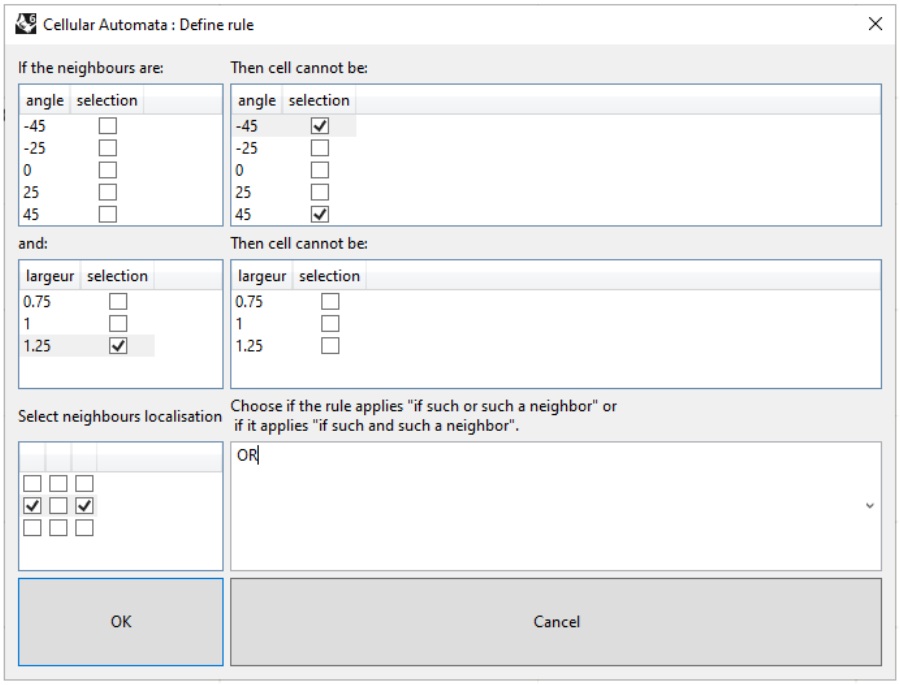


Figure 315 : Interface du composant « Ur_ACA_Rule_Repulsion »

Pour le composant « Ur_ACA_Rule_Fix_Cell », trois entrées sont saisies à l'aide d'une IHM : (1) les attributs concernés, (2) l'index de la colonne où se situe la cellule à geler, (3) l'index de la ligne où se situe la cellule à geler. Les deux derniers paramètres sont à répéter en fonction du nombre de cellules à geler. En Figure 316 est illustrée la saisie de l'exemple « fixer l'orientation des lames placées à l'extrémité de chaque niveau ».

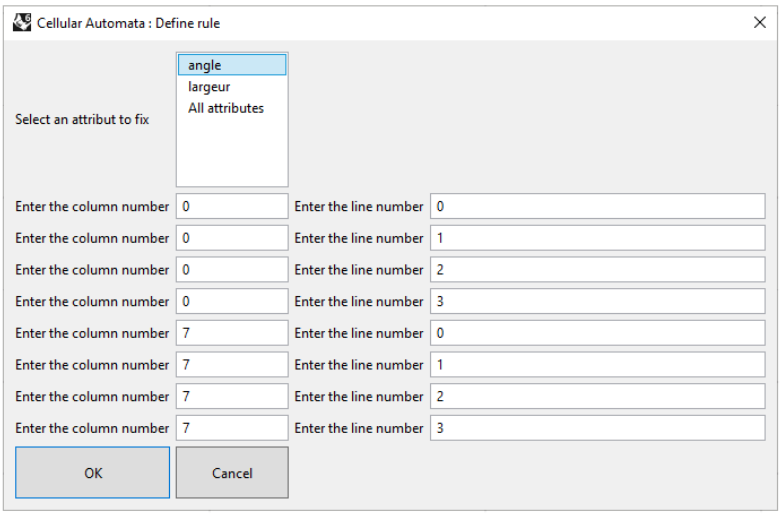


Figure 316 : Interface du composant « Ur_ACA_Rule_Fix_Cell »

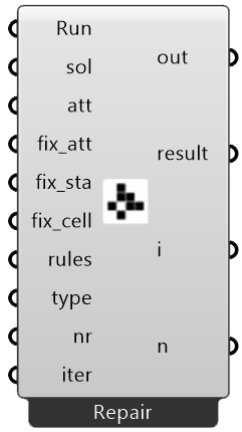
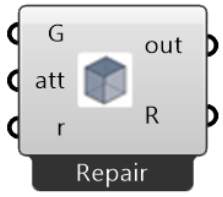
Le solveur d'automate cellulaire augmenté

Les paramètres du solver

Une fois les attributs et les règles définis, il ne reste plus qu'à lancer la simulation et transformer les résultats numériques en géométrie. Pour cela, il faut utiliser les deux composants présentés dans le Tableau 14. Pour la simulation, le nombre d'itérations est paramétrable par l'utilisateur, il est conseillé au début de tester l'algorithme avec un nombre faible d'itérations (2 ou 3). Pour faire de la réparation, il vaut mieux éviter un trop grand nombre d'itérations, même lorsque l'automate a trouvé un état d'équilibre, la simulation s'arrête automatiquement, seulement cet état d'équilibre peut ne jamais être atteint selon le type de problème et de configuration de départ.

Tableau 14 : Composants pour la simulation d'un ACA et la traduction géométrique

Composants	Données entrées	Données sorties
Ur_Augmented_Cellular_Automata_Solver	**Run** : « True » pour démarrer la simulation. **Sol** : Un arbre de données contenant les états possibles d'une solution initiale. **Att** : Arbre de données contenant toutes les informations sur les attributs.	**result** : Nouvel arbre de données avec les nouveaux états de chacune des cellules de la solution réparée.

	**fix_att** : Liste des noms d'attributs pour fixer la quantité d'états de ces attributs **fix_sta** : Liste de "nom_attribut;etat" pour fixer la quantité de certains états d'un attribut. **Fix_cell** : Liste de règles sous forme de liste python pour fixer les états d'une cellule spécifique. **rules** : Liste python permettant au solver de définir des règles **type** : Méthode de sélection des états possibles en cas de modification de la cellule (0 : aléatoire, 1 : ordre) **nr** : rayon du voisinage pris en compte pour les échanges états **iter** : nombre d'itérations	**i** : Nombre d'itérations effectuées. **n** : Nombre de cellules restantes ne respectant pas les règles.
Ur_ACA_Attribyts_ to_geometry 	**G** : Liste de données géométriques ou autres (les « Nicknane » des composants d'entrée doivent correspondre à ceux utilisés pour le composant "Cellular_Automata_Attributes_Definition"). **att** : Arbres contenant toutes les informations relatives aux attributs (générés à partir du composant "Cellular_Automata_Attributes_Definition"). **r** : Liste contenant le résultat des états pour un attribut spécifique.	**R** : Arbre de données contenant la géométrie de la solution réparée**.**

L'algorithme du solveur

L'algorithme général du solveur d'ACA (schématisé en Figure 317) fonctionne ainsi : pour chaque itération, chaque cellule est passée en revue. Le solveur sélectionne pour chaque attribut les règles qui concernent la cellule et qu'elle ne respecte pas déjà. Si l'état de l'attribut n'est pas fixe, le solveur sélectionne un autre état parmi la liste des états possibles, sinon l'état de la cellule est échangé avec celui d'une cellule voisine selon l'algorithme d'échange des états.

L'algorithme d'échange des états commence par établir la liste des voisins situés dans le rayon (nr) de la cellule indiquée par l'utilisateur. Si l'option aléatoire (type) est choisie, les cellules voisines sont sélectionnées aléatoirement, sinon elles sont choisies dans l'ordre (des cellules les plus proches au plus éloignées). Ainsi, tour à tour la compatibilité des cellules voisines avec la cellule concernée est vérifiée. Si la cellule voisine vérifiée est compatible pour échanger d'état avec la cellule concernée, alors l'échange est effectué et l'algorithme s'arrête, sinon il passe à la cellule voisine suivante.

Pour plus de précision sur le programme du solveur d'ACA, le code Python est disponible en annexe 3 (p.532).

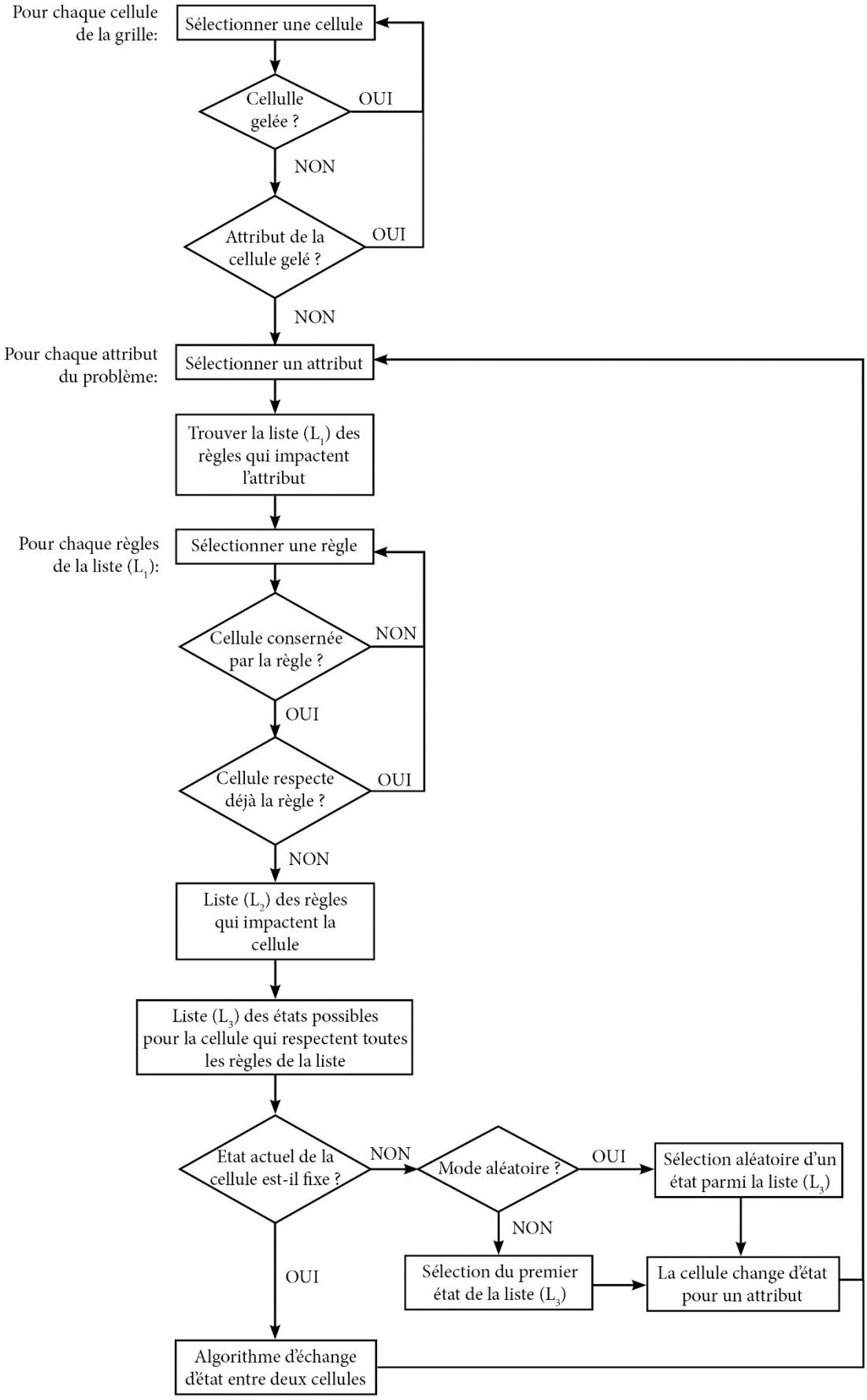


Figure 317 : Algorithme général du solveur d'ACA

4.2.1. Autres outils pour la réparation de solution

Le plugin Urchin contient d'autres composants utiles à la définition de fonction de réparation, notamment des composants pour créer des boucles conditionnelles et des systèmes d'auto-organisation.

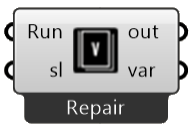
Boucles conditionnelles

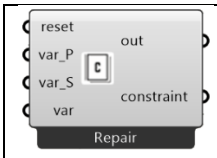
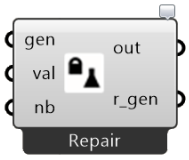
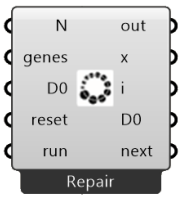
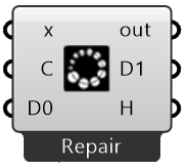
Description des composants

Plusieurs outils servent à faire des boucles conditionnelles avec Urchin (voir Tableau 15). Quatre composants servent uniquement à conditionner les valeurs des paramètres (variables, gènes) en fonction des uns et des autres. Il s'agit donc de boucles appliquant des contraintes uniquement sur des valeurs numériques. Trois de ces composants (« Ur_Variables_Definition », « Ur_Define_ Variable_Constraint » et « Ur_Variable_Constraint_Solver »), permettent de créer des contraintes sur les paramètres définis à l'aide de sliders. Ainsi, ces composants permettent de définir des contraintes du type : « Si l'épaisseur du bâtiment est supérieure à 10 mètres alors le ratio de surface vitré doit être supérieur à 50 % ». Un quatrième composant (« Ur_Genepool_Constraint_Variable ») s'utilise pour les variables définies avec le composant « Genepool » qui sert à créer plusieurs variables à la fois. Il permet de définir des contraintes suivantes : « Parmi les 100 panneaux qui composent la façades, 20 doivent être de type 2 ».

Deux autres composants (« Ur_Loop-Start », « Ur_Loop_End ») servent à faire des boucles à partir de tous les types de données. Ces composants fonctionnent un peu comme le plugin Anemone®, seulement ils sont compatibles avec le solver d'optimisation d'Urchin.

Tableau 15 : Liste des composants, données d'entrée et de sortie pour créer des boucles conditionnelles avec le plugin Urchin

Composants	Données entrées	Données sorties
Ur_Variables_ Definition 	**Run :** Saisir « True » pour faire fonctionner le composant. **sl :** Liste des noms (chaînes de caractères) de toutes les variables du problème. Les variables doivent être classées selon leur hiérarchie. Une variable ne peut pas avoir d'impact sur une variable située avant elle dans la liste.	**var :** Arbre de données contenant toutes les informations sur les variables du problème.
Ur_Define_ Variable_Constraint	**reset** : Saisir « True » avec un bouton pour faire apparaître la fenêtre de saisie des contraintes. **var_P** : Listes des noms (chaînes de caractères) des variables impactantes.	**constraint** : Liste de listes python qui permettent au solver de définir des contraintes.

	**var_S** : Listes des noms des variables impactées. Ces listes (var_P et var_S) doivent faire références aux surnoms (« Nickname ») donnés aux sliders. **var** : Arbre de données contenant toutes des informations sur les variables du problème.	
Ur_Variable_ Constraint_Solver 	**Run** : Saisir « True » pour démarrer le calcul. **genes** : Connecter tous les sliders du problème. Leurs surnoms (Nickname) doivent correspondre à ceux indiqués dans le composant « Ur_Varibles_Definition ». **var** : Arbre de données contenant toutes des informations sur les variables du problème. **C :** Liste de listes python qui permettent au solver de définir des contraintes.	**repair_genome**: Liste des nouvelles valeurs des variables qui correspondent aux génomes réparés. **variable_order** : Rappel de la hiérarchie des variables. Il s'agit de l'ordre dans lequel les sliders doivent être connectés au solveur d'optimisation.
Ur_Genepool_ Constraint_Variable 	**gen** : Connecter un composant « GenePool ». **var** : Liste des valeurs à contraindre (les valeurs données doivent appartenir au domaine saisie dans le GenePool. **nb** : Liste des quantités souhaitées (entiers naturels) pour chaque valeur donnée dans val.	**r_gen** : Liste des nouvelles valeurs des variables qui correspondent aux génomes réparés.
Ur_Loop_Start 	**N** : Slider avec le nombre d'itérations. **genes** : Connecter les sliders utilisés avec le solver d'optimisation. **D0** : Arbre de données génériques qui seront modifiées lors des itérations **reset** : Réinitialise la boucle. **run** : Saisir « True » pour démarrer la boucle.	**x:** Output à connecter au composant "Loop_End". **i** : Nombre d'itérations effectuées. **D0** : Données génériques.
Ur_Loop_End 	**x** : Connecter les données de sortie « x » du composant « Loop-Start ». **D0** : Données génériques modifiées dans la boucle. **C** : Condition pour arrêter la boucle (« True » pour mettre fin au calcul).	**D1** : Donnée réparée **H** : Historique de chaque itération.

Exemples de scripts

La Figure 318 montre comment programmer visuellement la contrainte « Si l'épaisseur du bâtiment est supérieure à 10 mètres alors le ratio de surface vitré doit être supérieur à 50 % » avec le plugin Urchin. Le composant « Ur_Define_Varible_ Constraint » permet d'afficher une fenêtre de saisie pour définir la contrainte.

La Figure 319 montre comment programmer visuellement la contrainte « Parmi les 100 panneaux qui composent la façade, 20 doivent être de type 2 ». Au départ, le GenPool compte 30 panneaux de type 2, après réparation il n'en compte plus que 20.

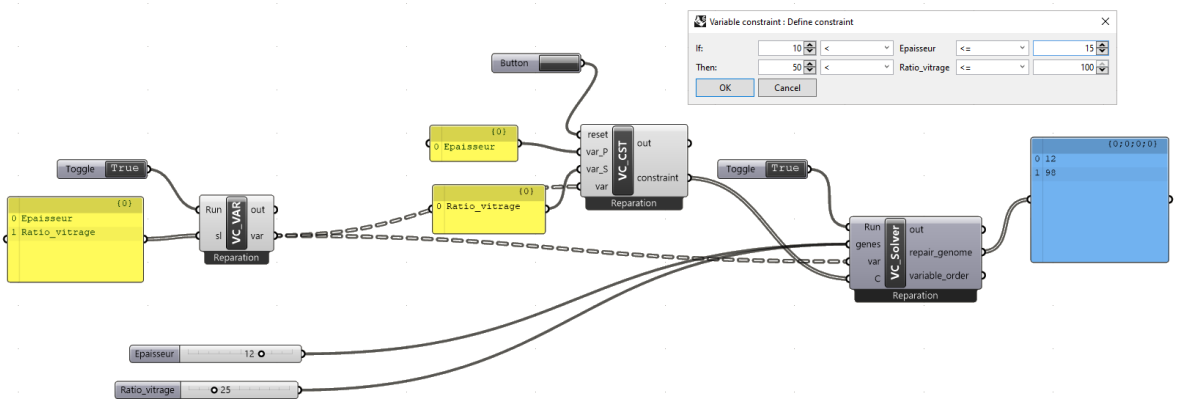


Figure 318 : Exemple de script pour contraindre des variables entre elles avec le plugin Urchin

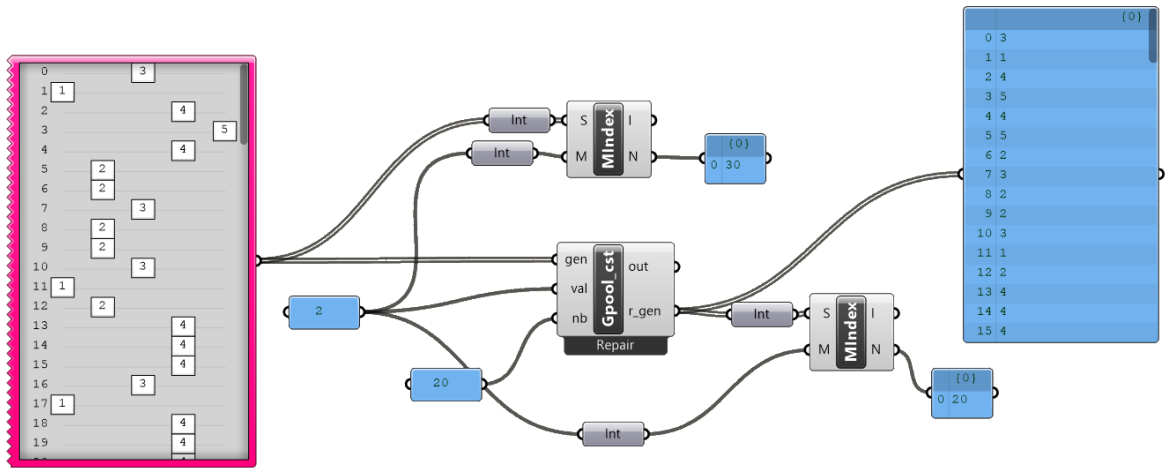


Figure 319 : Exemple de script pour contraindre les valeurs d'un Genepool avec le plugin Urchin

Auto-organisation 2D

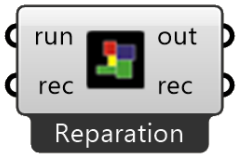
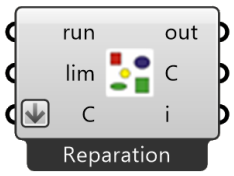
Description des composants

Deux composants permettent de faire de l'auto-organisation avec des courbes planes fermées (voir Tableau 16). Un premier composant, « Ur_Attraction_Rectangles », permet de concaténer entre eux des rectangles co-planaires. Ce composant peut être utilisé pour explorer la morphologie d'un bâtiment à la forme d'empreinte irrégulière ou pour l'allocation spatiale d'espaces intérieurs.

Le second composant, « Ur_Repulsion2D », permet d'éviter les collisions de courbes planes fermées situées dans un même plan. L'algorithme fonctionne avec des formes primitives (rectangles, triangles, cercles, ellipses, polygones…) mais aussi avec des formes libres. Ce composant sert principalement à faire de l'allocation spatiale de bâtiment ou de morceaux de

bâtiment, mais peut aussi être utilisé pour la conception d'enveloppe (allocation du vitrage ou de panneaux pour la protection solaire).

Tableau 16 : Liste des composants, données d'entrée et de sortie pour créer des systèmes d'auto-organisation en 2D avec le plugin Urchin

Composants	Données entrées	Données sorties
Ur_Attraction_Rectangles 	**run** : Saisir « True » pour démarrer la simulation. **rec** : Liste des rectangles coplanaires.	**rec** : Liste des rectangles réparés. **i** : Nombre d'itérations effectuées
Ur_Repulsion2D 	**run** : Saisir « True » pour démarrer la simulation. **lim** : Région dans laquelle les courbes doivent restées cantonnées. **C** : Liste des courbes planes fermées coplanaires.	**C** : Liste des courbes réparées **i** : Nombre d'itérations effectuées

Description des algorithmes

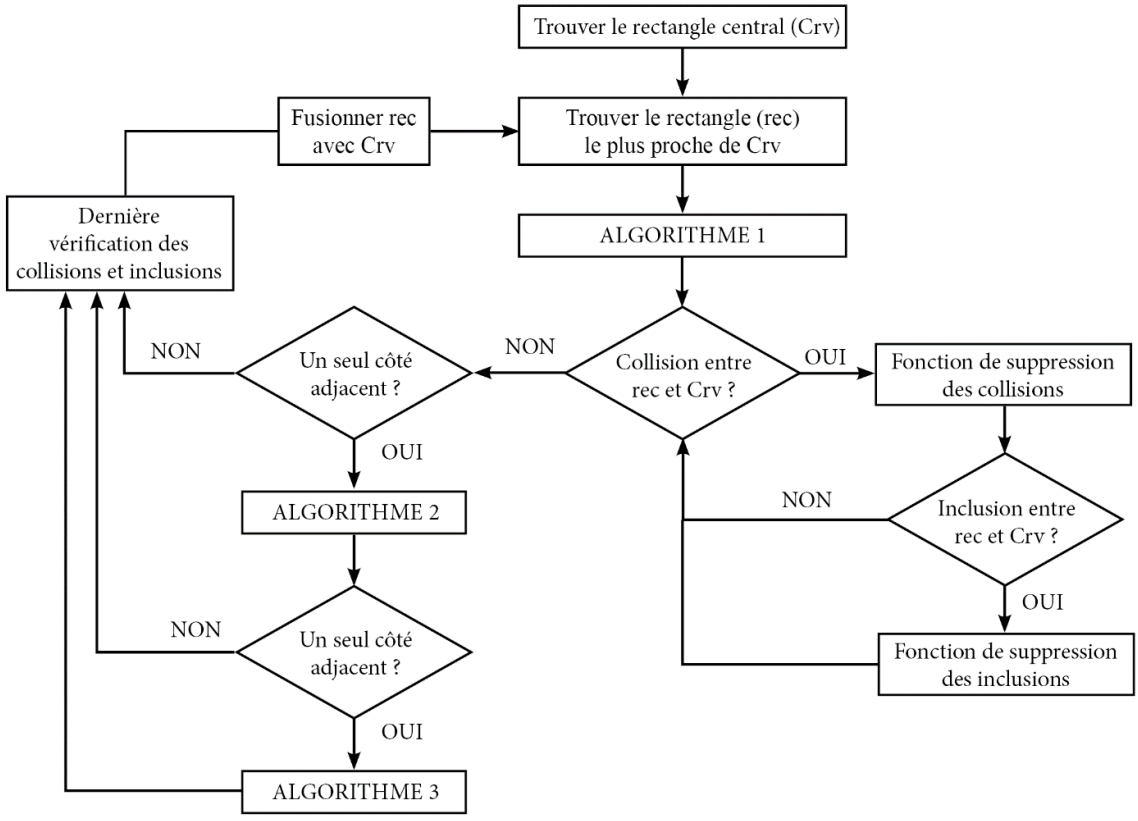


Figure 320 : Schéma du fonctionnement de l'algorithme d'attraction de rectangles 2D du plugin Urchin

En Figure 320 est schématisé le fonctionnement général de l'algorithme d'attraction de rectangles 2D et son code python se trouve en annexe 4 (p.532). L'algorithme d'attraction de rectangle n'est pas à proprement parler un MBA contrairement à l'algorithme de répulsion de courbe 2D. Il s'agit d'une méthode de génération procédurale (une simple suite d'instruction) dont certaines fonctions sont des MBA (comme la gestion des collisions). L'algorithme commence par identifier parmi l'ensemble des rectangles, le rectangle le plus central (Crv). Puis il va effectuer une procédure en trois étapes qu'il va répéter pour chaque rectangle restant (rec) en commençant toujours par le rectangle le plus proche :

(1) Le rectangle (rec) va être déplacé pour être adjacent à (Crv) en passant par le plus court chemin, il s'agit de l'algorithme 1 illustré en Figure 321.

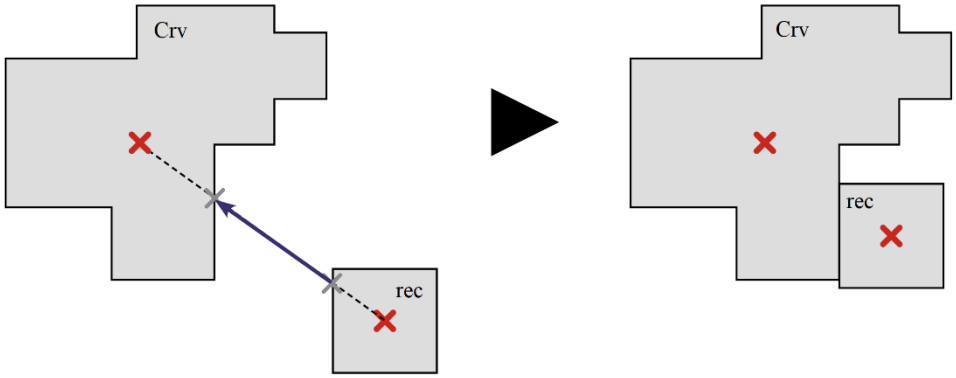


Figure 321 : Schéma de l'algorithme 1 du composant d'attraction de rectangles 2D

(2) Puis il va chercher à se déplacer de nouveau afin de trouver un second côté adjacent à (Crv), il s'agit de l'algorithme 2 illustré en Figure 322.

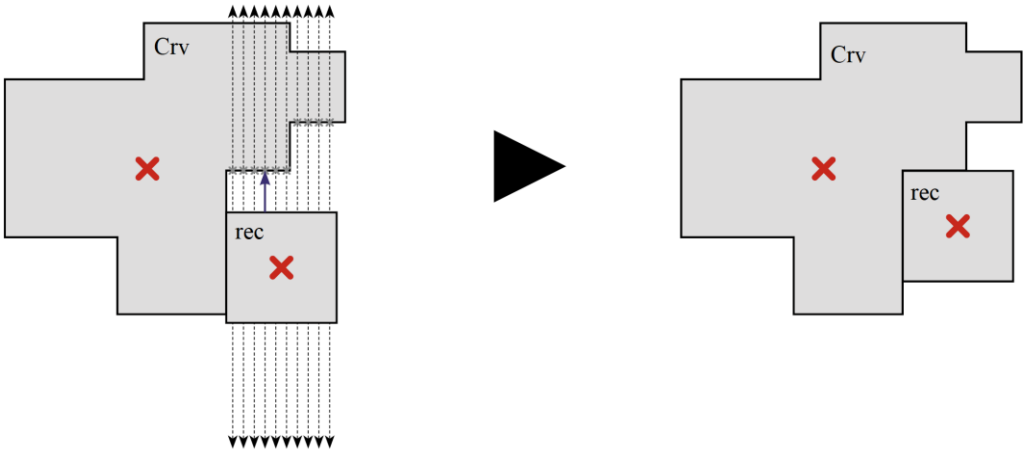


Figure 322 : Schéma de l'algorithme 2 du composant d'attraction de rectangles 2D

(3) S'il ne trouve pas de second côté adjacent, il va chercher à aligner (rec) avec un sommet de (Crv), il s'agit de l'algorithme 3 illustré en Figure 323.

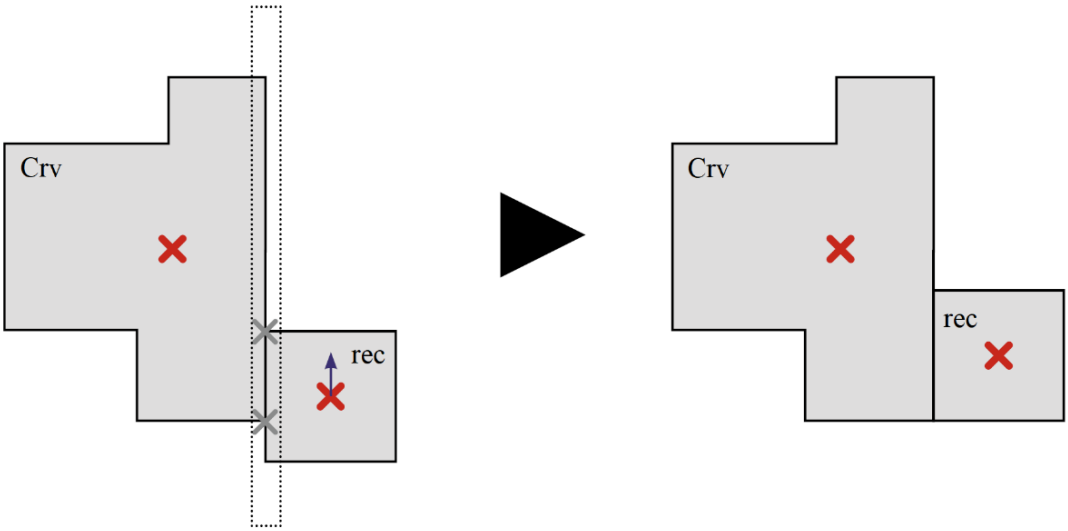


Figure 323 : Schéma de l'algorithme 3 du composant d'attraction de rectangles 2D

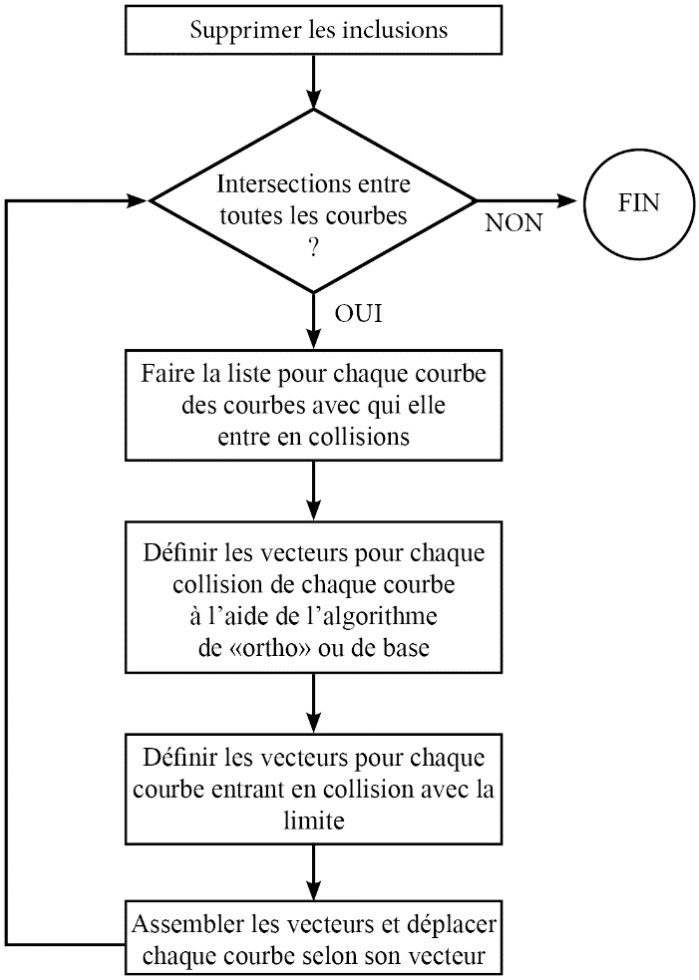


Figure 324 : Schéma du fonctionnement de l'algorithme de répulsion 2D du plugin Urchin

En Figure 324 est schématisé le fonctionnement général de l'algorithme de répulsion de courbes planes fermées et son code python se trouve en annexe 5 (p.532). Cet algorithme commence par vérifier si certaines boucles sont incluses dans d'autres avant de supprimer ces inclusions en déplaçant les courbes concernées. Puis il va effectuer une première boucle sur une centaine d'itérations durant laquelle les courbes seront déplacées uniquement avec des vecteurs orthogonaux lorsqu'il s'agit de collisions de courbes entre elle. Si cette première boucle ne permet pas d'obtenir le résultat attendu (à savoir des courbes n'entrant pas en collision), une seconde boucle est effectuée à l'aide d'une autre méthode. Les deux méthodes de répulsion utilisées dans cet algorithme sont illustrées en Figure 325.

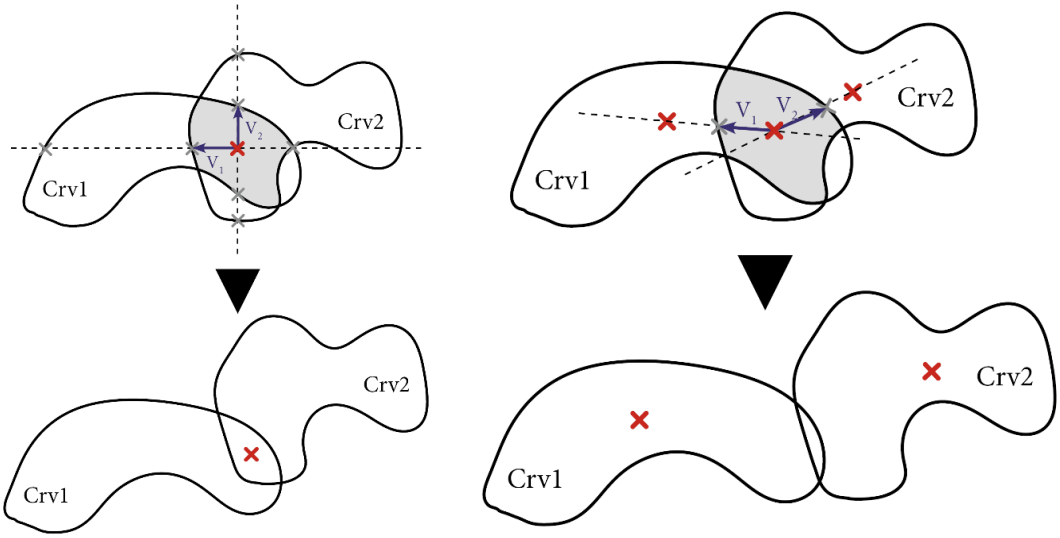


Figure 325 : Illustration des deux méthodes de répulsion, à gauche la méthode orthogonale, et à droite la méthode de base.

Il est aussi possible d'ajouter une contrainte en définissant une région limite dans laquelle l'ensemble des courbes doivent restaient cantonnées. Ainsi dans chaque boucle, des nouveaux vecteurs sont calculés selon la méthode décrites en Figure 326.

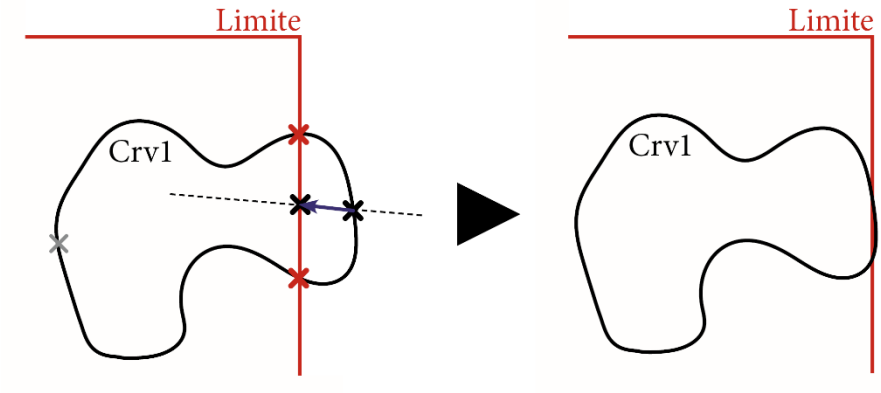


Figure 326 : Méthode de répulsion à l'intérieur de la région limite

4.2.3. Solveur et outils de visualisation

Solveur d'optimisation

Les composants pour l'optimisation

Avant de détailler le fonctionnement du composant principal du plugin *Urchin* qui permet de lancer un processus d'optimisation, il est nécessaire de présenter les trois composants utilitaires qui permettent d'assurer la qualité de l'exploration. Le composant « Ur_Timer » permet de mettre en pause le processus d'optimisation afin d'assurer que la solution en cours de génération est bien finalisée, c'est-à-dire que le processus de réparation, les analyses et la visualisation des résultats ont bien abouti. Il est particulièrement recommandé d'utiliser ce composant lorsqu'on utilise les boucles conditionnelles (Ur_LoopStart, Ur_LoopEnd), mais aussi lors de l'utilisation de simulations chronophages (consommations énergétique, confort thermique, et éclairement naturel annuel).

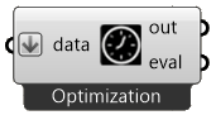
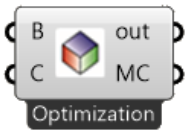
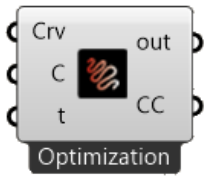
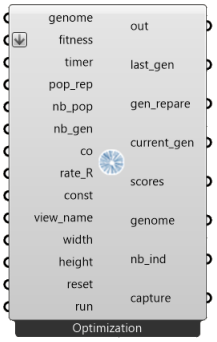
Deux composants permettent de transformer des objets géométriques (brep et surface) en maillages colorés (« Ur_ColorBrep ») et les courbes en courbes colorées (« Ur_ColorCurve »). Ces objets peuvent ainsi être connectés au composant « Ur_Timer » afin d'assurer que les captures d'écran automatiques réalisées pendant le processus d'optimisation ne précèdent pas la visualisation comme cela peut survenir avec le plugin Colibri habituellement utilisé pour faire des captures d'écran avec les autres solveurs d'optimisation.

Le composant « Ur_Optimization_Solver » permet de lancer le processus d'optimisation. Un seul algorithme est pour le moment implémenté, il s'agit de l'algorithme évolutionnaire multicritère NSGAII (Deb et al., 2002). Il contient trois modes d'utilisation :

(1) Sans contrainte, ou avec des fonctions de pénalisations (voir p. 327), ou avec des fonctions de réparation Baldwinienne (p. 335).
(2) Avec Constraint-NSGAII qui est une méthode d'intégration des contraintes interne spécifique à NSGAII (p. 330).
(3) Avec des fonctions de réparation Lamarckienne (voir p. 335).

Le détail des données d'entrée et de sortie de ces 4 composants est listé dans le Tableau 17, et un exemple de paramétrage pour une optimisation avec fonction de réparation de trois critères (lumière, radiation, et vue) est illustrée en Figure 327. Par défaut, le solveur cherche à minimiser les fonctions objectifs. Pour maximiser une fonction, l'utilisateur devra ajouter un composant Grasshopper pour calculer son négatif.

Tableau 17 : Tableau des données d'entrée et de sortie des composants d'optimisation du plugin Urchin

Composants	Données entrées	Données sorties
Ur_Timer 	**data** : Liste de données génétiques (résultats des fonctions objectifs, géométrie colorée à visualiser)	**eval** : valeur booléenne à connecter avec l'input « timer » du solveur d'optimisation.
Ur_ColorBrep 	**B** : Liste de géométrie (brep, surface). **C** : Couleur.	**MC** : Liste de maillages colorés pouvant être connectés à un timer.
Ur_ColorCurve 	**Crv** : Liste de courbes **C** : Couleur. **t** : Epaisseur du trait	**CC** : Liste de courbes colorées pouvant être connectées à un timer.
Ur_Optimization_Solver 	**genome** : Liste des variables du problème (sliders et Genepool uniquement). **fitness** : Liste des valeurs des fonctions objectifs à minimiser. **timer** : Booléen qui indique si toutes les données essentielles ont été calculées ou non. Utiliser le composant "Ur_Timer". **pop_rep** (option): Liste des génomes préparés (dans le même ordre que le génome). **nb_pop** : Taille de la population. **nb_gen** : Nombre de générations. **co** (option): Taux de violation des contraintes. **reset** : Appuyer sur le bouton pour faire apparaître la fenêtre de saisie. **rate_R** (option): Taux de remplacement (un nombre entre 0 et 1) pour les fonctions de réparation. **const** : Méthode de gestion des contraintes (0 : sans contraintes, 1 : Constraint-NSGAII, 2 : Fonction de réparation) **view_name** (option): Liste des noms de fenêtres de visualisation Rhino pour faire des captures d'écrans. **width** : Largeur de l'image en pixels (600 par défaut). **height** : Hauteur de l'image en pixels (600 par défaut). **reset** (réinitialiser) : Redémarre l'algorithme.	**last_gen** : Valeurs des fonctions objectifs de la dernière génération calculée. **gen_repare** : Valeurs des gènes réparés de toutes les générations. **current_gen** : Numéro de la génération en cours de calcul. **scores** : Valeurs des fonctions objectifs de toutes les générations. **genome** : Valeurs des gènes de toutes les générations. **nb_ind** : Nombre d'individus évalués dans la génération actuelle. **capture** : Chemins d'accès aux captures d'écran.

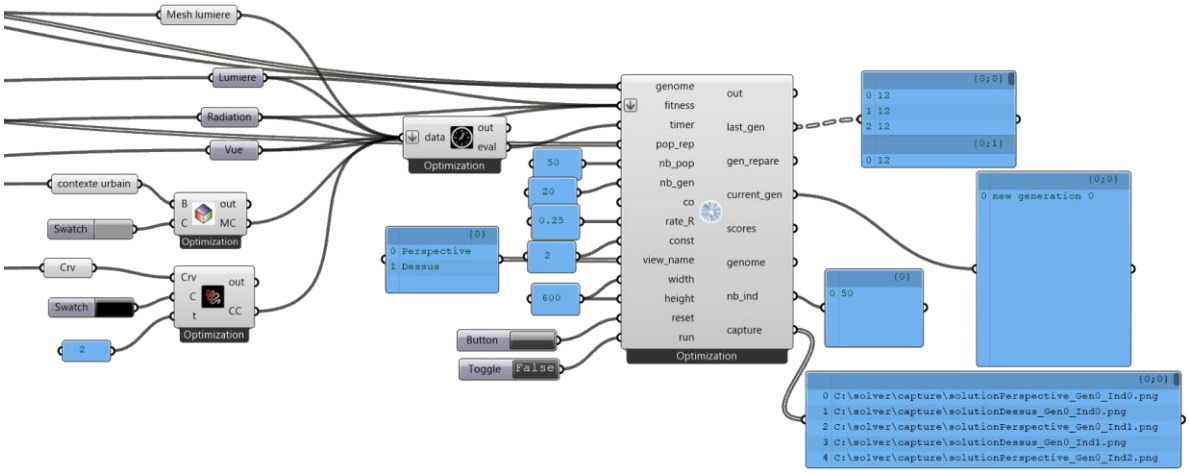


Figure 327 : Paramétrage du solveur d'optimisation Urchin

Architecture du solveur d'optimisation

Le composant « Ur_Optimization_Solveur » a été codé en Python (voir le code en annexe 6, p. 532). Comme l'illustre la 441, il ne contient pas d'algorithme d'optimisation. Il récolte et transmet les données d'entrée nécessaires à un script en Python extérieur pour fonctionner. Celui-ci est placé dans un dossier nommé « solver » sur le C:/ qui lui contient NSGAII. Cela nous a permis d'utiliser des librairies Python 3, notamment pymoo qui permet de faire de l'optimisation (Blank & Deb, 2020). Ce dossier contient trois sous-dossier :

(1) Le dossier « capture » qui sert à stocker les captures d'écran réalisées aux cours du processus d'exploration

(2) Le dossier « fichiers » où sont stockés les fichiers textes par lesquels transitent les données entre Grasshopper et NSGAII.

(3) Le dossier « scripts » où se trouvent les scripts python qui permettent de faire tourner NSGAII et certaines fonctionnalités des composants de visualisation qui seront présentées un peu plus tard.

L'utilisation du bouton « reset » permet de vider le contenu des dossiers « capture » et « fichiers ». Aussi, deux composants Urchin ne peuvent pas fonctionner de façon simultanée sur un même fichier ou sur deux fichiers différents exécutés sur un même ordinateur.

Pour pouvoir fonctionner avec Grasshopper, certaines fonctions de pymoo ont été légèrement modifiées notamment les classes « Algorithm », « GeneticAlgorithm », et

« NSGAII » afin d'utiliser leurs fonctions sans lancer la boucle générale de l'algorithme. Quelques fonctions ont aussi été ajoutées comme la fonction « lamarckian » qui permet d'utiliser des fonctions de réparation en remplaçant le génome pour une partie ou la totalité des solutions réparées.

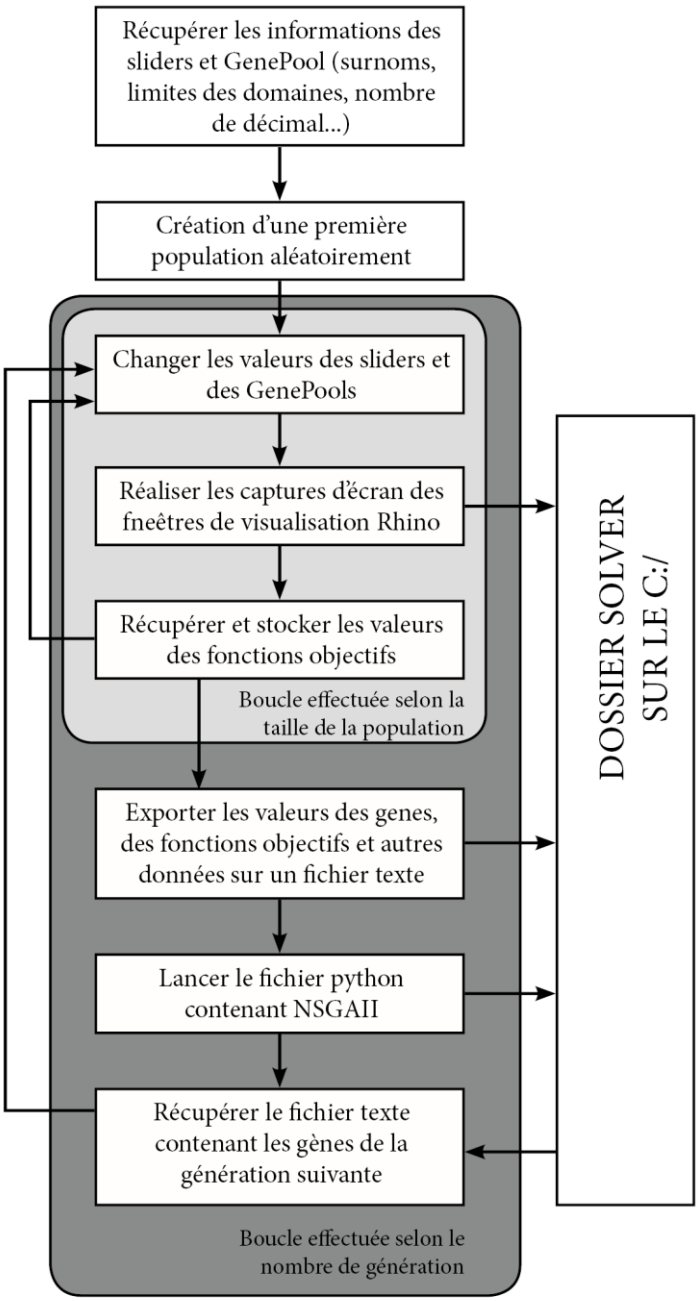


Figure 328 : Description du fonctionnement du composant "Ur_Optimization_Solver" du plugin Urchin

Outils d'analyse des résultats

Une fois le processus d'optimisation terminé, certains composants *d'Urchin* permettent d'analyser directement les résultats dans Grasshopper® à l'aide de graphique. Pour cela, il faut soit utiliser les données de sortie du solveur, si le fichier Grasshopper n'a pas était fermé suite à l'exploration, ou si les données ont été internalisées dans un stockeur de donné. Dans le cas contraire, il est possible d'utiliser le composant « Ur_import_results » pour importer les résultats de l'exploration stockés dans le fichier csv qui se trouve dans le dossier C:\solver\fichiers.

Avant l'analyse, il est parfois nécessaire d'éliminer quelques solutions de l'ensemble des solutions exploré, soit parce qu'une erreur est survenue durant l'analyse environnementale, soit parce qu'une fonction de pénalité a été utilisée pour gérer des contraintes. Dans ce cas, le composant « Ur_remove_penalty » permet d'évincer les solutions dont les scores des fonctions objectifs ne se situent pas dans un domaine de valeurs précis.

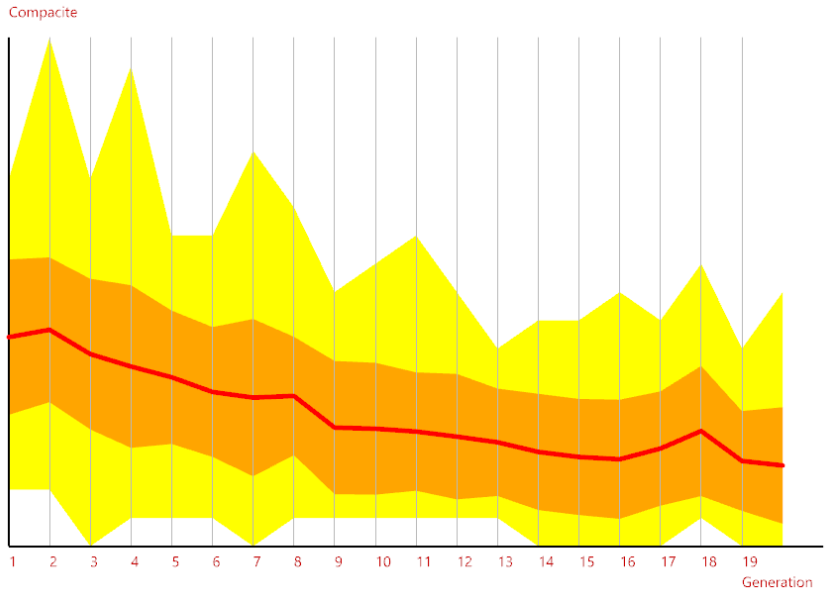


Figure 329 : Graphique généré via le composant « Ur_Fitness_Statistics » du plugin Urchin

On peut ensuite analyser l'évolution des scores par fonction objectif au cours des générations à l'aide du « Ur_Fitness_Statistics ». Dans le graphique généré (voir Figure 329), la surface jaune représente le domaine de valeur que prend la fonction à chaque génération, la surface orange représente la dispersion des valeurs, et la ligne rouge représente la valeur moyenne à chaque génération. Dans cet exemple, on observe que la fonction converge un peu plus à chaque génération vers une valeur minimale.

De plus, on peut rechercher des corrélations entre les valeurs des gènes et des fonctions objectifs en utilisant le composant « Ur_Parallel_Coordinate_Diagram » qui permet de générer des diagrammes de coordonnées parallèles comme sur la Figure 330.

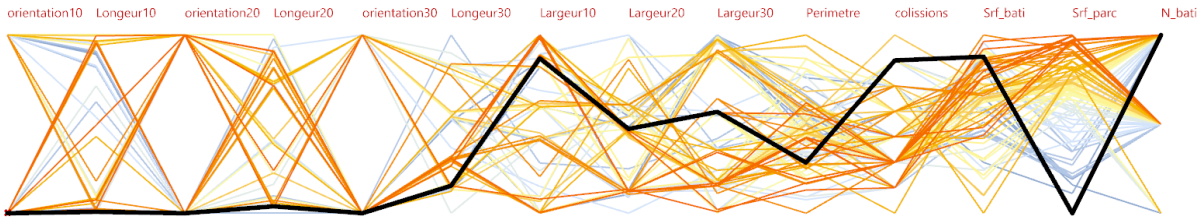


Figure 330 : Graphique généré via le composant « Ur_Parallel_Coordinate_Diagram » du plugin Urchin

Parmi les options de ce composant, il est possible de choisir les couleurs du dégradé, de choisir le paramètre (gène ou fonction objectif) à l'origine du dégradé (dans notre exemple il s'agit de la fonction « Srf_parc »), mais aussi de faire ressortir la polyligne représentant une solution précise en sélectionnant une génération puis une solution.

Enfin, le composant « Ur_Scatterplot » permet de créer un graphique en nuage de points 2D ou 3D en sélectionnant deux ou trois fonctions objectifs.

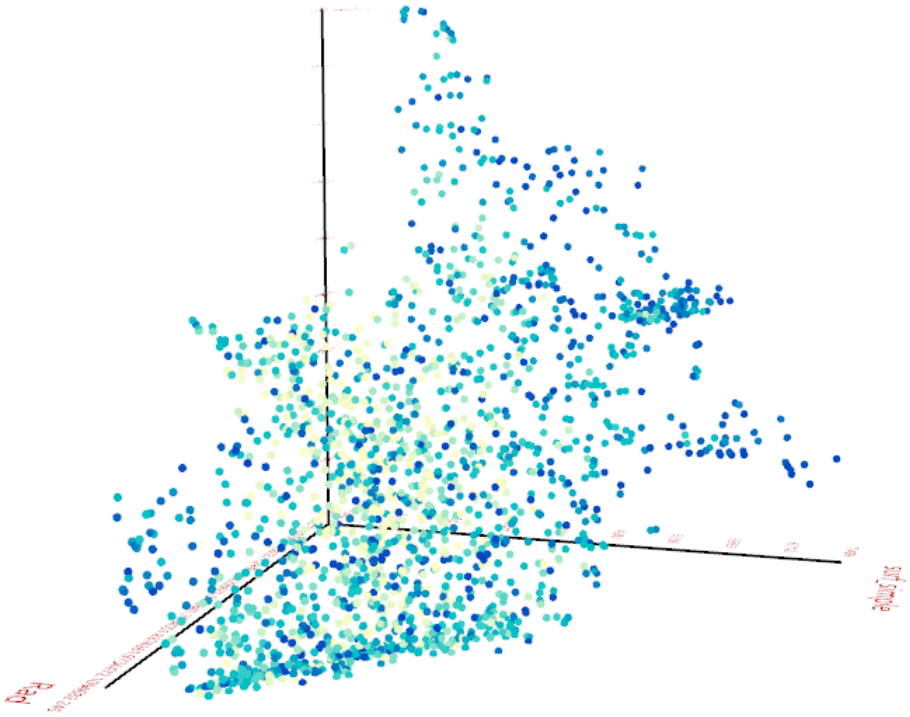
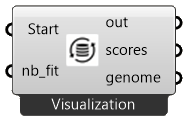
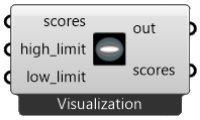
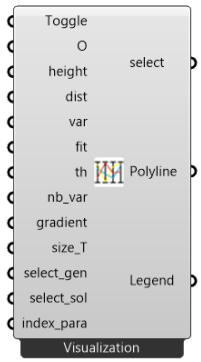
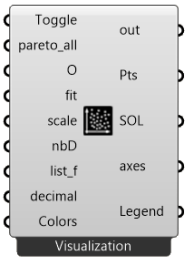


Figure 331 : Graphique généré via le composant « Ur_Scatterplot » du plugin Urchin

L'ensemble des données d'entrée nécessaires pour faire fonctionner ces composants ainsi que les données résultantes sont listées dans le Tableau 18.

Tableau 18 : Liste des données d'entrée et de sortie des composants de visualisation du plugin Urchin

Composants	Données entrées	Données sorties
Ur_Import_results 	**Start** : Saisir « True » pour importer les données. **Nb_fit** : Préciser le nombre de fonctions objectifs.	**scores** : Arbre de données contenant les valeurs des fonctions objectifs. **genome** : Arbre de données contenant les valeurs des gènes .
Ur_Remove_penalty 	**scores** : Arbre de données contenant les valeurs des fonctions objectifs. **high_limit** : Liste contenant la valeur limite haute pour chaque fonction objectif. **low_limit** : Liste contenant la valeur limite basse pour chaque fonction objectif.	**scores** : Arbre de données contenant les nouvelles valeurs des fonctions objectifs.
Ur_Fitness_Statistics 	**Toggle** : Saisir « True » pour lancer le calcul. **O** : Point d'origine du graphique. **dist** : Distance entre deux générations. **height** : Hauteur du graphique. **fit** : Valeurs de la fonction objectif **I_fit** : Un nombre entier pour sélectionner une fonction objectif. **size_T** : échelle de la légende.	**axes** : Axe Y et axe X du graphique. **M_ecart** : Surface maillée représentant les écarts types par génération. **M_DOM** : Surface maillée représentant les domaines de valeurs par génération. **LC_moy** : Courbe représentant les valeurs moyennes par génération. **Legend** : Textes composant la légende.
Ur_Parallel_ Coordinate_Diagram 	**Toggle** : Saisir « True » pour lancer le calcul. **O** : Point d'origine du graphique. **height** : Hauteur du graphique. **dist** : Distance entre deux générations. **var** : Arbres contenant les valeurs des gènes de toutes les générations. **fit** : Arbres contenant les valeurs des fonctions objectifs de toutes les générations. **th** : Épaisseur des lignes. **nb_var** : Sélectionner l'index d'une variable de référence pour la coloration du diagramme. **gradient** : Liste optionnelle de couleurs pour créer un dégradé. **size_T** : Échelle de la légende. **select_gen** : Sélectionner une génération. **select_sol** : Sélectionner une solution parmi la génération sélectionnée.	**Select :** Ligne de la solution sélectionnée. **Polyline** : Ensemble des lignes colorées du graphique. **Legend** : Textes composant la légende.

	index_para : Une liste d'entiers pour sélectionner les paramètres (gènes et fonctions objectifs).	
Ur_Scatterplot 	**Toggle** : Saisir « True » pour lancer le calcul. **pareto_all** : Spécifier si les calculs du front de Pareto doivent prendre en compte toutes les fonctions objectifs (True), ou seulement celles saisies dans l'entrée « list_f » (False). **O** : Point d'origine du graphique. **fit** : Arbres contenant les valeurs des fonctions objectifs de toutes les générations. **scale** : Échelle du graphique. **nbD** : Nombre de dimensions (2 pour 2D, 3 pour 3D). **list_f** : Liste des fonctions objectifs prises en compte (numéro d'index). **Decimal** : Nombre de décimales dans la légende. **Colors** : Liste optionnelle de couleurs pour créer un gradient.	**Pts** : Nuage de points **SOL :** Sphères ou disque maillés représentant les solutions. **axes** : Axe Y, X et Z du graphique. **Legend** : Textes composant la légende.

Outils pour générer un catalogue des meilleurs solutions

Cinq autres composants de visualisation sont inclus dans le plugin Urchin. Ils permettent de générer des catalogues de solutions, voir des catalogues contenant exclusivement les meilleures solutions générées lors de l'exploration.

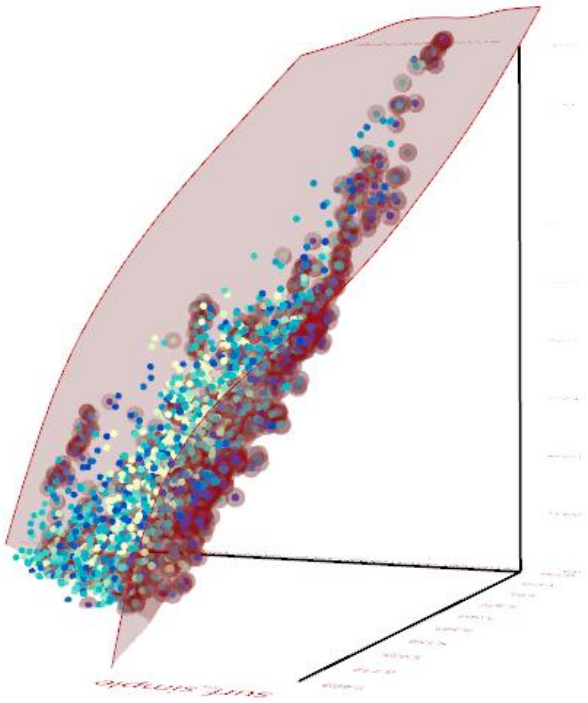


Figure 332 : Graphique généré via les composants « Ur_Scatterplot » et « Ur_Pareto_Front »

Pour cela, un premier composant, « Ur_Pareto_Front » permet de sélectionner les solutions générées par rang (le meilleur rang étant le Front de Pareto) en utilisant les données de sortie du composant « Ur_Scatterplot » déjà présenté. Ainsi, il permet de visualiser le Front de Pareto sur le graphique en nuage de points 2D ou 3D (voir Figure 332) et de générer une liste de listes contenant pour chaque solution sélectionnée le numéro de la génération à laquelle elle appartient et son numéro d'apparition dans cette génération. Cette liste permettra d'initialiser les autres composants nécessaires à la création du catalogue.

Il arrive que certaines solutions du font de Pareto aient exactement les mêmes gènes. Il n'est alors pas nécessaire de faire figurer deux solutions identiques dans le catalogue. Pour éviter cela, il faut utiliser le composant « Ur_Remove_Duplicates » qui permet d'actualiser la liste contenant les numéros des meilleurs solutions en supprimant les doublons.

Une dernière option avant la création du catalogue est la génération d'un graphique en diamant à l'aide du composant « Ur_Radar_Chart_propre » propre à chaque solution sélectionnée. Il permet de situer la performance de chaque solution par rapport à l'ensemble des solutions générées. Comme le montre la Figure 333, ce composant rappelle aussi en légende les numéros de la solution, les scores et les classements de la solution pour chaque fonction objectif calculée.

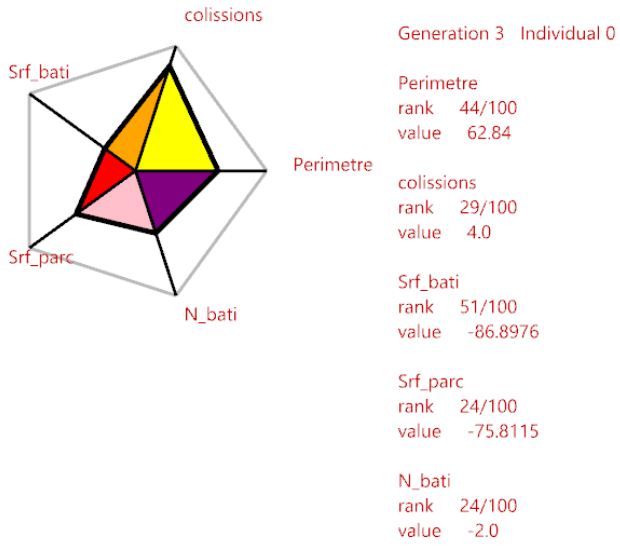


Figure 333 : Graphique généré via le composant « Ur_Radar_Chart » du plugin Urchin

Pour pouvoir intégrer ces graphiques au catalogue, une capture d'écran doit être effectuée pour chaque solution de la sélection. Ainsi, un quatrième composant permet de lancer des boucles pour faire des captures d'écran automatiques des graphiques qui seront stockées dans le dossier « C:\solver\capture\capture_radar ». Il est conseillé d'utiliser le composant « Ur_Timer » pour

réaliser ce type de boucle. Ainsi, pour pouvoir créer un catalogue illustré avec des graphiques personnalisés pour chaque solution, les composants doivent être assemblés comme sur la Figure 334.

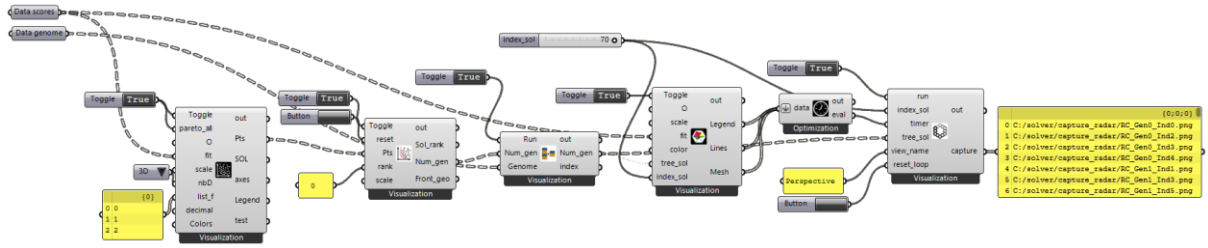


Figure 334 : Script pour réaliser des captures d'écran de graphiques personnalisés par solution

Il est aussi possible de réaliser des captures d'écran de diagramme de coordonnées parallèle individualisé par solution avec le composant « Ur_Parallel_Coordinate_Diagram » où chaque solution peut être mise en avant à l'aide des paramètres « select_gen » et « select_sol » . Cela est aussi possible avec des graphiques en nuage de points en utilisant le paramètre de sortie « Pts » du composant « Ur_Scatterplot » qui permet de récupérer un point représentant une solution particulière et de la mettre en avant dans le graphique.

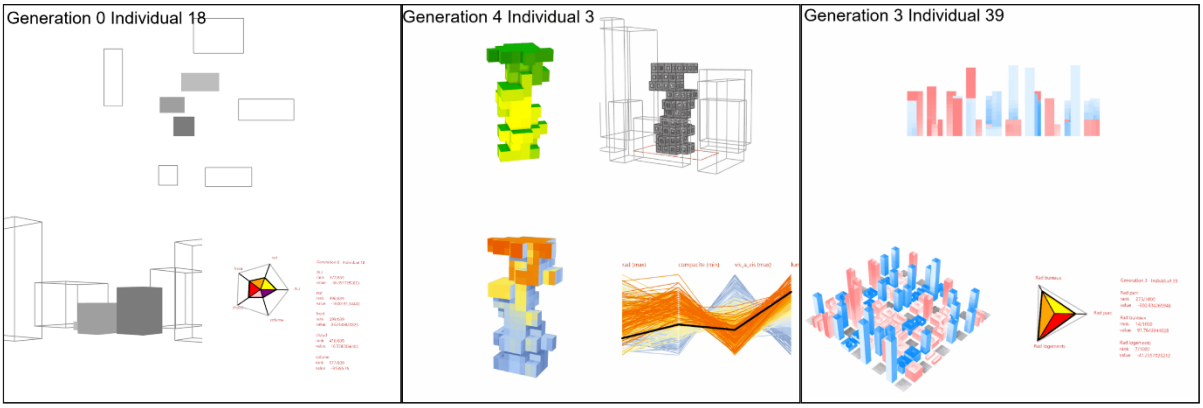


Figure 335 : Exemple de vignettes réalisées par le programme de génération de catalogue de solution

Enfin, un dernier composant, « Ur_Poster_Solution_Selection » permet de générer un pdf où sont rassemblées toutes les solutions sélectionnées. Pour fonctionner, ce composant lance l'exécution en arrière-plan un programme développé en python 3 dont le code est présenté en annexe 7 (p.532). Le programme commence par générer des vignettes pour chaque solution (stockées dans « C:\solver\capture\assembly ») dont quelques exemples sont présentés en Figure 335. Dans ces vignettes sont assemblées les différentes captures d'écran réalisées pendant la phase d'exploration (stockées dans « C:\solver\capture\solution »), et celles réalisées lors de la phase d'analyse (stockées dans « C:\solver\capture\capture_radar »). Les vignettes sont ensuite assemblées sur des pages (voir Figure 336) pour créer un catalogue au format pdf.

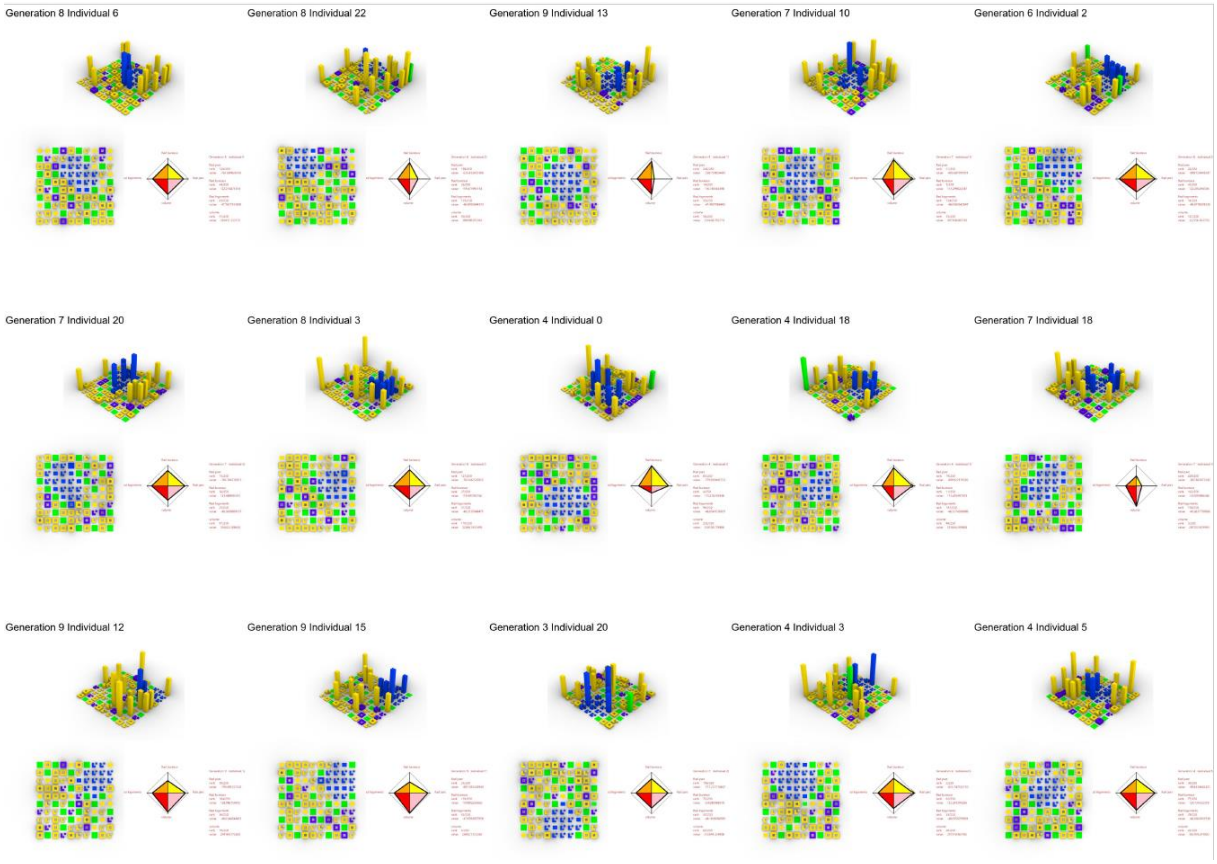
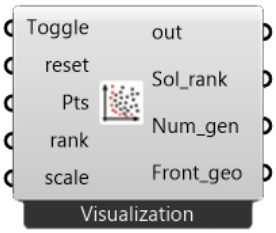
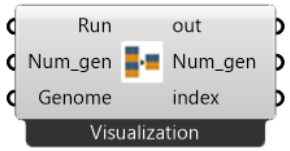
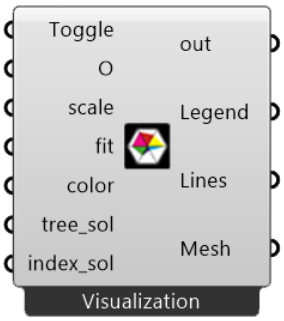
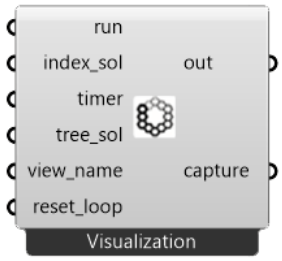


Figure 336 : Exemple de catalogue généré à l'aide du composant " Ur_Poster_Solution_Selection " du plugin *Urchin*

L'ensemble des données d'entrée nécessaires pour faire fonctionner ces composants ainsi que les données résultantes sont listées dans le Tableau 19.

Tableau 19 : Liste des données d'entrée et de sortie des composants nécessaires à l'export d'un catalogue de solutions avec le plugin Urchin

Composants	Données entrées	Données sorties
Ur_Pareto_Front 	**Toggle** : Saisir « True » pour démarrer le calcul. **reset** : Redémarre l'algorithme de tri. **Pts** : Liste des points 3D du composant « Ur_Scatterplot ». **rank** : Le rang que vous voulez sélectionné (0 étant le front de Pareto et la valeur par défaut). **scale** : Échelle de la solution sélectionnée.	**Sol_rank** : sphères ou disques maillés représentant toutes les solutions du rang sélectionné. **Num_gen** : Numéro de la génération et de l'individu de chaque solution sélectionnée. **Front_geo** : Une surface (3D) ou une courbe (2D) reliant tous les points représentant toutes les solutions du rang sélectionné.
Ur_Remove_Duplicate	**Run** : Saisir « True » pour démarrer les calculs. **Num_gen** : Arbre de données contenant pour chaque solution leur numéro de la génération et de la	**Num_gen** : Nouvel arbre donnée sans les doublons. **Index** : index des solutions

	solution. L'arbre peut contenir tout ou une partie de l'ensemble de solutions généré pendant l'exploration. **Genome** : Arbre de données contenant les valeurs des gènes de toutes les générations.	
Ur_Radar_Chart 	**Toogle** : Saisir "True" pour créer le graphique. **O** : Point d'origine du graphique. **scale** : Échelle du graphique. **fit** : Arbres de données contenant les valeurs des fonctions objectifs de toutes les générations. **color** : Liste optionnelle de couleurs (de la même longueur que le nombre de fitness). **tree_sol** : Arbre de données avec le nombre d'individus dans la génération et le nombre de générations. Liste de listes avec 2 éléments. Peut être générée avec le composant "ParetoFront" (Num_gen). **index_sol** : Sliders d'entiers naturels pour choisir une solution de « tree_sol ».	**Legend** : Textes qui composent la légende. **Lines** : Lignes du graphique. **Mesh**: Maillages colorées du radar.
Ur_Loop_Chart_Capture 	**run** : Saisir "True" pour démarrer la boucle. index_sol : Slider d'entiers naturels pour choisir une solution dans l'arbre (même slider que le composant "RadarChart"). timer : tree_sol : Arbre de données avec le nombre d'individus dans la génération et le numéro de génération. Liste de listes avec 2 éléments. Peut être générée avec le composant "ParetoFront" (Num_gen). **view_name** : Nom de la fenêtre Rhino qui sera capturée. **reset_loop** : Utiliser le bouton pour redémarrer la boucle (supprimera tous les éléments du dossier « C:\solver\capture\capture_radar »)	**capture** : Liste des chemins d'accès aux captures d'écran.
Ur_Poster_Solution_Selection	**run** : Saisir "True" pour générer un fichier pdf. **reset**: Utiliser un bouton pour réinitialiser la génération du fichier pdf. **tree_sol** : Arbre de données contenant pour chaque solution leur numéro de la génération et de la solution. Liste de listes avec 2 éléments. **resolution** : Définir 75, 150 ou 300 pour choisir un niveau de résolution. **size** : Entier naturel pour choisir un format	**path** : Chemin d'accès au fichier pdf.

	(0 : A0, 1 : A1, 2 : A2, 3 : A3, 4 : A4). **format** : Sélectionner le format du pdf avec un entier : soit paysage (0), soit portrait (1). view_name : Sélectionne les vues qui apparaîtront sur le pdf. Les vues sélectionnées doivent correspondre à la vue indiquée dans le solveur au moment du lancement du processus d'optimisation. (entre 1 et 3 vue). **number_X** : Nombre de solutions présentées horizontalement. **title** : Nom du document pdf. **radar** : Indiquer s'il faut (« True ») ou non (« False ») ajouter les graphiques capturés avec « Ur_Loop_Chart_Capture » dans le pdf.	

Au terme de cette seconde partie du chapitre 4, nous avons proposé un nouveau plugin d'optimisation pour Grasshopper®, « *Urchin* », qui a pour objectif de faciliter l'exploration numérique via des algorithmes d'optimisation dans la pratique des architectes. Ainsi, il contient un solveur d'optimisation avec NSGAII qui peut être utilisé avec des méthodes d'intégration des contraintes adaptée aux problèmes des architectes, notamment les fonctions de réparation avec une approche lamarckienne.

Des outils pour faciliter la programmation de fonction de réparation nécessitant l'usage de techniques génératives complexes ont aussi été intégrés à *Urchin*, notamment pour les modèles à base d'agents fixes comme notre automate cellulaire augmenté réservé aux problèmes à deux dimensions. Des outils pour faire de l'auto-organisation d'éléments en 2D ont aussi été intégrés avec des MBA mobiles.

Des outils pour la visualisation, comme on peut en trouver dans le plugin Wallacei ont été développés. Il a été délibérément choisi de prioriser ce type d'outils à une interface pour le solveur d'optimisation comme on peut en trouver sur Galapagos® ou Octopus® car nous les avions très peu utilisés finalement lors de nos expérimentations sur des projets d'AS. En effet, celles-ci ne nous semblent pas utiles, les calculs tournant la plupart du temps en l'absence de l'architecte. Il ne reste plus désormais qu'à vérifier la pertinence de ces outils à l'aide d'applications concrètes, ce qui est l'objet de la dernière partie de ce chapitre.

4.3 Applications du plugin sur différents types de problèmes

Pour tester le plugin Urchin nous avons imaginé pour cette troisième partie du chapitre 4, six problèmes inspirés des cas rencontrés dans la littérature scientifique ou dans la pratique professionnelle. Consciente que ce plugin est le fruit de recherches principalement réalisée sur l'enveloppe du bâtiment, un sujet plus accessible que les autres, nous avons choisi de nouveaux types de problèmes, pour interroger la robustesse de la méthode. Deux problèmes portent sur la morphologie du bâtiment, deux problèmes portent sur l'échelle urbaine et deux problèmes portent sur l'agencement des espaces intérieurs. Plus que les résultats des explorations en eux même, l'intérêt de ces expérimentations est de comprendre si le plugin Urchin permet véritablement d'étendre notre capacité à modéliser des problèmes de conception contraints.

4.3.1. Applications à la morphologie du bâtiment

Approche voxellaire pour explorer la morphologie d'un bâtiment

Description du problème

Pour cette première application nous avons choisi d'interroger la morphologie d'un bâtiment de grande hauteur dans un contexte urbain dense. Ce problème fait écho à une expérimentation réalisée pour le concours pour l'école européenne et le nouveau collège les renardières à Courbevoie. Le contexte urbain de La Défense rend l'accès à l'ensoleillement et à la lumière particulièrement complexe.

Puisqu'il s'agit d'une expérimentation sans esquisse préalable à optimiser, nous proposons de tester l'approche voxellaire (où l'on utilise des voxels) sur ce problème. Nous avons déjà vu au chapitre 1 (p.107) que cette méthode permet de générer des formes diversifiées (Marsault & Torres, 2019; Menges, 2012). Cependant, nous avons déjà évoqué au chapitre 3 (p.319) que cette approche a tendance à générer des aberrations (comme des voxels en lévitation). Il est donc nécessaire d'intégrer des contraintes au processus d'exploration pour éviter ce type de solutions.

Quatre fonctions objectifs sont prises en compte dans l'optimisation : la qualité de la vue avec une analyse des vis-à-vis avec le contexte urbain, une analyse de la radiation solaire en hiver, une analyse de l'éclairement naturel à 10h le 21 décembre et une mesure de la compacité.

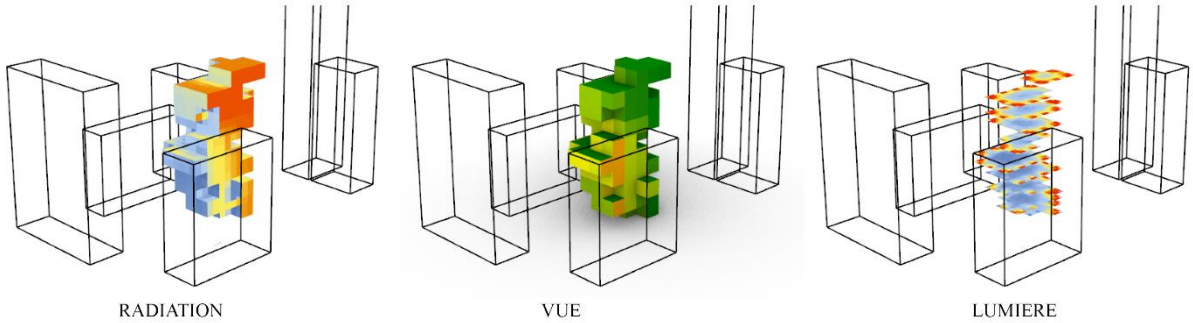


Figure 337 : Les trois analyses pour le confort pris en compte dans l'exploration

L'ensemble des analyses sont réalisées avec les outils présentés au chapitre 1 (p. 116), la fonction de réparation qui est une boucle conditionnelle, l'optimisation, et les images et graphiques générés pour l'analyse ont été réalisés avec les composants de notre plugin Urchin. Les calculs ont duré environ 25 heures pour 20 générations de 25 individus chacune.

Description du modèle

Pour pouvoir intégrer nos contraintes nous avons utilisé dans un premier temps le composant « Ur_Genepool_constraint_variable ». Un composant Genepool est utilisé pour pouvoir spécifier pour chacun des 500 voxels s'il est plein (1) ou vide (0), ainsi notre composant nous a servi à contraindre le nombre de voxel pleins à 100.

Dans un second temps, nous avons utilisé les composants « Ur_Loop_Start » et « Ur_Loop_End » pour créer un algorithme d'attraction des voxels plein par étages. A chaque itération et chaque étage, les voxels sont attirés en eux jusqu'à ce que l'union booléenne des voxels par étage ne forme qu'un unique volume. La Figure 338 montre les différentes étapes de la modélisation d'une solution allant de la discrétisation de l'espace constructible à la disposition du vitrage en façade.

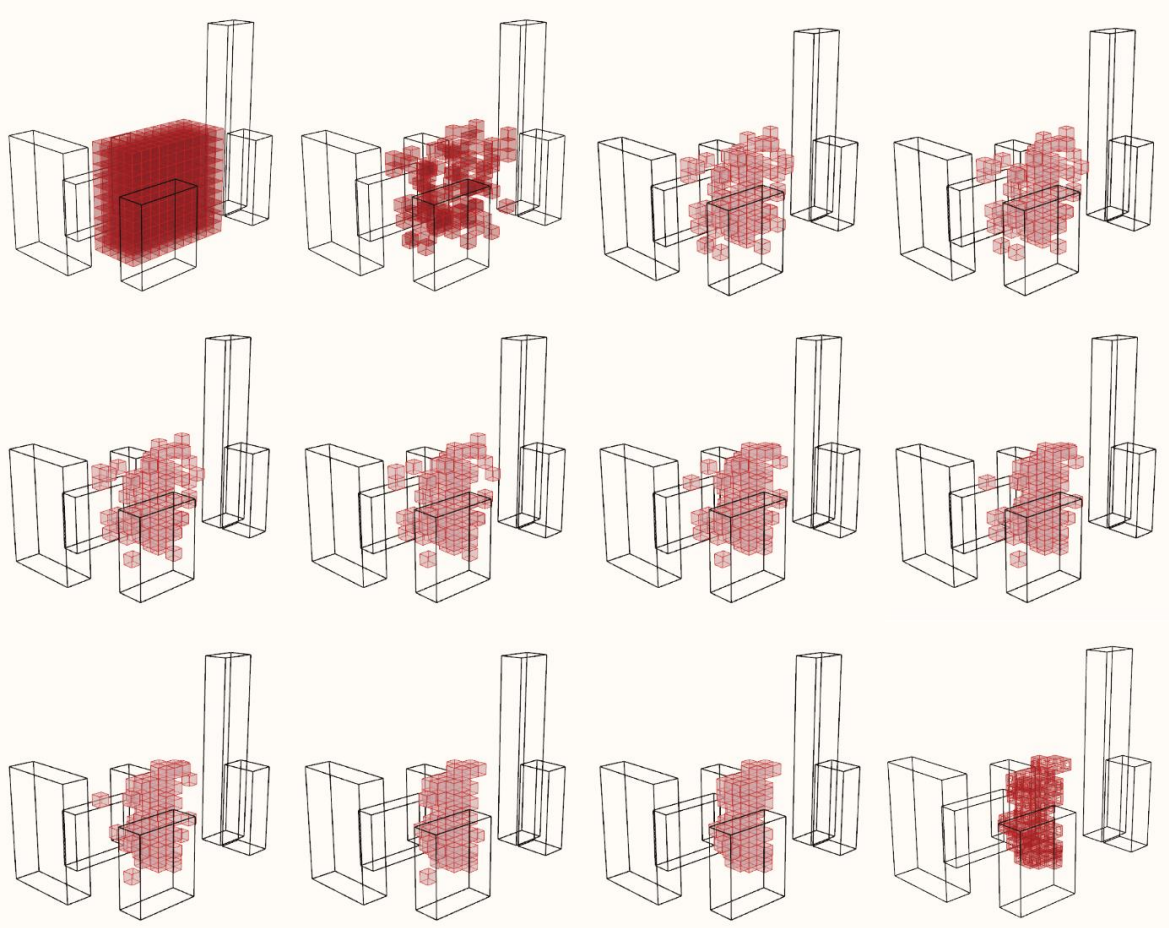


Figure 338 : Les différentes étapes de génération d'une solution

L'ensemble des étapes qui constituent l'élaboration du modèle sont identifiées directement sur le modèle Grasshopper® illustré en Figure 339. De nombreuses étapes peuvent être issues d'ancien modèle. Seules les étapes 2, 3, 4 ont dû être développées pour ce problème spécifique.

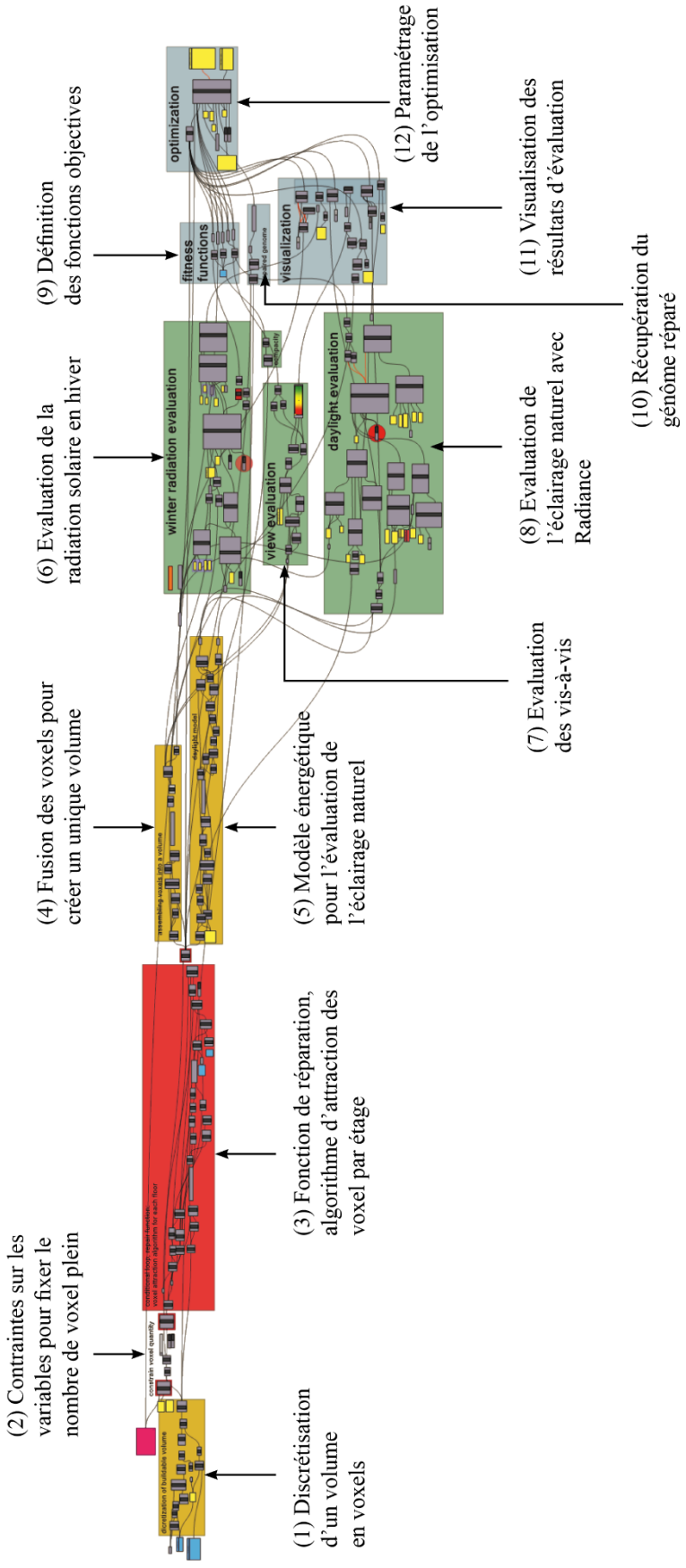


Figure 339 : Les différentes étapes de modélisation du problème

Résultats

Une fois l'exploration terminée, nous avons pu extraire les meilleures solutions pour les trois critères de confort. 59 solutions au total se trouvent sur le Front de Pareto. Elles sont surlignées en rouge dans le graphique présenté en Figure 340.

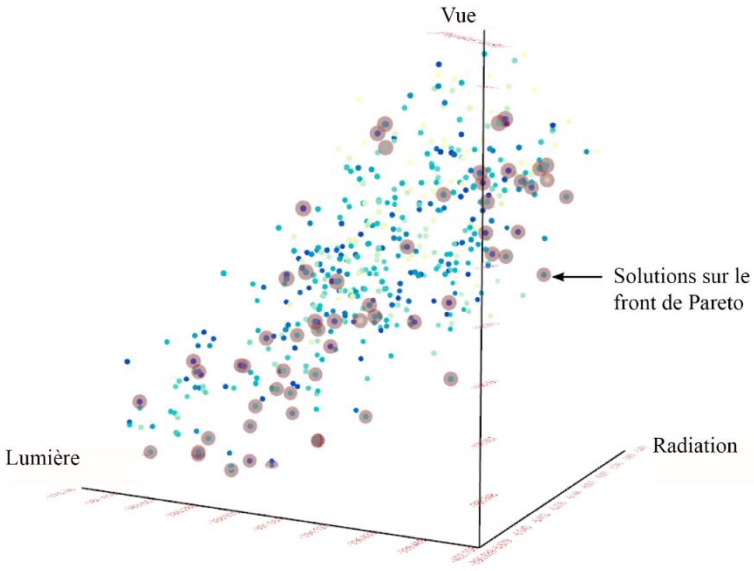


Figure 340 : Ensemble des solutions générées et front de Pareto

En dehors de la qualité de vue, les trois autres critères ont eu tendance à converger vers un minimum au moins pour les 7 premières générations.

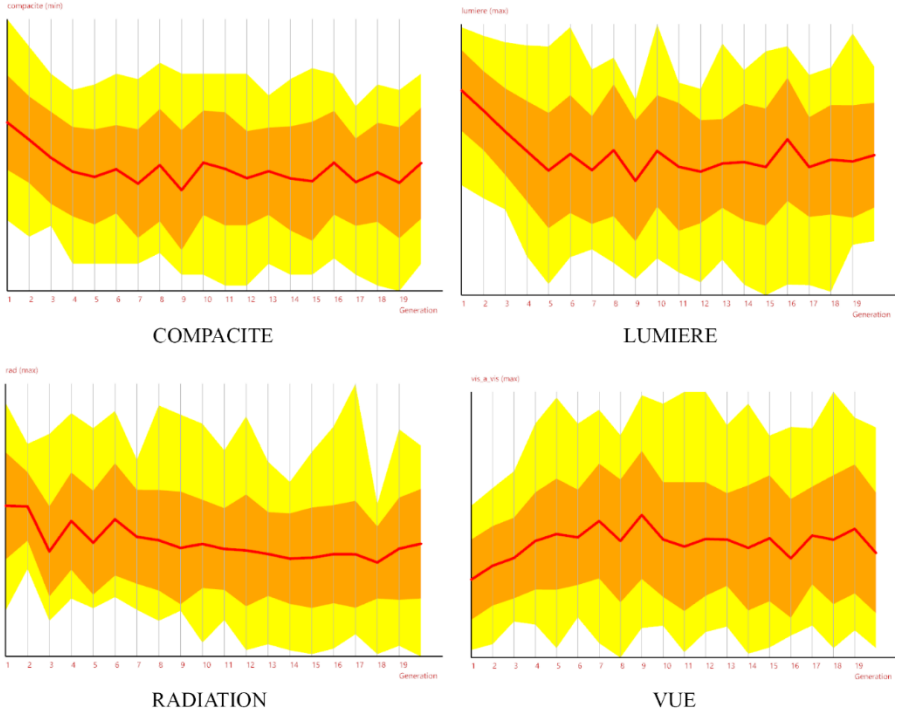


Figure 341 : Evolution des valeurs fonctions objectifs au cours des générations

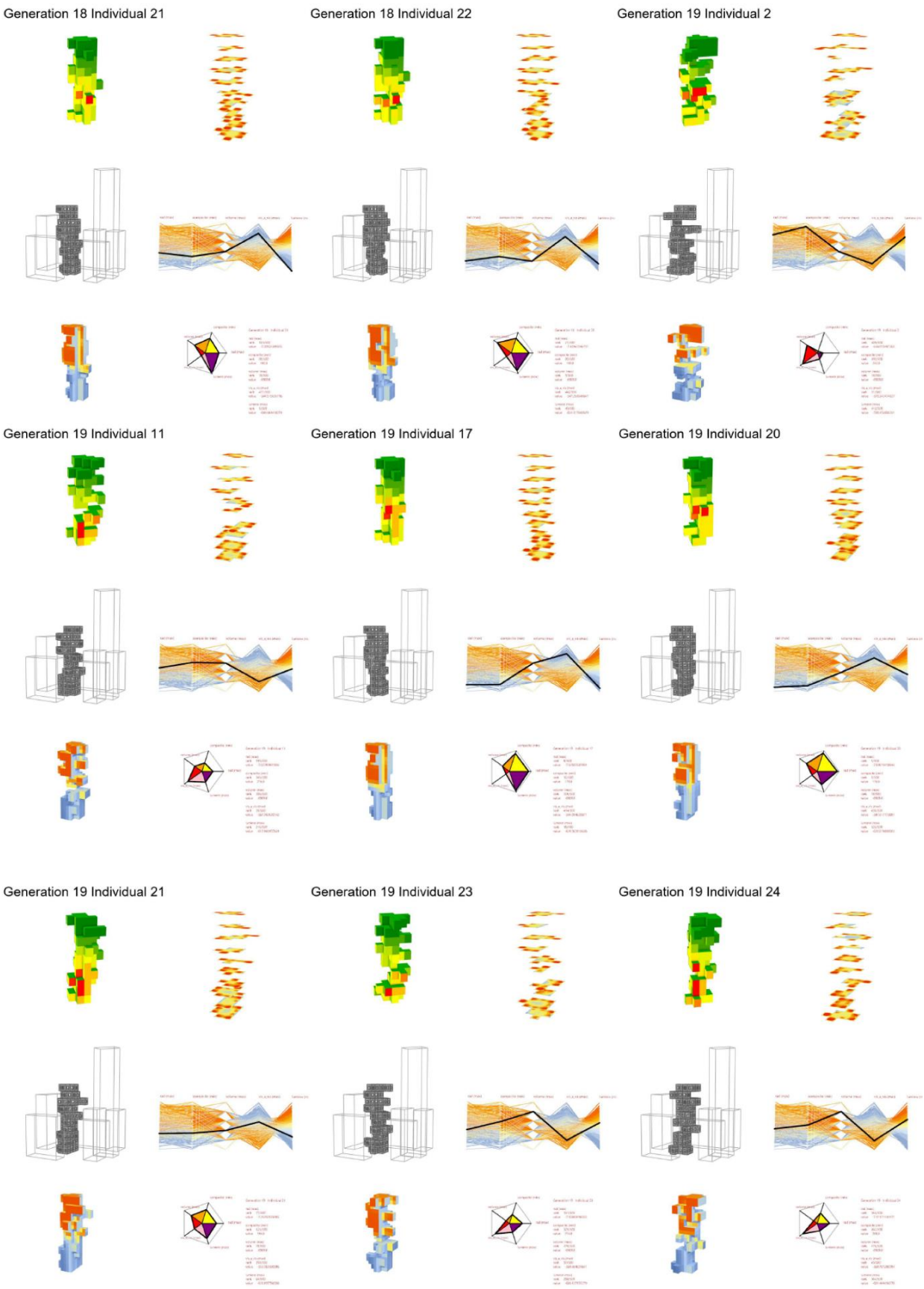


Figure 342 : Solutions extraites du front de Pareto

9 des 59 solutions du front de Pareto sont présentées en Figure 342. Il s'agit d'un extrait du catalogue des meilleures solutions générées à l'aide du composant « Ur_Poster_Solution_Selection ».

Finalement, la fonction de réparation s'est avérée très efficace puisque aucune solution générée ou presque ne contient d'aberration. Cependant, notre fonction de réparation a tendance à rassembler tous les voxels au centre de l'espace constructible. Ainsi, les solutions générées ont peut-être une morphologie trop similaire pour pouvoir en tirer des conclusions intéressantes. Une fois encore, trouver le bon algorithme de réparation peut être un défi.

Agents mobiles pour la morphologie d'un ensemble de bâtiment

Description du problème

Pour cette seconde expérimentation sur la morphologie du bâti, nous avons choisi d'interroger la forme et la disposition de plusieurs bâtiments sur une même parcelle. Il s'agit, comme pour l'expérimentation précédente du contexte urbain très dense de Courbevoie. Ce problème a été traités plusieurs fois dans la littérature (Showkatbakhsh & Kaviani, 2021; Y. K. Yi & Kim, 2015).

Trois critères, illustrés en Figure 343, sont pris en compte dans l'évaluation : l'éclairement naturel des espaces intérieurs, la radiation solaire en été sur les façades et l'analyse des vis-à-vis des bâtiments entre eux et avec les bâtiments du contexte urbain.

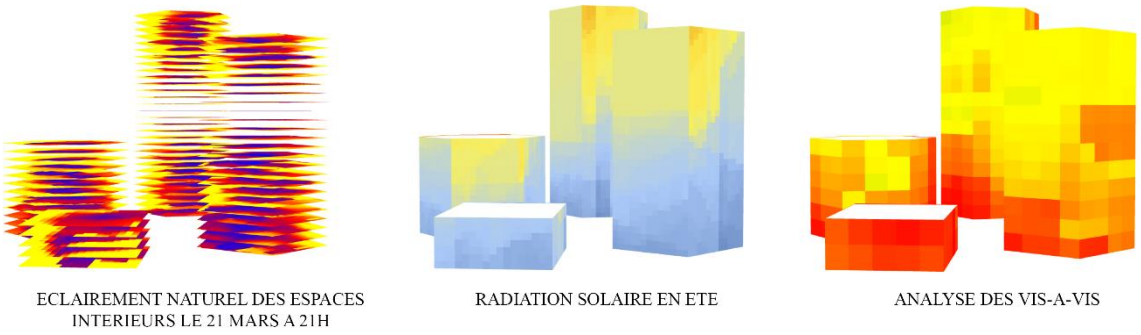


Figure 343 : Les trois critères d'optimisation

Description du modèle

Ce problème est relativement semblable à celui-ci réalisé dans la première partie de ce chapitre pour des logements à Aubervilliers (p.406). Cependant, pour ce problème, les empreintes peuvent prendre des formes et hauteurs différentes. En effet, plusieurs empreintes de bâtiments à la forme polygonale (et dont le nombre de côté est variable) sont placées dans le

périmètre constructibles. Un modèle à base d'agents permet d'éviter les collisions entre les bâtiments et avec la limite constructive. Une seconde fonction de réparation, qui est une simple boucle conditionnelle permet de contraindre le volume à bâtir (voir Figure 344) et augmentant ou diminuant le nombre d'étages de l'ensemble des bâtiments. Cette méthode permet de rendre le nombre de bâtiment sur la parcelle variable et de conserver une certaine homogénéité dans la répartition des hauteurs (plutôt que d'utiliser la méthode où la hauteur du dernier bâtiment est la résultante des hauteurs des premiers et du volume total à bâtir).

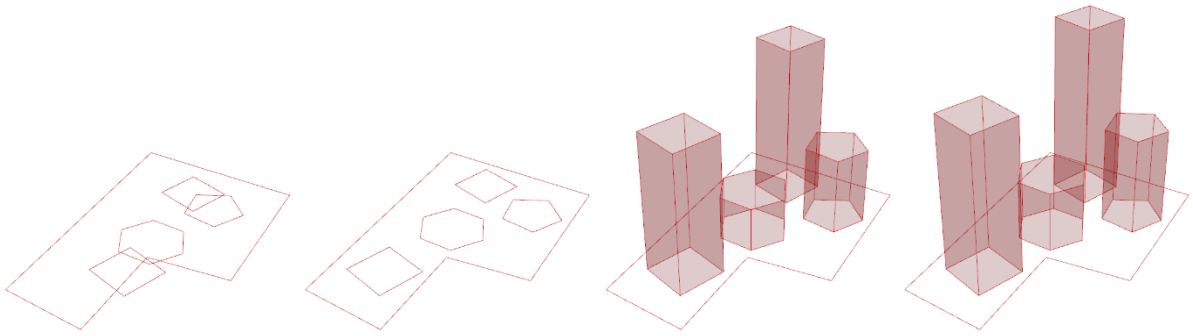


Figure 344: Les différentes étapes de modélisation d'une solution

L'ensemble des étapes nécessaires à l'élaboration du modèle de cette expérimentation sont expliquées en Figure 345. Si le script semble complexe, la plupart de ces étapes sont en réalité communes à beaucoup d'autres explorations (notamment les évaluations, ou la répulsion des empreintes). Finalement, seules les étapes 1 et 3 sont propres à ce problème et ont été spécifiquement programmées pour ce dernier.

Résultats

Au total 20 générations de 50 individus chacune ont été calculées. Le calcul aura duré un peu plus de 28h. Bien que l'algorithme semble avoir eu du mal à converger au vue des graphiques en Figure 346, au total 267 solutions sur le front de Pareto ont été trouvées. Elles sont représentées en rouge sur le graphique en nuage de point. Un extrait de ce front de Pareto est illustré en Figure 347, il a été généré à l'aide du composant « Ur_Poster_Solution_Selection ». Finalement, il apparait que le critère de vue et celui de la radiation solaire soient en concurrence. Les solutions qui permettent de trouver un équilibre entre les deux critères ont des bâtiments de hauteurs moyennes (environ 30 étages). Concernant la lumière des espaces intérieurs, les meilleurs solutions (avec le plus d'espaces au-dessus de 500 lux et le moins d'espaces en dessous de 100 lux) ont des bâtiments aux formes pentagonale ou hexagonale avec un rayon situé entre 15 et 16m.

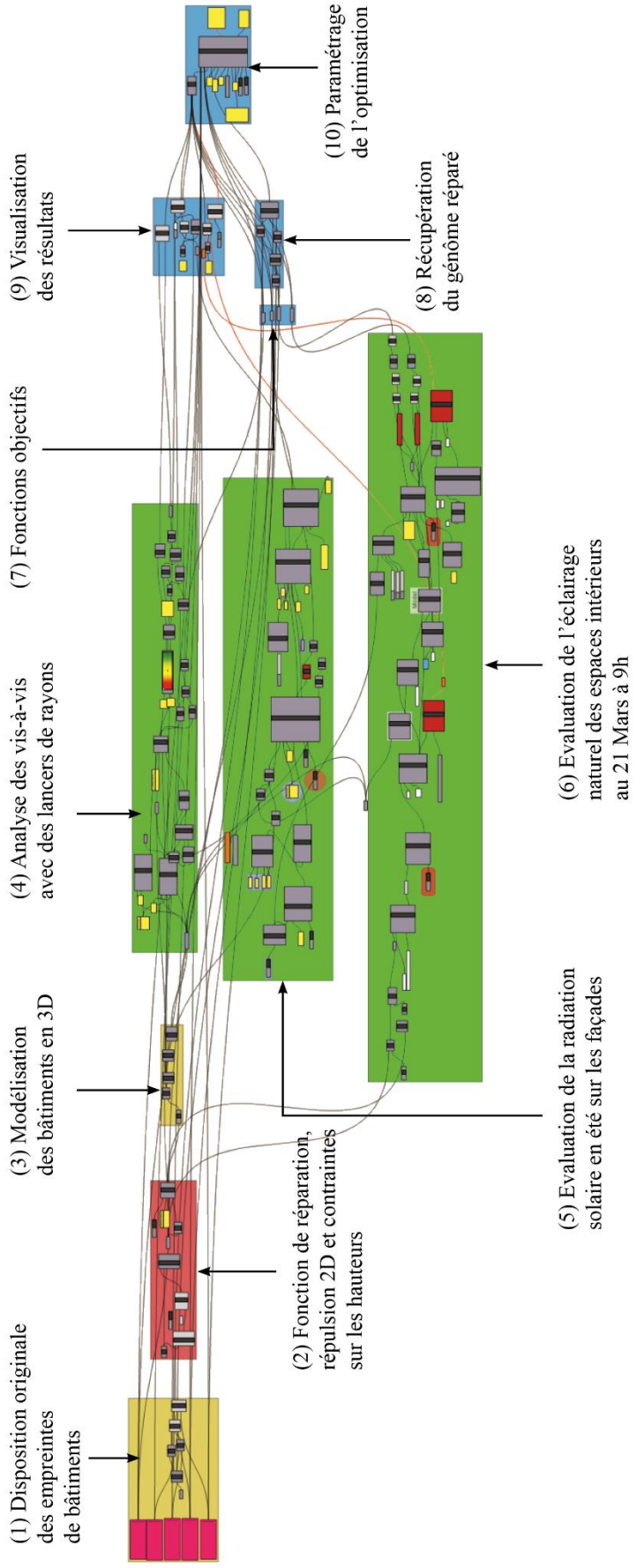
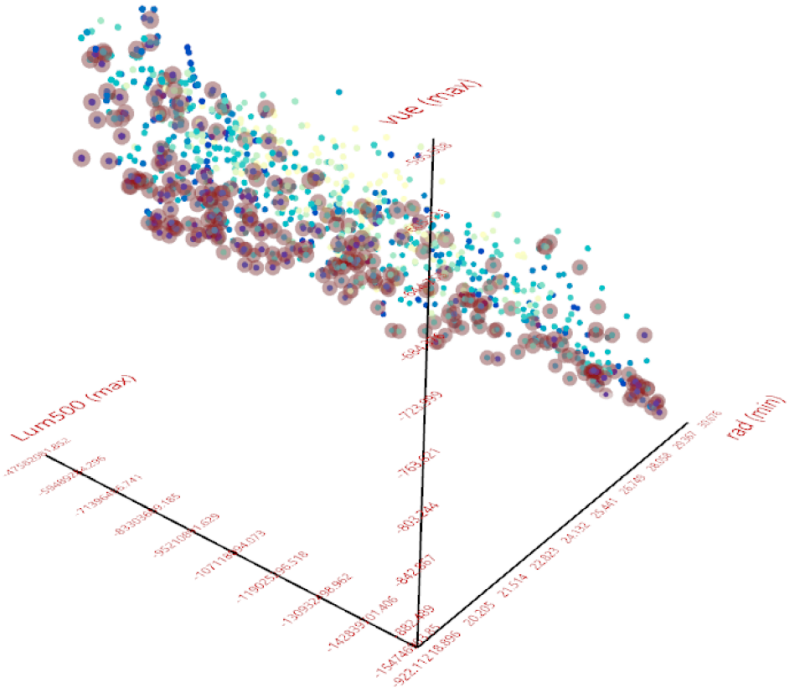


Figure 345: Les différentes étapes de modélisation du problème



Ensemble des solutins générés pendant l'exploration

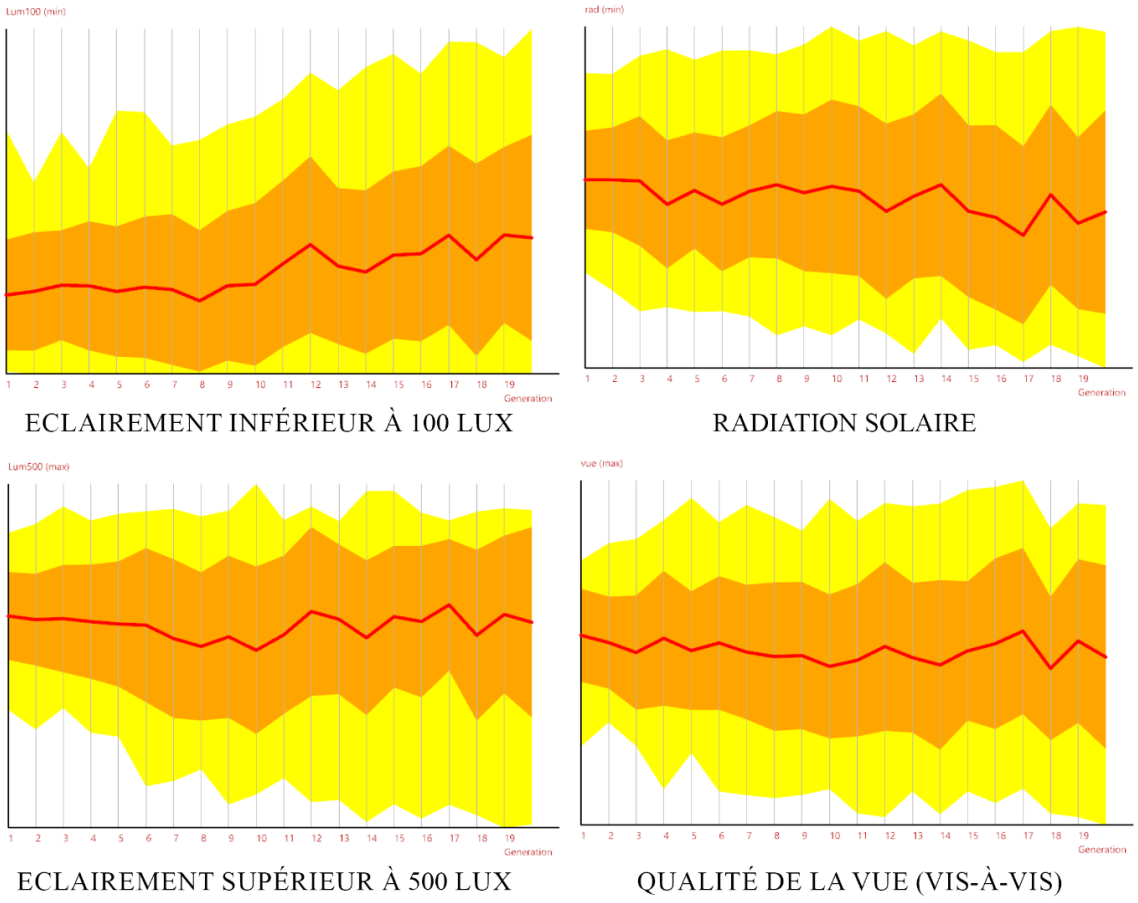


Figure 346: Résultats de l'exploration en termes de performance

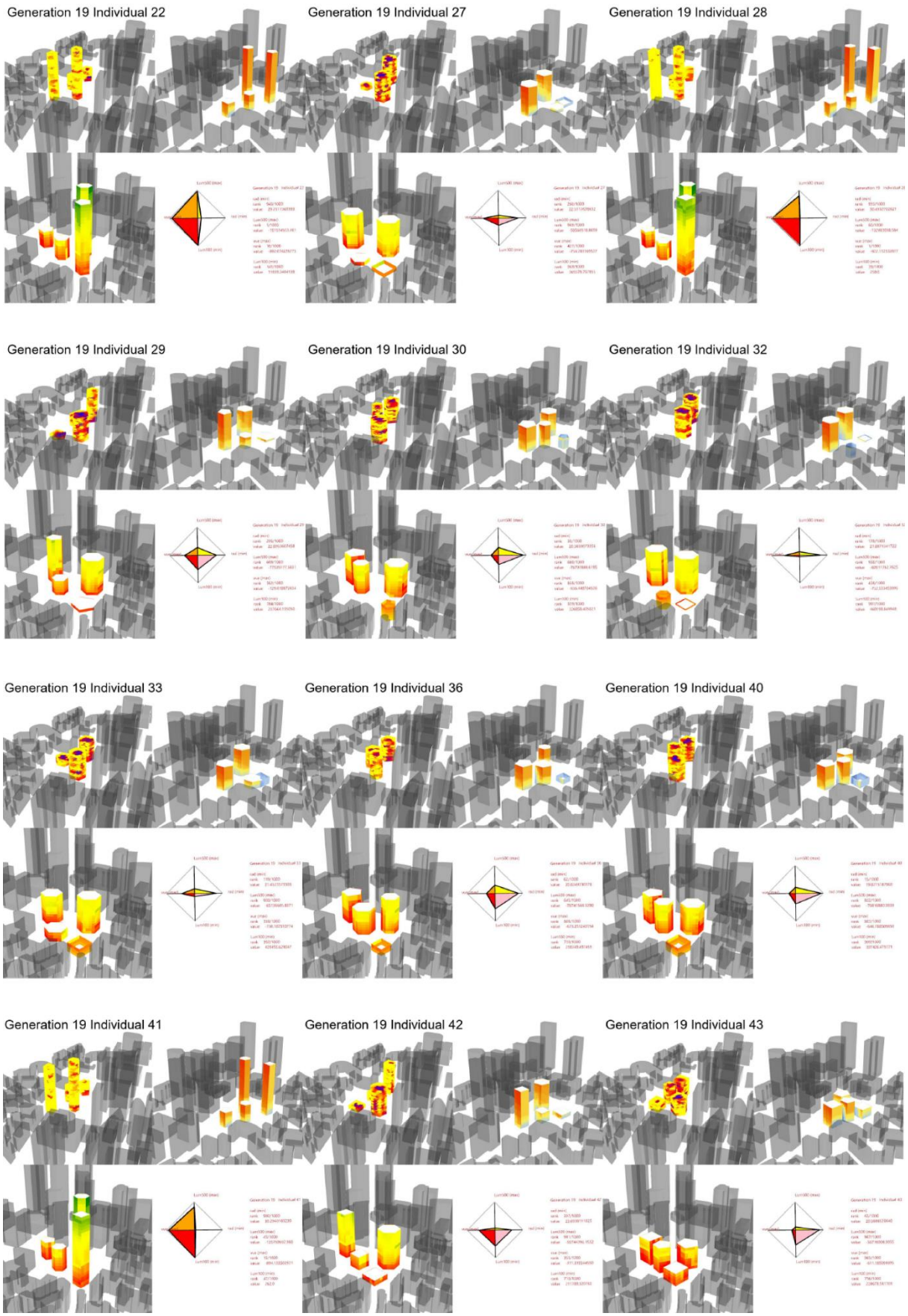


Figure 347: Solutions extraites du front de Pareto

4.3.2. Applications à l'échelle urbaine

Automate cellulaire augmenté pour interroger les typologies urbaines

Description du problème

La troisième application est une proposition de méthode de modélisation pour une étude pour la performance des typologies urbaines réalisée à l'aide d'un plan hippodamien comme il en existe déjà dans la littérature scientifique (António et al., 2014; Vermeulen et al., 2018; X.-Y. Wang et al., 2018). Pour cela, un morceau de ville constitué de 100 îlots carrés est généré à l'aide d'un ACA. Trois types de programmes peuvent être alloués à chaque îlots : un parc, des logements ou des bureaux. Pour les bâtiments, trois types de morphologie d'îlots sont possibles : l'îlot fermé, l'îlot ouvert (avec deux orientations possibles pour les bâtiments) , ou la tour de grande hauteur (5 dispositions possibles sur la parcelle).

Plusieurs contraintes doivent être respectées. La quantité d'îlots de logements et de bureaux doit être la même pour chaque solution générée, ainsi que la surface totale à bâtir de chaque programme. Les parcs ne peuvent pas être mitoyens. Les tours doivent faire au minimum 15 étages. Les ilots ouverts doivent faire au maximum 13 étages, et les îlots fermés doivent faire au maximum 7 étages.

Trois critères d'optimisation sont pris en compte : la radiation solaire en façade des bureaux doit être minimisée, la radiation solaire en hiver des logements doit être maximisée et la radiation solaire annuelle des parcs doit être maximisée. Les résultats graphiques des trois analyses sont illustrés en Figure 348.

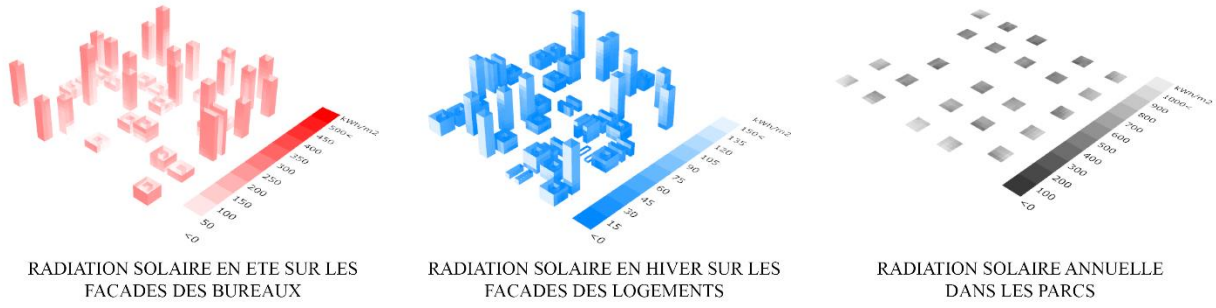


Figure 348 : Les trois critères d'optimisation

Description du modèle

Trois fonctions de réparation permettent de gérer les contraintes du problème. Le composant « Ur_Genepool_Constraint_Variable » est utilisé pour fixer la quantité d'îlots de logements et de bureaux. Un ACA est mis en place plus traiter le reste des contraintes à

l'exception des surfaces totales à bâtir qui sont traitées par une boucle conditionnelle. En Figure 349 sont illustrées les différentes itérations menant à la réparation d'une solution. L'ensemble des étapes nécessaires à la modélisation du processus d'exploration sont présentées en Figure 351. Les étapes 1, 2, 3 et 6 sont les seules à avoir été programmées spécifiquement pour ce problème.

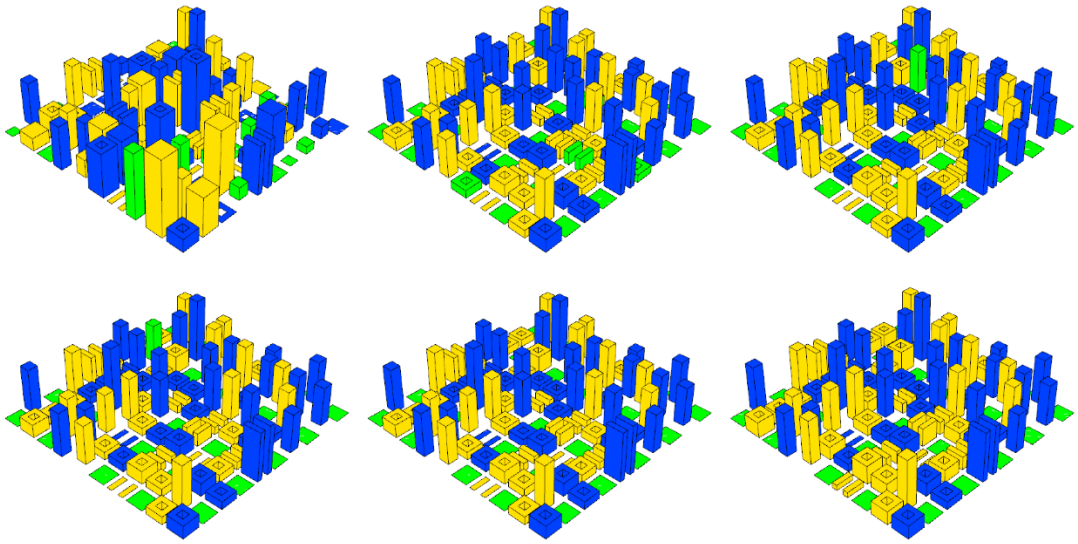


Figure 349 : Exemples de réparations nécessitant 4 itérations + la réparation des hauteurs

Résultats

Au total, 20 générations de 50 individus chacune ont été calculées, pour un temps de calcul final d'environ 11h durant lesquelles l'algorithme semblait converger lentement si on observe les graphiques des fonctions objectifs de la Figure 350. En Figure 352 est présenté un extrait du front de Pareto et en Figure 353, un extrait des plus mauvaises solutions. S'il est difficile de définir des règles pour la disposition des typologies, l'étude permet de voir que cette dernière peut avoir un fort impact sur les apports solaires.

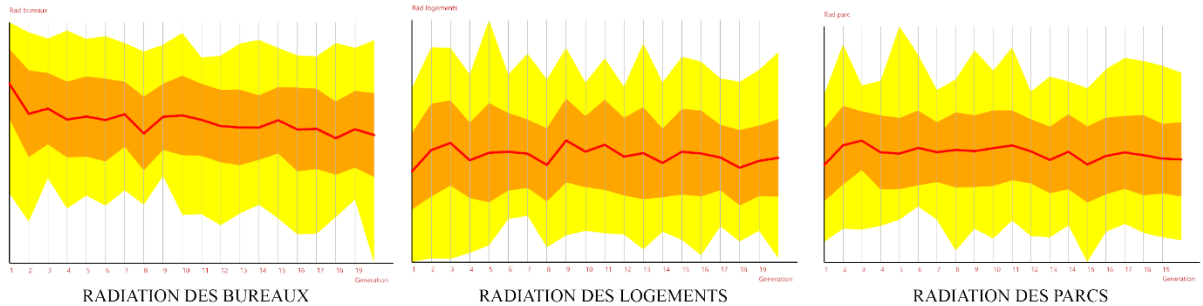


Figure 350 : Evolution des valeurs fonctions objectifs au cours des générations

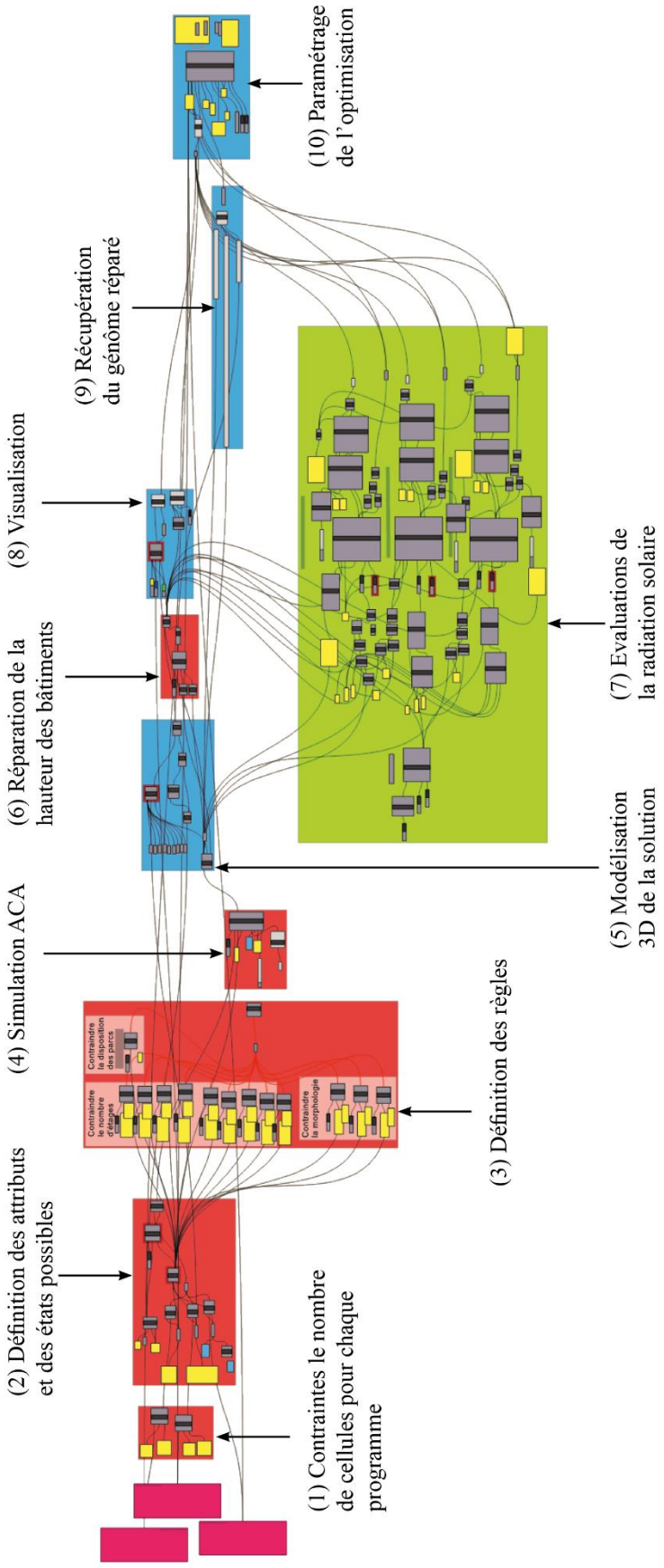


Figure 351 : Les différentes étapes de modélisation du problème

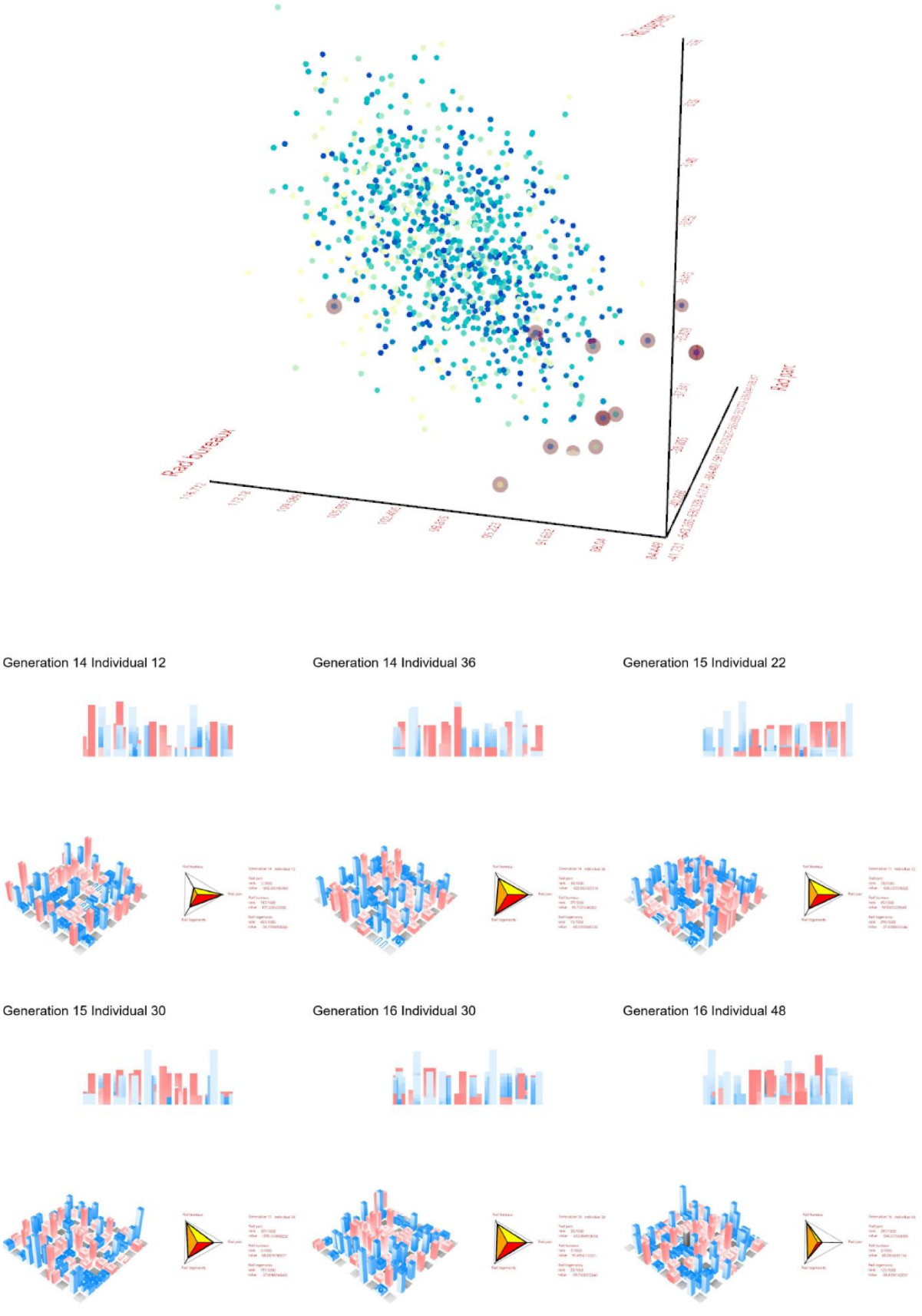


Figure 352 : Six solutions parmi les solutions du Front de Pareto

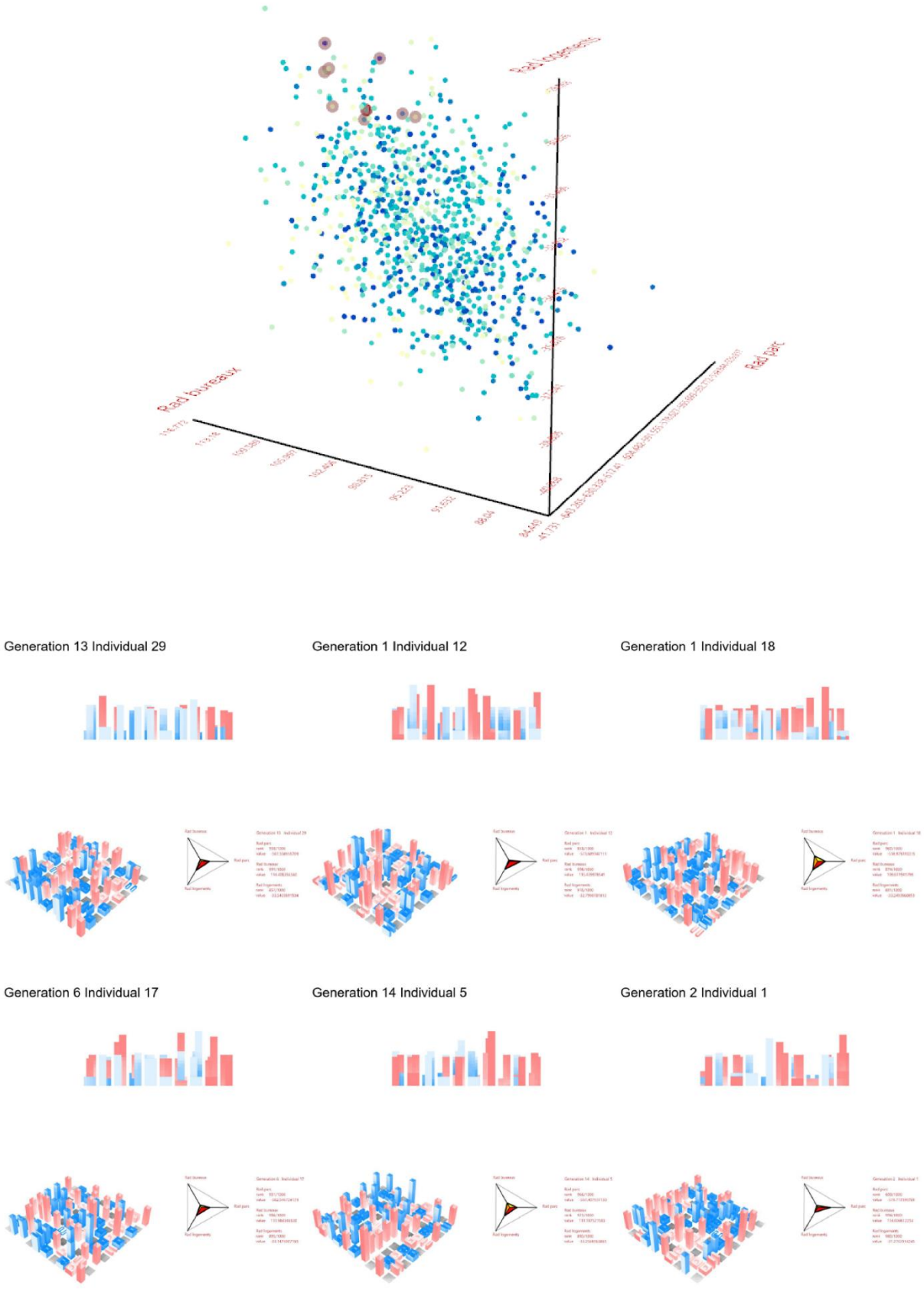


Figure 353 : Six solutions parmi les solutions du dernier rang

Agents mobiles pour l'optimisation d'un réseau viaire

Description du problème

Pour ce second problème de conception urbaine computationnelle, nous avons cherché à optimiser la conception d'un réseau viaire d'un cas d'étude imaginé de petit quartier de type petit ZAC. Ce problème peut être comparé à des cas issus de la littérature scientifique pour la conception de petit quartier (Duering et al., 2020; Nagy et al., 2018).

Pour ce problème, le terrain entouré de bâtiments construits peut être découpé en une quantité variable d'îlots. Pour modéliser ce découpage, les paramètres utilisés sont des points, représentant les nœuds des rues. Leur nombre est variable, mais ils doivent respecter une certaine distance minimale entre eux. Celle-ci est l'unique contrainte qui doit être intégrée au processus d'optimisation.

L'objectif de l'exploration est d'optimiser la surface de logements résultant du découpage parcellaire. Pour cela, quatre fonctions objectifs sont définies : (1) maximiser la surface de logements, (2) minimiser la surface de logements étant éclairée à moins de 100 lux, (3) maximiser la surface de logements éclairés à 500 lux, (4) maximiser la surface de logements éclairés à 300 lux. Une capture d'écran de l'analyse de l'éclairement naturel au 21 décembre à 10h réalisée au RDC pour une solution générée lors de l'exploration est illustrée en Figure 354.

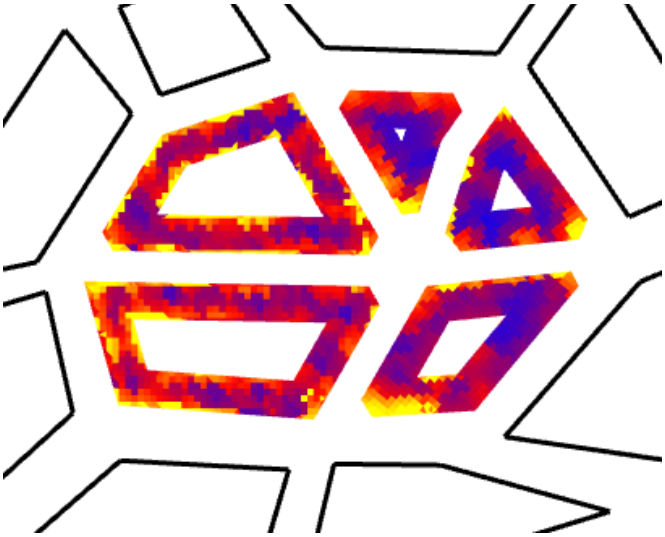


Figure 354 : Evaluation de l'éclairement naturel au RDC

Description du modèle

Pour générer une solution, des points au nombre variable représentant les nœuds des rues sont mis en mouvement dans une fonction de réparation à base d'agents où ces derniers

sont des cercles permettant d'assurer une distance minimale entre les nœuds. Un algorithme de triangulation Delaunay est ensuite utilisé pour discrétiser le terrain. Les triangles sont associés deux à deux pour définir le contour d'îlots urbains fermés. Lorsque les îlots sont petits, l'intégralité de leur surface devient l'empreinte d'un bâtiment. Ces différentes étapes de modélisation d'une solution sont illustrées en Figure 355. Les étapes de modélisation du processus d'optimisation sont décrites en Figure 356. Seules les étapes 1 et 3 sont spécifiques à ce problème.

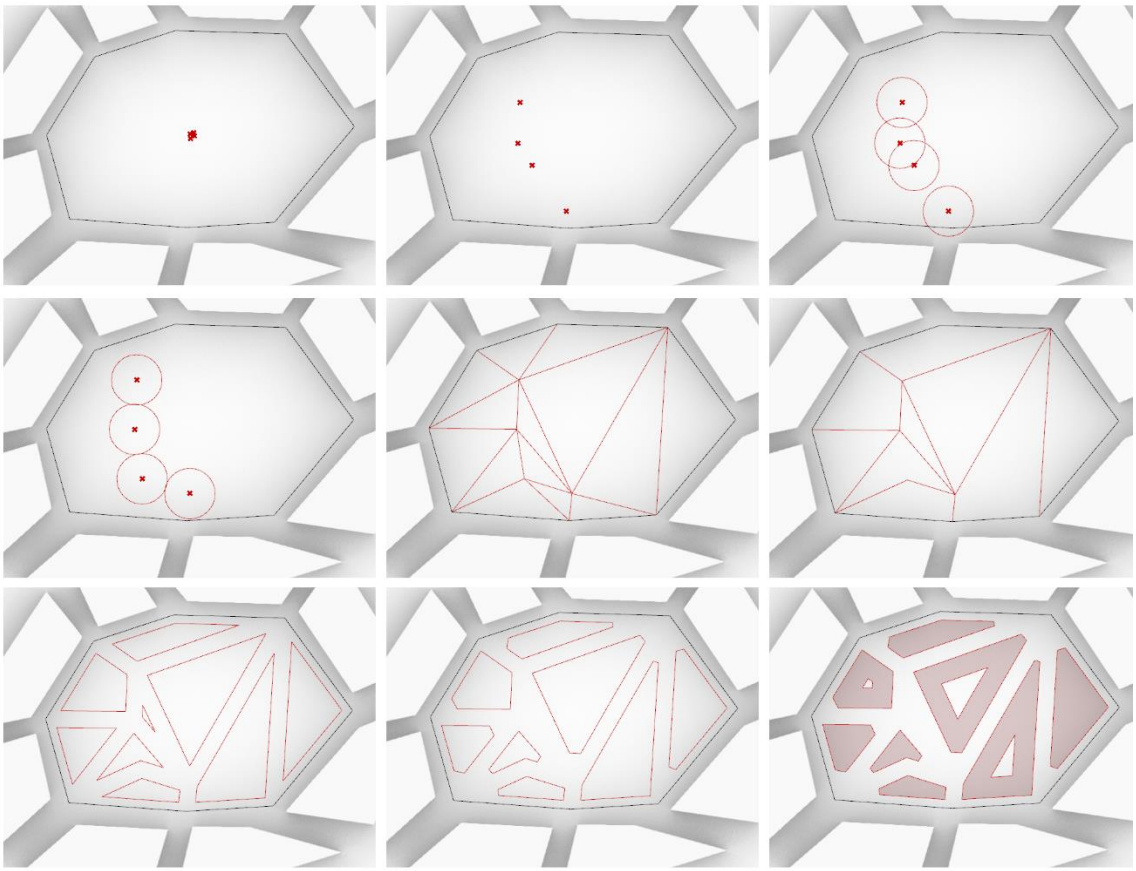


Figure 355 : Etapes de la modélisation d'une solution

Résultats

Finalement, 20 générations de 30 individus chacune ont été calculées. L'exploration a duré environ 13h. Le front de Pareto représenté en Figure 357 compte 39 solutions, dont 12 sont visibles en Figure 358. D'après les graphiques des fonctions objectifs, il semblerait que les scores des critères aient peu évolué au cours du processus d'optimisation. Les solutions avec le plus de surfaces éclairées et le moins de surface perdue (sans lumière) sont celles composées de deux très grands îlots et de 5 bâtiments isolés. La plus mauvaise configuration en termes de lumière est composée de 3 grands îlots et 2 bâtiments isolés.

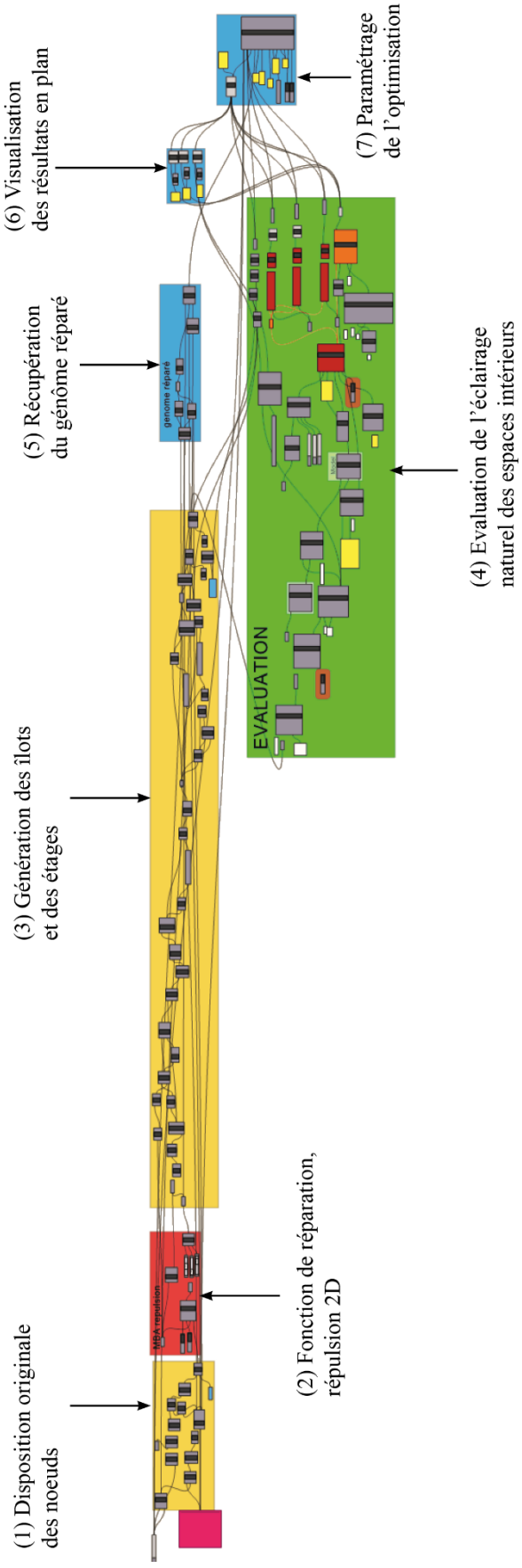
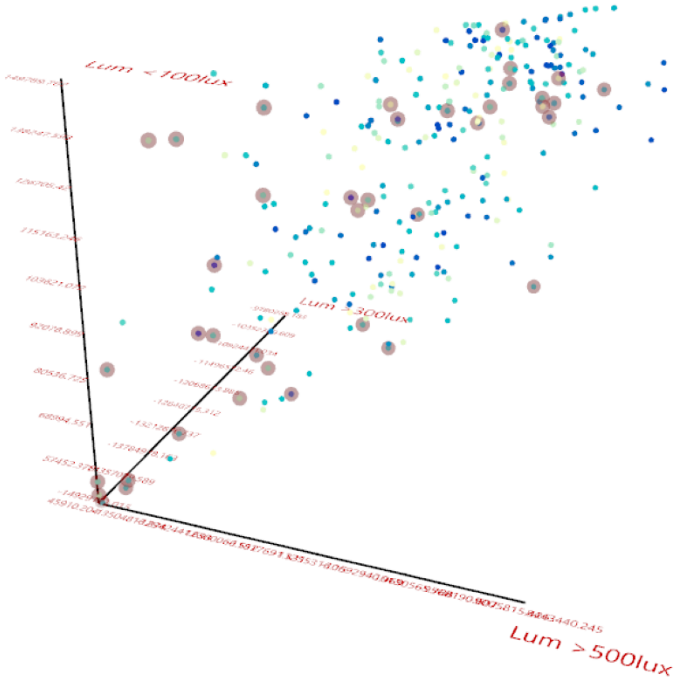
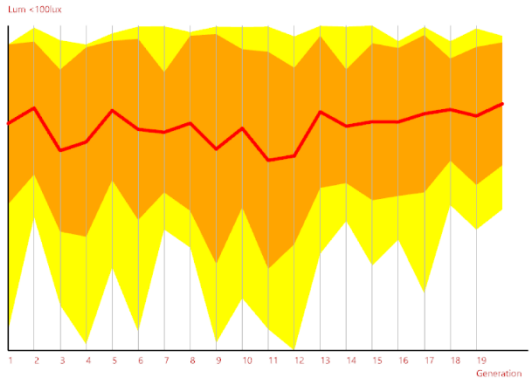


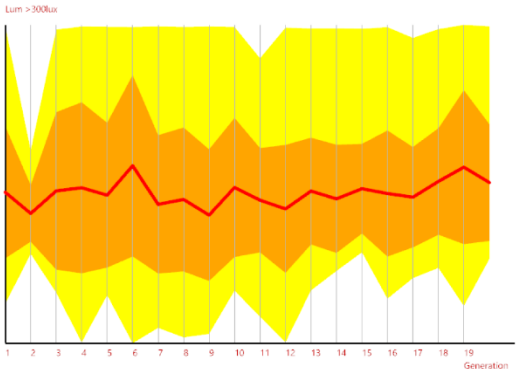
Figure 356 : Les différentes étapes de modélisation du processus d'exploration



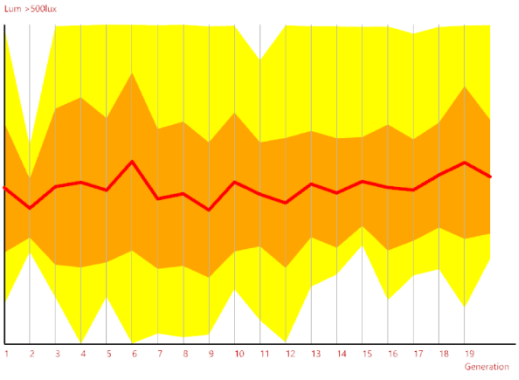
Ensemble des solutins générés pendant l'exploration



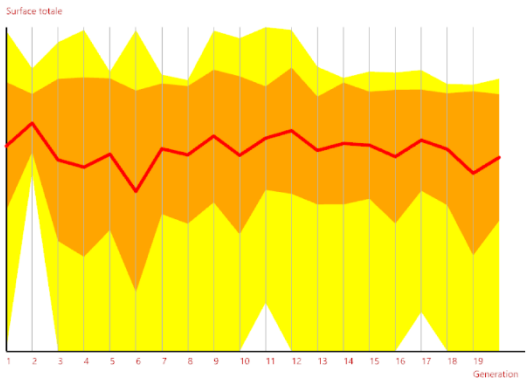
Eclairement inférieur à 100 lux



Eclairement supérieur à 300 lux



Eclairement supérieur à 500 lux



Surface constructible

Figure 357 : Performance de l'exploration

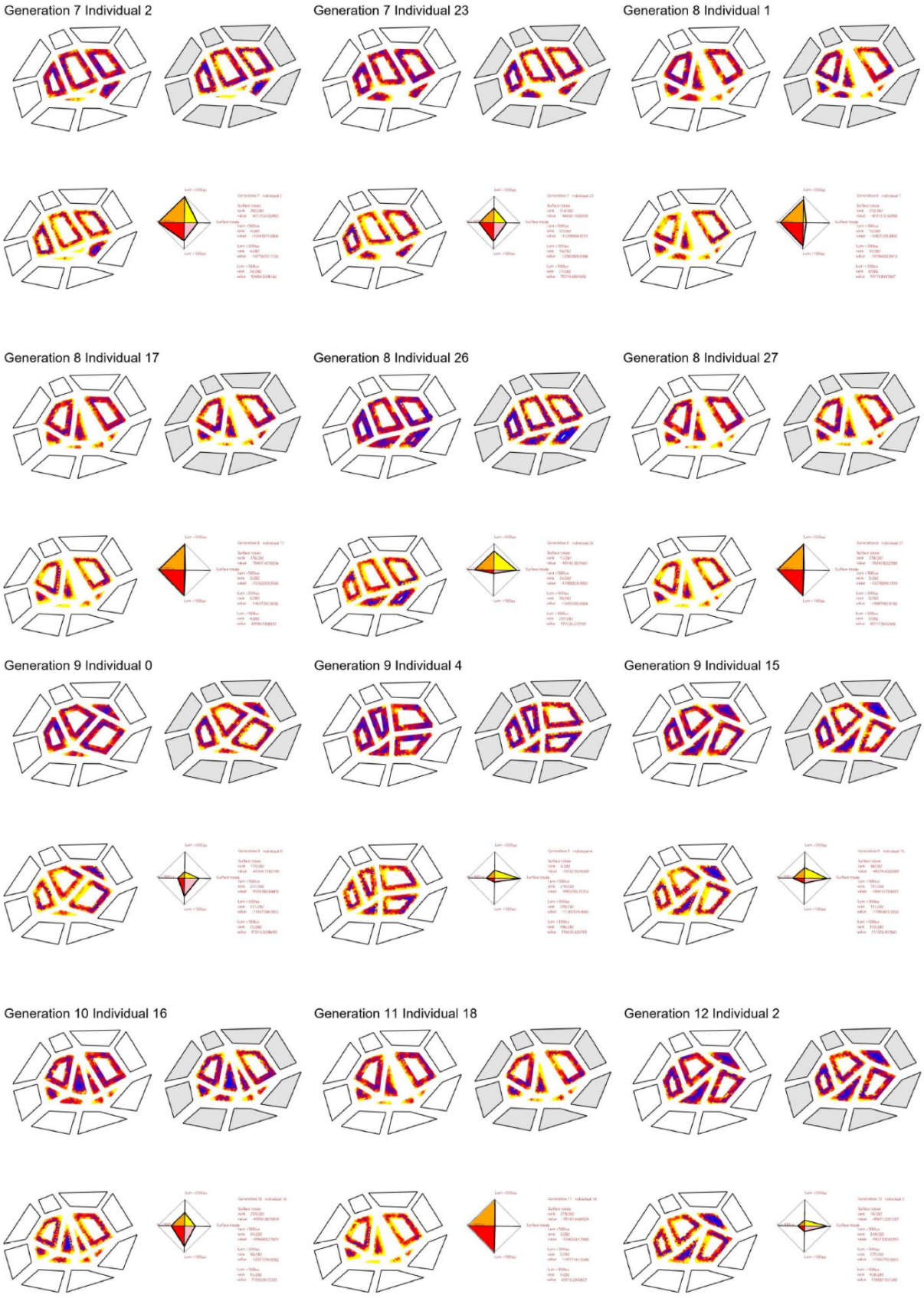


Figure 358 : Solutions extraites du front de Pareto

4.3.3. Applications à l'agencement d'espaces intérieurs

Automates cellulaires augmentés pour la génération d'un plan d'appartement

Description du problème

Pour cette cinquième application, il s'agit d'un problème simple d'allocation spatiale sous contraintes comme on a pu en trouver lors de l'élaboration de l'état de l'art des cas d'application au chapitre 1 (Schwartz et al., 2017; Su & Yan, 2015; H. Yi, 2016). Ce cinquième modèle permet de générer des plans d'appartements de 4 pièces composés d'un salon (en jaune), d'une salle à manger (en orange), de deux chambres (en vert) et une cuisine (en rouge), une salle de bain (en bleu). L'appartement de 80 m² est un rectangle de 8x10m discrétisé en 80 pixels de $1m^2$ chacun. Le problème compte une variable par pixel qui permet d'attribuer une fonction (une pièce) à chaque pixel.

Les critères d'optimisation pris en compte lors de l'exploration sont la compacité des pièces, le nombre de pièces (pour éviter qu'une fonction soit divisée en deux espaces distincts), et le respect des règles d'attraction qui sont les suivantes : la salle de bain et la cuisine doivent être adjacentes, ainsi que la cuisine et la salle à manger, ou encore la salle à manger et le salon.

Description du modèle

Pour pouvoir gérer toutes ces contraintes, 5 fonctions de réparation successives sont utilisées. L'évolution de la réparation est illustrée avec un exemple en Figure 360. La première permet de fixer le nombre de m² de chaque espace, sachant qu'au début, 10 cellules ne sont affiliées à aucune pièce, elles le seront au fur et à mesure de la réparation. La seconde fonction est un ACA qui permet d'attirer entre elles les fonctions similaires des espaces les plus grands (tous sauf la salle de bain). La troisième fonction, elle aussi un ACA, permet de rassembler les cellules de la salle de bain. La quatrième fonction et dernière fonction, toujours des ACA permettent de répartir les cellules grises (non affiliées) en leur attribuant une fonction selon leur voisinage.

Le détail du script et des étapes nécessaires au processus d'exploration pour ce problème est expliqué en Figure 359. Les étapes 1, 2, 3, 5, 7, 9, 11, 13 et 14 sont spécifiques à ce problème finalement complexe à modéliser.

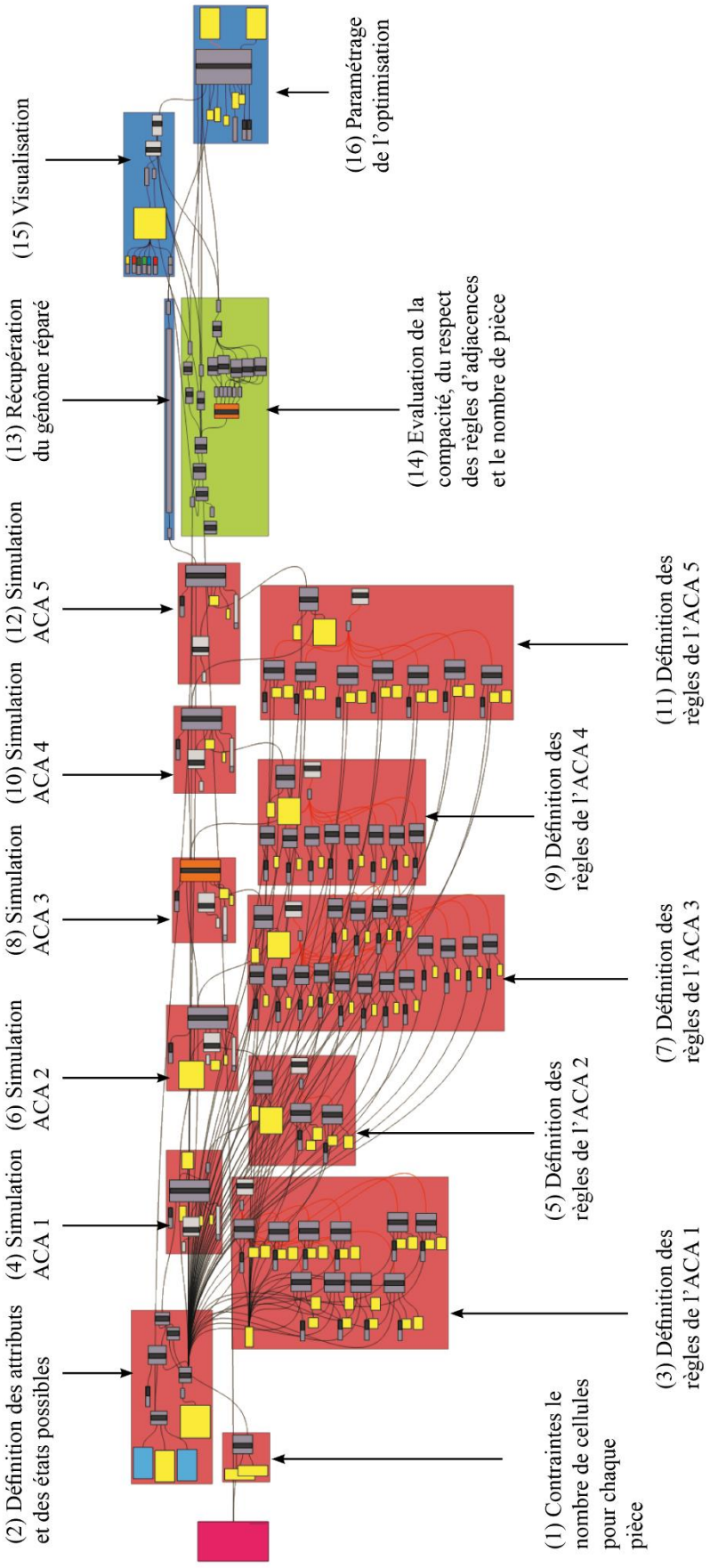


Figure 359 : Les différentes étapes de modélisation du processus d'exploration

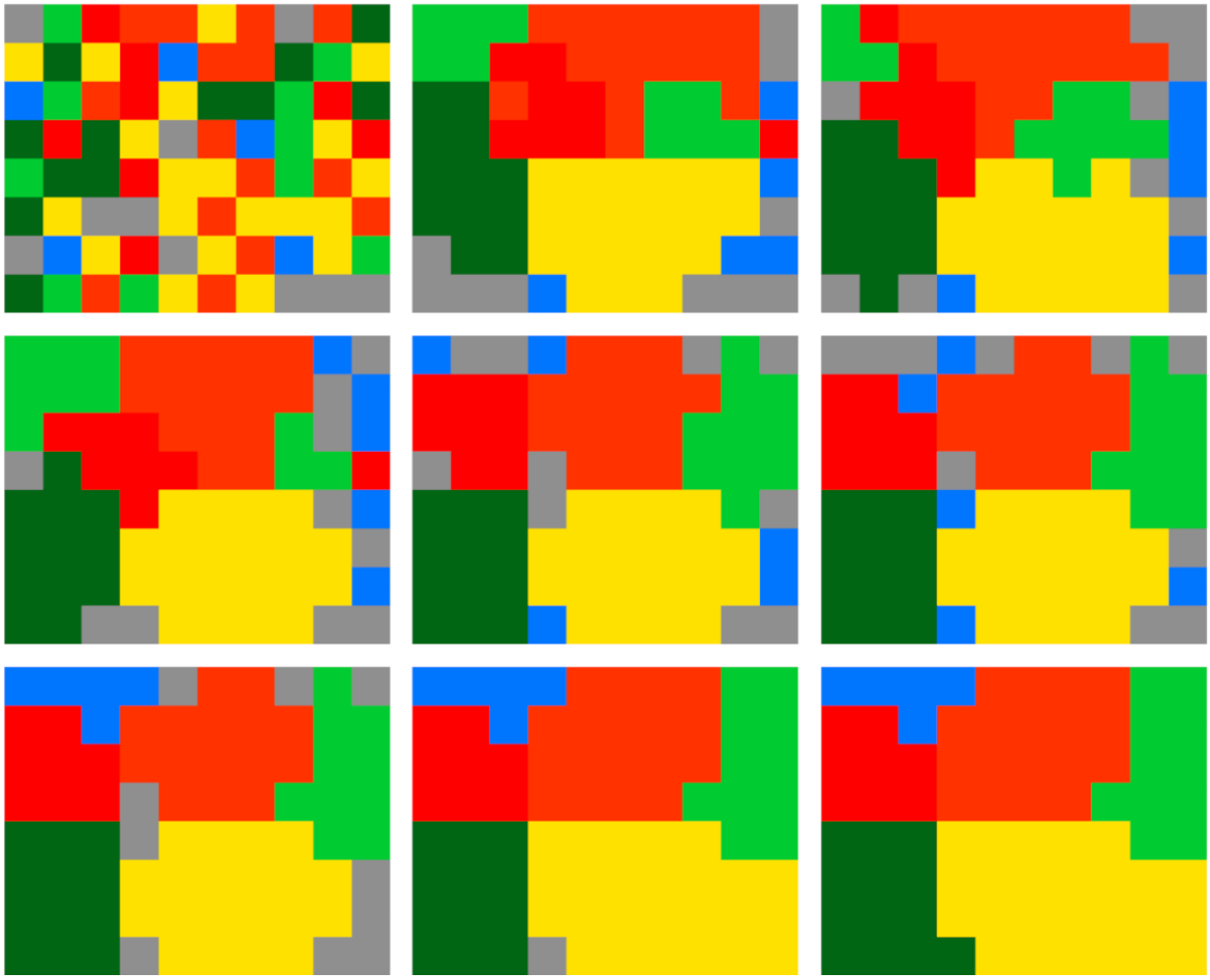


Figure 360 : Différentes itérations de l'algorithme de réparation

Résultats

Finalement, 20 générations de 50 individus chacune ont été calculées en 5h. L'algorithme semble avoir convergé facilement malgré le fort écart qu'il existe entre le génome des solutions originales et celui des solutions réparées, et les trois fonctions objectifs n'ont cessé au fil des générations de progresser vers un minimum comme le montre la Figure 361.

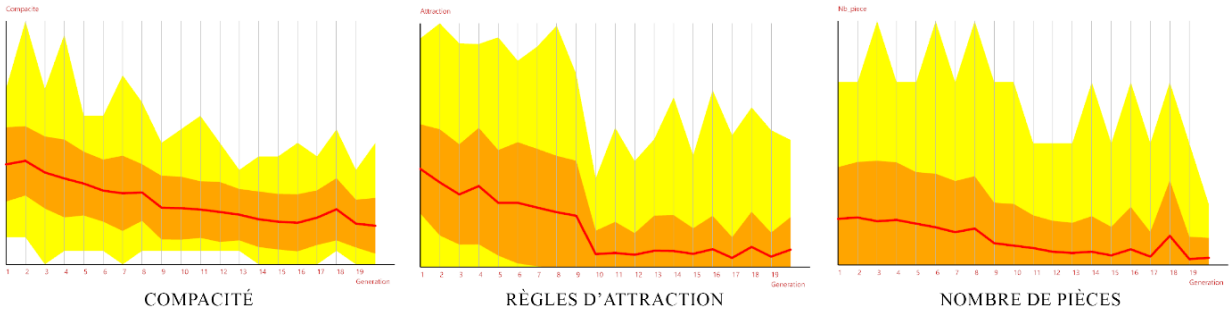


Figure 361 : Evolution des valeurs des fonctions objectifs au fil des générations

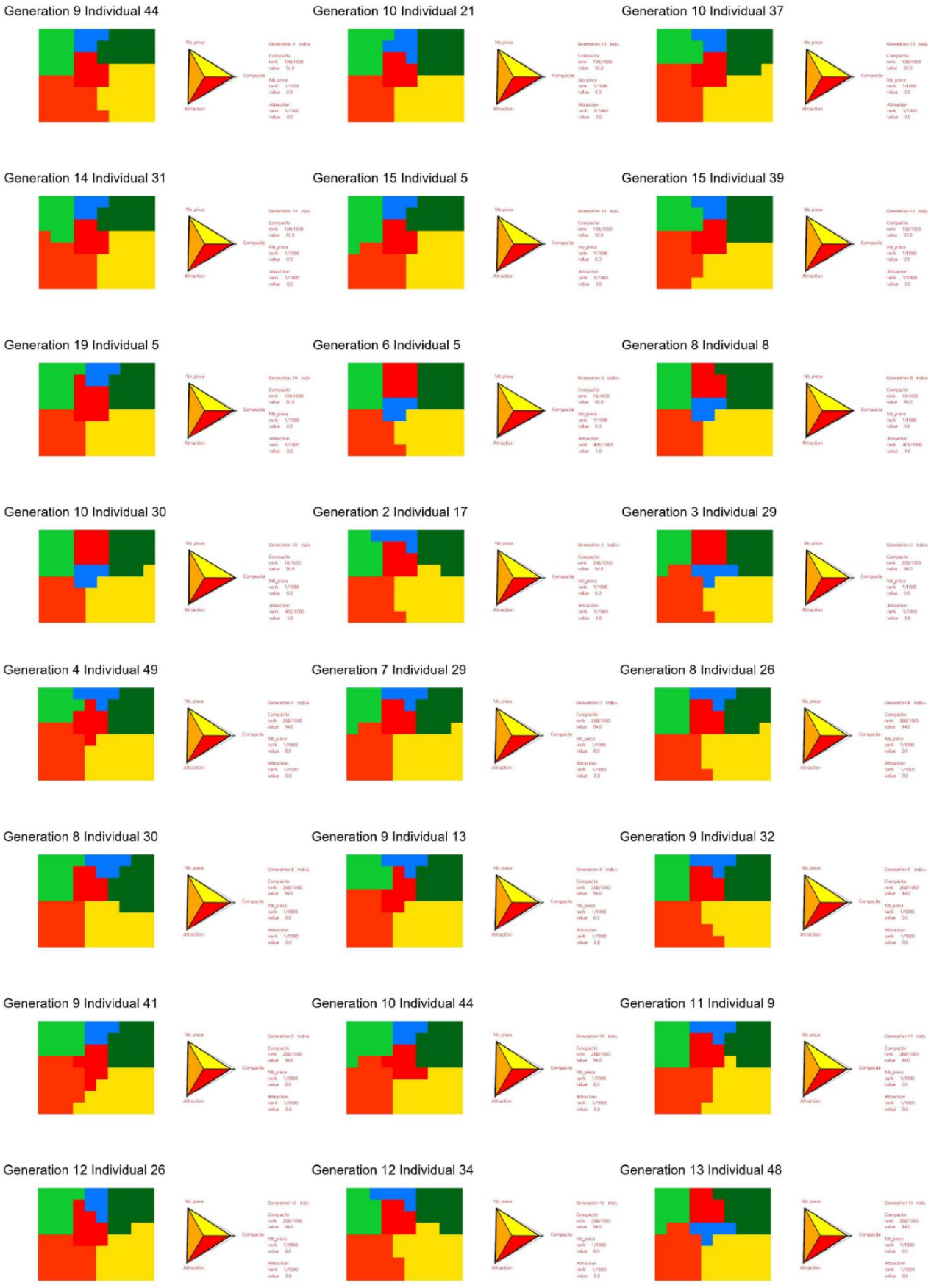


Figure 362 : Solutions parmi les meilleures générées (rang 1,2 et 3)

Si la Figure 362 présente un échantillon des meilleures solutions générées, le front de Pareto ne compte en réalité que 2 solutions présentées en Figure 363.

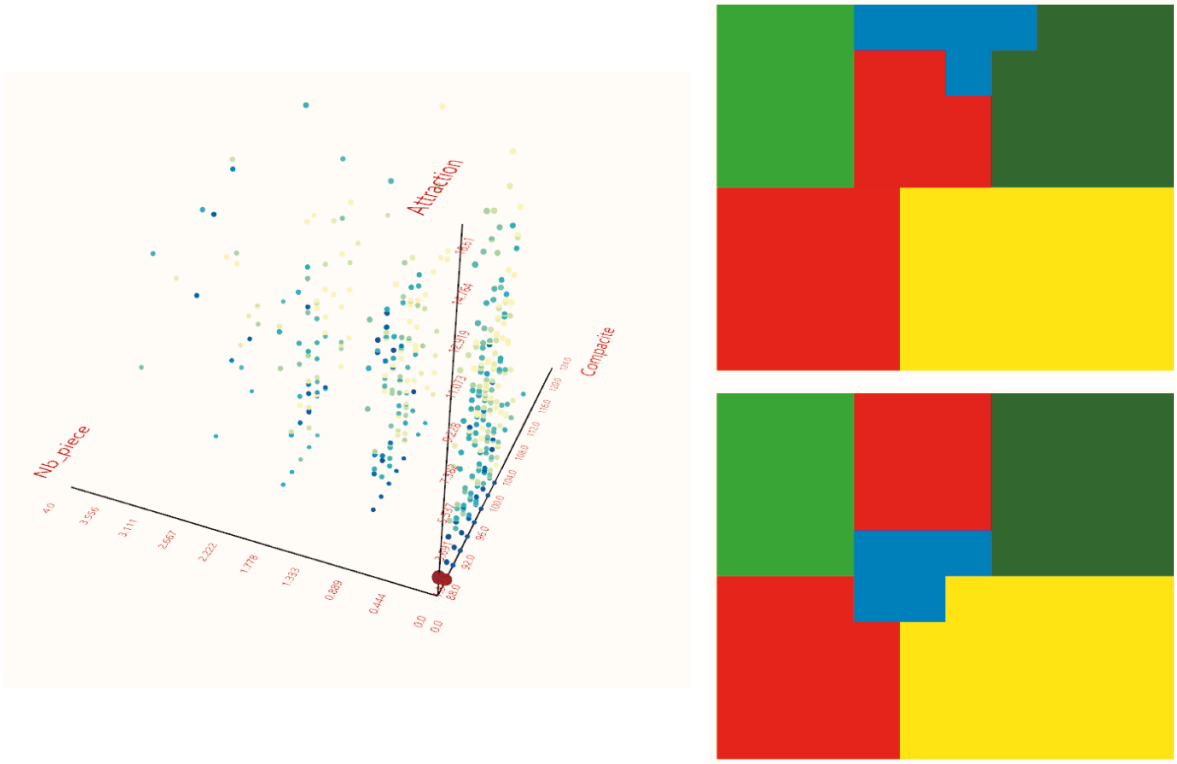


Figure 363 : Les deux meilleurs solutions générées

L'algorithme de réparation de cette application fut le plus difficile à trouver des 6 problèmes présentés dans cette partie du chapitre. Il a nécessité de nombreux tests et modification avant d'obtenir des résultats satisfaisants, mais il permet aussi d'obtenir l'un des résultats les plus probants.

Agents mobiles pour la spatialisation d'un organigramme de villa

Description du problème

Pour cette dernière application, nous avons cherché à générer des esquisses d'organigramme spatialisé pour aider à la conception d'une villa de plein pied. Ainsi, le point de départ du problème est un programme qui prend la forme d'un organigramme à bulles (souvent utilisé chez les architectes) illustré en Figure 364. Les bulles sont reliées entre elles par des flèches qui permettent de hiérarchiser les relations des espaces entre eux.

Pour cette application nous nous sommes inspirés de la méthode d'allocation spatiale basée sur la simulation particulaire (Z. Guo & Li, 2017). Nous avons présenté les principes de fonctionnement de cette méthode au chapitre 1 (p. 111).

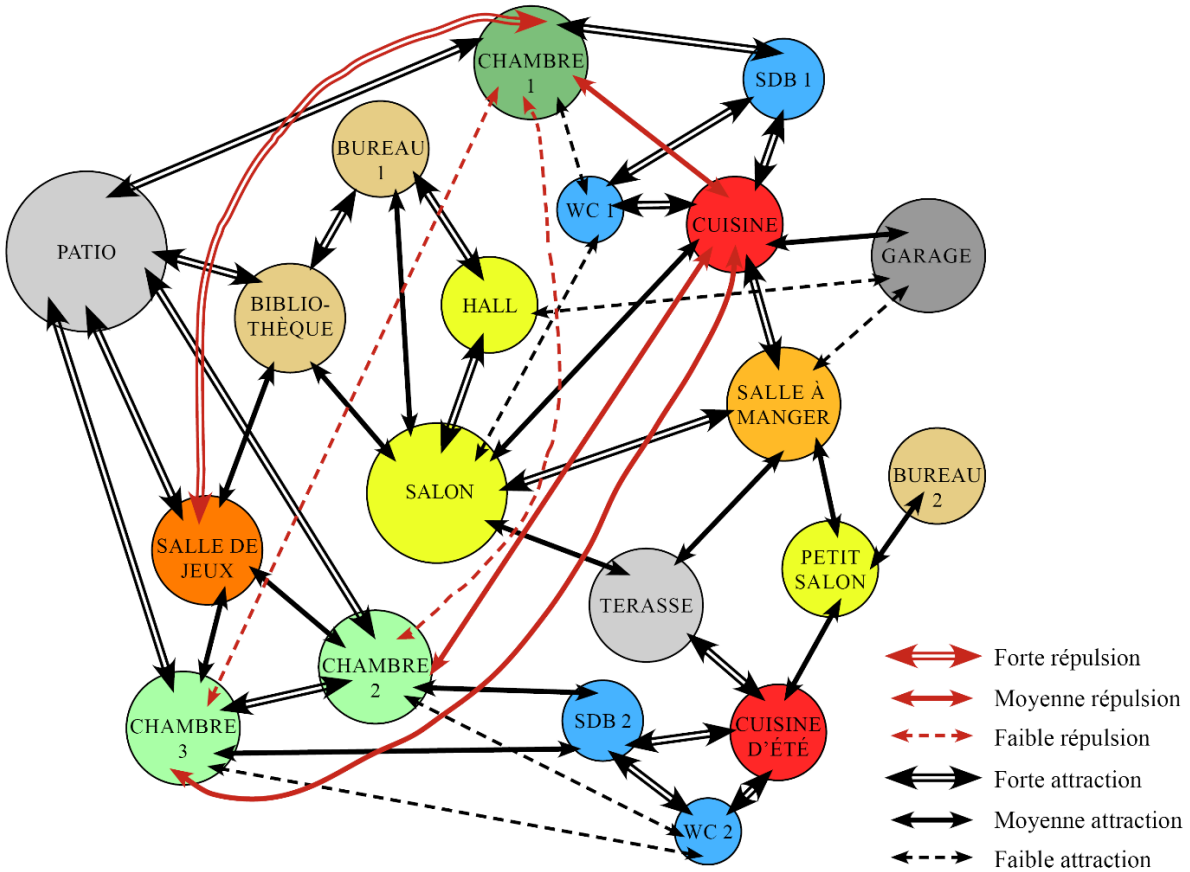


Figure 364 : Organigramme original de la villa synthétisant les règles d'adjacence et de répulsion

Pour pouvoir être pris en compte dans le modèle, les 6 types de relations qui existent entre les espaces de la villa ont été traduits dans une matrice Excel facilement importable sur Grasshopper® appelée la matrice d'adjacences.

Trois critères sont pris en compte dans le processus d'optimisation : (1) la compacité de la forme résultante de l'union de toutes les pièces, (2) le respect des règles d'attraction, (3) le respect des règles de répulsion.

Description du modèle

Une fois la liste des espaces et la matrice d'adjacence importée, la modélisation du problème se déroule en plusieurs étapes listées sur la Figure 365. Au départ, les bulles sont modélisées autour d'un cercle selon un ordre qui varie pour chaque individu. Pour le paramétrage de cet ordre, un composant Genepool est utilisé, ainsi qu'un composant « Ur_Genepool_Constraint_Variable » afin d'assurer que deux bulles ne soient pas positionnées au même endroit. Seule la bulle du patio est positionnée au centre du cercle, pour que les espaces s'organisent autour de lui.

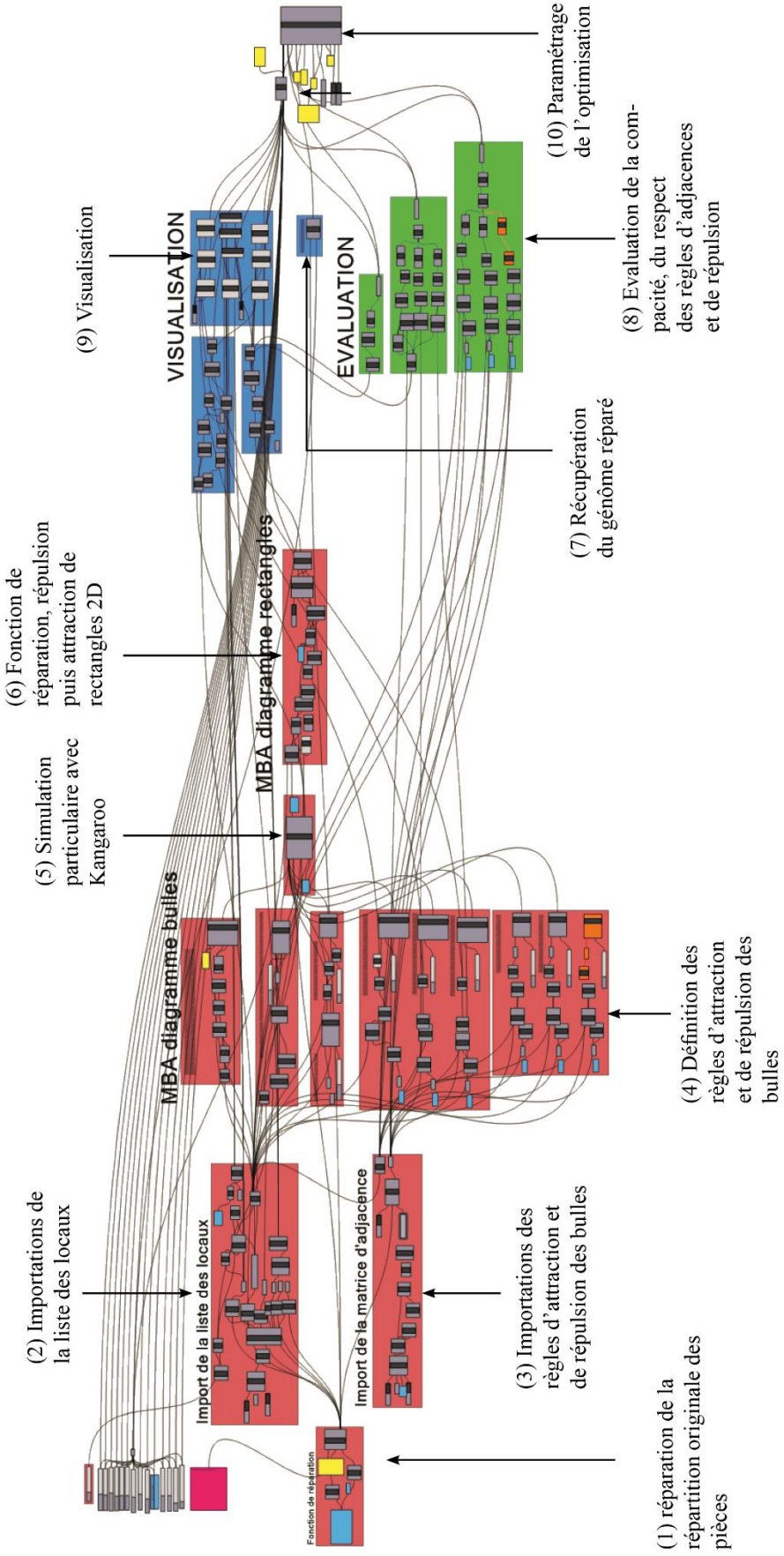


Figure 365 : Les différentes étapes de modélisation du processus d'exploration

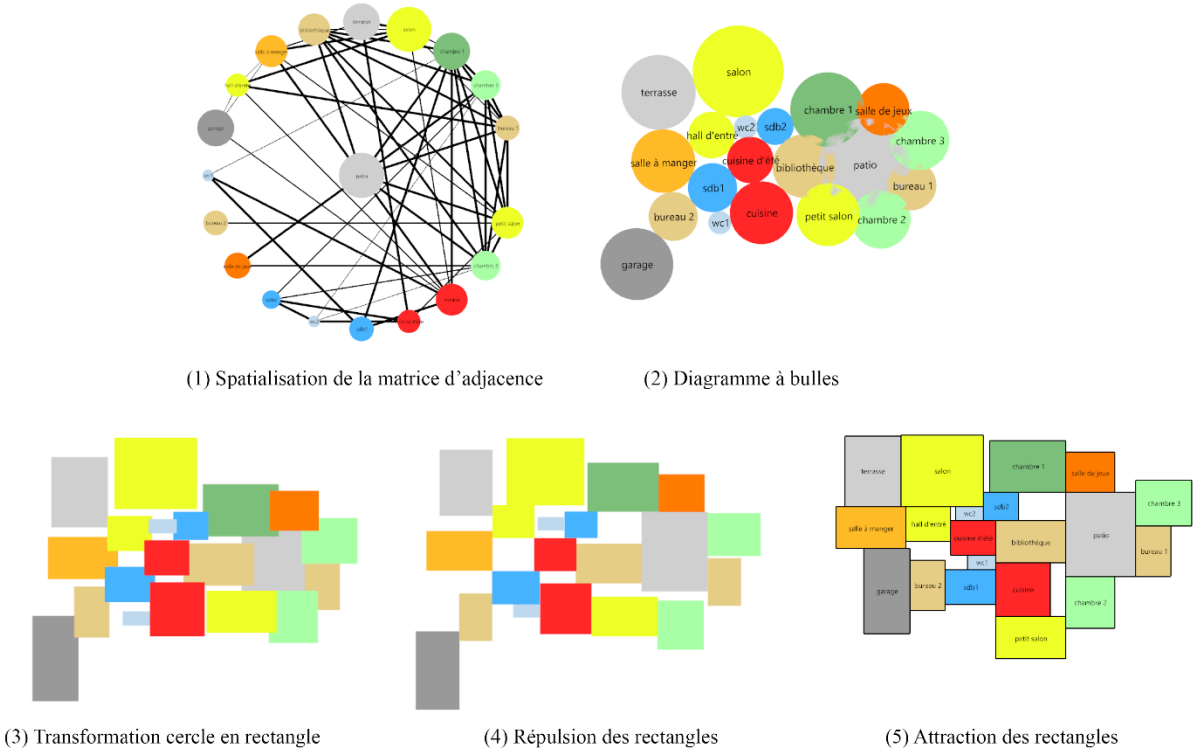


Figure 366 : Les différentes étapes de génération d'une solution

Une simulation particulaire réalisée à l'aide du plugin Kangaroo® est ensuite utilisée pour attirer les bulles entre elles. Des forces d'attraction et de répulsion coefficientée sont utilisées pour intégrer les prescriptions de l'organigramme original. Le résultat de cette simulation à base d'agents particulaire est un diagramme à bulles.

Chaque bulle est ensuite transformée en rectangle dont la surface est constante, mais la largeur varie à chaque nouvel individu généré. La disposition des rectangles est alors réparée une première fois à l'aide du composant « Ur_Repulsion2D », et une seconde fois avec le composant « Ur_Attraction_Rectangles ». Ces différentes étapes nécessaires à la génération d'une solution sont illustrées en Figure 366 avec un exemple.

Résultats

Au total,10 générations de 50 individus chacune vont être calculées avec ce modèle. Les calculs prendront en tout environ 26h30 de calcul. 44 solutions sur le front de Pareto seront trouvées par NSGAII. Elles sont représentées en rouge sur la Figure 367. Parmi elles, 12 sont présentées en Figure 369. Ces représentations sont extraites du catalogue de solutions crées à l'aide du composant « Ur_Poster_Solution_Selection ».

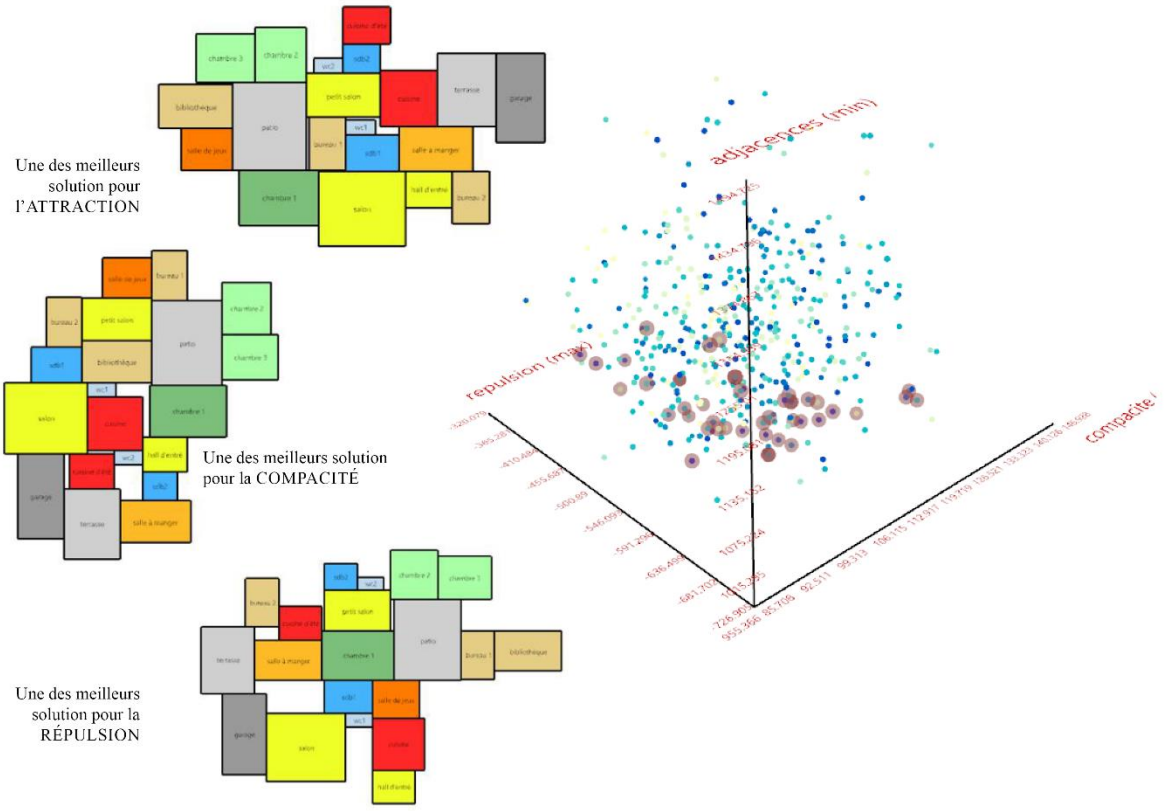


Figure 367 : Graphique en nuage de points représentant l'ensemble des solutions générées

Au vu des graphiques présentés en Figure 368, il semble que l'algorithme ait eu du mal à converger puisque les valeurs des fonctions objectifs semblent ne pas ou peu évoluer au fil des générations.

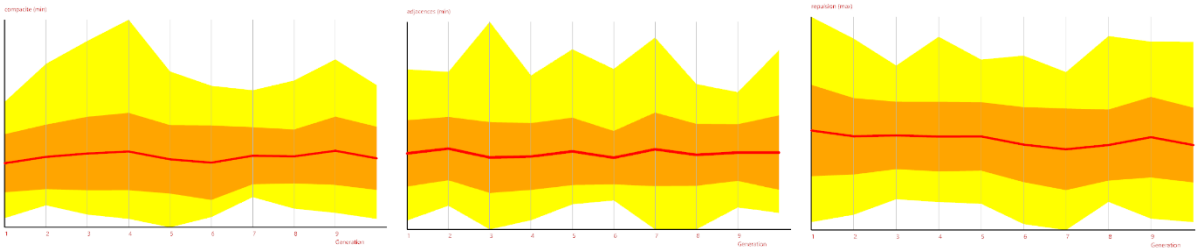


Figure 368 : Evolution des valeurs des fonctions objectifs au fil des générations

Les solutions générées prennent des formes très diversifiés, rarement très compactes (voir l'une des solutions générées les plus compactes dans la Figure 367). Il semble tout de même qu'un tel outil puisse aider les architectes et être un bon intermédiaire entre un organigramme et un plan, notamment ici lorsqu'il s'agit d'une pensée allant des parties (les pièces) au tout (la morphologie de la villa).

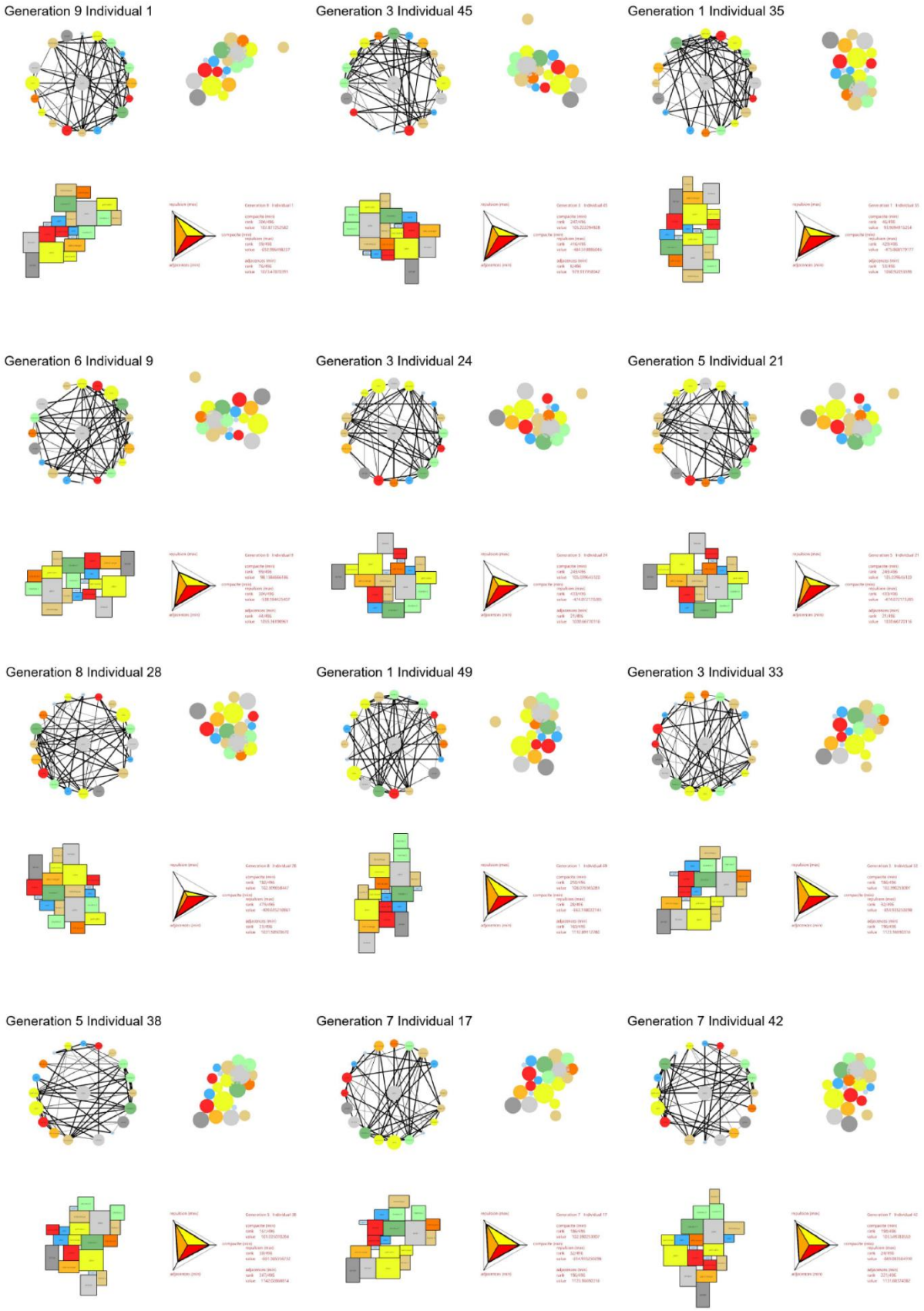


Figure 369 : Solutions extraites du front de Pareto

Au terme de cette troisième partie du chapitre 4, il apparait que le plugin *Urchin* puisse être utile pour modéliser une grande variété de problèmes de conception architecturale et urbaine. En effet, nous avions déjà au début du chapitre montré comment les modèles à base d'agents peuvent être utiles à la modélisation de problèmes relatifs aux enveloppes. Nous avons vu désormais que ces derniers peuvent aussi servir pour explorer la morphologie des bâtiments, les problématiques urbaines et même des problèmes très complexes comme l'agencement des espaces intérieurs.

Nous retenons qu'il y a deux types de réparations, celle avec des agents fixes et celle avec des agents mobiles. Les deux ont leur utilité car elles ne peuvent pas traiter les mêmes problèmes. Les agents mobiles permettent de concevoir des parties au tout, à l'inverse, les agents fixes (les ACA) permettent de concevoir du tout au partie. Les agents fixes peuvent compter beaucoup plus d'agents que les fonctions avec agents mobiles qui présenteraient des temps de calcul trop important.

Si la plupart des algorithmes de réparation ou outils permettant de créer des algorithmes de réparation développés mériteraient d'être approfondies, notamment pour intégrer des contraintes en trois dimensions, ces résultats sur ces quelques premières applications apparaissent comme prometteurs. Cependant, pour prouver leur apport comme solution technique, ces outils restent encore à confronter à la réalité des projets dans un cadre professionnel.

Au terme du chapitre 4, nous espérons avoir contribué à la résolution de l'enjeu qui existe en conception computationnelle sur l'intégration des contraintes lors de modélisation de processus d'exploration basés sur l'usage d'algorithme d'optimisation multicritère. Nous avons identifié des techniques génératives susceptibles de faire de bons candidats pour développer des fonctions de réparations comme les modèles à base d'agents ou les automates cellulaires. Nous les avons testées sur différents problèmes issus de la pratique professionnelle chez Architecture Studio, notamment des problèmes d'enveloppe.

Tout ce travail d'enquête dans la littérature scientifique et lors des nouvelles expérimentations nous a permis d'identifier, d'imaginer et de développer des outils susceptibles de faciliter la modélisation de fonctions de réparation par des non-experts de la programmation informatique. Ces outils et les algorithmes qu'ils contiennent ont été décortiqués, expliqués et présentés dans ce chapitre pouvant servir de guide d'utilisation. Trois catégories d'outils composent ce plugin : (1) les composants nécessaires à la réparation de solutions, qui peuvent servir à contraindre les variables, à créer des boucles conditionnelles, à créer des ACA, ou encore à faire de l'auto-organisation, (2) les composants pour l'optimisation, et (3) les composants pour la visualisation et l'analyse des résultats de l'exploration.

Enfin, une première version du plugin résultant de ces développements nommée « Urchin » a été testée sur six applications inspirées des problèmes rencontrés dans la littérature scientifique et dans la pratique. Nous avons alors pu confirmer l'idée que ce plugin peut être utile à de nombreux types de problèmes différents, autres que la conception des enveloppes, y compris des problèmes réputés complexes comme la génération de plans. Les résultats qu'offrent notamment les problèmes s'appuyant sur l'usage d'un ACA sont encourageants et laissent penser qu'une version 3D du simulateur d'ACA pourrait être une piste pour des recherches futures.

Au terme de cette seconde partie de la thèse, il apparaît que les algorithmes évolutionnaires multicritères sont adaptés à la résolution de problèmes d'optimisation rencontrés en conception architecturale. S'il existe déjà des outils notamment sur Grasshopper ou Dynamo permettant d'utiliser ces algorithmes, il reste difficile avec ces derniers d'intégrer des contraintes lors du processus d'exploration. Nous savons pourtant que les contraintes rencontrées sur des problèmes réels nécessitant d'être intégrées au processus d'optimisation sont fréquentes et de multiples natures : réglementaires, structurels, environnementales, fonctionnelles, voir esthétiques.

Pour les intégrer, deux types de méthodes existent. Les méthodes génériques, qui peuvent facilement être répliquées sur différents problèmes, mais qui fonctionnent mal lorsque la région faisable est petite proportionnellement à la taille de l'espace de solutions. D'autres approches, que nous appelons les méthodes « ad hoc » peuvent se révéler beaucoup plus efficaces, notamment celles qui consistent à utiliser une fonction de réparation, c'est-à-dire un algorithme permettant de rendre une fonction infaisable faisable. Cependant, nous avons pu constater sur différents problèmes que pour être efficaces et permettre à l'algorithme de converger au mieux, elles doivent être utilisées selon la méthode dite Lamarckienne. C'est-à-dire que l'algorithme génétique doit pouvoir être informé pour une partie ou l'ensemble des solutions de la réparation de son génome.

Aussi, nous avons pu constater que les fonctions de réparation efficaces sur les problèmes de conception architecturale étaient souvent des algorithmes génératifs spécifiques notamment à base d'agents qui, malgré les plugins existants, restent difficiles à modéliser avec des outils de programmation visuelle sans utiliser de langage de programmation. Ces derniers n'étant pas maîtrisés par les architectes, nous proposons une première librairie d'outils développés pour Grasshopper® qui permet de résoudre des problèmes connus comme les collisions d'éléments en 2D ou des problèmes plus spécifiques avec des règles personnalisées notamment avec une méthode inspirée des AC que nous appelons des « automates cellulaires augmentés ». Enfin, un nouveau solveur d'optimisation contenant NSGAII et permettant d'utiliser la méthode de réparation Lamarckienne complète cette librairie d'outils qui peuvent servir à l'exploration du design des enveloppes, mais aussi de la morphologie des bâtiments, au design urbain, voir à l'agencement des espaces intérieurs.

Conclusion

Le secteur du bâtiment a un très fort impact environnemental conduisant les réglementations de construction à devenir de plus en plus exigeantes et les architectes ont un rôle important à jouer dans la transition environnementale. Si le choix des matériaux est un critère essentiel, il a aussi été montré que les choix formels de l'enveloppe des bâtiments, de leurs morphologies et leurs orientations - voir même de l'agencement des espaces intérieurs - ont un impact important sur leurs consommations énergétiques. Dès lors, il apparaît que les phases amonts de projet, où les architectes ont un rôle prépondérant, sont essentielles pour assurer la performance énergétique et environnementale des projets. Les outils de simulation de la performance des bâtiments ont été essentiels aux architectes et ingénieurs des équipes de maîtrise d'œuvre pour avancer vers une conception plus écologique. Cependant, ils restent peu utilisés en phases amonts de projet, notamment par les architectes pour lesquels ces logiciels aux interfaces opaques restent difficiles d'accès.

Pourtant, depuis la fin des années 1990, des méthodes numériques innovantes sont développées par les chercheurs pour aider les architectes à prendre en compte les contraintes environnementales dès les premières phases du projet. Nous avons vu dans nos recherches que ces méthodes de conception computationnelles sont à la fois des méthodes de conception paramétriques, génératives, intégrées, environnementales ou performancielles. Nous les nommons dans cette thèse les processus d'« exploration numérique de solutions architecturales pour la performance environnementale ». Elles cherchent à automatiser le processus itératif qui existe dans la conception traditionnelle entre architecte et ingénieur en utilisant le plus souvent trois phases de modélisation : un modèle génératif créé à l'aide le plus souvent d'un modèle paramétrique, un modèle d'évaluation réalisé à l'aide de logiciels de simulation, et un modèle d'exploration avec la plupart du temps un algorithme d'optimisation génétique. Leur mise en œuvre comprend 5 étapes. La première est une phase d'analyse nécessaire à la formulation du problème. La deuxième est celle du prototype avec la réalisation d'un premier modèle paramétrique. La troisième est celle de l'élaboration du modèle d'évaluation. La quatrième est principalement constituée des temps de calcul de l'exploration. La dernière étape est celle de la visualisation et de l'analyse des résultats de l'exploration. Pour mettre en œuvre une exploration numérique, de nombreux outils différents sont nécessaires : (i) un modeleur 3D équipé d'une plateforme de programme visuelle comme Rhinocéros ou Revit est l'outil central ; (ii) les outils

d'analyse permettant de définir des critères d'optimisation, dans notre cas il s'agit d'outils d'analyse environnemental ; (iii) pour les faire fonctionner, des outils auxiliaires d'interopérabilité doivent être utilisés ; (iv) les outils d'optimisation permettant d'utiliser et de paramétrer des algorithmes mathématiques explorant de grands ensembles de solutions ; et, (v) les outils de *data visualisation* permettant d'analyser les résultats des explorations. En dehors des modeleurs 3D, la plupart de ces outils sont disponibles en *open source* sur internet.

De nombreux cas d'études appliquant ces méthodes sont décrits dans la scientifique, et plusieurs revues de littérature ont déjà fait l'objet de publications. Les ingénieurs s'intéressent depuis longtemps à l'optimisation de la conception des bâtiments. Toutefois, peu de revues se concentrent sur les applications prenant en compte des préoccupations architecturales. Ainsi, nous avons réalisé une revue de littérature sur les méthodes d'exploration numériques appliquées à l'enveloppe, la morphologie des bâtiments et des îlots urbains, à l'agencement des espaces intérieurs et la planification urbaine. 61 applications portant sur des caractéristiques qui intéressent les architectes ont été sélectionnées pour constituer un catalogue de références pour les praticiens de l'architecture qui souhaitent recourir à l'exploration numérique.

Cette revue de littérature a permis de montrer que peu d'applications ont été effectuées dans des contextes professionnels et que la conception d'enveloppe est le sujet le plus représenté, notamment pour la protection solaire. Nous avons aussi pu constater que si le triptyque modèle paramétrique/logiciel de simulation/algorithme génétique est surreprésenté, il existe d'autres méthodes moins représentées, par exemple les grammaires de forme pour modèle génératif, les modèles de substitution pour le modèle d'évaluation ou les fonctions aléatoires pour l'exploration. Certains types de problèmes sont beaucoup plus compliqués à modéliser que d'autres, comme l'agencement des espaces intérieurs ou la génération de réseaux viaires qui nécessitent la maîtrise de techniques génératives spécifiques. Peu d'exemples d'applications ont été capables d'intégrer d'autres critères que le critère environnemental, comme les exigences réglementaires qui constituent des contraintes inviolables. Finalement, nous avons pu constater que les applications centrées sur des préoccupations architecturales sont souvent plus complexes à modéliser que les problèmes relevant essentiellement d'ingénierie. Pour pratiquer la conception générative et performancielle, les architectes ont besoin de très solides compétences en conception computationnelle.

Pour faciliter la mise en œuvre de ces méthodes dans la pratique professionnelle, nous avons identifié puis rassemblé des outils d'évaluation compatibles avec des outils d'optimisation. Il existe de nombreux *plugins* disponibles en *open source* conçus pour aider les

architectes à intégrer l'analyse dans leur processus de conception, notamment des outils d'évaluation de l'éclairage naturel, de l'ensoleillement, de la qualité de vue, de l'éblouissement, du confort thermique, du confort au vent, des besoins ou consommations énergétiques et de l'analyse du cycle de vie. Nous avons complété cette bibliothèque avec le développement d'outils pour l'analyse des protections contre les précipitations, ou encore l'analyse de la vue sur un élément remarquable.

Malgré toutes ces méthodes et outils à disposition, l'usage de l'exploration numérique par les architectes reste rare. Dans la littérature scientifique, plusieurs explications sont données pour expliquer cette sous-utilisation. Elles peuvent être classées en quatre catégories : le manque de formation des architectes, un manque d'interface pour relier les différents outils et d'interfaces graphiques pour rendre ces outils faciles d'accès, la remise en cause de la robustesse de la méthode et les limites techniques comme les temps de calculs jugés trop longs. Avant de chercher comment surmonter ces difficultés, nous avons confronté ces méthodes théoriques à la réalité de la pratique en agence chez Architecture Studio afin de vérifier si ces explications sont toujours valables. 61 expérimentations au total ont été réalisées à la suite de sollicitations des équipes de conception allant de la simple évaluation à l'optimisation multicritère sous contraintes en passant par la méthode de la « géométrie informée », l'étude paramétrique, l'optimisation mono et multicritère. La plupart des expérimentations réalisées ont été appliquées à l'enveloppe du bâtiment.

Si d'autres chercheurs ont depuis montré la robustesse de la méthode, nos expérimentations nous ont permis de constater que le manque d'interface n'était plus un sujet, et que le manque de formation était à nuancer au vu des nombreuses formations données dans les universités du monde entier. Néanmoins, nous avons identifié un nouveau type de verrou, les limites « structurelles » liées aux méthodes de travail et d'organisation des architectes, peu adaptées à la conception générative. Pour formuler le problème, le « *computational designer* » (le dessinateur qui maîtrise les outils nécessaires à la conception générative) doit avoir une vision globale des contraintes du projet, ce qui n'est pas toujours le cas. Aussi, les habitudes de management des chefs d'agence où le dessinateur produit et l'architecte corrige sont difficiles à reproduire lorsqu'il s'agit de produire des centaines de variations de projets. Enfin, si les temps de calcul peuvent représenter un frein à l'usage des méthodes génératives et performancielles, elle n'est pas apparue comme la limite technique majeure. Certains problèmes rencontrés par les architectes sont parfois tellement contraints que leur modélisation devient impossible pour un non-expert de la programmation informatique. Ces observations ont pu être

illustrées à l'aide de différentes expérimentations, certaines ayant fonctionné, et d'autres pour lesquelles la mise en œuvre de techniques d'exploration numériques n'ont pu aboutir.

Pour tenter de réduire l'impact des temps de calcul sur le succès de ces méthodes, nous avons identifié et testé des techniques connues dans la littérature pour les réduire en utilisant des algorithmes de *Machine Learning*. Nous avons constaté qu'un grand nombre de ces algorithmes sont déjà à la disposition des architectes sur Grasshopper via des *plugins open source*. Ainsi, les régressions multi-variables, ou les réseaux de neurones permettent de faire des prédictions fiables pour des critères tels que les besoins et consommations énergétiques. Pour les indicateurs de confort comme l'éclairage naturel, l'ensoleillement, ou l'écoulement des vents, les IA génératives, comme les GAN (*Generative Adversal Network*), permettent de faire des prédictions plus précises qu'avec les autres algorithmes, bien qu'ils ne soient pas, à l'heure actuelle, aussi accessibles pour les novices de la programmation. En utilisant des GAN, on ne cherche pas à prédire la valeur d'une fonction d'évaluation, mais à prédire les résultats graphiques en générant une image correspondant à la visualisation en plan de l'ensemble des résultats d'une évaluation.

S'il existe des solutions accessibles au problème des temps de calcul, le problème de l'intégration des contraintes dans les processus d'optimisation restait à traiter. Pour cela, nous avons commencé par identifier la nature des problèmes d'optimisation rencontrés en conception architecturale. Ce sont principalement des problèmes d'optimisation globale, discontinue, sans gradient, et multicritère. Ainsi, il convient de les résoudre à l'aide d'algorithmes métaheuristiques comme les algorithmes génétiques. S'il existe plusieurs méthodes pour faire de l'optimisation multicritère comme la méthode lexicographique ou la scalarisation, la méthode de Pareto apparaît comme la plus adaptée pour résoudre les problèmes rencontrés par les architectes. Il faut alors utiliser des algorithmes adaptés comme NSGAII, l'un des plus utilisé pour la conception des bâtiments. Nous avons pu constater qu'il existe de nombreux outils à disposition des architectes pour faire de l'optimisation notamment sur Grasshopper, mais peu permettent de faire de l'optimisation multicritère, comme Octopus ou Wallacei, mais ils n'intègrent pas de fonction spécifique pour faciliter l'intégration de contraintes.

Pourtant ces dernières représentent un véritable enjeu pour la modélisation de processus d'exploration numérique appliqués à l'architecture. En optimisation mathématiques, les contraintes sont à distinguer des critères d'optimisation. Il s'agit de conditions qui doivent être nécessairement respectées pour que la solution générée numériquement présente un intérêt pour l'architecte. Ces contraintes peuvent être de différentes natures : réglementaires,

programmatiques, constructives, environnementales, esthétiques, ou simplement nécessaires pour éviter des aberrations géométriques. Pour les intégrer à un processus d'optimisation, il existe différentes méthodes plus ou moins efficaces selon la taille de la région faisable, c'est-à-dire la proportion de solutions qui respectent les contraintes dans un ensemble de solution donné. Les méthodes réplicables comme celles qui consistent à utiliser des fonctions de pénalité, ou à traiter les contraintes comme un critère d'optimisation ou encore des méthodes internes propres à certains algorithmes comme Constraint-NSGAII, sont efficaces si la région faisable est relativement grande. Nous avons montré en nous appuyant sur un cas d'étude issu de la pratique professionnelle que les méthodes *ad hoc*, qui consistent à déterminer une heuristique propre à un problème est beaucoup plus efficace, voir indispensable lorsque la région faisable est très petite. La méthode des fonctions de réparation avec un algorithme Larmarckien s'est révélée particulièrement appropriée. Elle consiste à modifier les solutions ne respectant pas les contraintes à l'aide d'une heuristique qui les rendent faisables et à informer l'algorithme génétique de la modification du génome pour au moins une partie des solutions réparées afin de faciliter la convergence de l'algorithme génétique.

L'expérimentation de cette méthode sur différents problèmes de conception d'enveloppe et de projets urbains, toujours issus de la pratique chez Architecture Studio, a permis de constater que les heuristiques de réparation conçues à l'aide de techniques génératives spécifiques telles que des automates cellulaires ou d'autres modèles à base d'agents comme l'intelligence en essaim étaient particulièrement efficaces. Cependant, à la suite d'un état de l'art sur les outils facilitant l'utilisation de ce type de techniques par les non-experts de la programmation informatique, nous avons compris qu'il n'existait rien d'adapté pour l'élaboration de fonctions de réparation. Aussi, dans le cadre d'expérimentations sur d'anciens projets, nous avons dû développer des outils codés en python directement dans Grasshopper. Ce temps de développement aurait été incompatible avec l'agenda d'un concours en architecture.

Cette thèse réalisée dans le cadre d'une convention CIFRE devant aboutir à des résultats exploitables par des architectes formés à la programmation visuelle, nous avons rassemblé l'ensemble des scripts produits lors des expérimentations sur les projets en cours chez Architecture Studio, mais aussi ceux produits pour les études comparatives dans un unique *plugin* Grasshopper dédié à l'optimisation multicritère sous contrainte nommé « Urchin ». Les différents prototypes réalisés ont été retravaillé afin de pouvoir s'adapter à différents types de problèmes. 34 composants pour Grasshopper ont été élaborés. Une vingtaine sont dédiés à la

réparation de solutions, avec notamment un ensemble d'outils permettant de modéliser des automates cellulaires complexes, mais aussi des algorithmes à base d'agents mobiles pensés pour traiter le problème connu des superpositions. Quatre composants permettent de lancer des processus d'optimisation avec l'algorithme génétique NSGAII. Les dix composants restants servent à l'analyse et la visualisation des résultats. Ils permettent notamment de trier les solutions, de produire des graphiques individualisés pour les solutions situées sur le Front de Pareto et de générer des catalogues de solutions au format pdf.

Conscients que les problèmes d'enveloppes sont les plus accessibles des problèmes d'optimisation rencontrés en conception architecturale, nous avons conclu cette recherche en testant ce nouveau *plugin* sur d'autres types problèmes inspirés des cas d'étude rencontrés dans la littérature scientifique et de la pratique. Ainsi, nous avons pu apprécier la capacité du *plugin* à nous assister dans la production rapide d'heuristiques pour intégrer des contraintes sur des problèmes d'exploration de la morphologie des bâtiments, de conception urbaine ou encore d'agencement d'espaces intérieurs.

Finalement, nous espérons avoir atteint l'objectif de rassembler dans cette thèse toutes les connaissances nécessaires pour mettre en pratique le design génératif et performanciel avec les outils à disposition des architectes aujourd'hui, afin qu'elle puisse être utile aux chercheurs, mais aussi aux praticiens qui souhaitent s'initier et se former à la conception générative. Cependant**,** nous avons bien conscience que certains sujets auraient nécessité des recherches complémentaires que nous souhaiterions pouvoir réaliser dans un avenir proche.

Notamment, les algorithmes de réparation de notre plugin sont le résultat d'une recherche centrée sur l'enveloppe des bâtiments qui est un type de problème parmi d'autres. Les contraintes traitées sont donc principalement en deux dimensions. En testant le *plugin* sur des problèmes autres, il apparait qu'un vrai potentiel pour une adaptation en trois dimensions existe, notamment pour l'automate cellulaire complexe. En effet, dans la pratique professionnelle, l'allocation spatiale de pièces en plan avec des voxels n'est pas suffisamment précise pour être intéressante, cependant il peut être pertinent avant de travailler les plans, d'être assisté dans la disposition du programme dans un grand volume, notamment pour la conception de grands équipements publics.

Concernant la réduction des temps de calcul des simulations, l'étude portant sur les GAN a permis de montrer des prédictions graphiques prometteuses. Une traduction numérique de ces résultats graphiques permettrait de véritablement mesurer l'écart de performance entre

cette méthode et les méthodes plus traditionnelles comme les régressions multi-variables. Ces IA génératives possibles avec les GAN ont un autre avantage, celui de pouvoir produire rapidement des visualisations des résultats dans l'espace, ce qui est essentiel aux architectes pour interpréter les résultats. Il serait très opportun pour un développement futur d'Urchin, d'intégrer des composants pour créer un modèle de substitution avec des GAN en utilisant les captures d'écrans des résultats des solutions générées lors des premières générations du processus d'optimisation.

Ensuite, pour pouvoir attester de l'utilité de ces différents apports techniques pour l'intégration de contraintes, il serait nécessaire de confronter l'utilisation de ce plugin à des cas pratiques dans un cadre professionnel. Il en va de même pour l'utilisation des algorithmes de *Machine Learning* pour la réduction des temps de calcul des simulations. Pour le moment, ils n'ont pas été testés sur des projets en cours en agence d'architecture. Aussi, il serait intéressant qu'ils soient testés par différents praticiens et non pas uniquement par des chercheurs. De façon générale, bien que nous ayons réalisé un grand nombre d'expérimentations dans un cadre professionnel au cours de ce doctorat, il reste nécessaire, selon nous, de poursuivre les expérimentations en agence d'architecture, si l'on souhaite réduire l'écart qui existe entre la théorie et la pratique concernant le design génératif. Il faudrait notamment plus d'expérimentations impliquant la participation de l'ensemble de l'équipe de conception.

Ce doctorat ayant été réalisé dans le contexte particulier de la crise sanitaire de 2020, nos expérimentations ont souvent eu lieu en périphérie des équipes. Les retours des membres des équipes sur les solutions générées, ou sur la clarté et la pertinence des résultats produits, l'accès à l'ensemble des informations concernant les projets étudiés, les décisions prises suite aux résultats des analyses sont autant d'éléments qui nous ont manqués. Nous conseillons à ceux qui seraient tenter d'expérimenter le design génératif en agence de privilégier la qualité à la quantité, en choisissant des projets propices à ce type d'approche avec des chefs de projet sensibilisés à la démarche et préalablement formés aux outils de data visualisation. Des expérimentations en collaboration avec un bureau d'étude d'ingénieurs sont aussi à imaginer, car ces méthodes remettent en question la répartition traditionnelle des rôles dans une équipe de maîtrise d'œuvre. Cela permettrait d'en savoir un peu plus sur l'impact du design génératif sur la relation architecte-ingénieur. Cela pourrait aussi permettre d'être plus ambitieux sur les critères à intégrer, en utilisant des outils de simulations plus sophistiqués comme la STD.

Enfin, bien que dans cette thèse une partie du chapitre 2 soit dédiée aux algorithmes de *Machine Learning*, cette recherche de doctorat sur le design génératif et performanciel se concentre principalement sur l'utilisation et la mise en œuvre des algorithmes d'optimisation. Or, nous savons que d'autres algorithmes, notamment les algorithmes de *Deep Learning* ont un fort potentiel pour la conception générative, en plus de la production de modèles de substitution. Ils peuvent être utilisés à chaque phase de la modélisation d'un problème multicritère sous contraintes. Ils pourraient servir pour générer l'ensemble de solution, pour créer des algorithmes de réparation, pour créer des fonctions d'évaluation de critères qualitatifs (esthétique, vue), pour améliorer les performances de l'algorithme d'optimisation, ou encore pour faciliter l'analyse des résultats lorsque la quantité de solutions générées est importante. Ainsi, les recherches sur les apports du *Deep Learning* pour l'optimisation multicritère sous-contraintes pourraient faire l'objet de développements futurs.

Pour conclure sur des réflexions et apprentissages plus généraux, nous retenons et souhaitons souligner plusieurs choses. D'abord, nous avons été témoin grâce à cette recherche de la grande richesse que représentent tous ces outils mis à la disposition des architectes, notamment sur Grasshopper, que ce soit pour faire de l'optimisation, de la modélisation à base d'agents, de l'évaluation environnementale ou du *Machine Learning*. Encore bien d'autres outils existent pour d'autres sujets. Ils sont souvent le fruit de travaux de chercheurs ou de passionnés et sont accessibles facilement et gratuitement pour la plupart. Leur potentiel semble immense pour les académiques comme les praticiens. Certains représentent un gain de productivité, d'autres donnent accès à de nouvelles compétences ou sont source de créativité.

Pourtant, ils restent encore sous-exploités dans les agences d'architecture trop occupées à transitionner vers des méthodes BIM, rendues obligatoires pour certains marchés publics, pour s'intéresser à cet univers.

Faire une thèse en agence dans le champ de la conception numérique fut aussi l'occasion de faire découvrir à nos collaborateurs architectes cet univers souvent plus connu des chercheurs que des praticiens, mais aussi de les convaincre de l'intérêt de faire de la veille technologique en agence d'architecture. La recherche scientifique ne fait pas partie de la formation des architectes, pourtant les innovations qui en découlent peuvent se révéler très utiles pour le quotidien des praticiens même dans un avenir proche. Ainsi, nous espérons que les agences d'architecture continueront à l'avenir à accueillir des doctorants.

Un autre enseignement à retenir est que la pratique du design génératif demande des compétences transdisciplinaires et une grande culture de la conception computationnelle. L'explosion récente des IA génératives peut laisser penser que le design génératif appliqué à l'architecture sera bientôt à portée de main, mais la conception architecturale computationnelle va bien au-delà de la prédiction d'images. Pour faire de l'exploration numérique de solutions architecturales à des fins, il est nécessaire de savoir-faire de la simulation environnementale, de la modélisation paramétrique, savoir utiliser des algorithmes d'optimisation, être familier de la méthode de Pareto, savoir utiliser des algorithmes de Machine Learning pour créer des modèles de substitution et connaître la modélisation à base d'agents pour intégrer des contraintes.

Clairement, une formation en master à la modélisation paramétrique en école d'architecture ne suffit pas pour faire du design génératif en pratique, une formation longue, comme l'on en trouve dans de nombreuses écoles à travers le monde, semble indispensable. En France, des écoles d'arts et de design et des écoles d'ingénieurs proposent ce type de formation longues, mais pas les écoles d'architecture.

Enfin**, s**i certains architectes, dont nous faisons partie, sont fascinés par les outils numériques et leurs potentiels, d'autres craignent que ces outils puissent automatiser en grande partie les tâches qui incombent aux architectes jusqu'à finir par les remplacer en détruisant au passage la qualité de l'architecture. Malgré la popularité récente d'applications d'IA génératives rendues possibles grâce aux réseaux de neurones, qui, on peut le comprendre, effraient une partie de la profession, nous pouvons affirmer, après plusieurs années de pratique et de recherches sur le design génératif, que cela n'a, pour le moment, pas grand-chose d'automatique.

Les tâches qui incombent à l'architecte sont très diverses. Les problèmes diffèrent selon les programmes, les climats, les clients, les budgets, les réglementations... Les contraintes ne cessent d'évoluer au cours des projets, rendant la formulation des problèmes de conception impossible à figer. Le design génératif permet d'assister les architectes pour la résolution de sous-problèmes de conception, seulement si ces derniers sont formulables. Lorsque c'est le cas, bien souvent ils nécessitent le développement d'un script spécifique. Ainsi, il faudra toujours un être humain pour formuler les problèmes et pour réaliser les scripts et les utiliser. En l'état, ces algorithmes ont peu de chance de remplacer les architectes. En revanche, un écart pourrait à terme se creuser entre ceux qui maîtrisent les algorithmes et les autres.

Bibliographie

Abdelwahab, S., & Elghazi, Y. (2016). A generative performance-based design for low cost brickwork screens. *Proceedings of the Building Simulation and Optimization Conference (BSO16)*.

ADEME Bâtiments. (s. d.). Agence de la transition écologique. Consulté 3 janvier 2023, à l'adresse https://www.ademe.fr/les-defis-de-la-transition/batiments/

Agence Internationale de l'énergie, & Programme des Nations Unies pour l'environnement. (2018). *2018 Global Status Report : Towards a Zero-emission, Efficient and Resilient Buildings and Construction Sector*. https://wedocs.unep.org/xmlui/handle/20.500.11822/27140

Agirbas, A. (2019). Façade form-finding with swarm intelligence. *Automation in Construction*, *99*, 140-151. https://doi.org/10.1016/j.autcon.2018.12.003

Alfaro, H. (2009). *Generating Interesting Patterns in Conway ' s Game of Life Through a Genetic Algorithm*. https://www.semanticscholar.org/paper/Generating-Interesting-Patterns-in-Conway-%E2%80%99-s-Game-Alfaro/ba7759e4d871d09459e3751d110137a8434591f6

Ali Elfa, M. A., & Dawood, M. E. T. (2023). Using Artificial Intelligence for enhancing Human Creativity. *Journal of Art, Design and Music*, *2*(2). https://doi.org/10.55554/2785-9649.1017

Alshoubaki, H., Rawaswheh, T., Al omari, K., & Hammad, R. (2016). *Innovative design strategy to develop facades opening for a commercial building in Amman, Jordan*. *11*, 5288-5292.

Anderson, C., Bailey, C., Heumann, A., & Davis, D. (2018). Augmented space planning : Using procedural generation to automate desk layouts. *International Journal of Architectural Computing*, *16*(2), 164-177.

Anton, I., & Tănase, D. (2016). Informed geometries. Parametric modelling and energy analysis in early stages of design. *Energy Procedia*, *85*, 9-16.

António, C. A. C., Monteiro, J. B., & Afonso, C. F. (2014). Optimal topology of urban buildings for maximization of annual solar irradiation availability using a genetic algorithm. *Applied thermal engineering*, *73*(1), 424-437.

Apellániz, D., Pasanen, P., & Gengnagel, C. (s. d.). *A Holistic and Parametric Approach for Life Cycle Assessment in the Early Design Stages*.

Apellániz, D., Pettersson, B., & Gengnagel, C. (2023). A Flexible Reinforcement Learning Framework to Implement Cradle-to-Cradle in Early Design Stages. In C. Gengnagel, O. Baverel, G. Betti, M. Popescu, M. R. Thomsen, & J. Wurm (Éds.), *Towards Radical Regeneration* (p. 3-12). Springer International Publishing. https://doi.org/10.1007/978-3-031-13249-0_1

Araújo, G. R., Gomes, R., Gomes, M. G., Guedes, M. C., & Ferrão, P. (2023). Surrogate Models for Efficient Multi-Objective Optimization of Building Performance. *Energies*, *16*(10), Article 10. https://doi.org/10.3390/en16104030

AREC, A. régionale énergie-climat-. (s. d.). *Guide Bio-tech : L'éclairage naturel*. AREC. Consulté 9 février 2023, à l'adresse https://www.arec-idf.fr/nos-travaux/publications/guide-bio-tech-leclairage-naturel/

AREP L'hypercube. (s. d.). Pluie entraînée par le vent. *L'hypercube*. Consulté 20 février 2023, à l'adresse https://lhypercube.arep.fr/aeraulique/pluie-entrainee-par-le-vent-2/

Ashrafi, N. and D. (2017). A shape-grammar for double skin facades—A basis for generating context sensitive facades solution. *Fioravanti, A, Cursi, S, Elahmar, S, Gargaro, S,*

Loffreda, G, Novembri, G, Trento, A (eds.), ShoCK! - Sharing Computational Knowledge! - Proceedings of the 35th eCAADe Conference - Volume 2, Sapienza University of Rome, Rome, Italy, 20-22 September 2017, pp. 471-476. http://papers.cumincad.org/cgi-bin/works/2015%20+dave=2:/Show?ecaade2017_133

Aste, N., Angelotti, A., & Buzzetti, M. (2009). The influence of the external walls thermal inertia on the energy performance of well insulated buildings. *Energy and buildings*, *41*(11), 1181-1187.

Attia, S. (2011). *State of the Art of Existing Early Design Simulation Tools for Net Zero Energy Buildings : A Comparison of Ten Tools*. https://doi.org/10.13140/RG.2.2.35424.23049

Attia, S., Beltran, L., Herde, A., & Hensen, J. (2009). « Architect friendly » : A comparison of ten different building performance simulation tools. *IBPSA 2009 - International Building Performance Simulation Association 2009*, 204-211.

Attia, S., Hamdy, M., O'Brien, W., & Carlucci, S. (2013). Assessing gaps and needs for integrating building performance optimization tools in net zero energy buildings design. *Energy and Buildings*, *60*, 110-124. https://doi.org/10.1016/j.enbuild.2013.01.016

Avison, D., Lau, F., Myers, M., & Nielsen, P. A. (1999). Action Research. *Communications of the ACM*, *42*(1), 94-95.

Aydın, E. E., Dursun, O., Chatzikonstantinou, I., & Ekici, B. (2015). Optimisation of energy consumption and daylighting using building performance surrogate model. *49th International Conference of the Architectural Science Association*, 536-546.

Bamdad, K., Cholette, M. E., & Bell, J. (2020). Building energy optimization using surrogate model and active sampling. *Journal of Building Performance Simulation*, *13*(6), 760-776. https://doi.org/10.1080/19401493.2020.1821094

Bao, F., Schwarz, M., & Wonka, P. (2013). Procedural facade variations from a single layout. *ACM Transactions on Graphics*, *32*(1), 1-13.

Barati, R. (2014). *Analysis and Evaluation of Optimization Algorithms Application for Parameter Estimation of Muskingum Flood Routing Models in Rivers*. DOI.

Bartz-Beielstein, T. (2016). A survey of model-based methods for global optimization. *Bioinspired Optimization Methods and Their Applications*, 1-18.

Bechthold, M., King, J., Kane, A., Niemasz, J., & Reinhart, C. (2011). *Integrated Environmental Design and Robotic Fabrication Workflow for Ceramic Shading Systems*. 70-75. https://doi.org/10.22260/ISARC2011/0010

Bellia, L., Cesarano, A., Iuliano, G. F., & Spada, G. (2008). Daylight glare : A review of discomfort indexes. *Visual Quality and Energy Efficienty in Indoor Lighting: Today for Tomorrow*.

Benedikt, M. L., & Burnham, C. A. (1985). Perceiving architectural space : From optic arrays to isovists. *Persistence and change*, 103-114.

Beni, G. (2004). From swarm intelligence to swarm robotics. *International Workshop on Swarm Robotics*, 1-9.

Bernal, M., Okhoya, V., Marshall, T., Chen, C., & Haymaker, J. (2020). Integrating expertise and parametric analysis for a data-driven decision-making practice. *International Journal of Architectural Computing vol. 18 - no. 4, 424–440*. http://papers.cumincad.org/cgi-bin/works/paper/ijac202018407

Best, A. C. (1950). The size distribution of raindrops. *Quarterly journal of the royal meteorological society*, *76*(327), 16-36.

Bevan, R. L. T., Poole, D. J., Allen, C. B., & Rendall, T. C. S. (2017). Adaptive Surrogate-Based Optimization of Vortex Generators for Tiltrotor Geometry. *Journal of Aircraft*, *54*(3), 1011-1024. https://doi.org/10.2514/1.C033838

Bierlaire, M. (2006). *Introduction à l'optimisation différentiable*. PPUR presses polytechniques.

Blank, J., & Deb, K. (2020). pymoo : Multi-objective Optimization in Python. *IEEE Access*, *8*, 89497-89509. https://doi.org/10.1109/ACCESS.2020.2990567

Błażejczyk, K., Jendritzky, G., Bröde, P., Fiala, D., Havenith, G., Epstein, Y., Psikuta, A., & Kampmann, B. (2013). An introduction to the Universal Thermal Climate Index (UTCI). *Geographia Polonica*, *86*(1), 5-10. https://doi.org/10.7163/GPol.2013.1

Blocken, B., & Carmeliet, J. (2004). A review of wind-driven rain research in building science. *Journal of wind engineering and industrial aerodynamics*, *92*(13), 1079-1130.

Boissieu, A. de. (2020). Super-utilisateurs ou super-spécialistes ? Cartographie des catalyseurs de la transformation numérique en agence d'architecture. *Les Cahiers de la recherche architecturale urbaine et paysagère*, *9|10*, Article 9, 10. https://doi.org/10.4000/craup.5551

Boon, C., Griffin, C. T., Papaefthimiou, N., Ross, J., & Storey, K. (2015). Evolutionary parametric analysis for optimizing spatial adjacencies. *of Architectural Research*, 312.

Botchkarev, A. (2019). A new typology design of performance metrics to measure errors in machine learning regression algorithms. *Interdisciplinary Journal of Information, Knowledge, and Management*, *14*, 045-076.

Boudon, P., Deshayes, P., Pousin, F., & Schatz, F. (1994). *Enseigner la conception architecturale. Cours d'architecturologie* (Editions de la Villette).

Bouhelal, A., & Smaïli, A. (2022, juin 19). *Introduction à la CFD (Computational Fluid Dynamics)*.

Brownlee, J. (2011). *Clever Algorithms : Nature-inspired Programming Recipes*. Jason Brownlee.

Building Energy Software Tools. (s. d.). IBPSA-USA Best Directory. Consulté 9 janvier 2023, à l'adresse https://www.buildingenergysoftwaretools.com/home?page=1

Caetano, I., Santos, L., & Leitão, A. (2020). Computational design in architecture : Defining parametric, generative, and algorithmic design. *Frontiers of Architectural Research*, *9*(2), 287-300. https://doi.org/10.1016/j.foar.2019.12.008

Caldas, L. (2008). Generation of energy-efficient architecture solutions applying GENE_ARCH : An evolution-based generative design system. *Advanced Engineering Informatics*, *22*(1), 59-70. https://doi.org/10.1016/j.aei.2007.08.012

Carlucci, S., Causone, F., De Rosa, F., & Pagliano, L. (2015). A review of indices for assessing visual comfort with a view to their use in optimization processes to support building integrated design. *Renewable and sustainable energy reviews*, *47*, 1016-1033.

Caruana, R., & Niculescu-Mizil, A. (2004). Data mining in metric space : An empirical analysis of supervised learning performance criteria. *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*, 69-78.

Catalina, T., Virgone, J., & Blanco, E. (2009). Création de modèles de régression pour prédire les consommations d'énergie des bâtiments à partir de simulations numériques. *CIFQ*, 6 pages. https://hal.archives-ouvertes.fr/hal-00985331

Catalina, T., Virgone, J., & Iordache, V. (2011). Study on the impact of the building form on the energy consumption. *Proceedings of building simulation*, 1726-1729.

Celani, G., & Vaz, C. E. V. (2012). CAD scripting and visual programming languages for implementing computational design concepts : A comparison from a pedagogical point of view. *International Journal of Architectural Computing*, *10*(1), 121-137.

Ceranic, D. B., & Nguyen, T. (2018, juin 25). *Shape grammar and kinetic façade shading systems : A novel approach to climate adaptive building design with a real time performance evaluation*. Geomapplica International Conference.

Ceren. (2018). Données et études statistiques pour le changement climatique, l'énergie, l'environnement, le logement et les transports. https://www.statistiques.developpement-durable.gouv.fr/consommation-denergie-par-usage-du-tertiaire

Cerf, M. (2018). *Techniques d'optimisation*.

Cerf, M. (2022). 2. Optimisation sans gradient. In *2. Optimisation sans gradient* (p. 85-188). EDP Sciences. https://doi.org/10.1051/978-2-7598-2769-5.c004

Chaillou, S. (2020). Archigan : Artificial intelligence x architecture. In *Architectural intelligence* (p. 117-127). Springer.

Chaillou, S. (2021). *L'intelligence artificielle au service de l'architecture*. Librairie Eyrolles. https://www.eyrolles.com/BTP/Livre/l-intelligence-artificielle-au-service-de-l-architecture-9782281144857/

Chatzikonstantinou, I., Turrin, M., Cubukcuoglu, C., Kirimtat, A., & Sariyildiz, S. (2019). A Comprehensive Optimization Approach for Modular Facades : The Case of PULSE Sunshading. *International Journal of Design Sciences & Technology*, *23*(2), Article 2.

Chegari, B., Tabaa, M., Simeu, E., Moutaouakkil, F., & Medromi, H. (2022). An optimal surrogate-model-based approach to support comfortable and nearly zero energy buildings design. *Energy*, *248*, 123584. https://doi.org/10.1016/j.energy.2022.123584

Chen, K. W., Janssen, P., & Schlueter, A. (2018). Multi-objective optimisation of building form, envelope and cooling system for improved building energy performance. *Automation in construction*, *94*, 449-457.

Chen, T., Lau, S. Y., Zhang, J., Xue, X., Lau, S. K., & Khoo, Y. S. (2019). A design-driven approach to integrate high-performance photovoltaics devices on the building façade. *IOP Conference Series: Earth and Environmental Science*, *294*(1), 012030. https://doi.org/10.1088/1755-1315/294/1/012030

Chen, T., Tang, K., Chen, G., & Yao, X. (2012). A large population size can be unhelpful in evolutionary algorithms. *Theoretical Computer Science*, *436*, 54-70. https://doi.org/10.1016/j.tcs.2011.02.016

Chronis, A., Dubor, A., Cabay, E., & Roudsari, M. S. (2017). Integration of CFD in computational design. *Proceedings of eCAADe 2017*, 601-610.

Ciardiello, A., Rosso, F., Dell'Olmo, J., Ciancio, V., Ferrero, M., & Salata, F. (2020). Multi-objective approach to the optimization of shape and envelope in building energy design. *Applied energy*, *280*, 115984.

Cichocka, J. M. (2017). Optimization in the Architectural Practice—An International Survey. *P. Janssen, P. Loh, A. Raonic, M. A. Schnabel (eds.), Protocols, Flows, and Glitches - Proceedings of the 22nd CAADRIA Conference, Xi'an Jiaotong-Liverpool University, Suzhou, China, 5-8 April 2017, pp. 387-396*. http://papers.cumincad.org/cgi-bin/works/paper/caadria2017_155

Cichocka, J. M., Migalska, A., Browne, W. N., & Rodriguez, E. (2017). SILVEREYE – The Implementation of Particle Swarm Optimization Algorithm in a Design Optimization Tool. In G. Çağdaş, M. Özkar, L. F. Gül, & E. Gürer (Éds.), *Computer-Aided Architectural Design. Future Trajectories* (p. 151-169). Springer. https://doi.org/10.1007/978-981-10-5197-5_9

Clerc, M. (2005). L'optimisation par essaim particulaire. *Hermès-Lavoisier, février*.

Coello Coello, C. A. (2002). Theoretical and numerical constraint-handling techniques used with evolutionary algorithms : A survey of the state of the art. *Computer Methods in Applied Mechanics and Engineering*, *191*(11), 1245-1287. https://doi.org/10.1016/S0045-7825(01)00323-1

Coello Coello, C. A. (2016). Constraint-handling techniques used with evolutionary algorithms. *Proceedings of the 2016 on Genetic and Evolutionary Computation Conference Companion*, 563-587.

Confort thermique : Généralité. (2007, septembre 25). *Energie Plus Le Site*. https://energieplus-lesite.be/theories/confort11/le-confort-thermique-d1/

Conti, Z. X. S. (2015). Multi-objective Optimisation of Building Geometry for Energy Consumption and View Quality. *Martens, B, Wurzer, G, Grasl T, Lorenz, WE and Schaffranek, R (eds.), Real Time - Proceedings of the 33rd eCAADe Conference - Volume 1, Vienna University of Technology, Vienna, Austria, 16-18 September 2015, pp. 287-294*. http://papers.cumincad.org/cgi-bin/works/2015%20+dave=2:/Show?ecaade2015_17

Costa-Carrapiço, I., Raslan, R., & González, J. N. (2020). A systematic review of genetic algorithm-based multi-objective optimisation for building retrofitting strategies towards energy efficiency. *Energy and Buildings*, *210*, 109690. https://doi.org/10.1016/j.enbuild.2019.109690

Courgey & Oliva. (2006). *La conception bioclimatique : Des maisons économes et confortables en neuf et en réhabilitation*. Terre vivante.

Cubukcuoglu, C., Ekici, B., Tasgetiren, M. F., & Sariyildiz, S. (2019). OPTIMUS : Self-Adaptive Differential Evolution with Ensemble of Mutation Strategies for Grasshopper Algorithmic Modeling. *Algorithms*, *12*(7), Article 7. https://doi.org/10.3390/a12070141

Darwin, C. (2004). *On the origin of species, 1859*. Routledge.

Das, S., Day, C., Hauck, J., & Davis, D. (2016). Space Plan Generator : Rapid Generationn & Evaluation of Floor Plan Design Options to Inform Decision Making. *ACADIA // 2016: POSTHUMAN FRONTIERS: Data, Designers, and Cognitive Machines [Proceedings of the 36th Annual Conference of the Association for Computer Aided Design in*

Architecture (ACADIA) ISBN 978-0-692-77095-5] Ann Arbor 27-29 October, 2016, pp. 106-115. http://papers.cumincad.org/cgi-bin/works/Show?acadia16_106

Davidson, S. (2018, décembre 12). *Writing Custom Eto forms in Python*. Www.Rhino3d.Com. https://developer.rhino3d.com/guides/rhinopython/eto-forms-python/

De Boissieu, A., & Guéna, F. (2012). Grasshopper et la programmation sur Rhinoceros 4DNArchi. *DNArchi*. http://dnarchi.fr/pedagogies/grasshopper-et-la-programmation-sur-rhinoceros-4-une-introduction/

De Luminae. (s. d.). *Facteur de lumière du jour*. Consulté 9 février 2023, à l'adresse https://www.guide-clea.fr/clea_projet/flj/

Deb, K. (2000). An efficient constraint handling method for genetic algorithms. *Computer Methods in Applied Mechanics and Engineering*, *186*(2), 311-338. https://doi.org/10.1016/S0045-7825(99)00389-8

Deb, K., & Agrawal, R. B. (1995). Simulated binary crossover for continuous search space. *Complex systems*, *9*(2), 115-148.

Deb, K., Pratap, A., Agarwal, S., & Meyarivan, T. (2002). A fast and elitist multiobjective genetic algorithm : NSGA-II. *IEEE transactions on evolutionary computation*, *6*(2), 182-197.

Dissaux, T. (2018). *Optimisation en conception architecturale : Les alternatives aux algorithmes génétiques*.

Du, T., Jansen, S., Turrin, M., & van den Dobbelsteen, A. (2020). Effects of Architectural Space Layouts on Energy Performance : A Review. *Sustainability*, *12*(5), 1829. https://doi.org/10.3390/su12051829

Du, T., Turrin, M., Jansen, S., Van Den Dobbelsteen, A., & Biloria, N. (2018). A review on automatic generation of architectural space layouts with energy performance

optimization. *Proceedings of the International Conference On Building Energy & Environment, COBEE*.

Du, T., Turrin, M., Jansen, S., van den Dobbelsteen, A., & Fang, J. (2020). Gaps and requirements for automatic generation of space layouts with optimised energy performance. *Automation in Construction, 116,* 103132. https://doi.org/10.1016/j.autcon.2020.103132

Duclos-Prévet, C., Guéna, F., & Efron, M. (2022a). Algorithme génétique ou automate cellulaire : Le cas d'une optimisation multicritère sous contraintes pour la conception d'une enveloppe. *SHS Web of Conferences, 147*.

Duclos-Prévet, C., Guéna, F., & Efron, M. (2022b). Constraint handling methods for a generative envelope design using genetic algorithms : The case of a highly constrained problem. *International Journal of Architectural Computing, 20*(3), 587-609.

Duclos-Prévet, C., Guéna, F., & Efron, M. (2021). Constrained Multi-Criteria Optimization for Integrated Design in Professional Practice. *Gomez, P and Braida, F (eds.), Designing Possibilities - Proceedings of the XXV International Conference of the Ibero-American Society of Digital Graphics (SIGraDi 2021), Online, 8 - 12 November 2021, pp. 29–40*. http://papers.cumincad.org/cgi-bin/works/paper/sigradi2021_56

Duering, S., Chronic, A., & Koenig, R. (2020). Optimizing urban systems : Integrated optimization of spatial configurations. *Proceedings of the 11th Annual Symposium on Simulation for Architecture and Urban Design*, 1-7.

Edwards, L., & Torcellini, P. (2002). *Literature review of the effects of natural light on building occupants*. 59.

Ehrgott, M. (2005). *Multicriteria Optimization*. Springer Science & Business Media.

Ekici, B., Cubukcuoglu, C., Turrin, M., & Sariyildiz, I. S. (2019). Performative computational architecture using swarm and evolutionary optimisation : A review. *Building and Environment*, *147*, 356-371.

El Sheikh, M., & Gerber, D. (2011). Building Skin Intelligence : A parametric and algorithmic tool for daylighting performance design integration. *ACADIA 11: Integration through Computation [Proceedings of the 31st Annual Conference of the Association for Computer Aided Design in Architecture (ACADIA)] [ISBN 978-1-6136-4595-6] Banff (Alberta) 13-16 October, 2011, pp. 170-177*. http://papers.cumincad.org/cgi-bin/works/Show&_id=caadria2010_016/paper/acadia11_170

Elghazi, Y., Wagdy Mohamed Ibrahim, A., Mohamed, S., & Hassan, A. (2014). Daylighting driven design : Optimizing kaleidocycle facade for hot arid climate. In D. Mueller & C. van Treeck (Éds.), *Human-centred building(s) : Proceedings of BauSIM 2014 : 5th German-Austrian Conference of IBPSA* (p. 314-321). International Building Performance Simulation Association. http://www.ibpsa.org/proceedings/bausimPapers/2014/p1155_final.pdf

Eltaweel, A., & Yuehong, S. U. (2017). Parametric design and daylighting : A literature review. *Renewable and Sustainable Energy Reviews*, *73*, 1086-1103.

Emmerich, M. T. M., & Deutz, A. H. (2018). A tutorial on multiobjective optimization : Fundamentals and evolutionary methods. *Natural Computing*, *17*(3), 585-609. https://doi.org/10.1007/s11047-018-9685-y

Enescu, D. (2017). A review of thermal comfort models and indicators for indoor environments. *Renewable and Sustainable Energy Reviews*, *79*, 1353-1379. https://doi.org/10.1016/j.rser.2017.05.175

Ercan, B., & Elias-Ozkan, S. T. (2015). Performance-based parametric design explorations : A method for generating appropriate building components. *Design Studies*, *38*, 33-53.

Evins, R. (2013). A review of computational optimisation methods applied to sustainable building design. *Renewable and Sustainable Energy Reviews*, *22*, 230-245. https://doi.org/10.1016/j.rser.2013.02.004

Fathy, A. G., El-Sayad, Z., Saadallah, D., & Bakr, A. F. (2021). BIMPO : A generative parametric technique for building envelope design. *Building Information Modelling (BIM) in Design, Construction and Operations IV*, *205*, 127.

Fathy, F., & Fareed, H. A. (2017). Performance-driven Façade Design Using an Evolutionary Multi-Objective Optimization Approach. *International Conference for Sustainable Design of the Built Environment-SDBE London*, 217.

Fathy, F., Mansour, Y., Sabry, H., Abdelmohsen, S., & Wagdy, A. (2015). *Cellular Automata for Efficient Daylighting Performance : Optimized Façade Treatment*. https://doi.org/10.26868/25222708.2015.2512

Fauconnier, R. (1992). L'action de l'humidité de l'air sur la santé dans les bâtiments tertiaires. *Chauffage, ventilation, conditionnement*, *68*(10), 57-62.

Fedorova, S. (2021). Generative adversarial networks for urban block design. *SimAUD 2021: A Symposium on Simulation for Architecture and Urban Design*.

Ferber, J. (1997). Les systèmes multi-agents : Un aperçu général. *Techniques et sciences informatiques*, *16*(8).

Food4Rhino. (s. d.). [Text]. Food4Rhino. Consulté 1 mars 2023, à l'adresse https://www.food4rhino.com/en

Gagne, J., & Andersen, M. (2012). A generative facade design method based on daylighting performance goals. *Journal of Building Performance Simulation*, *5*(3), 141-154.

Galasiu, A. D., & Reinhart, C. F. (2008). Current daylighting design practice : A survey. *Building Research & Information*, *36*(2), 159-174. https://doi.org/10.1080/09613210701549748

Gauzin-Müller, D. (2001). *L'architecture écologique* (Le Moniteur).

Gerber, D. J., & Lin, S.-H. E. (2014). Designing in complexity : Simulation, integration, and multidisciplinary design optimization for architecture. *SIMULATION*, *90*(8), 936-959. https://doi.org/10.1177/0037549713482027

Gerber, D. J., Pantazis, E., & Wang, A. (2017). A multi-agent approach for performance based architecture : Design exploring geometry, user, and environmental agencies in façades. *Automation in construction*, *76*, 45-58.

Ghorbania, M., & Shariatpour, F. (2021). *Procedural Modeling as an Analytical Tool for 3D Survey in Urban Design Assessment*. http://ijaup.iust.ac.ir/article-1-502-fa.pdf

Gokmen, S. (2013). A Morphogenetic Approach for Performative Building Envelope Systems Using Leaf Venetian Patterns. *Stouffs, Rudi and Sariyildiz, Sevil (eds.), Computation and Performance – Proceedings of the 31st eCAADe Conference – Volume 1, Faculty of Architecture, Delft University of Technology, Delft, The Netherlands, 18-20 September 2013, pp. 497-506.* http://papers.cumincad.org/cgi-bin/works/Show&_id=caadria2010_044/paper/ecaade2013_167

Gotshall, S., & Rylander, B. (2002). Optimal population size and the genetic algorithm. *Population*, *100*(400), 900.

Granadeiro, V., Pina, L., Duarte, J. P., Correia, J. R., & Leal, V. M. S. (2013). A general indirect representation for optimization of generative design systems by genetic algorithms : Application to a shape grammar-based design system. *Automation in Construction*, *35*, 374-382. https://doi.org/10.1016/j.autcon.2013.05.012

Grobman, Y. J. ; R. (2011). Digital Form Finding : Generative use of simulation processes by architects in the early stages of the design process. *RESPECTING FRAGILE PLACES [29th eCAADe Conference Proceedings / ISBN 978-9-4912070-1-3], University of*

Ljubljana, Faculty of Architecture (Slovenia) 21-24 September 2011, pp.107-115. http://papers.cumincad.org/cgi-bin/works/2015%20+dave=2:/Show?ecaade2011_018

Groenewolt, A., Schwinn, T., Nguyen, L., & Menges, A. (2018). An interactive agent-based framework for materialization-informed architectural design. *Swarm Intelligence*, *12*(2), 155-186. https://doi.org/10.1007/s11721-017-0151-8

Gunantara, N. (2018). A review of multi-objective optimization : Methods and its applications. *Cogent Engineering*, *5*(1), 1502242. https://doi.org/10.1080/23311916.2018.1502242

Gunn, R., & Kinzer, G. D. (1949). The terminal velocity of fall for water droplets in stagnant air. *Journal of Atmospheric Sciences*, *6*(4), 243-248.

Guo, W., Liu, X., & Yuan, X. (2015). Study on Natural Ventilation Design Optimization Based on CFD Simulation for Green Buildings. *Procedia Engineering*, *121*, 573-581. https://doi.org/10.1016/j.proeng.2015.08.1036

Guo, Z., & Li, B. (2017). Evolutionary approach for spatial architecture layout design enhanced by an agent-based topology finding system. *Frontiers of Architectural Research*, *6*(1), 53-62.

Gursel Dino, I. (2012). Creative design exploration by parametric generative systems in architecture. *METU Journal of the Faculty of Architecture*, *29*, 207-224. https://doi.org/10.4305/METU.JFA.2012.1.12

Haakonsen, S. M., Rønnquist, A., & Labonnote, N. (2023). Fifty years of shape grammars : A systematic mapping of its application in engineering and architecture. *International Journal of Architectural Computing*, *21*(1), 5-22. https://doi.org/10.1177/14780771221089882

Hamdan, M. (2012). The distribution index in polynomial mutation for evolutionary multiobjective optimisation algorithms : An experimental study. *International Conference on Electronics Computer Technology (IEEE, Kanyakumari, India, 2012)*.

Hamdy, M., Nguyen, A.-T., & Hensen, J. L. M. (2016). A performance comparison of multi-objective optimization algorithms for solving nearly-zero-energy-building design problems. *Energy and Buildings*, *121*, 57-71. https://doi.org/10.1016/j.enbuild.2016.03.035

Harding, J. E., & Shepherd, P. (2017). Meta-Parametric Design. *Design Studies*, *52*, 73-95. https://doi.org/10.1016/j.destud.2016.09.005

Hassanat, A., Almohammadi, K., Alkafaween, E., Abunawas, E., Hammouri, A., & Prasath, V. B. S. (2019). Choosing Mutation and Crossover Ratios for Genetic Algorithms—A Review with a New Dynamic Approach. *Information*, *10*(12), Article 12. https://doi.org/10.3390/info10120390

Haymaker, J., Bernal, M., Marshall, M. T., Okhoya, V., Szilasi, A., Rezaee, R., Chen, C., Salveson, A., Brechtel, J., Deckinga, L., Hasan, H., Ewing, P., & Welle, B. (2018). Design space construction : A framework to support collaborative, parametric decision making. *Journal of Information Technology in Construction (ITcon)*, *23*(8), 157-178.

Hellinga, H., & Hordijk, T. (2014). The D&V analysis method : A method for the analysis of daylight access and view quality. *Building and Environment*, *79*, 101-114. https://doi.org/10.1016/j.buildenv.2014.04.032

Henriques, G. C., Duarte, J. P., & Leal, V. (2012). Strategies to control daylight in a responsive skylight system. *Automation in Construction*, *28*, 91-105. https://doi.org/10.1016/j.autcon.2012.06.002

Hochscheid, E., & Halin, G. (2020). Baromètre BIM : Une enquête sur l'adoption du BIM dans les agences d'architecture en France. *SHS Web of Conferences*, *82*, 02004. https://doi.org/10.1051/shsconf/20208202004

Hollberg, A., & Ruth, J. (2016). LCA in architectural design—A parametric approach. *The International Journal of Life Cycle Assessment*, *21*(7), 943-960. https://doi.org/10.1007/s11367-016-1065-1

Hosseini, S. M., Mohammadi, M., & Guerra-Santin, O. (2019). Interactive kinetic façade : Improving visual comfort based on dynamic daylight and occupant's positions by 2D and 3D shape changes. *Building and Environment*, *165*, 106396. https://doi.org/10.1016/j.buildenv.2019.106396

Hoyet, N. (2020). *Matériaux et architecture durable* (Dunod). https://www.dunod.com/sciences-techniques/materiaux-et-architecture-durable-fabrication-et-transformations-proprietes-1

Hu, Y., Peng, Y., Gao, Z., & Xu, F. (2023). Application of CFD plug-ins integrated into urban and building design platforms for performance simulations : A literature review. *Frontiers of Architectural Research*, *12*, 148-174. https://doi.org/10.1016/j.foar.2022.06.005

Huang, C., Zhang, G., Yin, M., & Yao, J. (2022). *Energy-driven Intelligent Generative Urban Design, Based on Deep Reinforcement Learning Method With a Nested Deep Q-R Network*. 233-242. https://doi.org/10.52842/conf.caadria.2022.1.233

Huang, Y., & Niu, J. (2016). Optimal building envelope design based on simulated performance : History, current status and new potentials. *Energy and Buildings*, *117*, 387-398. https://doi.org/10.1016/j.enbuild.2015.09.025

Huang, Y.-S., Chang, W.-S., & Shih, S.-G. (2015). Building Massing Optimization in the Conceptual Design Phase. *Computer-Aided Design and Applications*, *12*(3), 344-354. https://doi.org/10.1080/16864360.2014.981465

Hudson, R., Shepherd, P., & Hines, D. (2011). Aviva Stadium : A Case Study in Integrated Parametric Design. *International Journal of Architectural Computing*, *9*(2), 187-203. https://doi.org/10.1260/1478-0771.9.2.187

Ilozor, B. D., & Kelly, D. J. (2012). Building information modeling and integrated project delivery in the commercial construction industry : A conceptual study. *Journal of engineering, project, and production management*, *2*(1), 23-36.

Iqbal, M. (2012). *An introduction to solar radiation*. Elsevier.

Isola, P., Zhu, J.-Y., Zhou, T., & Efros, A. A. (2017). Image-to-Image Translation with Conditional Adversarial Networks. *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 5967-5976. https://doi.org/10.1109/CVPR.2017.632

Izard, J.-L. (1985). *Rôle de l'architecte en conception thermique des bâtiments. Etude comparative de la sensibilité des paramètres architecturaux* (Research Report 343/86). Ecole nationale supérieure d'architecture de Marseille-Luminy / Groupe ABC - Laboratoire Architecture Bioclimatique et Constructions exposées aux risques naturels. https://hal.archives-ouvertes.fr/hal-01896660

Jalaeian-F, M. (2012, juillet 1). *Augmented Downhill Simplex a Modified Heuristic Optimization Method*. https://www.semanticscholar.org/paper/Augmented-Downhill-Simplex-a-Modified-Heuristic-Jalaeian-F./1e69e0fd736b54798cb53987a01aacd693bc03b2

Jasak, H. (2009). OpenFOAM : Open source CFD in research and industry. *International Journal of Naval Architecture and Ocean Engineering*, *1*(2), 89-94.

Jayathissa, P., Caranovic, S., Hofer, J., Nagy, Z., & Schlueter, A. (2018). Performative design environment for kinetic photovoltaic architecture. *Automation in Construction, 93*, 339-347. https://doi.org/10.1016/j.autcon.2018.05.013

Jia, M. (2021). *DAYLIGHT PREDICTION USING GAN : GENERAL WORKFLOW, TOOL DEVELOPMENT AND CASE STUDY ON MANHATTAN, NEW YORK*. https://doi.org/10.7298/xz3w-yc87

Kabošová, L., Katunskỳ, D., & Kmet, S. (2020). Wind-based parametric design in the changing climate. *Applied Sciences*, *10*(23), 8603.

Kämpf, J. H., Montavon, M., Bunyesc, J., Bolliger, R., & Robinson, D. (2010). Optimisation of buildings' solar irradiation availability. *Solar energy*, *84*(4), 596-603.

Kastner, P., & Dogan, T. (2022). Eddy3D : A toolkit for decoupled outdoor thermal comfort simulations in urban areas. *Building and Environment*, *212*, 108639.

Kennedy, J., & Eberhart, R. (1995). Particle swarm optimization. *Proceedings of ICNN'95-international conference on neural networks*, *4*, 1942-1948.

Kheiri, F. (2018). A review on optimization methods applied in energy-efficient building geometry and envelope design. *Renewable and Sustainable Energy Reviews*, *92*, 897-920.

Kim, J. (2015). Adaptive façade design for the daylighting performance in an office building : The investigation of an opening design strategy with cellular automata. *International Journal of Low-Carbon Technologies*, *10*(3), 313-320. https://doi.org/10.1093/ijlct/ctt015

Knight, T. (2003). Computing with Emergence. *Environment and Planning B: Planning and Design*, *30*(1), 125-155. https://doi.org/10.1068/b12914

Knight, T., & Stiny, G. (2001). Classical and non-classical computation. *Architectural Research Quarterly*, *5*(4), 355-372. https://doi.org/10.1017/S1359135502001410

Koenig, R., Miao, Y., Aichinger, A., Knecht, K., & Konieva, K. (2020). Integrating urban analysis, generative design, and evolutionary optimization for solving urban design

problems. *Environment and Planning B: Urban Analytics and City Science*, *47*(6), 997-1013. https://doi.org/10.1177/2399808319894986

Koenig, R. T. (2013). Graphical Smalltalk with My Optimization System for Urban Planning Tasks. *Stouffs, Rudi and Sariyildiz, Sevil (eds.), Computation and Performance – Proceedings of the 31st eCAADe Conference – Volume 2, Faculty of Architecture, Delft University of Technology, Delft, The Netherlands, 18-20 September 2013, pp. 195-203*. http://papers.cumincad.org/cgi-bin/works/2015%20+dave=2:/Show?ecaade2013_197

Kolarevic, B., & Malkawi, A. (2005). *Performative Architecture : Beyond Instrumentality*. Spon Press.

Kotsopoulos, S. D., Casalegno, F., Carra, G., Graybil, W., & Hsiung, B. (2012). A visual—Performative language of façade patterns for the connected sustainable home. *Proceedings of the 2012 Symposium on Simulation for Architecture and Urban Design*, 1-12.

Kramer, O., Echeverría Ciaurri, D., & Koziel, S. (2011). Derivative-Free Optimization. In *Studies in Computational Intelligence* (Vol. 356, p. 61-83). https://doi.org/10.1007/978-3-642-20859-1_4

Kunkle, D. (2005). A summary and comparison of MOEA algorithms. *Northeast. Univ. Boston Mass*.

Ladybug Tools. (2016, octobre 8). [Text]. Food4Rhino. https://www.food4rhino.com/en/app/ladybug-tools

Latha, H., Patil, S., & Kini, P. G. (2022). Influence of architectural space layout and building perimeter on the energy performance of buildings : A systematic literature review. *International Journal of Energy and Environmental Engineering*. https://doi.org/10.1007/s40095-022-00522-4

Lavoye, F., & Thellier, F. (2008). *Le confort thermique dans les bâtiments*.

Lawson, B. (2006). How Designers Think – The Design Process Demystified. *University Press, Cambridge*.

Le Berre, D., Lonca, E., Marquis, P., & Parrain, A. (2012). *Optimisation multicritère pour la gestion de dépendances logicielles : Utilisation de la norme de Tchebycheff*.

Lecheval, V., Sire, C., & Theraulaz, G. (s. d.). *La danse organisée des bancs de poissons*.

L'éco-conception des produits. (2019, février 7). Ministères Écologie Énergie Territoires. https://www.ecologie.gouv.fr/leco-conception-des-produits

Li, S., Liu, L., & Peng, C. (2020). A review of performance-oriented architectural design and optimization in the context of sustainability : Dividends and challenges. *Sustainability*, *12*(4), 1427.

Li, W., & Samuelson, H. (2020). A new method for visualizing and evaluating views in architectural design. *Developments in the Built Environment*, *1*, 100005. https://doi.org/10.1016/j.dibe.2020.100005

Li, Z., Chen, H., Lin, B., & Zhu, Y. (2018). Fast bidirectional building performance optimization at the early design stage. *Building Simulation*, *11*(4), 647-661. https://doi.org/10.1007/s12273-018-0432-1

Lindenmayer, A. (1968). Mathematical models for cellular interactions in development I. Filaments with one-sided inputs. *Journal of theoretical biology*, *18*(3), 280-299.

List of lighting design software. (2022). In *Wikipedia*. https://en.wikipedia.org/w/index.php?title=List_of_lighting_design_software&oldid=1085943544

Lu, W., Fung, A., Peng, Y., Liang, C., & Rowlinson, S. (2015). Demystifying construction project time–effort distribution curves : BIM and non-BIM comparison. *Journal of management in engineering*, *31*(6), 04015010.

Ma, L. (2015a). *Invention architecturale et algorithmes non-lineaires* [These de doctorat, Versailles-St Quentin en Yvelines]. https://www.theses.fr/2015VERS023S

Ma, L. (2015b). *Invention architecturale et algorithmes non-linéaires* [PhD Thesis]. Versailles-St Quentin en Yvelines.

Macal, C. M. (2016). Everything you need to know about agent-based modelling and simulation. *Journal of Simulation*, *10*(2), 144-156.

Macal, C. M., & North, M. J. (2005). Tutorial on agent-based modeling and simulation. *Proceedings of the Winter Simulation Conference, 2005.*, 14 pp.-. https://doi.org/10.1109/WSC.2005.1574234

Mahdavi, A., Spasojevic, B., & Brunner, K. A. (2005). Elements of a simulation-assisted daylight-responsive illumination systems control in buildings. *Building Simulation*, 15-18.

Marin, P., Bignon, J.-C., & Lequay, H. (2008). *Generative exploration of architectural envelope responding to solar passive qualities. 26*. https://shs.hal.science/halshs-00348544

Marsault, X. (2018). *Écoconception générative : Phase amont du projet d'architecture*. ISTE Group.

Marsault, X., & Torres, F. (2019). An interactive and generative eco-design tool for architects in the sketch phase. *Journal of Physics: Conference Series*, *1343*(1), 012136. https://doi.org/10.1088/1742-6596/1343/1/012136

Mathias, M., Martinovic, A., Weissenberg, J., & Gool, L. V. (2011). Procedural 3D Building Reconstruction Using Shape Grammars and Detectors. *Visualization and Transmission 2011 International Conference on 3D Imaging, Modeling, Processing*, 304-311. https://doi.org/10.1109/3DIMPVT.2011.45

Matsson, J. E. (2022). *An Introduction to ANSYS Fluent 2022*. SDC Publications.

McCormack, J., Dorin, A., & Innocent, T. (2004). Generative Design : A Paradigm for Design Research. *DRS Biennial Conference Series*. https://dl.designresearchsociety.org/drs-conference-papers/drs2004/researchpapers/171

Mei, Z., Pan, Y., Cheng, J., & Garcia del Castillo Lopez, J. L. (2021). *Cross-Scale and Density-Driven City Generator—Parametric assistance to designers in prototyping stage*. 563-570. https://doi.org/10.52842/conf.ecaade.2021.1.563

Menges, A. (2012). Biomimetic design processes in architecture : Morphogenetic and evolutionary computational design. *Bioinspiration & Biomimetics*, *7*(1), 015003. https://doi.org/10.1088/1748-3182/7/1/015003

Metropolis, N., Rosenbluth, A., Rosenbluth, M., Teller, A., & Teller, E. (1953). Simulated annealing. *Journal of Chemical Physics*, *21*(161-162), 1087-1092.

Miao, Y., Koenig, R., & Knecht, K. (2020). The Development of Optimization Methods in Generative Urban Design : A Review. *2020 Proceedings of the Symposium on Simulation for Architecture and Urban Design*, 247-254. https://www.research-collection.ethz.ch/handle/20.500.11850/408360

Michalewicz, Z., & Schoenauer, M. (1996). Evolutionary algorithms for constrained parameter optimization problems. *Evolutionary computation*, *4*(1), 1-32.

Miettinen, K., & Mäkelä, M. M. (2002). On scalarizing functions in multiobjective optimization. *OR Spectrum*, *24*(2), 193-213. https://doi.org/10.1007/s00291-001-0092-9

Miller, B. L., & Goldberg, D. E. (1995). Genetic algorithms, tournament selection, and the effects of noise. *Complex systems*, *9*(3), 193-212.

Mirjalili, S. (2019). Genetic algorithm. In *Evolutionary algorithms and neural networks* (p. 43-55). Springer.

Mohamadin, M. F., Abouaiana, A. A., & Wagih, H. H. (2016). Parametric Islamic Geometric Pattern for Efficient Daylight and Energy Performance—Façade retrofit of educational space in hot arid climate. *Parametricism Vs. Materialism: Evolution of Digital Technologies for Development [8th ASCAAD Conference Proceedings ISBN 978-0-9955691-0-2] London (United Kingdom) 7-8 November 2016, pp. 227-236*. http://papers.cumincad.org/cgi-bin/works/paper/ascaad2016_025

Mokhtar, S., Sojka, A., & Davila, C. C. (2020). Conditional generative adversarial networks for pedestrian wind flow approximation. *Proceedings of the 11th Annual Symposium on Simulation for Architecture and Urban Design*, 1-8.

Muehlbauer, M., Burry, J., & Song, A. (2017). Automated shape design by grammatical evolution. *Computational Intelligence in Music, Sound, Art and Design: 6th International Conference, EvoMUSART 2017, Amsterdam, The Netherlands, April 19–21, 2017, Proceedings 6*, 217-229.

Mukkavaara, J., & Sandberg, M. (2020). Architectural Design Exploration Using Generative Design : Framework Development and Case Study of a Residential Block. *Buildings*, *10*(11), Article 11. https://doi.org/10.3390/buildings10110201

Multiphysics, C. (1998). Introduction to comsol multiphysics®. *COMSOL Multiphysics, Burlington, MA, accessed Feb, 9*(2018), 32.

Nagy, D., Villaggi, L., & Benjamin, D. (2018). Generative urban design : Integrating financial and energy goals for automated neighborhood layout. *Proceedings of the Symposium on Simulation for Architecture and Urban Design*, 1-8.

Narangerel, A., Lee, J.-H., & Stouffs, R. (2016). Daylighting Based Parametric Design Exploration of 3D Facade Patterns. *Conference: Complexity and simplicity: Proceedings of the 34th eCAADe Conference*, 379-388.

Nault, É., Rey, E., & Andersen, M. (Éds.). (2016). A Multi-Criteria Decision-Support Workflow for Early-Stage Neighborhood Design based on Predicted Solar Performance. *Proceedings of PLEA 2016, 32nd international Conference on Passive and Low Energy Architecture*.

Negendahl, K., & Nielsen, T. R. (2015). Building energy optimization in the early design stages : A simplified method. *Energy and Buildings*, *105*, 88-99. https://doi.org/10.1016/j.enbuild.2015.06.087

Nelder, J. A., & Mead, R. (1965). A Simplex Method for Function Minimization. *The Computer Journal*, *7*(4), 308-313. https://doi.org/10.1093/comjnl/7.4.308

Newton, D. (2019). Generative Deep Learning in Architectural Design. *Technology|Architecture + Design*, *3*(2), 176-189. https://doi.org/10.1080/24751448.2019.1640536

Nguyen, A.-T., Reiter, S., & Rigo, P. (2014). A review on simulation-based optimization methods applied to building performance analysis. *Applied Energy*, *113*, 1043-1058. https://doi.org/10.1016/j.apenergy.2013.08.061

Nugroho, F., & Al-Sanjary, O. (2018). A Review of Simulation Urban Growth Model. *International Journal of Engineering and technology*, *7*. https://doi.org/10.14419/ijet.v7i4.11.20681

Orvosh, D., & Davis, L. (1993). *Shall We Repair ? Genetic Algorithms Combinatorial Optimization and Feasibility Constraints*. Proceedings of the 5th International Conference on Genetic Algorithms. https://dl.acm.org/doi/abs/10.5555/645513.657753

Oxman, R. (2006). Theory and design in the first digital age. *Design Studies*, *27*(3), 229-265. https://doi.org/10.1016/j.destud.2005.11.002

Oxman, R. (2009). Performative design : A performance-based model of digital architectural design. *Environment and Planning B: Planning and Design*, *36*(6), 1026-1037.

Pantazis, E., & Gerber, D. J. (2020). Behavioral Form Finding : A Multi Agent Systems Framework for Environmental Aware Form Finding of Shell Structures. In C. Gengnagel, O. Baverel, J. Burry, M. Ramsgaard Thomsen, & S. Weinzierl (Éds.), *Impact : Design With All Senses* (p. 146-158). Springer International Publishing. https://doi.org/10.1007/978-3-030-29829-6_12

Papanek, V. (1972). *Design for the Real World : Human Ecological and Social Change*.

Parascho, S. B. (2013). Design Tools for Integrative Planning. *Stouffs, Rudi and Sariyildiz, Sevil (eds.), Computation and Performance – Proceedings of the 31st eCAADe Conference – Volume 2, Faculty of Architecture, Delft University of Technology, Delft, The Netherlands, 18-20 September 2013, pp. 237-246*. http://papers.cumincad.org/cgi-bin/works/2015%20+dave=2:/Show?ecaade2013_230

Parish, Y. I. H., & Müller, P. (2001). Procedural modeling of cities. *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, 301-308. https://doi.org/10.1145/383259.383292

Peuportier, B. (2008). *Eco-conception des bâtiments et des quartiers* (p. 331). Les Presses MINES ParisTech. https://hal-mines-paristech.archives-ouvertes.fr/hal-00509347

Pham, D., & Karaboga, D. (2012). *Intelligent Optimisation Techniques : Genetic Algorithms, Tabu Search, Simulated Annealing and Neural Networks*. Springer Science & Business Media.

PMV Comfort Map. (s. d.). Consulté 16 février 2023, à l'adresse https://docs.ladybug.tools/hb-energy-primer/components/7_thermalmap/pmv_comfort_map

Portillo, M., & Dohr, J. H. (1994). Bridging process and structure through criteria. *Design Studies*, *15*(4), 403-416.

Punjabi, S., & Miranda, V. (2005). *DEVELOPMENT OF AN INTEGRATED BUILDING DESIGN INFORMATION INTERFACE*.

Qeisi, S. A., & Al-Alwan, H. (2021). Generative Urban Design Concepts and Methods : A Research Review. *IOP Conference Series: Materials Science and Engineering*, *1090*(1), 012085. https://doi.org/10.1088/1757-899X/1090/1/012085

Raanaas, R. K., Patil, G. G., & Hartig, T. (2012). Health benefits of a view of nature through the window : A quasi-experimental study of patients in a residential rehabilitation center. *Clinical Rehabilitation*, *26*(1), 21-32. https://doi.org/10.1177/0269215511412800

Rashid, M., & Zimring, C. (2008). A review of the empirical literature on the relationships between indoor environment and stress in health care and office settings : Problems and prospects of sharing evidence. *Environment and behavior*, *40*(2), 151-190.

Raynaud, D. (2004). Contrainte et liberté dans le travail de conception architecturale. *Revue française de sociologie*, *45*(2), 339-366. https://doi.org/10.3917/rfs.452.0339

RE2020. (s. d.). Ministères Écologie Énergie Territoires. Consulté 3 janvier 2023, à l'adresse https://www.ecologie.gouv.fr/reglementation-environnementale-re2020

Reinhart, C. (2019). Daylight performance predictions. In *Building performance simulation for design and operation* (p. 221-269). Routledge.

Reinhart, C. F., Morrison, M., & Dubrous, F. (2003). The lightSswitch wizard– reliable daylight simulations for initial design investigation. *Building Simulation*, *3*, 1093-1100.

Richardson, J. T., Palmer, M. R., Liepins, G. E., & Hilliard, M. (1989). Some guidelines for genetic algorithms with penalty functions. *Proceedings of the third international conference on Genetic algorithms*, 191-197.

Rizi, R. A., & Eltaweel, A. (2021). A user detective adaptive facade towards improving visual and thermal comfort. *Journal of Building Engineering*, *33*, 101554.

Robain, M., & AYACHE, R. (2013). *La puissance créatrice du collectif : La ligne singulière d'Architecture-Studio*. L'association des amis de l'école de Paris de management.

Roudsari, M. S., Pak, M., & Smith, A. (2013). Ladybug : A parametric environmental plugin for grasshopper to help designers create an environmentally-conscious design. *Proceedings of the 13th international IBPSA conference held in Lyon, France Aug*, 3128-3135.

Rutten, D. (2011, mars 4). Evolutionary Principles applied to Problem Solving. *I Eat Bugs For Breakfast*. https://ieatbugsforbreakfast.wordpress.com/2011/03/04/epatps01/

Sadineni, S. B., Madala, S., & Boehm, R. F. (2011). Passive building energy savings : A review of building envelope components. *Renewable and sustainable energy reviews*, *15*(8), 3617-3631.

Salcedo-Sanz, S. (2009). A survey of repair methods used as constraint handling techniques in evolutionary algorithms. *Computer science review*, *3*(3), 175-192.

Salomon, T., Mikolasek, R., & Peuportier, B. (2005). Outil de simulation thermique du bâtiment, COMFIE. *Journée thématique SFT-IBPSA*.

Sari, M., & Berawi, M. A. (2021). *The utilization of machine learning techniques in the building design stage : A qualitative review*.

Sarvani, V., & Kontovourkis, O. (2013). Parametric Design of a High-Rise Habitation Unit System through Lighting and Solar Energy Performances. *Journal of Sustainable Architecture and Civil Engineering*, *3*, 9-18. https://doi.org/10.5755/j01.sace.3.4.4723

Säwén, T., Magnusson, E., Kalagasidis, A. S., & Hollberg, A. (2022). Tool characterisation framework for parametric building LCA. *IOP Conference Series: Earth and Environmental Science*, *1078*(1), 012090. https://doi.org/10.1088/1755-1315/1078/1/012090

Schenk, D., & Amiri, A. (2022). Life cycle energy analysis of residential wooden buildings versus concrete and steel buildings : A review. *Frontiers in Built Environment*, *8*. https://www.frontiersin.org/articles/10.3389/fbuil.2022.975071

Schlierkamp-Voosen, D. (1993). Optimal interaction of mutation and crossover in the breeder genetic algorithm. *International Conference on Genetic Algorithms; Morgan Kaufmann Publishers Inc.: San Francisco, CA, USA.*

Schneider, C., Koltsova, A., & Schmitt, G. (2011, avril 3). *Components for parametric urban design in Grasshopper from street network to building geometry*. Spring Simulation Multiconference. https://www.semanticscholar.org/paper/Components-for-parametric-urban-design-in-from-to-Schneider-Koltsova/e60778347ffb55b8b42ba831ffb8f8f7269182a5

Schwartz, Y., Raslan, R., Korolija, I., & Mumovic, D. (2017). Integrated building performance optimisation : Coupling parametric thermal simulation optimisation and generative spatial design programming. *15th International Building Performance Simulation Association Conference, San Francisco, CA, Aug*, 7-9.

Sharaidin, K., Burry, J., & Salim, F. (2012). *Integration of digital simulation tools with parametric designs to evaluate kinetic façades for daylight performance*.

Shen, X. (2018). Environmental Parametric Multi-Objective Optimization for High Performance Facade Design. *T. Fukuda, W. Huang, P. Janssen, K. Crolla, S. Alhadidi (eds.), Learning, Adapting and Prototyping - Proceedings of the 23rd CAADRIA Conference - Volume 2, Tsinghua University, Beijing, China, 17-19 May 2018, pp. 103-112*. http://papers.cumincad.org/cgi-bin/works/Show&_id=caadria2010_003/paper/caadria2018_054

Shen, X., Singhvi, A., Mengual, A., Spastri, M., & Watson, V. (2018). Evaluating the Multi-Objective Optimization Methodology for Performance-Based Building Design in Professional Practice. *ASHRAE and IBPSA*, 646-653.

Shi, X., Tian, Z., Chen, W., Si, B., & Jin, X. (2016). A review on building energy efficient design optimization rom the perspective of architects. *Renewable and Sustainable Energy Reviews*, *65*, 872-884. https://doi.org/10.1016/j.rser.2016.07.050

Shi, Z., Fonseca, J. A., & Schlueter, A. (2017). A review of simulation-based urban form generation and optimization for energy-driven urban design. *Building and Environment*, *121*, 119-129. https://doi.org/10.1016/j.buildenv.2017.05.006

Showkatbakhsh, M., & Kaviani, S. (2021). Homeostatic generative design process : Emergence of the adaptive architectural form and skin to excessive solar radiation. *International Journal of Architectural Computing*, *19*(3), 315-330. https://doi.org/10.1177/1478077120951947

Singaravel, S., Suykens, J., & Geyer, P. (2018). Deep-learning neural-network architectures and methods : Using component-based models in building-design energy prediction. *Advanced Engineering Informatics*, *38*, 81-90. https://doi.org/10.1016/j.aei.2018.06.004

Singh, V., & Gu, N. (2012). Towards an integrated generative design framework. *Design Studies*, *33*(2), 185-207. https://doi.org/10.1016/j.destud.2011.06.001

Sleiman, H. A., Hempel, S., Traversari, R., & Bruinenberg, S. (2017). An assisted workflow for the early design of nearly zero emission healthcare buildings. *Energies*, *10*(7), 993.

Smith, J. E., & Fogarty, T. C. (1997). Operator and parameter adaptation in genetic algorithms. *Soft computing*, *1*, 81-87.

Solmaz, A. S. (2019). *A CRITICAL REVIEW ON BUILDING PERFORMANCE SIMULATION TOOLS*. *12*(2).

Sozer, H. (2010). Improving energy efficiency through the design of the building envelope. *Building and environment*, *45*(12), 2581-2593.

Spaeth, A. B. ; M. (2011). Performative Design for Spatial Acoustics : Concept for an evolutionary design algorithm based on acoustics as design driver. *RESPECTING FRAGILE PLACES [29th eCAADe Conference Proceedings / ISBN 978-9-4912070-1-3], University of Ljubljana, Faculty of Architecture (Slovenia) 21-24 September 2011, pp.461-468*. http://papers.cumincad.org/cgi-bin/works/2015%20+dave=2:/Show?ecaade2011_011

Stals, A., Elsen, C., & Jancart, S. (2022). Integration of parametric modeling tools in small architectural offices – between constraints and organizational strategies. *Architectural Engineering and Design Management*, *18*(5), 759-773. https://doi.org/10.1080/17452007.2022.2050347

Stevanović, S. (2013). Optimization of passive solar design strategies : A review. *Renewable and Sustainable Energy Reviews*, *25*, 177-196.

Stiny, G. (1980). Introduction to Shape and Shape Grammars. *Environment and Planning B, 7*(3), 343-351.

Stiny, G. (1994). Shape Rules : Closure, Continuity, and Emergence. *Environment and Planning B: Planning and Design*, *21*(7), S49-S78. https://doi.org/10.1068/b21S049

Su, Z., & Yan, W. (2015). A fast genetic algorithm for solving architectural design optimization problems. *Ai Edam*, *29*(4), 457-469.

Sundstrom, E., & Sundstrom, M. G. (1986). *Work places : The psychology of the physical environment in offices and factories*. CUP Archive.

Suyoto, W., Indraprastha, A., & Purbo, H. W. (2015). Parametric Approach as a Tool for Decision-making in Planning and Design Process. Case study : Office Tower in Kebayoran Lama. *Procedia - Social and Behavioral Sciences*, *184*, 328-337. https://doi.org/10.1016/j.sbspro.2015.05.098

Tabadkani, A., Valinejad Shoubi, M., Soflaei, F., & Banihashemi, S. (2019). Integrated parametric design of adaptive facades for user's visual comfort. *Automation in Construction*, *106*, 102857. https://doi.org/10.1016/j.autcon.2019.102857

Taleb, H., & Musleh, M. A. (2015). Applying urban parametric design optimisation processes to a hot climate : Case study of the UAE. *Sustainable Cities and Society*, *14*, 236-253.

Tan, K. C., Lee, T. H., & Khor, E. F. (2001). Evolutionary algorithms with dynamic population size and local exploration for multiobjective optimization. *IEEE Transactions on Evolutionary Computation*, *5*(6), 565-588. https://doi.org/10.1109/4235.974840

Tarabishy, S., Psarras, S., Kosicki, M., & Tsigkari, M. (2020). Deep learning surrogate models for spatial and visual connectivity. *International Journal of Architectural Computing*, *18*(1), 53-66.

Tepavčević, B., & Stojaković, V. (2012). Shape grammar in contemporary architectural theory and design. *Facta universitatis - series: Architecture and Civil Engineering*, *10*(2), 169-178.

Thornton Tomasetti. (2019, Aout). *Design Explorer 2*. Design Explorer. https://tt-acm.github.io/DesignExplorer/

Tomasowa, R., & Sjarifudin, F. U. (2017). Adaptive Façade : Variant-Finding using Shape Grammar. *IOP Conference Series: Earth and Environmental Science*, *109*, 012039. https://doi.org/10.1088/1755-1315/109/1/012039

Touloupaki, E., & Theodosiou, T. (2017). Performance Simulation Integrated in Parametric 3D Modeling as a Method for Early Stage Design Optimization—A Review. *Energies*, *10*(5), Article 5. https://doi.org/10.3390/en10050637

TRNLizard. (2017, avril 18). [Text]. Food4Rhino. https://www.food4rhino.com/en/app/trnlizard

Tutoriel pix2pix. (2023). TensorFlow. https://www.tensorflow.org/tutorials/generative/pix2pix?hl=fr

Van Veldhuizen, D. A., & Lamont, G. B. (1998). Evolutionary computation and convergence to a pareto front. *Late breaking papers at the genetic programming 1998 conference*, 221-228.

Vanegas, C. A., Aliaga, D. G., Benes, B., & Waddell, P. (2009). Visualization of Simulated Urban Spaces : Inferring Parameterized Generation of Streets, Parcels, and Aerial Imagery. *IEEE Transactions on Visualization and Computer Graphics*, *15*(3), 424-435. https://doi.org/10.1109/TVCG.2008.193

Vegas, G., Bernal, M., & Calvo, F. (s. d.). *Multi-Criteria Agent Based Systems*.

Veloso, P., Celani, G., & Scheeren, R. (2018). From the generation of layouts to the production of construction documents : An application in the customization of apartment plans. *Automation in Construction*, *96*, 224-235. https://doi.org/10.1016/j.autcon.2018.09.013

Verbeke, S., & Audenaert, A. (2018). Thermal inertia in buildings : A review of impacts across climate and building use. *Renewable and sustainable energy reviews*, *82*, 2300-2318.

Vermeulen, T., Merino, L., Knopf-Lenoir, C., Villon, P., & Beckers, B. (2018). Periodic urban models for optimization of passive solar irradiation. *Solar Energy*, *162*, 67-77.

Von Neumann, J., & Burks, A. W. (1966). Theory of self-reproducing automata. *IEEE Transactions on Neural Networks*, *5*(1), 3-14.

Waibel, C., Bystricky, L., Kubilay, A., Evins, R., & Carmeliet, J. (2017, août 7). *Validation of Grasshopper-based Fast Fluid Dynamics for Air Flow around Buildings in Early Design Stage*. 2017 Building Simulation Conference. https://doi.org/10.26868/25222708.2017.582

Waibel, C., Wortmann, T., Evins, R., & Carmeliet, J. (2019). Building energy optimization : An extensive benchmark of global search algorithms. *Energy and Buildings, 187,* 218-240. https://doi.org/10.1016/j.enbuild.2019.01.048

Wang, W., Liu, K., Zhang, M., Shen, Y., Jing, R., & Xu, X. (2021). From simulation to data-driven approach : A framework of integrating urban morphology to low-energy urban design. *Renewable Energy, 179*, 2016-2035. https://doi.org/10.1016/j.renene.2021.08.024

Wang, X.-Y., Yang, Y., & Zhang, K. (2018). Customization and generation of floor plans based on graph transformations. *Automation in Construction, 94*, 405-416. https://doi.org/10.1016/j.autcon.2018.07.017

Weber, B., Müller, P., Wonka, P., & Gross, M. (2009). Interactive geometric simulation of 4d cities. *Computer Graphics Forum*, *28*(2), 481-492.

Weerasuriya, A. U., Zhang, X., Wang, J., Lu, B., Tse, K. T., & Liu, C.-H. (2021). Performance evaluation of population-based metaheuristic algorithms and decision-making for multi-objective optimization of building design. *Building and Environment, 198*, 107855.

Wegman, E. J. (1990). Hyperdimensional data analysis using parallel coordinates. *Journal of the American Statistical Association*, *85*(411), 664-675.

Westermann, P., & Evins, R. (2019). Surrogate modelling for sustainable building design – A review. *Energy and Buildings, 198*, 170-186. https://doi.org/10.1016/j.enbuild.2019.05.057

Wienold, J., & Christoffersen, J. (2006). Evaluation methods and development of a new glare prediction model for daylight environments with the use of CCD cameras. *Energy and buildings*, *38*(7), 743-757.

Wolpert, D. H., & Macready, W. G. (1997). No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, *1*(1), 67-82. https://doi.org/10.1109/4235.585893

Wortmann, T. (2017). Model-based Optimization for Architectural Design : Optimizing Daylight and Glare in Grasshopper. *Technology|Architecture + Design*, *1*(2), 176-185. https://doi.org/10.1080/24751448.2017.1354615

Wortmann, T. and G. N. (2016). Black-Box Optimisation Methods for Architectural Design. *Living Systems and Micro-Utopias: Towards Continuous Designing, Proceedings of the 21st International Conference on Computer-Aided Architectural Design Research in Asia (CAADRIA 2016) / Melbourne 30 March–2 April 2016, pp. 177-186*. http://papers.cumincad.org/cgi-bin/works/paper/caadria2016_177

Wortmann, T., Costa, A., Nannicini, G., & Schroepfer, T. (2015). Advantages of surrogate models for architectural design optimization. *Artificial Intelligence for Engineering Design, Analysis and Manufacturing*, *29*(4), 471-481. https://doi.org/10.1017/S0890060415000451

Wortmann, T., & Nannicini, G. (2017). Introduction to Architectural Design Optimization. In A. Karakitsiou, A. Migdalas, S. Th. Rassia, & P. M. Pardalos (Éds.), *City Networks : Collaboration and Planning for Health and Sustainability* (p. 259-278). Springer International Publishing. https://doi.org/10.1007/978-3-319-65338-9_14

Wortmann, T., & Natanian, J. (2020). Multi-objective optimization for zero-energy urban design in China : A benchmark. *Proceedings of the 11th Annual Symposium on Simulation for Architecture and Urban Design*, 1-8.

Wortmann, T., & Natanian, J. (2021). Optimizing solar access and density in Tel Aviv : Benchmarking multi-objective optimization algorithms. *Journal of Physics: Conference Series*, *2042*(1), 012066.

Yang, D., Ren, S., Turrin, M., Sariyildiz, S., & Sun, Y. (2018). Multi-disciplinary and multi-objective optimization problem re-formulation in computational design exploration : A case of conceptual sports building design. *Automation in Construction*, *92*, 242-269. https://doi.org/10.1016/j.autcon.2018.03.023

Yi, H. (2014). Automated generation of optimised building envelope : Simulation based multi-objective process using evolutionary algorithm. *International Journal of Sustainable Building Technology and Urban Development*, *5*(3), 159-170.

Yi, H. (2016). User-driven automation for optimal thermal-zone layout during space programming phases. *Architectural Science Review*, *59*(4), 279-306. https://doi.org/10.1080/00038628.2015.1021747

Yi, H., & Yi, Y. K. (2014, octobre 1). *Performance bases architectural design optimization : Automated 3D space layout using simulated annealing*. ASHRAE/IBPSA Builing Simulation Conference, Atlanta, USA.

Yi, Y. K. (2019). Building facade multi-objective optimization for daylight and aesthetical perception. *Building and Environment*, *156*, 178-190.

Yi, Y. K., & Kim, H. (2015). Agent-based geometry optimization with Genetic Algorithm (GA) for tall apartment's solar right. *Solar Energy*, *113*, 236-250. https://doi.org/10.1016/j.solener.2014.11.007

Youssef, A. M., Zhai, Z. J., & Reffat, R. M. (2018). Generating proper building envelopes for photovoltaics integration with shape grammar theory. *Energy and buildings*, *158*, 326-341.

Zargar, S. H., & Alaghmandan, M. (2019). CORAL : Introducing a fully computational plug-in for stadium design and optimization; a case study of finding optimal spectators' viewing angle. *Architectural Science Review*, *62*(2), 160-170.

Zarrabi, A. H., Azarbayjani, M., & Tavakoli, M. (s. d.). *Generative Design Tool : Integrated Approach toward Development of Piezoelectric Façade System*.

Zawidzki, M. (2009). Implementing Cellular Automata for Dynamically Shading a Building Facade. *Complex Systems*, *18*. https://doi.org/10.25088/ComplexSystems.18.3.287

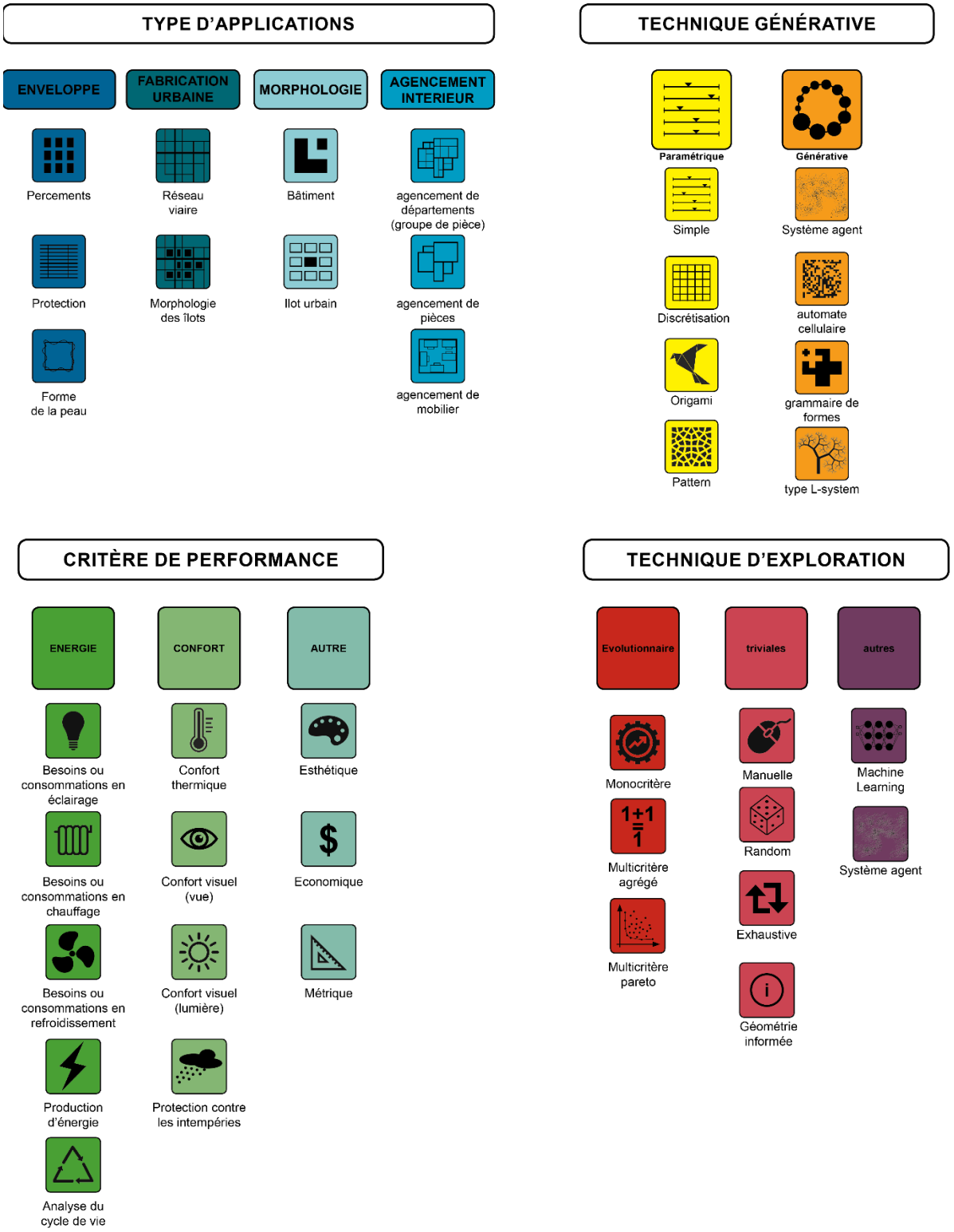
Zhao, S., & de Angelis, E. (2018). Performance-based Generative Architecture Design : A Review on Design Problem Formulation and Software Utilization. *Journal of Integrated Design and Process Science*, *22*(3), 55-76. https://doi.org/10.3233/JID190001

Zitzler, E., Laumanns, M., & Thiele, L. (2001). *SPEA2 : Improving the strength pareto evolutionary algorithm* (p. 21 p.) [Application/pdf]. ETH Zurich. https://doi.org/10.3929/ETHZ-A-004284029

Zlochin, M., Birattari, M., Meuleau, N., & Dorigo, M. (2004). Model-based search for combinatorial optimization : A critical survey. *Annals of Operations Research*, *131*, 373-395.

Annexes

Annexe 1 Catalogue des applications théoriques



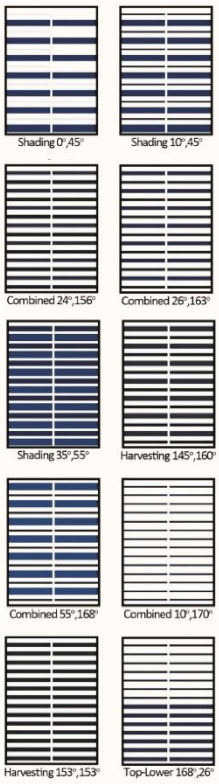
Cas d'étude théorique
Protection solaire adaptative
Pas de masques solaire
Etude d'un détail

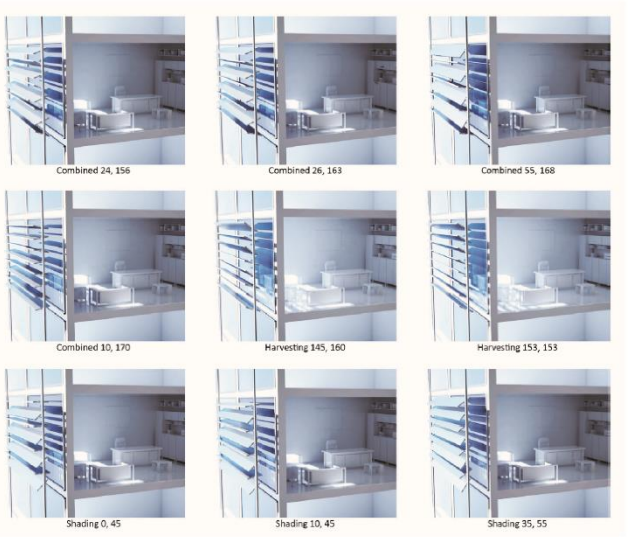
paramétrique simple dévelopé sur grasshopper

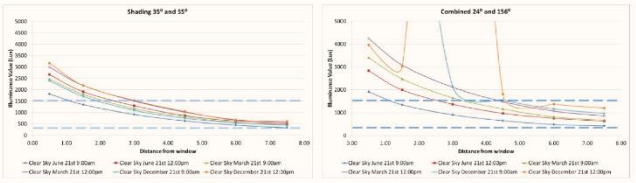
Analyse de l'éclairement utile
Répartition de l'intensité lumineuse
Méthode testée au 21/03, 2/06, 21/12 à 9h et 12h

Optimisation à un objectif
Solver Galapagos
Algorithme génétique
Méthode d'agrégation: pourcentage de la surface étudiée qui respecte plusieurs conditions d'éclairement et d'intensité lunimneuse

Le modèle mis en place a montré que l'automatisation de l'inclinaison des stores permet d'augmenter de 2 à 2,5 fois la pénétration de la lumière naturelle dans le bâtiment.







Source: El Sheikh, Mohamed, and David Gerber. «Building Skin Intelligence: A parametric and algorithmic tool for daylighting performance design integration.» (2011).

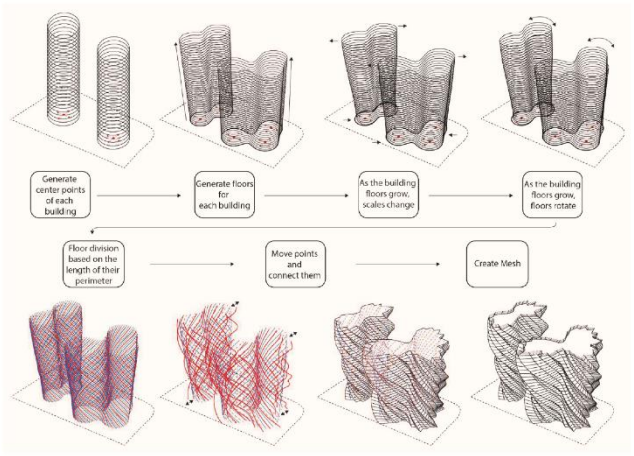
Retro-conception des Al-Bahr Towers à Abou Dabi aux Emirats arabes unis (conçue par Aedas et Diar Consult).
Ombres propres

Modèle paramétrique en deux étapes avec condition basé sur une évaluation de la radiation (grasshopper)

Surface brute de plancher (pour quelle soit la plus proche de la surface originiale)
Radiation solaire au 21/06 des façades pour minimiser les apports dans les bureaux,
Radiation solaire au 21/06 au sol pour maximiser le confort thermique des espaces publics
Qualité de vue depuis l'intérieur des bureaux

solveur nommé «Wallacei» agorithme NSGAII.

La température pouvant monter à 60°C dans les pays du Golfe, l'objectif était d'élaborer deux bâtiments capables se créent mutuellement et individuellement un maximum d'ombre pour limiter l'usage des protections solaire qui impacte fortement le confort visuel.



Source: Showkatbakhsh, Milad, and Saam Kaviani. «Homeostatic generative design process: Emergence of the adaptive architectural form and skin to excessive solar radiation.» International Journal of Architectural Computing (2020): 1478077120951947.

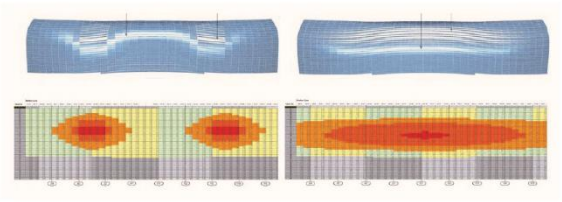
Expérimentation sur l'Aviva Stadium à Dublin, en Irlande.
Enveloppe adaptative
Pas de masques solaire

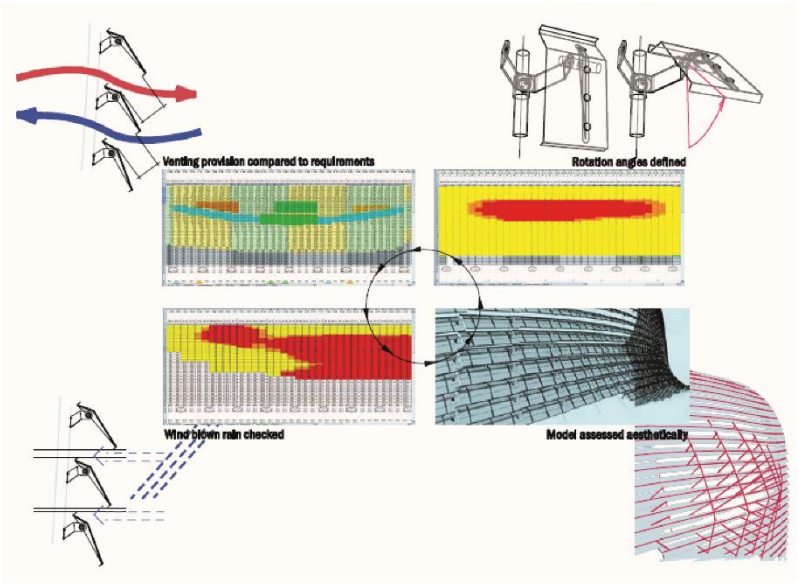
modèle paramétrique avec grasshopper

ventilation de la façade protection contre les intempéries

Mise en place d'un processus itératif permettant à l'aide du paramétrique d'accélérer les allers-retours entre les aspects esthétiques et les aspects techniques.

La géométrie de l'enveloppe est ajustée à l'aide du modèle paramétrique pour des questions esthétiques.
Le modèle paramétrique a aussi utilisée pour réaliser des rendus esthétiques et pour la réduction des coûts.





Source: Hudson, Roly, Paul Shepherd, and David Hines. «Aviva Stadium: A case study in integrated parametric design.» International Journal of Architectural Computing 9.2 (2011): 187-203.

Cas étude appliqués à un immeuble de bureau situé à Lacarna en République de Cyprus

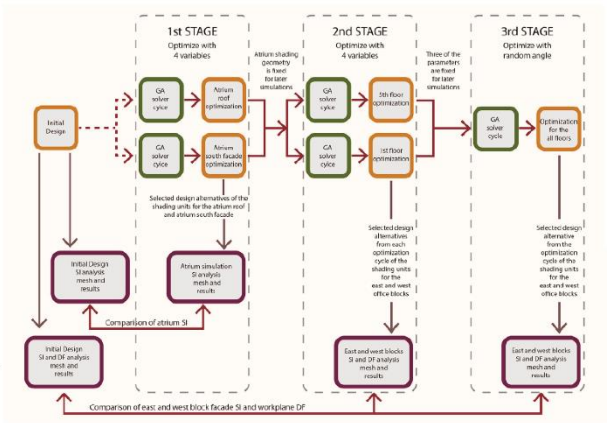
Etude d'un détail avant une étude plus générale
Forme générale: sans courbes

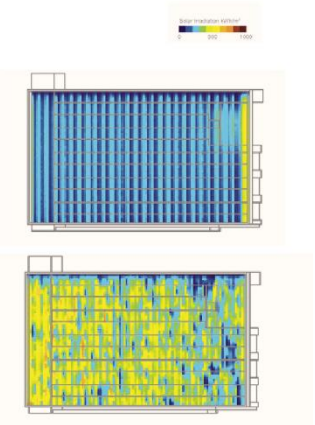
Modèle paramétrique sur grasshopper

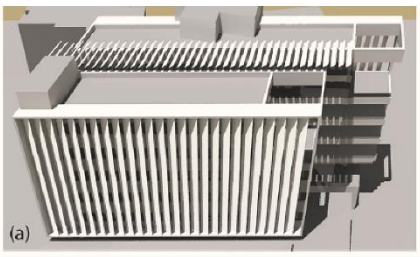
Facteur de lumière du jour
Radiation solaire en façade

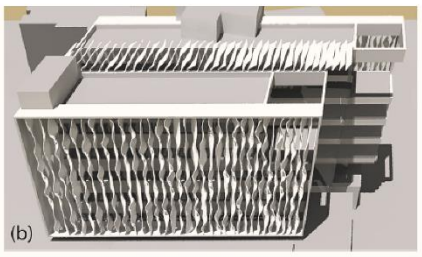
Neuf optimisations au total
Solveur Galapagos
Algorithme génétique
(1) dimentionnement des lames pour minimiser la radiation de l'atrium
(2) puis de la facade Sud
27 générations de 25 individus
(3) géométrie des lames en façades pour maximiser le FLJ sur 2 niveaux uniquement (28h de calcul).
(4)torsion des lames pour maximisant le FLJ sur l'ensemble des niveaux (48h de calcul).

Les variables sont fixées au fur et à mesure des différentes explorations.









Source: Ercan, Burak, and Soofia Tahira Elias-Ozkan. «Performance-based parametric design explorations: A method for generating appropriate building components.» Design Studies 38 (2015): 33-53.

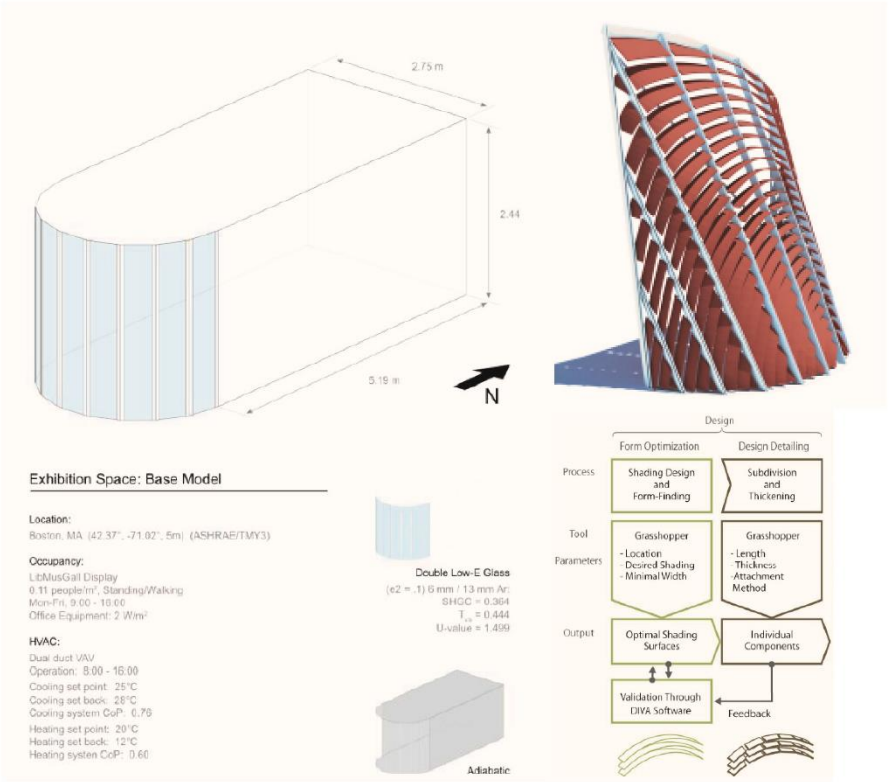
Cas d'étude théorique
Pas de masques solaire

Modèle paramétrique
grasshopper

confort visuel
consommations
météo de Boston.

boucle de calcul

Des contraintes de construction et de fabrication ont été intégré à l'approche dans un second temps.



Bechthold, Martin, et al. «Integrated environmental design and robotic fabrication workflow for ceramic shading systems.» (2011).

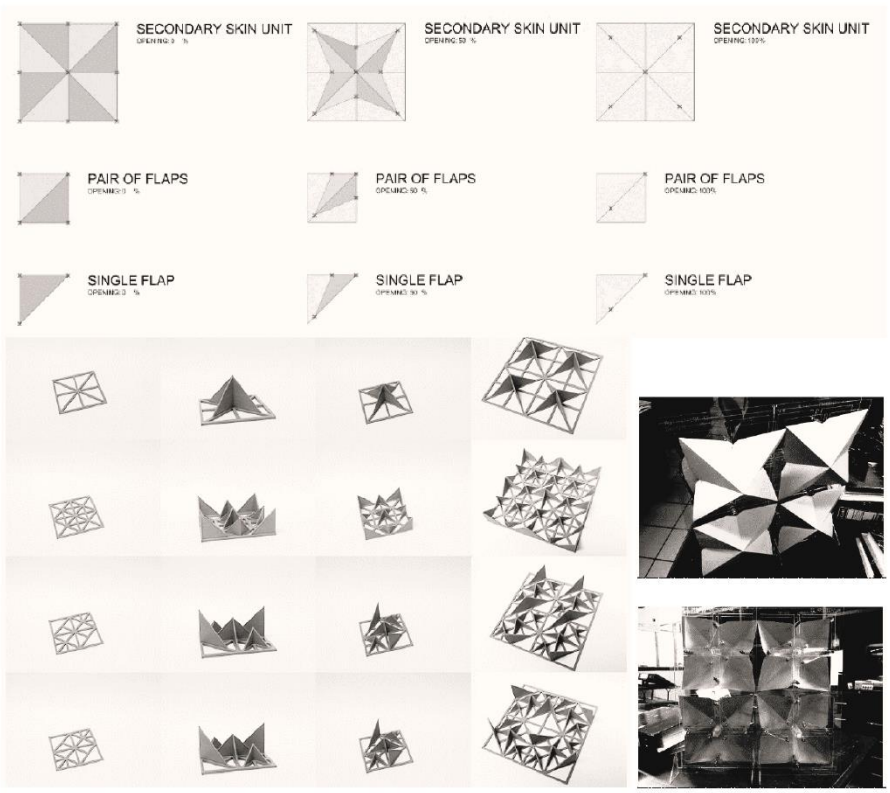
Cas d'étude appliqué à un SOHO (Small Office Home Office) à Jakarta
Façade adaptative
Pas de masques solaire
Etude d'un détail

Modèle génératif,
grammaire de forme

coefficient de transfert thermique

6 scénarios ont été évalués.

L'usage d'un processus génératif n'a pas d'impact sur les performances finales de la façade, ni sur l'aspect esthétique. Cependant, les grammaires de formes permettent de faire émerger des formes inattendues.



Source: Tomasowa, Riva, and Firza Utama Sjarifudin. «Adaptive Façade: Variant-Finding using Shape Grammar.» IOP Conference Series: Earth and Environmental Science. Vol. 109. No. 1. IOP Publishing, 2017.

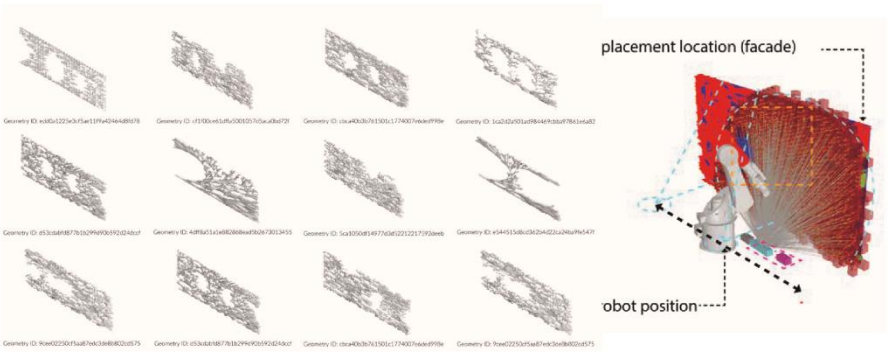
Cas d'étude théorique
Pas de masques solaire
Etude d'un détail

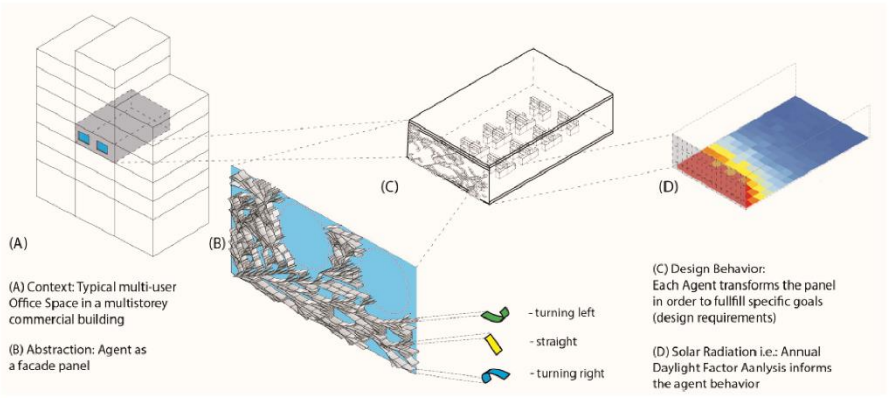
Modèle génératif en java

Facteur de lumière du jour
Eclairement utile
Autonomie en lumière du jour
Simulation annuelle
Radiation solaire

Cette approche mèle une approche analytique et une méthode d'optimisation car l'ensemble du processus est gérer par un système multi-agent. L'ensemble des critères est agrégé dans une unique fonction dont la valeur permet de faire évoluer les paramètre du processus génératif de la géométrie de départ.

L'ensemble des solutions explorées peuvent à la fin être comparée à la l'aide la méthode de Pareto.





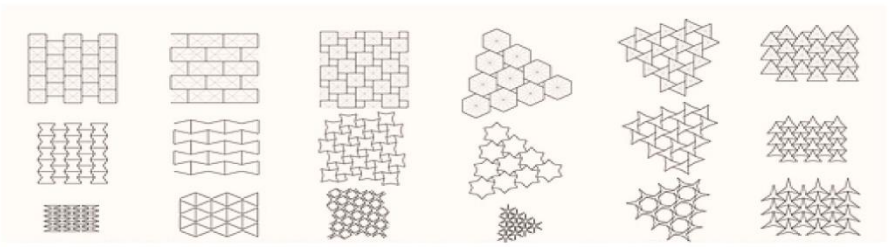
Gerber, David J., Evangelos Pantazis, and Alan Wang. «A multi-agent approach for performance based architecture: design exploring geometry, user, and environmental agencies in façades.» Automation in construction 76 (2017): 45-58.

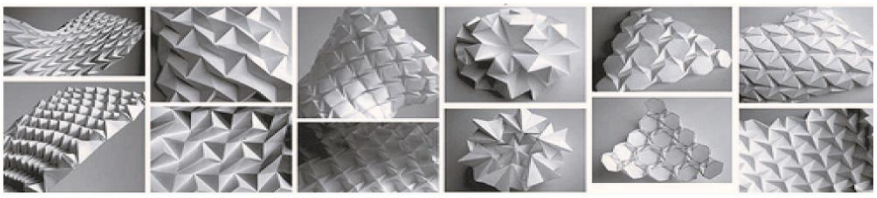
Cas d'étude théorique
Pas de masques solaire
Etude d'un détail

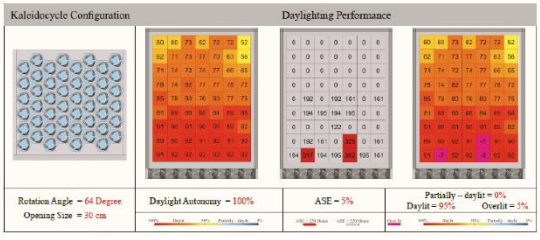
Modèle paramétrique
Grasshopper
Origami pour kaléidocycle

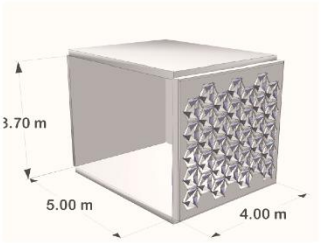
Autonomie en lumière du jour
Exposition annuelle à la lumière du soleil (ASE)
Météo Caire, Égypte

Algorithme génétique
73 générations de 30 individus









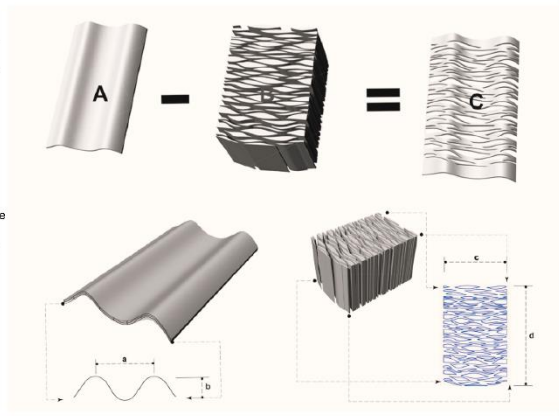
Source: Elghazi, Y., et al. «Daylighting driven design: optimizing kaleidocycle facade for hot arid climate.» Aachen: Fifth German-Austrian IBPSA Conference, RWTH Aachen University. 2014.

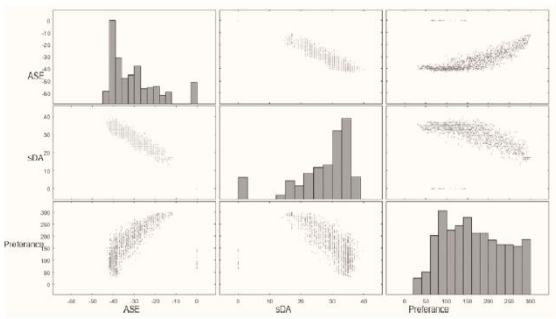
Retro-conception du Suited Avenue Building de Toyo Ito à Passeig de Gracia, Barcelone Espagne.
Masques solaires du contexte urbain

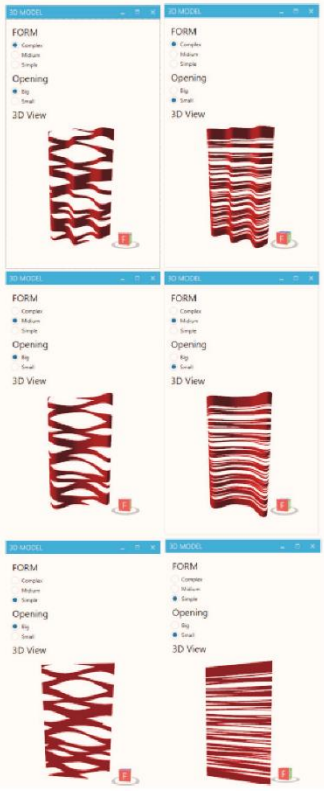
Modèle paramétrique
Grasshopper

Autonomie en lumière du jour
Exposition annuelle à la lumière du soleil (ASE)
Esthétique (learning machine)

NSGAII vis Matlab







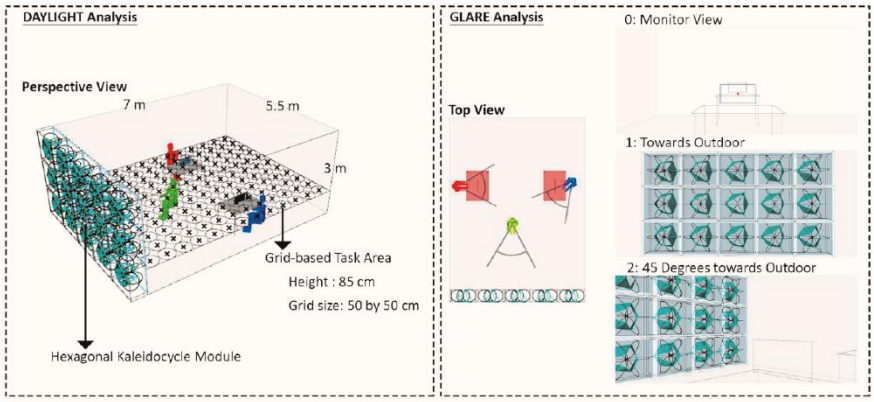
Source: Yi, Yun Kyu. «Building facade multi-objective optimization for daylight and aesthetical perception.» Building and Environment 156 (2019): 178-190.

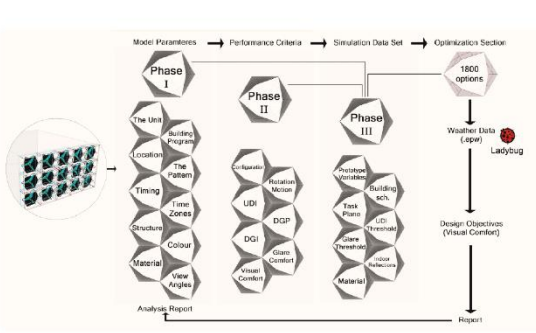
Cas d'étude théorique
Facade adaptative inspirée des Al-Bahr Towers à Abou Dabi aus Emirats arabes unis (conçue par Aedas et Diar Consult).
Pas de masques solaire
Etude d'un détail

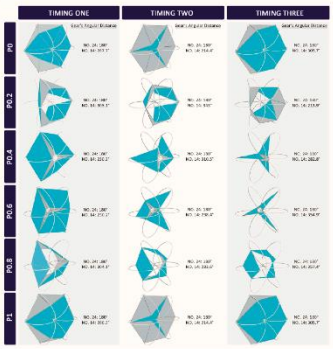
Modèle paramétrique
Grasshopper
origami pour kaléidocycle

Eclairement utile
Indicateur d'éblouissement DGP (Daylight Glare Probability), et DGI (Daylight Glare Index).
Météo de tehran en Iran
Différentes heures de la journée aux solstices et équinoxes.

Solveur Galapagos
Algorithme génétique
35 générations de 100 individus
Visualisation Design Explorer







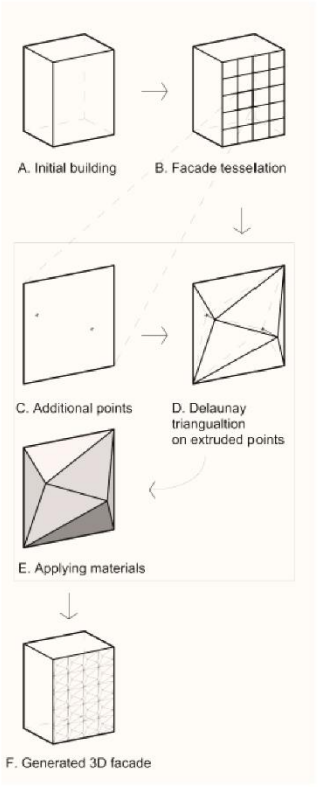
Source: Tabadkani, Amir, et al. «Integrated parametric design of adaptive facades for user's visual comfort.» Automation in Construction 106 (2019): 102857.

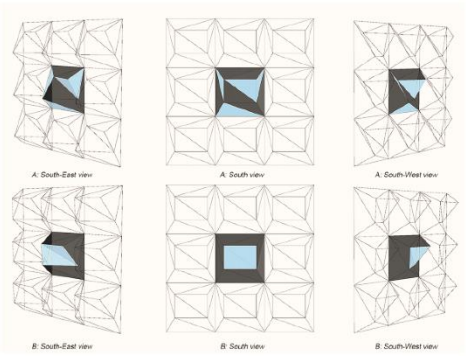
Cas d'étude théorique
Pas de masques solaire
Etude d'un détail

Modèle paramétrique
Grasshopper
triangulation de Delaunay

Eclairement utile (annuelle)
Production d'énergie à l'aide de panneaux photovoltaïque.

Solveur Octopus,
algorithme HypE





Source: Narangerel, Amartuvshin, Ji-Hyun Lee, and Rudi Stouffs. «Daylighting Based Parametric Design Exploration of 3D Facade Patterns.» (2016).

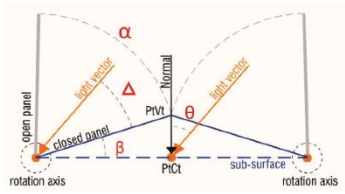
Cas d'étude théorique
Lucarnes adaptatives
Pas de masques solaire

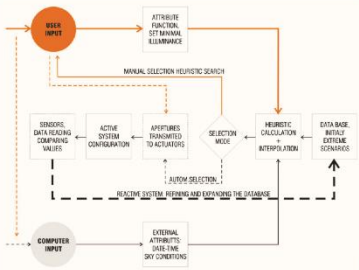
Modèle paramétrique
Grasshopper

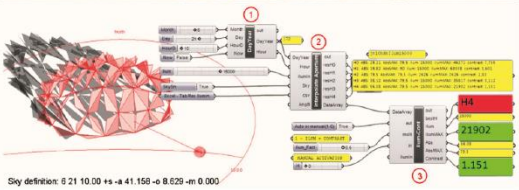
Eclairement naturel

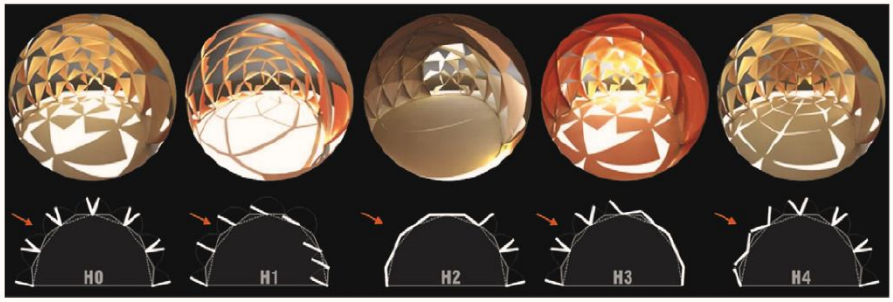
Cinq scénarios ont été analysés à différentes dates dans l'année et à différentes heures de la journée.

Ce processus permet de contrôler les lanternaux en temps réel afin d'augmenter les performances de la lumière du jour.









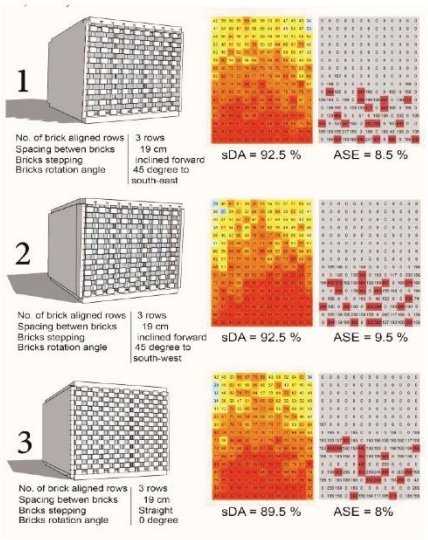
Source: Henriques, Gonçalo Castro, José Pinto Duarte, and Vitor Leal. «Strategies to control daylight in a responsive skylight system.» Automation in Construction 28 (2012): 91-105.

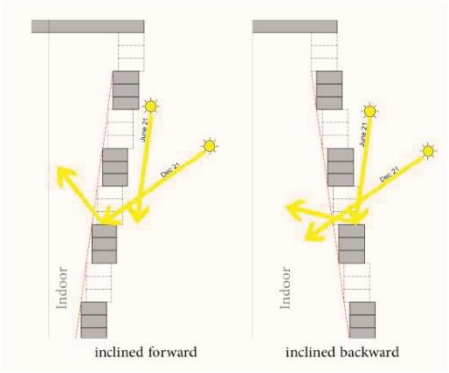
Cas d'étude théorique
Moucharabieh en brique
Pas de masques solaire
Etude d'un détail

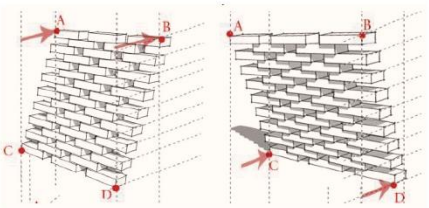
Modèle paramétrique
Grasshopper

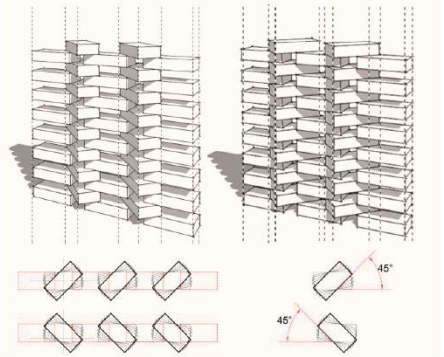
Autonomie spatiale en éclairage naturelle
Exposition annuelle à la lumière du soleil (ASE)
Analyses annuelles

Approche exhaustive
270 simulations réalisées
2 jours de calculs









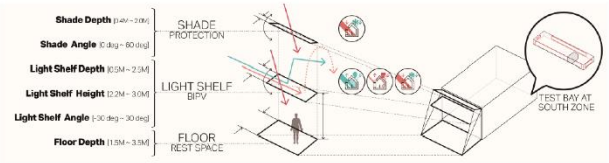
Sources: Abdelwahab, Sahar, and Yomna Elghazi. «A generative performance-based design for low cost brickwork screens.» Proceedings of the Building Simulation and Optimization Conference (BSO16). 2016.

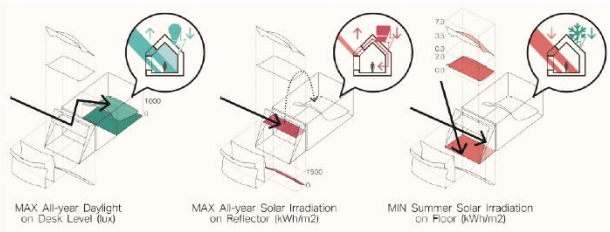
Cas d'étude théorique
Pas de masques solaire
Etude d'un détail

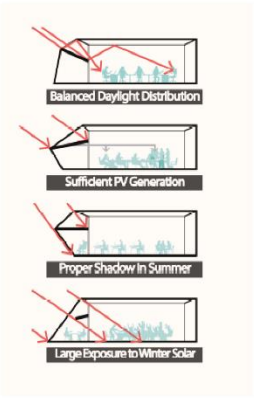
Modèle paramétrique

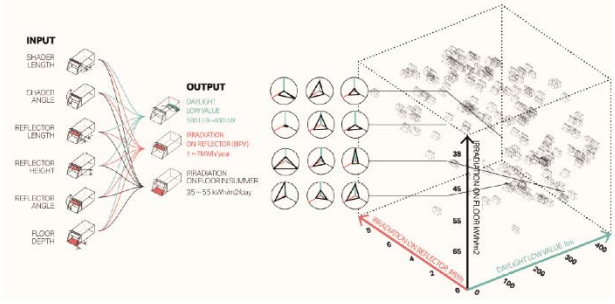
Radiation solaire sol +réflecteur
Eclairement naturel
Production panneaux photovoltaïque

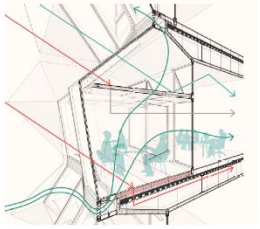
Solveur Octopus.
10 générations de 50 individus.











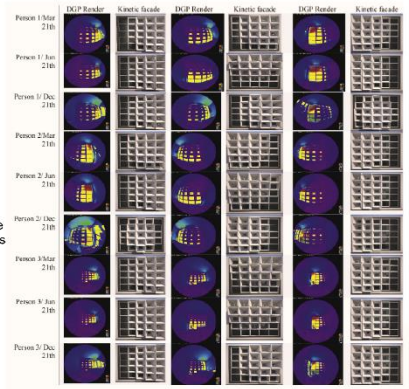
Sources: Shen, Xiaofei. «Environmental parametric multi-objective optimization for high performance facade design.» (2018).

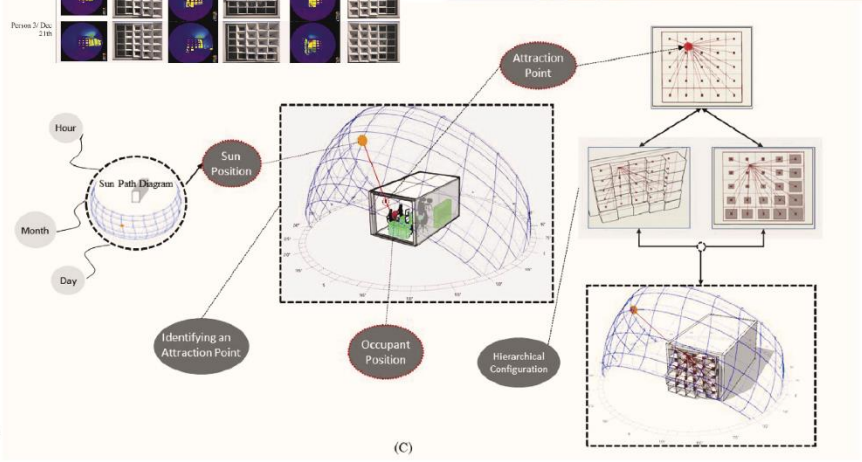
Cas d'étude théorique
Façade adaptative
Pas de masques solaire
Etude d'un détail

Modèle paramétrique
Grasshopper

Autonomie en lumière du jour
Eclairement utile
Risque d'éblouissement.

Approche exhaustive a été utilisée pour analyser l'éclairage naturelle sur différents scénarios de façades, à différentes dates et heures de la journée.





Sources: Hosseini, Seyed Morteza, Masi Mohammadi, and Olivia Guerra-Santin. «Interactive kinetic façade: Improving visual comfort based on dynamic daylight and occupant's positions by 2D and 3D shape changes.» Building and Environment 165 (2019): 106396.

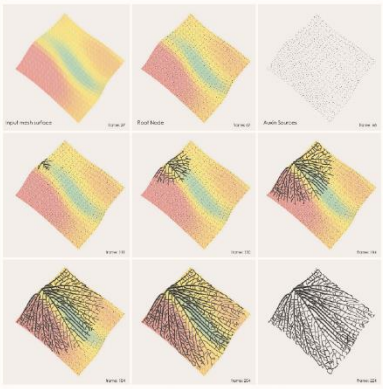
Cas d'étude théorique
Ombres propres

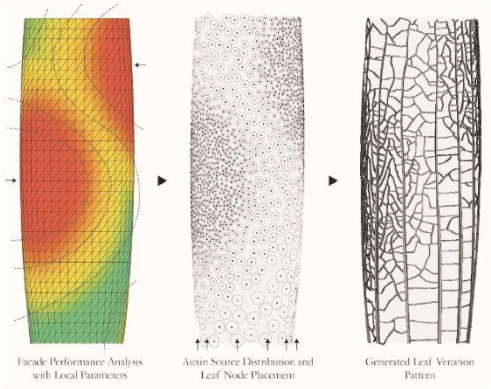
Modèle génératif, approche morphogénétique inspirée d'un motif végétal.

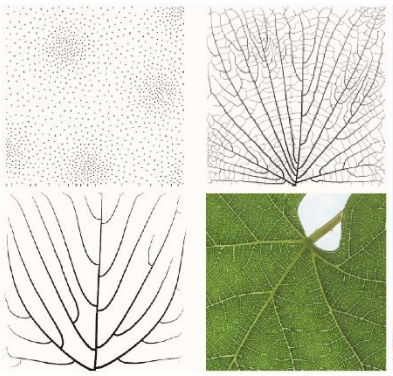
Radiation solaire

Géométrie informée sur laquelle le processus génératif se base pour faire «croître» sa façade morphogénétique

Algorithme inspirée de l'auxine qui est une phytohormone de croissance végétale indispensable au développement des plantes.







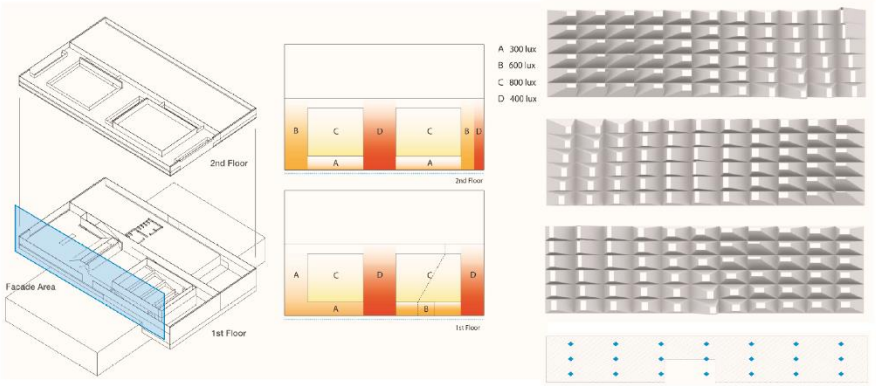
Source: Gokmen, Sabri. «A Morphogenetic Approach for Performative Building Envelope Systems Using Leaf Venetian Patterns.» eCAADe 2013: Computation and Performance–Proceedings of the 31st International Conference on Education and research in Computer Aided Architectural Design in Europe, Delft, The Netherlands, September 18-20, 2013. Faculty of Architecture, Delft University of Technology; eCAADe (Education and research in Computer Aided Architectural Design in Europe), 2013.

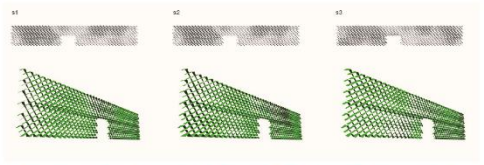
Expérimentation pour le nouveau bâtiment du campus de l'université de Delft (PULSE) avec l'agence Ector Hoogstad Architects
Pas de masques solaire

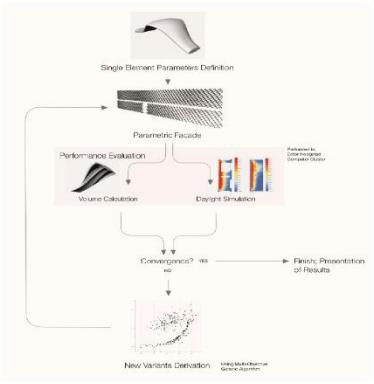
Modèle paramétrique
Panelisation 3D.

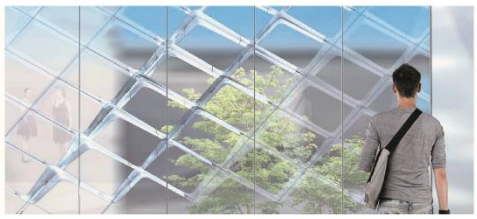
Eclairement utile
Quantité de matière

Solveur Octopus
Algorithme HypE
50 générations, population 100
25 machines pour un WE de calcul









Sources: Chatzikonstantinou, Ioannis, et al. «A Comprehensive Optimization Approach for Modular Facades: The Case of PULSE Sunshading.» International Journal of Design Sciences & Technology 23.2 (2019): 159-185.

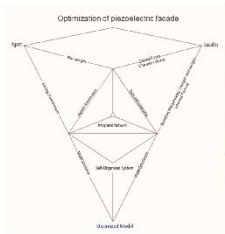
Cas d'étude théorique
Façade Piézoélectrique
Pas de masques solaire

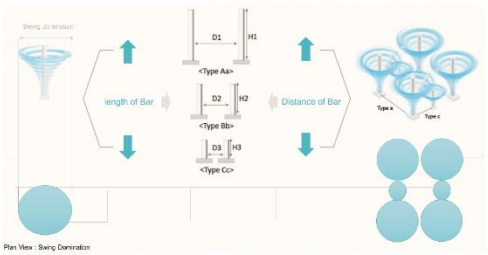
Processus génératif
Modèle à base d'agents pour la collision entre les paille

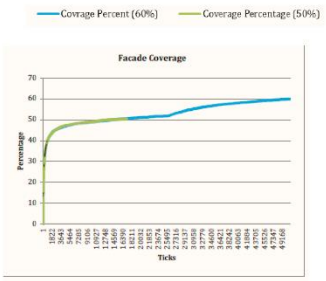
Vitesse et la pression des vents en façade.

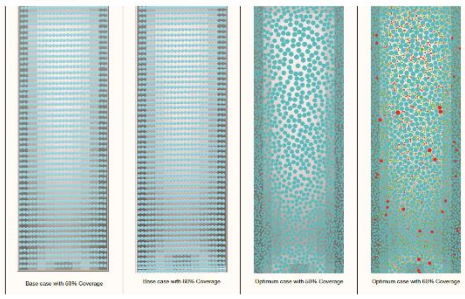
Géométrie informée à partir de laquelle le système multi-agents travaillent.

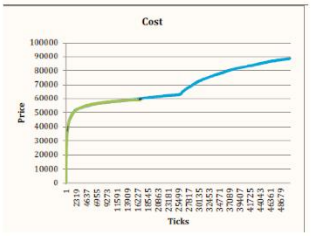
L'ensemble des solutions explorées pendant le processus sont ensuite comparées à la solution optimale pour minimiser le coût de fabrication (volume de matériaux).











Sources: Zarrabi, Amir Hosseinzadeh, Mona Azarbayjani, and Maryam Tavakoli. «Generative Design Tool: Integrated Approach toward Development of Piezoelectric Façade System.»

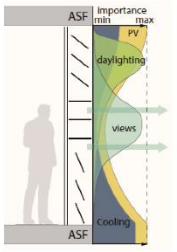
Expérimentation, hilo building, Duebendorf, Suisse, phase esquisse
Facade adapatative
Pas de masques solaire
Etude d'un détail

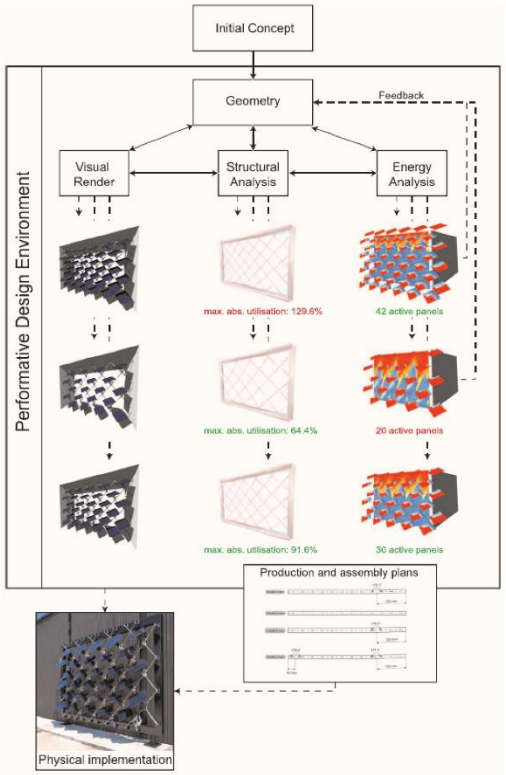
Modèle Paramétrique
Grasshopper

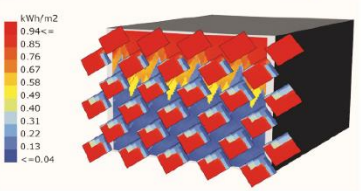
Radiation solaire pour la production d'électricité avec des panneaux photovoltaïques.
Autonomie en lumière du jour
Consommations énergétiques pour le chauffage et le refroidissement.

Approche exhaustive

Il est précisé qu'une optimisation a été réalisée pour faire un trade off entre la production d'énergie et le confort lumineux afin de minimiser les consommations d'énergie globale. Les analyse sont réalisée sur une journée typique de l'été et une journée typique de l'hiver.







Sources: Jayathissa, Prageeth, et al. «Performative design environment for kinetic photovoltaic architecture.» Automation in Construction 93 (2018): 339-347.

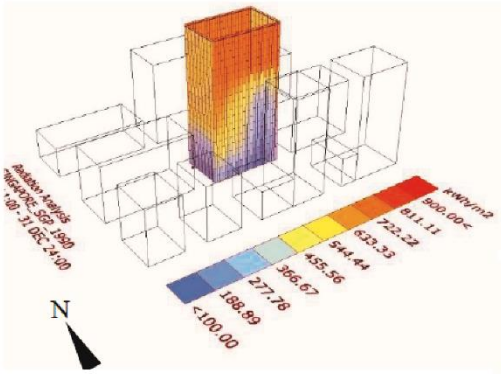
Cas d'étude théorique
Masques, contexte urbain hétérogène

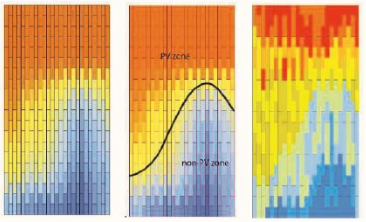
Modèle paramétrique
Grasshopper

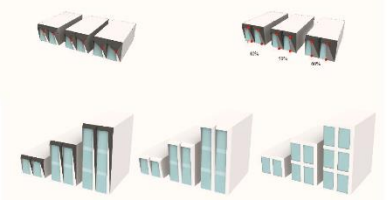
Radiation solaire

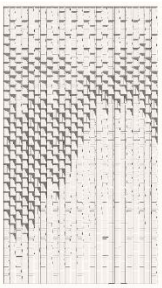
Méthode d'intégration des critères de performance par l'usage de la géométrie informée.

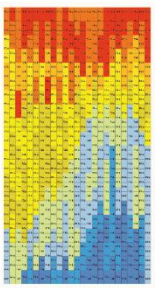
La phase d'évaluation a lieu avant la phase générative, et les données de la phase d'évaluation sont suffisamment hétérogène pour s'appuyer dessus pour un concept architecturale de façade incluant une stratégie de pixelisation.

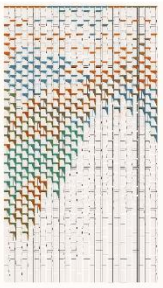


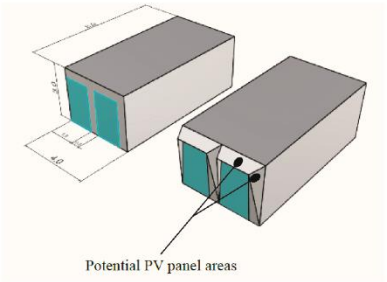












Sources: Chen, T., et al. «A design-driven approach to integrate high-performance photovoltaics devices on the building façade.» IOP Conference Series: Earth and Environmental Science. Vol. 294. No. 1. IOP Publishing, 2019.

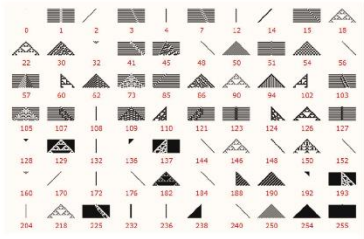
Cas d'étude théorique
Pas de masques solaire

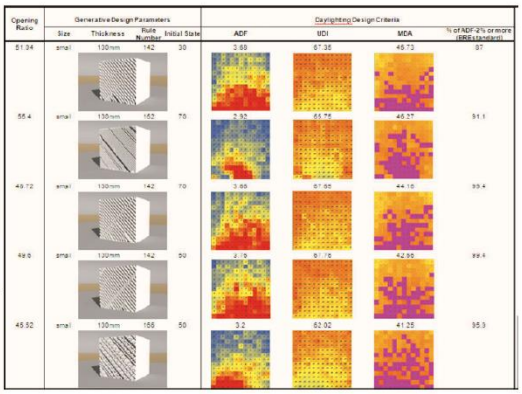
Modèle génératif, automate cellulaire sur grasshopper

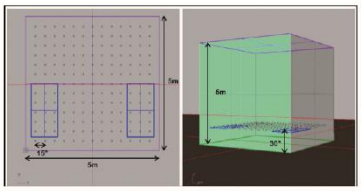
Facteur de lumière du jour
Eclairement utile
Autonomie en lumière du jour
21/9 à 9h et 15h
météo Birmingham, RU

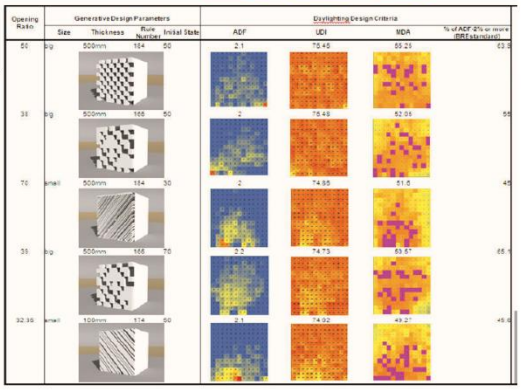
30 scénarios (avec des cellules de tailles différentes, des épaisseurs de murs variables) sont évalués et comparés

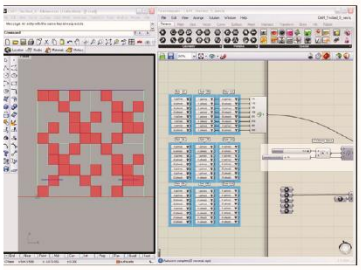
L'usage d'un processus génératif n'a pas d'impact sur les performances finales de la façade, l'emploi des automates cellulaires est ici un choix esthétique.











Kim, Jieun. «Adaptive façade design for the daylighting performance in an office building: the investigation of an opening design strategy with cellular automata.» International Journal of Low-Carbon Technologies 10.3 (2015): 313-320.

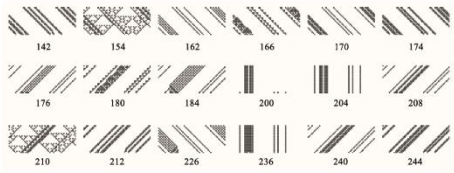
Cas d'étude théorique
Pas de masques solaire

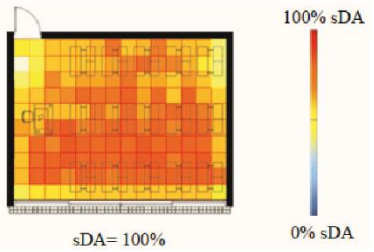
Modèle génératif, automate cellulaire sur grasshopper

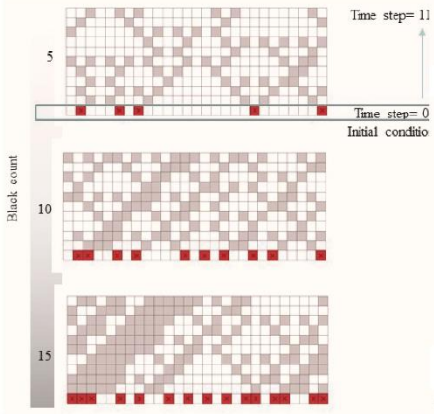
Autonomie en lumière du jour
Exposition annuelle à la lumière du soleil (ASE) pour prévenir l'éblouissement

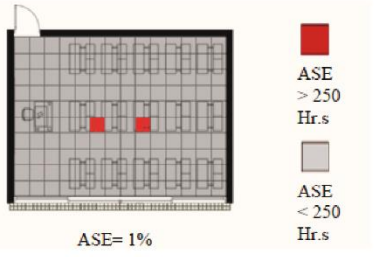
877 scénarios évalués

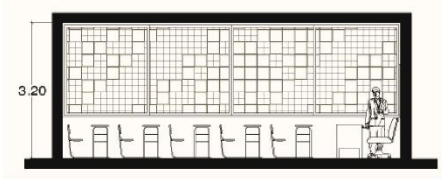
L'usage d'un processus génératif n'a pas d'impact sur les performances finales de la façade, l'emploi des automates cellulaires est ici un choix esthétique.

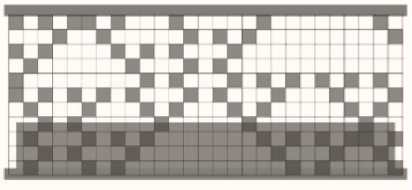












Source: Fathy, Fatma, et al. «Cellular automata for efficient daylighting performance: optimized façade treatment.» Proceedings of the 14th Conference of the International Building Performance Simulation Association (IBPSA). 2015.

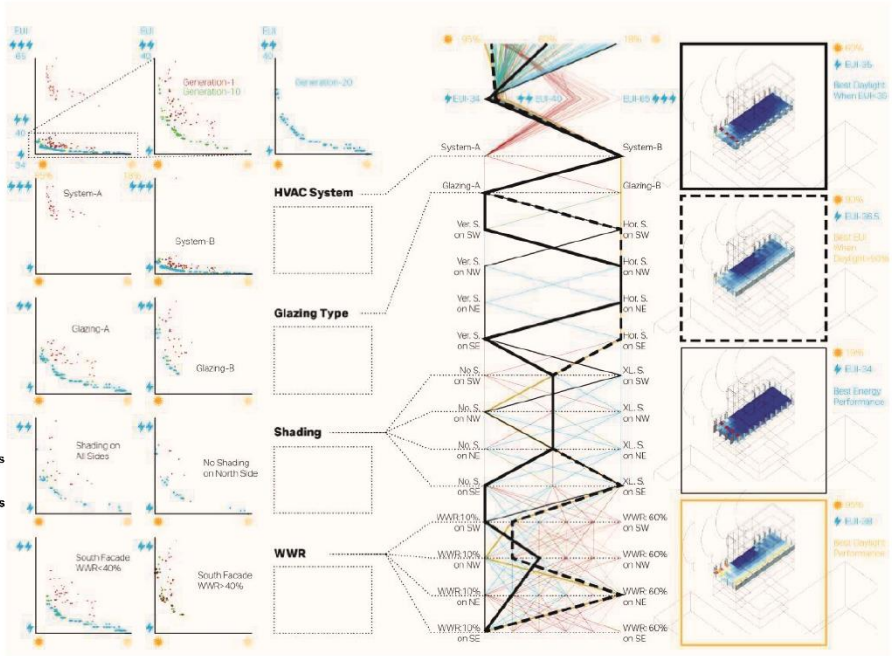
Expérimentations sur un laboratoire à San Francisco, chez AECOM

Modèle paramétrique
Grasshopper
Variables géométriques, matériaux et systèmes CVC

Une simulation d'éclairement à 9h et 15h pour 2 indicateurs:
Eclairement utile
Consommation énergétique

Solveur Octopus

Les auteurs ont conclus à des temps de calculs très longs, trop pour la pratique et envisagent d'étudier les méthodes de calculs décentralisées (cloud, paralélisation).



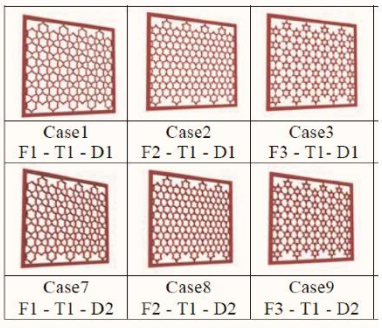
Sources: Shen, Xiaofei, et al. «Evaluating the multi-objective optimization methodology for performance-based building design in professional practice.» ASHRAE and IBPSA (2018): 646-653.

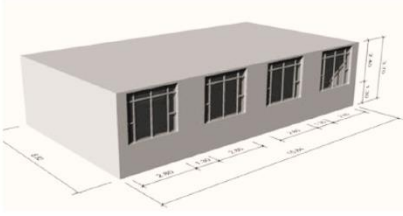
Cas d'étude théorique
Pas de masques solaire
Etude d'un détail

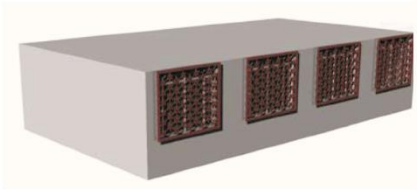
Modèle paramétrique
Motifs islamiques

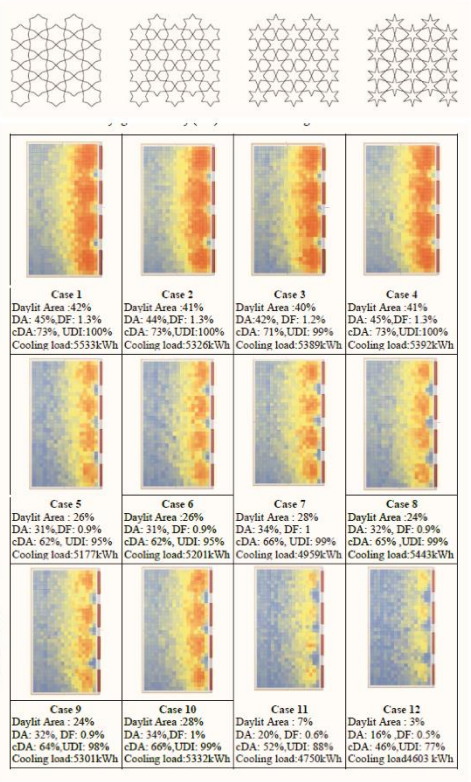
Consommations refroidissement
Facteur de lumière du jour
Autonomie spatiale en lumière du jour
Eclairement utile et Surface éclairée

Analyse de 12 scénarios de façades différents









Sources: Mohamadin, Mahmoud F., Ahmed A. Abouaiana, and Hala H. Wagih. «Parametric Islamic Geometric Pattern for Efficient Daylight and Energy Performance-Façade retrofit of educational space in hot arid climate.» (2016).

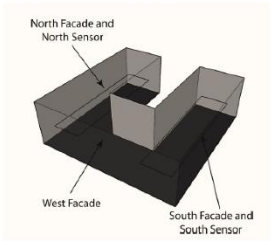
Cas d'étude théorique
Façade adaptative

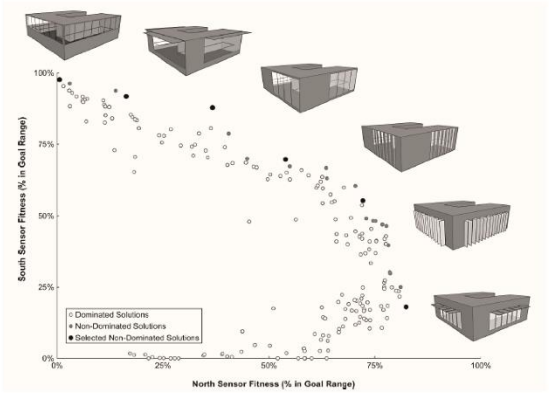
Modèle paramétrique
Grasshopper
Panélisation

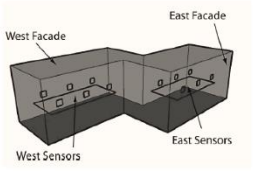
Modèle paramétrique
Méthode implémentée dans
Sketchup

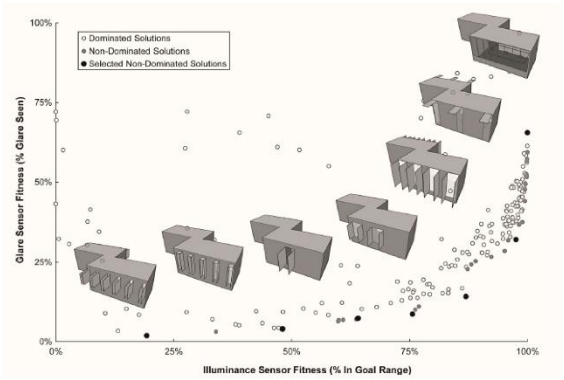
Eclairement
Risque d'éblouissement
Météo Boston, USA

Algorithme micro-génétique









Gagne, J., & Andersen, M. (2012). A generative facade design method based on daylighting performance goals. Journal of Building Performance Simulation, 5(3), 141-154.

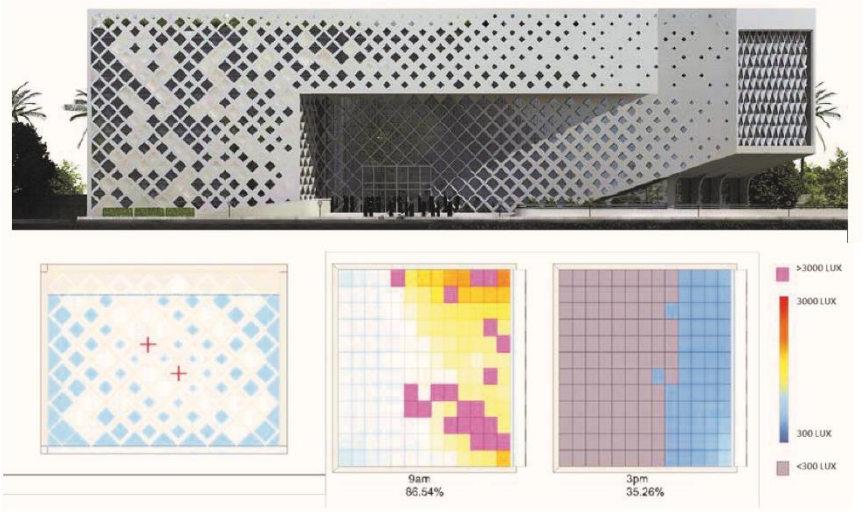
Retro conception d'une bibliothèque au Caire en Egypte
Pas de masques solaire
Etude d'un détail

paramétrique(grasshopper)
panélisation avec variation d'échelle

éclairement utile au 21/09 à 9h
clairement utile au 21/09 à 15h
besoins en chauffage +
refroidissement

Une optimisation multicritère
pour chaque orientation
solveur Octopus
3 fonctions dévaluation
10 générations de 5 individus.

Pour les façades Ouest et Est, aucune solution n'a été trouvée pour un équilibre entre éclairage et besoins énergétiques car la taille de la population est trop petite. Les auteurs spécifient que l'approche peut être utilisée en phase amont ou dans les phases ultérieures.



Source: Fathy, Fatma, and Hussein Ahmed Fareed. «Performance-driven Façade Design Using an Evolutionary Multi-Objective Optimization Approach.» International Conference for Sustainable Design of the Built Environment-SDBE London. 2017.

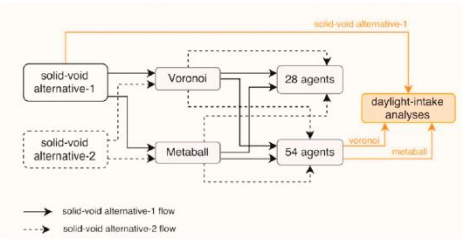
Cas d'étude théorique
Pas de masques solaire
Etude d'un détail

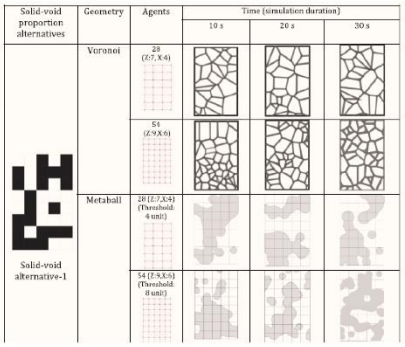
Modèle génératif
Swarm intelligence
Grasshopper, plugin Locust
Voronoi et Metaball

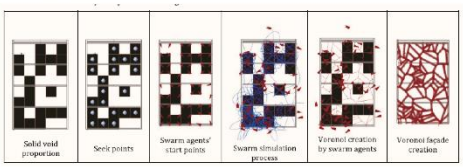
Facteur de lumière du jour
Eclairement utile
Autonomie en lumière du jour
Autonomie spatiale en éclairage naturel
Simulations annuelle.

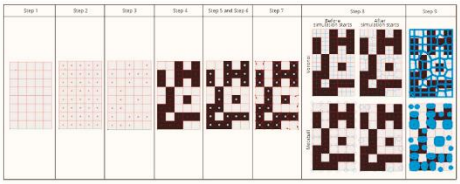
Différents scénarios ont été évalués

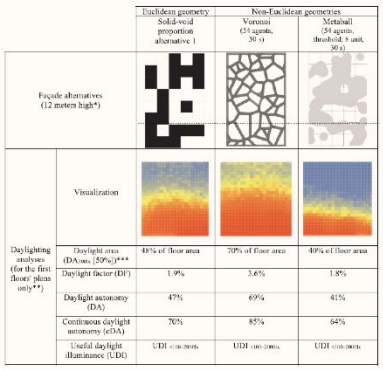
L'usage d'un processus génératif à défaut d'un modèle paramétrique classique ne se justifie que pour des questions esthétiques.











Source: Agirbas, Asli. «Façade form-finding with swarm intelligence.» Automation in Construction 99 (2019): 140-151.

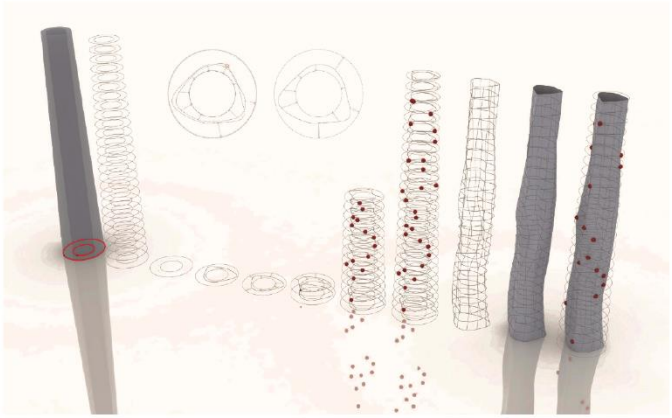
Cas d'étude théorique
Pas de masques solaire

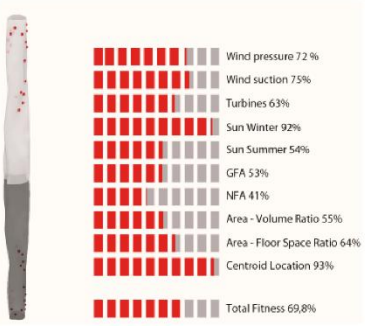
Modèle paramétrique

10 critères au total
des critères géométriques:
surface nette et brute ou le ratio volume/surface
Radiation solaire hiver/été
pressions ou de succion des vents

Algorithme évolutionnaire
méthode agrégée

Beaucoup de tests ont du être réalisé pour trouver les bon paramètrages et notamment les bons coefficients de pondération. D'après les auteurs, c'est en jouant avec ces coefficients que l'on peut comprendre le comportement de l'algorithme et ainsi qualibrer correctement les paramètres. Cette phase de tests est très chronophage.





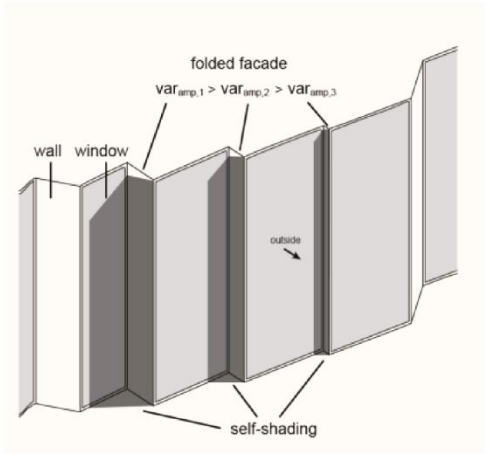
Sources: Parascho, Stefana, et al. «Design tools for integrative planning.» (2013).

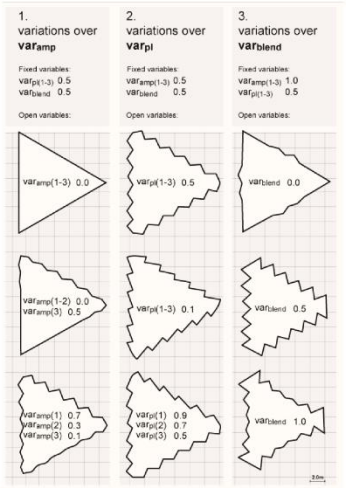
Cas d'étude théorique
Pas de masques solaire

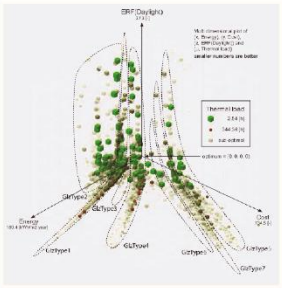
Modèle paramétrique

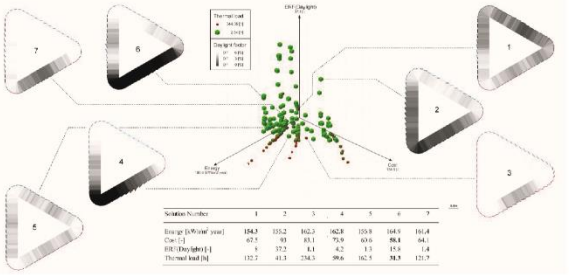
Consommations d'énergie
Facteur de lumière du jour
Confort thermique intérieur
Coût

Solveur Octopus
Algorithme SPEA2
32 générations de 300 individus
30 s de calcul par solutions









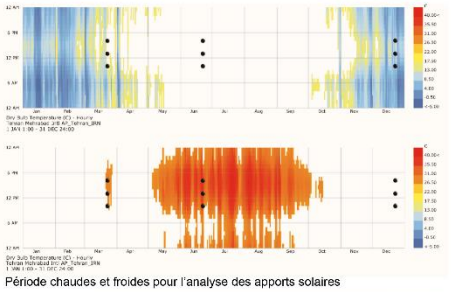
Sources: Negendahl, Kristoffer, and Toke Rammer Nielsen. «Building energy optimization in the early design stages: A simplified method.» Energy and Buildings 105 (2015): 88-99.

Cas d'étude théorique
Façade adaptative

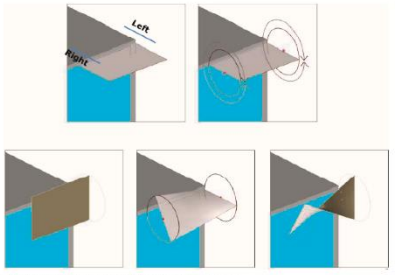
Modèle paramétrique
Grasshopper
Panélisation

Apports solaires été vs hiver
Eclairement naturel aux solstice et équinoxe à 10h, 13h, et 16h
Météo Téhéran, Iran

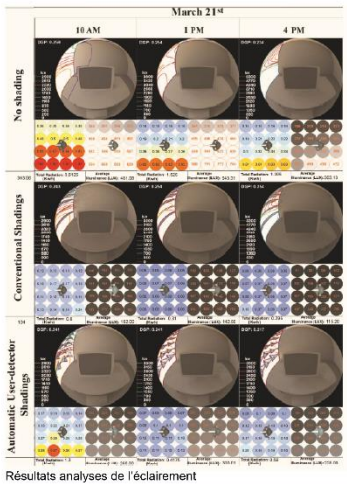
Solver Octopus



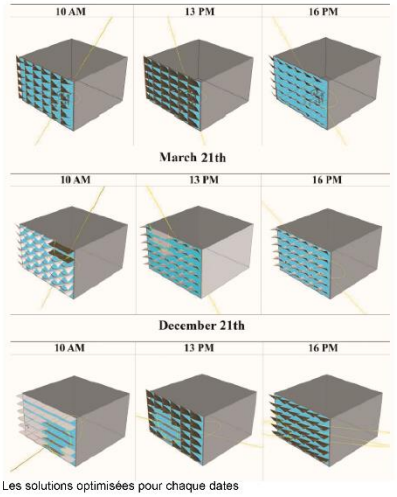
Période chaudes et froides pour l'analyse des apports solaires



Les paramètres de déformation du panneau



Résultats analyses de l'éclairement



Les solutions optimisées pour chaque dates

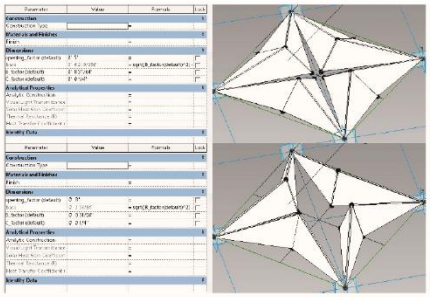
Rizi, R. A., & Eltaweel, A. (2021). A user detective adaptive facade towards improving visual and thermal comfort. Journal of Building Engineering, 33, 101554.

Cas d'étude théorique
Masque solaire (voisin)

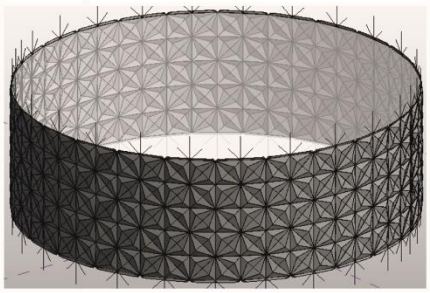
Modèle paramétrique,
Origami
Dynamo

Analyse de l'ensoleillement basé sur le diagramme solaire d'Alexandrie, Egypte
Simulation d'éclarage, méthode non précisée

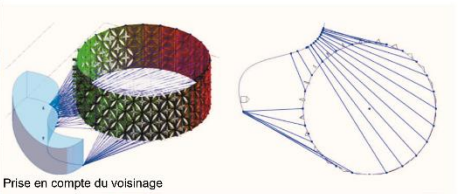
Algorithme Refinery (NSGAII)
20 generations de 48 individus



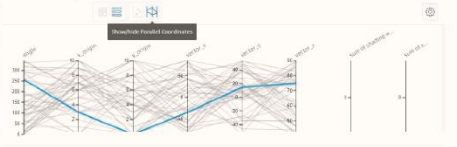
Paramètres du panneau

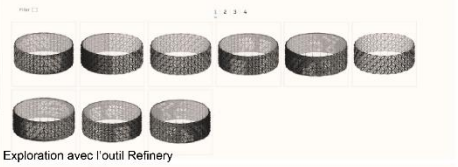


Application sur un bâtiment cylindrique

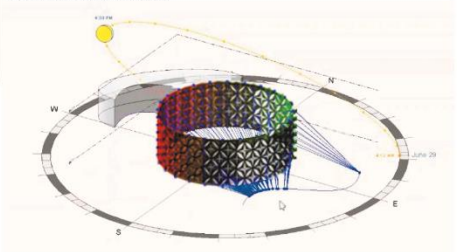


Prise en compte du voisinage





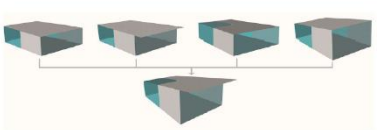
Exploration avec l'outil Refinery

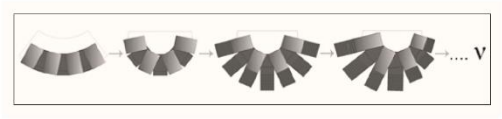


Solution sélectionnée

Fathy, A. G., El-Sayad, Z. E. Y. A. D., Saadallah, D., & Bakr, A. F. (2021). BIMPO: a generative parametric technique for building envelope design. Building Information Modelling (BIM) in Design, Construction and Operations IV, 205, 127.

Cas d'étude théorique
Assemblage d'unité de logements pour immeuble de grance hauteur
Pas de masques solaire
Etude d'un détail

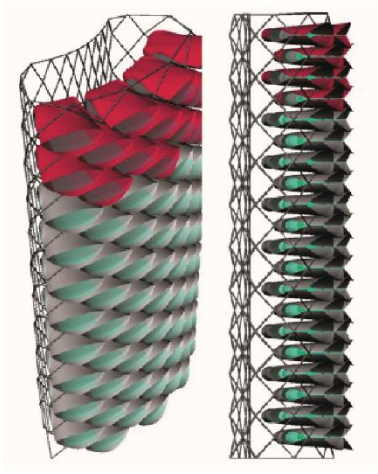


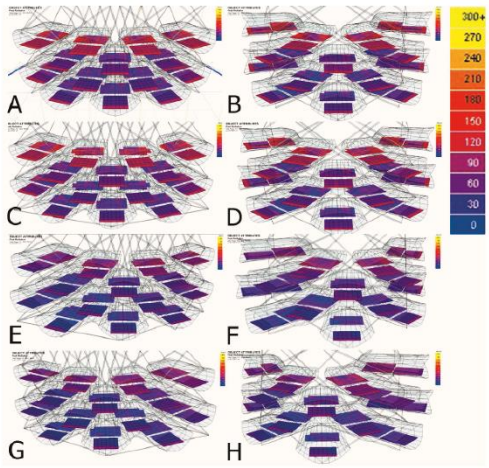


paramétrique (grasshopper)
paramètres d'orientation et de profondeur des logements

éclairement naturel
radiation solaire
21/03, 21/06 et 21/12
à 9h 12h, et 19h.

8 scénarios ont été évalué et comparé





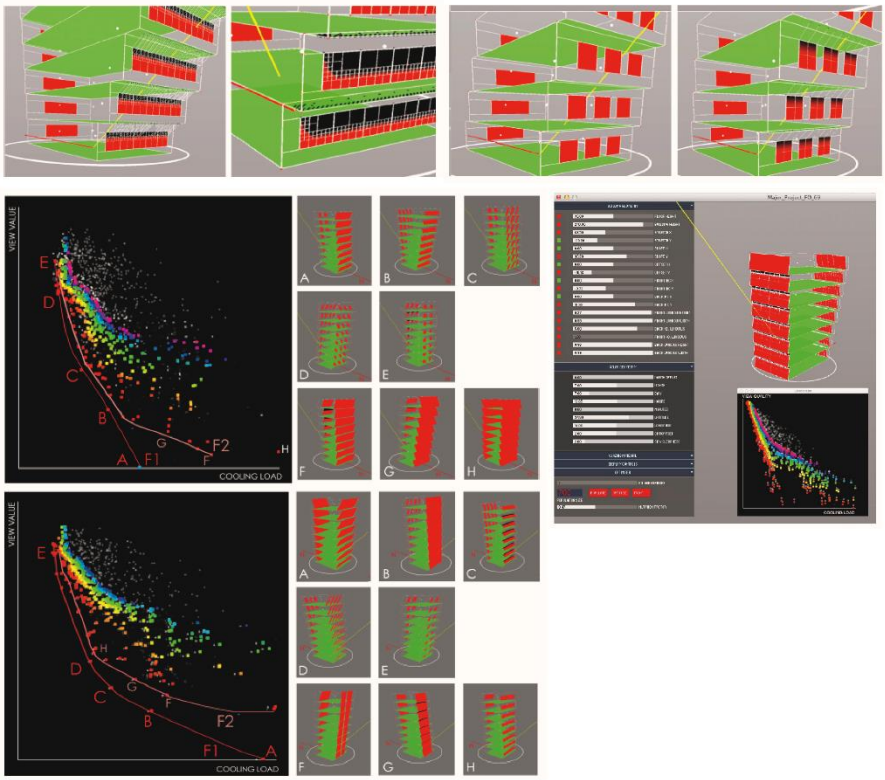
Source: Sarvani, Vasiliki, and Odysseas Kontovourkis. «Parametric design of a high-rise habitation unit system through lighting and solar energy performances.» Journal of Sustainable Architecture and Civil Engineering 3.4 (2013): 9-18.

Cas d'étude théorique
Pas de masques solaire

Modèle paramétrique

Besoins énergétiques en refroidisement en été
Qualité de la vue évaluée à l'aide la méthode «pixel-scoring» qui évalue pour chaque fenêtres à 4 endroits différents de la pièce les nuances de gris (le noir correspond au score maximum, et le blanc au score minimum).

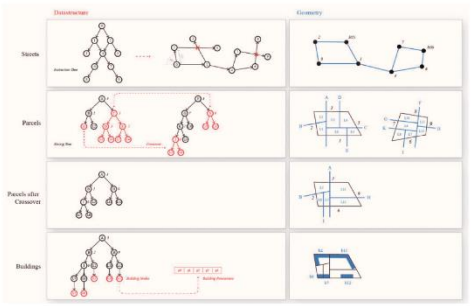
Algorithme NSGA-II
5 générations de 450 individus.



Sources: Conti, Zack Xuereb, Paul Shepherd, and Paul Richens. «Multi-objective Optimisation of Building Geometry for Energy Consumption and View Quality.» (2015).

Expérimentation à Weimar en Allemagne

Méthode générative
Système procédurale avec structure en arborescence
Grasshopper
Plugin DeCodingSpaces
Optimisation en deux étapes:
1/ réseau viaire et parcellaire
Réduction de l'espace de solution nécessaire pour assurer la convergeance de l'algorithme
2/ les typologies de bâtiments (après selection d'une solution dans l'étape 1)



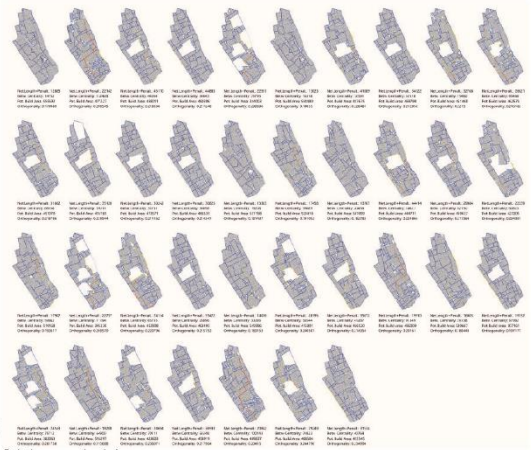
Méthode générative

Réseaux de rues générés aléatoirement

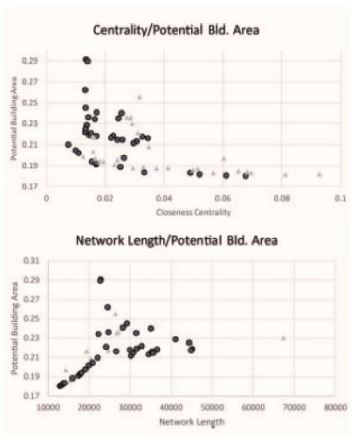
4 critères pr le réseau viaire et le parcellaire:
Longueur du réseau viaire
Surface à batir potentielle
Orthogonalité (angle des rues)
Centralité

2 critères pour le bâti:
maximiser la densité urbaine
minimiser la hauteur moyenne des bâtiments

Algorithme HypE



Solutions non dominées



Analyse des résultats

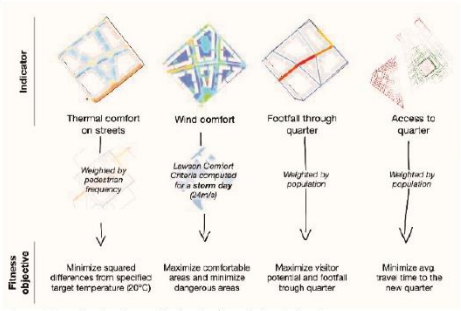
Koenig, R., Miao, Y., Aichinger, A., Knecht, K., & Konieva, K. (2020). Integrating urban analysis, generative design, and evolutionary optimization for solving urban design problems. Environment and Planning B: Urban Analytics and City Science, 47(6), 997-1013.

Expérimentation sur un quartier de Vienne, Autriche
Six îlots type

Modèle génératif
Génération du réseau viaire et des bâtiments
Grasshopper
Plugin DecodingSpaces a été utilisé pour générer le réseau viaire et les îlots

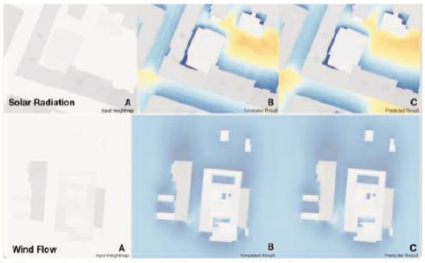
Accesibilité
Microclimat : radiation solaire + vitesse des vents
Méthode simplifiée à l'aide du Machine Learning (GAN)

Algorithme génétique
Solveur Octopus



Formulation des fonctions d'évaluation à partir des indicateurs


Paramètres géométrique intérrogés (orientation de la rue centrale, emplacement des ponts, hauteur et largeurs des bâtiments)



Méthodes évaluation de la radiation solaire et de la vitesse des vents
(A: géométrie, B: modèle physique, C: modèle simplifié)

Exemples de solutions générées

Duering, S., Chronic, A., & Koenig, R. (2020, May). Optimizing Urban Systems: Integrated optimization of spatial configurations. In Proceedings of the 11th Annual Symposium on Simulation for Architecture and Urban Design (pp. 1-7).

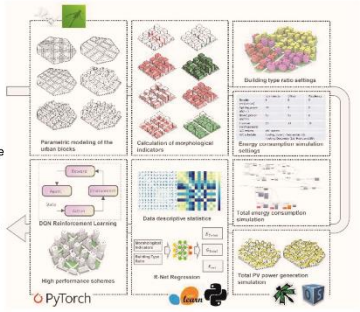
Cas d'étude théorique

Modèle parametrique
Grasshopper
3 topologies: îlots ouverts, îlots fermés, îlots mixtes
Le réseau viaire est une donnée d'entrée

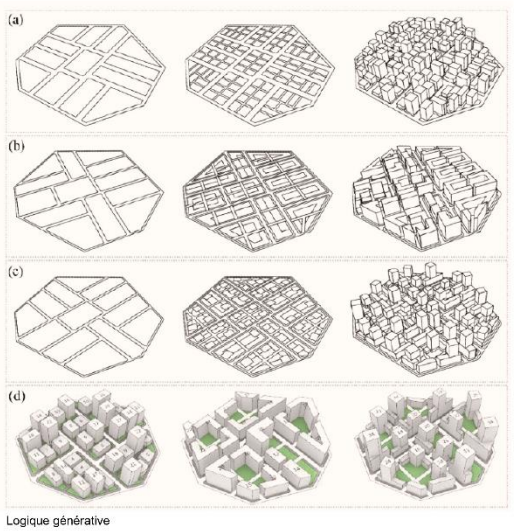
Consommations énergétiques à l'échelle urbain avec le plugin Dragonfly

Production d'énergie (PV)

Algorithme d'apprentissage par renforcement (Deep Q learning)



Méthode générale d'optimisation



Logique générative


Meilleures solution pour les trois types d'îlots

HUANG, C., ZHANG, G., YIN, M., & YAO, J. Energy-driven intelligent generative urban design.

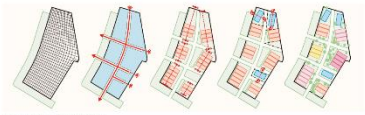
Expérimentation dans un contexte professionnel sur un quartier résidentiel de 7000 m² à Alkmaar aux Pays-Bas

Modèle paramétrique

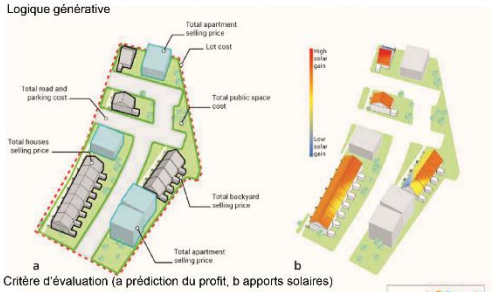
Radiation solaire en toiture

Prédiction du profit:
prix de vente - coût du projet
prix de vente = maisons + appartements
coût du projet = foncier + construction. développement + frais de vente et location + facteurs de risques

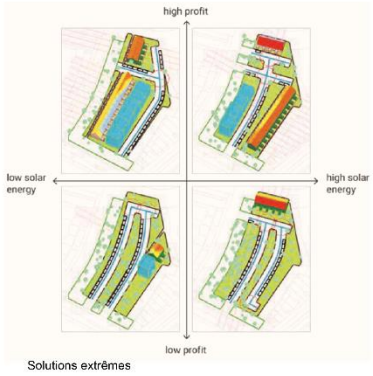
Algorithme génétique
NSGAII
200 générations de 200 individus



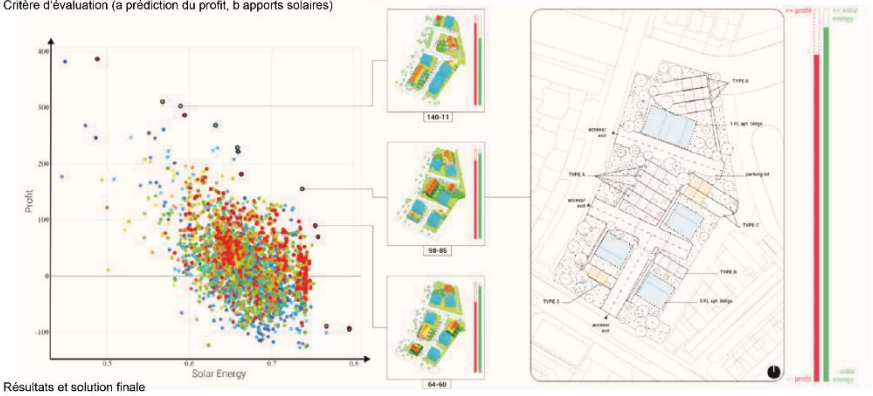
Logique générative



Critère d'évaluation (a prédiction du profit, b apports solaires)



Solutions extrêmes



Résultats et solution finale

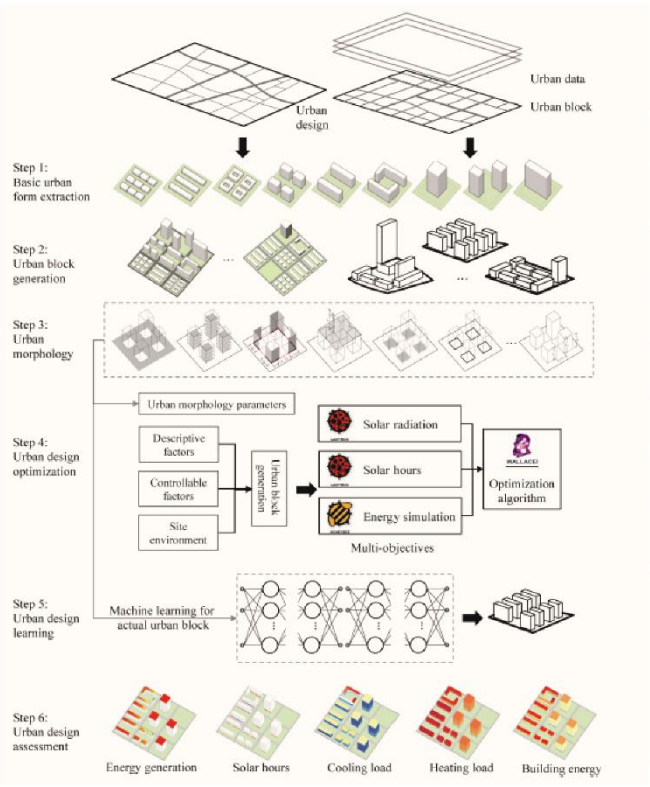
Nagy, D., Villaggi, L., & Benjamin, D. (2018, June). Generative urban design: integrating financial and energy goals for automated neighborhood layout. In Proceedings of the Symposium for Architecture and Urban Design Design, Delft, the Netherlands (pp. 265-274).

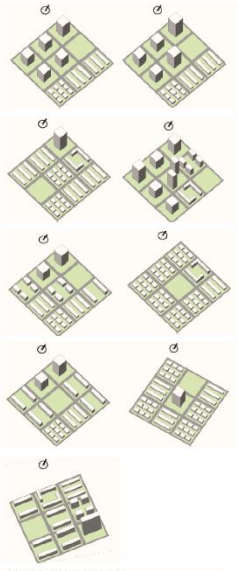
Cas d'étude théorique
Météo Jianhu, Chine

Modèle paramétrique
Géométrie simple
Grasshopper
9 typologies d'îlots

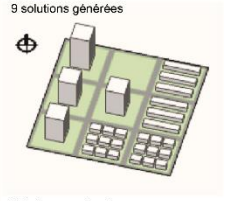
Radiation solaire
Heures d'ensoleillement
Consommations en refroidissement
Consommations en chauffage
Prise en compte du micro-climat dans les simulation énergétiques.
Utilisation du machine Learning pour réduire les calculs de performance énergétique

Algorithme évolutionnaire
Solveur Wallacei
NSGAII
240h de calcul
100 générations
297 solutions non dominées (136 en supprimant les doublons)
4821 solutions évaluées





9 solutions générées



Solution non-dominee

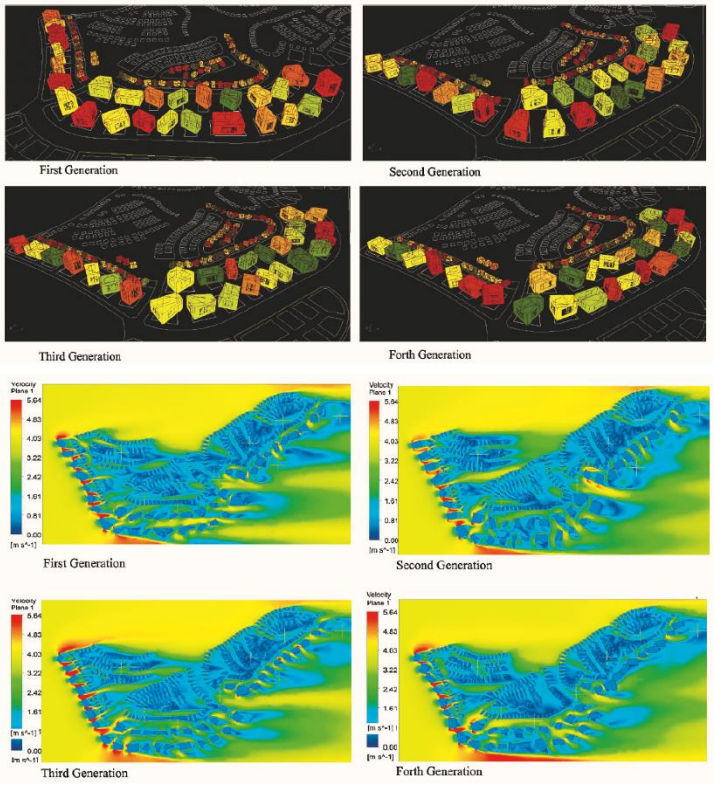
Wang, W., Liu, K., Zhang, M., Shen, Y., Jing, R., & Xu, X. (2021). From simulation to data-driven approach: A framework of integrating urban morphology to low-energy urban design. Renewable Energy, 179, 2016-2035.

Expérimentation à Dibai pour la Dubai Silicon Oasis

Modèle paramétrique
Grasshopper
Deux paramètres pour chaque bâtiments: la hauteur du bâtiment qui affectent aussi son échelle pour conserver le volume et l'orientation du bâtiment

Analyse de la vitesse des vents pour le confort thermique
Radiation solaire pour la réduction des consommations de refroidissement

Galapagos
Deux optimisation distinctes, une par critère
4 générations



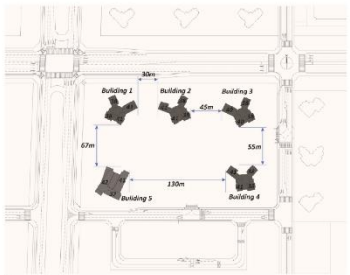
Taleb, H., & Musleh, M. A. (2015). Applying urban parametric design optimisation processes to a hot climate: Case study of the UAE. Sustainable Cities and Society, 14, 236-253.

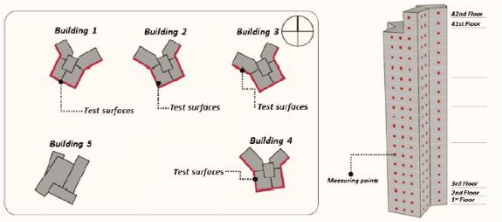
Expérimentation sur des immeubles de Logements Seoul, Corée du Sud

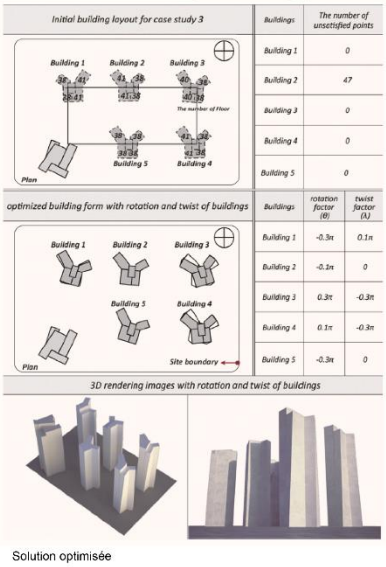
Modèle génératif à base d'agents pour génrer des formes complexes sans explosion combinatoire
Grasshopper
Les agents sont des points qui controlent d'autres points (sous-agents)

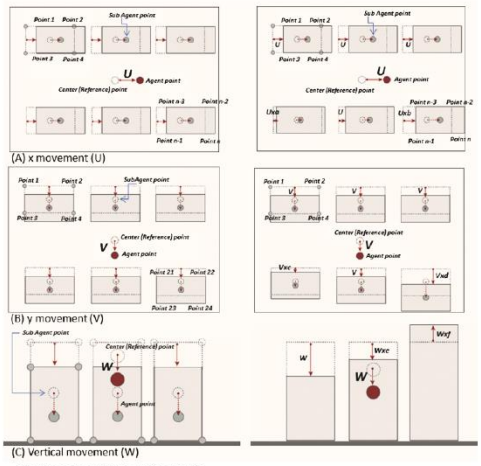
Heures d'ensoleillement

Algorithme génétique
Solveur Galapagos


Cas d'étude







Système de controle par les points

Solution optimisée

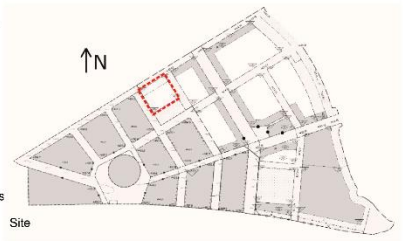
Yi, Y. K., & Kim, H. (2015). Agent-based geometry optimization with Genetic Algorithm (GA) for tall apartment's solar right. Solar Energy, 113, 236-250.

Cas d'étude théorique
Logements collectifs à Kiruna, Suède

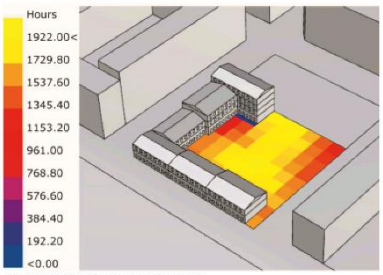
Modèle paramétrique
Grasshopper

Heures d'ensoleillement
Qualité de vue
Surface habitable
Surface des espaces extérieurs
Complexité géométrique

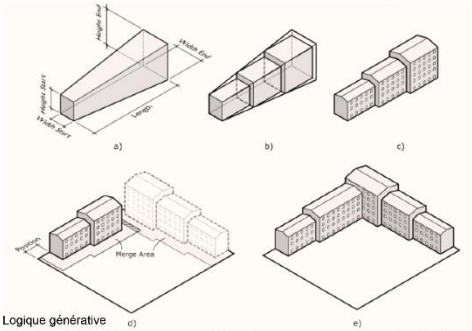
Aléatoire



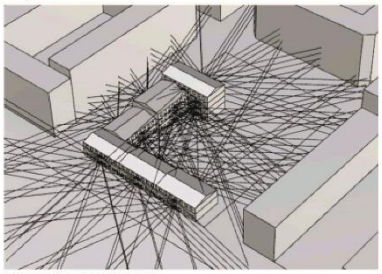
Site

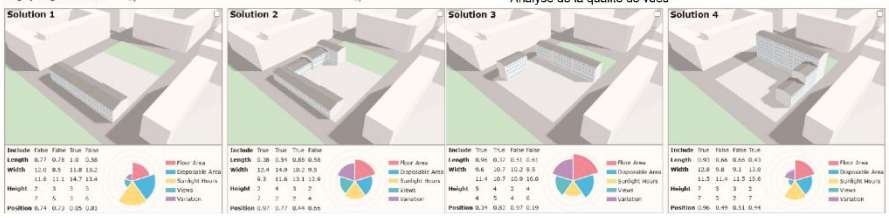


Analyse des heures d'ensoleillement



Logique générative


Analyse de la qualité de vues



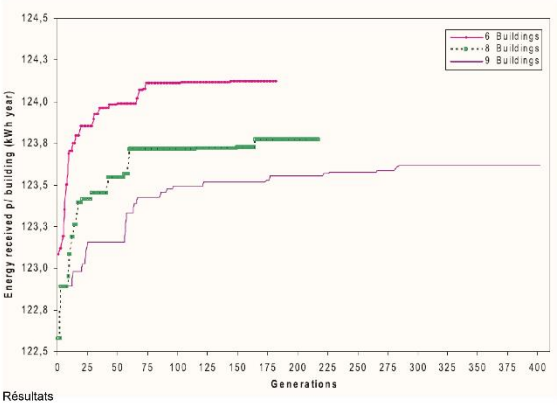
Mukkavaara, J., & Sandberg, M. (2020). Architectural design exploration using generative design: framework development and case study of a residential block. Buildings, 10(11), 201.

Cas d'étude théorique

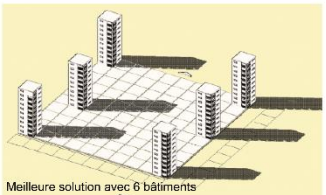
Modèle paramétrique
Trame orthogonale régulière de cellule de 10 m de côté.
Trame de 10x10.
6, 8, 9 ou 10 bâtiments de 30 m de haut

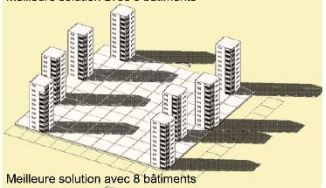
Radiation solaire
Maximiser les apports solaires en toiture et sur les facades

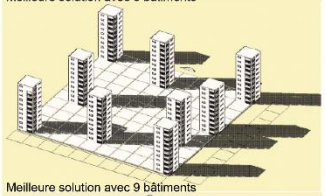
Algorithme génétique

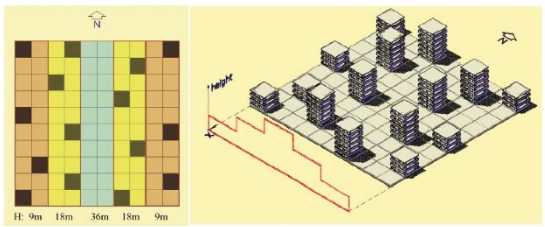


Résultats

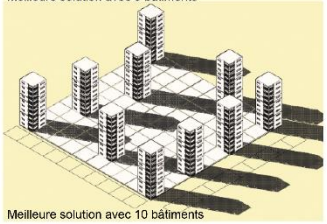

Meilleure solution avec 6 bâtiments


Meilleure solution avec 8 bâtiments


Meilleure solution avec 9 bâtiments



Meilleur solution avec 15 bâtiments de taille différentes


Meilleure solution avec 10 bâtiments

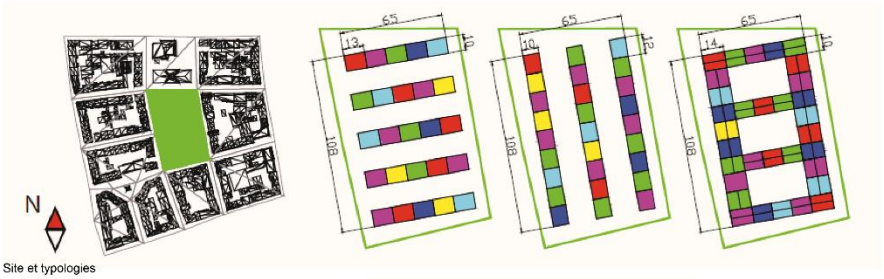
António, C. A. C., Monteiro, J. B., & Afonso, C. F. (2014). Optimal topology of urban buildings for maximization of annual solar irradiation availability using a genetic algorithm. Applied thermal engineering, 73(1), 424-437.

Expériementation sur un îlot de Bâle, en Suisse

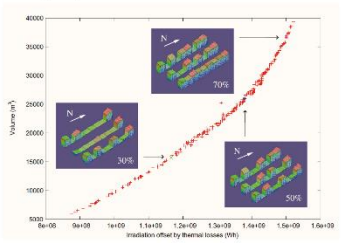
Modèle paramétrique
Trois typologies étudiée (toit terrasse, toiture inclinée, patio)
Seule la hauteur des bâtiments est variable.

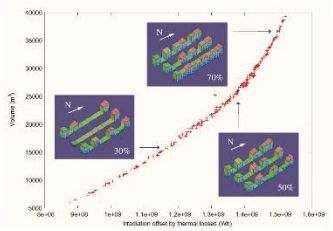
Radiation solaire
Volume construit

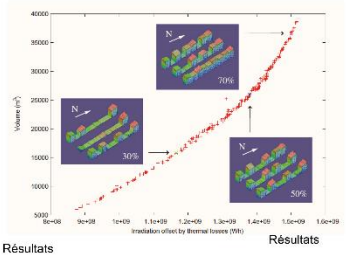
Algorithme génétique



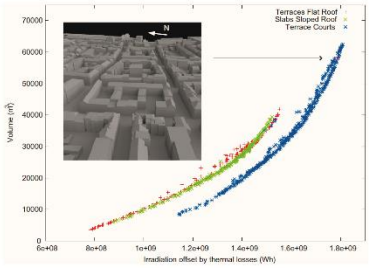
Site et typologies







Résultats



Résultats

Kämpf, J. H., Montavon, M., Bunyesc, J., Bolliger, R., & Robinson, D. (2010). Optimisation of buildings' solar irradiation availability. Solar energy, 84(4), 596-603.

Cas d'étude théorique
Développement d'un outil pour la génération d'îlots urbains EcoGen

Modèle génératif
Approche voxellaire

Facteur de lumière du jour
Consommation en chauffage
Apports solaire et courtoisie solaire
Facteur de compacité de la forme

Algorithme génétique

Interface de l'outil


Solution optimisée pour les consommations de chauffage

Solution optimisée pour le compromis entre lumière et compacité

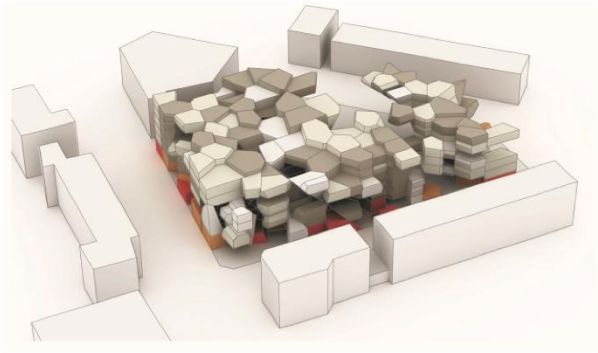
Marsault, X., & Torres, F. (2019, November). An interactive and generative eco-design tool for architects in the sketch phase. In Journal of Physics: Conference Series (Vol. 1343, No. 1, p. 012136). IOP Publishing.

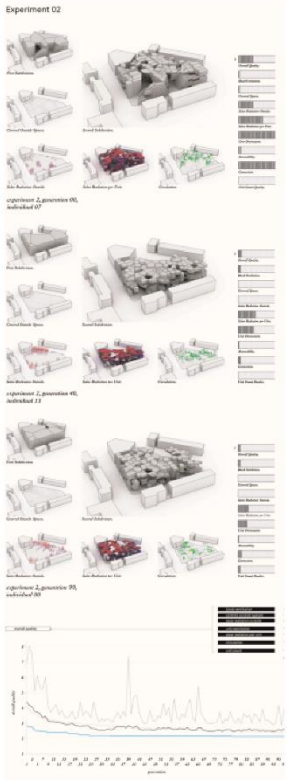
Cas d'étude théorique
Ilot urbain

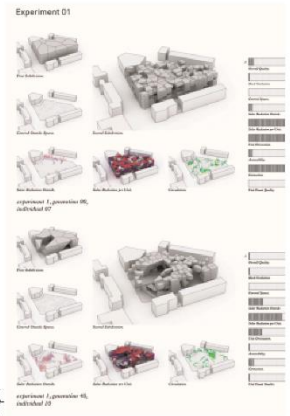
Modèle paramétrique
Approche voxellaire
Volumt et nombre d'unite fixe

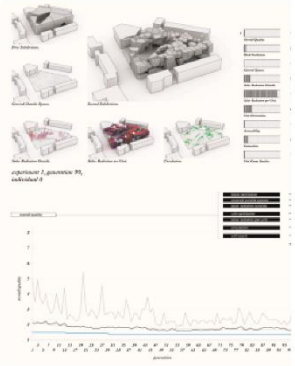
Ventilation
Radiation des espace extéieur
Radiation des voxels
Circulation
Surface extérieures couverte
Lumière naturelle

Optimisation multicritère
Algorithme évolutionnaire









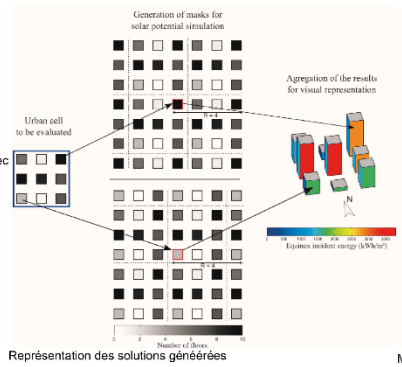
Menges, A. (2012). Biomimetic design processes in architecture: morphogenetic and evolutionary computational design. Bioinspiration & biomimetics, 7(1), 015003.

Cas d'étude théorique

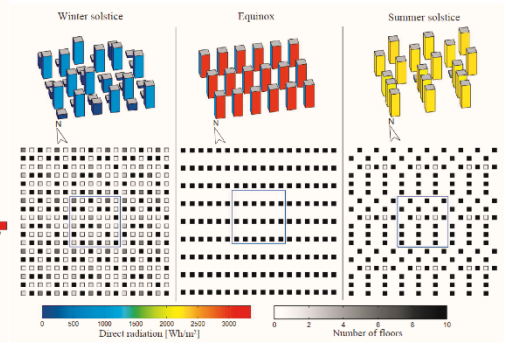
Modèle paramétrique
Une quantité variable de bâtiments carrés de hauteur variable chacun placé sur une celulle de 20 mètre de côté avec d'autres cellules pour contexte urbain

Radiation solaire

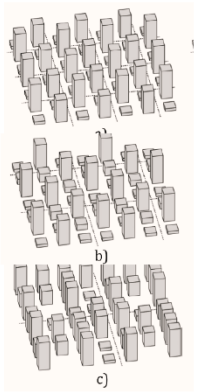
Algorithme évolutionnaire



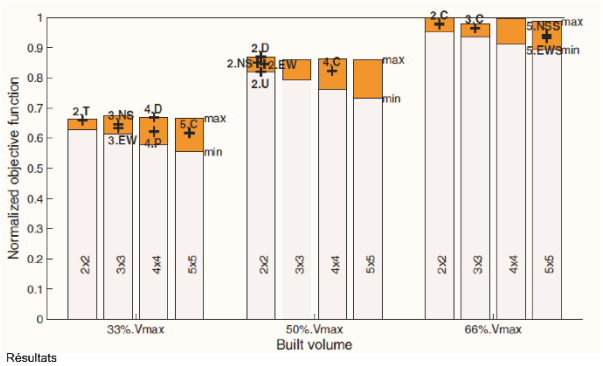
Représentation des solutions généérées



Meilleures solutions à différentes dates



Exemples de solutions générées



Résultats

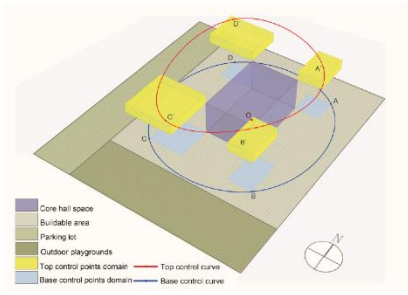
Vermeulen, T., Merino, L., Knopf-Lenoir, C., Villon, P., & Beckers, B. (2018). Periodic urban models for optimization of passive solar irradiation. Solar Energy, 162, 67-77.

Ca s d'étude théorique
Bâtiment résidentiel, Shenyang, Chine

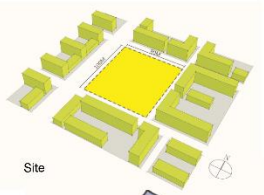
Modèle paramétrique
Grasshopper

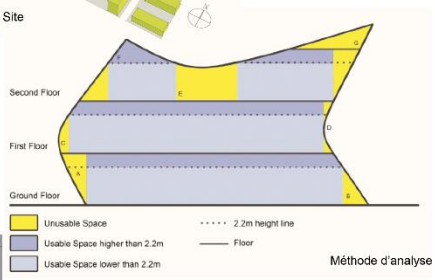
Radiation soalire
Coefficient de forme
Efficacité de la forme

Optimisation multicritère
Algorithme génétique
Solver Octopus
100 générations de 50 individus
84 solutions non-dominées

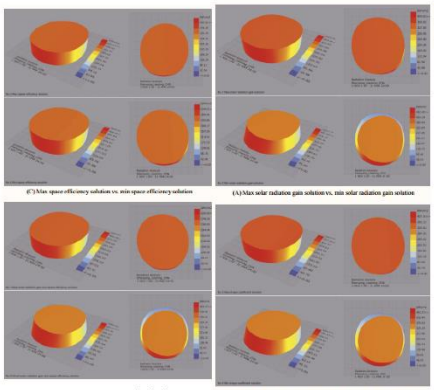


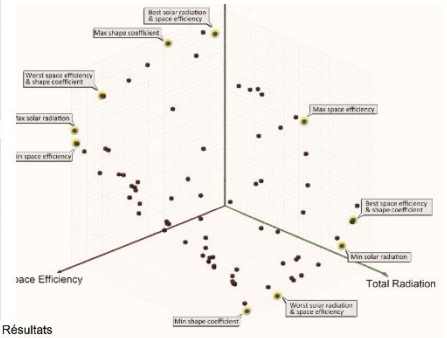
Paramètres


Site



Méthode d'analyse


Exemples de solutions générées



Résultats

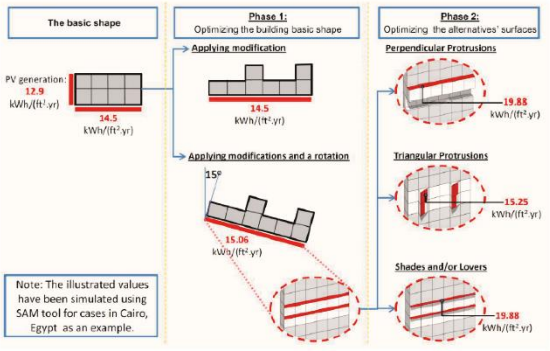
Zhang, L., Zhang, L., & Wang, Y. (2016). Shape optimization of free-form buildings based on solar radiation gain and space efficiency using a multi-objective genetic algorithm in the severe cold zones of China. Solar Energy, 132, 38-50.

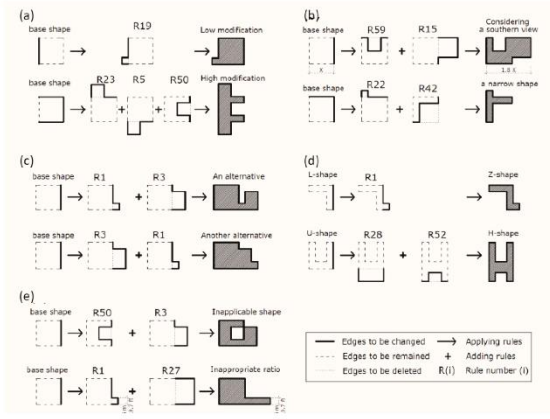
Cas d'étude théorique
Bâtiments commerciaux
Masque solaire (voisin)

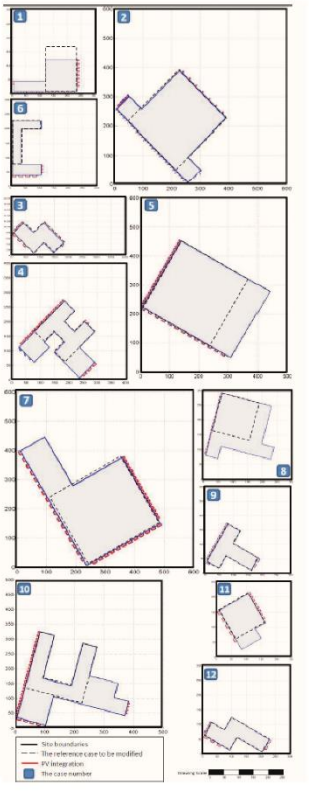
Modèle génératif,
Grammaire de forme
Matlab
Graphical User Interface

Radiation solaire
Consommations énergétiques
Météo Caire, Egypte

Deux phases d'optimisation
1/ la fome du bâtiment pour réduire les consommations énergétiques
2/ enveloppe pour l'intégration des PV







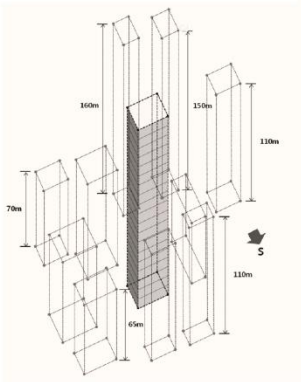
Youssef, A. M., Zhai, Z. J., & Reffat, R. M. (2018). Generating proper building envelopes for photovoltaics integration with shape grammar theory. Energy and buildings, 158, 326-341.

Cas d'étude théorique
Bâtiment commerciale de grande hauteur à Moscow, Russie

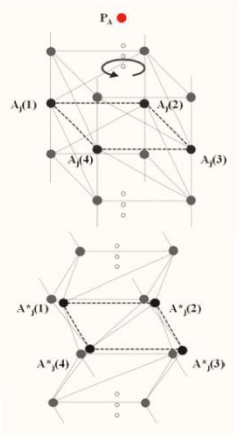
Modèle paramétrique
Grasshopper

Radiation solaire
Consommations énergétiques

Optimisation multicritère
Agrégée, Matlab

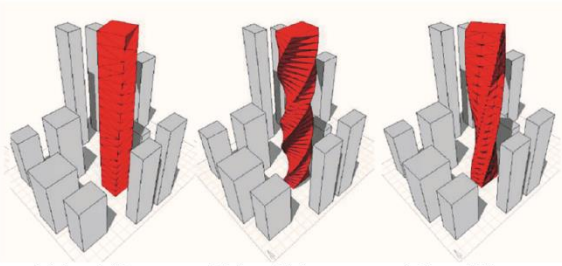


Contexte urbain



Logique générative

Gen.	individual1	individual2	individual3	individual4
1st	E: 498.80 R: 3.49	439.02 2.87	495.29 3.81	351.91 4.06
2nd	E: 341.33 R: 2.87	492.43 3.84	410.11 3.29	389.37 3.72
3rd	E: 391.16 R: 3.73	405.83 3.61	445.29 3.32	360.83 2.87
4th	R: 391.8 E: 3.95	401.5 3.46	310.8 2.99	461.33 3.01
27th	E: 312.54 R: 2.91	326.37 2.87	192.12 2.48	299.93 2.64
28th	E: 199.53 R: 2.46	195.43 2.47	270.86 2.51	206.97 2.46



Solution optimisée pour les consommations énergétiques

Solution optimisée pour la radiation solaire

Solution optimisée pour les deux

Yi, H. (2014). Automated generation of optimised building envelope: simulation based multi-objective process using evolutionary algorithm. International Journal of Sustainable Building Technology and Urban Development, 5(3), 159-170.

Expérimentation sur une tour de bureau à Kebayoran Lama
Ombres propres

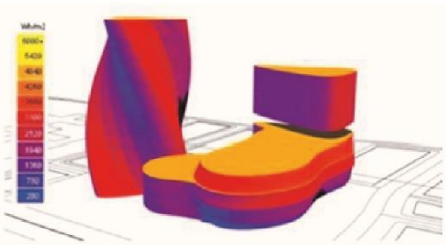
Modèle paramétrique
Grasshopper

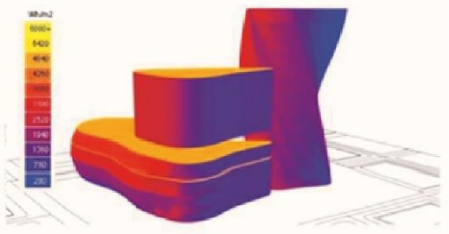
Radiation soleire
Heures d'ensoleillement

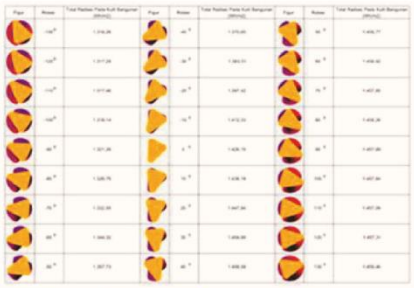
Aléatoire

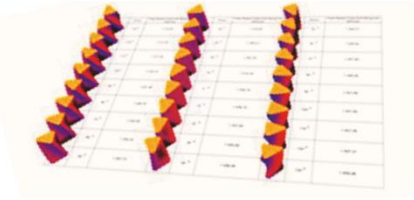
Une approche paramétrique en plusieurs phases intégrant des critères de performances du découpage de la parcelle à la conception des façades.

1/ optimisation des chemins piétons pour le découpage de la parcelle
2/ Tortion des masses pour réduire la radiation solaire pour un meilleur confort thermique en été
3/ Analyse de la radiation et de l'ensoleillement pour déterminé l'emplacement de l'espace public.









Sources: Suyoto, William, Aswin Indraprastha, and Heru W. Purbo. «Parametric approach as a tool for decision-making in planning and design process. Case study: Office tower in Kebayoran Lama.» Procedia-Social and Behavioral Sciences 184 (2015): 328-337.

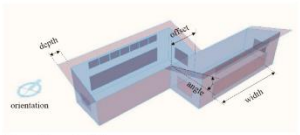
Cas d'étude théorique
Salle de classe, Los Angeles, USA

Modèle paramétrique

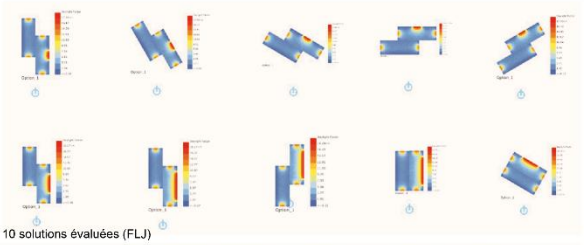
Consommations énergétique
Facteur de lumière du jour
Qualité de vue
Ligne directe de vue
Cout de construction
Cout d'explotation
Cout du cycle de vie
Emission de CO^2

Evaluation de 1296 scénarios
16h de calcul

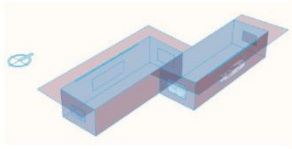
Haymaker, J., Bernal, M., Marshall, M. T., Okhoya, V., Szilasi, A., Rezaee, R., ... & Welle, B. (2018). Design space construction: a framework to support collaborative, parametric decision making. Journal of Information Technology in Construction (ITcon), 23(8), 157-178.



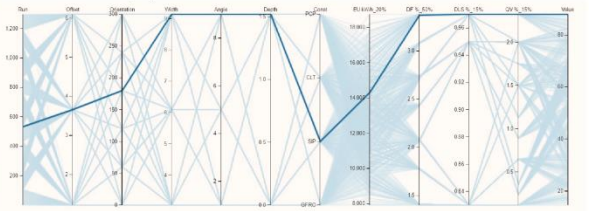
Paramètres étudiés

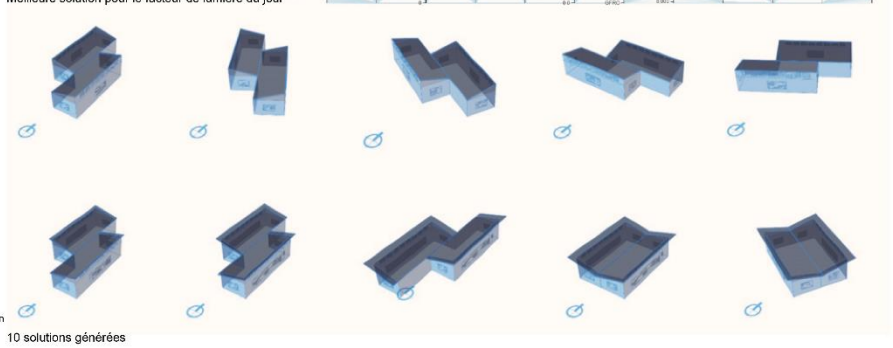


10 solutions évaluées (FLJ)



Meilleure solution pour le facteur de lumière du jour





10 solutions générées

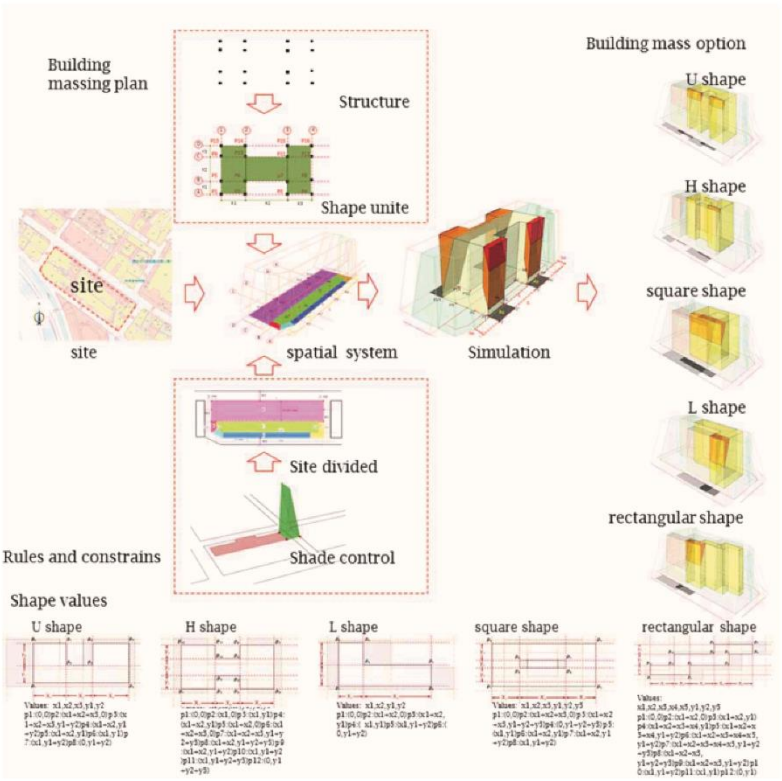
Expérimentation sur une parcelle à Taipei, Taiwan

Modèle paramétrique
Grasshopper
Intégration de contraintes urbaines

Surface de plancher maximum
Surface ombrée

Optimisation multicritère
Agrégation
Solver Galapagos
Algorithme génétique

Huang, Y. S., Chang, W. S., & Shih, S. G. (2015). Building massing optimization in the conceptual design phase. Computer-Aided Design and Applications, 12(3), 344-354.

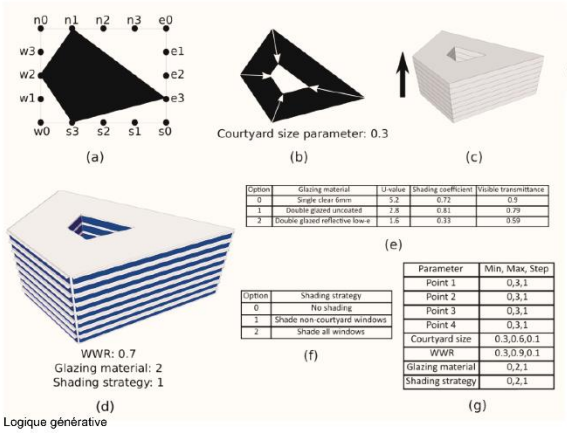


Cas d'étude théorique
Bâtiment à Singapore

Modèle paramétrique

Eclairement utile
Consommations en refroidissement

Optimisation multicritère
Algorithme NSGAII

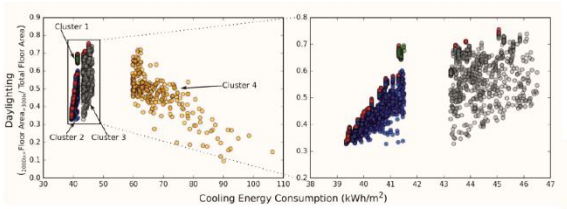


Option	Glazing material	U-value	Shading coefficient	Visible transmittance
0	Single clear 6mm	5.2	0.72	0.9
1	Double glazed uncoated	2.8	0.81	0.79
2	Double glazed reflective low-e	1.6	0.33	0.59

Option	Shading strategy
0	No shading
1	Shade non courtyard windows
2	Shade all windows

Parameter	Min, Max, Step
Point 1	0,3,1
Point 2	0,3,1
Point 3	0,3,1
Point 4	0,3,1
Courtyard size	0.3,0.6,0.1
WWR	0.3,0.9,0.1
Glazing material	0,2,1
Shading strategy	0,2,1

Logique générative



Résultats

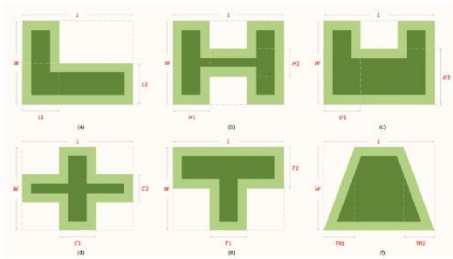
Chen, K. W., Janssen, P., & Schlueter, A. (2018). Multi-objective optimisation of building form, envelope and cooling system for improved building energy performance. Automation in construction, 94, 449-457.

Expérimentation sur une parcelle sur le campus de l'université de Tsinghua, Pékin, Chine

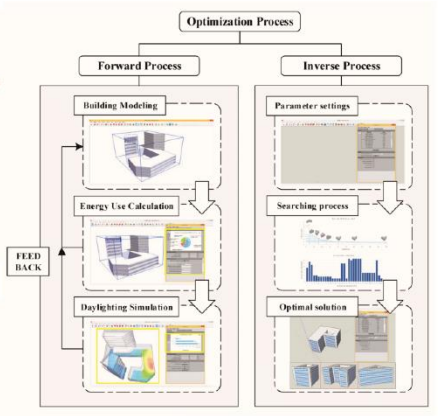
Modèle paramétrique

Consommations énergétique en éclairage, chauffage et refroidissement
Facteur de lumière du jour

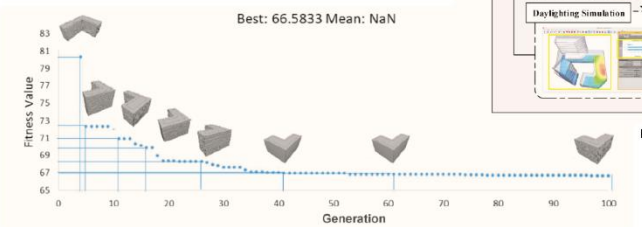
Optimisation multicritère
Algorithme génétique
Matlab

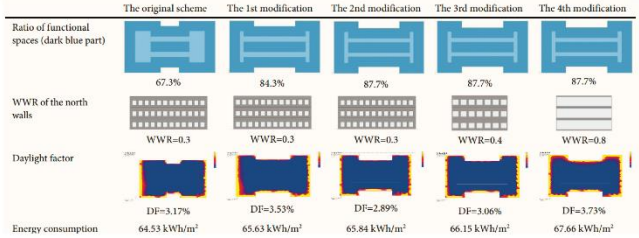


Logique générative

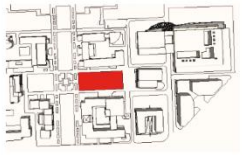


Processus d'optimisation





Résultats



Site

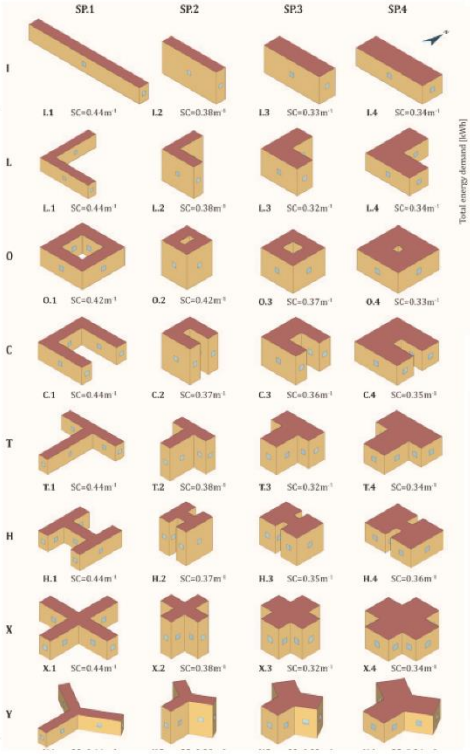
Li, Z., Chen, H., Lin, B., & Zhu, Y. (2018, August). Fast bidirectional building performance optimization at the early design stage. In Building Simulation (Vol. 11, No. 4, pp. 647-661). Tsinghua University Press.

Cas d'étude théorique
Bâtiment résidentiel
Méthode d'optimisation en 2 phases (forme puis dispositif passif)

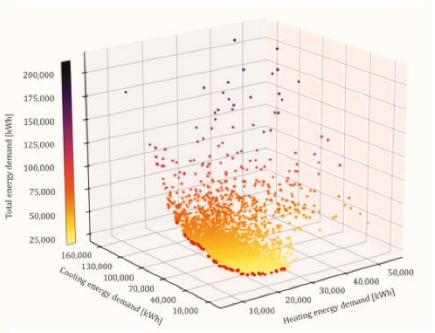
Modèle paramétrique
Forme et ratio de surface vitree

Consommationsen chauffage, refroidissement et totaux

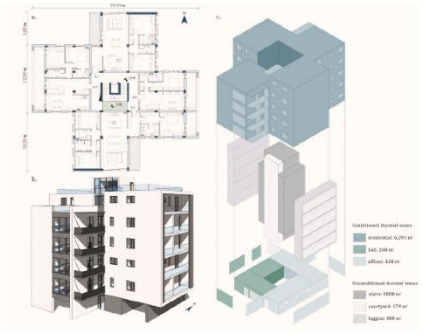
Optimisation multicritère
Algorithme NSGAII



Formes explorées



Résultats



Morphologie choisi pour une seconde phase d'optimisation

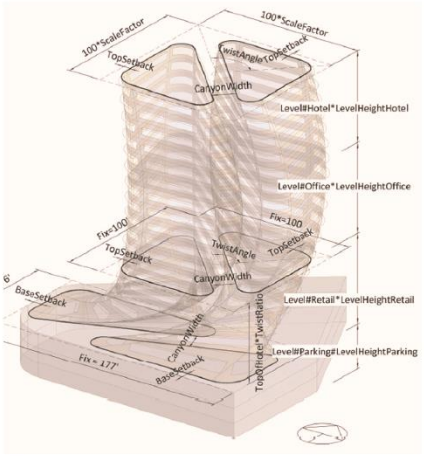
Ciardiello, A., Rosso, F., Dell'Olmo, J., Ciancio, V., Ferrero, M., & Salata, F. (2020). Multi-objective approach to the optimization of shape and envelope in building energy design. Applied energy, 280, 115984.

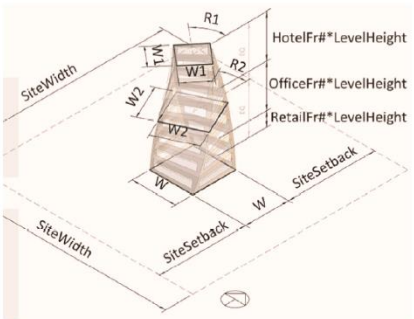
Cas d'étude théorique
Géométrie complexe
12 cas d'études

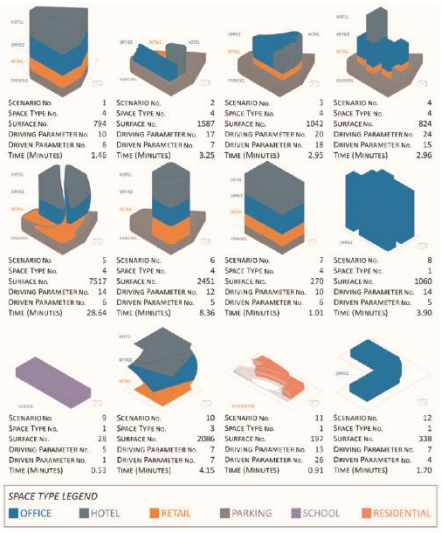
Modèle paramétrique
Outil développé pour Revit (H.D.S beagle)
Paramètre géométrique et thermiques

Evaluation du design
Coût financier
Consommations énergétiques

Optimisation multicritère
Algorithme génétique







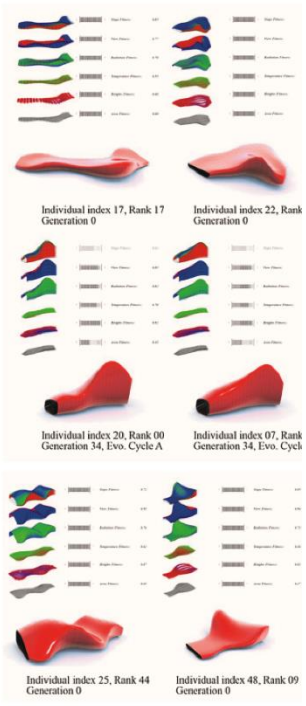
Gerber, D. J., & Lin, S. H. E. (2014). Designing in complexity: Simulation, integration, and multidisciplinary design optimization for architecture. Simulation, 90(8), 936-959.

Cas d'étude théorique
Forme libre pour Boracay; île aux Philippines

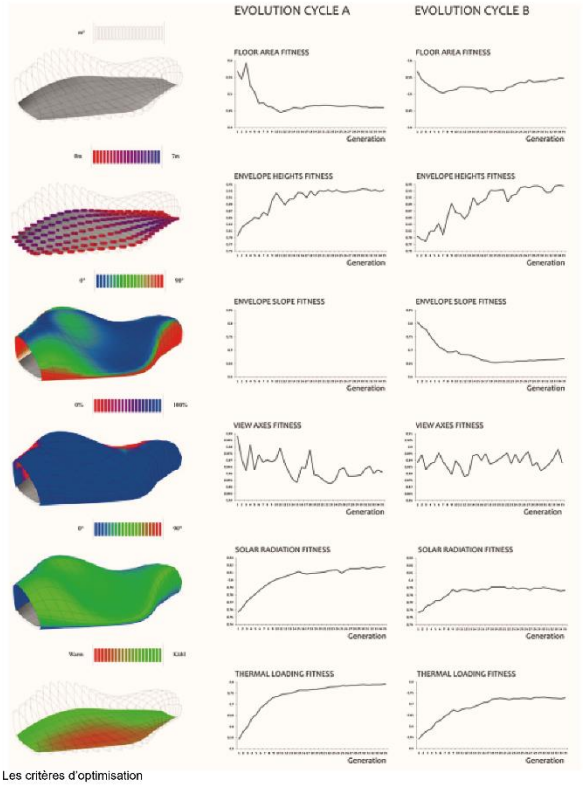
Modèle paramétrique

Radiation solaire
Accès à la vue
Pente de l'enveloppe
Charge thermique
Surface de plancher
Hauteur moyenne

Optimisation multicritère
Algorithme évolutionnaire



Exemple de 6 solutions générées



Les critères d'optimisation

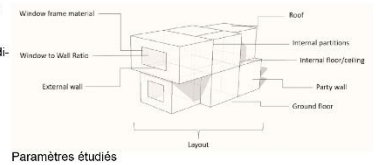
Menges, A. (2012). Biomimetic design processes in architecture: morphogenetic and evolutionary computational design. Bioinspiration & biomimetics, 7(1), 015003.

Cas d'étude théorique pour un configurateur en ligne (PLOOTO)
Applications à des maisons individuelles à Londres, RU

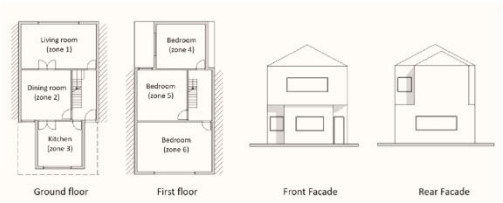
Modèle génératif
Python

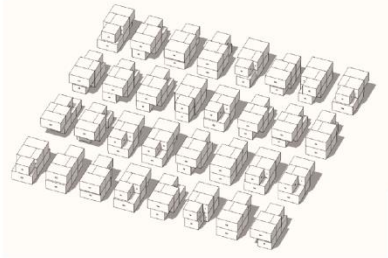
Consommations énergétiques
Coût du cycle de vie
Empreinte carbone du cycle de vie

Optimisation multicritère
Algorithme NSGAII



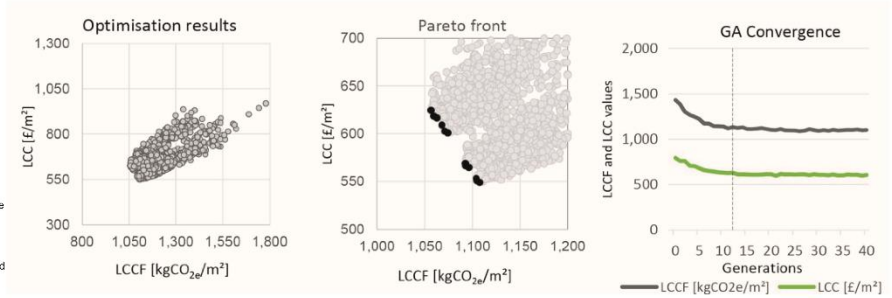
Paramètres étudiés





Exemple de solutions générées

Meilleure solution



Résultats

Schwartz, Y., Raslan, R., Korolija, I., & Mumovic, D. (2021). A decision support tool for building design: An integrated generative design, optimisation and life cycle performance approach. International Journal of Architectural Computing, 19(3), 401-430.
Schwartz, Y., Raslan, R., Korolija, I., & Mumovic, D. (2017). Integrated building performance optimisation: coupling parametric thermal simulation optimisation and generative spatial design programming. In 15th International Building Performance Simulation Association Conference, San Francisco, CA, Aug (pp. 7-9).

Cas d'étude théorique
Immeuble de logements
Assemblage d'appartements

Modèle génératif
Matrice d'adjacence
Grille orthogonale 3D
Processus addition
(des parties au tout)
Développé sur Matlab

Eclairement
Ensoleillement
Index de confort thermique
(PMV)

Optimisation multicritère
Algorithme recuit simulé

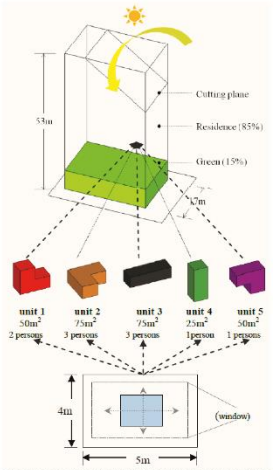
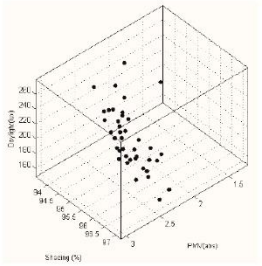
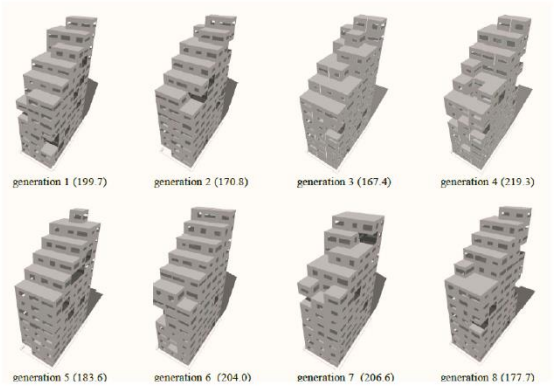


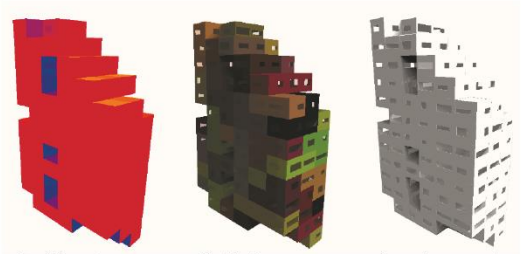
Diagramme représentant la formulation du problème



Résultats



Huit exemples de solutions générées



Ensoleillement | Combinatoire | forme et agencement optimisé

Solution finale

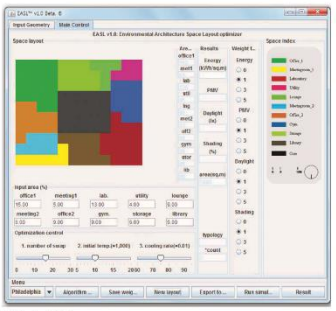
Yi, H., Yi, Y. K.. (2014, September). Performance Based Architectural design optimization: Automated 3D space Layout using simulated annealing. In Proceedings of the 2014 ASHRAE/IBPSA-USA Building Simulation Conference (pp. 292-299).

Expérimentation sur un projet réel
Plateau de bureau, Seoul
Assemblage de pièces

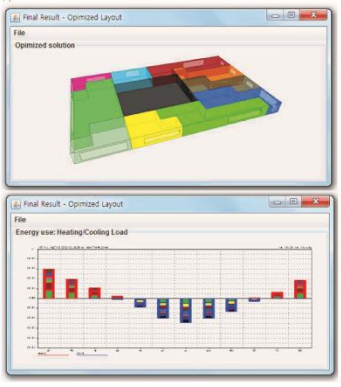
Outil spécifiquement développé (EASL) avec IHM
Modèle génératif
Grille orthogonale 2D
Processus soustraction
(du tout aux parties)

Consommations énergétiques
Eclairement
Ensoleillement
Index de confort thermique
(PMV)

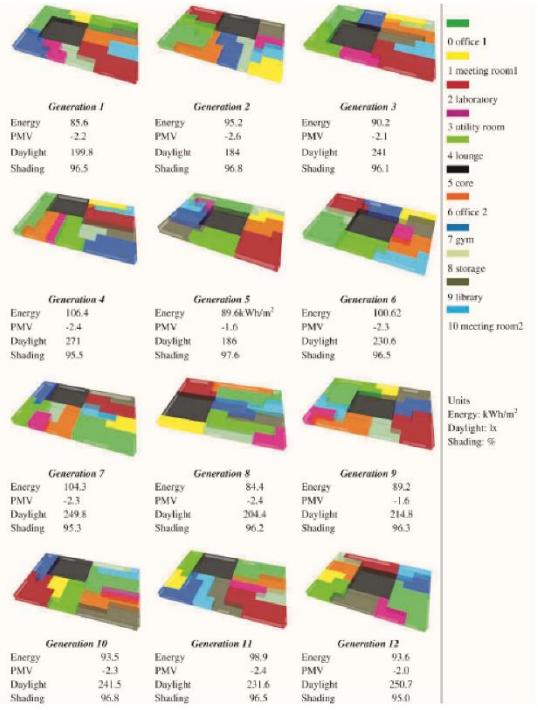
Optimisation multicritère
Algorithme recuit simulé



IHM de EASL



Solution optimisée



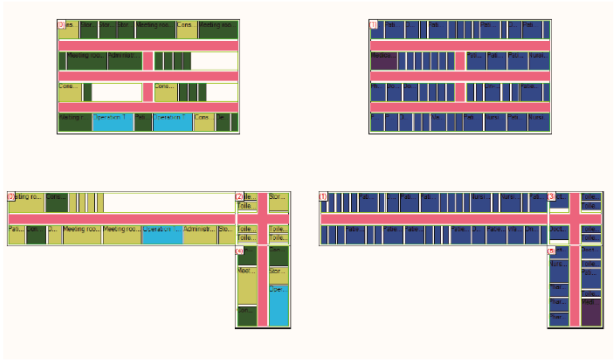
Exemples de 12 solutions générées

Yi, H. (2016). User-driven automation for optimal thermal-zone layout during space programming phases. Architectural Science Review, 59(4), 279-306.

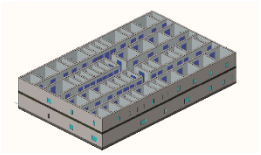
Cas d'étude théorique
Clinique médicale

Modèle génératif basé sur des règles
Outil spécifique (EDC, Early Design Configurator)
Approche BIM

Besoins et Consommations énergétiques en chauffage et refroidissement
Analyse de cycle de vie



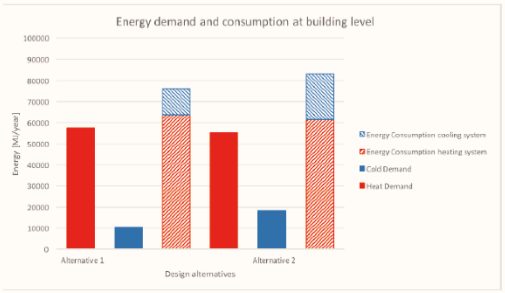
Deux scénarios alternatifs



Visualisation dans un logiciel BIM

Comparaison de deux scénarios uniquement

Visualisation des indicateurs



Résultats

Sleiman, H. A., Hempel, S., Traversari, R., & Bruinenberg, S. (2017). An assisted workflow for the early design of nearly zero emission healthcare buildings. Energies, 10(7), 993.

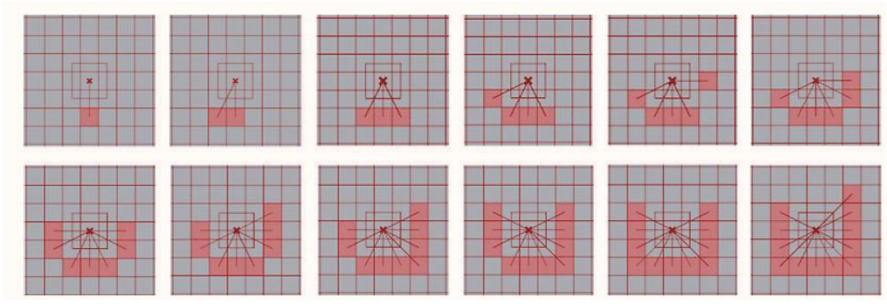
Cas d'étude théorique
Unité de soin infirmier
Assemblage de pièces

Modèle paramétrique avec régles et contraintes
Grasshopper
Modèle génératif
Grille orthogonale 2D
Processus soustraction
(du tout aux parties)

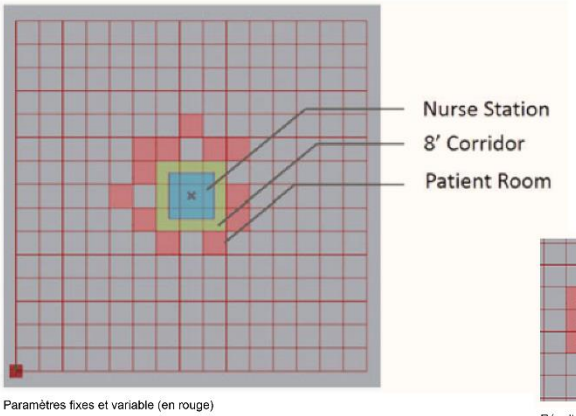
Eclairement
Distance de marche entre les chambres des patients et bureaux infirmières

Optimisation
Solver Galapagos
Algorithme génétique

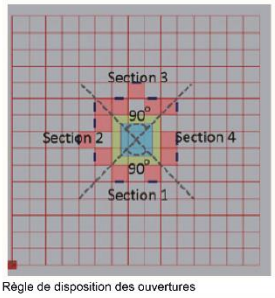
Il s'agit d'optimiser l'emplacement des chambre des patients.



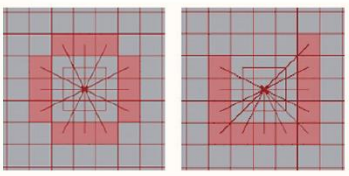
Processus génératif



Paramètres fixes et variable (en rouge)



Règle de disposition des ouvertures



Résultat final selon la pondération des critères

Su, Z., & Yan, W. (2015). A fast genetic algorithm for solving architectural design optimization problems. Ai Edam, 29(4), 457-469.

Annexe 2 Code de la fonction de réparation pour la cité du ministère de la justice de St Laurent de Maroni

```
"""Provides a scripting component.
    Inputs:
        lames 1: Première couche de lames (liste de courbes planes)
        lames2: Seconde couche de lames (liste de courbes planes)
        v1: Vecteur pour l'orientation des lames 1
        v2: Vecteur pour l'orientation des lames 2
        ep1: Epaisseur des lames 1 (nombre)
        ep2: Epaisseur des lames 2 (nombre)
        line1:Emplacement des lames 1 (ligne verticale)
        line2:Emplacement des lames 2 (ligne verticale)
        m: Maillage de la façade
        pts: Points lancer de rayon des gouttes de pluies
        vect:Rayons des gouttes de pluies
        n:nombre de rayons
        F:Fond de la façade (brep)
    Output:
        a: Première couche de lames réparée
        b: Seconde couche de lames réparée
        c:
        i:Nombre d'itération"""

import rhinoscriptsyntax as rs
import rhinoscriptsyntax as rs
import ghpythonlib.components as gh
import Grasshopper
import random
import time


def move_lames(lames, line):
    n = len(lames)
    pt= gh.DivideCurve(line, n+1, False)[0]
    t = gh.Line(gh.EndPoints(line)[0], pt)
    Lames= gh.Move(lames[0], t)[0]
    dist = gh.Length(gh.Line(pt[0], pt[1]))*5
    return Lames

def create_lames1(lames, v, ep):
    vv= gh.Vector2Pt(gh.EndPoints(v)[0], gh.EndPoints(v)[1], False)[0]
    V= gh.Amplitude(vv,ep)
    L= gh.Extrude(lames, V)
    F= gh.DeconstructBrep(L[0])[0][0]
    Pt= gh.SurfaceClosestPoint(gh.Area(F)[1], F)[1]
    Ev = gh.EvaluateSurface(F, Pt)
    L2 = gh.LineSDL(Ev[0], Ev[1], 0.05)
    V2= gh.Vector2Pt(gh.EndPoints(L2)[0], gh.EndPoints(L2)[1], False)[0]
    Lames = gh.SolidUnion(gh.Extrude(L, V2))
    return Lames

def create_lames2(lames, v, ep):
    vv= gh.Vector2Pt(gh.EndPoints(v)[0], gh.EndPoints(v)[1], False)[0]
    V= gh.Amplitude(vv,ep)
    S= gh.Extrude(lames, V)
    Pt= gh.SurfaceClosestPoint(gh.Area(S)[1], S)[1]
    Ev = gh.EvaluateSurface(S, Pt)
    L2 = gh.LineSDL(Ev[0], Ev[1], 0.05)
    V2= gh.Vector2Pt(gh.EndPoints(L2)[0], gh.EndPoints(L2)[1], False)[0]
    Lames = gh.SolidUnion(gh.Extrude(S, V2))
    return Lames

def eval(lames1, lames2, m, pts, vect, F, v1, v2, ep1, ep2):
    Lames1=create_lames2(lames1, v1, ep1)
```

```
    Lames2=create_lames2(lames2, v2, ep2)
    #S=gh.SettingsCustom(True, False, True, 16, 400, 20, 0, 20, 0, 0)
    M = gh.MeshJoin(flatten_list([m, gh.MeshBrep(Lames1), gh.MeshBrep(Lames2)]))
    PTS = gh.CleanTree(True, True, False, gh.MeshXRay(M, pts, vect)[0])
    pts= gh.CullPattern(PTS, gh.PointInBrep(F, PTS, False))
    Nb = gh.ListLength(pts)
    return Nb


if Toggle is True:
    i=0
    timeout=time.time() +60*3
    while i<10:
        if time.time()>timeout: break
        X = move_lames(lames1, line1)
        x = eval(X, lames2, m, pts, vect, F, v1, v2, ep1, -ep2)
        lames1[:]=[]
        lames1 = flatten_list(X)
        c = (x*100)/n
        if c<0.5: break
        Y = move_lames(lames2,line2)
        y = eval(lames1, Y, m, pts, vect, F, v1, v2, ep1, -ep2)
        lames2[:]=[]
        lames2 = flatten_list(Y)
        c = (y*100)/n
        if c<0.5: break

        ep1+=0.03
        x = eval(lames1, lames2, m, pts, vect, F, v1, v2, ep1, -ep2)
        c = (y*100)/n
        if c<0.5: break
        ep2+=0.03
        x = eval(lames1, lames2, m, pts, vect, F, v1, v2, ep1, -ep2)
        c = (y*100)/n
        if c<0.5: break
        i+=1
    a = create_lames2(lames1,v1,ep1)
    b = create_lames2(lames2, v2, ep2)
```

Annexe 3 Code du solveur d'automate cellulaire augmenté

```
"""This component allows to launch the cellular automaton.
    Inputs:
        Run: "True" to start calculation.
        sol: A tree with attributes of an initial solution.
        att: Tree containing all the information about the attributes.
            (use the "Cellular_Automata_Attributes_Definition" component).
        fix_att: List of attribut name to fix the quantity of those attributes
        fix_sta:List of "attribut name; special state" to fix the quantity
        of those specific state of an attribut.(For example, for an urban
        morphology attribute, fix the quantity of towers).
        fix_cell: List of rules to fix the attributes of a specific cell.
        Use the "Cellular_Automata_Rule_Fix_Cell" component.
        rules: List of list that will allow
            the solver to define a rule. (Differents types of rules
            can be create with differents components:
            "Cellular_Automata_self_rule", "Cellular_Automata_Rule_Neighborhood",
            "Cellular_Automata_Rule_Cell_repulsion", "Cellular_Automata_Rule_Attraction")
        type:Selection method of the attributes allowed in case
            of modification of the cell. 0 for random, 1 for in order.
        nr: radius of the neighborhood taken into account
        for interchanging attributes
        iter: Number of iteration.
    Output:
        result: New tree with new attributes of a new solution.
        i: Number of iteration performed.
        n: Number of remaining cells not respecting the rules.
"""

ghenv.Component.Name = "Ur_Augmented_Cellular_Automata_Solver"
ghenv.Component.NickName = 'ACA_Solver'
ghenv.Component.Message = 'Repair'
ghenv.Component.Category = 'Urchin'
ghenv.Component.SubCategory = 'Repair'
#ghenv.Component.AdditionalHelpFromDocStrings = '1'


import rhinoscriptsyntax as rs
import ghpythonlib.components as gh
import copy
import random
import Grasshopper

# fonctions de gestion des arbres
def tree_to_list(input, retrieve_base = lambda x: x[0]):

def flatten_list(_2d_list):

def list_to_tree(input, none_and_holes=True, source=[0]):


# fonctions utilitaires
def concatenate(R):
    result=[]
    i=0
    while i<len(R):
        X= list(R[i:i+5])
        result.append(X)
        i+=5
    return result

def concatenate2(fix_cell):
    result=[]
    i=0
    while i<len(fix_cell):
```

```
        X= list(fix_cell[i:i+2])
        result.append(X)
        i+=2
    return result

def list_ori(Att,Sol):
    result=[]
    for ele in Sol:
        lignes=[]
        for k in ele:
            lignes.append(Att)
        result.append(lignes)
    return result

def sort_rules(rules, F):

    # revoir ordre en fonction de la hiérarchie des attributs
    i=3
    R=[]
    while i>-1:
        for ele in rules:
            if ele[0][0][0]==i:
                R.append(ele)
        i-=1
    if len(F)>0:
        if len(R)>1:
            result=[]
            rest=[]
            for r in R:
                for f in F:
                    if r[4][f]:
                        result.append(r)
                        break
                    else: rest.append(r)
            for ele in rest: result.append(ele)
            return result
        else: return rules
    else: return R

def base(nr):
    i=1
    x=[]
    y=[]
    while i<nr+1:
        j=0
        k=0
        while k<2*i+1:
            x.append(-i+k)
            y.append(i)
            k+=1
        k=1
        while k<2*i+1:
            x.append(i)
            y.append(i-k)
            k+=1
        k=1
        while k<2*i+1:
            x.append(i-k)
            y.append(-i)
            k+=1
        k=1
        while k<2*i:
            x.append(-i)
            y.append(-i+k)
            k+=1
        i+=1
```

```
    return [x,y]

def nb_v(nr):
    result=[0]
    i=1
    while i<nr:
        if result:
            result.append(result[-1]+2*i*4)
        else: result.append(2*i*4)
        i+=1
    return result

def index_voisins(x,y,Sol,P,nr):
    Base= base(nr)
    result=[]
    lx=len(Sol)-1
    ly=len(Sol[x])-1
    for p in P:
        if x + Base[0][p]>=0 and x + Base[0][p]<=lx:
            if y+ Base[1][p]>=0 and y + Base[1][p]<=ly:
                result.append([p,x + Base[0][p], y+ Base[1][p]])
    return result

def voisins(x,y,Sol,P,nr):
    Base= base(nr)
    result=[]
    lx=len(Sol)-1
    ly=len(Sol[x])-1
    for p in P:
        if x + Base[0][p]>=0:
            if  x + Base[0][p]<=lx:
                if y+ Base[1][p]>=0 and y + Base[1][p]<=ly:
                    result.append(Sol[x + Base[0][p]][y+ Base[1][p]])
    return result

def verif_input():

    e = Grasshopper.Kernel.GH_RuntimeMessageLevel.Error
    w = Grasshopper.Kernel.GH_RuntimeMessageLevel.Warning
    r = Grasshopper.Kernel.GH_RuntimeMessageLevel.Remark

    start= True
    if not sol or sol is None:
        ghenv.Component.AddRuntimeMessage(e, 'Input "sol" is missing.')
        start=False
    if not att or att is None:
        ghenv.Component.AddRuntimeMessage(e, 'Input "att" is missing.')
        start=False

    if not rules:
        ghenv.Component.AddRuntimeMessage(e, 'Input "rules" is missing.')
        start=False
    else:
        for ele in rules:
            if ele is None:
                ghenv.Component.AddRuntimeMessage(e, 'Input "rules" is missing.')
                start = False

    if iter==0:
        ghenv.Component.AddRuntimeMessage(r, 'Input "iter" is equal to 0. \n No rules are
applied to the initial solution')
        start=False

    return start

def verif_input2():
```

```
    e = Grasshopper.Kernel.GH_RuntimeMessageLevel.Error
    w = Grasshopper.Kernel.GH_RuntimeMessageLevel.Warning
    r = Grasshopper.Kernel.GH_RuntimeMessageLevel.Remark

    Att = tree_to_list(att)

    nom_att=[]
    for ele in Att:
        nom_att.append(ele[0])
        ele.remove(ele[0])

    start= True
    for n in fix_att:
        if n not in nom_att:
            ghenv.Component.AddRuntimeMessage(e, 'A fixed attribut (fix_att) is not present
in the attributs informations (att).')
            start =False
            break

    for ele in fix_sta:
        list=[]
        a = ele.split(';')
        if a[0] not in str(nom_att):
            ghenv.Component.AddRuntimeMessage(e, 'An attribut of a fixed state (fix_sta) is
not present in the attributs informations (att).')
            start =False
            break
        elif a[1] not in str(Att[nom_att.index(a[0])]):
            ghenv.Component.AddRuntimeMessage(e, 'A  fixed state (fix_sta) is not present
in the attributs informations (att).')
            start =False
            break

    return start

def is_fix(x,y,N,nom_att, fix_cell):
    fix = False

    if len(fix_cell)>0:
        i=0
        while i<len(fix_cell):
            if fix_cell[i][0]==len(nom_att):
                j=0
                while j<len(fix_cell[i][1]):
                    if int(x)==int(fix_cell[i][1][j][0]):
                        if int(y)==int(fix_cell[i][1][j][1]):
                            fix =True
                    j+=1
            i+=1


    if fix is False:
        I=0
        while I<len(fix_cell):
            if int(fix_cell[I][0])==int(N):
                j=0
                while j<len(fix_cell[I][1]):
                    if int(x)==int(fix_cell[I][1][j][0]):
                        if int(y)==int(fix_cell[I][1][j][1]): fix = True
                    j+=1
            I+=1
    return fix

# fonctions d'applications des règles
```

```
def change_att(Sol, i, R, Att, x, y, nom_att):

    e = Grasshopper.Kernel.GH_RuntimeMessageLevel.Error
    w = Grasshopper.Kernel.GH_RuntimeMessageLevel.Warning
    r = Grasshopper.Kernel.GH_RuntimeMessageLevel.Remark

    new_att= copy.deepcopy(Att[i])
    for ele in R:
        if ele in new_att: new_att.remove(ele)
    if len(new_att)==0:
        ghenv.Component.AddRuntimeMessage(w, 'There are contradictory rules, the problem is
overconstrained on cell '+str(x)+';'+str(y)+' with attribut '+str(nom_att[i])+'.')
    elif type == 0:
        Sol[x][y][i]=random.choice(new_att)

    elif type==1:
        Sol[x][y][i]= new_att[0]

    return Sol

def change_att2(Sol, i, R, Att, x, y, nom_att, Rules, nr, K, fix_cell):

    e = Grasshopper.Kernel.GH_RuntimeMessageLevel.Error
    w = Grasshopper.Kernel.GH_RuntimeMessageLevel.Warning
    r = Grasshopper.Kernel.GH_RuntimeMessageLevel.Remark

    new_att= copy.deepcopy(Att[i])
    for ele in R:
        if ele in new_att: new_att.remove(ele)


    if len(new_att)==0:
        ghenv.Component.AddRuntimeMessage(w, 'There are contradictory rules, the problem is
overconstrained on cell '+str(x)+';'+str(y)+' with attribut '+str(nom_att[i])+'.')

    elif type == 0:

        nv=0
        h=0
        possible = False
        while nv<3:
            if h>5000: break
            lim= nb_v(nr)
            nt1=Sol[x][y][i]
            v = [range(lim[k],lim[k+1]) for k in range(0,len(lim)-1)]
            j=0
            while j<lim[-1]:
                for k in range(0,len(lim)-1):
                    if j>=lim[k] and j<lim[k+1]:
                        t = random.choice(v[k])
                        v[k].remove(t)

                V = voisins(x,y,Sol,[t],nr)
                I= index_voisins(x,y,Sol,[t],nr)
                h+=1
                if V:
                    fix = is_fix(I[0][1],I[0][2],i,nom_att, fix_cell)
                    if fix is False:
                        if V[0][i] in new_att:
                            a = modif_voisins(I, nt1, Rules, i, Sol, x, y, V[0][i], nr, nv)
                            if a[2]==1:h=10000
                            possible =a[0]
                            if possible is True:
                                Sol[x][y][i]=V[0][i]
                                Sol[I[0][1]][I[0][2]][i]=nt1
                                break
```

```
                if possible is True: break
                j+=1
            if possible is True: break
            nv+=1



    elif type==1:
        nv=0
        h=0
        possible = False
        while nv<3:
            if h>5000:break
            lim= nb_v(nr)
            nt1=Sol[x][y][i]
            j=0
            while j<lim[-1]:
                V = voisins(x,y,Sol,[j],nr)
                I= index_voisins(x,y,Sol,[j],nr)
                h+=1
                if V:
                    fix = is_fix(I[0][1],I[0][2],i,nom_att, fix_cell)
                    if fix is False:
                        if V[0][i] in new_att:
                            a = modif_voisins(I, nt1, Rules, i, Sol, x, y, V[0][i], nr, nv)
                            possible =a[0]
                            if a[2]==1:h=10000
                            if possible is True:
                                Sol[x][y][i]=V[0][i]
                                Sol[I[0][1]][I[0][2]][i]=nt1
                                break
                if possible is True: break
                j+=1
            if possible is True: break
            nv=+1

    if possible is False: ghenv.Component.AddRuntimeMessage(r, 'During iteration '+str(K) +
' the cell '+str(x)+';'+str(y)+' didn\'t find any neighbours to exchange his
'+str(nom_att[i])+' attribut with.')
    return Sol

def modif_voisins(I, nt, Rules, i, Sol, x, y, ntx, nr, nv):
    nv1=0
    SOL = copy.deepcopy(Sol)
    SOL[x][y][i]=ntx
    SOL[I[0][1]][I[0][2]][i]=nt
    test=[]
    possible = False
    R=[]
    v= SOL[I[0][1]][I[0][2]]
    for r in Rules: # lister les règles qui impactent l'attribut en question
        if r[4][i]: R.append(r)
    if len(R)==0: # si le voisin n'a pas de contraintes
            possible =True

    if len(R)==1:

        r=R[0]
        if nt not in r[4][i]: possible =True # vérifier si le nouvel attribut est conserné
par la règle
        else:
            if r[0][0][0]==0: # pour une règle de 0...
                k=0
                while k<len(r[1]):# ...vérifier que la cellule est consernée par la règle
                    if r[1][k] and v[k]!=r[1][k][0]:
                        possible = True
```

```
                    break
                k+=1


        elif r[0][0][0]==1: # pour une règle de 1...
            k=0
            while k<len(r[1]):# ...vérifier que la cellule est consernée par la règle
                if r[1][k] and v[k] not in r[1][k]:
                    possible = True
                    break
                k+=1
            if possible is False: # regarder si la règle n'est pas déjà respectée en
regardant les voisins
                V = voisins(I[0][1],I[0][2],SOL,r[2][0],r[0][0][1])
                if len(V)>0:
                    Na=0
                    k=0
                    while k<len(r[3]):
                        c=0
                        for v2 in V:# vérifier si l'ensemble des voisins ne respectent
pas les règles
                            if r[3][k]:
                                if v2[k] in r[3][k]:
                                    c+=1
                            else: c+=1
                        if c==len(V):Na+=1
                        k+=1
                    if Na!=len(r[3]): possible = True

        elif r[0][0][0]==2: # pour une règle de 2...
            V = voisins(I[0][1],I[0][2],SOL,r[2][0],r[0][0][1])
            next= False
            Nv=0
            for v in V:
                c=0
                k=0
                while k<len(r[3]):
                    if r[3][k]:
                        if v[k] in r[3][k]:c+=1
                    else: c+=1
                    k+=1
                if c==len(r[3]):Nv+=1

            if Nv==len(V):next = True # vérifier si l'ensemble des voisins est ne
respectent pas les règles
            if next is False: possible = True

        elif r[0][0][0]==3:
            V1 = voisins(x,y,Sol,r[2][0],r[0][0][1])
            N=0
            if len(V1)>0:
                for v1 in V1:
                    c=0
                    k=0
                    while k<len(r[3]):
                        if r[3][k]:
                            if v1[k] in r[3][k]:
                                c+=1
                        else: c+=1
                        k+=1
                    if c==len(r[3]):N+=1

            nv1= r[1][0][0]-nv
            if nv1>1:
                V = voisins(I[0][1],I[0][2],SOL,r[2][0], r[0][0][1])
                respect=False
```

```
                    if len(V)>0:
                        n=0
                        for v in V:
                            c=0
                            k=0
                            while k<len(r[3]):
                                if r[3][k]:
                                    if v[k] in r[3][k]:
                                        c+=1
                                else: c+=1
                                k+=1
                            if c==len(r[3]):n+=1
                            if n>=nv1: # and n>N:
                                respect =True
                                break
                        if respect is True:
                            possible = True


    if len(R)>1:
        R2=[]
        for r in R:
            if r[0][0][0]==0:
                c=True
                k=0
                while k<len(r[1]):# ...vérifier que la cellule est consernée par la règle
                    if r[1][k] and v[k]!=r[1][k][0]:
                        c= False
                        break
                    k+=1
                if c is True:
                    R2.append(r)



            elif r[0][0][0]==1: # pour une règle de 1...
                conserne=True
                k=0
                while k<len(r[1]):# ...vérifier que la cellule est consernée par la règle
                    if r[1][k] and v[k] not in r[1][k]:
                        conserne = False
                        break
                    k+=1
                if conserne is True:
                    V = voisins(I[0][1],I[0][2],SOL,r[2][0], r[0][0][1])
                    if len(V)>0:
                        Na=0
                        k=0
                        while k<len(r[3]):
                            c=0
                            for v1 in V:# vérifier si l'ensemble des voisins est ne
respectent pas les règles
                                if r[3][k]:
                                    if v1[k] in r[3][k]:
                                        c+=1
                                else: c+=1
                            if c==len(V):Na+=1
                            k+=1
                        if Na==len(r[3]): R2.append(r)

            elif r[0][0][0]==2: # pour une règle de 2...
                V = voisins(I[0][1],I[0][2],SOL,r[2][0], r[0][0][1])
                if len(V)>0:
                    next= False
                    Nv=0
                    for v in V:
```

```
                        c=0
                        k=0
                        while k<len(r[3]):
                            if r[3][k]:
                                if v[k] in r[3][k]:c+=1
                            else: c+=1
                            k+=1
                        if c==len(r[3]):Nv+=1
                    if Nv==len(V):next = True # vérifier si l'ensemble des voisins est ne
respectent pas les règles
                    if next is True:  R2.append(r)

            elif r[0][0][0]==3:
                V = voisins(x,y,Sol,r[2][0], r[0][0][1])
                if len(V)>0:
                    N=0
                    for v1 in V:
                        c=0
                        k=0
                        while k<len(r[3]):
                            if r[3][k]:
                                if v1[k] in r[3][k]:
                                    c+=1
                            else: c+=1
                            k+=1
                        if r[1][0][0]==1 and c==len(V):Na+=1
                        if r[1][0][0]==0 and c>0:Na+=1

                    nv1= r[1][0][0]-nv
                    respect=False
                    if nv1>0:
                        V = voisins(I[0][1],I[0][2],SOL,r[2][0], r[0][0][1])
                        respect=False
                        if len(V)>0:
                            n=0
                            for v in V:
                                c=0
                                k=0
                                while k<len(r[3]):
                                    if r[3][k]:
                                        if v[k] in r[3][k]:
                                            c+=1
                                    else: c+=1
                                    k+=1
                                if c==len(r[3]):n+=1
                                if n>=nv1: # and n>N:
                                    respect =True
                                    break
                            if respect is False: R2.append(r)


        if len(R2)==0:
            possible =True

        elif len(R2)==1:
            r=R2[0]
            if nt not in r[4][i]: possible =True

        elif len(R2)>1:# si plusieurs règles doivent d'être appliquées à la même cellule
sur le même attribut
            values=[]
            for r in R2:
                for ele in r[4][i]:values.append(ele)

            if nt not in values: possible =True
```

```
    return [possible, test,nv1]

def One_rule(Sol,x,y,r,i, Att, nom_att, F, Rules, nr, K, Fs, fix_cell):

    valid=True
    test =False
    if r[0][0][0]==0: # pour une règle de 0...
        k=0
        conserne=True
        while k<len(r[1]):# ...vérifier que la cellule est consernée par la règle
            if r[1][k] and Sol[x][y][k]!=r[1][k][0]:
                conserne = False
                break
            k+=1
        if conserne is True:
            if Sol[x][y][i] in r[4][i]:
                fixe = False
                if i in F: fixe=True
                for ele in Fs:
                    if i==ele[0] and Sol[x][y][i]==ele[1]:
                        fixe = True
                        test = "fix by sta"
                if fixe is True:
                    a =change_att2(Sol, i, r[4][i], Att, x, y, nom_att, Rules, nr, K,
fix_cell) # si la cellule ne respect pas cette règle, changer l'attribut
                    Sol =a
                    valid=False

                else:
                    Sol =change_att(Sol, i, r[4][i], Att, x, y, nom_att)
                    valid=False


    elif r[0][0][0]==1: # pour une règle de 1...
        conserne=True
        k=0
        while k<len(r[1]):# ...vérifier que la cellule est consernée par la règle
            if r[1][k] and Sol[x][y][k] not in r[1][k]:
                conserne = False
                break
            k+=1
        if conserne is True: # regarder si la règle n'est pas déjà respectée en regardant
les voisins
            V = voisins(x,y,Sol,r[2][0], r[0][0][1])
            if len(V)>0:
                Na=0
                k=0
                while k<len(r[3]):
                    c=0
                    for v in V:# vérifier si l'ensemble des voisins ne respectent pas les
règles
                        if r[3][k]:
                            if v[k] in r[3][k]:
                                c+=1
                        else: c+=1
                    if c==len(V):Na+=1
                    k+=1
                if Na==len(r[3]):
                    if Sol[x][y][i] in r[4][i]:
                        fixe = False
                        if i in F: fixe=True
                        for ele in Fs:
                            if i==ele[0] and Sol[x][y][i]==ele[1]:
                                fixe = True
                                test = "fix by sta"
                        if fixe is True:
```

```
                            a =change_att2(Sol,i, r[4][i], Att,  x, y, nom_att, Rules, nr,
K, fix_cell) # sinon vérifier si la cellule ne respecte pas déjà la règle
                            Sol =a
                            valid=False
                        else:
                            Sol =change_att(Sol,i, r[4][i], Att,  x, y, nom_att )
                            valid=False


    elif r[0][0][0]==2: # pour une règle de 2...
        V = voisins(x,y,Sol,r[2][0], r[0][0][1])
        if len(V)>0:
            next= False
            Nv=0
            for v in V:
                c=0
                k=0
                while k<len(r[3]):
                    if r[3][k]:
                        if v[k] in r[3][k]:c+=1
                    else: c+=1
                    k+=1
                if c==len(r[3]): Nv+=1
            if Nv==len(V):next = True # vérifier si l'ensemble des voisins est ne
respectent pas les règles

            if next is True:
                if Sol[x][y][i] in r[4][i]:
                    fixe = False
                    if i in F: fixe=True
                    for ele in Fs:
                        if i==ele[0] and Sol[x][y][i]==ele[1]:
                            fixe = True
                            test ="fix by sta"
                    if fixe is True:
                        a = change_att2(Sol,i, r[4][i], Att,  x, y, nom_att, Rules, nr, K,
fix_cell) # sinon vérifier si la cellule ne respecte pas déjà la règle
                        Sol = a
                        valid=False
                    else:
                        Sol = change_att(Sol,i, r[4][i], Att,  x, y, nom_att)
                        valid=False

    elif r[0][0][0]==3: # pour une règle de 2...
        if Sol[x][y][i] in r[4][i]:# vérifier si la cellule est consernée par la règles
            V = voisins(x,y,Sol,r[2][0], r[0][0][1])
            respect = False
            if len(V)>0:
                n=0
                for v in V:
                    c=0
                    k=0
                    while k<len(r[3]):
                        if r[3][k]:
                            if v[k] in r[3][k]:
                                c+=1
                        else: c+=1
                        k+=1
                    if c==len(r[3]):n+=1
                    if n>=r[1][0][0]:
                        respect =True
                        break
                if respect is False:
                    fixe = False
                    if i in F: fixe = True # si l'attribut est fix (en quantité)
                    for ele in Fs: # si l'état est fix (en quantité)
```

```
                        if i==ele[0] and Sol[x][y][i]==ele[1]:
                            fixe = True
                            test= "fix by sta"
                    if fixe is True:
                        a = change_att2(Sol,i, r[4][i], Att,  x, y, nom_att, Rules, nr, K,
fix_cell)
                        Sol = a
                        valid=False
                        test = "banane"
                    else:
                        Sol = change_att(Sol,i, r[4][i], Att,  x, y, nom_att)
                        valid=False

    return [Sol,valid, test]

def Multi_rule(Sol,x,y,R,i, Att, nom_att, F, Rules, nr, K, Fs, fix_cell):
    valid= True
    test=[]
    R2=[]
    for r in R:
        if r[0][0][0]==0:
            c=True
            k=0
            while k<len(r[1]):# ...vérifier que la cellule est consernée par la règle
                if r[1][k] and Sol[x][y][k]!=r[1][k][0]:
                    c= False
                    break
                k+=1
            if c is True:
                R2.append(r)


        elif r[0][0][0]==1: # pour une règle de 1...
            conserne=True
            k=0
            while k<len(r[1]):# ...vérifier que la cellule est consernée par la règle
                if r[1][k] and Sol[x][y][k] not in r[1][k]:
                    conserne = False
                    break
                k+=1
            if conserne is True:
                V = voisins(x,y,Sol,r[2][0], r[0][0][1])
                if len(V)>0:
                    Na=0
                    k=0
                    while k<len(r[3]):
                        c=0
                        for v in V:# vérifier si l'ensemble des voisins est ne respectent
pas les règles
                            if r[3][k]:
                                if v[k] in r[3][k]:
                                    c+=1
                            else: c+=1
                        if c==len(V):Na+=1
                        k+=1
                    if Na==len(r[3]): R2.append(r)

        elif r[0][0][0]==2: # pour une règle de 2...
            V = voisins(x,y,Sol,r[2][0], r[0][0][1])
            if len(V)>0:
                next= False
                Nv=0
                for v in V:
                    c=0
                    k=0
                    while k<len(r[3]):
```

```
                        if r[3][k]:
                            if v[k] in r[3][k]:c+=1
                        else: c+=1
                        k+=1
                    if c==len(r[3]):Nv+=1
                if Nv==len(V):next = True # vérifier si l'ensemble des voisins est ne
respectent pas les règles

                if next is True:  R2.append(r)

        elif r[0][0][0]==3: # pour une règle de 2...
            V = voisins(x,y,Sol,r[2][0], r[0][0][1])
            respect=False
            if len(V)>0:
                n=0
                for v in V:
                    c=0
                    k=0
                    while k<len(r[3]):
                        if r[3][k]:
                            if v[k] in r[3][k]:
                                c+=1
                        else: c+=1
                        k+=1
                    if c==len(r[3]):n+=1
                    if n>=r[1][0][0]:
                        respect =True
                        break
                if respect is False: R2.append(r)


    if len(R2)==0:
        test.append("0")

    elif len(R2)==1:

        r=R2[0]
        if Sol[x][y][i] in r[4][i]:
            fixe = False
            if i in F: fixe=True
            for ele in Fs:
                if i==ele[0] and Sol[x][y][i]==ele[1]: fixe = True
            if fixe is True:
                a = change_att2(Sol, i, r[4][i], Att, x, y, nom_att, Rules, nr, K,
fix_cell) # si la cellule ne respecte pas cette règle, changer l'attribut
                Sol = a
                test.append("swith 1")
                valid=False
            else:
                Sol = change_att(Sol, i, r[4][i], Att, x, y, nom_att)
                test.append("normal 1")
                valid=False

    elif len(R2)>1:# si plusieurs règles doivent d'être appliquées à la même cellule sur le
même attribut
        test.append("plusieurs regles")
        values=[]

        for r in R2:
            for ele in r[4][i]:values.append(ele)

        if Sol[x][y][i] in values:
            fixe = False
            if i in F: fixe=True
            for ele in Fs:
                if i==ele[0] and Sol[x][y][i]==ele[1]: fixe = True
```

```
            if fixe is True:
                a = change_att2(Sol, i, values, Att, x, y, nom_att, Rules, nr, K, fix_cell)
                Sol=a
                test.append("swith 2")
                valid=False
            else:
                Sol = change_att(Sol, i, values, Att, x, y, nom_att)
                valid =False
                test.append("normal 2")

    return [Sol,valid,test]


# main functions

def run_cell(x,y, Sol, n, Rules, Att, nom_att, F, nr, K, fix_cell, Fs):

    valid= True
    test = "girafe"

    start = True

    if len(fix_cell)>0:
        i=0
        while i<len(fix_cell):
            if fix_cell[i][0]==len(nom_att):
                j=0
                while j<len(fix_cell[i][1]):
                    if int(x)==int(fix_cell[i][1][j][0]):
                        if int(y)==int(fix_cell[i][1][j][1]):
                            start =False
                    j+=1
            i+=1


    if start is True:
        i=0
        while i<n: # pour chaque attribut...
            next = True #...vérifier si l'attribut de la cellule n'est pas fixe,...
            I=0
            while I<len(fix_cell):
                if int(fix_cell[I][0])==int(i):
                    j=0
                    while j<len(fix_cell[I][1]):
                        if int(x)==int(fix_cell[I][1][j][0]):
                            if int(y)==int(fix_cell[I][1][j][1]): next =False
                        j+=1
                I+=1

            if next is False: test ="fix by cell"
            if next is True:
                R=[]
                for r in Rules: # lister les règles qui impactent l'attribut en question
                    if r[4][i]: R.append(r)

                if len(R)==0: # si aucune règle impacte l'attribut en question
                    pass

                if len(R)==1:
                    a = One_rule(Sol,x,y,R[0],i, Att, nom_att, F, Rules, nr, K, Fs,
fix_cell) #si une seule règles impactent l'attribut en question
                    Sol =a[0]
                    valid = a[1]
                    test = a[2]
```

```
                if len(R)>1: #regarder si l'ensemble des règles impactent l'attribut en
question
                    a =Multi_rule(Sol,x,y,R,i, Att, nom_att, F, Rules, nr, K, Fs, fix_cell)
                    Sol=a[0]
                    valid = a[1]


            i+=1
    return [Sol, valid, test]

def run_grid(Sol,n, Rules, Att, nom_att, F, nr, K, fix_cell, Fs):
    valid=0
    x=0
    test=[]
    while x<len(Sol):
        lignes=[]
        y=0
        while y<len(Sol[x]):
            a = run_cell(x,y, Sol, n, Rules, Att, nom_att, F, nr, K, fix_cell, Fs)
            Sol=a[0]
            test.append(a[2])
            if a[1] is False: valid+=1
            y+=1
        x+=1
    return [Sol, valid, test]

def main():
    Sol = tree_to_list(sol)
    Att = tree_to_list(att)

    nom_att=[]
    for ele in Att:
        nom_att.append(ele[0])
        ele.remove(ele[0])
    n =len(nom_att)
    #R=concatenate(rules)
    R =rules

    F= []
    for ele in fix_att:
        F.append(nom_att.index(ele))



    Rules = sort_rules(R,F)
    Fix_cell= concatenate2(fix_cell)
    Fs=[]
    for ele in fix_sta:
        list1=[]
        a = ele.split(';')
        i = nom_att.index(a[0])
        list1.append(i)
        list1.append(a[1])
        Fs.append(list1)
    X=run_grid(Sol,n, Rules, Att, nom_att, F, nr, 0, Fix_cell, Fs)
    Sol=X[0]
    test=X[2]

    valid=X[1]

    K=1
    while K<iter and valid>0:
        X=run_grid(Sol, n, Rules, Att, nom_att, F, nr, K, Fix_cell, Fs)
        Sol=X[0]
        valid = X[1]
        test=X[2]
```

```
        K+=1

    return [Sol,K,valid,test]


if Run is True:
    start = verif_input()
    if not type:type==0
    if not fix_cell: fix_cell=[]
    if not fix_sta: fix_sta = []
    if start is True:
        start = verif_input2()
        if start is True:

            a = main()
            result = list_to_tree(a[0])
            i = a[1]
            n = a[2]
            test=a[3]


        else:
            result= sol
    else:
        result= sol
else:
    result= sol
```

Annexe 4 Code de l'algorithme d'attraction

```
"""This component allows you to define a rule allowing an attribute of a cell
to impact another attribute of the same cell.
    Inputs:
        run: Set True to start simulation.
        rec: List of coplanar rectangles.
    Output:
        rec: List of rectangles resulting from simulation.
        i: number of iteration
"""

ghenv.Component.Name = "Ur_Attraction_Rectangles"
ghenv.Component.NickName = 'Att_Rec'
ghenv.Component.Message = 'Reparation'
ghenv.Component.Category = 'AS_Generative'
ghenv.Component.SubCategory = 'Reparation'
#ghenv.Component.AdditionalHelpFromDocStrings = '1'

import rhinoscriptsyntax as rs
import ghpythonlib.components as gh
import Rhino

# Fonctions pour passer des arborescences GH aux listes Python

def list_to_tree(input, none_and_holes=True, source=[0]):

def flatten_list(_2d_list):

def tree_to_list(input, retrieve_base = lambda x: x[0]):


################################################################################################
# FONCTIONS UTILITAIRES

def first_rec(rec):
    # Fonctions pour trouver le rectangle le plus au centre parmi l'ensemble des rectangles
    donnés en entrée
    index= range(0, len(rec))

    pt=[]
    for ele in rec:
        pt.append(gh.Area(ele)[1])
    plan= gh.XYPlane(pt[0])
    T = gh.FaceBoundaries(gh.DelaunayMesh(pt, plan ))

    PT= gh.Area(gh.RegionUnion(T, plan))[1]
    L=[]
    for p in pt:
        L.append(gh.Length(gh.Line(PT, p)))
    l = sorted(L)[0]
    I = L.index(l)

    Rec= [rec[I], I]
    index.remove(I)

    rec2=[]
    for i in index:
        rec2.append([rec[i], i])

    return [Rec, rec2]

def rect_inclu(crv):
    # fonction pour déterminer le rectangle dans lequel sont inclus tous les rectangles
donnés en entrée
```

```
    v = gh.Explode(crv, True)[1]
    x=[]
    y=[]
    for ele in v:
        x.append(gh.Deconstruct(ele)[0])
        y.append(gh.Deconstruct(ele)[1])
    x = sorted(flatten_list(x))
    y = sorted(flatten_list(y))
    pt1= gh.ConstructPoint(x[0], y[0], 0)
    pt2= gh.ConstructPoint(x[0], y[-1], 0)
    pt3= gh.ConstructPoint(x[-1], y[-1], 0)
    pt4= gh.ConstructPoint(x[-1], y[0], 0)

    return gh.PolyLine([pt1, pt2, pt3, pt4], True)

def col(crv1, crv2, int, j):
    # fonction pour supprimer les collisions entre les rectangles donnés en entrée
    if type(int) is list and j>2:
        o=gh.Area(crv1)[1]
        V=gh.VectorXYZ(0,0,0)[0]
        for ele in int:
            pt = gh.Area(ele)[1]
            v=gh.Vector2Pt(pt, o, False)[0]
            V+=v

    else:
        if type(int) is list:
            aire=gh.Area(int)[0]
            I = aire.index(sorted(aire)[-1])
            int= int[I]

        pt= gh.Explode(int, True)[1]
        del pt[-1]
        PT=[]
        i=0
        while i<len(pt):
            if gh.PointInCurve(pt[i],crv2)[0]==2:
                PT.append(pt[i])
                del pt[i]
            i+=1
        if len(PT)==2:
            if j<2 or j%2==0:
                del PT[0]
                Long=[]
                for ele in pt:
                    Long.append(gh.Length(gh.Line(PT, ele)))
                PT2= pt[Long.index(sorted(Long)[0])]
                V = gh.Vector2Pt(PT, PT2, False)[0]
            else:
                c = gh.Area(crv2)[1]
                long=[]
                for ele in PT:
                    long.append(gh.Length(gh.Line(ele, c)))
                I1= long.index(sorted(long)[0])
                I2= long.index(sorted(long)[1])
                V = gh.Vector2Pt(PT[I1], PT[I2], False)[0]

        elif len(PT)==1:
            test = True
            Long=[]
            for ele in pt:
                Long.append(gh.Length(gh.Line(PT, ele)))
            if j<2 or j%2==0:PT2= pt[Long.index(sorted(Long)[0])]
            else: PT2= pt[Long.index(sorted(Long)[1])]
            V = gh.Vector2Pt(PT, PT2, False)[0]
```

```
        elif len(PT)==0:

            s = gh.Explode(int, True)[0]
            l= gh.Length(s)
            l2= sorted(l)[0]
            L= s[l.index(l2)]
            A = gh.EndPoints(L)
            a = gh.Area(crv1)[1]
            dist=[]
            for ele in A:
                dist.append(gh.Length(gh.Line(a, ele)))
            dist_sort= sorted(dist)
            I1 = dist.index(dist_sort[1])
            I2 = dist.index(dist_sort[0])
            V = gh.Vector2Pt(A[I1], A[I2], False)[0]

    rec= gh.Move(crv1, V)[0]
    return rec

def inclusion(Crv, Rec):
    #Fonction pour éviter qu'un rectangle soit inclus dans un autre rectangle
    pt=gh.Area(Rec)[1]
    pt2 = gh.CurveClosestPoint(pt, Crv)[0]
    pt3 = gh.CurveXCurve(Rec,gh.Line(pt,pt2))[0]
    V= gh.Amplitude(gh.Vector2Pt(pt,pt2, False)[0],
gh.Length(gh.Line(pt,pt2))+gh.Length(gh.Line(pt,pt3)))
    Rec2= gh.Move(Rec, V)[0]

    return Rec2

def closest_rec(Crv, Rec):

    ptC= gh.Area(Crv)[1]
    L=[]
    for ele in Rec:
        L.append([gh.Length(gh.Line(gh.Area(ele[0])[1], ptC)), ele[0], ele[1]])
    rec = sorted(L)
    for ele in rec:
        del ele[0]
    rec1= rec[0]
    del rec[0]
    return [rec1, rec]

def nb_cote(rec, Crv):
    s =gh.Explode(rec, True)[0]
    result=[]
    cote =[]
    all = True
    for ele in s:
        pts=gh.DivideCurve(ele, int(100*gh.Length(rec)), False)[0]
        del pts[0]
        del pts[-1]
        pat = gh.PointInCurve(pts, Crv)[0]
        if 1 in pat:
            result.append(1)
            cote.append(ele)
            if 0 in pat: all = False
        else:result.append(0)
    stop=False
    if result[-1]==1 and result[-1]==result[0]: stop =True
    else:
        i=0
        while i<len(result)-1:
            if i< len(result)-1:
                if result[i]==1 and result[i]==result[i+1]:
                    stop= True
```

```
                    break
            i+=1
    return [stop, all, cote]

def point_repere(k, pt, Rec2):
    V =[ gh.VectorXYZ(k, k, 0)[0], gh.VectorXYZ(-k, k, 0)[0], gh.VectorXYZ(k, -k, 0)[0],
gh.VectorXYZ(-k, -k, 0)[0]]
    pt1=[]
    for v in V:
        pt1.append(gh.Move(pt, v)[0])
    c = gh.CullPattern(pt1, gh.PointInCurve(pt1, Rec2)[0])
    return c


################################################################################
# DIFFERENTS ALGORITHMES D'ATTRACTION SELON LES SITUATIONS

def algo1(Rec, Crv):

    #Attraction de Rec vers Crv par le plus court chemin
    ptR = gh.Area(Rec)[1]
    ptC = gh.Area(Crv)[1]
    L = gh.Line(ptR, ptC)
    SR = gh.BoundarySurfaces(Rec)
    SC = gh.BoundarySurfaces(Crv)

    if gh.PointInCurve(ptC, Crv)[0]!=2:
        C =gh.Circle(gh.XYPlane(ptC), gh.Length(Crv)/20)
        C2 = gh.Project(C,SC, gh.UnitZ(1))
        ptC = gh.DivideCurve(C2, 5, False)[0][1]

    L = gh.Line(ptR, ptC)
    Lr = gh.Project(L,SR, gh.UnitZ(1))
    Lc = gh.Project(L,SC, gh.UnitZ(1))

    if type(Lc) is list:
        pt=[]
        for ele in Lc:
            o = gh.EndPoints(ele)
            pt.append(o[0])
            pt.append(o[1])
        x=[]
        y=[]

        for ele in pt:
            x.append(gh.Deconstruct(ele)[0])
            y.append(gh.Deconstruct(ele)[1])
        sorted(x)
        sorted(y)
        P=gh.PolyLine([gh.ConstructPoint(x[0], y[0], 0), gh.ConstructPoint(x[0], y[-1], 0),
gh.ConstructPoint(x[-1], y[-1], 0), gh.ConstructPoint(x[-1], y[0], 0)], True)
        pattern = gh.PointInCurve(pt, P)[0]
        pt2=[]
        i=0
        while i<len(pt):
            if pattern[i]==1: pt2.append(pt[i])
            i+=1
        Lc = gh.Line(pt2[0], pt2[1])


    A = gh.Length(L) -(gh.Length(Lr) + gh.Length(Lc))

    V= gh.Amplitude(gh.Vector2Pt( ptR, ptC , False)[0], A)
    Rec1= gh.Move(Rec, V)[0]
    return Rec1
```

```
def algo2(Crv, Rec, c):

    # Recherche d'un second côté adjacent
    m = gh.CurveMiddle(c)
    m2 = gh.Area(Rec)[1]
    v = gh.Vector2Pt(m,m2, False)[0]
    m3=gh.Move(m2, v)[0]
    pt = gh.DivideCurve(gh.Line(m,m3), 10, False)[0]
    del pt[0]
    del pt[-1]
    L=[]
    d = gh.Length(Rec)
    for p in pt:
        l= gh.Move(c, gh.Vector2Pt(m, p, False)[0])[0]
        L.append(gh.ExtendCurve(l, 0, d,d))

    pt_C=[]
    for l in L:
        PT= gh.CurveXCurve(Crv, l)[0]

        if not PT:  pt_C.append(False)
        elif str(type(PT)) == "<type 'Point3d'>": pt_C.append(PT)
        elif len(PT)>1:
            long=[]
            for ele in PT:
                long.append(gh.Length(gh.Line(gh.CurveMiddle(l), ele)))
            short= long.index(sorted(long)[0])
            pt_C.append(PT[short])

    pt_R=[]
    i=0
    while i< len(L):
        if pt_C[i] == False: pt_R.append(False)
        else:
            PT= gh.CurveXCurve(Rec, L[i])[0]
            long=[]
            for p in PT:
                long.append(gh.Length(gh.Line(p,pt_C[i])))
            pt_R.append(PT[long.index(sorted(long)[0])])
        i+=1

    L2=[]
    i=0
    while i<len(pt_C):
        if pt_C[i] == False: pass
        else:
            l=gh.Line(pt_R[i], pt_C[i])
            if gh.Length(l) > gh.Length(Rec)/10000:
                L2.append(l)
        i+=1

    if len(L2) == 0:
        rec = Rec
        next = True
    else:
        next = False
        if len(L2) >1:
            long_L2=gh.Length(L2)
            I = long_L2.index(sorted(long_L2)[0])
            L_final = L2[I]
        else:
            L_final = L2[0]
        A = gh.EndPoints(L_final)
        V = gh.Vector2Pt(A[0], A[1], False)[0]
        rec= gh.Move(Rec, V)[0]
```

```
    return [rec, next]

def algo3(Crv, Rec):
    # Si pas de second coté adjacent possible, aligner le rectangle au sommet de Crv le
plus proche
    pt = gh.Explode(Crv, True)[1]
    del pt[0]
    Pt=[]
    Ptb =[]
    for p in pt:
        if gh.PointInCurve(p, Rec)[0]==1: Pt.append(p)
        else:Ptb.append(p)

    ptr = gh.Explode(Rec, True)[1]
    del ptr[0]
    Ptr=[]
    Ptrb =[]
    for p in ptr:
        if gh.PointInCurve(p, Crv)[0]==1:
            Ptr.append(p)
        else:Ptrb.append(p)


    if len(Pt)==0 and len(Ptr)==0 : pass
    elif len(Ptr)>1 or len(Pt)>1: pass

    elif Ptr == Pt:
        l=[]
        for ele in Ptrb:
            l.append(gh.Length(gh.Line(ele, Ptr[0])))
        I = l.index(sorted(l)[0])
        V = gh.Vector2Pt(ptr[I], Ptr[0], False)[0]
        Rec = gh.Move(Rec, V)[0]


    else:
        plan = gh.XYPlane(Pt[0])
        d = gh.Length(Rec)*5
        d2 = gh.Length(Rec)/100
        L = gh.ExtendCurve(gh.Line(Pt[0], Ptr[0]), 0, d, d)
        C0 = gh.OffsetCurve(L, d2, plan, 1)
        ptC0 = gh.EndPoints(C0)
        C1 = gh.OffsetCurve(L, -d2, plan, 1)
        ptC1 = gh.EndPoints(C1)
        Pol = gh.PolyLine([ptC0[0], ptC0[1], ptC1[1], ptC1[0]], True)

        Pt_Pol=[]
        Pt_Pol.append(Pt[0])
        for p in Ptb:
            if gh.PointInCurve(p, Pol)[0]==2:
                Pt_Pol.append(gh.CurveClosestPoint(p, L)[0])
        Ptr_Pol=[]
        Ptr_Pol.append(Ptr[0])
        for p in Ptrb:
            if gh.PointInCurve(p, Pol)[0]==2: Ptr_Pol.append(p)

        x=[]
        y=[]
        for p in Pt_Pol:
            cc=gh.Deconstruct(p)
            x.append(p[0])
            y.append(p[1])

        X = gh.MassAddition(x)[0]/len(x)
        Y = gh.MassAddition(y)[0]/len(y)
        if abs(X-x[0])>abs(Y-y[0]): base =x
```

```
        else: base=y

        base_sort=sorted(base)
        PtA= Pt_Pol[base.index(base_sort[0])]
        PtB= Pt_Pol[base.index(base_sort[-1])]

        L2 = gh.Line(PtA, PtB)
        C0 = gh.OffsetCurve(L2, d2/2, plan, 1)
        ptC0 = gh.EndPoints(C0)
        C1 = gh.OffsetCurve(L2, -d2/2, plan, 1)
        ptC1 = gh.EndPoints(C1)
        Pol2 = gh.PolyLine([ptC0[0], ptC0[1], ptC1[1], ptC1[0]], True)

        x=[]
        y=[]
        for p in Ptr_Pol:
            cc=gh.Deconstruct(p)
            x.append(p[0])
            y.append(p[1])

        X = gh.MassAddition(x)[0]/len(x)
        Y = gh.MassAddition(y)[0]/len(y)
        if abs(X-x[0])>abs(Y-y[0]): base =x
        else: base=y

        base_sort=sorted(base)
        PtrA= Ptr_Pol[base.index(base_sort[0])]
        PtrB= Ptr_Pol[base.index(base_sort[-1])]
        A = gh.PointInCurve(PtrA, Pol2)[0]
        B = gh.PointInCurve(PtrB, Pol2)[0]

        if A==2 and B==2: pass
        elif A==2 or B==2:
            l1 = gh.Length(gh.Line(PtrA, PtA))
            l2 = gh.Length(gh.Line(PtrB, PtB))
            v1=gh.Vector2Pt(PtrA, PtA, False)[0]
            v2=gh.Vector2Pt(PtrB, PtB, False)[0]
            if l1<l2: Rec = gh.Move(Rec, v1)[0]
            else: Rec = gh.Move(Rec, v2)[0]


        else: pass
    return Rec

##############################################################################################
# FONCTION PRINCIPALE

def attraction(Crv, rec):

    Rec = rec[0]
    Rec1= algo1(Rec, Crv)
    plan = gh.XYPlane(gh.Area(Rec)[1])

    # vérifier si Rec2 n'est pas inclue dans Crv
    U = gh.Area(gh.RegionUnion(gh.RegionUnion(flatten_list([Rec1, Crv]), plan), plan))[0]
    if type(U) is float:
        if abs(U-gh.Area(Crv)[0])<0.0001:
                Rec1= inclusion(Crv, Rec1)

    # vérifier si Rec2 n'est pas en collision avec Crv
    int = gh.RegionIntersection(Crv, Rec1, plan)
    i=0
    if int:
        i=0
        recS= Rec1
        while int and i<10:
```

```
            Rec2 = col(recS, Crv, int, i)
            recS = Rec2
            int = gh.RegionIntersection(Crv, Rec2, plan)
            uni=gh.RegionUnion(gh.RegionUnion(flatten_list([Rec2, Crv]), plan), plan)
            if int:
                if abs(gh.Area(uni)[0]-gh.Area(Crv)[0])<0.0001:
                    Rec2= inclusion(Crv, Rec2)
                    int = gh.RegionIntersection(Crv, Rec2, plan)
            else:
                if type(uni) is list: Rec2= algo1(Rec2, Crv)
            i+=1
    else: Rec2= Rec1

    # regarder si Rec2 est adjacent à Crv sur seulement 1 coté
    stop = nb_cote(Rec2, Crv)

    if stop[0] == False:
        v = gh.Explode(Rec2, True)[1]
        del v[-1]
        pt = gh.CullPattern(v, gh.PointInCurve(v, Crv)[0])
        if pt:
            if len(pt)==2 and stop[1]== True:
                X = algo2(Crv, Rec2, stop[2][0])
                Rec3 =X[0]
            elif str(type(pt)) == "<type 'Point3d'>":
                X = algo2(Crv, Rec2, stop[2][0])
                Rec3 =X[0]
                if X[1]==True:
                    Rec3= algo3(Crv,Rec3)
            else: Rec3= Rec2
        else: Rec3= Rec2
    else: Rec3= Rec2

    # regarder à nouveau si Rec3 n'est pas en collision avec Crv
    int = gh.RegionIntersection(Crv, Rec3, plan)
    if int:
        i=0
        recS= Rec3
        while int and i<10:
            Rec4 = col(recS, Crv, int, i)
            recS = Rec4
            int = gh.RegionIntersection(Crv, Rec4, plan)
            if int:
                if abs(gh.Area(gh.RegionUnion(gh.RegionUnion(flatten_list([Rec4, Crv]),
plan), plan))[0]-gh.Area(Crv)[0])<0.0001:
                    Rec4= inclusion(Crv, Rec4)
                    int = gh.RegionIntersection(Crv, Rec4, plan)
            i+=1
    else: Rec4= Rec3

    Crv2= gh.RegionUnion(gh.RegionUnion(flatten_list([Crv, Rec4]), plan), plan)


    return [Crv2, [Rec4,rec[1]], Rec2]




##########################################################################################
# ALGORITHME

if run is True:

    result=[]
```

```
x = first_rec(rec)
result.append(x[0])
n =len(x[1])

reste = x[1]
Crv = x[0][0]

i=0
while i<n:
    y = closest_rec(Crv, reste)
    A = attraction(Crv, y[0])
    Crv = A[0]
    reste = y[1]
    result. append(A[1])
    i+=1

result.sort(key=lambda x:x[1])
rec =[]
for ele in result:
    a.append(ele[0])
```

Annexe 5 Code de l'algorithme de répulsion

```
"""This component allows you to define a rule allowing an attribute of a cell
to impact another attribute of the same cell.
    Inputs:
        run: Set True to start simulation.
        lim: Region in which agents must remain (optional closed curve)
        C: List of plane and closed coplanar curves.
    Output:
        C: List of curves resulting from simulation.
        i: Number of iteration.
"""

ghenv.Component.Name = "Ur_Repulsion2D"
ghenv.Component.NickName = 'Repulsion'
ghenv.Component.Message = 'Reparation'
ghenv.Component.Category = 'AS_Generative'
ghenv.Component.SubCategory = 'Reparation'
#ghenv.Component.AdditionalHelpFromDocStrings = '1'

import rhinoscriptsyntax as rs
import ghpythonlib.components as gh

# Fonctions pour passer des arborescences GH aux listes Python

def list_to_tree(input, none_and_holes=True, source=[0]):

def flatten_list(_2d_list):

def tree_to_list(input, retrieve_base = lambda x: x[0]):

################################################################################
# FONCTIONS UTILITAIRES
def inclusion(C):
    i=0
    while i<len(C):
        pts= gh.Explode(C[i], True)[1]
        if len(pts)<3: pts= gh.DivideCurve(C[i], 10, False)[0]
        del pts[-1]
        j=0
        while j<len(C):
            if j!=i:
                a = gh.PointInCurve(pts, C[j])[0]
                u=0
                for ele in a:
                    if ele>0: u+=1
                if u==len(pts):
                    pt=gh.Area(C[i])[1]
                    pt2= gh.CurveClosestPoint(pt, C[j])[0]
                    pt3= gh.CurveClosestPoint(pt, C[i])[0]
                    v1 = gh.Vector2Pt(pt, pt2, False)
                    l = v1[1]+ gh.Length(gh.Line(pt, pt3))
                    v =gh.Amplitude(v1[0], l)
                    C[i]= gh.Move(C[i],v)[0]
            j+=1
        i+=1
    return C



def list_col(crv):
    # trouver pour chaque agent les index des agents avec qui il entre en collisions
    int = gh.MultipleCurves(crv)
    i=0
    list=[]
```

```
    result=[]
    if int[1]:
        while i<len(int[1]):
            a = [int[1][i], int[2][i]]
            if a not in list:
                list.append(a)
            i+=1


        d=flatten_list(list)
        i=0
        while i<len(crv):
            if i in d:
                index=[]
                for ele in list:
                    if i in ele:
                        for I in ele:
                            if I!=i: index.append(I)
                result.append(index)
            else: result.append([False])
            i+=1

    return result


##########################################################################################
# FONCTIONS POUR DEFINIR DES VECTEURS POUR CHAQUE AGENTS

def col(crv1, crv2):
    # fonction de base pour supprimer les collisions
    pt1= gh.Area(crv1)[1]
    pt2= gh.Area(crv2)[1]
    int = gh.RegionIntersection(crv1, crv2, gh.Plane3Pt(pt1, pt2, gh.EndPoints(crv1)[1]))
    if int:
        pt3= gh.Area(int)[1]
        L = gh.Line(pt1, pt3)
        pt4= gh.CurveXCurve(L, crv2)[0]
        V=gh.Vector2Pt(pt3, pt4, False)[0]
        l = gh.Line(pt3, pt4)
        if gh.Length(l)<gh.Length(crv1)/1000: V=gh.UnitX(0)
    else:
        V=gh.UnitX(0)
    return [V]

def col_ortho(crv1, crv2):
    # seconde fonction pour gérer les collisions (translations uniquement orthogonales)
    pt1= gh.Area(crv1)[1]
    pt2= gh.Area(crv2)[1]
    int = gh.RegionIntersection(crv1, crv2, gh.Plane3Pt(pt1, pt2, gh.EndPoints(crv1)[1]))
    if int:
        pt3= gh.Area(int)[1]
        l1 = gh.LineSDL(pt3,gh.UnitX(1), 1)
        l2 = gh.LineSDL(pt3,gh.UnitY(1), 1)
        l=gh.Length(crv1)/2
        L1=gh.ExtendCurve(l1,0,l,l)
        L2=gh.ExtendCurve(l2,0,l,l)
        pt=[]
        pt4= gh.CurveXCurve(L1, crv2)[0]
        if type(pt4) is list:
            for ele in pt4: pt.append(ele)
        else: pt.append(pt4)
        pt5= gh.CurveXCurve(L2, crv2)[0]
        if type(pt5) is list:
            for ele in pt5: pt.append(ele)
        else: pt.append(pt5)
```

```
        X = gh.Length(gh.Line(pt3, pt))
        x = sorted(X)[0]
        if x<gh.Length(crv1)/1000: V=gh.UnitX(0)
        else:
            V=gh.Vector2Pt(pt3,pt[X.index(x)],False)[0]
    else:
        V=gh.UnitX(0)
    return [V]

def vector_col(crv, ortho):
    # boucle pour trouver les vecteurs de chaque agents en collision entre eux
    L=list_col(crv)

    result=[]
    if L:
        i=0
        while i<len(crv):
            if L[i][0] is False: result.append([gh.UnitX(0)])
            else:
                V=[]
                for ele in L[i]:
                    if ortho is False:v = col(crv[i], crv[ele])
                    else: v = col_ortho(crv[i], crv[ele])
                    V.append(v)
                X=flatten_list(V)
                Y=[]
                if len(X)>1:
                    for ele in X:
                        if ele ==False and ele in Y: pass
                        else: Y.append(ele)
                    X=[]
                    X=Y
                result.append(X)
            i+=1
    return result


def lim_parcelle(lim, crv):
    # boucle pour trouver les vecteurs de chaque agents en collision avec la limite
    V=[]
    PT=[]

    for i in crv:
        A = gh.CurveXCurve(lim, i)
        if A[0] is None or len(A[0])==1:
            V.append([gh.UnitX(0)]
        elif len(A[0])==2:
            B = gh.Shatter(i, A[2])
            PT.append(A[0])
            pt= gh.CurveMiddle(B)
            pt2= gh.CurveMiddle(gh.Line(A[0][0],A[0][1]))
            g=0
            for j in pt:
                if gh.PointInCurve(j,lim)[0]==0:
                    long = gh.Length(gh.Line(j,pt2))
                    Long = long *(1+(5/100))
                    if Long<0.01: Long=0.01
                    v = gh.Amplitude(gh.Vector2Pt(j, pt2, False)[0], Long)
                    V.append([v])
                else: g+=1
            if g==len(pt): V.append([gh.UnitX(0)])
        else:
            V.append([gh.UnitX(0)])
    return V

################################################################################
```

```
# DEFINIR UN VECTEUR UNIQUE PAR AGENT ET BOUGER L'AGENT

def vector_addition(V):
    result=[]
    i=0
    while i<len(V):
        if len(V[i])==1: pass
        else:
            V[i] = gh.MassAddition(V[i])[0]
        i+=1
    return flatten_list(V)

def assembler_vecteur(v1, v2):
    O = gh.ConstructPoint(0,0,0)
    V1= vector_addition(v1)
    V2= vector_addition(v2)
    V=[]
    i=0
    while i<len(V1):
        if V1[i] is False:
            V.append([V2[i]])
        elif V2[i] is False: V.append([V1[i]])
        else: V.append([V1[i],V2[i]])
        i+=1
    result= vector_addition(V)
    return result

def move_crv(crv, V):
    result=[]
    i=0
    while i<len(crv):
        if V[i] is False or V[i]==0: result.append(crv[i])
        else:
            x = gh.Move(crv[i], V[i])[0]
            result.append(x)
        i+=1
    return result

################################################################################
# FONCTIONS PRINCIPALES

def main(crv, n, ortho, lim):
    i=0
    end = False
    while i<n:
        v1 = vector_col(crv, ortho)
        if lim: v2= lim_parcelle(lim, crv)

        if v1 and lim:
            V1 = vector_addition(v1)
            if v2: V2= vector_addition(v2)
            V = assembler_vecteur(V1, V2)

        elif lim:
            if v2: V= vector_addition(v2)
        elif v1: V = vector_addition(v1)
        else:
            end = True
            break
        j=0
        for ele in V:
            if ele!=False:j=1
        if j==0:
            end = True
            break
        X= move_crv(crv, V)
```

```
        crv =[]
        crv = X
        aire = gh.MassAddition(gh.Area(crv)[0])[0]
        crv2 = [gh.RegionUnion(crv)]
        aire2=0
        test=[]
        if lim:
            for c in crv2:
                r= gh.RegionIntersection(c,lim, gh.XYPlane(gh.ConstructPoint(0,0,0)))
                aire2 += gh.MassAddition(gh.Area(r)[0])[0]

            if abs(aire-aire2)<aire*0.0002:
                end = True
                break
        else:
            aire2= gh.MassAddition(gh.Area(flatten_list(crv2))[0])[0]
            if abs(aire-aire2)<aire*0.0002:
                end = True
                break
        i+=1

    return [crv,i, end]


if run is True:

    C2= inclusion(C)
    x = main(C2, 100, True, lim)
    if x[2]==False: x = main(C2, 100, False, lim)
    C =x[0]
    i=x[1]
    a = x[2]
```

Annexe 6 Code du solveur d'optimisation

```
"""This component is a multi-criteria optimization solver
that allows NSGAII to be used with repair functions.
    Inputs:
        genome: List of problem variables (sliders and Genepool only).
        fitness: List of the values of the fitness functions to minimize.
        timer: Boolean that indicates whether or not all
        the evaluation functions have been calculated.
        Use the "AS Eval" component.
        pop_rep: List of the repared genome (in the same order as the genome).
        nb_pop: Size of the population.
        nb_gen: Number of generations.
        co: Constraint Violation Rate.
        reset: Press Button to bring up the input window.
        rate_R: Replacement rate ( a number between 0 and 1).
        const: Constraint management method.
        view_name: Optional input for the Rhino viewport name
        which you would like to take a snapshot of.
        width: Image width in pixel (600 by default).
        height: Image height in pixel (600 by default).
        reset: Restart the algorithm.
        run: Start the algorithm
    Output:
        last_gen: List with fitness values of the last calculated generation.
        gen_repare: Trees containing the repared genes of all generations.
        current_gen: Generation currently under calculation.
        scores: Trees containing the fitness values of all generations.
        genome: Trees containing the genes of all generations.
        nb_ind: Number of individuals assessed in the current generation.
        capture:List of the paths of the screenshots.

"""

ghenv.Component.Name = "Urchin_Optimization_solver"
ghenv.Component.NickName = 'URCHIN Solver'
ghenv.Component.Message = 'Optimization'
ghenv.Component.Category = 'Urchin'
ghenv.Component.SubCategory = 'Optimization'
#ghenv.Component.AdditionalHelpFromDocStrings = '1'

import rhinoscriptsyntax as rs
import ghpythonlib.treehelpers as th
import ghpythonlib.components as gh
from scriptcontext import sticky
import scriptcontext as sc
import subprocess
import random
import math
import time
import json
import os
import csv
import Rhino
import System
import os, shutil
import Grasshopper
from System import Decimal

class GA:
    def __init__(self):
        self.thisDoc = ghenv.Component.OnPingDocument()
        self.sliders = ghenv.Component.Params.Input[0].Sources
        self.ghComp = ghenv.Component
        self.genome_or=[]
        self.genome_rep=[]
```

```
        self.scores=[]
        self.nbGen=0
        self.nbInd=0
        self.results=[]
        self.individus=[]
        self.individus_rep=[]
        self.traces=[]
        self.all_scores=[]
        self.all_var=[]
        self.cv=[]
        self.CV=[]
        self.capture=[]
        self.limB=[]
        self.limH=[]
        self.dec=[]
        self.R=[]
        self.nom_var=[]
        self.Nb=[]
        self.tc=[]
        self.nbFit=0

# Fonctions pour passer des arborescences GH aux listes Python

def flatten_list(_2d_list):

def list_to_tree(input, none_and_holes=True, source=[0]):

########################################################################################################
#FONCTIONS UTILITAIRES

def float_range(start, stop, step):
    result=[]
    while start < stop:
        result.append(float(start))
        start += step
    return result

def start():
    start =True
    if fitness==[]: start =False
    return start

def SampleViewCaptureToFile(name, view_name, width, height):
    # Fonction pour lancer une capture d'écran d'une fnêtre de visualisation Rhino
    for v in sc.doc.Views:
        if v.ActiveViewport.Name == view_name: view=v
    if view:
        view_capture = Rhino.Display.ViewCapture()
        view_capture.Width = width #view.ActiveViewport.Size.Width
        view_capture.Height = height #view.ActiveViewport.Size.Height
        view_capture.ScaleScreenItems = False
        view_capture.DrawAxes = False
        view_capture.DrawGrid = False
        view_capture.DrawGridAxes = False
        view_capture.TransparentBackground = False
        bitmap = view_capture.CaptureToBitmap(view)
        if bitmap:
            folder = System.Environment.SpecialFolder.Desktop
            path = 'C:\solver\capture\solution'
            filename = System.IO.Path.Combine(path, str(name)+'.png');
            bitmap.Save(filename, System.Drawing.Imaging.ImageFormat.Png);
    return path+ str(name)+'.png'

def del_folder(folder):
    # Fonction pour supprimer des éléments des dossiers en arrière plan
    for filename in os.listdir(folder):
```

```
        file_path = os.path.join(folder, filename)
        try:
            if os.path.isfile(file_path) or os.path.islink(file_path):
                os.unlink(file_path)
            elif os.path.isdir(file_path):
                shutil.rmtree(file_path)
        except Exception as e:
            print('Failed to delete %s. Reason: %s' % (file_path, e))

def partition_values(order, values, sliders, Nb):
    Order=[]
    Values=[]
    i=0
    j=0
    while i <len(sliders):
        list1=[]
        list2=[]
        for n in range(0,Nb[i]):
            list1.append(order[j])
            list2.append(values[j])
            j+=1
        Order.append(list1)
        Values.append(list2)
        i+=1
    return [Order, Values]

########################################################################################################
if not timer: timer=True
if not width: width=600
if not height: height=600
if reset:
    sticky['GA']= GA()
    folder = 'C:\solver\capture\solution'
    del_folder(folder)
    folder ='C:/solver/fichiers'
    del_folder(folder)
ga = sticky['GA']

# Démarrage de l'algorithme

if reset:
    ga.nbFit=len(fitness)
    LimB = [float(s.Slider.Minimum) if type(s)== Grasshopper.Kernel.Special.GH_NumberSlider
else float(s.Minimum.ToString()) for s in ga.sliders]
    LimH = [float(s.Slider.Maximum) if type(s)== Grasshopper.Kernel.Special.GH_NumberSlider
else float(s.Maximum.ToString()) for s in ga.sliders]
    nom_var = [str(s.NickName)for s in ga.sliders]
    dec = [1/(math.pow(10,s.Slider.DecimalPlaces)) if type(s)==
Grasshopper.Kernel.Special.GH_NumberSlider else 1/(math.pow(10,s.Decimals))  for s in
ga.sliders]
    r= [s.Slider.DecimalPlaces if type(s)== Grasshopper.Kernel.Special.GH_NumberSlider else
s.Decimals for s in ga.sliders]
    ga.Nb= [1 if type(s)== Grasshopper.Kernel.Special.GH_NumberSlider else s.Count for s in
ga.sliders]

    i=0
    while i<len(ga.sliders):
        j=0
        while j<ga.Nb[i]:
            ga.limB.append(LimB[i])
            ga.limH.append(LimH[i])
            ga.dec.append(dec[i])
            ga.R.append(r[i])
            ga.nom_var.append(nom_var[i]+str(j))
            ga.tc.append(ga.sliders[i].TickCount)
            j+=1
```

```
        i+=1

# Première génération
if run is True and ga.nbGen==0 and timer is True:

    if ga.nbInd==0:
        ga.traces.append('new generation 0')

        order = [random.randint(0, ele) for ele in ga.tc]
        values=[]
        i=0
        while i<len(order):
            v=(order[i]*ga.dec[i])+ga.limB[i]
            values.append(v)
            i+=1

        X= partition_values(order, values, ga.sliders, ga.Nb)
        Order = X[0]
        Values = X[1]

        a = list_to_tree(X)

        for i in range(0, len(ga.sliders)):
            if type(ga.sliders[i])!= Grasshopper.Kernel.Special.GH_NumberSlider:
                for j in range (0, ga.Nb[i]):
                    ga.sliders[i][j] = Decimal(Values[i][j])
                ga.sliders[i].ExpireSolution(True)

        def Allocate(doc):
            for s, o in zip(ga.sliders, Order):
                s.TickValue = o[0]
            ga.ghComp.ExpireSolution(False)

        ga.thisDoc.ScheduleSolution(1, Allocate)
        ga.nbInd +=1



    elif ga.nbInd < nb_pop:

        if view_name:
            for view in view_name:
                name = view+'_Gen'+str(ga.nbGen)+'_Ind'+str(ga.nbInd-1)
                cap=SampleViewCaptureToFile(name, view, width, height)
                ga.capture.append(cap)

        i=0
        while i<ga.nbFit:
            if i<len(fitness):
                if fitness[i] is None:
                    fitness=[999999,999999,999999]
                    break
            else: fitness=[999999,999999,999999]
            i+=1
        ga.results.append(fitness)
        ga.individus_rep.append(pop_rep)
        ga.cv.append(co)
        ga.individus.append(genome)


        TC=[]
        i=0
        while i<len(ga.sliders):
            j=0
            t=[]
            while j<ga.Nb[i]:
```

```
            TC.append(ga.sliders[i].TickCount)
            j+=1
        i+=1

    order = [random.randint(0, ele) for ele in TC]
    values=[]
    i=0
    while i<len(order):
        v=(order[i]*ga.dec[i])+ga.limB[i]
        values.append(v)
        i+=1


    X= partition_values(order, values, ga.sliders, ga.Nb)
    Order = X[0]
    Values = X[1]

    for i in range(0, len(ga.sliders)):
        if type(ga.sliders[i])!= Grasshopper.Kernel.Special.GH_NumberSlider:
            for j in range (0, ga.Nb[i]):
                 ga.sliders[i][j] = Decimal(Values[i][j])
            ga.sliders[i].ExpireSolution(True)

    def Allocate(doc):
        for s, o in zip(ga.sliders, Order):
            s.TickValue = o[0]
        ga.ghComp.ExpireSolution(False)

    ga.thisDoc.ScheduleSolution(1, Allocate)
    ga.nbInd +=1


else:

    if view_name:
        for view in view_name:
            name = view+'_Gen'+str(ga.nbGen)+'_Ind'+str(ga.nbInd-1)
            cap=SampleViewCaptureToFile(name, view, width, height)
            ga.capture.append(cap)

    i=0
    while i<ga.nbFit:
        if i<len(fitness):
            if fitness[i] is None: fitness[i]=999999
        else: fitness.append(999999)
        i+=1

    ga.results.append(fitness)
    ga.individus_rep.append(pop_rep)
    ga.cv.append(co)
    ga.individus.append(genome)

    ga.genome_or = ga.individus
    ga.genome_rep = ga.individus_rep
    ga.CV=ga.cv
    ga.scores = ga.results
    A= ghenv.Component.Params.Input[1].Sources
    nom_fit = [str(s.NickName)for s in A]
    ga.all_scores.append(nom_fit)
    ga.all_scores.append(ga.scores)
    ga.all_var.append(ga.nom_var)
    Nom=[]
    for ele in ga.nom_var: Nom.append(ele)
    for ele in nom_fit: Nom.append(ele)
    if const ==2:ga.all_var.append(ga.genome_rep)
    else: ga.all_var.append(ga.genome_or)
```

```
        if const ==2: export= [ga.genome_or, ga.genome_rep, ga.scores, ga.R, ga.limB,
ga.limH, [], [rate_R], Nom]
        elif const ==1: export= [ga.genome_or, [], ga.scores, ga.R, ga.limB, ga.limH,
ga.CV, [rate_R], Nom]
        elif const ==0: export = [ga.genome_or, [], ga.scores, ga.R, ga.limB, ga.limH, [],
[0], Nom]

        path = "C:/solver/fichiers/externe.txt"
        with open(path,"w") as fichier:
            json.dump(export, fichier)


        if const ==1: Path="C:/solver/scripts/nsgaII_pymoo_ihm_C.py"
        else: Path="C:/solver/scripts/nsgaII_pymoo_ihm.py"
        p= subprocess.Popen(["cmd /c ", Path], shell=True, stdout=subprocess.PIPE,
stderr=subprocess.PIPE)
         out, err = p.communicate()


        ga.nbGen+=1

        path = "C:/solver/fichiers/offspring.txt"
        with open(path,"rb") as fichier:
            X=json.load(fichier)
        ga.genome_or=X

        ga.results = []
        ga.individus = []
        ga.individus_rep = []
        ga.cv=[]
        ga.nbInd = 0
        ga.ghComp.ExpireSolution(True)*

##############################################################################################
# Second et autres générations

elif run and ga.nbGen < nb_gen and timer is True:

    if ga.nbInd==0:
        values = flatten_list([ga.genome_or[ga.nbInd]])

        order=[]
        i=0
        while i<len(values):
            t=(values[i]-ga.limB[i])*(1/ga.dec[i])
            order.append(t)
            i+=1

        X= partition_values(order, values, ga.sliders, ga.Nb)
        Order = X[0]
        Values = X[1]

        for i in range(0, len(ga.sliders)):
            if type(ga.sliders[i])!= Grasshopper.Kernel.Special.GH_NumberSlider:
                for j in range (0, ga.Nb[i]):
                     ga.sliders[i][j] = Decimal(Values[i][j])
                ga.sliders[i].ExpireSolution(True)

        def Allocate(doc):
            for s, o in zip(ga.sliders, Order):
                s.TickValue = o[0]
            ga.ghComp.ExpireSolution(False)

        ga.thisDoc.ScheduleSolution(1, Allocate)
        ga.nbInd +=1
```

```
elif ga.nbInd < nb_pop :

    if view_name:
        for view in view_name:
            name = view+'_Gen'+str(ga.nbGen)+'_Ind'+str(ga.nbInd-1)
            cap=SampleViewCaptureToFile(name, view, width, height)
            ga.capture.append(cap)

    i=0
    while i<ga.nbFit:
        if i<len(fitness):
            if fitness[i] is None: fitness[i]=999999
        else: fitness.append(999999)
        i+=1
    ga.results.append(fitness)
    ga.individus_rep.append(pop_rep)
    ga.cv.append(co)
    ga.individus.append(genome)


    values = flatten_list(ga.genome_or[ga.nbInd])
    order=[]
    i=0
    while i<len(values):
        t=(values[i]-ga.limB[i])*(1/ga.dec[i])
        order.append(t)
        i+=1

    X= partition_values(order, values, ga.sliders, ga.Nb)
    Order = X[0]
    Values = X[1]

    for i in range(0, len(ga.sliders)):
        if type(ga.sliders[i])!= Grasshopper.Kernel.Special.GH_NumberSlider:
            for j in range (0, ga.Nb[i]):
                 ga.sliders[i][j] = Decimal(Values[i][j])
            ga.sliders[i].ExpireSolution(True)

    def Allocate(doc):
        for s, o in zip(ga.sliders, Order):
            s.TickValue = o[0]
        ga.ghComp.ExpireSolution(False)

    ga.thisDoc.ScheduleSolution(1, Allocate)
    ga.nbInd +=1

else:

    if view_name:
        for view in view_name:
            name = view+'_Gen'+str(ga.nbGen)+'_Ind'+str(ga.nbInd-1)
            cap=SampleViewCaptureToFile(name, view, width, height)
            ga.capture.append(cap)

    i=0
    while i<ga.nbFit:
        if i<len(fitness):
            if fitness[i] is None: fitness[i]=999999
        else: fitness.append(999999)
        i+=1
    ga.results.append(fitness)
    ga.individus_rep.append(pop_rep)
    ga.cv.append(co)
    ga.individus.append(genome)
```

```
        ga.traces.append('new generation ' + str(ga.nbGen))
        ga.genome_or = ga.individus
        ga.genome_rep = ga.individus_rep
        ga.CV=ga.cv
        ga.scores = ga.results
        ga.all_scores.append(ga.scores)
        if const ==2:ga.all_var.append(ga.genome_rep)
        else: ga.all_var.append(ga.genome_or)

        if const ==2: export= [ga.genome_rep, ga.scores, ga.R, ga.limB, ga.limH, [],
[rate_R]]
        elif const ==1:export= [[], ga.scores, ga.R, ga.limB, ga.limH, ga.CV, [rate_R]]
        elif const ==0: export = [[], ga.scores, ga.R, ga.limB, ga.limH, [], [0]]

        os.remove("C:/solver/fichiers/offspring.txt")

        path = "C:/solver/fichiers/externe2.txt"
        with open(path,"w") as fichier:
                json.dump(export, fichier)


        if const ==1: Path="C:/solver/scripts/nsgaII_pymoo_loop_ihm_C.py"
        else: Path="C:/solver/scripts/nsgaII_pymoo_loop_ihm.py"
        p= subprocess.Popen(["cmd /c ", Path], shell=True, stdout=subprocess.PIPE,
stderr=subprocess.PIPE)
         out, err = p.communicate()

        ga.nbGen+=1

        if ga.nbGen < nb_gen:
            path = "C:/solver/fichiers/offspring.txt"
            with open(path,"rb") as fichier:
                X=json.load(fichier)
            ga.genome_or=X

        ga.results = []
        ga.individus = []
        ga.individus_rep = []
        ga.nbInd = 0
        ga.cv=[]
        ga.ghComp.ExpireSolution(True)

########################################################################################
#Résultats
last_gen =th.list_to_tree(ga.scores)
gen_repare = th.list_to_tree(ga.genome_rep)
current_gen = ga.traces
scores =th.list_to_tree(ga.all_scores)
genome =th.list_to_tree(ga.all_var)
nb_ind = ga.nbInd
capture = ga.capture
```

Annexe 6 Code pour la génération d'un catalogue de solutions

```
# -*- coding: utf-8 -*-

from PIL import Image, ImageDraw, ImageFont
import json
import copy
import math

path = "C:/solver/fichiers/poster_sol.txt"
with open(path,"rb") as fichier:
    data=json.load(fichier)



path = "C:/solver/fichiers/poster_para.txt"
with open(path,"rb") as fichier:
    para=json.load(fichier)

dpi = int(para[0])
size = int(para[1])
form = int(para[2])
view = para[3]
num = int(para[4])
radar = bool(para[5])
title = str(para[6])


if dpi==75:
    if size == 0:
        width=3511
        height=2483
    if size == 1:
        width=2483
        height=1754
    if size == 2:
        width=1754
        height=1240
    if size == 3:
        width=1240
        height=877
    if size == 4:
        width=877
        height=620
elif dpi==150:
    if size == 0:
        width=7022
        height=4967
    if size == 1:
        width=4967
        height=3508
    if size == 2:
        width=3508
        height=2480
    if size == 3:
        width=2480
        height=1754
    if size == 4:
        width=1754
        height=1240
elif dpi==300:
    if size == 0:
        width=14043
        height=9933
    if size == 1:
        width=9933
```

```
        height=7016
    if size == 2:
        width=7016
        height=4961
    if size == 3:
        width=4961
        height=3508
    if size == 4:
        width=3508
        height=2480

if format==1:
    t = copy.deepcopy(height)
    height=copy.deepcopy(width)
    width=copy.deepcopy(t)

if size == 0: s=int(609/num)
if size == 1: s=int(429/num)
if size == 2: s=int(302/num)
if size == 3: s=int(213/num)
if size == 4: s=int(150/num)


if dpi==150: s=int(s/2)
if dpi==75: s=int(s/4)

def exportPDF(width, height,data,view,file,title, s):

    gen=str(data[0][0])
    ind=str(data[0][1])
    path= 'C:/solver/capture/'+file + view+'_Gen'+gen+'_Ind'+ind+'.png'
    image_sol = Image.open(path)
    Wid_ori = image_sol.size[0]
    Hei_ori = image_sol.size[1]
    Wid_pdf = int(width/num)
    Hei_pdf = int((Wid_pdf*Hei_ori)/Wid_ori)
    num_v = int(math.floor(height/Hei_pdf))
    delta = int(math.floor((height-(num_v*Hei_pdf))/(num_v-1)))
    num_page= math.floor(len(data)/(num*num_v))+1
    myFont = ImageFont.truetype("C:\Windows\Fonts\Arial.ttf", s)
    canvas = [Image.new("RGB", (width, height), color = (255, 255, 255)) for i in
range(0,num_page)]

    p=0
    stop=False
    while p<num_page:

        T=0
        for i in range(0, num_v):
            if stop==True: break
            D=0
            for j in range(0,num):
                if j+(i*num)+(p*(num_v*num))>=len(data):
                    stop=True
                    break
                sol=data[j+(i*num)+(p*(num_v*num))]
                gen=str(sol[0])
                ind=str(sol[1])
                path_sol = 'C:/solver/capture/'+file + view+'_Gen'+gen+'_Ind'+ind+'.png'
                image_sol = Image.open(path_sol)
                image_sol_resized= image_sol.resize((Wid_pdf,Hei_pdf))
                draw = ImageDraw.Draw(image_sol_resized)
                draw.text((10,10), "Generation "+gen+" Individual "+ind, font=myFont,
fill=(0,0,0))
                image_sol_resized.save('C:/solver/capture/'+file +
view+'_Gen'+gen+'_Ind'+ind+'.png')
```

```
                canvas[p].paste(image_sol_resized,(D,T))
                D+=Wid_pdf

            T+=Hei_pdf+delta

        p+=1

    All=[]
    for i in range(1,num_page):
        All.append(canvas[i])

    canvas[0].save('C:/solver/capture/poster/'+title+'.pdf', save_all=True,
append_images=All)



file=[]
for n in view:
    file.append('solution/')


if radar==True:
    view.append('RC')
    file.append('capture_radar/')




if len(view)==1: # avec une seule image

    exportPDF(width, height,data,view,'solution/','test')




if len(view)==2: # avec une deux images

    gen=str(data[0][0])
    ind=str(data[0][1])
    path=['C:/solver/capture/'+file[i]+view[i]+'_Gen'+gen+'_Ind'+ind+'.png' for i in
range(0,len(view))]
    im1 = Image.open(path[0])
    wid = im1.size[0]*2
    hei = im1.size[1]

    pt_can=[Image.new("RGB", (wid, hei), color = (255, 255, 255)) for i in
range(0,len(data))]
    i=0
    while i<len(data):
        sol=data[i]
        gen=str(sol[0])
        ind=str(sol[1])
        path=['C:/solver/capture/'+file[j]+view[j]+'_Gen'+gen+'_Ind'+ind+'.png' for j in
range(0,len(view))]
        im = [Image.open(path[j]) for j in range(0,len(view))]

        can= Image.new("RGB", (int(wid/2), hei), color = (255, 255, 255))
        W = int((hei*im[1].size[0])/im[1].size[1])
        im[1]=im[1].resize((W, hei))
        w = int(((wid/2)-im[1].size[0])/2)
        if w<0: w=0
        can.paste(im[1],(w,0))

        pt_can[i].paste(im[0],(0,0))
        pt_can[i].paste(can,(im[0].size[0],0))
        pt_can[i].save('C:/solver/capture/assembly/'+view[0]+'_Gen'+gen+'_Ind'+ind+'.png')
```

```
        i+=1

    exportPDF(width, height, data, view[0],'assembly/', title, s)

if len(view)==3: # avec une trois images

    gen=str(data[0][0])
    ind=str(data[0][1])
    path=['C:/solver/capture/'+file[i]+view[i]+'_Gen'+gen+'_Ind'+ind+'.png' for i in
range(0,len(view))]
    im1 = Image.open(path[0])
    hei = im1.size[1]*2
    wid = hei*2

    pt_can=[Image.new("RGB", (wid, 2*hei), color = (255, 255, 255)) for i in
range(0,len(data))]
    i=0
    while i<len(data):
        sol=data[i]
        gen=str(sol[0])
        ind=str(sol[1])
        path=['C:/solver/capture/'+file[j]+view[j]+'_Gen'+gen+'_Ind'+ind+'.png' for j in
range(0,len(view))]
        im = [Image.open(path[j]) for j in range(0,len(view))]
        can0= Image.new("RGB", (wid, hei), color = (255, 255, 255))
        can1= Image.new("RGB", (hei, hei), color = (255, 255, 255))
        can2= Image.new("RGB", (hei, hei), color = (255, 255, 255))


        for j in range(0,len(view)):
            W = int((hei*im[j].size[0])/im[j].size[1])
            im[j]=im[j].resize((W, hei))

        w = int((wid-im[0].size[0])/2)
        if w<0: w=0
        can0.paste(im[0],(w,0))

        w = int((hei-im[1].size[0])/2)
        if w<0: w=0
        can1.paste(im[1],(w,0))

        w = int((hei-im[2].size[0])/2)
        if w<0: w=0
        can2.paste(im[2],(w,0))


        pt_can[i].paste(can0,(0,0))
        pt_can[i].paste(can1,(0,hei))
        pt_can[i].paste(can2,(hei,hei))
        pt_can[i].save('C:/solver/capture/assembly/'+view[0]+'_Gen'+gen+'_Ind'+ind+'.png')
        i+=1

    exportPDF(width, height, data, view[0],'assembly/',title, s)


if len(view)==4: # avec une quatre images

    gen=str(data[0][0])
    ind=str(data[0][1])
    path=['C:/solver/capture/'+file[i]+view[i]+'_Gen'+gen+'_Ind'+ind+'.png' for i in
range(0,len(view))]
    im1 = Image.open(path[0])
    hei = im1.size[1]
    wid = hei
```

```
    pt_can=[Image.new("RGB", (wid*2, hei*2), color = (255, 255, 255)) for i in
range(0,len(data))]
    i=0
    while i<len(data):
        sol=data[i]
        gen=str(sol[0])
        ind=str(sol[1])
        path=['C:/solver/capture/'+file[j]+view[j]+'_Gen'+gen+'_Ind'+ind+'.png' for j in
range(0,len(view))]
        im = [Image.open(path[j]) for j in range(0,len(view))]
        can= [Image.new("RGB", (wid, hei), color = (255, 255, 255))for j in
range(0,len(view))]



        for j in range(0,len(view)):
            W = int((hei*im[j].size[0])/im[j].size[1])
            im[j]=im[j].resize((W, hei))

            w = int((hei-im[j].size[0])/2)
            if w<0: w=0
            can[j].paste(im[j],(w,0))



        pt_can[i].paste(can[0],(0,0))
        pt_can[i].paste(can[1],(wid,0))
        pt_can[i].paste(can[2],(0,hei))
        pt_can[i].paste(can[3],(wid,hei))
        pt_can[i].save('C:/solver/capture/assembly/'+view[0]+'_Gen'+gen+'_Ind'+ind+'.png')
        i+=1

    exportPDF(width, height, data, view[0],'assembly/',title, s)


print(s)
```

Claire DUCLOS-PREVET

La modélisation de processus numérique d'exploration en Architecture

Résumé

Les contextes climatiques et réglementaires imposent aux architectes une prise en compte de plus en plus précoce des contraintes environnementales lors la conception de leurs projets. Depuis la fin des années 1990 des méthodes innovantes qui permettent de prendre en compte les paramètres environnementaux en phase amont de projet en s'appuyant sur la puissance de calcul des ordinateurs sont décrites dans la littérature scientifique. Elles suivent toutes une même approche, à savoir l'élaboration d'une technique générative reliant (1) un modèle génératif, souvent paramétrique, pour définir un espace de solutions, (2) un modèle d'évaluation impliquant l'utilisation éventuelle d'outils de simulation, et (3) un modèle d'exploration nécessitant l'usage d'algorithmes métaheuristiques d'optimisation comme les algorithmes évolutionnaires multicritères.

Ces méthodes cherchent à reproduire automatiquement et rapidement, et dans des quantités beaucoup plus importantes, le processus itératif qui existe entre architectes et ingénieurs dans la méthode de conception traditionnelle. Toutefois, si elles sont très populaires auprès des chercheurs, elles restent peu utilisées en agence d'architecture. Les suppositions énoncées dans la littérature pour expliquer ce « gap » entre leur intérêt théorie et leur utilisation en pratique restent à démontrer dans un contexte professionnel de conception architecturale. Le premier objectif de cette thèse est d'expérimenter ces techniques sur des projets réels. La quarantaine d'expérimentations réalisées a permis d'identifier de nouveaux verrous sur l'usage de ces pratiques dans un contexte professionnel jusqu'ici non relevés par la littérature, notamment des limites « structurelles » liées aux méthodes de travail et d'organisation des architectes, mais aussi à la nature même du projet d'architecture.

D'un point de vue technique, le frein majeur rencontré lors de l'usage de techniques d'optimisation s'est révélé être la difficile intégration de contraintes permettant de restreindre l'espace de solutions à des solutions réalisables. Les méthodes de gestion des contraintes génériques se sont révélées inefficaces lors de nos expérimentations sur ce type de problèmes. Ainsi, le second objectif de cette thèse est d'identifier des méthodes de gestion des contraintes adaptées. Une étude comparative de 7 méthodes a permis de révéler que la méthode des fonctions de réparation qui nécessite l'usage de techniques génératives telles que des modèles à base d'agents peut être très efficace. Ainsi, nous avons entrepris le développement d'une bibliothèque d'outils numériques pour Grasshopper permettant aux architectes d'utiliser des techniques génératives pour faire de l'optimisation multicritère sous contraintes.

Mots clés : Design génératif, architecture, performance environnementale, algorithmes évolutionnaires multicritères optimisation sous-contrainte, modèle à base d'agents, machine learning

Résumé en anglais

Climatic and regulatory contexts urge architects to take environmental constraints into account at the earliest stages of design processes. Since the end of the 1990s, the scientific literature has described innovative methods for taking environmental parameters into account using the calculation power of computers. These methods share the same approach based on a generative technique linking (1) a generative model, often parametric, to define a space of solutions, (2) an evaluation model involving the possible use of simulation tools, and (3) an exploration model requiring the use of metaheuristic optimization algorithms such as multi-criteria evolutionary algorithms.

These methods aim at reproducing, automatically and at much larger scale, the iterative process usually established between architects and engineers in the traditional design method. However, while they are very popular with researchers, these methods remain little used in architectural practice. The assumptions made by the literature to explain this "gap" between theory and practice have yet to be tested in a professional architectural design context. The first objective of this thesis is to experiment such techniques on real projects. The forty or so experiments carried out identified new obstacles to the use of these practices in a professional context, hitherto unnoticed in the literature, notably "structural" limits linked to architects' working and organizational methods, but also to the very nature of the architectural project.

From a technical perspective, the major obstacle encountered when using optimization techniques was the difficulty of integrating constraints to restrict the solution space to feasible solutions. Generic constraint management methods proved ineffective in our experiments on this kind of problems. As a consequence, the second objective of this thesis is to identify suitable constraint management methods. A comparative study of 7 methods revealed that the repair function method, which requires the use of generative techniques such as agent-based models, can be very effective. We have therefore undertaken the development of a library of numerical tools for Grasshopper, enabling architects to use generative techniques for multi-criteria optimization under constraints.

Key words: Generative design, architecture, environmental performance, evolutionary multi-criteria algorithms, under-constrained optimization, agent-based model, machine learning