Couplage d'algorithmes d'optimisation par un système multi-agents pour l'exploration distribuée de simulateurs complexes : application à l'épidémiologie

The Nhan Ho

UNIVERSITÉ PIERRE ET MARIE CURIE

École doctorale Informatique, Télécommunications et Électronique (Paris)

Unité Mixte Internationnale de Modélisation Mathématique et Informatiques des Systèmes Complèxes (UMMISCO)

Sujet de la thèse :

Couplage d'algorithmes d'optimisation par un système multi-agents pour l'exploration distribuée de simulateurs complexes : Application à l'épidémiologie

Présentée par : **HO The Nhan**

Thèse de doctorat de spécialité : **Informatique**

Dirigée par : **Jean-Daniel Zucker**

Présentée et soutenue publiquement le 27 juin 2016

Devant un jury composé de :

Directeur de thèse :	**Jean-Daniel Zucker**	Directeur de Recherche, IRD, Bondy
Encadrant de thèse :	**Nicolas Marilleau**	Ingenieur de Recherche, IRD, Bondy
Rapporteurs :	**Guillaume Deffuant**	Directeur de Recherche de l'IRSTEA, Laboratoire d'Ingénieurie pour les Systèmes Complexes
	David Hill	Professeur des Universités, Université Blaise Pascal, Clermont-Ferrand
Examinateurs :	**Nicolas Bredeche**	Professeur des Universités, Université de Pierre et Marie Curie, Paris
	Hong Quang Nguyen	Professeur, IFI-VNU, Hanoi
	Laurent Philippe	Professeur des Universités, Université de Franche-Comté, Besançon

Université Pierre & Marie Curie - Paris 6
Bureau d'accueil, inscription des doctorants
Esc G, 2ème étage
15 rue de l'école de médecine

75270-PARIS CEDEX 06 Tél. Secrétariat : 01 44 27 28 10
Fax : 01 44 27 23 95
Tél. pour les étudiants de A à EL : 01 44 27 28 07
Tél. pour les étudiants de EM à MON : 01 44 27 28 05
Tél. pour les étudiants de MOO à Z : 01 44 27 28 02
E-mail : scolarite.doctorat@upmc.fr

Cette thèse est dèdiée à Maman et Papa,
sans lesquels je n'aurais jamais vu le jour.

Remerciements

En effet, je n'aurais jamais pu aller à ce jour sans le soutien d'un grand nombre de personnes dont la confiance m'ont permis de progresser dans cette phase. Il me sera très difficile de remercier tout le monde car c'est grâce à l'aide de nombreuses personnes que j'ai pu mener cette thèse à son terme.

Je voudrais tout d'abord remercier mon directeur de thèse, M. Jean-daniel Zucker, pour la confiance qu'il m'a accordée en acceptant d'encadrer ce travail doctoral, pour les conseils pédagogiques et scientifiques qu'il a donné à diriger cette recherche.

J'adresse de chaleureux remerciements à mon encardrement de thèse, Nicolas Marilleau, pour toute son aide. J'ai pris un grand plaisir à travailler avec lui car outre son appui scientifique, il a toujours été là pour me soutenir et me conseiller au cours de l'élaboration de cette thèse. J'aimerais également lui dire à quel point j'ai apprécié sa grande disponibilité des délais serrés de relecture des documents que je lui ai adressés. Enfin, j'ai été extrêmement sensible à ses qualités humaines d'écoute et de compréhension tout au long de ce travail doctoral.

Je voudrais remercier M. David Hill et M. Guillaume Deffuant qui m'ont fait l'honneur d'être rapporteurs de ma thèse, ils ont pris le temps de m'écouter et de discuter avec moi. Leurs remarques m'ont permis d'envisager mon travail sous un autre angle.

Je voudrais également remercier M. Nguyen Hong Quang, Professeur à l'IFI, Vietnam d'avoir accepté de m'accueillir dans le laboratoire MSI à Hanoi et de m'avoir accepté d'examiner mon travail. Je le remercier particulièrement pour ses conseils qui m'ont aidé de passer les difficultés de travailler occasionnellement en France et au Vietnam.

Je tiens à remercier M. Laurent Philippe d'avoir accepté de m'accueillir dans le laboratoire LIFC à l'Université de Franche-Comté et de m'avoir accepté d'examiner mon travail. Je le remercier particulièrement pour toutes nos discussions et son relecture qui m'ont accompagné tout au long de mon travail invité à LIFC.

J'associe à ces remerciements M. Nicolas Bredeche, Professeur des Universités à l'UPMC pour avoir accepté de participer à mon jury de thèse en tant que examinateur.

Heureusement que mes parents, ma petite sœur, ma tante, mes amis et mes collègues sont là pour me changer les idées. Ils ont tous cru en moi et ouf ! maintenant j'y suis ! Un grand merci à tous pour m'avoir conduit à ce jour mémorable.

Résumé

0.1 résumé

L'étude des systèmes complexes tels que des systèmes écologiques ou urbains, nécessite souvent l'usage de simulateurs qui permettent de comprendre les dynamiques observées ou d'avoir une vision prospective de l'évolution du système. Cependant, le crédit donné aux résultats d'une simulation dépend fortement de la confiance qui est accordée au simulateur, et donc de la qualité de sa validation. Cette confiance ne s'obtient qu'au travers d'une étude avancée du modèle, d'une analyse de sensibilité aux paramètres et d'une confrontation des résultats de simulation et des données de terrain. Pour cela, pléthore de simulations est nécessaire, ce qui est coûteux du point de vue des ressources mobilisés (temps de calcul, processeurs et mémoire) et est souvent impossible compte tenue de la taille de l'espace des paramètres à étudier. Il est donc important de réduire de manière significative et intelligente le domaine à explorer. L'une des particularités des simulateurs représentatifs de phénomènes réels est d'avoir un espace des paramètres dont la nature et la forme est fonction : (i) des objectifs scientifiques ; (ii) de la nature des paramètres manipulés ; et (iii) surtout du systèmes complexes étudiés. Ainsi, le choix d'une stratégie d'exploration est totalement dépendante du domaine de l'étude. Les algorithmes génériques de la littérature ne sont alors pas optimaux.

Compte tenu de la singularité des simulateurs complexes, des nécessités et des difficultés rencontrées de l'exploration de leur espace de paramètres. Nous envisageons de guider le tâche d'exploration des systèmes complexes en proposant le protocole d'exploration stratifié coopérative GRADEA qui hybride trois algorithmes d'exploration de différents classements dans un même environnement : la recherche en criblage pour zones d'intérêt, la recherche globale et la recherche locale. Différents stratégies d'exploration vont en parallèle parcourir l'espace de recherche pour trouver l'optimum globale du problème d'optimisation et également pour désigner partiellement la cartographie de l'espace de solutions pour comprendre le comportement émergent du modèle. Les premiers résultats du protocole d'exploration stratifié avec un exemple d'algorithmes présélectionnés d'exploration sont appliquées au simulateur du domaine environnemental pour l'aide à la conception de la planification des politiques de vaccination de la maladie rougeole au Vietnam. Le couplage d'algorithmes d'exploration est intégré sur une architecture modulaire à base d'agents qui sont en interaction avec des noeuds de calcul où sont exécutés les simulations. Cet environnement facilite d'une part le rapprochement et l'interaction entre une selection d'algorithmes d'exploration, et d'autre part l'utilisation de ressources de calcul haute performance. L'enjeu résolu jusqu'à ce temps est de proposer, à la communauté, un environnement optimisé où l'utilisateur sera en mesure : (i) de combiner des algorithmes d'exploration adaptés à son cas d'étude ; (ii) et de tirer partie des ressources disponibles de calcul haute performance pour réaliser l'exploration.

Mots clefs : exploration de paramètres ; optimisation de simulations ; calcul haute performance ; épidémiologie ; exploration distribuée ; système multi-agents

Table des matières

Table des figures

Liste des tableaux

Introduction

L'étude des systèmes complexes tels que des systèmes sociologiques, écologiques, ou environnementaux nécessite de plus en plus l'utilisation de modèles complexes (de simulateurs complexes) afin de reproduire des phénomènes réels, les comprendre et permettre la prise de décision. Cela demande l'exploration de l'espace des paramètres des modèles complexes pour valider le modèle, calibrer et caractériser les phénomènes dynamiques ou mesurer son comportement. Néanmoins, aller vers une représentation de plus en plus proche du système réel étudié conduit à ce que les modèles deviennent de plus en plus réalistes, de plus en plus descriptifs, mais surtout ils embarquent une dynamique de plus en plus complexe. En conséquence, l'exploration du modèle devient aussi compliquée compte tenu de l'explosion du nombre de paramètres, de la méconnaissance de leur rôle dans le modèle et du coût de calcul.

Dans ce contexte l'utilisation de moyens de calcul haute performance (HPC) s'avère une nécessité. D'autre part, des plate-formes spécialisées comme Kepler [ABE+09a], OpenMole [RCL+10] ou SimExplorer [CDFD10] facilitent l'accès à ces ressources. Ces dernières affichent des fonctionnalités intéressantes, en rendant transparents la parallélisation et le déploiement de plans d'expériences sur clusters, grilles ou serveurs isolés de calcul. Ainsi, la soumission d'une multitude d'expérimentations sur un cluster est possible par des chercheurs venant d'autres horizons [SRC11].

Malgré les avancées technologiques (capacité de calcul croissante, algorithmes optimisés), elles ne suffisent pas à répondre à cette explosion de la complexité dans les modèles si bien qu'il est impossible d'avoir une connaissance totale du comportement de ces modèles. Il est alors alors nécessaire d'en faire une analyse approfondie sans pour autant en faire une exhaustive. Il s'agit alors de détecter les variables significatives ou trajectoires importantes ; de calibrer les paramètres pour minimiser une erreur ou pour désigner l'espace de réponses ; de calibrer automatiquement des grandeurs caractéristiques du système pour caractériser les phénomènes dynamiques et de mesurer le comportement des simulations correspondantes. Cela se concrétise par l'exploration de l'espace des paramètres. Une exploration complète de l'espace des paramètres d'un simulateur s'avère généralement impossible compte tenu du nombre de simulations à réaliser et de la durée de chacune d'elles : il faut réfléchir à accélérer l'exploration.

L'utilisation des algorithmes d'exploration est une solution prometteuse pour l'exploration de l'espace de paramètres des simulateurs complexes. Cependant, différents algorithmes d'exploration ont des comportements et des effets différents quand ils sont appliqués sur l'exploration de l'espace de paramètres.

Le choix d'une stratégie d'exploration entre différentes classes (la recherche locale, la recherche globale ou la recherche par criblage) s'effectue au cas par cas selon les attentes du scientifique et les données de terrain en présence. C'est pourquoi quelques approches interactives d'exploration ont vu le jour. Elles reposent sur l'expérience du scientifique et sur sa

participation tout au long du processus d'exploration d'un simulateur. D'autres approches tentent de combiner, au cas par cas, des algorithmes traditionnels d'exploration pour constituer un nouvel algorithme adapté à la problématique abordée par le simulateur. Cependant, cette association d'algorithmes entraîne un important travail de mise au point.

Cette thèse s'intéresse à l'exploration distribuée de modèles complexes en combinant justement calcul haute performance et mise en place de stratégies efficaces d'exploration. Compte tenu de la singularité des simulateurs complexes, des nécessités et des difficultés rencontrées lors de l'exploration de leur espace de paramètres, nous souhaitons guider la tâche d'exploration des systèmes complexes en proposant une démarche d'exploration coopérative qui associe trois stratégies d'exploration différentes dans un même environnement. Ces stratégies d'exploration vont en parallèle parcourir l'espace de recherche, se coordonner pour : (i) trouver l'optimum global du problème d'optimisation ; (ii) et dessiner une cartographie partielle de l'espace de solutions. Afin de mettre en place ce protocole, il nous faut faire face à plusieurs verrous auxquels nous tentons de répondre dans cette thèse :

— **Modularité des calculs :** comment intégrer différents simulateurs complexes de façon générique sur une plate-forme pour les rendre réutilisable ? L'enjeu est de permettre de générer les jeux de paramètres à partir d'un ensemble de paramètres d'entrée, et d'assurer la capacité de distribuer l'exécution des simulations sur différents systèmes de calcul intensif.
— **Modularité des algorithmes d'exploration :** comment intégrer différents algorithmes d'exploration sur une plate-forme et rendre ces algorithmes réutilisables dans différentes expérimentations ?
— **Coordination des algorithmes d'exploration :** comment concevoir une stratégie de coopération entre les algorithmes d'exploration pour que ces derniers se coordonnent ? Cela implique d'identifier un protocole de communication générique pour que ces différents algorithmes partagent des informations dans une phase d'exploration coopérative.

En réponse, nous proposons une architecture originale que nous décrivons dans cette thèse. Pour cela, nous nous reposons sur une architecture modulaire à base d'agents en interaction avec des nœuds de calcul où sont exécutées les simulations. Cette architecture s'appelle GRADEA.

La plate-forme GRADEA est appliquée à un simulateur du domaine de l'épidémiologie s'intéressant à la planification des politiques de vaccination de la maladie rougeole au Vietnam.

Contributions du mémoire

Afin d'avoir une vue d'ensemble de nos travaux, nous avons découpé ce mémoire en sept chapitres :

Chapitre 1 : Simulateurs complexes
Ce chapitre fait une analyse portant sur les concepts importants du domaine de la simulation des systèmes complexes. L'enjeu est de raffiner la problématique de cette thèse en identifiant les besoins auxquels nous devons répondre. Alors, nous aborderons dans ce chapitre la notion de système complexe ; ensuite nous déterminons les spécificités des simulateurs dans ce contexte ; et finalement, nous illustrons nos propos au travers de plusieurs cas d'application abordant des problématiques d'actualité et de terrain.

Chapitre 2 : Méthodes d'exploration des simulateurs complexes

Ce chapitre présente une revue des travaux existants sur l'exploration des simulateurs complexes. Nous présentons premièrement les définitions générales, différents enjeux de l'exploration et nous nous focalisons sur l'exploration des simulateurs complexes ; deuxièmement, les méthodes d'exploration sont présentées dans leur contexte ; finalement, différents algorithmes disponibles seront qualifiés en fonction du comportement d'exploration sur l'espace de recherche.

Chapitre 3 : Techniques et outils pour l'exploration large échelle de simulations complexe
Ce chapitre présente une revue de travaux existants sur les techniques et outils pour permettre un passage à l'échelle. Ce chapitre s'organise comme suit : premièrement, une classification des techniques de passage à l'échelle est présentée ; deuxièmement, les différentes stratégies utilisées pour l'exploration à large échelle des simulateurs complexes ; finalement, la disponibilité de l'exploration large échelle sur les systèmes de calcul intensif.

Chapitre 4 : La solution conceptuelle de la coordination des méthodes d'optimisation pour l'exploration coopérative
Ce chapitre premièrement fait une analyse de l'état de l'art afin de mettre en avant les verrous scientifiques compte-tenu de la contribution de cette thèse. La deuxième partie du chapitre va définir les contours de cette thèse, sa problématique et ses verrous scientifiques compte-tenu des travaux existants. Finalement, la troisième section du chapitre vous présente, succinctement, la solution que nous avons imaginée pour résoudre les problèmes identifiés : l'approche GRADEA pour explorer le modèle de système complexe.

Chapitre 5 : Méta-Modèle de GRADEA
Ce chapitre présente en détail l'architecture conceptuelle de plate-forme GRADEA que nous avons présentée en tant que notre contribution de la thèse dans le chapitre 4. La première partie du chapitre fait une description du méta-modèle de GRADEA. Ensuite nous présentons la solution conceptuelle à base d'agents en suivant l'approche de modélisation VOYELLES.

Chapitre 6 : Plate-forme GRADEA
Ce chapitre présente l'implémentation précise de GRADEA à partir de la description conceptuelle établie dans le chapitre 5. Cette dernière se compose d'une architecture logicielle de GRADEA et d'un guide d'implémentation des nouveaux modules dans GRADEA. L'architecture logicielle repose sur une architecture modulaire des stratégies d'exploration et de calcul. Après ça, on explique comment intégrer les nouveaux modules d'exploration et les modules de calcul sur GRADEA.

Chapitre 7 : Application GRADEA : l'exploration du modèle de vaccination de la maladie rougeole au Vietnam
Dans ce chapitre, nous utilisons GRADEA pour évaluer le modèle de vaccination de la maladie rougeole au Vietnam et pour évaluer la performance de l'exploration coopérative dans GRADEA.

Première partie

Étude de l'exploration parallelisée des simulateurs complexes

Chapitre 1

Simulateurs complexes

Sommaire

A partir d'une analyse de la littérature, nous identifions et clarifions les concepts importants du domaine de la simulation des systèmes complexes. L'enjeu est de raffiner la problématique de cette thèse en identifiant les besoins auxquels cette thèse doit répondre.

Ainsi, nous aborderons dans un premier temps la notion de "système complexe". Ensuite, nous verrons quels sont les spécificités des simulateurs dans ce contexte. Finalement, nous illustrons nos propos au travers de plusieurs cas d'application abordant des problématiques d'actualité et de terrain.

1.1 Notions conceptuelles

1.1.1 Systèmes complexes

Nous sommes obligés d'admettre qu'il existe beaucoup de systèmes constitués d'un grand nombre d'éléments, organisés, faisant émerger des phénomènes inattendus qui ne sont pas la somme des caractéristiques et des dynamiques individuelles. On qualifie ces systèmes de «complexes».

Les systèmes complexes se distinguent des *«systèmes non complexes »* par leur caractère naturellement non prédictible. Compte tenu du nombre d'interactions intrinsèques conduisant à l'émergence d'un comportement, seul le modèle et l'analyse de ce modèle (par le calcul mathématique ou la simulation) permettent d'élaborer des prédictions aidant à la décision. Les systèmes non complexes peuvent quand à eux être analysés sans nécessité l'utilisation d'un modèle.

Il peut être utile de distinguer un système complexe d'un système compliqué avant de proposer une définition concise de la complexité. Pour cela, prenons l'exemple d'un avion de ligne et d'une colonie de fourmis, tous les deux sont des systèmes qui comportent des éléments.

— Un avion est un système compliqué car un entraînement approfondi permet de faire voler un avion et de prédire toutes les conséquences d'un problème.
— Une colonie de fourmis est quant-à elle un système complexe parce qu'il est impossible de prédire précisément ce qui se passe à l'intérieur d'une colonie. Les fourmis qui communiquent via des phéromones afin de se repérer et de transporter de la nourriture vers la fourmilière. Dans ce contexte seuls les modèles (tels qu'ils sont proposés dans les plates-formes comme Netlogo [Wil99] ou Gama [GTG+13]) et l'exécution de ces derniers (souvent par simulation) permettent de comprendre la dynamique du phénomène étudié et de l'anticiper.

Il n'existe ni de définition concise d'un système complexe, ni de définition établie sur laquelle tous les scientifiques s'accordent. Les productions des réseaux pluridisciplinaires nationaux et internationaux tel que le RNSC (Réseau National des Systèmes Complexes) et le SCS (Society of complex system), les propositions de quelques chercheurs dans [Sim62, MdW04, Rou07, CBP+09] établissent une définition basée sur leur axe de recherche, tout en faisant un effort sur la généricité.

Au début, les sciences de la complexité, [Sim62] et [MdW04] se sont principalement concentrées sur le caractère décomposable de la complexité pour la modélisation d'un système complexe. Plus tard, [Rou07] donne une définition dans le domaine du médical et la sécurité, domaines où la compétitivité industrielle est l'objectif principal tandis que [CBP+09] a mis son attention sur les disciplines scientifiques. Il donne une vision large de la complexité en abordant les thèmes de la physique théorique, des systèmes microbiens, des sciences de la vie, de l'écologie, de l'économie, de la sociologie et du management.

[Rou07] considère dans son rapport avec l'Académie nationale de l'Ingénierie aux États-Unis que l'on a besoin d'une éducation plus large et plus flexible qui permet de comprendre la complexité des systèmes comme les services médicaux, l'infrastructure, l'environnement, la sécurité ou l'économie au sein desquels leurs technologies sont déployées. La capacité de comprendre et de traiter des systèmes complexes nécessite une considération de nombreux phénomènes qui ne sont pas intrinsèquement considérés comme des phénomènes d'ingénierie. Les communautés des sciences traditionnelles d'ingénierie trouvent qu'il est très difficile à adapter

à ces nouveaux besoins éducatifs. Il donne la définition dans ce contexte : *«A system whose perceived complicated behaviors can be attributed to one or more of the following characteristics : large numbers of elements, large numbers of relationships among elements, nonlinear and discontinuous relationships, and uncertain characteristics of elements and relationships. Complexity is perceived because apparent complexity can decrease with learning. »* [Rou07]. C'est-à-dire *Un système complexe est un système dont les comportements complexes perçues peuvent être attribués à une ou plusieurs des caractéristiques suivantes : un grand nombre d'éléments, un grand nombre de relations entre les éléments des relations non linéaires et discontinues, et les caractéristiques incertaines des éléments et des relations. La complexité est perçue à cause de la complexité apparente qui peut diminuer avec l'apprentissage.*

Dans le cadre de cette thèse, nous nous intéresserons principalement à la définition proposée par le RNSC dans le «French Complex Systems Roadmap» [CBP+09] : *«A complex system is in general any system comprised of a great number of heterogeneous entities, among which local interactions create multiple levels of collective structure and organization »*. C'est-à-dire *les systèmes complexes sont composés d'un grand nombre d'entités hétérogènes interagissant localement et faisant émerger des niveaux multiples de structure et d'organisation.*

Cette définition indique qu'un grand nombre d'éléments constitutifs d'un système complexe en interaction empêchent l'observateur de prévoir son comportement ou son évolution et s'accorde sur le fait que la complexité d'un système ne se découpe pas, s'apprécie dans sa globalité.

Plusieurs caractéristiques permettant de discerner les systèmes complexes ont été identifiées dans [Par11] :

- Un système complexe comporte plusieurs composantes interdépendantes. Il est possible que les entités constitutives soient homogènes mais le plus souvent, elles sont hétérogènes avec une structure interne qui ne peut être négligée.
- Il se caractérise par la nature des interactions. Ces interactions sont catégorisées en deux types : (1) les interactions entre les éléments constitutifs (le système) ; et les interactions entre le système et le reste du monde (l'environnement / l'extérieur).
- Les dynamiques d'un système complexe appartiennent à l'ensemble du système entier sans appartenir à aucun des constituants. Les propriétés ou les comportements globaux d'un système ne peuvent pas être anticipées à partir des propriétés ou comportements des différentes parties. Il faut comprendre le système pour étudier les «phénomènes collectifs» ou de «propriétés émergentes».

Ces caractéristiques donnent lieu à des phénomènes propres aux systèmes complexes.

- *Émergence multi-niveaux* : une propriété émergente d'un système complexe est observée à une échelle différente de celles des éléments qui la construisent. Plus exactement, un phénomène émergent est observé à une échelle supérieure et ne peut pas être visualisé au niveau individuel [CGB+12]. Ces propriétés émergentes se produisent quand nous observons des phénomènes émergents globaux lors d'expérimentations ou de simulations informatiques à base d'agents. Dans le modèle de Schelling [Sch69] présenté dans [DL06], la ségrégation spatiale dans les zones résidentielles, lorsque les agents ne sont pas satisfaits avec leur position actuelle (ou voisinages), ils choisissent aléatoirement un nouveau lieu de résidence. De toute évidence, les véritables raisons d'un déménagement sont loin d'être aléatoire (proximité au travail, qualité de vie du quartier, etc). Pour contrôler la discrimination au niveau micro du modèle, on utilise le paramètre de pourcentage de la similarité souhaitée. On simule jusqu'à ce que tous les agents soient satisfaits et ensuite, on évalue le pourcentage de similarité. Par exemple,

dans le modèle de ségrégation, les liens dynamiques entre le niveau micro et macro ne sont pas pris en compte. L'action d'un individu au comportement intégrateur dépend de l'action d'un autre individu de même comportement. La ségrégation observable à un niveau agréé est le résultat de processus interdépendants au niveau micro.

— *Multi-échelle et évolution* : Il existe au moins de deux nivaux descriptifs : une locale et une globale. Cependant, des descriptions intermédiaires peuvent s'avérer nécessaires. Il faut considérer différentes échelles d'organisation que peuvent présenter les systèmes. Ce sont à la fois l'évolution des éléments au sein de chaque niveau, mais également les rapports qu'entretiennent les différents niveaux entre eux. L'émergence du système évolue à travers des niveaux et au cours du temps. Dans ces niveaux, les entités interagissent à l'échelle locale de temps et d'espace. Mais le point de vue va changer quand on observe le système entier au niveau global. Par exemple quand on veut modéliser un système complexe biologique, on peut considérer plusieurs niveaux d'échelle pour présenter le système, si on souhaite modéliser une cellule, il y a par exemple trois niveaux : le niveau des atomes, le niveau des molécules (ADN, protéines) et le niveau des organites. Dans ce cas, une question intéressante est posée : comment les molécules opèrent au sein du niveau cellule ou comment les cellules et les tissus échangent les informations ?

— *Sensibilité aux conditions initiales et dynamiques non-linéaires* : On considère que dans un système complexe, de très petites différences dans les conditions initiales produisent des différences significatives aux niveaux des sorties quantitatives et qualitatives. L'émergence est conséquence de la dynamique non linéaire des interactions si bien que : (i) la sortie d'un système complexe n'est pas proportionnelle à son entrée ; (ii) et donc la dynamique observée n'est pas la simple somme des comportements locaux. Dans les systèmes complexes, la sensibilité aux conditions initiales ainsi que la dynamique non linéaire signifie qu'une petite perturbation peut provoquer un effet important. Par exemple dans le modèle de fourragement, les fourmis laissent des phéromones sur un trajectoire lorsqu'elles rapportent de la nourriture à la fourmilière pour aider les autres à trouver le chemin à la source de nourriture. Nombre de paramètres contrôlent le modèle tels que : vitesse des fourmis, taux d'évaporation des phéromones, taux de diffusion des phéromones. Un phénomène émergent visualisable est le comportement fourragement : une recherche aléatoire de nourriture ou une recherche via des files de fourmis. Les travaux de Calvez [CH06] montrent la sensibilité aux conditions initiales et dynamiques non-linéaires. Si le taux d'évaporation est trop fort, cela amène qu'il n'y a pas de phéromones laissées, les fourmis effectuent une recherche aléatoire. Si le taux d'évaporation est faible (proche de zero), les phéromones saturent tout l'environnement, on observe alors une recherche aléatoire des fourmis. C'est seulement quand l'évaporation est moyenne que des files de fourmis sont observables. Quand l'évaporation est moyenne, on peut s'intéresser plus précisément à l'influence du changement des valeurs de deux paramètres de la diffusion et de l'évaporation sur la dynamique des lignes de fourmis. Il y a trois modèles avec de petites modifications pour les deux paramètres : le nombre de files de fourmis change entre 1 file avec [diffusion :60, évaporation :20], 2 files avec [diffusion :50, évaporation :15] et 3 files avec [diffusion :40, évaporation :15]. Ces petites variations peuvent conduire à des dynamiques différentes dans l"exploitation des sources alimentaires.

Nous avons présenté quelques définitions de systèmes complexes ainsi que certaines caractéristiques émergentes pour l'étude des systèmes complexes. Nos citations ne couvrent pas toutes les caractéristiques possibles d'un système complexe mais portent uniquement sur les critères qui nous intéressent. Dans la partie suivante, nous vous présentons les études dans le domaine de systèmes complexes.

1.1.2 Modélisation et simulation de systèmes complexes

L'avènement des TIC connu ces 20 dernières années favorise aujourd'hui la conception, le développement et l'exécution de modèles et de méthodes statistiques et heuristiques d'exploration. Leur objectif est de reproduire, par simulation, un phénomène en vue de le comprendre et de prendre une décision dans les études prospectives. L'étude des systèmes complexes, avec les modèles de systèmes complexes et les moyens analytiques et numériques [TDZ08], vise précisément à comprendre et à prédire les comportements (niveau «macroscopique») sur plusieurs échelles d'espace et de temps, à partir des caractéristiques des éléments constitutifs (niveau «microscopique»).

Modèles de systèmes complexes

Dans le domaine des sciences sociales, l'étude des systèmes complexes requiert premièrement une source à représenter dont on souhaite reproduire une dynamique, des connaissances sur le système et un modèle qui reproduit la dynamique du système complexe. Un tel modèle, selon [Min65], *«est une abstraction simplifiée, systémique et dynamique du domaine d'objet formalisée à l'aide d'un langage non ambigu, qui sert un but précis : un modèle existe pour fournir un outil pour répondre à certains buts, questions ou aspects particuliers du système que l'on cherche à étudier par l'observation et la manipulation du modèle »*. Nous allons utiliser la notion de «modèle » pour simplifier la notion «modèle informatique dynamique » dans la suite du mémoire. Cela veut dire que dans ce mémoire de thèse, on n'aborde pas le modèle statique qui représente uniquement le système à un moment donné. D'autre part, ce mémoire n'aborde pas les modèles non-informatiques qui ne sont pas exécutables par simulation.

Choix de la complexité d'un modèle

La complexité n'est pas une qualité intrinsèque d'un système mais plutôt est une propriété attribuée au système via des modèles. Dans plusieurs cas, la complexité ne peut pas être prévue mais doit être analysée au cours des simulations. Elle se distingue donc de la *complexité de l'algorithme* qui correspond à une évaluation du nombre d'opérations effectuées par le calcul pour aboutir à un résultat et se distingue également d'autres types de complexité abordés dans la vision de (complexité mathématique, complexité de Von Neumann) [DBC+15].

Dans le cadre de cette thèse, visant à proposer des méthodes et outils adaptés pour explorer des modèles de systèmes complexes, nous adoptons un point de vue extérieur au modèle. Un modèle est une boîte noire que nous explorons. Dans ce contexte, la complexité peut être catégorisée en deux types [Mon08] :

— *La complexité des processus :* Cette complexité permet de décrire et d'expliquer les évolutions du système. Celle-ci assure qu'on ne simplifie pas les caractéristiques qui peuvent être la clé d'une certaine propriété que l'on ne veut pas rater. Il faut prendre en compte différentes dynamiques : biologiques, écologiques, sociales, économiques. Cette complexité laisse à l'explorateur des contraintes d'analyse : nombres des paramètres qui sont difficiles à tracer.
— *La complexité des organisations :* Cette complexité se situe au niveau de la représentation. La plupart des problèmes environnementaux et épidémiologiques veulent être proches de la réalité doivent être spatialisés (différentes échelles d'espace et de temps), multi-niveaux (différents intervenants, hybrides), différents points de vue (en vue in-

dividuelle ou collective des constituants). Cette complexité laisse à l'explorateur des contraintes de simulation : temps d'exécution et mémoire.

Le choix de la complexité d'un modèle est important pour approcher virtuellement un phénomène réel et caractériser sa dynamique. En effet la définition du niveau d'abstraction et du niveau de détail est intimement lié à la question posée et aux enjeux scientifiques.

Le niveau de la complexité d'un modèle du système dépend de la question posée. Mais elle est nécessaire pour éviter toute réticence, comparer les résultats avec les données de terrain et améliorer la confiance vis à vis des conclusions. Néanmoins, les modèles peuvent être compliqués, c'est-à-dire longs et difficiles à construire et leurs comportements aussi complexes que les phénomènes qu'ils sont censés représenter. [Aum07].

Cette partie vise à étudier les définitions d'un modèle de systèmes complexes. Il se caractérise par un système cible avec une question sur ce système. La réponse à cette question nécessite la construction d'un modèle (abstraction) de ce système. On essaie de construire la description abstraite, un modèle réduit à certains aspects, d'un système que l'on décrit pour rendre ce modèle plus facile à manipuler que le système source. Néanmoins, un modèle est relatif à un but précis dès le départ et est toujours imparfait, c'est pourquoi il faut explorer le modèle pour que : le modèle implémenté corresponde aux spécifications, corresponde aux attentes des utilisateurs et/ou aux propriétés attendues. L'exploration des modèles complexes est présentée dans la partie suivante.

1.1.3 Explorer des modèles par simulation

La modélisation des systèmes complexes est souvent indissociable de l'exploration des modèles [AP06]. Il faut explorer des modèles pour (1) comprendre le système modélisé (élaboration d'hypothèses, simulation prospective, formalisation/vérification de théories sociologiques) ; et pour (2) décider (simulation prédictive pour l'aide à la décision, test de scénarios par la simulation et artefacts pour l'aide à la négociation ou gestion coordonnée).

Traditionnellement, on utilise l'étude analytique des systèmes d'équations différentielles. Néanmoins, il n'y a pas toujours les compétences ou les outils mathématiques pour faire une exploration mathématique (analytique) à cause du (1) non-existence ou de la complexité des solutions analytiques (grande dimension des systèmes, non-existence de formulation analytique, le manque d'exactitude des règles de décision pour un modèle en réalité, la difficulté de résoudre analytiquement les équations associées), (2) la stochasticité des modèles et (3) l'inflexibilité au changement de variable d'état pour les propriétés émergentes. L'exploration analytique devient impossible. Explorer des modèles par simulation informatique est alors une solution raisonnable.

La simulation informatique représente le système via les structures des entités et dépend de l'interconnexion entre eux en utilisant la théorie évolutionniste des jeux [LACJ03], la simulation cellular automata, la simulation Monte-Carlo, la simulation à base d'agents [Mat]. La technique s'intéresse plus à l'aptitude à apprendre et à créer un réseau complexe en interaction des acteurs constituant un système plutôt qu'à la complexité relative des systèmes et de leurs éléments. Explorer des modèles par simulation est très nouveau mais plus utilisé depuis peu de temps pour l'étude de systèmes complexes dans la science sociologique (socio-environnementale, socio-écologique, sécurité environnementale) où nombreux agents très hétérogènes interagissent, s'auto-organisent, et génèrent des propriétés émergentes du système dans un environnement physique. En outre, l'environnement physique (le territoire) des modèles géographiques dynamiques est un acteur très influencé dans un modèle. Cette

méthode permet récemment de considérer l'environnement physique comme un système complexe [Moi06] et de l'intégrer dans le modèle à partir des données de terrain comme un système informatique géographique (SIG). Cependant, cette technique a ses limites : (i) sur la réalisation : il est très difficile, voire impossible, de résoudre un système analytiquement (spécialement avec un système multi-agent) et (ii) sur l'utilisation : la demande de ressources de calcul coûteuse, la limite de multi-utilisation et la demande de validité avec les données réelles. Dans ce cas, on considère le modèle comme une boîte noire, il faut avoir une approche raisonnée pour explorer des systèmes complexes par simulation : approche non exhaustive, approche statistique, théorie des plans d'expériences ou méthode indépendante au modèle.

Vers un cycle de modélisation par simulation des systèmes complexes

La simulation informatique interactive, surtout la simulation à base d'agents, est l'une des techniques des plus évoluées pour étudier des systèmes complexes et pour calculer les résultats face à des changements dans les paramètres [Joh08]. Un premier courant distingue l'étude de systèmes complexes par deux processus différents pour l'étude d'un système complexe : la "*modélisation*" et la "*simulation*. Tandis que la modélisation est un processus visant à concevoir des modèles par l'usage de méta-modèles, la simulation est un processus visant à animer ces modèles par l'usage d'un programme informatique, d'un simulateur ou d'une plate-forme telle que GAMA, Netlogo, Repast. Il s'agit, alors de manipuler le modèle via ses paramètres pour comprendre le comportement du système, appréhender ses caractéristiques dynamiques ou évaluer différentes décisions [Hil93].

Mais, selon [Fis95], la modélisation et la simulation sont indissociables dans un cycle de modélisation par simulation (figure 1.1). La figure 1.2 est une analyse des processus de modélisation par simulation de la littérature afin de mettre avant le processus d'exploration pour le clarifier. Fishwick s'appuie sur trois étapes : *l'élaboration de la simulation, l'exécution du modèle sur ordinateur et l'analyse des résultats obtenus*. Cette méthode générique a été source d'inspiration pour de nombreux travaux dont [DVM03], [TCM$^+$06], [Aum07] ou [Mor09]. Celles-ci sont d'accord que la modélisation par simulation de systèmes complexes est un processus cyclique : le modèle est progressivement amélioré en introduisant des nouvelles connaissances et en corrigeant les effets indésirables. Toutes ces approches établissent un déroulé du processus adapté à leur domaine d'application. Elles respectent néanmoins l'idée originelle de [Fis95] :

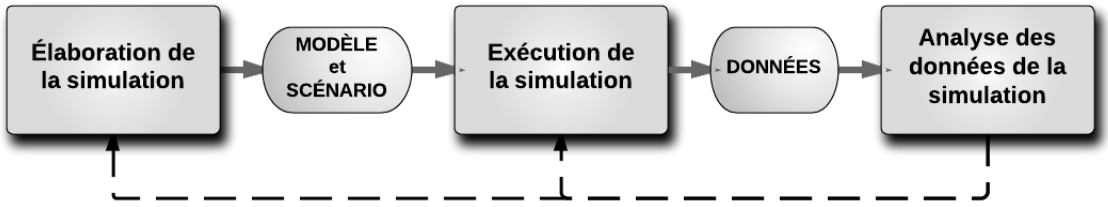

Figure 1.1 – Le processus de modélisation de [Fis95]

— *Élaboration de la simulation* : Cette première étape vise à élaborer et à créer une première abstraction du problème à partir des connaissances de la littérature, d'expertises, de données observées du système et des questions posées. L'objectif de cette étape est d'appliquer des méthodes et outils de modélisation pour caractériser à la fois la structure du système complexe et sa dynamique. Ainsi que la programmation est réalisée. Le résultat de cette étape est un modèle A et des scénarios d'utilisation B. Le modèle A repose sur des algorithmes qui réalisent le calcul de la simulation avec les

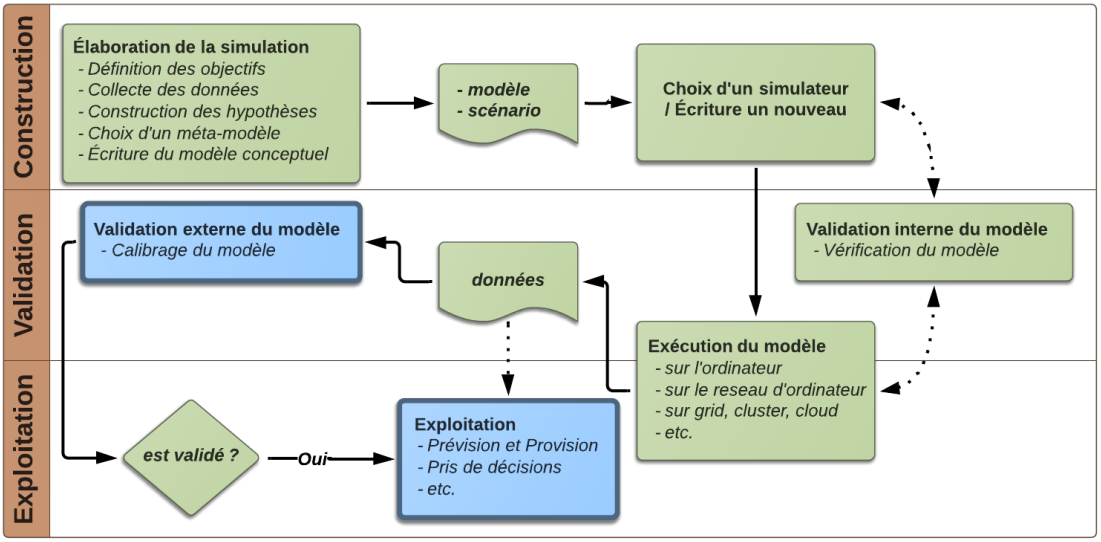

Figure 1.2 – Le processus de modélisation modifié inspiré des travaux dans [Fis95, DVM03, TCM+06, Rio09, Mor09]

données de l'état courant d'une simulation pendant que le scénario d'utilisation de B est déterminé par les paramètres qui indiquent l'état initial de la simulation. [Rio09].

— *Exécution de la simulation* : Cette deuxième étape vise à réaliser des simulations en tenant compte des scénarios précédemment choisis pour reproduire la dynamique du système réel. Pour cela, on a besoin de l'outil exécutable appelé *simulateur*. Le simulateur est une transformation de la spécification du modèle en une suite d'instructions exécutables par un ordinateur. Il correspond donc à l'implémentation du modèle [TDZ08]. C'est un code de calcul constitué (figure 1.3) de : (1) modèles élaborés ; (2) des variables d'entrée descriptives de l'état du système simulé selon un scénario ; et (3) les observables. Les variables d'entrée définissent : (i) les conditions initiales du système (paramètres caractérisants) ; (ii) certaines lois ou relations utilisées (contraintes) ; et (iii) tout ce qui influe sur le comportement du modèle. Les observables sont : les variables, les attributs souhaités au cours de la simulation ou des variables agrégées, les indicateurs, construits à partir de la simulation [DBQ12] désirés dans la modélisation. Ainsi, le simulateur peut être vu comme une boîte noire en prennant des paramètres en entrée et délivrant en sortie les réponses sur les variables d'état du système en fonction du temps. Alors, une simulation consiste à fixer un ou plusieurs jeux de variables d'entrée, réaliser les calcul, puis à analyser les réponses fournies par le simulateur [Ngu13]. La simulation permet une exploration des comportements possibles du système au travers des scénarios qui permettent de tester l'influence de configurations initiales multiples et cela dans un temps limité.

— *Analyse des données de la simulation* : cette troisième étape vise à analyser les données obtenues. Cette étape est constituée par la vérification, la validation et la critique [Aum07]. Plusieurs méthodes dans les sciences expérimentales sont utilisées pour réaliser cette phase telles que l'analyse analytique, statistique ou heuristique. Dans ce domaine, une *expérience de simulation* est définie comme une simulation dans la deuxième étape. Dans cette étude, les observations réalisées au cours de l'expérimentation sont alors rapprochées des hypothèses conceptuelles ou comparées à des données empiriques collectées sur le phénomène. L'issue permet souvent de prédire des trajectoires pour générer de nouvelles expériences, pour améliorer le modèle, voire

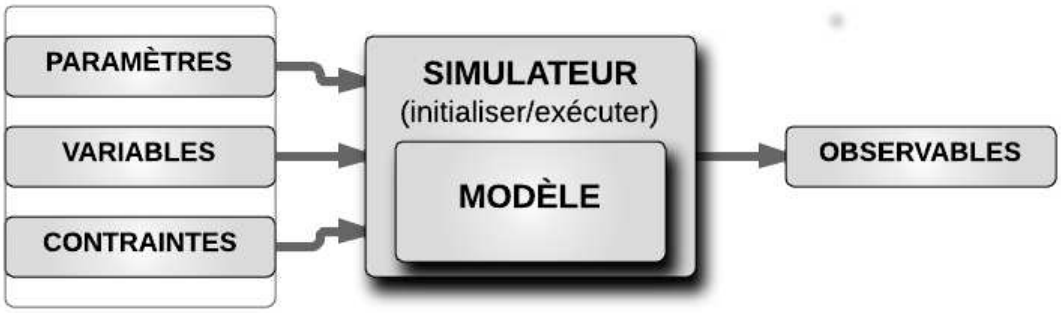

Figure 1.3 – Simulateur complexe

pour le remodeler totalement. En outre, dans les sciences sociales, les données générées hors des observations (logs/bugs) par les méthodes d'expérience intégrée pendant la simulation sont également la source de données pour explorer [CQ13].

1.1.4 Champs d'application de l'exploration des systèmes complexes

Étant donné la piètre performance des humains à gérer un système complexe, il est essentiel de spécifier le but visé. On retrouve habituellement quatre principaux buts : (i) apprendre (acquérir une meilleure compréhension à l'aide d'une rétroaction des actions portées) sur le fonctionnement d'un système complexe ; (ii) résoudre des problèmes impliquant un système complexe ; (iii) prendre des décisions impliquant un système complexe ; (iv) et formuler des politiques (les décisions robustes pouvant être utilisées dans plus d'un contexte) impliquant un système complexe [Spe08].

Au début, les directions d'exploration des modèles tournent autour des méthodes pour la vérification et la validation des modèles complexes. Ces étapes sont importantes pour la compréhension d'un modèle et pour observer tous ses futurs possibles. En effet, d'une manière générale, la validation permet, selon le libre choix de l'expérimentateur, d'affiner le modèle avec le degré de finesse souhaitable pour approcher au mieux la réalité via la calibration. Récemment, d'autres directions de recherche se retrouvent dans l'analyse de sensibilité et l'utilité des simulateurs complexes comme la formation, le contrôle, l'aide à la décision ou la prévision [TDZ08, CDFD12]. Tous ces directions de recherche sont présentées dans les parties suivantes.

Vérification automatique

La vérification cherche à répondre à la question «Avons nous développé un simulateur qui traduit fidèlement le modèle ? ». Les modèles de simulation doivent être vérifiés pour assurer leur fonctionnement vis à vis de la question posée. Cette étape vise à vérifier la conformité entre les spécifications et le modèle implémenté. Alors, l'activité de vérification consiste à s'assurer que le simulateur génère bien le comportement spécifié par le modèle. En d'autres termes, la vérification vise à s'assurer de la bonne transformation des exigences d'un modèle en l'implémentation du modèle (son simulateur) [Pre09]. En effet, cette activité dépend du langage de programmation, du compilateur, de l'architecture matérielle ou des erreurs potentielles logicielles de l'implémentation. Les communautés proposent des méthodes formelles

et non-formelles comme des solutions de ce problème [LW, ULB+15] de deux catégories : l'analyse statistique du code ou le test dynamique via la simulation du modèle.

Analyse de sensibilité

L'analyse de sensibilité cherche à répondre à la question «Quels sont les facteurs qui ont une influence sur les sorties d'un modèle ? ». L'analyse de sensibilité d'un modèle est l'identification et la hiérarchisation des facteurs influents du modèle qui sont la cause des variations les plus importantes des sorties du modèle. La démarche d'une analyse de sensibilité est de faire varier l'ensemble des entrées du modèle et analyser les variations des sorties du modèle.

Validation du modèle

La validation cherche à répondre à la question «Construisons-nous le bon modèle ? ». Les modèles de simulation doivent être validés pour s'assurer de la cohérence entre les sorties de la simulation et les indicateurs du système réel. Le but est donc de lier le modèle à la réalité attendue ou perçue afin d'acquérir une confiance suffisante. Ainsi, il pourra par la suite servir de référence pour évaluer des scénarios d'aide à la décision, par exemple. Le problème de validation est connu comme un test d'un modèle avec les entrées et les sorties déjà connues (figure 1.4). En effet, la validation consiste à approcher des valeurs des paramètres à partir de données expérimentales et des données de terrain.

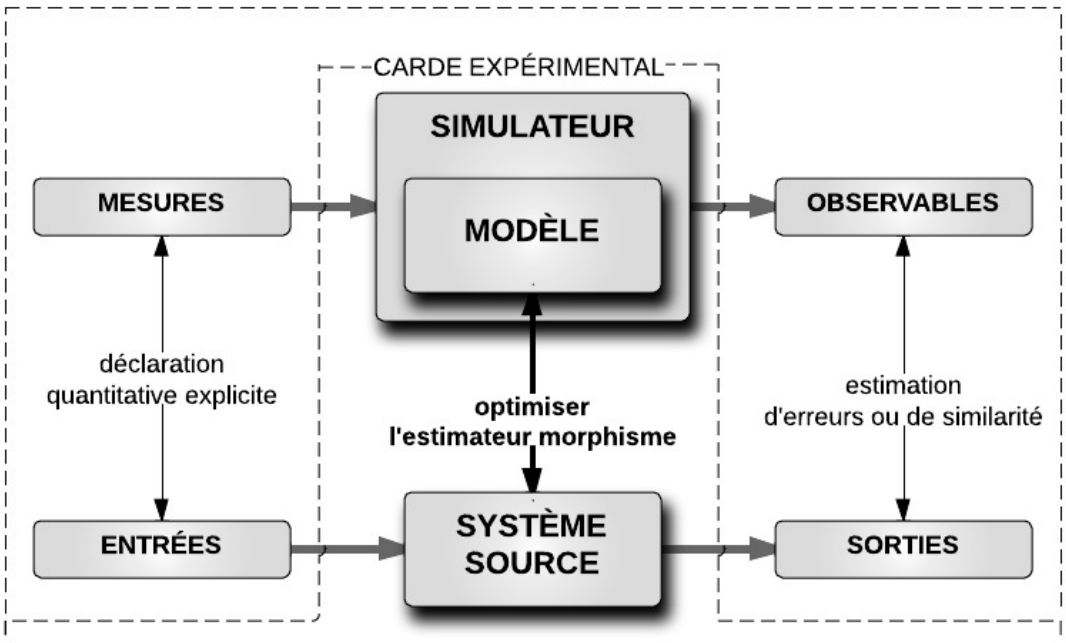

Figure 1.4 – Le processus de validation

Lors de la construction d'un modèle et du simulateur associé, on intègre la structure complexe des entités avec plusieurs niveaux de détail nécessaires dans modèle, ce qui conduit à inclure un certain nombre de paramètres inconnus, non mesurables. C'est pourquoi il est nécessaire d'avoir une phase de calibration du modèle avant d'obtenir des résultats, notamment à visée

prédictive. Cette phase de calibration consiste à ajuster et à déterminer des valeurs pour les paramètres endogènes du modèle qui ne sont pas disponibles dans littérature ou dans les données de terrain.

Validation interne : c'est l'étape de la vérification automatique d'un modèle. Dans ce cas, ces approches reposent sur le test du modèle avec de nombreux paramètres aléatoires pour vérifier que certaines conditions n'ont pas été violées. La vérification du modèle implique la recherche de l'espace d'état d'un système pour déterminer qu'une propriété du système est satisfaite. Les conditions se basent sur les hypothèses de modélisation ou sur le phénomène modélisé. Cependant, l'espace des paramètres est gigantesque pour faire une exploration exhaustive. Alors, ces conditions établissent un domaine de validité du modèle assurant des résultats corrects du modèle, dans une gamme. Ces conditions peuvent alors être considérées comme un objectif d'évaluation à atteindre, un domaine de valeurs de paramètres dans lesquels les conditions internes sont vérifiées. Par exemple : dans le travail de [DPL+] dans le domaine des systèmes embarqués, on compte sur l'utilisation de modèles de comportement, qui sont des modèles décrivant une abstraction du système, en utilisant des variables d'état, et les opérations qui peuvent être exécutées. Ces dernières représentent des fonctions de transition décrites en utilisant des substitutions généralisées. On simule après pour générer des tests de ces modèles. Les séquences obtenues représentent des cas de tests abstraits qui doivent être concrétisés pour être exécutés sur le système en cours de test. Une des techniques est basée sur les critères de sélection dynamiques en utilisant des scénarios définis par l'utilisateur.

Validation externe : Elle confronte les résultats de la simulation avec des données réelles ou compare les résultats simulés avec les résultats théoriques mathématiques ou avec les résultats de modèles divers à partir de différents méta-modèles [FIM+13]. Dans le cas des équations différentielles, grâce à la possibilité de résolution analytique du système d'équations, on peut utiliser les outils mathématiques pour étudier les comportements du système en fonction des valeurs de paramètres. Dans le cas des modèles linéaires, cette tâche est réalisée par l'intermédiaire de l'algèbre linéaire. Néanmoins, les approches mathématiques ne sont, la plupart du temps, pas adaptées dans le contexte de la modélisation à base d'agents. Il faut tester individuellement chaque configuration des variables de déclaration quantitative explicite du modèle comportemental en développant des stratégies spécifiques d'exploration de l'espace des paramètres. Ainsi, le processus de validation externe est un processus par tâtonnement (essais et erreurs) qui cherche les valeurs des paramètres qui ont la probabilité la plus grande ou la fonction de vraisemblance maximale d'être précis avec une tolérance d'erreur acceptable.

Aide à la décision

«L'aide à la décision est l'activité de celle qui, prenant appui sur des modèles (...) aide à obtenir des éléments de réponse aux questions que se pose un intervenant dans un processus de décision » [Roy96]. Le processus de la prise de décision a pour but de trouver une solution appropriée répondant précisément à des problèmes complexes souvent de grandes dimensions. Ce processus se réalise via un système informatique d'aide à la décision (SIAD) pour faciliter la prise de décision stratégique ou opérationnelle adaptée à un contexte et un environnement imprécis ou incertain. Le SIAD est implémenté par des techniques issues de divers domaines : (1) la structuration de l'information et la représentation des connaissances (ontologies, base de connaissances, systèmes experts, etc.), (2) des techniques spécifiques de calcul adaptées au traitement et à la compréhension des données (apprentissage automatique, recherche opérationnelle, intelligence artificielle) et (3) des outils informatiques spécifiquement développés (ingénierie logicielle, interaction homme-machine, télécommunication) [BB03].

Un SIAD permet d'exécuter et d'évaluer des pistes d'amélioration en réponse à des problématiques complexes. On peut distinguer deux grandes catégories : d'une part, on vient de parler de la présentation des préférences du décideur et de la traduction des effets prévisibles permettant un choix rationnel compte tenu des préférences des décideurs. Elles consistent à la valorisation des effets prévisibles, leur agrégation et le choix final entre les différentes actions au vu de leurs conséquences valorisées. D'autre part, on se concentre sur ceux qui permettent d'identifier les alternatives possibles et leurs effets prévisibles sur les objectifs recherchés. L'aide à la décision consiste ici à enrichir l'information disponible et l'état des connaissances concernant le système pour adapter efficacement le comportement du système aux actions envisagées [BB03].

En somme, les simulateurs de système complexe sont une source de prise de décision. Il s'agit :

- premièrement, le modèle qui emporte essentiellement la représentation des préférences du décideur (des ressources disponibles, des contraintes existantes, des intérêts). Il existe de nombreuses formes de représentation : linguistique, mathématique, graphique (une carte routière), analogique (une maquette). Pendant que les formes de représentation linguistique et mathématique sont très fréquentes, l'ajout des cartes graphiques et analogiques permet de compléter les travaux des décideurs dans les applications multi-échelles de représentation comme la planification urbaine et l'aménagement du territoire, spécifiquement avec l'intégration de systèmes d'informations géographiques (SIG).
- deuxièmement, les paramètres d'entrée d'un simulateur qui permettent d'identifier les alternatives possibles compte tenu des objectifs recherchés et permettent un choix rationnel à partir des sorties de simulation. Dans ce cas, l'espace de recherche ou l'espace de décision est défini à partir des domaines de variations des paramètres ou variables caractérisant les solutions du problème.
- troisièmement, les observables du simulateur sont utilisés pour traduire les objectifs recherchés en variables calculables et évaluables. Ils sont les fonctions mathématiques permettant d'évaluer quantitativement ou qualitativement les performances des solutions potentielles. Dans ce cas, l'espace des objectifs est défini à partir des domaines de variations de critères de décision, indicateurs ou attributs.

Les deux derniers points se réalisent qualitativement via des variables catégorisées (par exemple "mauvais", "bien" ou "excellent") ou quantitativement via une métrique appropriée (par exemple les décibels pour le bruit, le taux de propagation des phéromones). Un SIAD qui utilise le modèle informatique pour la prise de décision est systématiquement résumé dans [PS07] ((figure 1.5)).

1.2 Simulateurs complexes

Un modèle de système complexe va être formé par le modélisateur lors de la modélisation. Il est possible de modéliser un système complexe par différents outils conceptuels qui s'appellent "*méta-modèles*" pour faciliter la construction d'un modèle en rendant la modélisation manipulable, structurée, réutilisable. Alors, le modèle repose sur un ensemble d'instructions constituées, de structures de données et d'algorithmes affectés au calcul de l'état de la simulation. Pour être facile à explorer un modèle a besoin d'outils capables d'interpréter ces instructions, d'exécuter des algorithmes affectés pour perturber un scénario de simulation (le scénario comprend les paramètres indiquant l'état initial de la simulation et les données

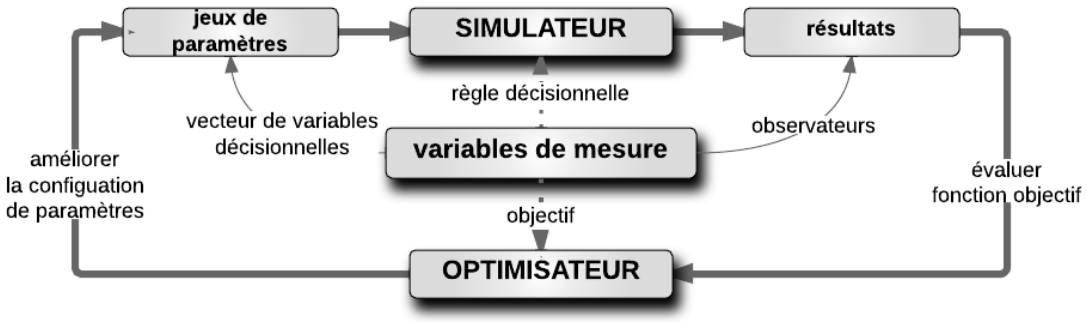

Figure 1.5 – Processus de décision

perturbées au cours de la modélisation) [TDZ08]. C'est le rôle du simulateur complexe dont on fait l'étude dans cette section.

1.2.1 Définition de simulateur complexe

Selon [TDZ08] cité dans l'étude de [Ngu13] : le simulateur est un «programme (ou plate-forme) informatique capable d'interpréter des modèles dynamiques, et utilisé(e) pour produire les perturbations désirées sur ces modèles »

Pour la représentation et la modélisation des systèmes complexes, il repose sur un ensemble d'outils conceptuels (méta-modèles) et/ou logiciels (des simulateurs) [Mar06]. Le méta-modèle garde le fait que le simulateur respecte le modèle dynamique. Le méta-modèle est conçu indépendamment de l'implémentation d'un simulateur. Ça veut dire qu'il y a plusieurs simulateurs implémentés pour un même modèle par suite du méta-modèle [Ngu13].

1.2.2 Caractéristiques des simulateurs complexes

Les principales particularités des simulateurs complexes résident dans leur stochasticité, leurs dynamiques internes complexes, leur temps d'exécution souvent long (dans la sous-section 1.1.2) mais surtout leur lien direct avec la problématique et la question scientifique posée.

Les simulateurs complexes sont qualifiés par leur :

- **forte non-linéarité :** les comportements et les réponses des modèles de systèmes dynamiques ne sont pas déterministes et sont influencés par la présence de relations non linéaires et de boucles de rétroaction.
- **nombre d'états atteignables indéfini voire infini :** la dynamique interne complexe de ces modèles, jumelée aux paradigmes et outils de simulation induit une stochasticité dans les résultats. Par exemple dans la modélisation-simulation multi-agents, l'utilisation de l'aléatoire est un artefact permettant réorganiser l'ordonnancement des agents et ainsi simuler une exécution parallèle de ces derniers. En cas expérimental, pour obtenir le comportement appréciable, il faut exécuter un très grand nombre de fois le même code.
- **grand nombre de paramètres à étudier :** ce nombre important de paramètres jumelé à des temps de simulation et à la dimension du phénomène complexe étudié rend coûteux voire impossible une exploration exhaustive de l'espace des paramètres.

Cela signifie que les méthodes déterministes, qui garantissent un optimum global, ne parviennent pas à fournir la solution en un temps de calcul raisonnable.

— **une sensibilité de la réponse vis à vis des entrées :** une importante variation des sorties de simulation est constatée face à des changements mineurs de la valeur des paramètres. Cela signifie qu'un grand nombre d'optimum locaux existe si bien que les méthodes de recherche par itération successive sur le voisinage proche peuvent s'avérer inefficace.

— **la connaissance sur les données expérimentales disponibles est souvent rare :** ce manque de données pourrait être un obstacle dans la calibration du modèle quand l'on veut comparer les résultats simulés avec les résultats collectés en réalité.

Comme indiqué ci–dessus, un simulateur complexe comporte plusieurs éléments : un modèle de simulation ; l'entrée qui comporte les variables mesurées et les paramètres d'entrée ; et la sortie qui contient des variables observables. Cette section analyse le rôle de ces types de variables.

1.2.3 L'espace de décision des simulateurs complexes

Les paramètres d'un modèle et donc d'un simulateur (ou variables de commande) traduisent, sous la forme de variables qualitatives et quantitatives, des hypothèses étudiées par l'utilisateur, des données de calibration assurant le lien entre la simulation et le système réel, et des éléments liés au fonctionnement interne du modèle comme la graine du générateur de nombre aléatoire [Sto11].

On peut distinguer les paramètres qui sont manipulés par les utilisateurs pour expérimenter des scénarios, des paramètres fixes au système étudié, pour le fonctionnement et le calibrage d'un modèle. Mais la communauté préférera discerner les paramètres *endogènes* des paramètres *exogènes*. Les paramètres endogènes sont *"des variables explicables et expliquées par le modèle"* tandis que les paramètres exogènes sont *"des variables tirées de l'observation ou d'hypothèses"* [Gou84]. Plus simplement, les paramètres manipulables sont endogènes. Les données de terrain, par exemple, un SIG, des données de calibration sont des paramètres exogènes.

Un facteur est vu pour l'utilisateur comme une variable ayant une influence sur la simulation et ses résultats. Il est qualifié par sa nature qualitative ou quantitative intrinsèque à un format de données (booléen, réel, entier, chaîne de caractères) et un domaine caractérisant l'ensemble des valeurs possibles. Ainsi explorer exhaustivement un modèle revient à réaliser des simulations en testant toutes les valeurs et combinaisons possibles de facteurs. Ceci est coûteux en temps de calcul et souvent impossible, d'où la nécessité d'utiliser des algorithmes permettant de trouver rapidement les valeurs intéressantes de ces facteurs.

Le contexte de l'étude et le point de vue sur la question scientifique posée sont déterminants dans le choix des paramètres endogènes et des paramètres exogènes dans la mesure où ces derniers vont contraindre l'usage du modèle et du simulateur. Ainsi, des paramètres endogènes d'un simulateur peuvent devenir des paramètres exogènes dans un autre contexte. Prenons l'exemple d'un modèle de simulation des sols en vue d'optimiser les productions agricoles. Dans ce cadre, on considère que la répartition de la lumière, la quantité d'eau dans le sol, les éléments nutritifs, la température et le vent comme des paramètres endogènes parce que ils sont manipulables et contrôlent le comportement que l'on veut observer (i.e productions agricoles). Mais dans un contexte de gestion des maladies, ces mêmes paramètres deviennent exogènes quand ils sont les données permettant de définir les scénarios du modèle. Les paramètres endogènes peuvent être par exemple le taux d'hormones, la concurrence trophique

entre les organes, etc.

On remarque ici que l'un des rôles du modélisateur est de trouver un juste milieux entre la multiplication des paramètres endogènes afin d'augmenter l'explication et la réduction de ces même paramètres afin de faciliter l'utilisation et minimiser le temps consacrer à l'exploration.

1.2.4 Paramètre endogène vs paramètre exogène

En statistique et économétrie [Pea09], on distingue deux sortes de paramètres. Un paramètre peut être considéré comme endogène ou exogène par rapport à une spécification d'un modèle représentant les relations causales produisant les résultats de sortie. Un paramètre est dit endogène si sa valeur est déterminée ou influencée par un ou plusieurs des paramètres indépendants. Un facteur purement endogène est un facteur qui est entièrement déterminés par l'état des autres facteurs dans le système. Si un paramètre est purement endogène, on peut remplacer ce paramètre par la composition d'autres. Dans les systèmes réels, cependant, il peut y avoir une gamme d'endogénéité. Certains facteurs sont influencés par des facteurs causals dans le système, mais aussi par des facteurs non inclus dans le modèle. Ainsi, un facteur donné peut être partiellement endogène et exogène qui n'est pas entièrement déterminé par les valeurs des autres facteurs du modèle. À l'inverse, un facteur est exogène quand sa valeur est totalement indépendante de l'état des autres facteurs dans le système. C'est un facteur dont la valeur est déterminée par des facteurs en dehors du système étudié. Donc, la catégorie des facteurs exogènes est en contraste avec ceux des facteurs purement endogènes et partiellement endogènes.

On considère un système agricole de causalité simple comme l'exemple. Le résultat que nous sommes intéressés à expliquer est la production végétale. De nombreux facteurs influencent la production végétale : la compétence agricole, des travailleurs, la variété des semences, le climat, la météo, la qualité et le type du sol, l'irrigation, les nuisibles, la température, les pesticides et les engrais, les animaux, et la disponibilité de traction. Si nous modifions les niveaux de ces paramètres, le niveau de récolte varie. Nous pouvons également remarquer, cependant, qu'il y a des relations causales entre certains mais pas tous ces facteurs. Par exemple, le niveau d'invasion de nuisible est influencé par les précipitations et engrais (positivement) et les pesticides, les travailleurs et la compétence (négativement). Donc l'invasion de nuisible est partiellement endogène dans le système et partiellement exogène. Il est également influencé par des facteurs qui sont externes à ce système (la température moyenne, la présence de vecteurs parasites, le déclin des prédateurs, etc.). Inversement, par exemple, le facteur de précipitations est exogène au système. Il existe des facteurs qui déterminent le niveau des précipitations dans un modèle de météo si le facteur de précipitations devient endogène, mais ces facteurs ne font pas partie du modèle de causalité que nous utilisons pour expliquer le niveau de la production végétale.

Un paramètre peut être endogène en intégrant des facteurs supplémentaires et les relations causales dans le modèle causal. Il y a des interprétations causales et statistiques de l'exogénéité. L'interprétation causale est primaire, et définit exogénéité en matière de l'indépendance par rapport aux autres paramètres inclus dans le modèle. L'état du paramètre est relatif à la spécification d'un modèle et notamment les relations entre les paramètres indépendants.

1.2.5 L'espace des objectifs des simulateurs complexes

L'exécution d'un modèle au travers d'un simulateur produit des résultats qui répondent aux paramètres. A l'image des paramètres, les résultats, leur nature sont à la discrétion du modélisateur et de la question scientifique posée.

Les résultats (ou les observables, les variables d'objectif) sont premièrement utilisés comme les indicateurs pour la visualisation et la compréhension via le point de vue des explorateurs. Les sorties s'organisent en 2 groupes : les sorties d'aide à la décision - qui permettent la génération de nouvelles connaissances et les sorties fonctionnelles - les sorties pour tester le modèle, le simulateur etc. Premièrement, cette partie s'intéresse à étudier la façon de traduire les observables en des comportements afin d'évaluer automatiquement les configurations de variables de contrôle en appliquant des approches algorithmiques. C'est parce que pour pouvoir explorer, il faut aussi disposer d'une représentation des comportements pour exprimer de quelle façon un changement de valeur d'une ou plusieurs variables de commande influence les comportements du système, celles que l'on cherche à étudier. Deuxièmement, c'est similaire, pour l'expérience sur un système réel, cette mesure de performance est déterminée dès le début de l'expérience. Le comportement identifié par les variables d'objectif ou la mesure de performance peut-être soit des capteurs ou des moyens de collecte de données sur le terrain dans les systèmes réels, soit des données accessibles dans les systèmes simulés. Cependant, il est difficile pour un explorateur de choisir une mesure convenable pour évaluer : par exemple ce n'est pas des résultats en valeur absolue mais des mesures relatives entre un ensemble de facteurs par rapport à un autre.

Le travail de Calvez [CH06] surtout de Stonedahl [Sto11], qui se concentre sur l'exploration du comportement des simulations à base d'agents, a proposé une reconnaissance de différents types de mesures et son utilisation pour l'évaluation d'une simulation multi-agents basée sur plusieurs niveaux de constituants comme : niveau individuel, niveau intermédiaire (groupe de divers agents) (interne-individu) ou niveau agrégé (externe-individu). Cependant, notre travail n'arrive pas au détail de la relation entre eux. Nous distinguons seulement deux types de mesures d'évaluation pour guider le processus d'exploration sur deux objectifs. Premièrement, la validation utilise une mesure d'erreur (i.e méthode des moindres carrés) ou alternativement une mesure de similarité (i.e méthode du maximum de vraisemblance) [CT05]. Pendant que, la deuxième demande une variable numérique qui représente les désirs de trois méthodes de l'aide de la décision abordées dans la partie dessus.

1.3 Cas d'étude

On a montré que le simulateur complexe est un instrument permettant de reproduire une dynamique d'un système complexe. Dans cette partie, nous voulons donner quelques cas d'étude de systèmes complexes pour lesquels on a construit un simulateur complexe pour les étudier. Ces cas d'étude s'inscrivent dans le domaine du développement durable tels que : l'épidémiologie, la gestion de ressources naturelles et l'aménagement de la ville.

1.3.1 Élaboration d'une politique de vaccination

Au milieu du siècle dernier, les politiques de vaccination sont mises en place afin d'éradiquer les maladies infectieuses et leur menaces. Cependant, on remarque aussi qu'une utilisation abusive des produits pharmaceutiques entraîne l'apparition de bactéries résistantes aux an-

tibiotiques, de fréquentes mutations de certains virus. D'autre part les nouveaux rythmes de vie des hommes rendent difficile la mise en place d'une vaccination raisonnée et efficace. On constate alors l'émergence d'anciennes maladies dans les pays du Sud comme la rougeole ou de maladies à l'échelle mondiale comme le Syndrome d'Immunodéficience Acquise (SIDA), les grippes aviaires et porcines (H5N1, H1N1). Encore aujourd'hui, les maladies infectieuses sont la cause de beaucoup de décès annuels à travers le monde. Le niveau actuel de couverture vaccinale anti-rougeoleuse dans les pays du Sud reste encore problème quand l'immigration est mal connue, la sécurité sanitaire n'est pas assurée pour l'ensemble des individus et les vaccins sont encore honéreux. Afin d'anticiper le risque inhérent à une telle situation pour les années à venir, les chercheurs et décideurs ont recours à une modélisation informatique de la maladie.

Des modèles épidémiologiques sont récemment utilisés pour représenter des phénomènes naturels, et après pour caractériser des risques épidémiques, ainsi d'optimiser les politiques de contrôle. Le modèle épidémiologique reproduit la propagation et l'évolution de la maladie qui est l'idée d'origine de [KM32] à partir d'un des objectifs initiaux de l'épidémiologie : « Combien d'individus seront infectées lorsque l'épidémie se termine ? ». L'idée principale de modélisation est que la population est décomposée en quartes catégories libellées : susceptible (S) qui se met en contact avec une personne qui a la maladie et est infecté (I), ou les susceptibles viennent dans une catégorie d'exposé (E) qui contient ceux qui ont la maladie, mais ne sont pas encore infectieux. Après une courte période, les individus guérissent et se déplacent à la catégorie de guéri (R) où ils restent jusqu'à la mort. La population dans son ensemble est supposée constante (N). Ce système est décrit avec trois paramètres : β : le ratio de transmission, θ : le ratio d'exposé et ν : le ratio de guérison. Quand la vaccination est étudiée au moyen d'un nouveau compartiment de la population, dite « vacciné », ou ajoute un paramètre μ appelé le taux de renouvellement pour considérer également la dynamique des populations. A partir de l'idée originale dessus, différent méta-modèles sont appliqués pour la modélisation : équations différentielles ordinaires, équation de type diffusion, modèle Makovien spatial explicite, modèle à base d'agents, etc. Ces deux derniers nous intéressent parce qu'ils sont des simulations interactives spatio-temporelles qui expriment l'évolution et la propagation de la maladie en temps continu à travers différents niveaux spatiaux.

La politique de contrôle dans ce cas est définie implicitement par un vecteur de variables endogènes (w_1, w_2, w_3, w_4) à base du nombre d'individus dans diverses catégories pendant le temps d'exécution d'un modèle stochastique spatial explicite de la maladie rougeole au Vietnam. Pour explorer l'espace de paramètres de quatre variables (w_1, w_2, w_3, w_4) et évaluer les combinaisons possibles, il faut définir et minimiser une fonction objectif liée au nombre d'infectés et nombre de doses de vaccin utilisées pendant quelques années. La politique de vaccination signifie que si $w_1 \times (I) + w_2 \times (E) \geq w_3$ alors vacciner $w_4\%$ de (S). La fonction objectif dans ce cas est $\alpha \times (M) + (1-\alpha) \times (V)$. (M) est le nombre de personnes qui ont été malades. (V) est le nombre de personnes qui ont été vaccinées et donc de vaccins utilisés. α est le coefficient de pondération entre le coût de vaccination et le nombre d'individus épargnés.

1.3.2 Gestion et gouvernance des ressources en eau

Le projet MAELIA est un projet financé par la Fondation STAE « Sciences et Technologies de l'Aéronautique et de l'Espace »de Toulouse (France) [1]. Dans un contexte de changement climatique, MAELIA [GSBT+13] vise à développer une plate-forme numérique de simulation pour étudier les effets socio-environnementaux des différentes politiques concernant la gestion

1. MAELIA est une collaboration entre AGIR/INRA, GET/OMP (Univ. Toulouse 3), IRIT (Univ. Toulouse 1) et de la MSHS Toulouse (Univ. Toulouse 2), website : http ://maelia-platform.inra.fr

et la gouvernance des ressources en eau et en lien avec l'eau, sur le territoire Toulousain. MAELIA combine :

- des modèles spatio-temporels de l'écologie couplés à un Système d'information géographique (ex. précipitations, changements de température, débit des cours d'eau et croissance des plantes)
- les processus de prise de décision de l'homme notamment dans le milieu agricole ;
- la dynamique socio-économiques (démographie, utilisation des terres et couverture des changements terrestres)
- les autres sous-modèles constituant le système (par exemple, le système de prévision de la météo, les contraintes normatives des comportements).

L'ensemble de ces modèles sont rassemblés dans un modèle à base d'agent. On utilise ce modèle pour étudier les usages et les exploitations concurrentes des ressources en couplage d'un nombre croissant de dynamiques stylisées comme : l'hydrologie à l'échelle du bassin ; l'occupation et l'usage des sols et leurs incidences sur les ressources ; le comportement et les activités humaines liées aux usages et aux exploitations des ressources ou à leur gestion ; les effets du changement climatique notamment sur la ressource en eau ; etc.

Pour étudier, les utilisateurs veulent calibrer les paramètres du modèle. En effet, de nombreux paramètres du modèle influencent les résultats du modèle, avec un haut niveau d'interactions entre les paramètres. Ce niveau d'interaction demande une analyse de sensibilité pour distinguer les différentes influences et interactions. Ensuite, on calibre leurs paramètres. Dans un exemple proposé dans [LMSB+14], l'objectif souhaité est la reproduction de la valeur de flux d'eau et de la dynamique anthropiques qui sont traduites par différents critères numériques tels que l'utilisation conjointe de L_2norm (une caractéristique de l'eau) avec la matrice de variance-covariance et les indices d'erreurs carrés sur le degré de gravité des situations de crise de gestion de l'eau. La calibration est résolue par un problème d'optimisation mono-objectif (selon un objectif conjoint de multi-critère ou selon l'ordre d'importance) ou multi-objectifs. Cependant, ce modèle prend du temps et des ressources de calcul (chaque simulation prend environ 6 heures de simulation et environ 6 Go de mémoire vive). Il ne sera pas possible de le réaliser par les méthodes exhaustives comme la méthode bayésienne. Il est alors recommandé d'utiliser une méthode d'optimisation globale (pour avoir la convergence prématurée) distribuée (pour diminuer le temps de calcul) sur le système de calcul intensif du réseau européen. D'autre part, il a pensé à l'utilisation de méta-modèles de surface de réponse pour l'approximation de la relation d'entrée/sortie en une fonction implicite.

1.3.3 Aménagement de la ville

MIRO [BBMC+10, FBB+16] est un projet financé, au début de 2002 à 2006, par le ministère des transports via le PREDIT [2].

Le projet MIRO fait l'étude de la mobilité urbaine par la modélisation multi-agents dans une ville virtuelle couplées à des données SIG existantes. Il vise à mieux comprendre les comportements quotidiens de la mobilité urbaine et les déplacements dans la ville. Surtout, par la simulation informatique et l'analyse exploratoire, il permet de définir, d'évaluer et de

2. MIRO ensuite est nommé MIRO 2 qui est financé, de 2009 à 2013, par l'Agence Nationale de la Recherche (l'ANR Ville Durables). Ce projet est labélisé par le pôle Véhicules du Futur et par l'institut des Systèmes Complexes de Paris-Ile de France (www.iscpif.fr). Il regroupe des travaux variés du domaine de systèmes complexes et de la simulation à base d'agent. Le projet rassemble des personnels pluridisciplinaires de géographiques, économiques, statistiques et informatiques de plusieurs institutions et universités : CNRS (géographie cité), IRD (UMMISCO), EDF (R&D), Université de Strasbourg (UMR LIVE), Université de Franche-Comté à Besançon (UMR CRESE et UMR FEMTO-ST), de Bourgogne (THEMA), de Grenoble (PACTE, LTHE), d'Orléans (CEDETE), de Rouen (IDEES), d'Avignon (LIA)

comparer l'impact des différentes politiques de mobilité sur l'accessibilité spatio-temporelle des citadins à la ville.

Un des résultats du projet est le prototype GaMiroD développé au sein de la plate-forme GAMA et appliqué sur les villes de Grenoble et Dijon. Ce prototype permet de décrire le modèle d'un environnement urbain virtuel aussi réaliste que possible. Ce prototype repose sur :

- de multiples informations sur la structure de la ville (ex. bâtiments, routes) rassemblées dans un système d'information géographique (SIG),
- une population synthétique de plusieurs centaines de milliers d'agents à partir de données statistiques socio-démographiques,
- une bibliothèque de programmes d'activités,
- des rapports sur les politiques territoriales encouragées par des objectifs de développement durable.

En validation, GaMiroD permet de mettre en œuvre des protocoles de validation pour déterminer, assurer la confiance au modèle construit et aux résultats de simulation produits. D'autre part, Une plate-forme orientée utilisateur (EPIS) [BCC+11] a été mise en place. Elle facilite l'accès aux ressources de calcul haute performance telle qu'un cluster, une grille, etc. pour exécuter simultanément un grand nombre simulations. Ce choix a été motivé par la nécessité d'explorer les scénarios et aussi de valider le modèle. Ce simulateur se sépare un ensemble des paramètres des acteurs, une carte dynamique et les indicateurs pour les jeux sérieux collaboratifs agents/acteurs. Dans l'avenir, on peut penser à explorer le modèle automatiquement en utilisant les acteurs virtuels pour donner les décisions ou calibrer les modèles avec les données réelles.

Discussion

L'étude des systèmes complexes est un problème transdisciplinaire. Par exemple, dans l'aménagement du territoire, chaque plan de secteur, chaque décision d'affectation du sol implique plusieurs acteurs : économiques, juridiques, institutionnelles, sociologiques, démographiques, etc. Le temps de simulation est très longue quand les conséquences du phénomène sont à long terme. Il contient aussi le caractère géographique quand on considère différent niveaux spatials. Cela rend le modèle très difficile à explorer. Un simulateur de système complexe essaie de reproduire la dynamique de ce genre. C'est pourquoi un simulateur de système complexe est souvent considéré comme une boîte noire quand on n'a pas assez de connaissances pour le décomposer pou le comprendre. Il reste aussi un ensemble de paramètres à explorer pour valider le modèle ou étudier les phénomènes d'intérêt et l'on n'a pour seul choix d'exécuter le simulateur souvent coûteux et stochastique. L'exploration de l'espace de paramètres d'un simulateur complexe devient impossible avec les solutions traditionnelles telles que l'intelligence artificielle centralisée et le système de calcul centralisé. La question posée est comment on peut concevoir une stratégie efficace pour explorer l'espace de paramètres en temps acceptable et trouver les jeux de paramètres adaptés.

Chapitre 2

Méthodes d'exploration des simulateurs complexes

Sommaire

L'analyse que nous faisons des systèmes complexes dans le chapitre 1 montre que leur étude passent le plus souvent par l'utilisation d'un simulateur complexe. Plus exactement, il s'agit d'explorer l'espace des paramètres de ce simulateur complexe afin de lier les phénomènes et les changements observés aux valeurs de paramètres. D'autre part, l'exploration de simulateurs complexes demande quelques conditions assez particulières que l'on a discuté dans la partie 1.2.1.

Ce chapitre présente une revue des travaux existants sur l'exploration des simulateurs complexes. Nous l'organisons comme suit : premièrement, les définitions générales, différents objectifs à l'étude de l'exploration et la limitation de l'étude au cadre de la thèse seront présentées ; deuxièmement, les méthodes d'exploration et leur connexion avec travaux actuels sont présentées ; finalement, différents algorithmes disponibles sont qualifiés en fonction du comportement de recherche sur l'espace de recherche.

2.1 L'exploration de simulateurs complexes

On commence au début de clarifier la notion de l'exploration de simulateur complexe dans le cadre de la thèse. Ensuite, les éléments qui participent au cycle de l'exploration sont présentés.

Au début, on essaie de comprendre la notion de l'exploration de simulateur complexe dans le cadre de la thèse. Il n'existe pas de définition exacte de l'exploration de simulateurs complexes. Dans ce domaine, trois familles complémentaires existent pour étudier les systèmes complexes :

- L'exploration consiste à faire l'étude des interactions entre les paramètres du système étudié.
- L'optimisation est de faire la recherche des configurations optimales de paramètres pour un certain problème. L'optimisation utilise le principe de l'exploration afin d'identifier une combinaison intéressante de paramètres.
- L'approximation répond à un besoin de généralisation des hypothèses en appliquant des règles déduites à des scénarios inexplorés dans l'espace des paramètres. Cette dernière approche étudie la réponse d'un simulateur pour un ensemble de facteurs qualifiés d'importants.

Selon le travail de [Sto11], pour l'exploration des simulateurs complexes à base d'agents, on pourrait parfois utiliser mutuellement les algorithmes dans trois familles. En effet, l'exploration en effectuant une épuration des paramètres avec un plan d'expériences en nombre fixe d'échantillons avant de faire simuler tous ces jeux de paramètres pour garder les facteurs ayant un effet significatif sur la réponse n'est pas suffisant dans certaines cas [Sto11]. Par exemple, ces méthodes classiques supposent souvent que les interactions entre les facteurs ont des effets linéaires ou des effets d'ordre inférieur ce qui n'est pas vrai dans la simulation multi-agents. De plus, avec un simulateur coûteux, un méta-modèle d'approximation de simulation pour prédire la réponse d'un modèle est efficacement utilisé afin d'économiser du temps de calcul. Ensuite, on veut trouver la combinaison de facteurs qui optimise la réponse du modèle. Malheureusement, une telle politique robuste saura trouvée un résultat satisfaisant plutôt qu'optimale par la complexité. [Sto11] a dit que, parfois, l'exploration de simulateur complexe doit compter à la fois l'exploration, l'optimisation et l'approximation dans un même processus d'exploration. Il vaut mieux alors proposer pour nous-même une définition de l'exploration des simulateurs complexes.

Dans ce cas, nous donnons une définition générale du concept d'exploration : l'exploration consiste à rassembler de la connaissance sur un système étudié de manière à extraire un maximum d'informations en effectuant un minimum de simulations. Avec l'association des familles de méthodes de plan d'expérience dans une seule définition d'exploration comme dessus, un plan d'expériences interactif qui est réalisé par l'union des méthodes d'explorations, d'optimisation et d'approximation dans un même cadre expérimental pour inclure les connaissances acquises au cours de l'analyse pour la compréhension du système de sorte que nous tirons le maximum d'informations du système complexe en un minimum de simulations (échantillons). Alors, différentes méthodes sont appliquées parallèlement ou successivement. D'autre part, il est souvent possible de choisir de nouveaux jeux de paramètres à évaluer successivement, en utilisant les informations obtenues à partir des résultats précédents pour générer plusieurs générations d'échantillons. De cette façon, on souhaite que l'exploration et ses méthodes de réalisation puisse faire trois missions en parallèle pour un but désiré d'exploration.

Comme dernière remarque, l'exploration d'un système complexe doit éviter deux erreurs trop souvent commises qui sont soit de ne faire varier qu'un seul paramètre à la fois avec le risque de rater une importante combinaison de facteurs, soit de faire varier trop de paramètres à la fois ce qui rend impossible de distinguer les facteurs les plus influents.

2.2 Concept et enjeux

Les modèles informatiques montrent que la connaissance complète de ses mécanismes ne permet pas de prévoir le comportement d'un simulateur complexe à cause des caractéristiques. L'explorateur du modèle n'a pas les outils mathématiques pour faire une exploration analytique. La compréhension du comportement du modèle nécessite de pratiquer des expérimentations (des simulations), en testant différentes valeurs des paramètres et différentes conditions initiales. De cette façon, l'exploration comprend l'étape de structurer ces expérimentations pour les automatiser partiellement.

L'étude de [CDFD12] considère que l'exploration du modèle procède par une évolution successive adaptée. Au début, les expériences sont assez larges et aléatoires en répétant les éléments les plus saillants de la dynamique, puis ces expériences sont progressivement raffinées dans directions qui sont identifiées comme intéressantes. L'exécution des expériences est coûteux et des fois est répétée s'il n'arrive pas à déterminer de directions sur les paramètres de décision intéressants. D'autre part, il y a nombreux de facteurs : les tests en changeant un seul facteur à la fois sont simples à faire et comprendre, mais on va rater des solutions prometteuses ; les tests en changeant plusieurs facteurs à la fois sont très puissants, mais la production des expériences est difficile à diriger et les résultats sont difficiles à interpréter. Il est important d'optimiser la production de ces expériences par le stratégie d'exploration. Elle se réalise par le biais de plans d'expérience permettant de faciliter la production des expériences, ainsi que la définition des variables agrégées résultats.

2.2.1 Plan d'expériences

De manière générale, les plans d'expériences correspondent à une suite ordonnée d'essais d'une expérimentation pour lesquelles les valeurs des paramètres d'entrée sont différents afin d'obtenir des résultats sur un phénomène avec une bonne économie. Les méthodes de plan d'expériences reposent essentiellement sur la création et l'exploitation de modèles de la fonction objectif (réponses). Réaliser un plan d'expériences est l'art de planifier intelligemment plusieurs simulations, de les ordonner afin d'acquérir rapidement, de nouvelles connaissances sur le modèle et les systèmes étudiés. Selon[KSLC05], un plan d'expériences répond à trois objectifs principaux : (1) produire des connaissances sur le système complexe, (2) améliorer, voire identifier des politiques décisionnelles robustes, (3) évaluer les politiques décisionnelles.

Les problèmes de coût liés à la réalisation de l'expérimentation impliquent qu'il faut effectuer des mesures en nombre fini. Il faut choisir le protocole de sélection d'une partie dans un tout mais il s'avère aussi important d'assurer un échantillonnage en grande dimension. Un protocole de sélection s'appelle plan d'échantillonnage. Le plan d'échantillonnage est souvent conçu et réalisé de manière indépendante du système et des connaissances des experts pour éviter un biais sur certains paramètres et un vide dans une région de l'espace. Il existe plusieurs stratégies possibles pour en faire [Mon] : (1) cribler et hiérarchiser des facteurs en conditions limitées en sélectionnant précisément les points où échantillonner, (2) augmenter la qualité de l'exploration numérique par répartir l'échantillon dans tout le domaine en se fondant sur la direction de convergence des estimateurs, (3) cibler les zones concernées en estimant la probabilité d'événements rares.

La réalisation d'un plan d'expérience passe par l'exécution d'un enchaînement de simulations, qualifiées par des valeurs différentes de facteurs. Ainsi, la taille d'un plan d'expériences, et naturellement le temps nécessaire pour le réaliser, sont intimement liés au nombre de facteurs

à explorer, à la taille du domaine de chacun de ces facteurs, et au nombre de répétitions de chaque simulation. Plusieurs milliers, voire centaines de milliers de simulations sont souvent nécessaires pour mener à bien un plan d'expérience. C'est pourquoi, une automatisation et une parallélisation du processus s'impose afin de réduire les coûts humains et les délais d'attente.

2.2.2 Le cycle d'exploration de simulateur complexe

Cette partie parle des éléments à assurer pour faire l'exploration de simulateur complexe. L'exploration de simulateur complexe se situe dans un contexte précis. Ce contexte est expliqué dans la figure 2.1.

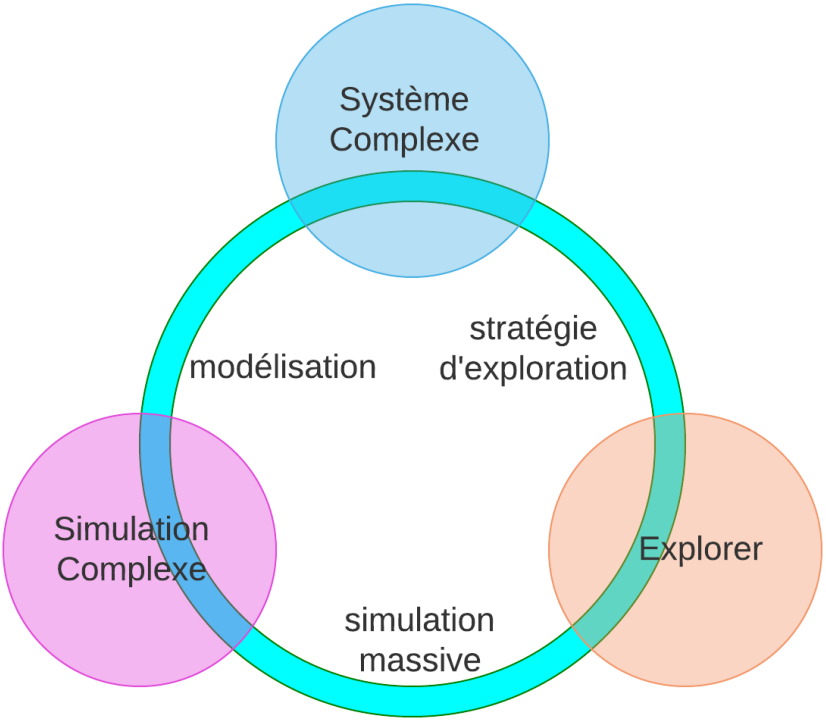

Figure 2.1 – Le cycle d'exploration

— *Modélisation :* On a déjà étudié la notion de simulateur complexe dans le chapitre précédent. Le fonctionnement interne du modèle d'un simulateur est une boîte noire où les seuls matériels pour étudier le modèle sont les entrées et les sorties du simulateur. Il est considéré que le modèle est une spécification de la dynamique du système étudié. Plusieurs versions d'implémentation d'un même modèle qui s'appellent simulateur complexe apparaît comme l'outil informatique permettant de calculer le modèle de système complexe d'un phénomène réel. Récemment, le simulateur complexe de types géographiques est souvent utilisé pour étudier les phénomènes spatials. Les géographes s'appuient sur l'hypothèse que les interactions microgéographiques produisent l'émergence de «dynamiques stylisées»aux échelles macro-géographiques, qui constituent une des caractéristiques universelles de ces systèmes complexes [Pum03]. Cependant, un simulateur est souvent plus compliqué quand on ajoute plusieurs facteurs pour représenter un phénomène réel. Par exemple, le modèle des modalités de gestion durable des étiages à l'échelle des bassins versants du projet MAELIA est spatialisé multi-niveaux, avec de fortes interactions entre processus(ex. irrigation, lâcher de barrage), de forte non linéarité (effet de seuil, durée entre les seuils d'alerte). Son simulateur est

modulaire avec des contraintes de temps de calcul et de ressources (simulation modèle complet pour 10 ans : 5 heures de calcul avec 5 Go de RAM).

— *Stratégies d'exploration :* Une stratégie d'exploration repose sur le choix d'une ou plusieurs méthodes d'exploration de l'espace des paramètres d'un modèle afin d'en faire une étude optimisée. Cette stratégie a pour but de bien sélectionner les jeux de valeurs de paramètres d'entrée. Compte tenu de la non linéarité de la réponse aux paramètres d'un simulateur complexe, surtout lorsque ce dernier aborde une problématique spatialisée, les stratégies d'exploration usuelles ne sont souvent pas satisfaisantes. Le lien étroit qui existe entre le cas d'étude et un simulateur complexe permet toutefois d'envisager la création d'algorithmes d'exploration. Mais ces derniers doivent être définis au cas par cas en toute connaissance du cas d'étude, des enjeux scientifiques et de la nature des paramètres. Des stratégies d'exploration guidée et/ou collaboratives semblent des pistes intéressantes à explorer.

— *Simulations massives :* L'exécution d'une simulation est souvent coûteuse. De plus, l'exploration demande un nombre important de simulations à la fois pour tester l'ensemble les jeux de paramètres définis par un plan d'expérience, mais aussi pour répéter chaque jeu un nombre de fois suffisant afin d'obtenir des résultats statistiquement significatifs. Dans ce contexte, l'utilisation des moyens de calcul haute performance est un atout pour gagner du temps voire une nécessiter lorsque la puissance de calcul des ordinateurs de bureau est insuffisante.

2.3 Outils de l'exploration des simulateurs complexes

Dans cette section, nous allons présenter différentes approches de la littérature dédiées à l'exploration de simulateurs complexes. Etant donné que nous considérons un simulateur complexe comme une boîte noire à explorer, dans cette section on étudie des approches d'exploration, des plus simples aux plus compliquées. Ensuite, on réduit le cadre des approches d'exploration de la thèse en concentrant sur l'exploration inverse en utilisant les algorithmes d'optimisation. Et enfin, on rétrécie le cadre d'appliquer ces approches à la validation et la prise de décision.

2.3.1 Exploration interactive visuelle vs exploration automatique

L'exploration interactive de l'espace des paramètres de la simulation complexe vise à tirer partie de l'expertise des utilisateurs (via une interface graphique) afin d'accélérer le processus d'exploration. Ainsi, l'expérimentateur devient moteur de l'exploration : en fonction des résultats d'une ou plusieurs simulations, il décide, itérativement, d'exécuter une ou plusieurs autres simulations qui vont petits à petits répondre au questionnement scientifique initial.

Il est pratique pour les expérimentateurs d'interpréter différents aspects de leurs modèles en fonction de leurs intuitions quand il permet de regarder différents niveaux d'affichage des résultats et de manipuler les points d'évaluation de l'espace de recherche. On suppose que les modèles étudiés sont déjà conçus mais il est capable d'apporter des modifications dans les scénarios et sur les paramètres associés des modèles par l'intervention d'humains au début ou pendant son exécution.

Pour distinguer ces deux types de l'exploration, on étudie sur les travaux de Nicolas Bredeche [DMBP11] et ceux de Rioux [Rio09].

Le travail de Rioux [Rio09] a proposé une méthode d'analyse intégrée dans l'outil de visualisation interactif appelé "Multichronia". Ce dernier vise à aider ses utilisateurs à comprendre un système en suivant quatre boucles interactives pour interagir avec les données dans le pipeline 2.2 :

— *La boucle d'exploration de l'espace des paramètres* permet à un utilisateur de modifier des paramètres d'entrée des simulations associées à l'exécution de modèles par un simulateur afin de tester des hypothèses. La sortie d'une simulation est un flot de données brute. Les flots de données brutes sont convertis dans un format compatible pour les boucles suivantes.

— *La boucle d'exploration de l'espace des simulations* permet à l'utilisateur de manipuler les données générées par des simulations en utilisant des opérations de sélection et d'ordonnance via une interface graphique. Parce que Multichronia offre la possibilité d'exécuter plusieurs simulations simultanément, il faut fournir la capacité d'inclure dans le pipeline de données ces deux opérations (à aligner et à sélectionner) les flots de données se propageant dans le reste du pipeline. L'alignement sert à synchroniser différents flots de données sur une condition définie. La sélection consiste à choisir quelques flots de données parmi ceux qui sont disponibles afin de les propager plus loin dans le pipeline de données.

— *La boucle d'exploration de l'espace des données* Cette bouche accepte en entrée les flots de données alignés et sélectionnés dans la bouche précédente et exécute des calculs/traitements sur les données provenant de la simulation (e.g. moyenne, fenêtre de données, transformée de Fourier). Il offre en sortie des flots de données modifiées et/ou enrichis de nouvelles données structurées, adaptées pour la visualisation ;

— *la boucle d'exploration de l'espace visuel* rend les données disponibles à l'utilisateur. Elle permet d'afficher des données et de les rendre visuellement intelligibles. Plusieurs options sont configurables, dont l'aspect visuel des données et la possibilité de rendre la totalité des données ou seulement un sous-ensemble. Il laisse les utilisateurs percevoir et traiter les résultats à différents points, de désigner la forme de l'espace de recherche du modèle, de notifier la tendance ou d'étudier les points d'intérêt.

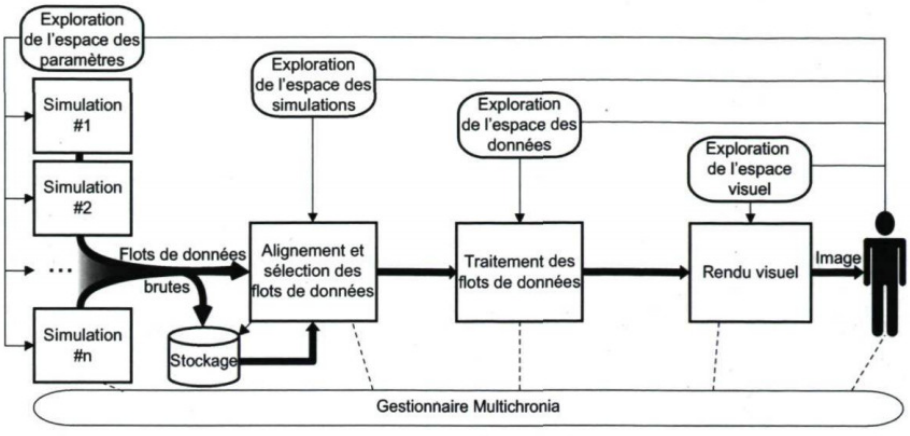

Figure 2.2 – Quatres boucles d'interaction dans l'outil Multichronia [Rio09]

L'une des difficultés majeurs dans ces travaux est de proposer aux utilisateurs une visualisation intelligible permettant de comprendre les espaces multidimensionnels de paramètres et de résultats. Les travaux de Kornhauser et Grignard [Kor09, GDZ13] tentent de proposer des solutions en développant des outils graphiques de visualisation et d'exploration dédiés au domaine de la simulation à base d'agent dans deux plate-formes NetLogo et GAMA.

Fondamentalement, cette approche donne l'avantage de lire et tracer les résultats quand les expertises ont beaucoup de connaissances sur le modèle. Il est efficace pour guider la prise de

décision en temps réel. Cependant, laisser la manipulation du processus d'exploration pour l'homme augmente les décisions biaisées quand l'homme fait des hypothèses erronées sur le comportement du modèle. Ces décisions biaisées négligeront des régions prometteuses, tomberont dans des régions localement intéressantes et ne permettra pas une recherche profonde d'une région. Cette méthode n'est pas efficace en appliquant sur les modèles dont l'espace de paramètre est rugueuse. En outre, le temps d'exploration d'un simulateur complexe est très long et peut-être fastidieux pour les humains. Tous ces inconvénients rendent cette méthode comme une méthode d'exploration partiale plutôt qu'une approche avec une indépendance complète. En fait, on cherche à intégrer l'exploration humaine interactive avec des méthodes algorithmiques automatisées pour avoir les meilleurs résultats.

Inversement, l'exploration automatique, techniquement, est l'approche d'utiliser les algorithmes d'exploration pour parcourir automatiquement et inlassablement l'espace de recherche. Le travail de Nicolas Bredeche et ses collègues [DMBP11] discute de l'utilisation des robotiques évolutionnistes dans le processus de conception de robots autonomes d'exploration (un robot pour objectif d'éviter des obstacles et d'aller se recharger automatiquement dans un environnement simple). On a classifié les travaux de robotique évolutionniste en quatre catégories (figure 2.3) : (1) ajustement de paramètres (en simulation ou sur robot réel) ; (2) conception aidée par évolution ; (3) adaptation en ligne évolutionniste ; (4) synthèse évolutionniste. Toutes ces catégories ne possèdent pas la même maturité. Le réglage des paramètres de (1) consiste en utilisant un algorithme d'optimisation. Conception assistée évolutive (2) est une tendance plus récente qui diffère des réglages des paramètres dans l'utilisation des résultats. Les experts analysent des paramètres optimisés identifiés à la fin du processus de réglage des paramètres pour obtenir une meilleure compréhension du problème. Les experts seront alors en mesure de proposer une nouvelle solution dans une étape ultérieure.

L'adaptation en ligne évolutionniste consiste à utiliser algorithme d'optimisation non seulement lors de l'étape de conception, mais aussi lors de la vie de robot, afin de lui permettre de s'adapter en temps réel et continuellement pour changer radicalement les situations. Enfin, la synthèse de l'évolution est de concevoir la construction à partir de zéro d'une communauté d'agents autonomes en s'inspirant des mécanismes d'évolution réelles. les robots peuvent adapter pleinement les dynamiques qui émergent des interactions entre leur morphologie et leur contrôleur afin de résoudre de manière optimale une tâche.

Outre le domaine de conception robotique, en générale, l'exploration automatique se décline en différentes catégories dépendant de la perception d'un modèle comme une boîte noir/-blanche ou au processus d'exploration direct/inverse. On les aborde dans les deux parties suivantes. C'est la direction de recherche de cette thèse.

2.3.2 Boîte blanche vs boîte noire

Si on modélise un phénomène en cherchant une façon de le décomposer en sous-modèles pour clarifier ce qui se passe dans le modèle, on considère alors le simulateur comme une «boîte blanche ». Une telle méthode s'appelle le calibrage «boîte blanche »[FKP05]. Le principe est d'utiliser la connaissance de la structure du modèle pour améliorer le processus de calibration. Cette approche réalise une décomposition du modèle en unités plus petites de comportements individuels en utilisant différentes méthodes : décomposition générale du modèle, décomposition d'unitaires fonctionnelles, décomposition de comportement des agents, décomposition de phases et de situations temporelles, etc. L'objectif de la décomposition est de réduire la complexité de la configuration de l'espace de paramètres et de la dépendance des paramètres. Après avoir calibré des sous-modèles indépendamment, on les fusionne pour former le mo-

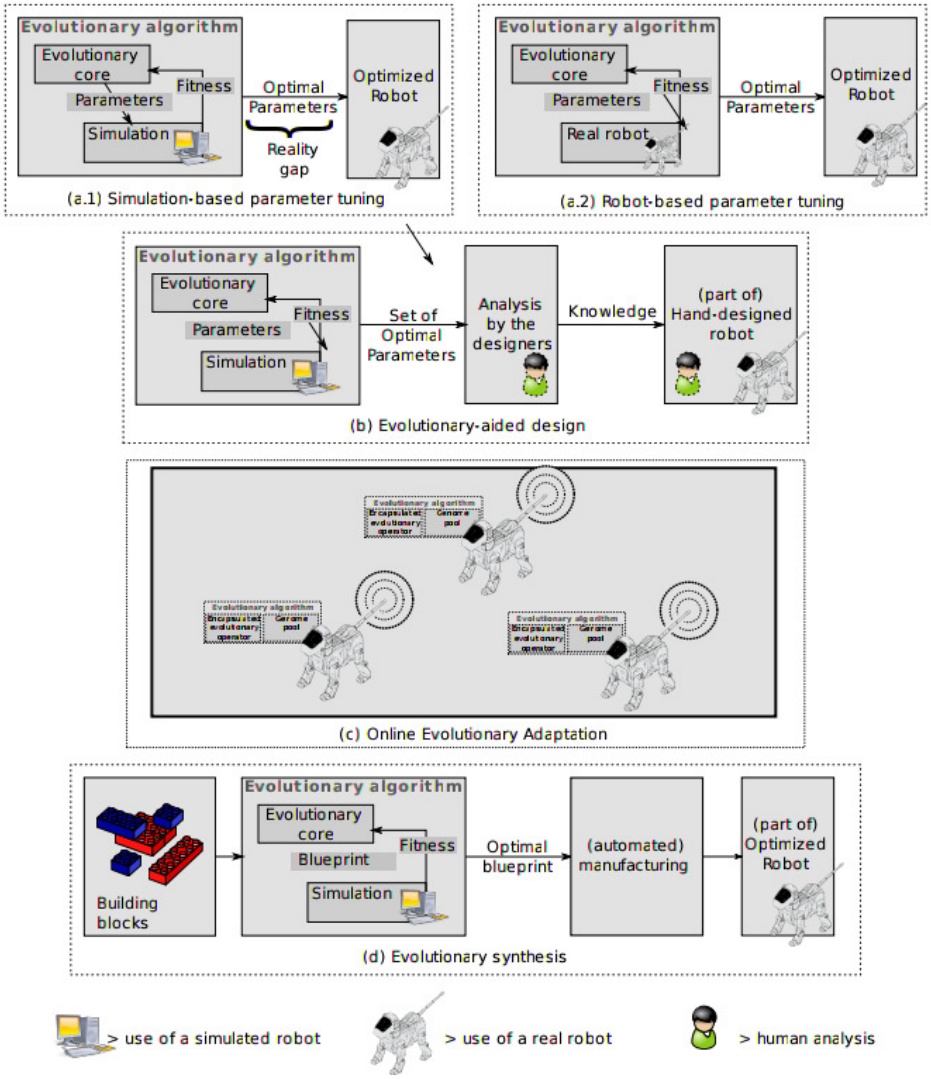

Figure 2.3 – Différents contextes d'utilisation des algorithmes évolutionnistes dans le cadre d'un processus de conception de robots autonomes. [DMBP11]

dèle. Cette approche tire meilleur avantage de la connaissance des fonctionnements implicites. Cependant, les opérations de division et de fusionnement sont difficiles. Pendant que la division nécessite l'ajout de connaissances sur le modèle, ce qui peut ne pas être disponible, spécifiquement avec les modèles multi-agents, le fusionnement n'est pas automatique. Alors qu'avec «boîte blanche », les approches offrent des avantages clairs, dans nombreux cas, la décomposition du modèle peut être extrêmement difficile, voire impossible.

Dans cette thèse, on n'aborde que la considération d'un simulateur comme une «boîte noire» quand les simulateurs sont impossibles de décomposer. Dans ce cas, il faut explorer un modèle en exécutant le simulateur correspondant entier. Cependant, le paradigme d'exploration se subdivise en exploration directe ou exploration inverse [Sto11].

2.3.3 Exploration directe vs exploration inverse

Lorsque le simulateur est considéré en tant que «boîte noire», on peut simplement utiliser le paradigme traditionnel «exploration direct » qui requiert les simulations de certains mo-

dèles et la prédiction du comportement du système en fonction de ses résultats. L'objectif est de décrire expérimentalement comment les paramètres du modèle se traduisent en effet observables. On cherche ainsi à reproduire un comportement en réglant des paramètres du simulateur : (i) initialiser une configuration des paramètres théoriques ; (ii) lancer les simulateurs de modèles plusieurs fois pour chaque configuration ; (iii) observer le comportement du modèle via les résultats et la dynamique mise en avant. Cependant, ce paradigme est trop général et difficile à utiliser. Il est difficile d'identifier un comportement précis ou l'ensemble de comportements via une simple analyse des résultats : d'une part, on ne peut pas expérimenter toutes les combinaisons possibles de paramètres pour avoir une vue globale sur les résultats ; d'autre part, les résultats expriment un point de vue sur le modèle et ne sont en aucun cas représentatif de l'ensemble du modèle. L'exploration directe est finalement devenu seulement une étape initiale de la modélisation du phénomène qui construit le modèle à partir des premières hypothèses [Tar05].

«L'exploration inverse» cherche à identifier les configurations suffisantes pour un comportement proposé. Il s'agit donc d'identifier un comportement par des mesures de terrain ou à dire d'expert et d'utiliser ensuite la simulation et les méthodes algorithmiques d'optimisation pour retrouver les configurations permettant d'amener au comportement identifié.

La figure 2.4 illustre les différences entre l'exploration directe et l'exploration inverse. L'exploration inverse est abordée au travers de cette thèse quand on considère l'exploration comme un problème d'optimisation. Cette approche utilise des méthodes mathématiques qui permettent d'appréhender la calibration des modèles, la recherche d'une décision optimale selon un critère prédit par le modèle. En fait, toute classe de l'exploration peut conduire à un problème d'optimisation, pourvu que l'on y introduise des paramètres ou variables à optimiser.

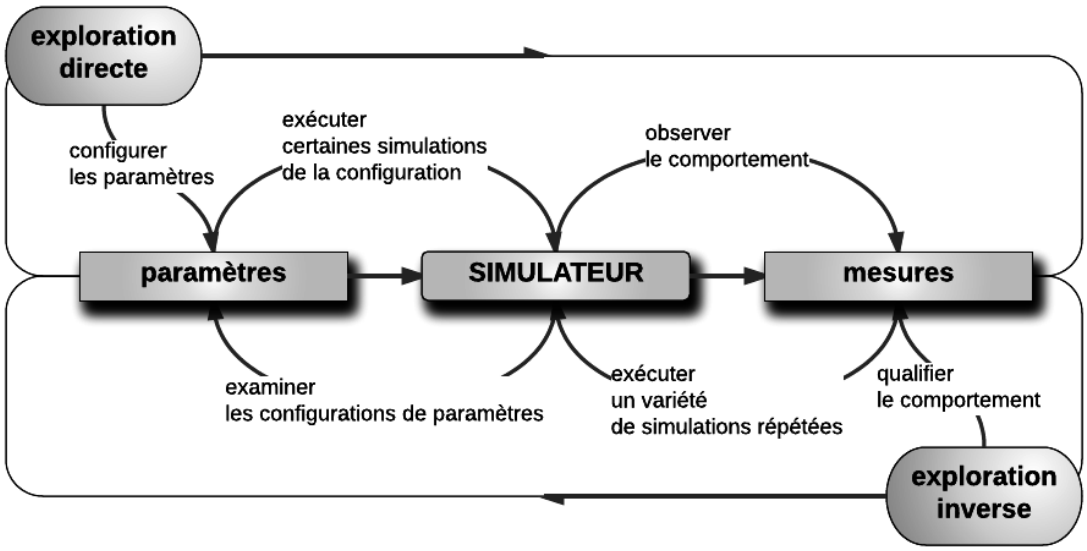

Figure 2.4 – Le paradigme d'exploration directe et celui d'exploration inverse inspiré de [Sto11]

2.3.4 L'exploration inverse dans l'élaboration de politiques et de décisions robustes

Comme nous l'avons vu dans la section 1.2.1, l'exploration exhaustive des paramètres devient impossible à mesure que le nombre de paramètres augmente. Il faut alors se tourner vers des techniques spécifiques de réglage des paramètres qui utilisent une exploration empirique, expérimentale et précisément un champ de méthodes mathématiques d'optimisation pour faire la recherche sur l'espace des paramètres. De cette façon, il faudra se concentrer autour :

— de la paramétrisation optimale du modèle boîte noire, code de calcul long ou coûteux,
— de nombreux paramètres continus et discrets et des processus stochastiques.

Ces techniques se concentrent sur deux buts : (1) trouver les valeurs des paramètres qui reproduisent au mieux les observations du système modélisé ou (2) les valeurs de variables décisionnelles qui répondent à certains critères prédits.

La notion de l'aide à la décision a été définie dans la section 1.1.4. Dans cette partie, on discute la méthodologie pour l'aide à la décision en considérant l'exploration de l'espace de paramètres comme un problème d'optimisation.

L'optimisation est une branche des mathématiques, cherchant à analyser et à résoudre analytiquement ou numériquement les problèmes qui consistent à déterminer le meilleur élément d'un ensemble selon un critère quantitatif donné et des contraintes [BB03]. Le calibrage nécessite dans un premier temps la définition d'un certain nombre d'objectifs résumant la qualité d'un jeu de paramètres par rapport à des résultats attendus du modèle. Ces fonctions objectives sont calculées à partir de l'exécution d'un ensemble de réplications d'une simulation utilisant un même jeu de paramètres (le modèle étant stochastique, chaque réplication correspond à un flux de nombres aléatoires indépendant de ceux des autres réplications). Ensuite, le processus de calibrage sélectionne les meilleurs jeux de paramètres d'après les valeurs atteintes pour ces objectifs. Pour calibrer des modèles multi-agents l'évaluation répétée des objectifs constitue donc une charge de calcul très importante qui ne peut être menée à bien que sur des environnements de calcul haute performance.

L'optimisation devient naturellement une des stratégies pour prendre la décision (figure 1.5). Le processus est de (1) initialiser un ensemble de jeu de paramètres, (2) lancer le simulateur pour calculer les résultats de ces jeux de paramètres, (3) tirer les connaissances de ces premiers résultats, (4) générer les jeux de paramètres adaptatifs aux connaissances obtenues, (5) revenir au (1). En ce sens, on va chercher les configurations optimales pour optimiser la configuration des variables de déclaration quantitative explicite du modèle comportemental.

La calibration dans l'activité d'élaboration de politiques et de décisions robustes est souvent d'utiliser les algorithmes d'optimisation pour explorer l'espace de paramètres. Mais la différence avec un problème d'optimisation classique est que l'on n'essaie pas de trouver une configuration optimale mais une configuration satisfaisante. Il s'agit aussi d'établir une cartographie des réponses pendant le processus d'exploration. Tous ces éléments faciliteront la compréhension de la dynamique face au changement de paramètres

2.4 Algorithmes fondamentaux d'exploration

Récemment, différents méthodes d'exploration sont étudiés [FIM+13]. Dans la plupart des cas, les approches utilisées sont ad'hoc. Ainsi il nous est difficile d'établir une liste exhaustive des approches. Nous nous concentrons sur les méthodes d'exploration qui sont utilisées pour

la validité et l'utilité des modèles dynamiques et plus particulièrement des modèles multi-agents représentant des systèmes complexes socio-écologiques ou socio-environnementaux. Il y a deux grandes catégories : les algorithmes d'exploration systématique, les algorithmes d'exploration stochastique et heuristique.

2.4.1 Algorithme d'exploration systématique

Cette classe d'algorithmes réalise une stratégie de discrétisation de l'espace des paramètres pour fragmenter les zones de recherche. La discrétisation peut être itérative ou non. Dans le cadre d'un processus de discrétisation non-itérative, l'ensemble des jeux de paramètres nécessaires à la simulation est généré en même temps, avant de lancer l'expérimentation. Dans le cadre d'un processus de discrétisation itératif, quelques jeux de paramètres sont déterminés, simulés. En fonction des résultats des simulations, de nouveaux jeux de paramètres sont identifiés simulés et ainsi de suite. La figure 2.5 montre cette classification.

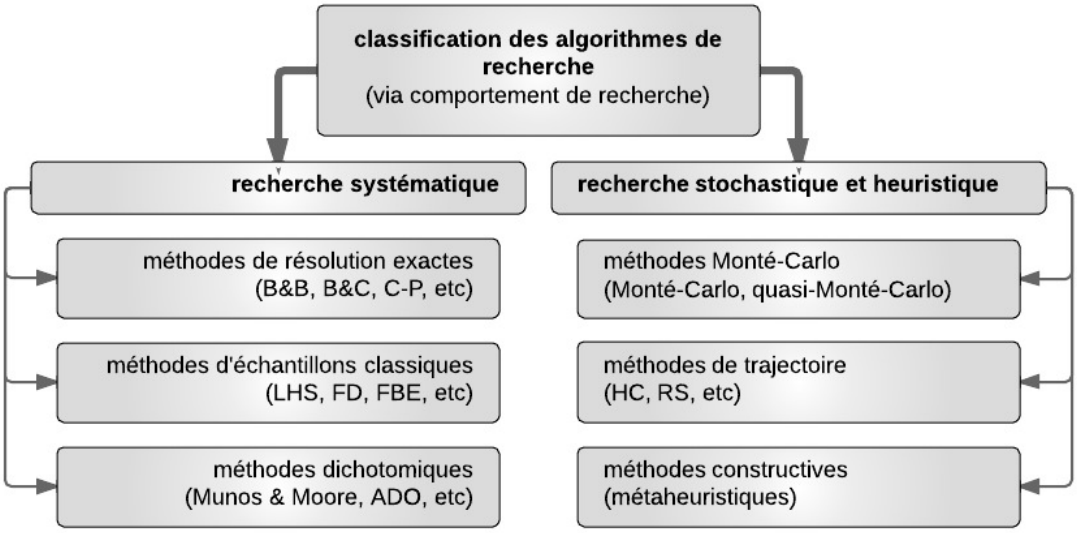

Figure 2.5 – Classification des algorithmes en fonction du comportement de recherche

Les méthodes de plan d'expériences classique (par exemple, l'échantillonnage des hypercubes latins, plan factoriel ou plan d'expériences basé sur les fréquences) du domaine des statistiques sont des exemples dans cette catégorie (algorithme d'exploration systématique). Par exemple, Kleijnen [KSLC05] préconise l'utilisation d'approches basées soit sur l'échantillonnage des hypercubes latins (LHS) quand il y a un grand nombre de facteurs, soit sur les plans factoriels (factoriel design, FD) à plusieurs niveaux ou un plan d'expériences basé sur les fréquences (frequency-based experiments, FBE) pour le cas inverse. Ces techniques s'intéressent à la façon de générer efficacement les points d'échantillonnage dans un espace afin de comprendre l'effet de facteurs dans une expérimentation. Dans ces approches, l'espace de paramètres est découpé en plusieurs sous-espaces de façon systématique et après, des points de test générés dans ces sous-espaces. LHS garantit un certain degré de représentativité tout en découvrant peu l'espace. FD essaie de générer à parcourir l'espace. FBE tente de couvrir efficacement l'espace tout en limitant le nombre de points. Ce type d'algorithme se limite à un nombre déterminé d'échantillons réalisés. Il ne s'utilise que pour résoudre les problèmes dont les facteurs sont indépendants et de l'équidistance entre les échantillons.

Une autre solution pour l'exploration lorsque'une recherche exhaustive par énumération explicite est impensable est d'utiliser les méthodes exactes. Parmi les méthodes exactes, on trouve la plupart des méthodes traditionnelles telles les techniques de séparation et d'évaluation (branch-and-bound, B&B), la méthode de coupes planes (Cutting-Plane, C-P), les méthodes Branch and Cut ou B&C. Les méthodes exactes s'appliquent à des problèmes de taille raisonnable. Il s'agit de faire une balance entre la manière de trouver les solutions de bonne qualité sans garantie d'optimalité et le temps de calcul réduit. Cependant l'inconvénient est que ces méthodes n'appliquent efficacement que sur l'optimisation combinatoire des paramètres discrets. Le temps de calcul nécessaire pour trouver une solution risque également d'augmenter exponentiellement avec la taille du problème [HGH99].

Dans la gamme des algorithmes systématiques, on trouve les algorithmes d'exploration dichotomiques. L'idée originale de cette gamme est d'adopter une politique de recherche quasi-optimales [MM02]. Cet algorithme définit un ensemble de politiques de résolution et de représentations en fonction des résultats d'évaluations. Par exemple, on divise l'espace de paramètres successivement en des cellules afin de trouver un point optimal. Chaque cellule est évaluée par un ensemble de critères de fractionnement, par exemple, la valeur moyenne et la variance qui permettent d'évaluer l'impact d'une cellule sur les autres au moment de décider de diviser.

Benoît Calvez et Patrick Taillandier se sont inspirés de l'algorithme dichotomique dans leur travail pour explorer l'espace des paramètres de la simulation multi-agents. L'idée de Benoît Calvez [CH$^+$07] est d'explorer l'espace de paramètres en fonction de l'intérêt potentiel des différentes régions de l'espace (figure 2.6). Dans cet exemple, l'espace de paramètres multi-dimensionel est initialement divisé en hypercubes. La valeur de chaque paramètre est aléatoirement choisie parmi les intervalles. Chaque intervalle d'un paramètre est évalué par la valeur moyenne de la fonction objectif en exécutant une ou plusieurs simulations. En fonction de ces résultats, on choisit les nouveaux intervalles dans lesquels les valeurs des paramètres ont été choisies. La récompense est proportionnelle à la fonction objective globale du modèle. Pour chaque paramètre, on va diviser ou fusionner les différents intervalles en fonction des récompenses reçues. L'idée est de diviser les zones intéressantes pour affiner l'évaluation dans cette zone et de fusionner les intervalles de zones de l'espace des paramètres de faibles intérêts pour réduire le temps d'exploration. Dans un autre travail, Patrick Taillandier [TA$^+$11] propose une stratégie de cartographie de l'espace de paramètres. Cette idée est d'utiliser la stratégie dichotomique. La différence est que il pré-traite en divisant l'espace de paramètres en grille de points. Chaque zone est liée par $2s$ points (s : nombre de paramètres). L'algorithme va diviser k cellules les plus intéressants en deux. Une évaluation des intéressantes cellules repose sur deux indicateurs : la moyenne de la distance euclidienne entre les points frontières et sa variance. Et en même temps, pour éviter la convergence rapide dans une zone, on génère aléatoirement r zones à diviser en deux. De cette façon, la cartographie représentative de l'espace de réponse va être désignée en reposant sur la décomposition réactive de l'espace de paramètres.

2.4.2 Algorithme d'exploration stochastique et heuristique

Cette catégorie commence avec les techniques d'échantillonnage stochastique avec un nombre indéterminé d'échantillons nécessaires. Les méthodes Monte-Carlo et quasi Monte-Carlo [Lem09] sont les plus populaires de ce type. Ces techniques permettent de réaliser l'exploration de l'espace de paramètres avec grand nombre de dimensions. De plus, ils favorisent l'exploration uniforme et non biaisée. Ces deux se basent sur l'idée de l'approximation de l'intégrale d'une fonction par la moyenne des valeurs de la fonction évaluée en un ensemble de points d'es-

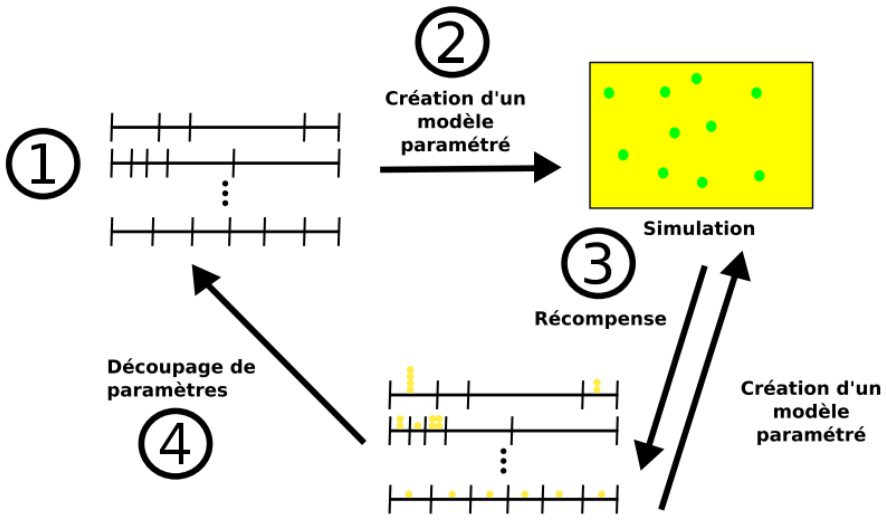

Figure 2.6 – Schéma résumant la méthode ODA [CH+07]

sai. Pendant que la méthode Monte-Carlo utilise une suite de nombres pseudo-aléatoires, la méthode quasi-Monte-Carlo utilise une suite à discrépance faible telle que la suite de Halton, la suite de Sobol ou la suite de Faure [Lem09]. La méthode quasi-Monte-Carlo possède l'avantage de permettre une convergence plus rapide en utilisant la vitesse de convergence des suites à discrépance faible tout en ayant un intervalle de confiance grâce à l'introduction de l'aléatoire. Mais Monte-Carlo manque d'un mécanisme pour guider la recherche dans les zones prometteuses.

Les approches par Métaheuristique [OL96] est un ensemble d'outils pour concevoir un processus itératif qui subordonne et guide une heuristique pour explorer et exploiter tout l'espace de recherche. Il n'est pas déterministe, il souvent suit un principe stochastique qui utilise l'expérience accumulée durant la recherche pour mieux guider la suite du processus de recherche. L'information est structurée par les stratégies d'apprentissage pour trouver efficacement des quasi-optimales dans les problèmes complexes. On peut distinguer deux classes de métaheuristiques : les méthodes de trajectoire et les méthodes basées sur une population.

Les méthodes orientés trajectoire démarrent avec une solution initialement complète (obtenue de façon exacte, ou par tirage aléatoire). L'algorithme s'en éloigne ensuite de manière itérative pour essayer d'améliorer cette solution en explorant son voisinage. Le processus désigne un parcours progressif dans l'espace de paramètres. Les exemples dans cette catégorie comptent : la méthode de descente (hill-climbing, HC), le recuit simulé (simulated annealing, SA), la méthode Tabou (tabu search, TB), etc.

Les méthodes basées sur une population génèrent un ensemble de solutions à partir d'une solution initiale en essayant d'en ajouter petit à petit des éléments jusqu'à obtenir une solution complète. Un des algorithmes les plus utilisés dans cette gamme d'approche est les algorithmes évolutionnaires (evolutionary algorithms, EA). Les méthodes basées sur une population sont donc des techniques de recherche et d'optimisation qui s'inspire de l'idée de la sélection naturelle darwinienne. Elles n'ont pas besoin d'information sur le gradient ou hessien et ne nécessitent pas d'information de dérivation. Chaque candidat (jeu de paramètres) joue le rôle d'un individu et compose un génotype. Le génotype est un vecteur ordonné de valeurs des paramètres associés. Chaque génération est composée d'individus qui s'appelle population. À partir de la population, EA applique des opérateurs darwiniens (sélection, re-

combinaison, mutation et reproduction) pour manipuler les individus et pour produire une nouvelle génération. Plusieurs générations sont nécessaires pour obtenir une population de jeux de paramètres plus prometteuse de temps en temps en se basant sur les fonctions objectives. La fonction de EA s'inspire des processus biologiques. Elle initialise par tirage aléatoire (via une distribution de probabilité) une population initiale (une première génération). Après avoir évalué la totalité ou une partie des individus, les meilleurs individus sont sélectionnés en tant que parents à travers un mécanisme de sélection, pour la reproduction. Ensuite, elle produit les nouveaux individus par recombinaison et/ou mutation. Après cela, elle évalue les valeurs fitness de ces nouveaux individus. A la fin d'une génération, EA remplace les pires individus de la population par de nouveaux individus. L'algorithme évolue jusqu'aux conditions d'arrêt. Il y a une grande variété des algorithmes évolutionnaires comme l'algorithme génétique (genetic algorithm, GA), les stratégies d'évolution (evolution strategies, ES), la programmation évolutionniste (evolutionary programming, EP), etc. De nombreux autres algorithmes se situent dans cette catégories tels que : l'optimisation par essaim de particules (particle swarm optimization, PSO) et les algorithmes de colonies de fourmis (ant colony optimisation, ACO) ou la recherche dispersée (Scatter Search, SS).

En couvrant tout l'espace de paramètre, une méthode basée sur une population évalue simultanément plusieurs jeux de paramètres et évite la convergence vers un minimum local, une zone non prometteuse[Óla06] comme on pourrait le constater avec les méthodes orientées trajectoire. En comparaison avec les plans d'expériences, cette méthode cherche soigneusement dans une région spécifique où certains points émergents se produisent [Aza99]. Cette seconde propriété est très importante dans l'exploration de simulations complexes, car il est généralement chaotique. L'EA possède néanmoins quelques inconvénients dont nous discutons dans les sections suivantes.

2.4.3 Qualification des méthodes l'exploration

Une des difficultés principales dans l'exploration des simulateurs complexes se situe dans la variabilité de l'espace des solutions. La forme de l'espace de solutions rugueuse ou lissée influence le choix d'une stratégie plutôt qu'une autre. Parmi plusieurs algorithmes d'exploration étudiées, nous les catégorisons selon leur comportement d'exploration sur l'espace de paramètres et notre perspective en 3 catégories sur la figure 2.7 :

- **Recherche locale :** à partir d'un jeu de paramètre de départ, cette stratégie vise à parcourir de proche en proche l'espace des paramètres tout en maximisant (ou minimisant) une fonction d'utilité. Il s'agit donc d'approches dirigées par le voisinage ou à granulation discrétisée si bien que l'exploration peut s'enfermer et s'achever dans un optimum local. Dans un contexte favorable, ces approches sont rapides et efficaces. La recherche locale s'appuient sur des algorithmes orientés trajectoire tel que recuit simulé, la recherche tabou, les méthodes exactes.
- **Recherche globale :** cette stratégie vise à couvrir la totalité de l'espace de recherche et déterminer par raffinement successifs une solution optimale. Ces approches sont particulièrement performantes lorsque l'espace de recherche est multimodal. Une stratégie de recherche globale peut être réalisée de diverses manières : par de multiples réplications d'un algorithme de recherche locale à partir des différents points initiaux ; par un algorithme de recherche globale ; une hybridisation des algorithmes d'exploration (tous les deux types globaux et locaux). Les algorithmes évolutionnaires s'inscrivent dans cette catégorie.
- **Recherche par criblage :** la recherche par criblage est une stratégie de recherche systématique visant à cribler l'espace des paramètres en vue de détecter des zones

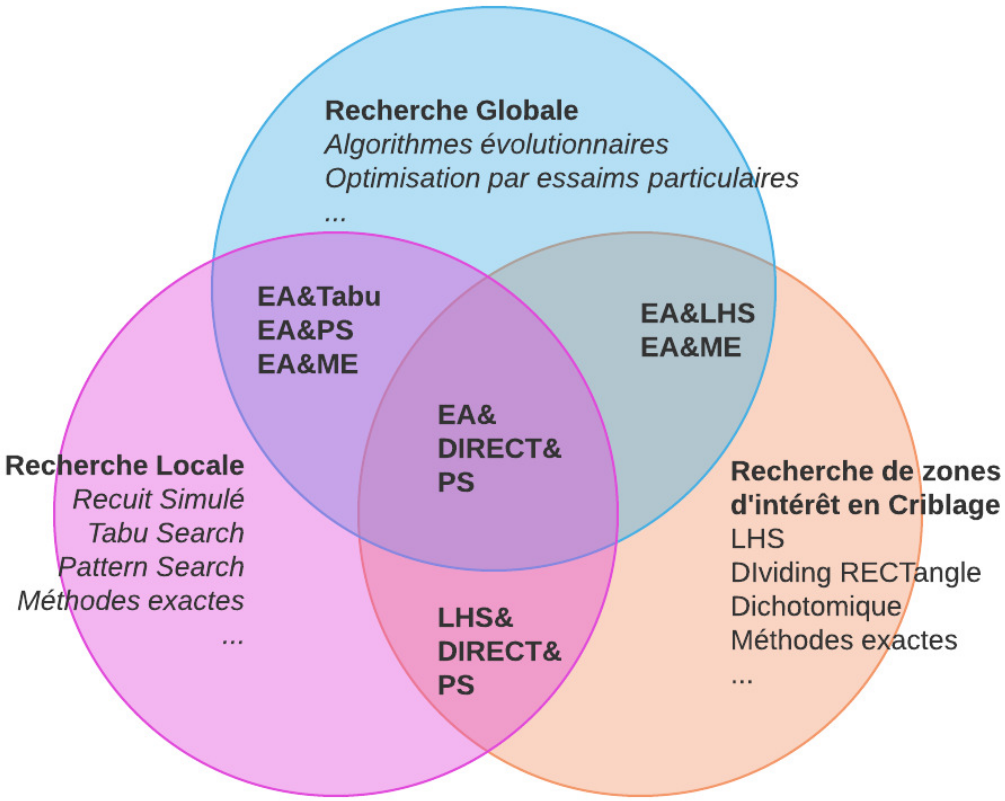

Figure 2.7 – Classification des algorithmes en fonction du comportement de recherche

prometteuses. Une solution simple de criblage vise à échantillonner l'espace des paramètres à intervalles réguliers. D'autres stratégies s'appuient sur des heuristiques pour diviser et fusionner afin de focaliser la recherche dans des zones optimales. Ce type d'approche est efficace et donne avec peu d'itération un résultat. Mais le résultat reste imprécis. Les exemples typiques de cette catégorie sont les techniques d'échantillonnage, ou de recherche dichotomique.

En résumé, les 3 stratégies précédemment citées sont complémentaires. Les approches de recherche locales sont précises et efficaces mais tombent dans le piège de l'optimum local. Inversement, les approches de recherche globale s'affranchissent d'optimums locaux mais restent moins précises. Les approches de criblage sont quant à elles adaptées pour réaliser une cartographie de l'espace des paramètres. Cette complémentarité des approches nous laisse imaginer l'intérêt de les associer.

Discussion

Nous pouvons conclure qu'un simulateur est souvent considéré comme un transformateur de variables d'entrée, provenant du milieu environnant, en variables de sortie. Cette transformation est réalisée à l'intérieur d'une boîte noire, pour souligner le caractère complexe du système. Pour l'explorer, il faudra lancer un ensemble de simulations avec différentes configurations de paramètres, de répétition, ce qui va générer l'information, de la donnée, permettant d'analyser et comprendre le système étudié. L'exploration est une phase du processus de mo-

délisation et simulation pour laquelle une stratégie doit être identifiée au cas par cas, selon les caractéristiques étudiés dans 1.2.1. Alors, l'exploration devient une étape visant à réaliser une recherche sur l'espace de paramètres en vue de déterminer une solution optimale pour un objectif prédéfini de l'étude. Cette approche s'appelle un problème inverse ou l'exploration inverse. L'exploration inverse utilise les algorithmes d'optimisation pour faire la recherche. Plusieurs classes des algorithmes d'exploration existent (recherche par criblage, recherche locale, recherche globale). Chaque classe de méthodes est adaptable à une grande variété de problèmes et mènent à des résultats pertinents. Mais il est difficile de montrer qu'une sera plus efficace qu'une autre sur un problème donné. Certains algorithmes présentent l'avantage d'être simples à implémenter tels que les recherches dichotomiques, d'autres sont bien adaptés à la résolution de certains problèmes de façon déterministe comme les méthodes exactes.

En somme, il n'y a pas d'algorithme optimisé pour tous les cas de figure. Il n'y a pas non plus une implémentation pertinente pour un problème.

Dans ce chapitre, nous avons présenté les méthodes d'exploration existantes en littérature et la promesse de ces méthodes pour l'exploration de simulateurs complexes. Cependant, l'exploration de simulateurs complexes demande des ressources pour le calcul. Dans le chapitre suivant, nous présentons les outils et les techniques utilisés pour intégrer les simulateurs complexes et faciliter l'utilisation des méthodes d'exploration dans le cadre de logiciels distribués.
.

Chapitre 3

Techniques et outils pour l'exploration large échelle de simulations complexes

Sommaire

Ce chapitre présente une revue de travaux existants sur les techniques et outils pour passer à l'échelle de l'exploration de simulations complexes. Quand le temps d'exécution et le nombre de paramètres du simulateur augmentent, il faut trvailler à l'accélération des méthodes de l'exploration originale. De plus, il faut également concevoir les outils qui aident les utilisateurs en facilitant la conception d'une stratégie pour accélérer l'exploration. Dans ce chapitre, on parle principalement de ces deux axes. Le chapitre est organisé comme suit : premièrement, une classification des techniques de passage à l'échelle de l'exploration des paramètres dans les simulations complexes ; deuxièmement, les différentes stratégies utilisées pour l'exploration à large échelle des simulateurs complexes ; finalement, l'exploration large échelle sur les systèmes de calcul intensif.

3.1 Passage à l'échelle de l'exploration de paramètres

Pour faire face à la complexité croissante des systèmes complexes, le calcul haute performance (High Performance Computing, HPC) est une des clefs permettant d'accélérer le calcul et donc de réduire les temps avant d'avoir des résultats.

L'adaptation des algorithmes d'exploration traditionnels devient une mission urgente pour le passage à l'échelle. Ces algorithmes sont modifiés pour profiter des ressources de calcul intensif telles que les grilles ou les clusters. Nous avons identifié 3 techniques majeures pour le passage à l'échelle de l'exploration des simulateurs complexes : (i) la parallélisation des algorithmes ; (ii) la construction d'algorithmes hybrides ; (iii) la mise en place de stratégies collaboratives entre les algorithmes. Ces trois points sont développés dans la suite de la section.

3.1.1 La parallélisation des algorithmes d'exploration

Plus un problème d'optimisation devient compliqué, plus les temps de calcul et les ressources utilisées pour le résoudre sont importantes. Il devient nécessaire de paralléliser ce processus pour passer à l'échelle l'exploration de simulations complexes. De nombreuses implémentations des algorithmes d'exploration que l'on a présenté dans le chapitre 2, ont été proposées afin de diminuer le temps de calcul nécessaire et passer à l'échelle lors de l'exploration.

Une exploration de simulateur complexe se compose en deux parties : (1) évoluer les algorithmes d'exploration et (2) évaluer la performance des jeux de paramètres. Cela veut dire qu'il y a deux directions pour passer une exploration à l'échelle : passage à l'échelle du calcul des paramètres et passage à l'échelle de l'algorithme d'exploration. Ces deux directions sont discutées dans la suite.

Passage à l'échelle des algorithmes d'exploration

La parallélisation et la distribution sont des approches permettant de lancer plusieurs simulateurs en même temps pour tirer le meilleur parti du système de calcul intensif. La parallélisation de l'algorithme, tente de changer le comportement de l'algorithme original en divisant ou ajoutant les éléments parallélisables. La parallélisation de l'algorithme peut être catégorisée en trois parties : (i) la parallélisation des opérations de l'algorithme, (ii) la décomposition de l'espace de recherche ou (iii) l'exécution de multiples instances du solveur [CT03]. La parallélisation de l'algorithme combine souvent la distribution et la parallélisation du calcul.

1. *la parallélisation des opérations d'un algorithme* : elle se réalise en exécutant concurremment une opération de recherche dans une itération de l'algorithme. Alors, il reste la même procédure de recherche, même comportement à chaque itération pour résoudre une seul problème dans le domaine de recherche. La différence est qu'une opération peut être gérée par un thread. Un exemple est la division de la population de l'algorithme génétique pour les évaluer en parallèle dans une itération.
2. *la décomposition explicite de l'espace de recherche* : dans ce cas, on décompose l'ensemble des variables importantes en sous-groupes disjoints dans un processus *maître*. Un algorithme différent est appliqué pour chaque sous-groupe dans un processus *esclave* pour trouver une sous-solution dans ce sous-espace. Plusieurs *esclaves* s'exécutent concurremment et indépendamment. Il y a alors un point de synchronisation pour construire

la solution globale après l'exécution de tous les *esclaves*.

3. *multiple instances des solveurs* : un solveur est un processus d'exécution d'un algorithme. il est capable que plusieurs solveurs fonctionnent pour résoudre une solution dans tout l'espace. Les solveurs concurrent peuvent exécuter la même heuristique ou des heuristiques différentes. Ils commencent par un état initial (identique ou différent) et communiquent au cours de la recherche ou seulement à la fin pour identifier la meilleure solution globale. Les communications peuvent être effectuées de manière synchrone ou asynchrone et peuvent être exécutées à des moments prédéterminés ou décidés dynamiquement. De cette classification, on peut voir que les algorithmes d'exploration coopérative (présentés dans la sous-section suivante) sont mis comme un type spécial d'algorithmes parallèles. Le multi-threading est souvent utilisé pour effectuer une exploration plus approfondie de l'espace des solutions. Plusieurs études ont montré que les multi-threads donnent de meilleures solutions que les méta-heuristiques séquentiels correspondants, même lorsque la durée d'exploration de chaque thread est significativement inférieure à celle du calcul séquentiel. Des études ont également montrés que la combinaison de plusieurs threads qui mettent en œuvre différents paramètres augmentent la robustesse de la recherche globale.

Le passage à l'échelle du calcul des paramètres est une autre direction de passage à l'échelle de l'exploration de simulateur complexe que l'on discute dessous.

Parallélisation et distribution des calculs des paramètres

Pour paralléliser le processus du calcul de l'évaluation des paramètres, il y a deux approches : la parallélisation du calcul et la distribution du calcul. Ces deux dernières sont présentées dans la suite :

La parallélisation du calcul : dans cette approche, pour réduire la durée d'exécution d'une seule simulation, le simulateur doit être distribué. En général, la fonction peut être considérée comme une composition de fonctions partielles. Les fonctions partielles pourraient également être identiques. Pour les fonctions partielles identiques, l'évaluation en parallèle et les accès à la base de données sont effectuées en parallèle. En outre, les fonctions partielles pourraient également être différentes. Pour chaque fonction partielle différente, il faut indiquer un opérateur d'agrégation de ces fonctions partielles. Autrement dit, dans le domaine des systèmes complexes, une seule simulation est partagée par plusieurs nœuds de calcul. Par exemple, la population d'agents et son environnement sont divisés en plusieurs groupes et peuvent être distribués sur chacun des nœuds. Pourtant, une telle parallélisation de simulateurs, spécialement simulateurs à base d'agents, implique de nombreux problèmes tels que la synchronisation de l'espace et le temps dans un contexte distribué. Pour cette raison, notre travail concentre sur la principe de distribuer le calcul de l'évaluation des paramètres.

La distribution du calcul : dans cette approche, un plan d'expérience est distribué. L'évaluation de la performance des jeux de paramètres est constituée de calculs totalement indépendants entre eux. La distribution consiste à déployer le calcul de l'évaluation et exécuter ses simulations sur plusieurs nœuds (processeurs ou cœurs). Par exemple, si on a mille simulations qui ont besoin d'être lancées sur un cluster de de 5 nœuds, deux cents simulations pourraient être affectées à chaque nœud. Chaque simulation est indépendante de l'autre. Les simulations n'échangent pas de donnée. Cette approche est plus simple à appliquer. Elle ne réduit pas la durée de calcul d'une seul simulation mais permet de profiter de l'architecture de grille de calcul pour lancer, en même temps, plusieurs simulations et donc de réduire le temps globale.

Dans la partie suivante, on va montrer le passage à l'échelle de l'exploration de simulateur complexe en utilisant l'exemple des algorithmes évolutionnaires, un type d'algorithme métaheuristique que l'on peut utiliser pour l'exploration des simulations complexes ceux qui ont été discutés dans le chapitre 2. Ce sont des algorithmes coûteux en temps calcul. Mais il est facile de voir que l'étape la plus coûteuse, l'évaluation de la performance d'une population, est constituée de calculs totalement indépendants entre eux.

Parallélisation d'un algorithme évolutionnaire

Quand on a la possibilité d'utiliser de ressources distribuées pour le calcul, la première idée est de lancer plusieurs instances d'un même algorithme d'exploration en même temps afin d'accélérer la vitesse d'exploration. Cette approche combine un ensemble d'instances entières ou partielles d'un algorithme dans un même environnement pour résoudre un même problème. Par exemple pour la gamme d'algorithmes évolutionnaires, il y a trois types de passage à l'échelle d'un algorithme métaheuristique : modèle en île (island models, IM, figure 3.1), modèle de diffusion (diffusion models, DM, figure 3.2) et le modèle de distribution globale (figure 3.4) [AT02] que l'on va étudier en détail dans la partie suivante.

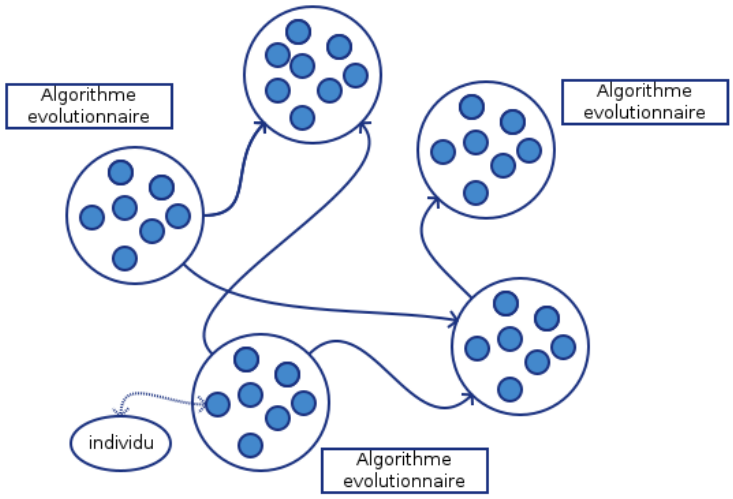

Figure 3.1 – Le modèle en île pour paralléliser l'algorithme évolutionnaire

Le modèle en île (island models, IM, figure 3.1) réalise un ensemble d'exécutions d'un algorithme. Par exemple, l'algorithme évolutionnaire considère simultanément un ensemble de sous-populations divisées en îles. Chaque île isolée exécute indépendamment une des stratégies d'évolution complète. Elles échangent les individus occasionnellement. Cet échange d'individus est appelé "migration". L'avantage du modèle est d'accélérer la vitesse de convergence. Cependant, outre les paramètres classiques de l'algorithme évolutionnaire, le nombre d'îles, le taux de migration, la fréquence de migration, la topologie de migration, le mécanisme de remplacement des individus quand on fait l'échange ont un impact sur l'efficacité de l'algorithme. Par exemple, le remplacement peut soit se faire par la méthode aléatoire soit par le tirage au choix. La topologie de communication peut être en topologie complète, en topologie en anneau ou en topologie étoile. D'autre part, l'implémentation doit distribuer le codage des instances sur un nombre de nœuds disponibles qui exigent l'inter-communication entre ces nœuds. Avec un risque de défaillance de certains nœuds, elle doit assurer la stabilité de cette approche, car une instance sera perdue et ne peut plus évoluer. Alors, ce modèle s'adapte au réseau pair-à-pair des ordinateurs.

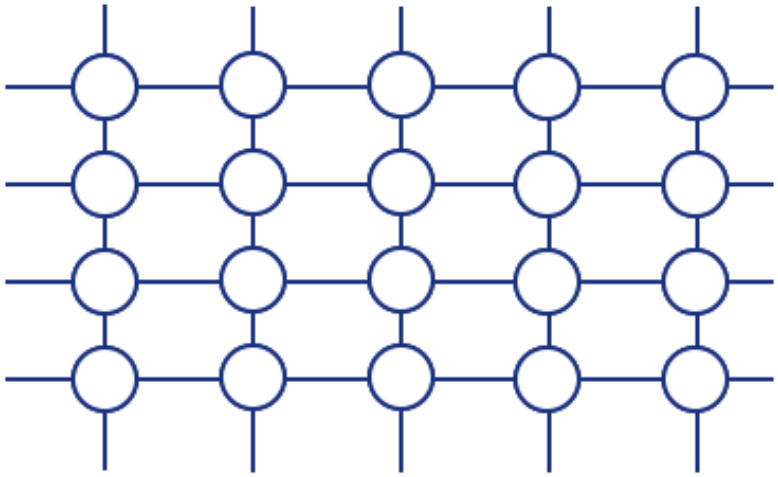

Figure 3.2 – Le modèle en diffusion pour paralléliser l'algorithme évolutionnaire

Deuxièmement, le modèle de diffusion (diffusion models, DM, figure 3.2) réalise l'algorithme avec un groupe de petite taille d'individus par nœud de calcul. Dans ce modèle, la population totale est divisée en plusieurs groupes d'individus interactivement associés à une structure spatiale durant plusieurs phases de recherche. Seuls les individus dans un voisinage proche peuvent coopérer. La reproduction ne se fait aussi qu'entre individus voisins. Malgré cette reproduction locale, les caractéristiques intéressantes vont être propagés de voisins en voisins. Ce modèle maintient plusieurs niveaux de diversification et d'intensification : la garantie de la convergence locale par le niveau individuel et la garantie de la diversité globale par le niveau collectif. L'évolution de l'algorithme est globale pendant que la sélection et la reproduction sont locales. L'avantage du modèle est qu'il est capable de s'adapter aux tailles différentes de la structure de grille. Cette approche est efficace uniquement lorsque le nombre de ressources disponibles est grand, homogène et l'inter-communication est rapide. Ce modèle est approprié pour les ordinateurs massivement parallèles tels que cluster, grid.

On voit que les deux types de parallélisation de l'algorithme évolutionnaire (présenté ci dessus) demandent toujours une division en sous-groupe de solutions et l'inter-connexion entre les individus. L'évolution de l'algorithme s'appuie sur plusieurs nœuds de calcul. Il faut avoir un mécanisme pour gérer le cycle de vie sur chaque nœuds. L'approche change l'idée classique de l'algorithme. Il y a une autre approche pour distribuer le calcul de l'évaluation globalement sans changer le comportement de l'algorithme. Nous discutons de cette approche dans la section suivante.

Distribution globale du calcul de l'évaluation

Cette approche essaie de maintenir l'algorithme évolutionnaire original mais accélère la vitesse pour intensifier la capacité de calcul du système distribué : le modèle maître-esclaves (master-slaves, MS), le modèle de l'évolution stationnaire (steady-state, SS), ou modèle de grille (grid model, GM) de la gamme des algorithmes évolutionnaires distribués [AT02, NLLA08].

Dans le modèle maître-esclaves (figure 3.4), à chaque génération, l'ensemble des nouveaux individus est distribué entre les différents nœuds de calcul. Puis ces individus sont évalués et leur résultat est renvoyé au centre. L'EA est divisé en processus d'un maître et plusieurs esclaves. Le maître gère l'emploi d'une seule population, effectue l'initialisation des individus

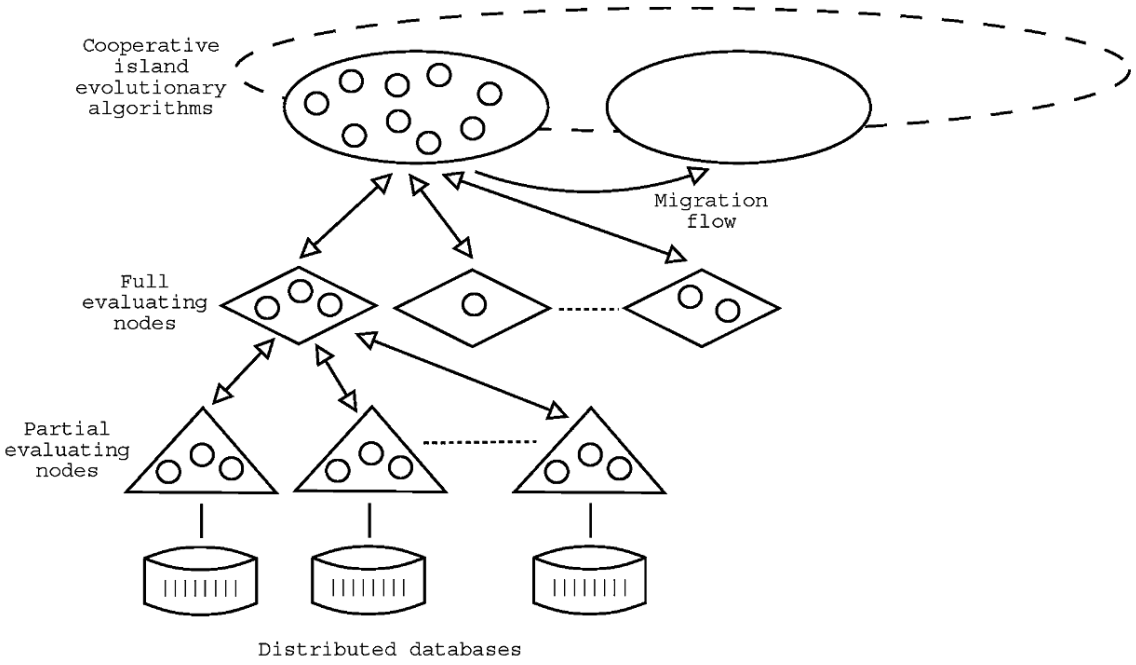

Figure 3.3 – La taxonomie des approches de parallélisation des algorithmes [AT02]

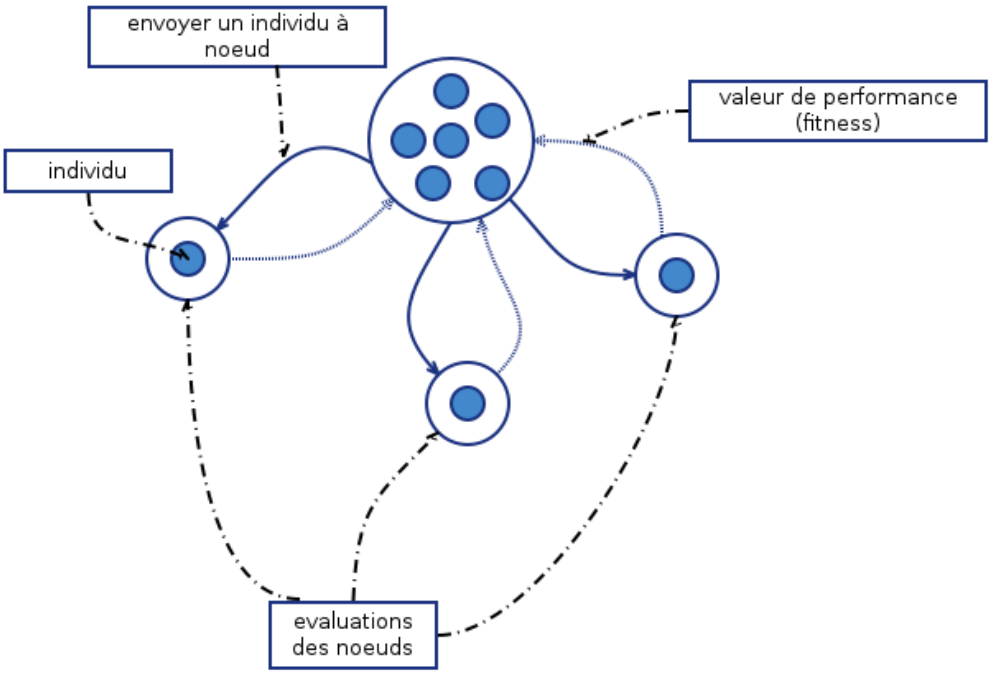

Figure 3.4 – Le modèle de distribution globale du calcul pour passage à l'échelle du calcul de l'évaluation de paramètres

et l'exécution des opérateurs (recombinaison, mutation, sélection) en recevant des données à partir d'un ensemble d'esclaves. Les esclaves font le calcul de l'évaluation des individus et envoient les données partielles évaluées au maître. Chaque esclave s'exécute de manière indépendante. Ce modèle est très simple à mettre en œuvre et adapté à un système comme cluster, grille de calcul haut débit. Il permet de conserver la structure de l'algorithme classique en

dépit du grand nombre de processeurs. La structure centralisée peut contrôler l'équilibre de la charge sur les groupes hétérogènes de processeurs. En outre, il simplifie la collecte et l'analyse des résultats. Cependant, il apparaît un problème de surcharge du côté du maître lorsqu'il y a un grand nombre de gènes, une panne du nœud maître, et des coût de la communication importants entre le maître et les esclaves.

Le modèle de l'évolution stationnaire est basé sur le modèle maître-esclaves. Dans ce modèle, un seul enfant est généré à chaque génération en utilisant un ou deux parents, le plus souvent sélectionnés par tirage. Cet enfant, après avoir été évalué, est intégré dans la population. Le mécanisme de sélection est appliqué pour faire la comparaison entre l'enfant et ses parents. L'individu sélectionné survit. La comparaison entre le nouvel individu avec ses parents est réalisée dès qu'il est disponible (c'est à dire évalué) et le maître n'a pas besoin d'attendre l'évaluation de tous les individus d'une population de solution. Cette approche nécessite de nombreuses communications entre processeurs maître-slaves pour envoyer les nouveaux individus à évaluer l'un après l'autre. Cette approche est différente de la méthode maître-slaves : le processeur maître n'attend pas que tous les processeurs esclaves aient terminé leurs calculs pour passer à la génération suivante. Cette méthode est est avantageuse si l'évaluation des individus est assez longue et que le coût de communication maître-slaves n'est pas trop important. Un inconvénient commun de ces deux approches est qu'elles ne sont pas adaptées pour être exécutées sur un environnement de calcul dynamique hétérogène comme une grille de calcul volontaire (`https://boinc.berkeley.edu/`), les grilles de calcul large-échelle, les clusters avec les politiques de gestion des ressources différentes et séparées géographiquement qui constituent une grille où les nœuds de calcul apparaissent et disparaissent fréquemment dans le système .

Le modèle de grille GrEA [NLLA08, DSV08] est basé sur le modèle maître-esclaves. Le maître exécute la boucle principale de l'algorithme pendant que les esclaves exécutent les calculs de l'évaluation des individus (calcul de fonction fitness) de façon asynchrone. Comme le modèle de l'évolution stationnaire, dans GrEA plusieurs évaluations sont effectuées en parallèle. Idéalement, il devrait y avoir le même nombre d'évaluations parallèles que le nombre de processeurs disponibles dans la grille de calcul. On essaie de remplir la capacité maximale du calcul. Dans la première phase de l'algorithme où la taille de la population est inférieure à la taille maximale de la population : le maître initialise aléatoirement une population pendant qu'une mise à jours des solutions évaluées est renvoyées par les esclaves. Quand il y a assez de solutions évaluées dans la population, le maître évolue en appliquant les opérateurs génétiques sur chaque individu reçu et évalué. Il insère un nouvel individu quand ce dernier est mieux que le pire membre actuel, et retire le pire individu pour maintenir la population. Quand un nœud disponible est détecté, un message de demande d'une reproduction ou d'une mutation pour conduire à un nouvel individu sera envoyé à un esclave. Cette méthode est abordée dans le travail de Travis Desell [DSV08] (figure 3.5). De cette façon, le nombre de tâches en cours d'évaluation est peut-être plus nombreux que la taille de population et fixe avec le nombre de nœuds de calcul disponibles. Cet algorithme présente des avantages importants dans des environnements hétérogènes. L'algorithme progresse plus vite qu'une évaluation finie. Les évaluations les plus rapides peuvent s'enchainer sans attendre les calculs les plus lents. En outre, les calculs plus lents peuvent améliorer encore la vitesse de l'algorithme parce que leurs résultats peuvent encore être utiles où ils sont reçus. Cela introduit un changement du principe du comportement de l'algorithme original parce que les individus anciens peut arriver dans la population lorsque cela a évolué plusieurs générations. Le seul facteur limitant à l'évolutivité est la vitesse d'insertion d'un individu dans la population et la vitesse de traitement des opérateurs. Il est également possible d'avoir plusieurs maîtres qui partagent la même population pour accélérer l'évolution.

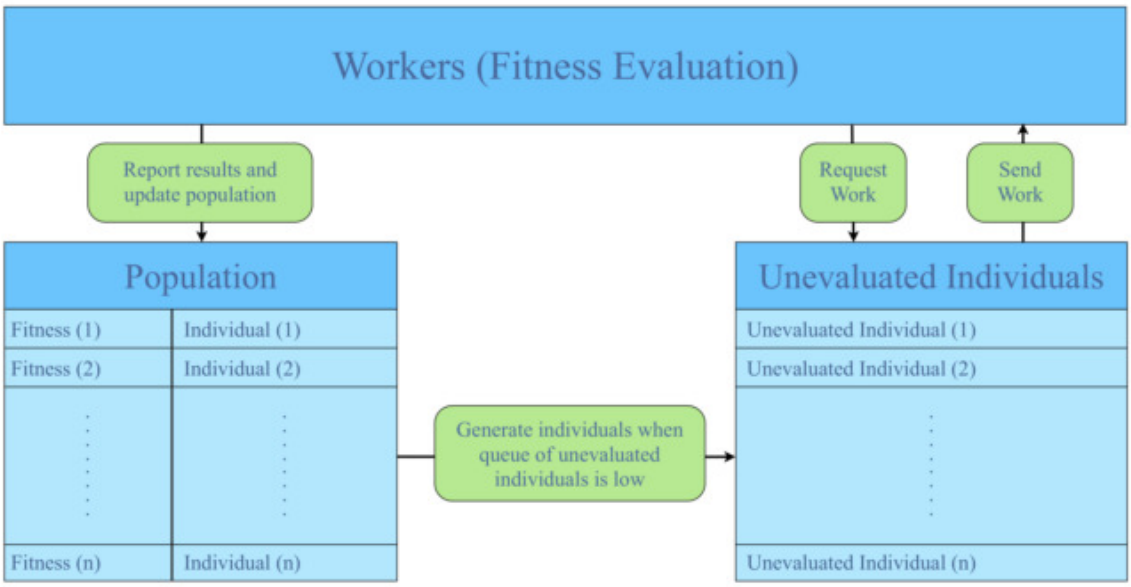

Figure 3.5 – Le modèle du grille GrEA pour distribuer l'algorithme évolutionnaire

Dans cette parti, on a étudié les différentes approches pour paralléliser ou distribuer un seul algorithme évolutionnaire pour passer à l'échelle l'exploration des simulateurs complexes. Dans la partie suivante, on va étudier les méthodes pour hybrider l'algorithme évolutionnaire avec les autres algorithmes et ainsi accélérer l'exploration des simulateurs complexes.

3.1.2 Hybridation des algorithmes

La deuxième approche pour renforcer l'exploration est d'hybrider des algorithmes d'exploration. Un algorithme hybride est une combinaison d'algorithmes issus d'une même ou de différentes catégories [Tal02] par exemple : la combinaison entre un algorithme de recherche globale avec un algorithme de recherche locale. Il ne faut pas oublié que la distribution du calcul de l'évaluation dans la partie dessus n'est pas obligatoire dans l'hybridation. L'hybridation peut se faire sans avoir besoin de la distribution du calcul. Cela dépend du type d'implémentation de l'hybridation.

Selon la taxonomie proposée par [Tal02] (cf figure 3.6) de l'hybridation des algorithmes d'exploration, il y a deux critères hiérarchiques pour catégoriser les hybridations entre les algorithmes : (1) le niveau d'hybridation et (2) le mode d'hybridation.

Le premier critère, le niveau d'hybridation, se distingue entre l'hybridation à haut niveau et l'hybridation à bas niveau (figure 3.7). Dans l'hybridation à bas niveau, une méthode devient un composant fonctionnel (un opérateur) dedans une autre méthode qui l'englobe qu'on va expliquer cela dans la suite. A l'inverse, dans l'hybridation à haut niveau, les méthodes gardent leur intégrité propre mais communiquent avec les autres. Le second critère pour la classification est le mode (figure)3.8).

A côté de deux niveaux d'hybridation, on peut également distinguer les hybridations par utilisation d'un autre critère : le mode d'hybridation. Il y a deux modes d'hybridation : (1) le mode relais et (2) le mode co-évolutionnaire. Dans le mode relais, les solveurs sont exécutés successivement l'un après l'autre. le résultat de la précédente servant de l'entrée de la suivante.

Dans le mode co-évolutionnaire, les différents solveurs fonctionnent en parallèle dans un même l'espace de solutions pour généralement pouvoir explorer des différentes régions de l'espace de recherche.

On peut ainsi créer quatre groupes d'hybridation à partir des deux critères ci-dessus :

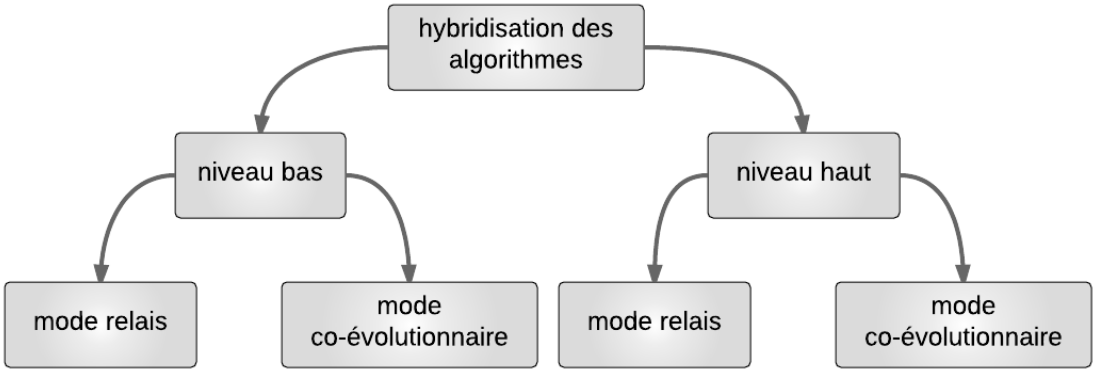

Figure 3.6 – La taxonomie des approches de hybridation

1. *hybridation à bas niveau en mode relais* : Cette classe regroupe les méthodes à base de solutions uniques telles que la recherche tabou, la recherche par descente de gradient ou recuit simulé, avec une autre méthode de même catégorie appelée en relais dedans pour modifier le comportement de l'algorithme originel. Par exemple, une méthode de recherche par descente de gradiant peut être utilisée pour perturber un optimum local d'un recuit simulé. La nouvelle solution (jeu de paramètres) devient un candidat pour le recuit simulé dans l'itération suivante. Son comportement est similaire à celui des recherches locales à voisinage multiple.
2. *hybridation à bas niveau en mode co-évolutionnaire* : Dans ce type d'hybridation, on remplace des opérateurs de croisement ou de mutation d'un métaheuristique à base de population par d'autres méthodes, souvent des métaheuristiques à base de solution unique. L'avantage de ce type de coopération est de combiner la diversification des algorithmes à base de population avec l'intensification des méthodes à base de solutions uniques. L'un explore le grand espace de recherche mais converge lentement. L'autre raffine les optimums locaux et converge plus vite. Cela renforce la capacité de l'exploration de l'hybridation. Un exemple est d'échanger l'opérateur de mutation par une recherche par descente, une recherche tabou ou encore par une recherche par recuit simulé.
3. *hybridation à haut niveau en mode relais* : Dans cette classe d'hybridation, les méthodes de recherche maintiennent leur intégrité mais collaborent de manière séquentielle. L'un finit avant que l'autre commence avec le pipeline entre eux. Cette procédure peut s'organiser avec une phase de génération des solutions initiales par la méthode gloutonne, ensuite, une phase de diversification pour couvrir une grande partie de l'espace de recherche par l'algorithme génétique et de finir avec une phase d'intensification pour raffiner les solutions obtenues par une recherche tabou.
4. *hybridation à haut niveau en mode co-évolutionnaire* : Dans cette classe d'hybridation, un ensemble des algorithmes d'exploration s'exécute imdépendament et en parallèle (i) les algorithmes partagent les solutions trouvées ; (ii) différents algorithmes d'exploration sont dans une même catégorie ou dans des catégories variées.

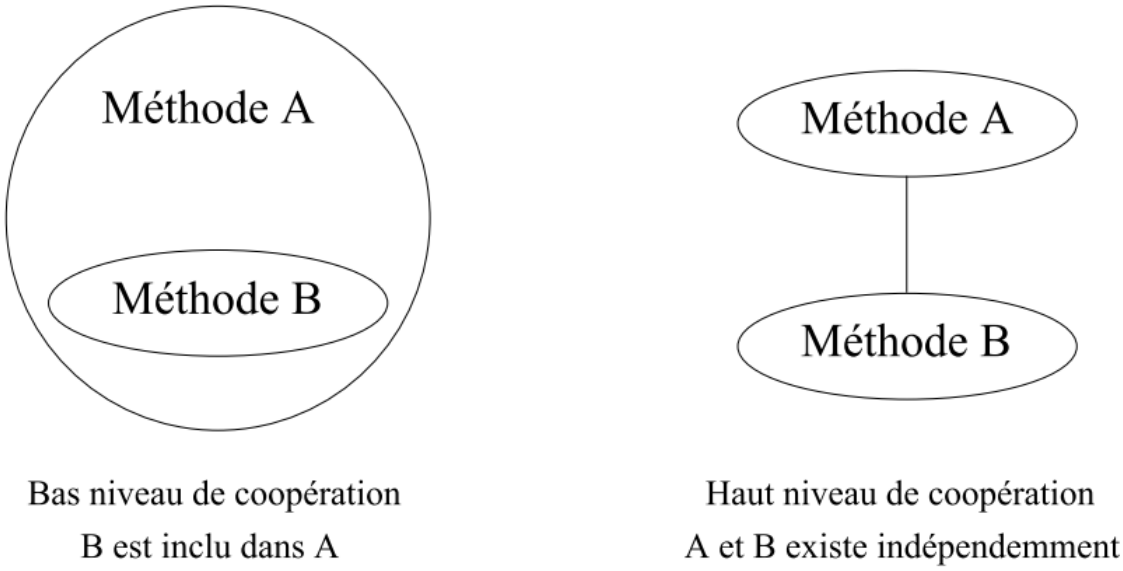

Figure 3.7 – La comparaison des approches d'hybridation, niveaux haut et bas

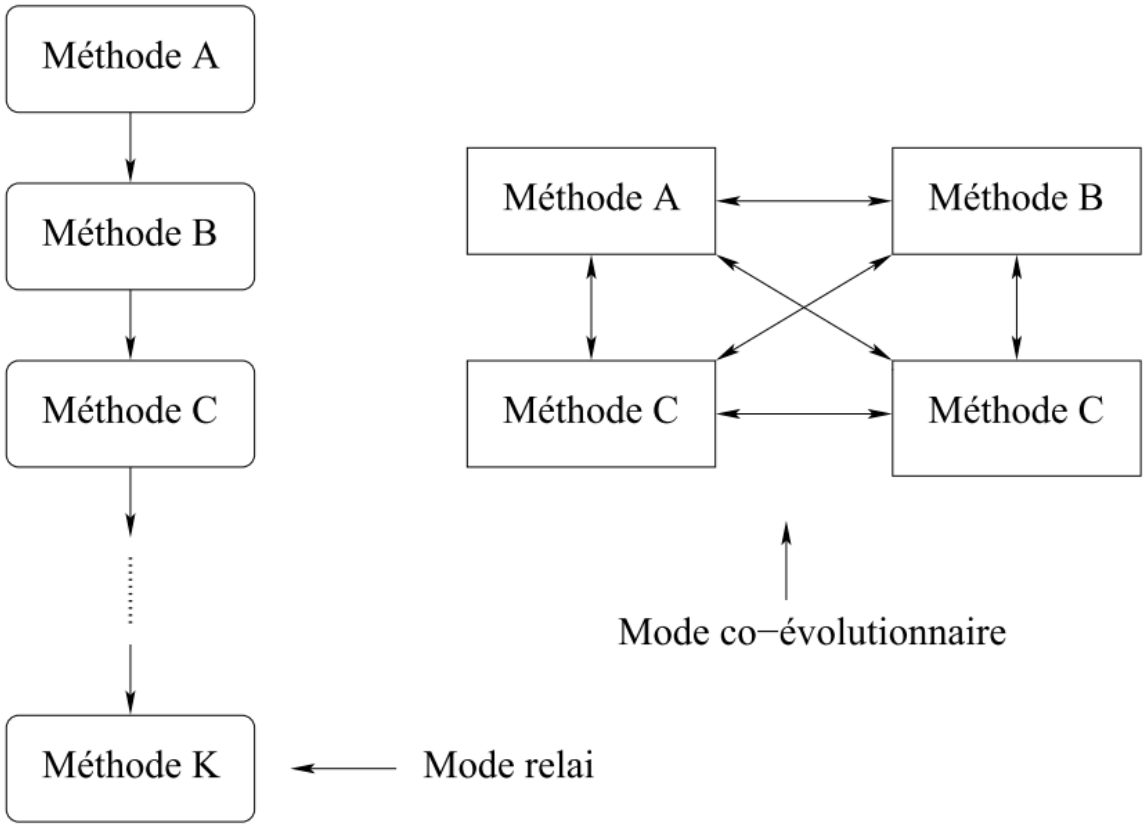

Figure 3.8 – La comparaison des approches d'hybridation, relais et co-évolutionnaire

Dans les deux premières hybridations de bas niveau, les algorithmes participent à l'hybridation comme un composant faisant partie intégrante d'un système unique. Si l'on perd l'un, le système ne s'exécute pas. Alors que, la partie coopérative est identifiée dans la technique de haut niveau, la recherche coopérative a été mise comme une catégorie distincte : stratégie coopérative. La coopération de métaheuristiques permet d'obtenir des algorithmes hybrides

avec une grande efficacité mais avec une mise au point coûteuse. Néanmoins, on remarque un grand nombre de caractéristiques similaires devant être identifiées lors de la mise en place de ces algorithmes d'exploration. Ainsi, un protocole générique, non dédié à un domaine particulier, peut être imaginé avant de formaliser la coopération entre ces algorithmes. Ceci laisse envisager une plate-forme qui profite de tous les algorithmes actuels pour les combiner et explorer des simulateurs complexes.

3.1.3 Stratégies coopératives

Les stratégies coopératives sont une des trois techniques pour passer à l'échelle de l'exploration des simulateurs complexes avec les deux autres que l'on a discuté précédemment (la parallélisation des algorithmes et la construction d'algorithmes hybrides). Cette sous-section fait une présentation rapide des stratégies coopératives. Après il y a une section spécifique pour discuter de stratégies coopératives pour l'exploration.

La nature des problèmes d'exploration de système complexe influence sur le choix des méthodes d'exploration. La nature de l'espace de réponse, son caractère complexe rend difficile la mise en place d'une méthode de recherche qui tire partie de la forme de l'espace et évite les pièges [KLT03]. Ensuite, si les évaluations de performance des solutions sont coûteuses, l'exécution d'un algorithme peut être trop coûteuse à cause d'un grand nombre d'évaluations [Neu04]. En outre, les méthodes d'exploration originales ne sont pas faciles paralléliser.

De nombreux travaux proposent d'associer, par exemple, les algorithmes évolutionnaires avec l'algorithme de recherche direct [KLT03]. Par exemple, l'algorithme mémétique est une hybridation très utilisée est qui combine un EA avec un algorithme de recherche local. Cet algorithme s'exécute en deux temps : (i) sélection d'un jeu de paramètres à l'aide d'un algorithme évolutionnaire ; (ii) utilisation d'un algorithme de recherche locale pour raffiner les solutions trouvées. Cette hybridation fait coopérer un algorithme évolutionnaire avec une recherche locale en différentes stratégies : (i) par l'intégration comme un opérateur génétique [MCM04] ; (ii) par une mémoire partagée [TB06a] ; (iii) ou par des interactions entre agents [MKC10]. La recherche locale (une méthode de trajectoire) est appliquée sur les nouveaux individus d'une population générés par des opérateurs de croisement, de mutation. Pendant que la recherche locale explore un espace de recherche restreint, les opérateurs génétiques parcourent la région d'exploration plus large.

L'objectif de l'hybridation de plusieurs algorithmes de classes différentes est de combiner les avantages de chacun au sein d'un même algorithme à travers leur coopération. Par exemple, leur association offre une cartographie de l'espace des paramètres et une meilleure exploration ensuite. Combiner différentes stratégies d'exploration permet de tirer parti de chacune d'elle voire d'inhiber les lacunes. Ce dernier point peut cependant devenir une faiblesse. De plus, un algorithme résultant de l'hybridation de plusieurs algorithmes peut avoir une complexité importante. La parallélisation de la stratégie d'exploration est souvent définie par les stratégies coopératives que l'on va discuter dans la section suivante.

3.2 Stratégies coopératives pour l'exploration

La combinaison de différentes méthodes, leur coopération peut permettre de trouver un équilibre entre les avantages et les inconvénients de chaque méthode associée. Selon No Free Lunch Theorems [WM97], il n'existe pas une meilleure méthode d'optimisation. Leur effi-

cacité dépend du problème étudié et du contexte de l'étude. En comparaison avec les deux autres techniques étudiées (la parallélisation des algorithmes et la construction d'algorithmes hybrides) qui posent la difficulté de mettre en œuvre et d'analyser leur convergence, l'exploration en utilisant la stratégie coopérative fournit quelques avantages :

— la préservation des caractéristiques de recherche déjà montrées ;
— l'accélération de la vitesse de convergence par l'échange des connaissances ;
— l'accélération de la vitesse de calcul par le calcul distribué sur les environnements de calcul intensif.

Dans la partie suivante, on fait l'étude sur cette notion, la méthodologie et son application jusqu'à maintenant.

3.2.1 Définition des stratégies coopératives

La conception de stratégies coopératives vient de plusieurs travaux qui veulent combiner les métaheuristiques pour résoudre un problème complexe [BHPT98, TCS99, CT08]. Une stratégie coopérative générale est, selon la définition dans [CT08], un ensemble de programmes très autonomes, mettant en œuvre sur chacun d'eux un algorithme d'exploration particulier, et un système de coopération qui associe ces programmes en une seule stratégie de résolution pour un problème donné. Ainsi l'exploration coopérative, selon [TCS99], est un ensemble de méthodes de recherche parallèles qui combinent plusieurs programmes individuels dans un système d'exploration unique. Précisément, le paradigme agent contient des caractéristiques prometteuses pour définir un processus de coopération. Une définition dans [Tal02] utilise le paradigme agent pour formuler une définition plus concise : « Co-evolutionary hybridization represents cooperative optimization models, in which we have many parallel cooperating agents, each agent carries out a search in a solution space. »qui est traduit : l'exploration coopérative est un modèle dans lequel on intègre de nombreux agents de coopération parallèles, où chaque agent effectue une recherche dans un espace de solution.

La coopération n'est pas simplement la décomposition d'un exécuteur en processus concurrent. Elle se compose d'un ensemble de solveurs *"standalones"* qui fonctionnent indépendamment avec une coopération explicite : chacun des solveurs (avec algorithme intégré) est capable d'exécuter une recherche de façon réactive et interagit avec les autres. D'une stratégie coopérative émerge plusieurs caractéristiques que l'on analyse dans la partie suivante.

3.2.2 Taxinomie de stratégies coopératives

Une stratégie coopérative d'exploration se situe dans la catégorie d'hybridation à haut niveau en mode co-évolutionnaire (cf l'énumération 3 dans la sous-section 3.1.1). Dans ce cas, les solveurs peuvent être hétérogènes, c'est pourquoi une stratégie coopérative peut être également considérée comme une hybridation des algorithmes de haut niveau, en mode relais ou co-évolutionnaire.

Différents classements des stratégies coopératives sont proposés dans la littérature [BHPT98, TCS99, CT08]. Il en ressort trois grandes catégories : (i) échanger des composants entre les algorithmes, qui est similaire à la l'hybridation bas niveau, (ii) une équipe de solveurs autonomes collaborent, (iii) l'intégration des méta-heuristiques et méthodes systématiques. Dans ce travail, nous considérons une taxinomie basée sur ces travaux.

Ainsi, les algorithmes d'exploration coopératifs présentent les caractéristiques suivantes :

1. *hybride séquentiel ou hybride parallèle :* Dans l'hybride séquentiel, plusieurs algorithmes sont utilisés les uns après les autres. Les résultats d'un algorithme d'exploration deviennent l'entrée de l'algorithme suivant au travers d'un pipeline. Les stratégies parallèles font appel à plusieurs algorithmes (de même ou différents types) qui réalisent l'exploration concurremment. Ces concurrences partagent les solutions trouvées (jeux de paramètres) selon une stratégie prédéfinie, par exemple après quelques itérations.
2. *hybride homogène ou hybride hétérogène :* Dans l'hybride homogène, on applique plusieurs instances d'un seul algorithme métaheuristique. Par exemple le modèle d'îlot de l'algorithme évolutionnaire. Cependant, il est possible de varier les différents paramètres pour les instances. Dans l'hybride hétérogène, différentes métaheuristiques sont utilisés. L'algorithme mémétique associant une recherche globale avec une recherche local en est un exemple.
3. *coopération synchrone ou coopération asynchrone* : La coopération synchrone est considérée lorsque le partage des connaissances est réalisé à un point d'état fixé entre tous les participants en vue d'améliorer l'efficacité de la recherche. Les solutions sont partagées globalement et la décision est réalisée de façon collective. Au contraire, la coopération asynchrone laisse à la discretion de chaque solution, le choix du moment et des connaissances à partager. Par rapport à des stratégies asynchrones, la coopération synchrone nécessite plus de calcul et semble moins réactive. Par contre, une stratégie asynchrone peut conduire à une convergence prématurée.
4. *coopération globale ou coopération locale* : Dans l'hybride globale, tous les algorithmes explorent l'ensemble de l'espace de recherche. C'est une décomposition de l'espace implicite. Par contre, chaque individu est dirigé par une mission d'exploration différente. Chacun partage et tient compte de toutes les solutions partagées. Dans l'hybride partiel ou la décomposition explicite, l'espace de solution est décomposé en sous-problèmes. Ainsi, chaque individu se dédie à explorer un sous espace (quelques dimensions, par exemple), et partage les résultats pour construire une solution globale. La décomposition est choisie statiquement avant la recherche ou dynamiquement pendant la recherche. Par conséquent, les individus communiquent afin de recevoir leur espace de recherche ou respecter celui des autres
5. *L'échange de solutions ou l'échange de l'espace de solutions* : La communication entre les solveurs peut se faire pour des solutions précises ou l'espace de solutions. Dans le permier cas, les algorithmes partagent les solutions qu'ils considèrent utiles pour la recherche globale. Dans l'échange de l'espace de solutions, un algorithme partage les contraintes qui définissent ensemble un espace de recherche pour les autres.

Nous avons donné de notre point de vue les différents modes de comportement d'une coopération. Il existe des travaux qui implémentent, en prenant en compte ces caractéristiques d'une plate-forme générique afin de faciliter la conception d'une stratégie coopérative. Nous les présentons dans la suite.

3.2.3 Etat de l'art de la structuration de l'exploration coopérative

Plusieurs travaux tentent de concevoir leur plate-forme pour combiner différents algorithmes de différentes classes de recherche. Ces travaux inspirent la conception d'agent coopératif pour concevoir leur plate-forme.

COSEARCH : COSEARCH [BT00] est une méthode co-évolutionniste et à la fois une plate-forme permettant de combiner les algorithmes définis pour la recherche coopérative

[TB06b]. Elle utilise trois agents ayant des rôles bien définis et complémentaires : un agent de recherche (Searching Agent, SA), un diversificateur (Diversifying Agent, DA) et un intensificateur (Intensifying Agent, IA). Ces trois agents évoluent en parallèle et coopèrent via une mémoire centrale et partagée (Adaptive Memory, AM). Ils échangent de l'information par cette mémoire centrale qui s'appelle la mémoire adaptative commune. Pendant la recherche, la mémoire adaptative regroupe l'ensemble des connaissances acquises par les trois agents. L'agent SA utilise un mécanisme d'amélioration conditionnelle, par exemple la recherche tabou, pour améliorer la solution initiale au début du processus. L'agent DA génère de nouvelles solutions sur des régions inexplorées de l'espace de recherche à partir des régions déjà visitées en utilisant l'opérateur génétique de croisement. L'agent IA génère de nouvelles solutions à partir des régions prometteuses déjà visitées en utilisant la recherche locale et ainsi définir l'ensemble des points de départ pour l'agent SA.

L'avantage de cette approche est la balance entre l'intensification et la diversification de l'exploration. La version séquentielle donne de bons résultats cependant elle n'est pas optimisée en terme de temps de calcul. Afin de réduire ces temps de calcul les auteurs ont alors proposé une version parallèle [TB06b] qui lance plusieurs agents SA en même temps via la technologie multi-threading présente sur les systèmes de calcul multi-cœurs. L'utilisation d'une approche parallèle implique des traitements particuliers pour la gestion de la mémoire centrale.

MAGMA (Multi-Agent Meta-heuristic Architecture) MAGMA [MR04] est une architecture hiérarchique d'un système multi-agents. Un système se compose de plusieurs niveaux d'agents dont l'objectif est de réaliser une tâche particulière du système. Plusieurs agents peuvent participer à un algorithme à chaque niveau. Par exemple, le niveau 0 fournit la capacité à construire une solution de l'échantillon. Le niveau 1 permet de se déplacer sur l'espace de recherche pour améliorer la qualité de cette solution. Le niveau 2 contrôle la balance entre l'intensification et la diversification. Étant donnée qu'il y a plusieurs agents au niveau 2, les stratégies de coordination du niveau 3 vont assurer la coopération entre différents agents métaheuristiques. De cette façon, pour construire un algorithme métaheuristique, on va diviser sa construction en plusieurs composants, puis agentifier certaines fonctions de l'algorithme et les organiser. D'autre part, de nouveaux algorithmes coopératifs des métaheuristiques peuvent facilement être conçus par combinaison des agents et en définissant leurs interactions. Malheureusement, cette approche ne propose qu'une distribution fonctionnelle et ne prend pas en compte la distribution de l'exécution.

AMF (Agent metaheuristic framework) : les auteurs de l'AMF [MCK08b] proposent une plate-forme basée sur un modèle organisationnel orienté agent qui décrit des métaheuristiques. Un métaheuristique se construit en appliquant les trois concepts suivants : le rôle, l'interaction et de l'organisation du méta-modèle RIO [MRS03]. Un agent métaheuristique est contrôlé par un agent qui joue un rôle spécifique dans cette organisation. Ces rôles spécifiques, correspondent aux composants principaux ou aux opérateurs d'une métaheuristique habituelle. Il s'agit de : (i) l'intensificateur (effectue une recherche locale), (ii) le diversificateur (cherche de nouvelles régions prometteuses), (iii) le guide de l'adaptation ou de l'auto-adaptation (qui structure un mécanisme d'interaction et d'apprentissage). L'AMF fournit une méthodologie plus complète que MAGMA pour développer des métaheuristiques : analyse, modélisation et implémentation. Cette approche prend en compte la distribution de l'exécution par le paradigme agent.

Malheureusement, ni exemples expérimentaux, ni détails concernant sa mise en œuvre ne sont proposés. Son implémentation [MCK08a] (A Coalition-Based Metaheuristic for the Vehicle

Routing Problem, CBM) est réduite au domaine de l'optimisation combinatoire pour la planification des véhicules. De plus, la coopération dans ce cas est de regrouper un grand nombre d'opérateurs simples (intensification et diversification) pour qu'ils soient automatiquement combinés via un agent qui intégrè un métaheuristique. Ensuite la coopération est faite entre l'ensemble homogène de cet agent métaheuristique. L'exécution des différents opérateurs est coordonnée par un mécanisme de décision permettant de choisir l'opérateur à appliquer en fonction du contexte d'optimisation. Dans ce cas, un agent se compose de plusieurs opérateurs. Ces opérateurs ne s'exécutent pas en parallèle, ils s'exécutent selon un plan ordonné, adapté. L'approche ne prend pas en compte le parallélisme dans l'exploration.

DAFO (Distributed Agent Framework pour Optimisation) : DAFO [DBB10] est une plate-forme multi-agent organisationnelle qui s'inspire de l'idée de coopération entre sous-populations en interaction. Le point d'intérêt de DAFO est que elle ne se concentre pas sur l'amélioration de l'échange des informations entre les sous-populations mais elle fournit une abstraction fonctionnelle pour organiser deux modes de coopération : le mode coopératif avec interaction synchrone et le mode compétitif avec interaction asynchrone. Ce travail fournit également le méta-modèle MAS4EVO (Multiagent system for evolutionary optimisation). Il présente trois types d'agents : les agents *"solveur"* qui font l'exploration sur l'espace de recherche, les agents *"fabrique"* qui sont responsables d'instancier et de configurer l'exécution de l'application, et les agents *"observateur"* qui génèrent les sorties pour l'utilisateur. Le méta-modèle MAS4EVO et la plate-forme DAFO facilitent la construction, la compréhension et la manipulation de la structure de algorithme génétique co-évolutionniaire. Cependant, ils ne résolvent que les problèmes de décomposition co-évolutionaire [Par96] : dans ces problèmes, pour évaluer la qualité d'une solution, il faut utiliser un ensemble de fonctions d'évaluations de la solution. Cette approche envisage la co-évolution pour combiner les résultats obtenues de ces fonctions pour évaluer la solution globale. En outre, la distribution de l'exécution est au niveau fonctionnel ce qui laisse son contrôle au mécanisme de multi-theading de la plate-forme dans lequel il est implémenté.

Le tableau suivant 3.1 résume les critères des différentes plate-formes.

Dans la section suivante, on va étudier les plate-formes qui permettent de passer à l'échelle les calculs intensifs.

3.3 Exploration large échelle par le calcul intensif

3.3.1 Etude de l'environnement de calcul intensif

Il y a plusieurs types d'environnements de calcul intensif tels que : les clusters, les grilles de calcul, les réseaux de stations de travail, le réseau pair-à-pair d'ordinateurs, la grille volontaire sur Internet ou encore le cloud. Cependant, dans cette partie, nous nous intéresserons aux clusters et aux grilles de calcul qui sont plus souvent disponibles dans les environnements académiques.

Cluster

Un cluster est un groupe d'ordinateurs homogènes indépendants appelés nœuds, localisés et organisés en grappe [Bak00]. Chaque cluster utilise généralement des certaines de processeurs

	Agent	Organisation	Interaction
COSEARCH	Un agent = un composant fonctionnel spécifique et unique. (Recherche, Diversification, Intensification)	Les 3 agents (Recherche, Diversification, Intensification) participent et ont des rôles définis	Les agents échangent des informations via une mémoire adaptative commune
MAGMA	Un agent = une fonction, un opérateur de l'algorithme méta-heuristique ou un algorithme entier.	Définit la hiérarchie d'agents en plusieurs niveaux. Plusieurs agents de niveaux différents constituent un algorithme (aussi un agent)	Niveau 0 : construire une solution de l'échantillon. Niveau 1 : se déplacer sur l'espace de recherche, une solution. Niveau 2 : contrôler la balance entre l'intensification et la diversification. Niveau 3 : assurer la combinaison avec les autres agents.
AMF	Un agent = joue un rôle spécifique entre : Stratège, Guide, Intensifieur et Diversifieur	Les agent coopérent dans une coalition. Une coalition se compose d'un ensemble d'agents (chaque agent contient les autres agents jouants un des rôles : Stratège, Guide, Intensifieur et Diversifieur)	Les agents s'informent des zones prometteuses de l'espace de solution. Le rôle Stratège permet de choisir l'opérateur à appliquer en fonction du contexte d'exploration.
DAFO	Un agent = joue un rôle spécifique de trois types : solveur, fabrique ou observateur.	Les agents participent à la constitution prédéfinie avec les rôles solveur, fabrique et observateur se conformant au méta-modèle MAS4EVO	Le mode coopératif avec interaction synchrone et le mode compétitif avec interaction asynchrone.

Table 3.1 – Comparaison des plates-formes de structuration de l'exploration coopérative

connectés via un réseau à faible latence et une connexion à très haut débit. L'avantage est de créer un calculateur comme un seul ordinateur ayant plus de performances (puissance du processeur, taille de l'espace de stockage, quantité de mémoire vive, etc.). A titre informatif, un classement des clusters les plus puissants au monde est proposé à l'adresse suivante `http://top500.org`.

Les programmes s'exécutent sur les clusters soit en mode séquentiel soit en mode parallèle. En mode séquentiel le traitement par lots (ou mode batch) : un très grand nombre de jobs est envoyé dans une file d'attente globale pour lancer des jobs de manière asynchrone. De cette manière nous n'avons pas besoin d'attendre la fin de l'exécution du calcul. Cependant, de cette façon, le programme n'est pas découpé en diverses tâches exécutées sur différents processus répartis. En mode parallèle, le programme utilise une API (Application Programming Interfaces) standard telle que Message Passing Interface (MPI) [CGH94] ou OpenMP [DM98]. MPI assure la communication et la synchronisation des processus répartis qui s'exé-

cutent sur les différents nœuds pendant que l'OpenMP exécute le programme sur plusieurs cœurs d'un même nœud avec une mémoire partagée.

Grille de calcul

Une grille [KF99] de calcul est un ensemble de ressources informatiques hétérogènes (ordinateurs, serveurs, clusters, ...), organisés en groupe, dont l'objectif est de mettre à disposition des scientifiques une puissance de calcul distribuée. Les grilles de calculs sont délocalisées dans la mesure où les nœuds d'un même groupe sont souvent physiquement positionnés dans plusieurs lieux géographiques. Les ressources sont hétérogènes dans leurs caractéristiques techniques, leur système d'exploitation, leur système de gestion et de soumission des jobs. Il n'y a pas une unité commune avec un moniteur de contrôle (comme ce que est fait pour le cluster) qui contrôle les ressources alors que l'accessibilité aux ressources est forcement dépendante du pare-feu, de la planification, de la politique inter-organisationnelle. En conclusion, l'apparition de la grille de calcul est une manière de faciliter l'accès à des ressources informatiques partagés à l'échelle mondiale pour effectuer des milliers calculs intensifs et de manière asynchrone et pour différents usages applicatifs plutôt que d'augmenter la vitesse d'une application à l'intérieure d'une organisation.

Dans le livre [AT04] de Mike Ault, il résume la différence entre le cluster et le grille de calcul dans le tableau 3.2 :

Characréristique	Cluster	Grille de calcul
Population	ordinateurs standards (commodity computers)	ordinateurs standards et ordinateurs high-end
Propriété	unique	multiple
Distribution des utilisateurs	centralisé	décentralisé
Distribution de ressource	centralisé	distribué
Scheduling	centralisé	décentralisé
Interopérabilité	propriétaire et standardisée	pas de normes en cours d'élaboration
Scalabilité	moyenne	large
Capacité	garantie	varié, mais haute
Débit	moyenne	élevé
Vitesse/Bande passante	rapide/faible	lente/élevé

Table 3.2 – Comparaison entre le cluster et la grille de calcul

Enjeux du développement en calcul intensif

Pour qu'une application puisse profiter des ressources de calcul haute performance, il faut prendre en compte les caractéristiques du système tels que : l'extensibilité, la tolérance aux pannes, l'hétérogénéité et l'ouverture. Le besoin d'utiliser un système distribué pour le développement est souvent dérivé de ces exigences non fonctionnelles [BFH03] :

1. *Extensibilité*, un grand nombre d'unités de calcul (cœurs, processeurs) sont mis à disposition. Les systèmes distribués permettent une expansion aisée (à la demande), en ajoutant dynamiquement des ressources de calcul (processeurs, nouveau site de calcul)

2. *Tolérance aux pannes*, Un noeud peut potentiellement être en panne ou en retard. Les systèmes distribués peuvent être plus tolérants aux pannes car ils permettent de répliquer facilement les composants.
3. *Hétérogénéité*, Chaque nœud a des exigences et des politiques de sécurité différentes. Ils ont aussi des politiques différentes de gestion des ressources. Leur puissance de calcul et leur mémoire sont différentes. Ils sont connectés par un réseau multi-niveaux et hétérogène. Ils sont susceptibles d'être géographiquement séparés.
4. *Ouverture*, ces systèmes fournissent un moyen de partager les ressources. Il s'agit à la fois le matériel mais aussi du logiciel et des données. Les composantes des systèmes possèdent des interfaces bien définies pour assurer l'extensibilité et le fait de pouvoir les modifier facilement, par exemple : Utiliser les services web possède une grande ouverture.

Toutes ces caractéristiques doivent être gérées par le logiciel de gestion d'un système de calcul tandis que seulement certaines d'entre elles devraient être prises en compte au niveau de l'application. Par exemple : pour profiter de l'extensibilité, le développement doit prendre en compte et détecter les nouvelles unités de calcul (processeurs) qui viennent d'être ajoutées pour accélérer le programme. D'autre part, si une exécution n'est pas réussie, il faut que le programme ait la capacité de faire un remplacement pour continuer le calcul.

3.3.2 Plate-forme pour la simulation large échelle de modèles complexes

L'objectif de cette partie est de fournir une étude des plate-formes qui fournissent des gestionnaires et des APIs pour diminuer la complexité d'exécuter les applications sur les unités de calcul (processeurs, cœurs). Il y a plusieurs niveaux d'abstraction logicielle pour faciliter l'utilisation des ressources de calcul de la grille informatique (figure 3.9). Malgré leur recouvrement, il est possible de les classer en trois niveaux :

— **Niveau infrastructure**, qui intégrè un système de gestion de charge (workload management systems, WMS) pour fournir un ensemble d'API et de lignes de commandes permettant de transferer et de lancer des jobs sur une grille informatique spécifique ; Pour la gestion d'un système de batch dans cluster, il y a les systèmes de gestion et de planification des ressources de calcul distribué (DRMS) qui sont développés et gérés par différentes organisations et entreprises tels que : PBS/Torque [EGGA$^+$04], Condor [TTL05] ou Sun Grid Engine (SGE) [Gen01]. Alternativement, pour la gestion et la planification des jobs dans une grille de calcul, il existe des Middlewares (workload management system, WMS et data management system, DMS) tels que globus [FK97] ou gLite [LEP$^+$06]. Tous ces logiciels fournissent un ensemble de primitives et de services pour soumettre les jobs et pour transférer les données entre l'utilisateur et les ressources de calculs. Ces logiciels dissimulent la couche d'infrastructure d'une grille informatique en proposant un langage de script dédié à la description des jobs et sous forme de lignes de commande UNIX. En utilisant ces fonctionnements, les applications peuvent s'exécuter sur les grilles informatiques. Un fonctionnement commun des systèmes est de permettre et de soumettre les jobs en mode batch. Il y a des différences entre eux comme : la possibilité de lancer les applications parallèles sur un cluster, la possibilité d'accéder au système d'exploitation virtuel dans la grille de calcul. Ces systèmes partagent de nombreux concepts, mais la description des commandes et les API fournies restent spécifiques. Une application développée se basant sur un système (comme Globus) ne fonctionne pas sur les autres systèmes (tel que SGE). Récemment, il y a aussi des middlewares plus génériques qui gèrent toutes les

DRMS précédemment mentionnées pour bénéficier facilement de la puissance de calcul distribué tout en restant indépendant de l'architecture sous-jacente tels que : GridWay Metaschedules (`http://www.gridway.org/`), DIRAC (`http://diracgrid.org`).

— **Niveau d'abstraction** qui fournit un niveau logiciel comportant un ensemble d'APIs qui uniformise l'utilisation des WMS ;
Les outils de la couche supérieure simplifient l'accès à un environnement de calcul distribué. Mais ils ne cachent pas l'hétérogénéité matérielle et logicielles des ressources informatiques. Leurs commandes sont spécifiques à leur propriétaire. C'est mieux qu'il y ait une spécification standard à suivre pour que l'on puisse configurer les applications une fois et les exécuter plusieurs fois sur plusieurs environnements. En outre, un niveau d'abstraction plus haut est nécessaire pour fournir les fonctionnalités au développeur des applications plutôt que les fonctionnalités de l'environnement de calcul. Une couche d'abstraction générique bien conçue pour les ressources de calcul devrait permettre l'utilisation de plusieurs ressources de calcul (comme cluster, les grilles de calcul, grille volontaire et cloud) à travers une même interface de haut niveau. DIRAC et GridWay se dédient à cela. Cependant, plusieurs étapes restent nécessaires pour soumettre un job :

1. obtenir un certificat ;
2. écrire un fichier de description des jobs (SGE, JDL, etc) avec les ressources nécessaires ;
3. créer un script pour exécuter l'application sur le nœud worker ; et
4. suivre l'avancement des jobs pour récupérer les résultats.

A cette couche, il y a plusieurs librairies qui fournissent une interface APIs pour connecter avec des couches d'infrastructure telles que : DRMAA (Distributed Resource Management Application API) [HHML07], Ganga [Mai08] ou JSAGA [Rey10]. Ce sont aussi des spécifications standardisées à la communauté.

— **Niveau du domaine spécifique** qui utilise les APIs fournis par les logiciels de deuxième niveau pour concevoir un runtime virtualisé et des fonctionnements pour les objectifs scientifiques de differents domaines ou d'un domaine pluridisciplinaire. Ce point est abordé plus en détail dans la suite de la section.

Le niveau du domaine spécifique est le niveau où on développe une application dédiée au métier, par exemple, pour but de l'expérimentation, fouille des données, l'exploration ou l'optimisation pour les utilisateurs de différents domaines. Dans la couche d'abstraction précédente, DRMAA et Ganga ne fournissent que de fonctionnalités proches de l'infrastructure. La plate-forme JSAGA fournit les fonctionnalités à un autre niveau mais n'essaie pas de renforcer la compatibilité avec tous les DRMS et middleware possibles. Autrement dit, elle conserve juste les fonctionnalités qu'elle considère utile pour les développeurs d'applications à base de grille. Dans la couche d'application, on déploie les logiciels qui fournissent un environnement d'exécution virtualisé pour que des algorithmes scientifiques puissent être déployés indépendamment de l'architecture de l'environnement de calcul.

Kepler : Kepler [ABE+11] est une application en Java conçu pour aider les scientifiques, les analystes et les développeurs à créer, exécuter et partager les modèles dans de nombreux domaines scientifiques. Kepler fournit une interface graphique pour sélectionner puis raccorder des composants analytiques pertinents et les sources de données pour créer un workflow scientifique (Scientific WorkFlow – SWF constitue une représentation exécutable en formalisant les étapes nécessaires d'un processus et, ensuite, en définissant l'interaction entre les étapes pour produire des résultats). Le logiciel permet aux utilisateurs de réutiliser et partager des données, des workflows, et des composants développés par la communauté scientifique pour répondre aux besoins communs : (1) Kepler peut fonctionner sur des données stockées dans

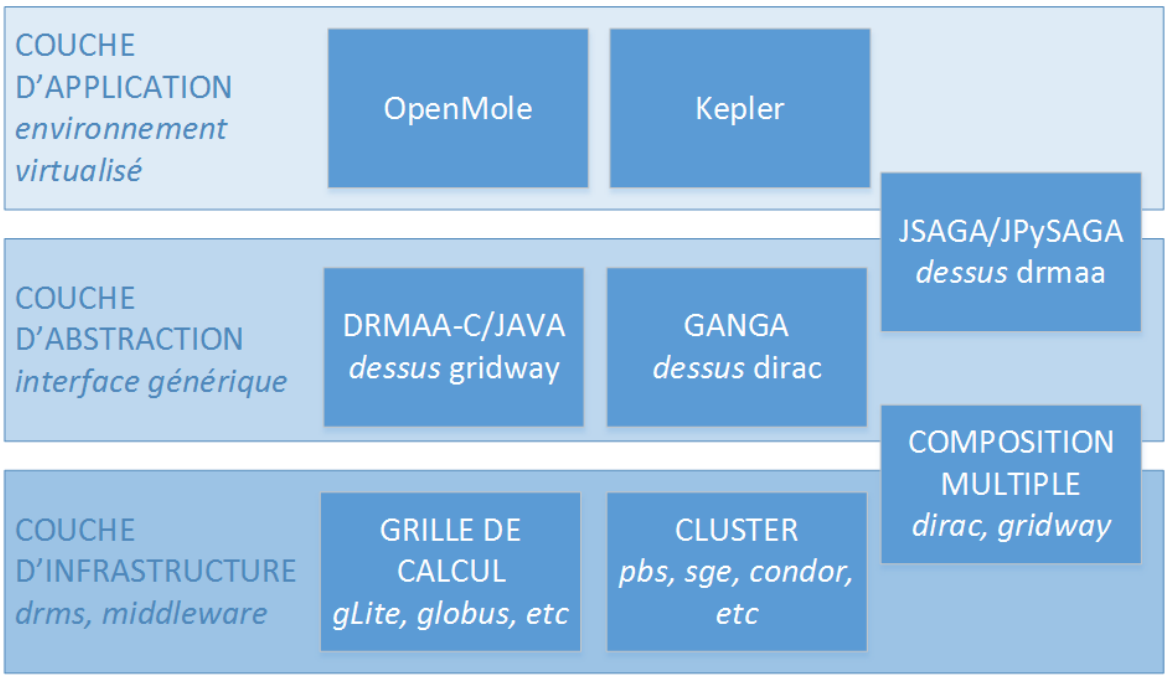

Figure 3.9 – Les abstractions, au dessous d'une application, à la base de la grille informatique

une variété de formats (en fichier, en base de données), localement et sur Internet. (2) Il fournit un environnement efficace pour l'intégration de composants logiciels disparates externes, tels que les scripts R avec le code compilé C (ou via Java Native Interface, JNI), Matlab, Python (3). Il facilite les exécutions distribuées de modèles en permettant d'intégrer les nouveaux modules comme plugin, par exemple le module d'importation automatique des webservices trouvés sur une page web ou les modules d'accès aux DRMS ou middlewares via l'interface ad-hoc SSH.

Kepler, le scénario d'utilisation est que d'extraire des données à partir d'un instrument scientifique, de passer à travers un logiciel d'analyse fonctionnant sur un système de calcul haute performance, de stocker les résultats dans un stockage distribué de données, et ensuite le visualiser sur un grand affichage. Dans ce modèle, les composants prennent typiquement l'entrée et produisent une sortie comme une partie d'un pipeline. Le système de workflow planifie les calculs sur la ressource la plus appropriée seulement lorsque les entrées sont disponibles. Lorsque la sortie est produite, elle est transmise à l'autre calcul dans le pipeline. Kepler hérite du paradigme de modélisation orientée acteurs avec la capacité de planifier et d'exécuter des composants : l'accès aux données à distance et l'accès aux méta-données, les transformations de données, l'analyse des données, l'interface avec les applications externes, l'invocation et le déploiement de services Web, et le suivi de la provenance. Il y deux catégories de wokflows [DGST09] : (i) le flux de contrôle ou (ii) le flux de donnée.

Pour le flux de contrôle, le contrôle de la tâche précédente est délégué à la tâche suivante. Cela inclut les structures de contrôle telles que les séquences, les conditions et les itérations. Pour le flux de données, les dépendances représentent le flux de données du producteur au consommateur. En outre, Kepler supporte le temps continu pour représenter la valeur d'un signal en temps continu et les événements discrets pour faire la communication entre composants via une file d'attente des événements. Ainsi, Kepler permet une imbriquation des modèles et la distribution de leur exécution.

Les workflows scientifiques de Kepler peuvent fonctionner sur plusieurs niveaux. Par exemple,

au niveau bas, un flux de données transfère des données entre les éléments. Au niveau haut, un flux conceptuel de contrôle exécute des étapes complexes d'analyse de données pour un domaine métier. Au niveau d'exécution, Kepler permet aussi d'exécuter le workflow en parallèle sur le système de calcul intensif en déléguant l'exécution d'un workflow sur les threads indépendants.

Il y a une extension appelé Nimrod/K [ABE+09b] qui utilise le workflow de Kepler pour l'exploration des paramètres des systèmes complexes. Nimrod/K engendre un large nombre de threads parallèles dynamiquement sans programmation supplémentaire. Les utilisateurs écrivent de façon simple un workflow, et le système sous-jacent est responsable de le reproduire lors de l'exécution selon les besoins. Cette approche permet d'intégrer un simulateur dans un workflow. Après ça, on va définir les paramètres d'entrée et de sortie dans ce workflow. La valeur de ces paramètres est changeable indépendamment. De cette façon, Kepler découpe le workflow en un ensemble de modules de même simulateur avec différentes valeurs de paramètres et les calcule en parallèle.

Bien qu'un workflow puisse être exécuté en parallèle, Kepler fixe l'opérateur de composition des données dès le début à l'intérieur d'un seul élément du workflow : il ne peut pas être explicitement spécifié par l'utilisateur. Cela signifie que juste une seule stratégie d'exploration prédéfinie et fixée est implémentée dans un workflow. Les éléments de Kepler sont bloqués lors de la lecture des données d'entrée. Cela rend difficile la définition d'une stratégie d'exploration hybride dans un workflow quand on veut transférer des données lors de l'exécution.

OpenMOLE : [RCL+10] fournit un environnement d'exécution virtualisé en tirant partie de l'interface générique de JSAGA afin de développer des algorithmes scientifiques distribués tout en restant indépendamment de l'architecture de l'environnement d'exécution. La délégation transparente des algorithmes scientifiques à l'environnement à distance devient possible avec OpenMOLE grâce à la virtualisation. Avec OpenMole, (1) les utilisateurs peuvent définir et intégrer leurs composants logiciels exécutables externes. L'utilisateur peut travailler sur ses propres algorithmes, les exécuter directement sans se soucier de la manière de déployer ou d'exécuter les différents environnements de calcul en dépit de l'hétérogénéité matérielle et logicielle ; (2) l'accès via OpenMOLE au calcul distribué est autonome pour différents environnements d'exécution à distance, sans nécessiter de serveurs intermédiaires ou de fournisseurs d'adresse IP publiques ce qui assure une scalabilité et une convivialité de l'application.

Afin de résoudre deux problèmes : la portabilité des composants logiciels des utilisateurs sur les environnements d'exécution hétérogènes et l'accès générique indépendamment des environnements d'exécution différents, OpenMOLE fournit des services génériques implémentées dans la classe « Task ». Une « Task » est exécutée dans un « Context » spécifique et utilise des « Ressouces » indiquées. De cette façon, la plate-forme OpenMOLE est basée sur le formalisme de workflow. Ce formalisme est très approprié pour la représentation de processus parallèles et permet de migrer et d'exécuter des algorithmes à distance. La classe « Task » est polymorphe. Chaque conception de « Task » est un nouveau service pour réaliser une fonctionnalité. Une « Task » embarque des « Ressouces » tels que les fichiers, les bibliothèques ou les logiciels. Ces « Ressouces » doivent être disponibles lors de son exécution. Le « Context » contient les variables qu'une « Task » utilise lors de l'exécution. Chaque « Context » correspond à des conditions d'exécution particulières. Les « Tasks » peuvent être déplacées et portables entre les environnements d'exécution hétérogènes. Pour déléguer l'exécution du méta-modèle « Task/Ressources/Context », OpenMOLE applique une approche de cloud : il expose les environnements d'exécution à distance comme s'ils étaient les services qu'offre un ordinateur. Pour faire cela, le workload est délégué directement à partir de l'ordinateur de l'utilisateur sur les environnements d'exécution à distance : les composants

logiciels nécessaires sont installés de manière transparente et à la volée. De plus, les variables constituent le workflow, qui peut être échangé entre les « Tasks » via les « Transitions ». En workflow, une « Tasks » est un composant de l'exécution atomique. Plusieurs « Tasks » sont liées les unes avec les autres par des « Transitions » formant un workflow. De cette façon, les « Tasks » sont indépendantes et peuvent être exécutées simultanément. Cela signifie que les « Tasks » ont été conçues de sorte qu'elles n'ont pas d'effet secondaire parasite. Par conséquent, elles peuvent être envoyées en toute sécurité sur plusieurs threads, processus ou ordinateurs.

OpenMOLE fournit un langage dédié permettant la description de plans d'expériences réutilisables afin de résoudre les problèmes inverses portant sur les modèles de simulation de système complexe. OpenMOLE se focalise sur l'expérimentation en exécutant les simulations de manière distribuée. Il supporte de l'intégration des simulateurs de système complexe comme un composant logiciel.

A partir de ces deux cas d'étude, nous voulons montrer les caractéristiques de Kepler et OpenMole pour l'exploration des simulateurs complexes. On présente aussi les points forts et points faibles de ces deux environnements.

— Pour les points forts, la conception d'un formalisme de workflow dans OpenMOLE et Kepler est très souple et cruciale. Elle permet de déléguer l'exécution des tâches entièrement transparente. OpenMole fournit des structures de traitement sur ce principe : cycles (boucles), le branchement conditionnel, les workflows imbriqués et la représentation implicite de workflow massivement parallèle. Ces fonctionnalités permettent à OpenMole et à Kepler une hybridation entre un flux de donnée et un flux de contrôle. Cela signifie également que le simulateur, les données d'entrée, les données de sortie sont imdépendantes et modulaires.
— Ces deux plates-formes Kepler et OpenMole permettent d'utiliser les appels externes à des composants logiciels extensibles. De cette façon, les simulateurs complexes et les algorithmes d'exploration sont portables. Cela assure la reproductibilité et la réutilisation d'une méthode de l'exploration pour différents simulateurs ou celles de différentes méthodes de l'exploration pour un simulateur [DGST09].
— Pour l'exploration de simulateurs complexes, OpenMole est plus adapté que Kepler étant donné qu'il fournit une description indépendante des composants nécessaires pour l'échantillonnage des entrées d'un modèle et l'exploration distribué de l'espace des paramètres. Cependant, tous les deux contiennent des inconvénients similaires : la stratégie d'exploration et d'hybridation entre différents algorithmes est implémentée de manière fixe. Nous ne voyons pas la modularité pour l'implémentation et l'intégration des différentes algorithmes d'exploration dans une stratégie d'exploration : c'est difficile à changer une stratégie d'exploration en réutilisant les modules d'algorithmes déjà implantés [ABE+09b, ABE+09a]. Changer ce type de workflow en ajoutant la capacité d'intégration de différents modules d'algorithme d'exploration pour l'exploration collaborative de simulateur complexe est ce que nous faisons dans cette thèse.

3.3.3 Exigences de passage à l'échelle de l'exploration collaborative

Pour le passage à l'échelle de l'exploration collaborative, on doit chercher des solutions pour résoudre les verrous scientifiques liés à la distribution des calculs, et plus exactement, pour réduire les temps de simulation, améliorer la gestion de la concurrence et la tolérance au pannes.

Le temps de simulation n'est pas identique et la tolérance aux pannes des processeurs décident comment traiter des réponses de la calcul. Dans le modèle en ilot, étant donné que les ressources sont potentiellement défectueuses et de nouvelles ressources peuvent être agrégées dans le système, les topologies régulières de communication entre les ilots sont difficiles à mettre en œuvre parce qu'il faut tenir compte d'une topologie devant être reconfigurée dynamiquement lors de l'exécution. Il est possible de mettre en œuvre une mécanique de migration individuelle entre les ilots sur le cluster en utilisant des messages MPI. Mais cela est difficile voire impossible sur un système constitué de milliers de nœuds distants comme la grille de calcul à cause de la dynamique des nœuds de calcul et du temps d'échange des données entre les nœuds à distant : la topologie en anneau unidirectionnel ne fonctionne pas bien dans un contexte distribué.

Dans le modèle de diffusion, le coût entre le temps de calcul et la communication peuvent être défavorables. On a besoin d'une grande vitesse de connexion pour la transmission d'informations entre les voisinages organisés en grille. Heureusement, il reste un modèle comme le modèle maître/esclave qui offre plusieurs avantages. Tout d'abord, le modèle est conceptuellement simple : le maître envoie de manière itérative et asynchrone des tâches impliquant l'évaluation des individus aux esclaves. Après évaluation, les résultats sont envoyés au maître qui détermine de nouveaux individus sans attendre l'évaluation de tous les individus.

La mise en œuvre réussie des évaluations basées sur le modèle maître/esclave nécessite de faire face aux problèmes suivants : (1) les erreurs dans le maître ; (2) les erreurs d'un esclave ; (3) l'algorithme doit réagir avec de nouveaux processeurs dynamiquement ajoutés aux systèmes afin de les utiliser le plus rapidement possible ; (4) temps de réponse différents des esclaves (en cause de la puissance différente des processeurs et les cœurs de calcul, ou le retard du réseau) ; (5) le goulot d'étranglement chez le maître lorsqu'il reçoit les résultats des calculs des nombreux jeux de paramètres.

En fait, l'algorithme doit prendre en compte un blocage dans la partie maître : quand l'exécution d'esclave a échoué, il peut demander une nouvelle évaluation de l'individu à un autre esclave ou simplement ignorer cette évaluation. Si il y a les unités de calcul qui sont dynamiquement intégrées, il faut fournir un mécanisme pour annoncer la disponibilité de ces nouvelles ressources. L'algorithme doit réagir aux changement des ressources. Enfin, l'ajustement du grain de calcul de l'esclave est directement dépendant de la complexité de la fonction d'évaluation qui dépend du problème à résoudre.

Discussion

L'exploration parallélisée d'un simulateur complexe exige non seulement la parallélisation d'un algorithme d'exploration existant mais encore le choix des algorithmes d'exploration pour une coopération efficace. De cette façons, différentes plate-formes comme Kepler et OpenMole ont nativement intégré le support de l'exploration de l'espace de paramètres. Ces outils fournissent aux explorateurs, des formalismes et favorisent la construction de workflows. La combinaison des composants est nécessaire pour l'expérimentation : des simulateurs externes interoperable, les outils de paramétrage des modèles avec différents types de données, les outils d'analyse graphique des sorties des modèles, et enfin des composants prédéfinis pour implémenter des stratégies d'exploration. D'autre part, l'exigence du travail demande aussi une intégration des stratégies d'exploration flexible et coopérative comme comme cela est fait dans le cadre d'une étude. Les plate-formes, que nous avons étudié, n'ont pas encore un mécanisme modulaire pour définir une stratégie d'exploration résultant de l'association

différentes algorithmes existants. Dans cette thèse nous tentons de répondre à ces limitations.

Deuxième partie

Contribution de la thèse

Chapitre 4

La solution conceptuelle de la coordination des méthodes d'optimisation pour l'exploration coopérative

Sommaire

Cet chapitre fait une analyse de l'état de l'art afin de mettre en avant les verrous scientifique compte-tenu de la contribution de cette thèse. Dans un premier temps, ce chapitre met en avant les point fort de l'état de l'art. La deuxième section du chapitre définir les contours de cette thèse, sa problématique et ses verrous scientifique compte-tenu des travaux existants. Finalement, la troisième section du chapitre vous présente, succintement, la solution que nous avons imaginée pour résoudre le problème.

4.1 Une synthèse des études de l'exploration de simulateurs complexes

Les simulateurs complexes, surtout les simulateurs à base d'agents permettent d'étudier les systèmes complexes de nature diversifiée. La réalisation de simulations reproduisant un phénomène réel vise à répondre à une problématique scientifique de terrain, par exemple en écologie, en urbanisme ou en épidémiologie. La reproduction de phénomènes réels et la prise de décision sont des questions majeures pour un modèle et un simulateur. La réponse à ces questions nécessite de mettre l'accent sur les actions suivantes : détecter les variables significatives ou trajectoires importantes ; calibrer les paramètres pour minimiser une erreur ou pour désigner les surfaces de réponses. Cela se concrétise par l'exploration de l'espace des paramètres. Cependant, plus un système complexe est proche du monde réel, plus le modélisateur fait face à une explosion du nombre de paramètres et à un manque de connaissances de leur effet sur la dynamique du modèle. On pourra aussi s'intéresser à déterminer la gamme des comportements possibles du système pour l'ensemble des paramétrages. Cela implique un calibrage automatique de grandeurs caractéristiques du système pour caractériser les phénomènes dynamiques et mesurer le comportement des simulations correspondantes.

L'utilisation des algorithmes d'exploration est une solution prometteuse pour l'exploration de l'espace de paramètres des simulateurs complexes [Sto11, FIM+13, CH+07]. Différents algorithmes d'exploration ont des comportements et des effets différents quand ils sont appliqués sur l'exploration de l'espace de paramètres, c'est pourquoi nous avons catégorisé en trois classes : la recherche locale, la recherche globale et la recherche par criblage.

Plusieurs plate-formes et méthodologies ont été conçues comme [BT00, MR04, MCK08b, NCR+02, DBB10] afin de créer un outil favorisant l'association des algorithmes d'exploration. En utilisant une plate-forme permettant l'association de différentes (méta-heuristiques) on pourrait améliorer une méthode de recherche et augmenter le niveau de généralité. De cette façon, un algorithme d'exploration est considéré comme une étape d'initialisation pour recevoir un point de commencement, ce qui génère ou reçois un ensemble des jeux de paramètres, une étape d'exécution d'un opérateur de recherche pour sortir les résultats prometteurs en forme de jeux de paramètres. Les jeux de paramètres deviennent l'objet de communication entre les constituants de la coopération. L'exécution d'un opérateur est possible d'être abstraite. Il est capable de créer un prototype défini une interface qui spécifie une série de méthodes de mettre en œuvre pour étendre les fonctionnalités d'un exécuteur. Le choix de moyens de calcul est flexible.

On peut résumer deux points importants que il faut résoudre pour améliorer l'exploration de simulateur complexe :

— Réduire le temps de calcul avec les plate-formes intermédiaires pour le calcul intensif
— Accélérer l'exploration par l'association des algorithmes d'exploration

On va développer ces deux points dans les deux sous-section suivantes.

4.1.1 Réduire le temps de calcul avec les plate-formes intermédiaires pour le calcul intensif

Il est possible de réaliser un passage à l'échelle d'un ordinateur personnel à une grille informatique. Le temps de calcul d'une seule simulation du simulateur complexe peut coûter de quelques minutes à plusieurs heures, jours, voire semaines. Bien que les méthodes d'opti-

misation évitent l'exploration exhaustive ou systématique, elles requièrent souvent un grand nombre de simulations avant de donner des résultats fiables.

Il y a deux approches principales qui peuvent être utilisées pour réduire le temps de calcul de l'expérimentation pour obtenir des résultats en tirant la puissance d'une grille informatique. La première vise à distribuer une simulation unique sur le système de calcul distribué. Cependant, le découpage et la fusion ne sont pas faciles à utiliser parce qu'un modèle abordé dans cette thèse sont souvent multi-niveau et individu-centrée. La deuxième est plus facile à aborder en terme de technique, elle vise à exécuter en parallèle des simulations sur une grille informatique, de manière indépendante.

Cette thèse aborde la deuxième approche. La grille informatique est un outil utile pour accélérer la réalisation des plans d'expériences. Elles offrent une solution pour le calcul intensif, mais le passage de calcul à grande échelle reste un véritable défi. La probabilité cumulée de pannes et l'impossibilité de distribuer de manière optimale la charge de travail du système de grille informatique rend leur utilisation effective très difficile. Les plate-formes telles que Kepler, OpenMOLE veulent trouver des solutions à ces difficultés en fournissant un pont qui permet aux modélisateurs de surmonter cette effort technologique. Ces outils proposent des concepts et des briques cohérents permettant la construction d'expérimentations réutilisables en vue : d'éviter de développer un outil ad-hoc pour chaque modèle ; d'assister la conception de scénarios d'exploration ; de faciliter l'utilisation de clusters ou de grilles, de faciliter l'accès aux bibliothèques de plans d'expériences et d'analyse statistique, de gérer la reproductibilité / traçabilité des expériences.

4.1.2 Accélérer l'exploration par l'association des algorithmes d'exploration

La parallélisation d'un algorithme, par exemple avec l'algorithme évolutionnaire permet d'exploiter la capacité de calcul des systèmes de calcul distribués en même temps qu'accélérer la vitesse d'exploration. Plusieurs modèles d'association d'un algorithme d'exploration ont été proposés, analysés [CT10] tels que : l'algorithme métaheuristique en île et en diffusion 4.1 ; l'algorithme métaheuristique s'associe avec l'algorithme de recherche local 4.2. Le type de coopération peut être décentralisé ou peut être centralisé 4.3. Illustrons nos propos avec les figures 4.2 et 4.3 afin de distinguer la centralisation de la décentralisation. La centralisation est considèrée quand les informations se transfèrent via un structure de mémoire central : toutes les informations sont partagées entre les algorithmes associés doivent être envoyés à la mémoire centrale. La décentralisation est considérée quand les informations sont envoyées entre deux algorithmes associées et n'influence un troisième algorithme s'il existe : chaque algorithme doit gérer les informations reçues lui-même.

En réalité, une stratégie algorithmique générale pour résoudre des problèmes différents d'une manière efficace n'est pas et ne sera jamais disponible (voir la théorème «No Free Lunch »de Wolpert et Macready [WM97]). Ainsi les algorithmes d'exploration peuvent être très efficaces sur certains problèmes mais risquent de donner des résultats insufisant pour d'autres.

Dans ce contexte, les méthodes de couplage des algorithmes globales et locales sont souvent prometteuses [SZS09, JFT05, HH06, OB04], par exemple :

— l'algorithme mémétique combiné un Algorithme Evolutionaire (AE) et un algorithme de recherche local 4.5. Cet algorithme s'exécute en deux temps : (i) sélection d'un jeu de paramètres à l'aide d'un AE ; (ii) utilisation d'un algorithme de recherche locale pour raffiner les solutions trouvées. L'interaction entre l'AE et l'algorithme de

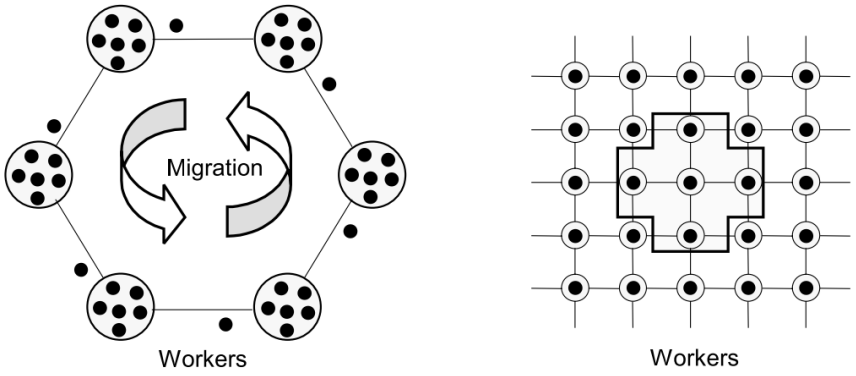

Figure 4.1 – L'algorithme évolutionnaire coopèrent dans le modèle en île et le modèle de diffusion

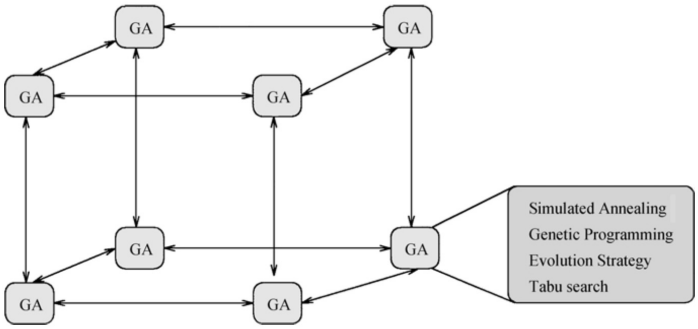

Figure 4.2 – Le hétérogénéité dans le modèle en île

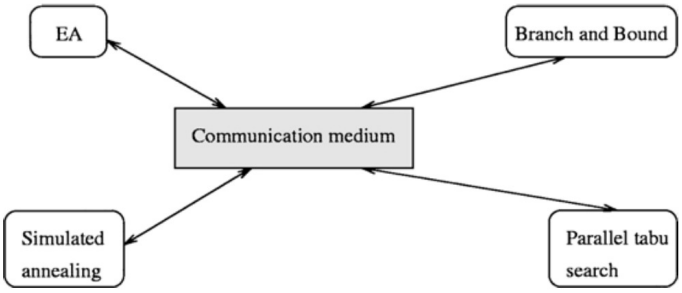

Figure 4.3 – La centralisation dans le modèle en île hétérogène

recherche locale est permise : (i) par une mémoire partagée [TB06a] 4.4 ; (ii) ou par des interactions entre agents [MKC10] 4.6.

— la combinaison entre les méthodes déterministes avec des approches heuristiques et stochastiques [Tal02, CMT04, BKB96].
— la combinaison entre l'algorithme de recherche dichotomique, l'algorithme de recherche globale et l'algorithme de recherche locale [SK98, KS05, MKC10, HHM07, EVBM09, GK10, XNH10, HMP+13].

Dans les travaux de *Hiwa et al.* [HHM07], l'algorithme de criblage (DIRECT) a été associé à la fois avec un algorithme de recherche globale (Algorithme génétique) et un algorithme de recherche locale SQP. Cette solution offre une meilleure couverture de l'espace des paramètres

lorsque ce dernier n'est pas homogène.

Le travail de *Griffin et al.* [GK10] tente de capitaliser des algorithmes pour les combiner sous la forme de composants logiciels 4.7. Plusieurs composants de recherche sont alors disponibles (LHS, DIRECT, APPS) et rendent la plate-forme flexible, distribuable sur plusieurs nœuds de calcul. De cette façon, l'espace des solutions est attentivement exploré en se basant sur une stratégie prédéfinie, choisie en fonction du cas d'application.

Combiner différentes stratégies d'exploration permet de tirer partie de chacune d'elles voire d'inhiber leurs lacunes. De nombreux travaux proposent d'associer, par exemple, les algorithmes évolutionnaires avec d'autres approches comme DIRECT [HHM07]. Leur association offre une cartographie de l'espace des paramètres [SW10, CH06] et de mieux l'explorer ensuite.

Quand l'exploration d'un simulateur complexe est considérée comme un problème d'optimisation : nous tentons de trouver un jeu de paramètres qui minimise (ou maximise) une fonction d'utilité, fonction nécessaire pour évaluer la réponse du modèle vis-à-vis de la question scientifique. Nombreux sont les algorithmes d'exploration. Ils s'organisent dans trois catégories [CH06, Sto11] : Recherche locale (ex. HillClimbing, Recuit Simulé, Tabu Search, Pattern Search), Recherche globale (ex. Algorithme génétique, Stratégie d'évolution, Optimisation par essaims particulaires) ou Recherche par criblage (ex. Latin Hypercube Samling, DIviding RECTangle, Recherche Dichotomique). En résumé, les trois stratégies précédemment citées sont complémentaires. Les approches de recherche locales sont précises et efficaces mais tombent dans le piège de l'optimum local. Inversement, les approches de recherche globales s'affranchissent des optimums locaux mais restent imprécises. Les approches de criblages sont quant-à elles adaptées pour réaliser une cartographie de l'espace des paramètres. Pour ces raisons, l'utilisation d'un (méta)-heuristique unique peut être plutôt restrictive lorsqu'on veut résoudre un problème en monde réel. Cette complémentarité des approches nous laisse imaginer l'intérêt de les associer. L'idée clé derrière la recherche coopérative est de combiner les points forts des différences (méta-heuristiques) pour équilibrer intensification et la diversification et pour diriger la recherche vers des régions prometteuses de l'espace de l'espace paramètres [OP10].

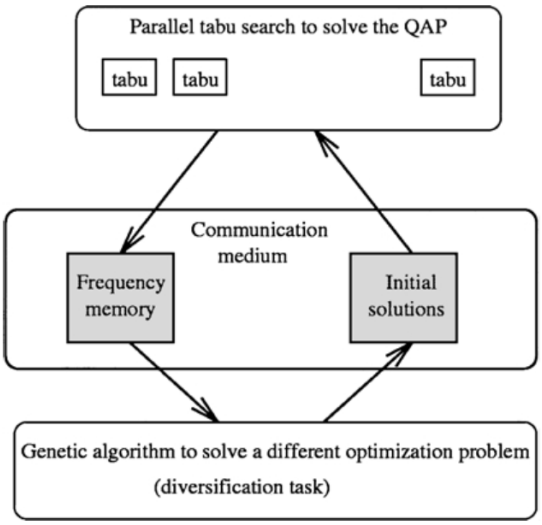

Figure 4.4 – CoSearch : Plusieurs algorithmes de recherche hétérogènes spécialistes

L'étude des travaux existants nous permet de comprendre l'état existant de l'exploration, de simulateur complexe et d'identifier les verrous scientifiques que nous devons faire face. Ces

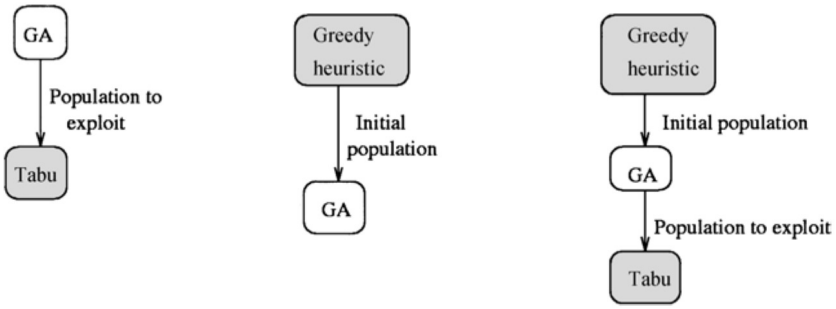

Figure 4.5 – Algorithme mémétique : coopération entre l'algorithme évolutionnaire et la recherche locale

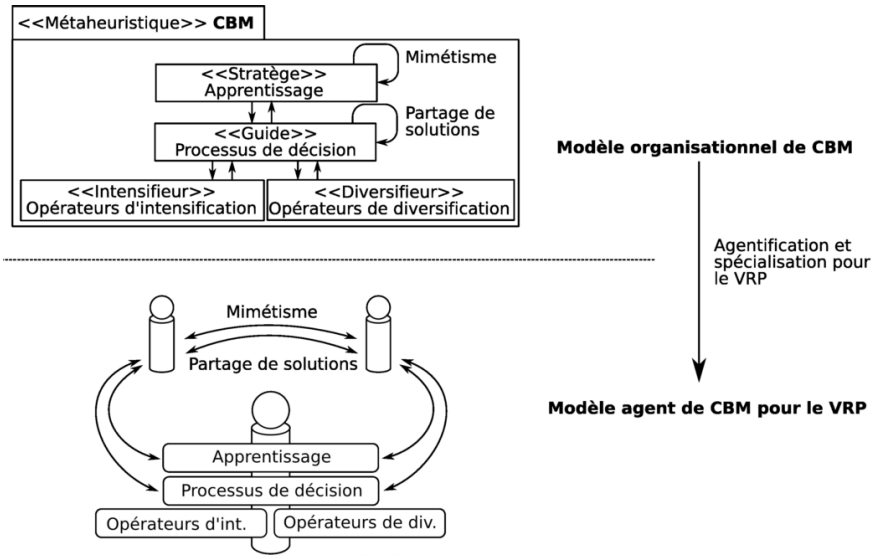

Figure 4.6 – Métaheuristique à base de la coalition pour recherche coopérative

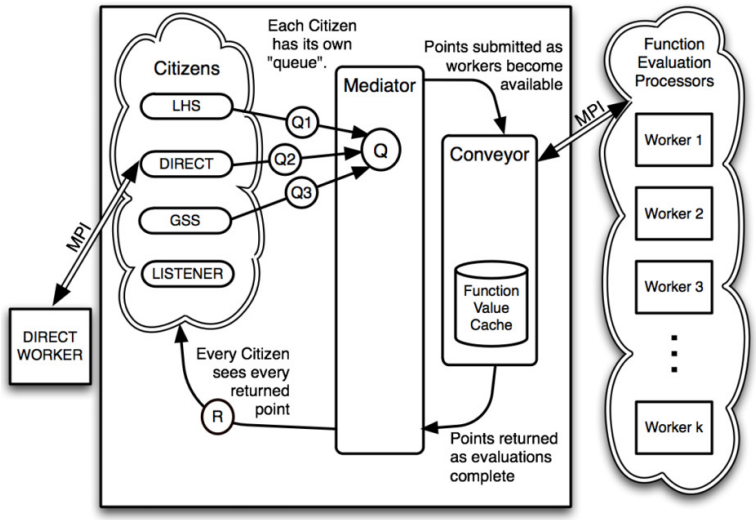

Figure 4.7 – DIRECT, GSS et LHS dans une cadre parallélisé de recherche coopérative

verrous sont abordés en détail dans la section suivante.

4.2 Les verrous scientifiques de la thèse

Pour le rappel, on a trouvé dans la section précédente deux points importants dont il faut chercher une solution pour explorer efficacement les simulateurs complexes :

— la réduction du temps de calcul en utilisant la calcul intensif et

— l'accélération de l'exploration en utilisant l'association des algorithmes d'exploration.

Dans cette section, on donne les verrous informatiques qu'il nous faut faire face pour atteindre les deux points.

— **Modularité des calculs :** comment intégrer différents simulateurs complexes de façon générique sur un plate-forme pour les rendre réutilisable ? L'enjeu est de permettre de générer les jeux de paramètres à partir d'un ensemble de paramètres d'entrée, et d'assurer la capacité de distribuer l'exécution des simulations sur différents systèmes de calcul intensif. Comment facilité l'exécution des simulations sur différent système de calcul intensif ?

— **Modularité des algorithmes d'exploration :** comment intégrer différents algorithmes d'exploration sur une plate-forme et rendre ces algorithmes réutilisables dans différentes expérimentations ?

— **Coordination des algorithmes d'exploration :** comment concevoir une stratégie de coopération entre les algorithmes d'exploration pour que ils s'associent ? Comment concevoir un protocole de communication générique pour que ces différents algorithmes partagent des informations dans une phase d'exploration coopérative ? Quelles informations sont échangées entre les algorithmes d'une stratégie ? Comment gérer les informations échangées ?

Pour résoudre ces verrous informatiques, nous présentons notre solution dans la section suivante.

4.3 GRADEA - une plate-forme pour l'exploration coopérative

L'utilisation de stratégies hybrides parallèles d'exploration, établies au cas par cas en tenant compte des ressources de calcul, s'avère nécessaire. Mais la mise en place de tels algorithmes est coûteuse en termes de développement et de mise au point. Il devient important de proposer des outils nouveaux facilitant la mise en place de tels algorithmes hybrides.

Compte tenu de la singularité des simulateurs complexes et des difficultés rencontrées pour le couplage d'algorithmes d'exploration, nous proposons un outil de conception. Cet outil facilite d'une part le rapprochement et l'interaction entre une sélection d'algorithmes d'exploration, et d'autre part l'utilisation de ressources de calcul haute performance. L'enjeu est de proposer, à la communauté, un environnement optimisé où l'utilisateur sera en mesure de :

— *combiner des algorithmes d'exploration* adaptés à son cas d'étude ;

— *tirer partie des ressources disponibles de calcul haute performance* pour réaliser l'exploration.

Pour cela, nous nous reposons sur une architecture modulaire à base d'agents qui sont en interaction avec des nœuds de calcul où sont exécutées les simulations. Cette architecture s'appelle GRADEA. Sa conception à nécessité de répondre au différents verrous précités. En réponse, nous proposons une architecture originale que nous décrivons dans la suite de la section.

4.3.1 Infrastructure logicielle de GRADEA

Cette partie présente une infrastructure logicielle pour la conception de la coordination des algorithmes d'exploration existantes. L'infrastructure logicielle est conçue pour faciliter l'accès de l'utilisateur au système GRADEA. La conception se décompose en quatre parties (figure 4.8) :

1. Le modèle du système complexe – il est responsable de transformer un problème réel spécifique en une représentation informatique. Par conséquent, l'exploration d'un système complexe devient le processus visant à explorer le modèle informatique.
2. la méthode d'exploration – elle est responsable de fournir les algorithmes de recherche pour l'exploration. Un ensemble d'algorithmes (algorithme d'optimisation globale, locale ou criblage) est déjà disponible via l'interface générique.
3. la méthode de coordination – elle est responsable de manipuler la coopération et l'exécution des méthodes d'exploration. Cela signifie que la stratégie de coordination des algorithmes et la stratégie d'exécution sur différents moyens de calcul.
4. les moyens de calcul intensif – ils sont responsable de réduire le temps de calcul quand on explore un simulateur large échelle par des algorithmes s'exécutant de manière concurrente.

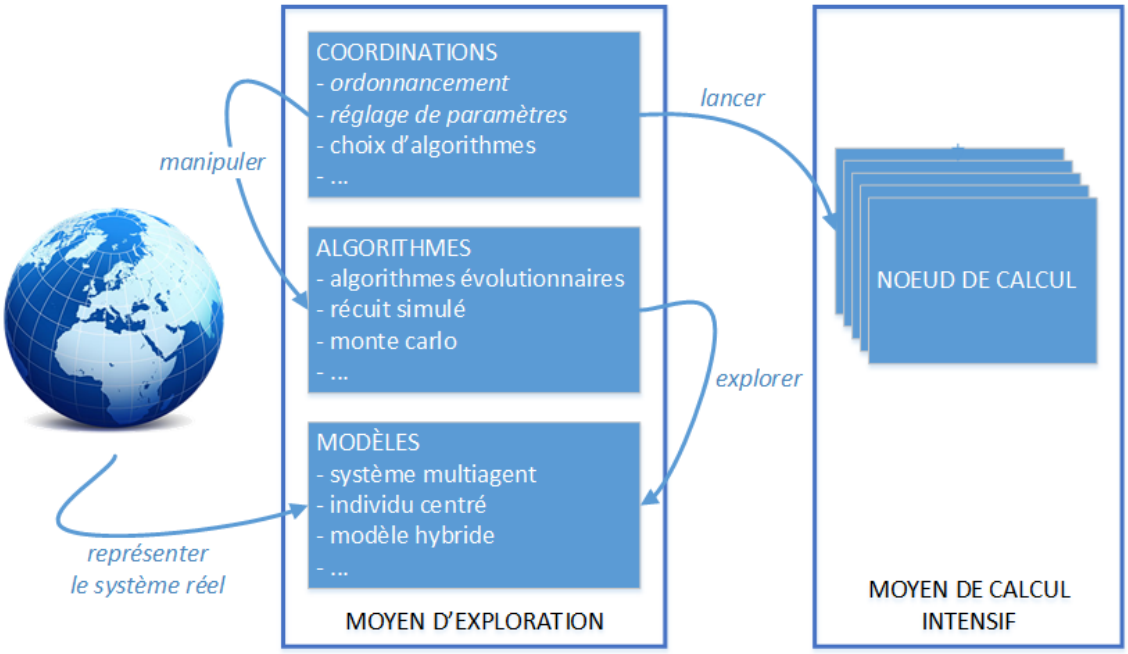

FIGURE 4.8 – La conception de coordination des explorations

Nous précisons différentes parties dans les sous-sections suivantes :

4.3.2 Modulation les calculs

La distribution des modèles informatiques au moyen du calcul intensif est réalisable en proposant une architecture modulaire et extensible permettant de : (i) sélectionner un simulateur ; (ii) créer un plan d'expérimentation ; (iii) lancer un grand nombre de simulations sur un cluster, (iv) mettre en place de nouveaux simulateurs. Ce travail est réalisable par la généricité

de l'entrée et de la sortie d'un modèle de système complexe en variables numériques. Un ensemble d'exécutions d'un modèle (avec la même ou une variété de configurations) est lancé sur les nœuds de calcul d'un cluster. Les résultats sont récupérés automatiquement dès que le calcul est terminé.

A titre d'exemple, dans les paragraphes suivants un exemple avec le module de calcul sur cluster en utilisant le script SGE est présenté.

Un plan d'expérience correspond habituellement à plusieurs exécutions du même simulateur avec des paramètres différents. Comme le nombre d'exécutions d'un plan d'expérience peut être très important, il est rarement possible de simplement soumettre l'ensemble des exécutions dans une file d'attente d'un cluster à cause de ses politiques d'administration. Toutefois, il est possible de limiter le nombre maximum de jobs soumis en même temps comme cela est présenté par le script SGE (Sun Grid Engine) dans la figure 4.9. Le gestionnaire de ressources lance d'abord 20 jobs sur 100 en même temps. Puis il lance un nouveau job dès que l'un des jobs en cours est achevé.

```
#!/bin/bash -l
#$ -q normal2h
#$ -t1-100 -tc 20
#$ -o \$JOB_NAME.\$JOB_ID.out
#$ -e \$JOB_NAME.\$JOB_ID.err
./epis input\$SGE_TASK_ID.xml output\$SGE_TASK_ID.xml
```

FIGURE 4.9 – Un exemple du script SGE pour lancer un plan d'expérience

Dans le script SGE, on peut regarder le chemin au fichier d'entrée et celui du fichier de sortie. Pour que le script SGE peut être lancé, le fichier d'entrée devait exister sur le cluster. A partir des variables d'environnement et les chemins de répertoire qui sont spécifiques à cette exécution, le serveur d'application de la couche de soumission génère un fichier SGE et un fichier des paramètres de la simulation. Les fichiers de paramètres sont, en fait, les fichiers XML différenciés par $JOB_ID. Ces fichiers déterminent par exemple dans la figure 4.10 :

- le simulateur utilisé (i.e. driver="Simulator.Sworm") ;
- l'étape finale (i.e. finalstep="1000") ;
- la valeur des paramètres (i.e. nbOfAnomala - le nombre des verres de terre au début) ;
- les sorties et leur fréquence d'affichage (i.e. 2DDisplay-15 - la capture de terre au 15e profond du volume de terre).

```
<?xml version="1.0" encoding="UTF-8"?>
<Simulation id="2" driver="Simulator.Sworm" finalstep="1000">
    <Parameters>
        <Parameter name="nbOfAnomala" type="INT" value="200" />
        <Parameter name="assimRateAnomala" type="FLOAT" value="9.0f" />
    </Parameters>
    <Outputs>
        <Output id="2" name="SoilOrganicMatter" framerate="1"/>
        <Output id="3" name="2DDisplay-15" framerate="10"/>
    </Outputs>
</Simulation>
```

FIGURE 4.10 – Un extrait du fichier d'entrée pour la simulation de ver de terre

Le format de fichier des paramètres de la simulation que l'on utilise dans l'exemple ci-dessus (figure 4.10) va être utilisé en tant que format générique pour décrire différentes simulations. Dans le chapitre 6, nous abordons plus en détail l'architecture logicielle de l'environnement GRADEA. Avant cela, il est important d'établir les principes d'interconnexion entre les algorithmes.

4.3.3 Modularisation des algorithmes d'exploration

GRADEA tente d'associer plusieurs algorithmes d'exploration dans une même phase d'expérimentation et faciliter leur intégration, leur réutilisation pour différentes expérimentations et leur modification dynamique (pendant une expérimentation). Pour cela, Pour cela, il faut prendre en compte les trois critères suivants :

Modularité – Chaque algorithme d'exploration étant indépendant et autonome, il est possible d'en réduire le nombre ou d'en ajouter de nouveaux sans impacter le système et être obligé de modifier son architecture et son fonctionnement.

Autonomie – Les algorithmes sont capables de gérer de manière flexible les messages reçus en provenance des autres algorithmes. Ils décident de ce qu'il va accepter, ignorer ou pré-traiter.

Hétérogénéité – Différents algorithmes d'exploration peuvent coopérer pour l'exploration de l'espace de paramètres. Dans ce contexte, le couplage des méthodes d'exploration de différentes catégories (global et local, criblage, etc) est souvent plus prometteuse qu'une approche unique.

Pour répondre à ces critères, une interface générique a été mise en place. Elle considère chaque algorithme d'exploration comme une boîte noire définie par ses entrées et ses sorties. Les entrées et les sorties sont généralement des jeux de paramètres. Chaque algorithme doit être initié par ses paramètres définis. Chaque algorithme prend un ou plusieurs jeux de paramètres initiaux comme les candidats de solutions. Il les encode via en une représentation interne à chaque algorithme, les modifie d'une manière particulière par les opérateurs de l'algorithme, et produit les solutions à la sortie. Ces dernières sont décodées et encodées en jeux de paramètres pour envoyer à d'autre algorithme.

La représentation interne de solutions candidates pour un jeu de paramètres pouvant être compris algorithme est la première chose que nous avons à traiter au cours du processus d'unification des algorithmes. Une telle représentation est spécifique à chaque algorithme. Des transformations doivent être réalisées pour le passage des solutions candidates d'algorithme en algorithme. Nous proposons donc une description uniformisée des jeux de paramètres, description pouvant être transformée par chaque algorithme dans leur représentation. La forme générique d'un algorithme est dans la figure 4.11.

A partir de la forme générique d'un algorithme d'exploration, nous proposons une interface générique pour que les algorithmes d'exploration doivent implémenter pour être intégré dans GRADEA (la figure 4.12). L'interface se compose par :

- Solution Initializer (SI) qui définit le plan d'expérience et générer les candidats de solution en utilisant les informations de Problem Simulator ;
- Problem Simulator (PS) qui définit les jeux de paramètres d'un problème ;
- Algorithm State (AS) qui exécute les opérateurs de l'algorithme ;
- Solution Analyzer (OA) qui observe l'exécution de l'algorithme et fournir des interfaces de sortie à l'autre algorithme ;
- Solution Evaluator (SE) qui est en charge d'une fonction objectif et cd lancer les calculs ;
- Config Builder (CB) génère les paramètres de l'algorithme.

Nous allons préciser cette modularité dans le chapitre 5.

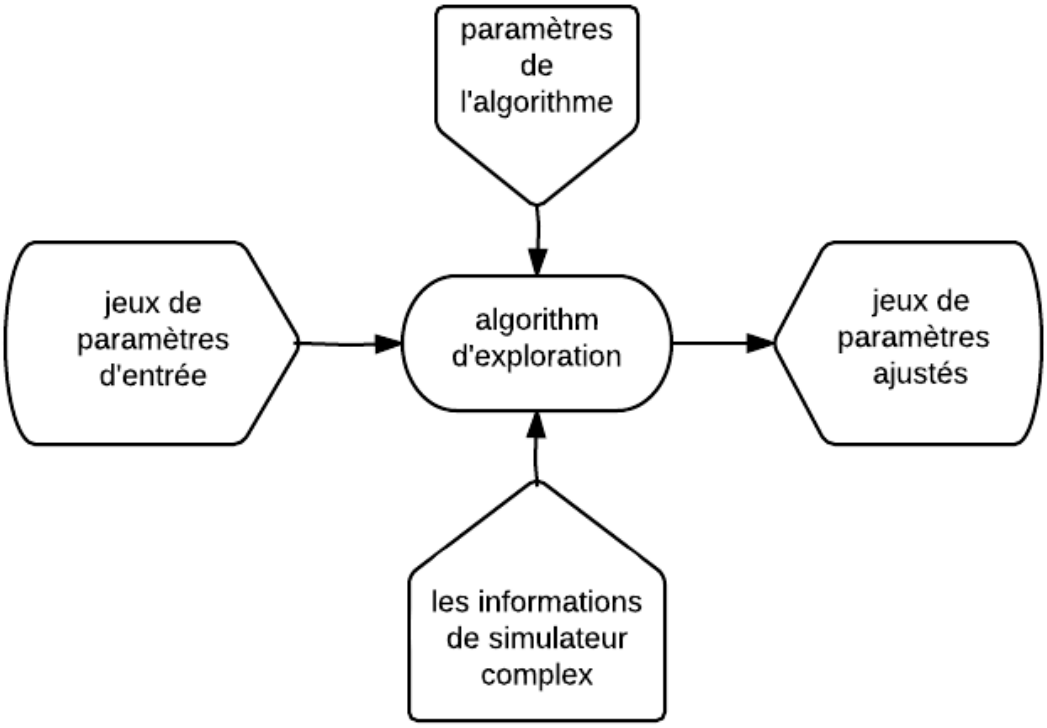

Figure 4.11 – Le modèle d'abstraction de l'algorithme d'exploration (interface d'adaptateur de l'algorithme)

4.3.4 Coordination des algorithmes d'exploration

Pour coordonner les algorithmes d'exploration, l'approche à base d'agents a été suggérée dans les travaux récents qui veulent proposer une plateforme de conception des stratégies de recherche coopératives 4.13. Le paradigme des systèmes multi-agents supporte au début caractéristiques tels que :

- un organisme des entités homogènes ou hétérogènes ;
- leurs comportements sont autonomes et sont capables d'être formalisés et d'être automatisés ;
- les comportements des agents sont adaptés de manière événementielle ;
- ils interagissent (coordination, coopération et communication) pour transférer les messages entre eux dans un environnement pour les objectifs communs ou divergents.

Le système multi-agent (SMA) dans ce contexte est un outil étendu permettant de relier les stratégies d'exploration avec les méthodes d'optimisation choisies pour un cas d'application spécifique. Ainsi l'agent, le SMA offrent une architecture modulaire qui assurent le couplage de plusieurs algorithmes d'exploration, selon les besoins. L'enjeu est de permettre aux utilisateurs de définir simplement des stratégies d'exploration qui s'appuient sur des algorithmes existants.

Grâce à cette architecture et par la combinaison et l'ordonnancement de d'algorithmes, nous souhaitons faciliter la définition de stratégies personnalisées et adaptées à l'exploration de simulateurs complexes.

Aydin [Ayd07] a souligné que les systèmes multi-agents offrent un moyen naturel pour mettre

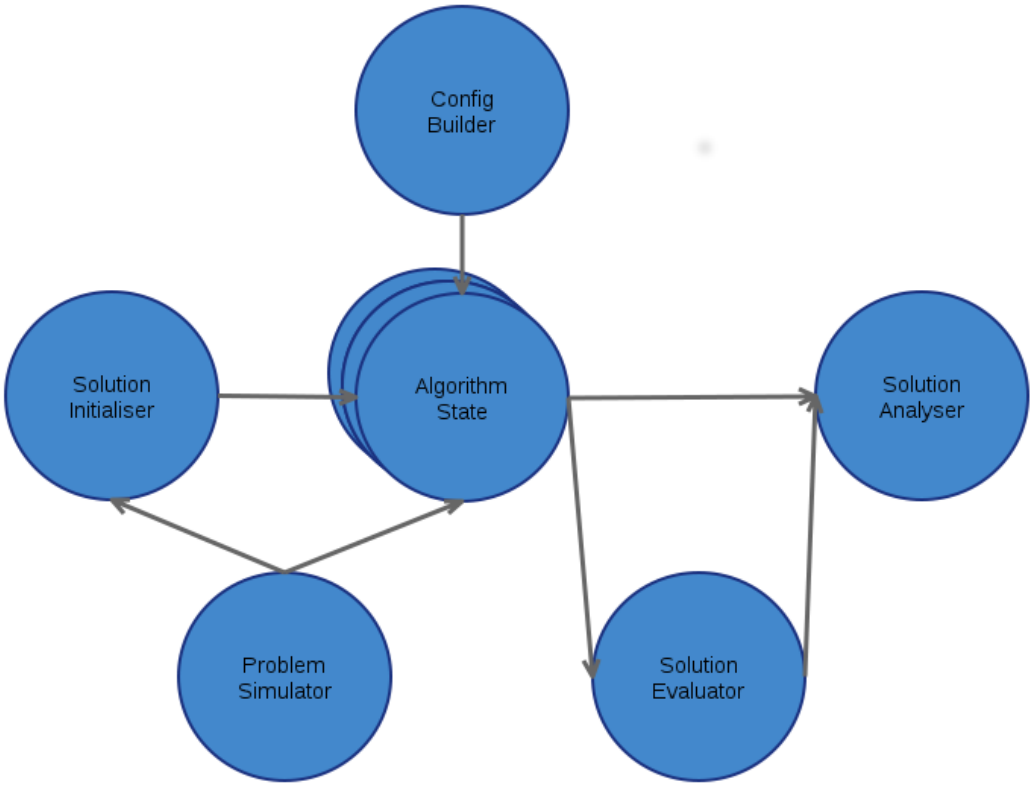

Figure 4.12 – Modularisation d'un algorithme d'exploration

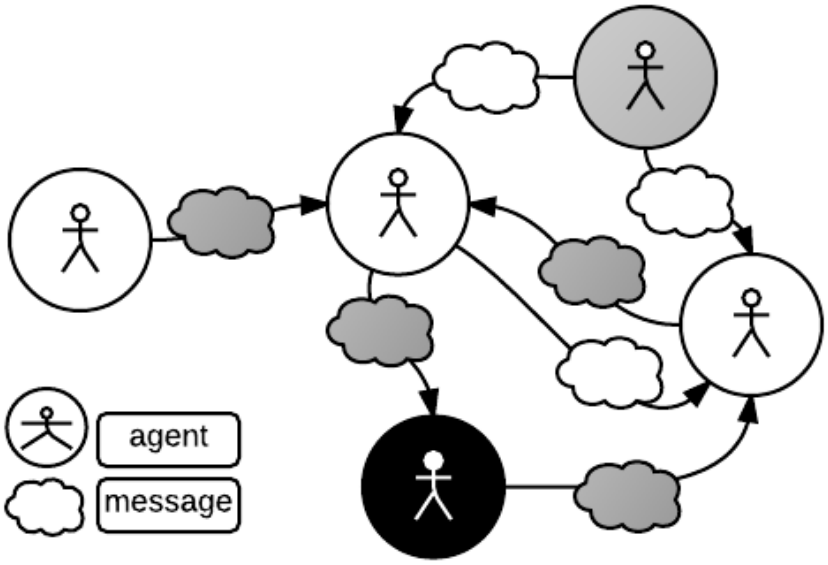

Figure 4.13 – Les entités d'un système multi-agents échangent l'information en messages

en œuvre l'exploration coopérative où chaque agent met en œuvre différents ou les mêmes algorithmes tout en échangeant des informations utiles sur la recherche entre eux. Il n'existe pas encore un cadre suffisant de ce type du système de taxonomie de [CT10] ou d'un système

multi-agents complet de transmission de messages. Il est important de réaliser un système à base d'agent qui n'est pas seulement une collection de processus ou threads distribués pour résoudre un problème. Chaque agent dans un système à base d'agent est un programme autonome de son propre. Il peut effectuer une tâche attribuée sans recours à un processus de gouvernance. Cela signifie que le système multi-agent aura une représentation interne de son environnement et sera en mesure de répondre à d'autres agents. Une autre caractéristique de ces systèmes est qu'ils communiquent par le passage de messages qui s'oppose à des appels de fonctions des autres processus. Dans un système de transmission de messages, l'agent doit encoder un message à partir de sa propre représentation en suivant un protocole pour et transmettre le message à un autre agent. Un agent récevant le message doit décoder le message en suivant le protocole pour le transcrire dans sa propre représentation interne. Pour cette raison, des systèmes d'agents sont souvent, mais pas toujours, répartis sur plusieurs machines. Une telle résolution assure :

la flexibilité – L'interaction entre les instances d'algorithmes dans les travaux existants est limitée par les choix du concepteur. Il semble plus intéressant qu'une stratégie coopérative décide, d'elle même, d'envoyer les différents types de messages durant l'exécution. L'interaction influence l'émergence de l'exploration. Par exemple, une instance peut décider d'envoyer une solution qu'elle considère comme un point très prometteur pendant la recherche au lieu de ne faire qu'une le processus terminé. il faut réfléchir une architecture distribuée en deux aspects : chaque instance d'exploration peut être distribuée implicitement et la coopération est asynchrone ou synchrone. Le choix de la synchronisation/l'asynchronisation dépend de la stratégie de coopération. Parfois, les deux type d'échange apparaissent dans une même coopération afin, par exemple, d'assurer une répartition de charge.

Versatilité : Les algorithmes pouvant être choisis à la demande, la stratégie de coordination pouvant être déterminée par l'utilisateur, il faut permettre une importante versatilité du système afin de pouvoir l'appliquer dans de nombreux domaines. La combinaison de méthodes d'exploration ainsi que le paramètrages de stratégie sont fixés en avance par les expérimentations déjà réalisées ou possiblement d'être modifiées en cours d'exploration.

Le travail de cette thèse a permis de fournir un outil pour décrire la coordination entre les algorithmes d'optimisation pour l'exploration coopérative distribuée de simulateurs complexes. Nous avons souhaité proposer des outils qui encouragent la modularité et la réutilisation des modèles. Ce travail est réalisé en utilisant l'approche organisationnelle à base d'agent pour fournir un ensemble de concepts communs à différents algorithmes d'optimisation et un modèle générique de la coordination de ces agents. D'autre part, il faut prendre en compte la capacité à mettre le système disponible avec le calcul distribué.

Discussion

L'approche GRADEA montre tout son potentiel lors que nous souhaitons explorer un modèle de système complexe. Le paradigme agent qu'elle embarque permet d'identifier 3 catégories d'*agents de recherche* : (i) les Agents de Recherche Globale (ARG) ; (ii) les Agents de Recherche par Criblage (ARC) ; (iii) et les Agents de Recherche Locale (ARL) présentés dans 4.14. Ces agents ont la particularité d'implémenter un algorithme d'exploration de manière décentralisée ou centralisée.

A titre d'exemple, un déroulement très simple d'une coopération centralisée d'une simulation complexe serait :

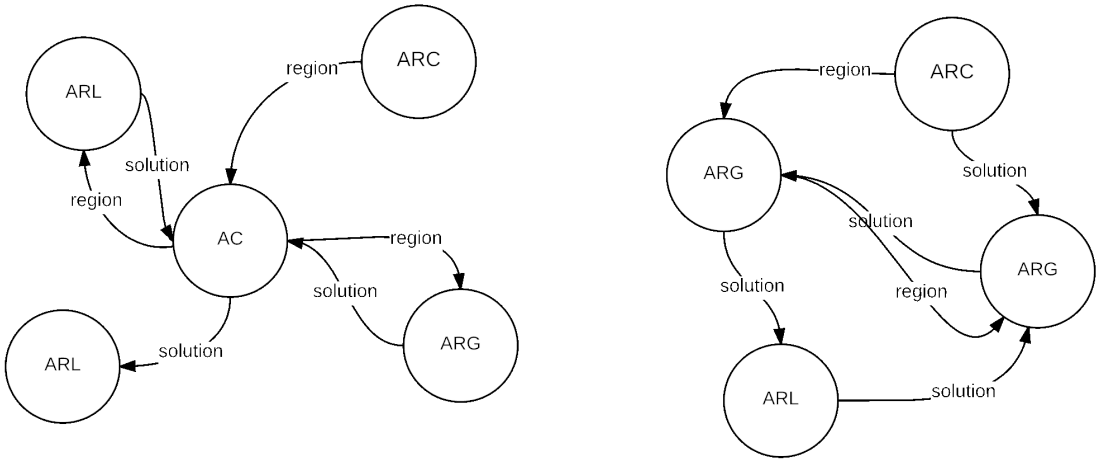

Figure 4.14 – La coopération centralisée (gauche) et décentralisée (droite) entre ARG, ARC et ARL

1. L'agent de coordination ordonne à une instance d'*ARC* de cribler l'espace des paramètres.
2. L'agent de coordination ordonne le transfert une région prometteuse à une instance d'*ARG*, et à une instance d'*ARL*, une meilleure solution à une autre instance d'*ARL*.
3. L'*ARG* qui est implémenté un algorithme évolutionnaire, explore le simulateur pendant un certain nombre de générations et prévient l'agent de coordination de la découverte d'une solution intéressante. L'*ARG* continue ensuite à itérer.
4. Pendant ce temps-là, l'agent de coordination choisit une instance d'*ARL* et lui transmet le jeu de paramètres correspondant à la solution précédemment découverte par l'*ARG*.
5. et ainsi de suite jusqu'à l'obtention d'une solution satisfaisante...

Un autre déroulement d'une coopération décentralisée d'une simulation complexe serait :

1. une instance de d'*ARC*, deux instances de d'*ARG* et une instance de d'*ARL* sont lancées en parallèle.
2. L'agent d'*ARC* envoie à une instance d'*ARG* une région prometteuse, une meilleure solution à une autre instance d'*ARL*.
3. L'*ARL* raffine la solution reçu.
4. L'*ARG* explore la région prometteuse réçu, et après renvoie la meilleure solution trouvé à l'instance d'*ARL*.
5. L'*ARL* raffine d'une solution intéressante reçoit à partir de l'instance d'*ARG* qui a réçu la région prometteuse, et après renvoie la meilleure solution trouvé à l'instance d'*ARG*.
6. Pendant ce temps là, deux agents d'*ARG* partagent les régions et les solutions prometteuses ensemble.
7. et ainsi de suite jusqu'à l'obtention d'une solution satisfaisante...

Deux types d'agents sont alors considérés dans l'approche :

— *les agents d'exploration* réalisent l'exploration d'un espace (ou sous espace) de paramètres selon un algorithme d'exploration défini dans son comportement (ex. Monte-Carlo, évolutionnaire, etc) ;

— *les agents de coordination* ordonnancent, quant à lui, les différentes explorations en interagissant avec les *agents de recherche* pour ordonner une exploration et/ou récupérer et analyser des résultats (même partielle). La stratégie d'ordonnancement est définie par l'utilisateur au sein du comportement de l'agent coordinateur.

Ces deux composants deviennent le noyau de conception de GRADEA que nous présentons dans la section suivante.

Chapitre 5

Méta-Modèle de GRADEA

Sommaire

Ce chapitre présente en détail l'architecture conceptuelle de plate-forme GRADEA que nous avons présentée en tant que notre contribution de la thèse dans le chapitre 4. L'architecture conceptuelle se compose d'un méta-modèle de GRADEA. Nous le formalisons en suivant l'approche de modélisation VOYELLES. Ce chapitre se compose deux parties : la première partie fait une description du méta-modèle de GRADEA. Ensuite nous présentons la solution conceptuelle à base d'agents en suivant l'approche de modélisation VOYELLES.

5.1 Méta-modèle de GRADEA

5.1.1 Concept clés de GRADEA

Dans cette section, nous présentons le méta-modèle de GRADEA (figure 5.1). Ce méta-modèle vise à essayer de distinguer la tâche d'exploration de la tâche de coordination. Dans ce cas, on considère qu'une instance de l'algorithme d'exploration avec une configuration donnée est une unité de résolution. La communication entre ces unités est manipulée de manière indépendante par un élément coordination. De cette façon, en effectuant une exploration, l'explorateur peut exécuter différentes combinaisons des algorithmes avec différents paramètres.

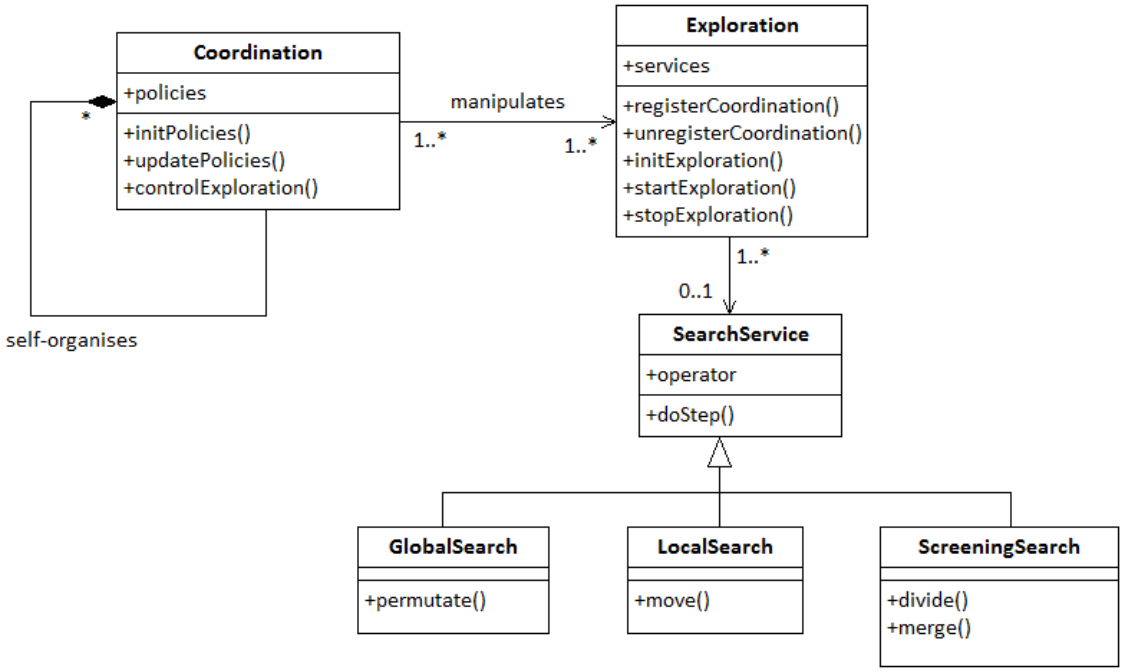

Figure 5.1 – le méta-modèle de GRADEA

Le méta-modèle organise deux éléments principaux : l'exploration et la coordination.

— *Exploration :* Le constituant d'exploration joue le rôle de manipulation d'un algorithme d'exploration en appelant les services spécifiques structurés de l'algorithme. L'exploration, d'une part communique avec les coordinations afin : (i) de s'inscrire auprès des coordinateurs, d'appeler les services de recherche, et d'appeler les services de calcul pour réaliser la recherche. L'exploration contient une représentation abstraite pour être intégrée avec différents algorithmes d'exploration afin d'établir les comportements d'exploration.

— *Coordination :* La coordination manipule la coopération entre les explorateurs. Ce constituant contient un ensemble de politiques de contrôle et de dépôts des informations collectées pour définir la stratégie de coopération. Les stratégies se diversifient mais se basent sur deux directions : (1) en direct via une topologie de communication (fixée et/ou adaptée) ou (2) en indirect via une mémoire centrale. Il joue les rôles de surveillance des explorateurs et de partage des connaissances entre les explorations. Pour partager les connaissances, la coordination reçoit les informations des explora-

tions et envoit aux explorations qui ont besoin.

— *SearchService :* Les services de recherche sont les instances de résolutions pour un domaine donné. Ce sont les méthodes de recherche, méta-heuristiques ou heuristiques disponibles, accessibles et capables d'explorer un espace de paramètre. Dans ce méta-modèle, les méthodes de recherche sont catégorisées en trois groupes : la recherche globale, la recherche local ou la recherche en criblage. Cette taxonomie semble être la plus intéressante au regard de l'état de l'art que nous avons fait.

5.1.2 Principe de fonctionnement de GRADEA

GRADEA s'appuie sur une approche simple visant à explorer un simulateur complexe via un ensemble des instances d'algorithme coopérantes. Chaque instance d'algorithme développe une stratégie identifié (souvent différente) pour résoudre le problème de façon indépendante. Les instances d'algorithme s'exécutent un algorithme pendant qu'ils échangent de manière asynchrone des informations entre eux.

La coordination contrôle les nouvelles informations collectées à partir des instances d'algorithme et décide si son comportement doit être adaptée en utilisant sa base de règles de décision. Si ce n'est pas le cas, un nouveau comportement sera envoyé. Ainsi les règles de décision sont responsables de prédéfinir quelles connaissances se partagent (par exemple, partager un jeu de paramètres ou une espace de paramètres) et en quelle moment on améliorent le comportement (par exemple, changer le taux de mutation et le taux de croisement de l'algorithme génétique). Une telle règle de décision peut être appelée par une politique de contrôle. Cette notion est adoptée à partir d'un apprentissage par renforcement [SB99]. La politique se déduit à partir des connaissances de l'exécution précédente de l'algorithme comme entrée et fournit un ensemble de nouvelles valeurs de paramètres en sortie. En cas où il n'y a pas de cycle précédent au début de l'algorithme, les valeurs des paramètres par défaut sont définies.

Le fonctionnement d'un instance d'algorithme reste assez simple : pour chaque exécution débutée, un rapport d'expérimentation sur l'exécution en cours (contentant des indications sur les performances) est émis en direction de la coordination. Les informations sur la solution à un problème est périodiquement sauvegardé et partagé via un protocole de communication. La coordination observe ces informations et dirige le comportement de la recherche.

A partir du méta-modèle de GRADEA, on présente la solution conceptuelle à base d'agents suivant l'approche de modélisation VOYELLES [Dem95] et le langage de modélisation AML [CT07]. La méthode VOYELLES utilise les concepts : agent, environnement, organisation et interaction pour modéliser un système multiagent. Le cadre de l'exploration coopérative dans GRADEA est un système à base d'agents qui coopèrent avec une population d'agents explorateurs indépendants à travers un seul agent ou un ensemble d'agents coordinateurs.

5.2 Agents

Compte tenu du caractère distribué et versatile des agents, nous avons choisi de nous reposer sur ce paradigme afin de concevoir l'approche GRADEA. Ainsi, GRADEA repose sur 2 types d'agents : les agents *explorateurs* et les agents coordinateurs (voir figure 5.2).

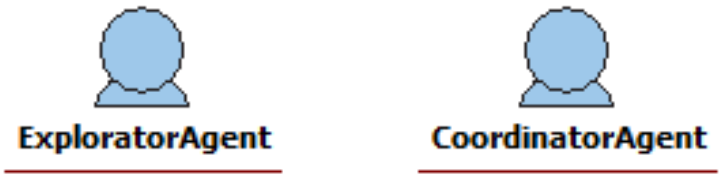

Figure 5.2 – Les agents dans GRADEA

5.2.1 Agent d'Explorateur («ExploratorAgent »)

L'agent explorateur est capable de réaliser une exploration indépendante. Cette agent perçoit l'espace de paramètres, reçois les paramètres de contrôle de l'algorithme, les paramètres de l'environnement d'exécution et les candidats de solution en entrée (voir figure 4.11). Ensuite, il exécute l'algorithme d'exploration. Après, il publie ses résultats.

Chaque agent explorateur implémente un algorithme d'exploration particulier. Son workflow est le suivant :

- configurer l'algorithme d'exploration avec les paramètres de l'algorithme d'exploration.
- initialiser les candidats de solution initiales.
- exécuter l'algorithme d'exploration avec les paramètres et les candidats reçus.
- la recherche terminée, envoyer les meilleures solutions trouvées à l'agent de coordination qui est responsable de la gestion du pool de candidats de solutions.
- envoyer un rapport de l'algorithme au mêm moment à l'agent coordinateur.

Plus précisement, les trois premières étapes vise à : (1) lire les paramètres d'entrée du simulateur complexe, les paramètres de l'algorithme utilisé, et les conditions d'arrêt ; (2) construire les configurations initiales pour les agents, les algorithmes ; (3) initialiser la procédure d'exploration pour générer les jeux de paramètres initiaux. Cette mission mesure également le temps global de l'exploration et donc la date d'expiration. Au cours de l'exploration, dans l'étape 4, les candidats sont évalués via les services de calcul intensif. Il y a une interface générique pour passer les évaluations des candidats aux ressources de calcul que l'on va représenter précisément dans le chapitre suivant.

Ces actions sont effectuées jusqu'à recevoir la signal d'arrêt de l'agent coordinateur.

5.2.2 Agent de Coordinateur («CoordinatorAgent »)

L'agent de Coordinateur contient le comportement pour associer les agents d'Explorateur. Le comportement d'un coordonnateur vise à (i) analyser l'espace de recherche et l'espace des explorateurs à partir des informations qu'il reçoit des explorateurs ; (ii) et à envoyer à ces derniers des commandes pour modifier leurs exécutions. En fonction du contexte d'exploration, l'agent de coordination à la capacité d'activer un tableau noir pour stocker les connaissances des explorateurs.

L'agent de coordination est responsable de réaliser les objectifs suivants :

- La gestion des explorateurs : il est responsable de la gestion du cycle de vie des agents dans GRADEA. Il gère une population d'agents (*Agent Management System*, AMS) et d'un répertoire de service d'exploration (*Directory Facilitator*, DF). Ce mécanisme

de gestion est tirée de la plateforme JADE [BPR01]. Elle présente l'avantage d'être modulaire. Au début et en cours d'une expérimentation, les services d'exploration (SearchService dans la figure 5.1) sont inscrits aux agents explorateur via l'enregistrement et le dé-enregistrement. Quand un agent explorateur a besoin d'un service, il interroge l'agent coordinateur pour demander le service d'exploration requis.

— L'observation : Dans notre proposition, chaque agent explorateur est abonné à un agent coordinateur pour envoyer et recevoir les candidats de solution partagés. L'observation est la capacité de l'agent coordinateur pour sauvegarder périodiquement des informations sur la solution qui sont sauvegardés dans un pool propre pour chaque agent explorateur ou soit un pool pour tous les agents d'explorateur. Tout dépend de la configuration de l'utilisateur. Avec cette capacité, l'agent coordinateur gère la population de candidats de solution et fournit des solutions à tous les agents de l'algorithme. Il reçoit des messages à partir des agents explorateur avec des demandes, soit pour la demande des candidates de solution, soit pour le stockage des candidates.

— La négociation : cet objectif est de gérer le partage des connaissances entre les explorateurs. La motivation principale est de séparer la chaîne de négociation et la chaîne de réalisation d'un algorithme indépendant. Après avoir trouvé le service d'exploration nécessaire, ce rôle permet aux agents d'explorateur de communiquer en groupe ou deux à deux.

— Le renseignement : c'est la capacité de l'agent coordinateur pour contrôler la progression de tous les agents explorateur. Si nécessaire, il (1) modifie le comportement des agents explorateur en modifiant les paramètres de stratégie des agents, (2) arrête l'agent explorateur ou (3) ajoute de nouveaux groupe d'agents sur les nouvelles régions de recherche prometteuses qu'il a trouvées en utilisant les agents explorateur.

5.3 Environnements

Les agents explorateur et coordinateur évoluent dans trois environnements distincts : l'espace de paramètres, l'environnement de calcul et les algorithmes d'exploration. Nous précisons ces trois points dans la suite de la section.

5.3.1 Espace de paramètres

L'espace des paramètres constitue l'environnement des agents exploratateurs et des agents coordinateurs (figure 5.3). Dans cet environnement, ces deux agents consomment et partagent les ressources tels que : les entrées et les sorties. Les entrées peuvent être synthétisées par une variété d'ensembles de jeux de paramètres (listes de jeux de paramètres, l'espace de recherche en cercle ou l'espace de recherche en rectangles).

5.3.2 Environnement de calcul

L'environnement de calcul décrit l'espace d'exécution des agents explorateurs et des agents coordinateurs (figure 5.4). Dans cet environnement, ces agents utilisent des ressources de calcul de types diverses tels que : simple processus leger (thread) local ou système de calcul intensif.

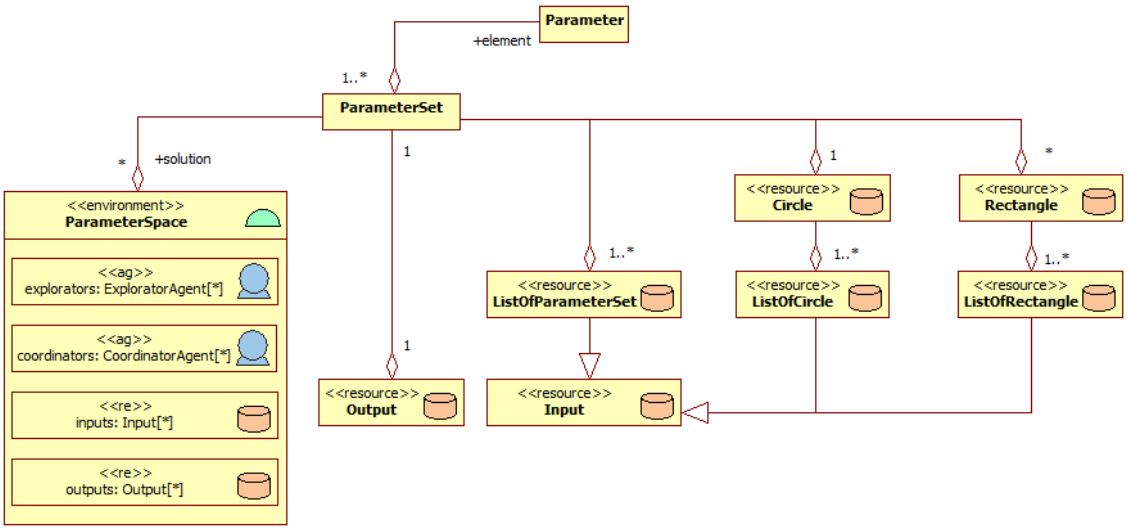

Figure 5.3 – L'espace de solutions dans GRADEA

Les thread sont particulièrement intéressantes pour l'exécution de services d'exploration. Dans cas, un thread est créé par service initié mais nous pouvons imaginer limiter ce nombre de thread par un système de *pool* calibré par l'utilisateur.

D'autre part, les ressources de calcul intensif tels que les grilles, clusters permettent d'exécuter des tâches à distance. L'environnement de calcul que nous proposons permet de déléguer des tâches de manière transparente sur des noeuds hétérogènes de calcul. L'environnement de calcul multiple est utile dans de nombreux cas, notamment lorsqu'une quantité considérable des évaluations de performance des candidats de solution sont lancés pour évaluer les fonctions objectifs.

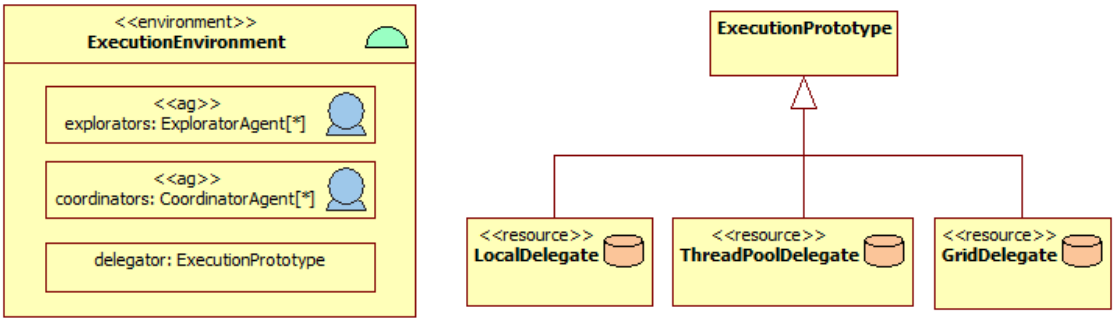

Figure 5.4 – L'environnement de calcul dans GRADEA

5.3.3 L'espace d'algorithmes

L'espace d'algorithmes est un ensemble d'algorithmes que les agents d'explorateur utilisent pour faire l'exploration de l'espace de paramètres. Dans cette environnement, les agents tirent des états actuels d'un algorithme via les informations suivantes : la solution courante qui évolue via l'application des opérateurs de l'algorithme, la meilleure solution trouvée lors de l'évolution de la solution courante, la meilleure solution connue obtenue pendant la recherche

ou/et pendant le partage avec les autres explorations, les valeurs actuels de la configuration de l'algorithme, le temps de calcul.

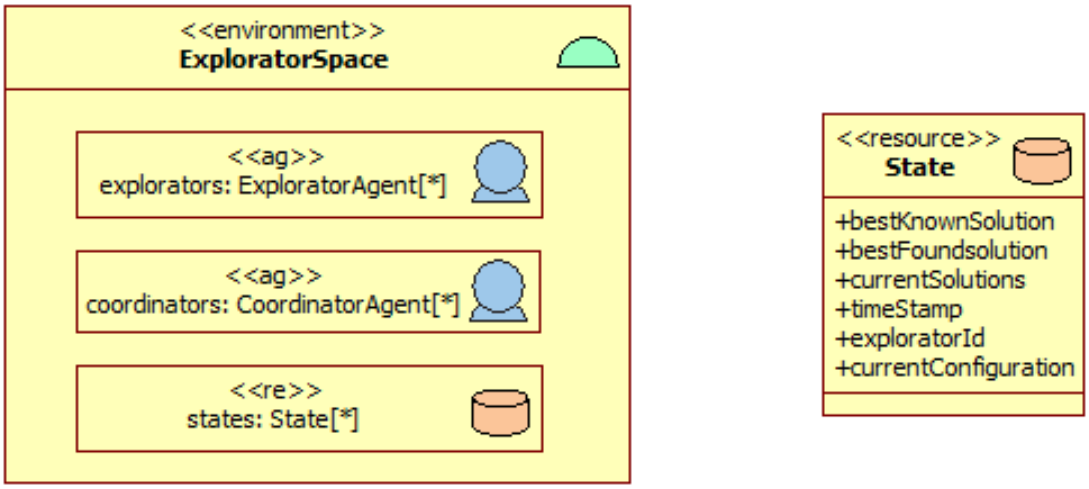

Figure 5.5 – L'espace des ex de recherche dans GRADEA

5.4 Organisations

GRADEA repose sur le concept d'Agent-Group-Role développé dans [FGM04]. Nous développons dans la suite de la section l'utilisation que nous faisons d'AGR pour notre problématiques. Nos propos sont illustrés par la figure 5.6.

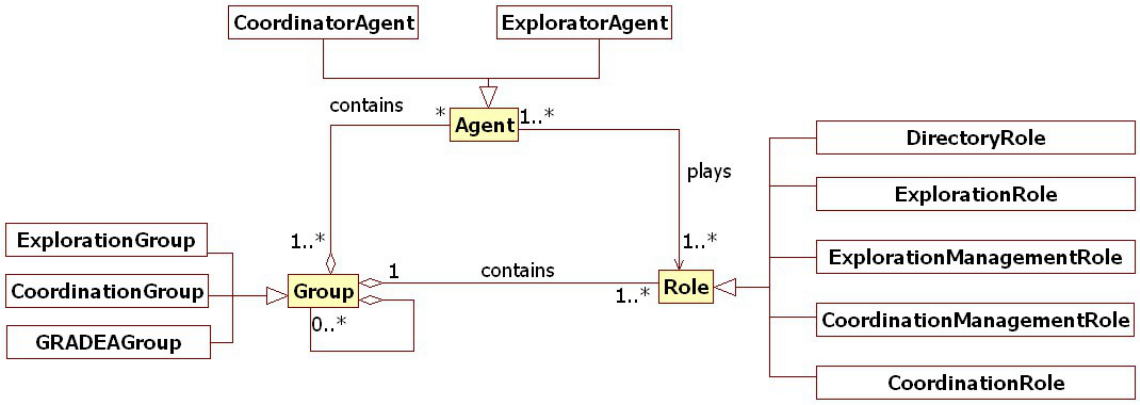

Figure 5.6 – Architecture conceptuelle de GRADEA

5.4.1 Rôles

Les deux agents de GRADEA peuvent jouer plusieurs rôles. Un agent n'est pas capable de jouer tous les rôles mais peut jouer quelques uns d'entre eux. Il y a deux type d'agents : l'agent Coordinateur et l'agent Explorateur et quatre rôles : ExplorationRole, ExplorationManagementRole, CoordinationRole et CoordinationManagementRole. Les agents Explorateur ne

peuvent que jouer le rôle "ExplorationRole", tandis que les agents de Coordination peuvent jouer trois rôles : CoordinationRole, ExplorationManagementRole et CoordinationManagementRole (voir la figure 5.7).

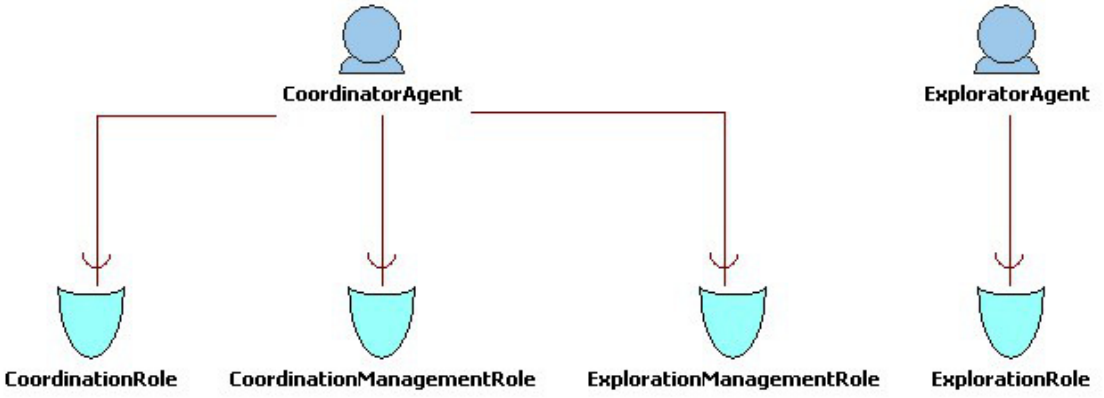

Figure 5.7 – La relation Agent-Rôle de GRADEA

Le rôle ExplorationRole

Quand l'agent d'Explorateur joue ce rôle, il s'équipe la capacité de réaliser une processus d'exploration avec une algorithme d'exploration. Ce rôle (figure **??**) contient les comportements ExecutionBehavior, FabricBehavior et TerminationBehavior. Ces trois comportements se combinent pour compléter le fonctionnement d'un algorithme d'exploration : initialiser les candidats de solution, exécuter les évaluations des candidats et terminer l'algorithme quand les conditions d'arrêt sont satisfaites. Ces trois comportements sont décrit dans le tableau 5.1.

comportement	description
FabricBehavior	charger des configurations et instancier une phase d'exploration selon les paramètres récupérés. Pour cela, il traduit les paramètres utiles à l'exécution d'un opérateur à l'aide d'une configuration initiale ou d'une mise à jours pendant l'exécution.
ExecutionBehavior	réaliser le processus principale de l'exploration d'un algorithme d'exploration, par exemple : l'exploration d'une région prometteuse ou le raffinement d'une solution
TerminationBehavior	identifier les conditions d'arrêt spécifiques à chaque agent Explorateur. En fait, ces conditions sont également conçu comme les autres configurations.

Table 5.1 – Les comportements du rôle ExplorationRole

Le rôle ExplorationManagementRole

Le rôle ExplorationManagementRole est responsable d'observer les comportements des algorithmes d'exploration, tracer leur trajet, enregistrer les informations nécessaires et établir les rapports aux coordinateurs. Il est également responsable d'envoyer les actions aux opérateurs des algorithmes d'exploration (cf ExplorationRole). Le rôle ExplorationManagemen-

tRole contient deux comportements : ObservationBehavior et OrderingBehavior. Ces deux comportements sont décrits dans le tableau 5.2.

comportement	description
ObservationBehavior	il permet aux utilisateurs, d'une part, de définir les mesures associées à l'exécution des algorithmes et à l'algorithme lui-même tels que : temps de calcul, les meilleurs solutions, les performances d'un opérateur, etc. D'autre part, de faire les analyses statistiques.
OrderingBehavior	il est responsable de traduire les demandes des rôles CoordinationRole en actions pour être compris par les ExplorationRole qui réaliseront ces actions en s'appuyant sur leur opérateur. Il envoie ces demandes sous forme d'un message. Les types de demande se compose : RegisterAction qui demande l'enregistrement un queue de messages et l'identifiant d'un service de recherche qu'il fournit. ConfiguringAction qui demande l'amélioration des configurations actuelles et ExploreAction qui demande l'exploration d'une région, d'une solution qu'il envoie.

Table 5.2 – Les comportements du rôle ExplorationManagementRole

Le rôle CoordinationManagementRole

Le rôle de CoordinationManagementRole est responsable de fournir un outil pour partager les informations entre les agents Explorateur. Cet outil est capable de percevoir l'environnement du système. Ça veut dire que il perçoit l'espace de solutions, l'espace des solveurs et l'environnement de calcul. Il se compose de deux comportements : BlackboardBehavior et SolutionContainerBehavior. Ces deux comportements sont décrits dans le tableau 5.3.

comportement	description
BlackboardBehavior	il est responsable de fournir un moyen de d'écrire les renseignements utiles sur le tableau noir (les paramètres de l'algorithme d'exploration, les états sauvegardés et les évaluations de performance des algorithmes d'exploration). Ces informations arrivent des observations récupérées par les agents jouant le rôle ExplorationManagementRole.
SolutionContainerBehavior	Il gère un ensemble de candidats de solution et fournit des candidats aux agents jouant ExplorationRole. Il reçoit les solutions à partir des agents jouant le rôle ExplorationRole pour le chargement ou le stockage des solutions. Il y a des conditions d'acceptation définis par les CoordinationRole pour contrôler le changement dedans le dépôt des solutions.

Table 5.3 – Les comportements du rôle CoordinationManagementRole

Le rôle CoordinationRole

Le rôle CoordinationRole est responsable de gérer les politiques de contrôle de la coordination. La politique est sous la forme : "si *conditions* alors *actions*". Ce role est aussi responsable d'observer et d'annoncer l'évolution de l'état de l'algorithme d'exploration pour appliquer les

politiques de contrôle adaptées. Il contient deux comportements : PolicyBuilderBehavior et ObservationBehavior dont le tableau 5.4 détaille

comportement	description
PolicyBuilderBehavior	est responsable de définir et appliquer les politiques de contrôle de la coopération entre les explorations dans un moteur de règles.
ObservationBehavior	met en ouvre un outil d'observation des changements qui s'intéresse une politique. Le but est d'annoncer le changement à temps pour déterminer les politiques de contrôle.

Table 5.4 – Les comportements du rôle CoordinationRole

le rôle DirectoryRole

Le rôle DirectoryRole est responsable du chargement des différents agents et de ses services. Ce rôle est joué par un agent indépendant avec les agents Explorateur et les agents Coordinateur. Ces comportements sont détaillés dans le tableau 5.5.

comportement	description
AgentFacilitatorBehavior	le répertoire des queues de messages des agents
ExplorationServiceFacilitatorBehavior	le répertoire des algorithmes d'exploration
ComputingServiceFacilitatorBehavior	le répertoire des services de calcul
CoordinationPolicyFacilitatorBehavior	le répertoire des politiques de contrôle
ConfigurationFacilitatorBehavior	le répertoire des configurations (paramètres de l'algorithme d'exploration)

Table 5.5 – Les comportements du rôle DirectoryRole

5.4.2 Groupes

GRADEA propose trois groupes de base à partir duquel il est possible d'établir des stratégies d'exploration. Nous explicitons ces trois groupes dans la suite de la section.

ExplorationGroup

Le groupe ExplorationGroup 5.8 est un groupe composés d'agent dotés de rôles de deux types : le rôle ExplorationRole et le rôle ExplorationManagementRole. Ce groupe permet de regrouper les agents jouant le rôle Explorateur avec un même ensemble de conditions initiale, même ensemble de politiques de contrôle de la coordination mais différents opérateurs d'évolution de l'algorithme (par exemple : différents opérateurs de mutation et croisement dans l'algorithme génétique).

CoordinationGroup

Le groupe CoordinationGroup 5.9 est un groupe composés d'agent dotés des rôles CoordinationManagerRole et CoordinationRole. De cette façons, ce groupe vise à définir les outils,

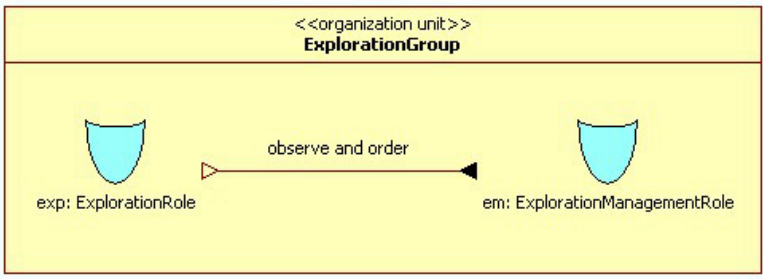

Figure 5.8 – le groupe d'ExplorationGroup

modifier les scénarios de coordination en utilisant les politiques de contrôle sur les connaissances extraites du groupe ExplorationGroup. Le scénario de coordination est effectué de façon autonome et adaptée quand les politiques sont réalisées.

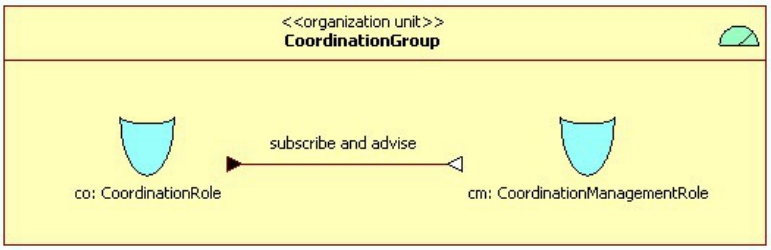

Figure 5.9 – le groupe de CoordinationGroup

GradeaGroup

Le groupe GradeaGroup 5.10 est un groupe composés d'agent dotés du rôle DirectoryRole et appartenants au sous groupes ExplorationGroup et CoordinationGroup. Ce groupe est responsable d'organiser la structure de coordination. Il fournit les outils de communication entre les agents et le mécanisme d'enregistrement les agents.

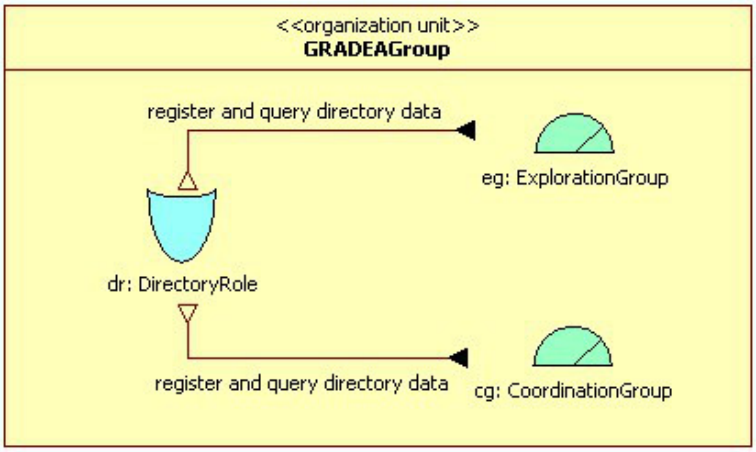

Figure 5.10 – le groupe de GradeaGroup

5.4.3 Interactions

Pour faire l'échange des connaissances entre les agents d'exploration, nous utilisons la technique Blackboard [Cor91]. Dans ce cas, le dépôt de solutions et de configurations assure le vecteur coopération entre les agents. Une communication directe entre les agents est analogue aux instanciations. La façon pour l'agent d'emettre et recevoir des messages est prédéfinit. La similitude entre les méthodes utilisées favorise la coopération entre les agents.

L'interaction dans une example de coopération centralisée dans GRADEA

Nous reutilisons l'exemple de coopération centralisé que nous avons présenté dans la partie de discussion 4.3.4 du chapitre précédent. L'interaction dans la coopération centralisée entre les agents ayant les rôles ExplorationRole, ExplorationManagerRole, CoordinationRole et CoordinationManagerRole est présentée dans la figure 5.11. Il y a trois agent ARG, ARL et ARC jouant le rôle de ExplorationRole. Il y a aussi un agent de Coordinateur qui joue trois rôles ExplorationManagerRole, CoordinationRole et CoordinationManagerRole.

L'interaction entre ces agents se déroule en plusieurs étapes comme suit :

- (1, 3, 5) : L'agent de Coordinateur jouant le rôle ExplorationManagerRole. Il crée une observation sur trois agents d'Explorateur jouant le rôle ExplorationRole et demander les candidats de solution.
- (2, 4, 6) : Les agents d'Explorateur initialisent les algorithmes d'exploration et démarrent.
- (7, 8) : Grâce à l'observation que l'agent de Coordinateur a impliqué, il a réçu les candidats de solution à partir des agents d'Explorateur. Il met à jours le containeur de candidats et le tableau noir.
- (9, 10, 11) : A partir des mis à jours, l'agent de Coordinateur prend la décisions et envoyer les demandes avec les candidats de solution correspondants à l'agent d'Explorateur à qu'il veut réaliser la demande.
- (12, 13, 14, 15, 16, 17) : L'agent jouant le rôle ExplorationRole atteindre la condition d'arrêt.

5.5 Exemples d'organisations

Cette section vous représente des exemples de coopération en utilisant l'organisation présentée dans la section précédent.

5.5.1 Coopération centralisée

Cette modélisation est dans la figure 5.12. Il y a un seul agent de CoordinatorAgent jouant le rôle de CoordinationManagementRole dans un groupe de CoordinationGroup pendant qu'il joue plusieurs rôles d'ExplorationManagementRole dans différents groupes de ExplorationGroup. Cela signifie qu'il se compose d'une seul population commune de candidats et d'un ensemble sous-algorithmes de bas niveau. Il y a aussi un seul tableau noir pour contenir les connaissances.

De cette façons, les algorithmes d'exploration peuvent être différents mais doivent être un niveau de similarité pour retirer le même type de connaissances. L'idée est que tous les sous-algorithmes envoient leurs solutions trouvées à l'agent unique selon une politique gérée par un autre agent jouant le rôle CoordinationRole avec un stratégie d'acceptation. La stratégie de réception détermine si la nouvelle solution obtenue devrait remplacer la solution de la population mère.

Un autre agent jouant le rôle CoordinationRole. Il est responsable de comprendre les connaissances dans le tableau noir selon sa politique et appliquer, par exemple, une stratégie de sélection pour essayer de sélectionner le sous-algorithme méta-heuristique la plus appropriée en quelques itérations.

Le sous-algorithme sélectionné est alors appliqué avec les solutions candidates extraites de la population commune. Cette stratégie a été considérée comme prometteuse puisque chaque entité peut utiliser des opérateurs uniques au niveau le plus bas. C'est utile pour traiter l'espace de recherche spécifique. En outre, l'utilisation de l'information globale permet la propagation de l'information entre les différentes entités.

5.5.2 Coopération décentralisée

Contrairement à l'organisation précédentes, il y a plusieurs agents de CoordinatorAgent jouant le rôle de CoordinationManagementRole dans un groupe de CoordinationGroup (voir figure 5.13). Ainsi, il y a une population de solution candidates pour chaque groupe de ExplorationGroup. Il y a aussi plusieurs tableaux noirs pour contenir les connaissances de chaque sous organisation. De cette façons, les algorithmes d'exploration peuvent être totalement variés et indépendants de leur comportement d'adaptation.

L'idée est que tous les sous-algorithmes partagent connaissances et les solutions via un agent dédié ayant un rôle de coordination (CoordinationRole). Chaque agent jouant le rôle de CoordinationManagementRole est contrôlé par un autre agent de CoordinationRole qui possède une stratégie propre d'acceptation des solutions et de traduction des connaissances.

Cette organisation détermine comment améliorer et faire évoluer la recherche sans se préoccuper des autres. Chaque groupe est une métaphore d'agents autonomes, apprenants et coopérants, en vue de faire évoluer leurs solutions candidates. De cette façon, le système de résolution repose sur la distribution et la décentralisation du contrôle pour obtenir un comportement collectif de recherche.

Discussion

Dans ce chapitre, nous avons présenté l'architecture conceptuelle du GRADEA. Il s'agit d'une architecture multi-agents pour l'exploration coopérative par l'utilisation des algorithmes d'exploration.

Dans cette proposition, une famille des agents homogènes ou hétérogènes s'exécutant dans un cadre centralisé ou/et décentralisé sont définis. Ils agissent selon les résultats obtenu par l'exploration de l'espace de recherche et par la mesure des consommation en ressources de calcul.

De plus, les agents sont capables de communiquer les uns avec les autres via les stratégies de

coordination et d'apprentissage ce qui facilite le comportement coopératif qui tend à atteindre un objectif commun.

L'architecture de GRADEA définit un cadre, qui permet à plusieurs algorithmes d'être utilisés pour résoudre de manière coopérative et de manière flexible un même problème inverse. La conception et l'ajout de nouveaux agents est facilitée.

De plus GRADEA offre la possibilité d'utiliser les techniques de calcul parallèle et/ou de calcul distribué.

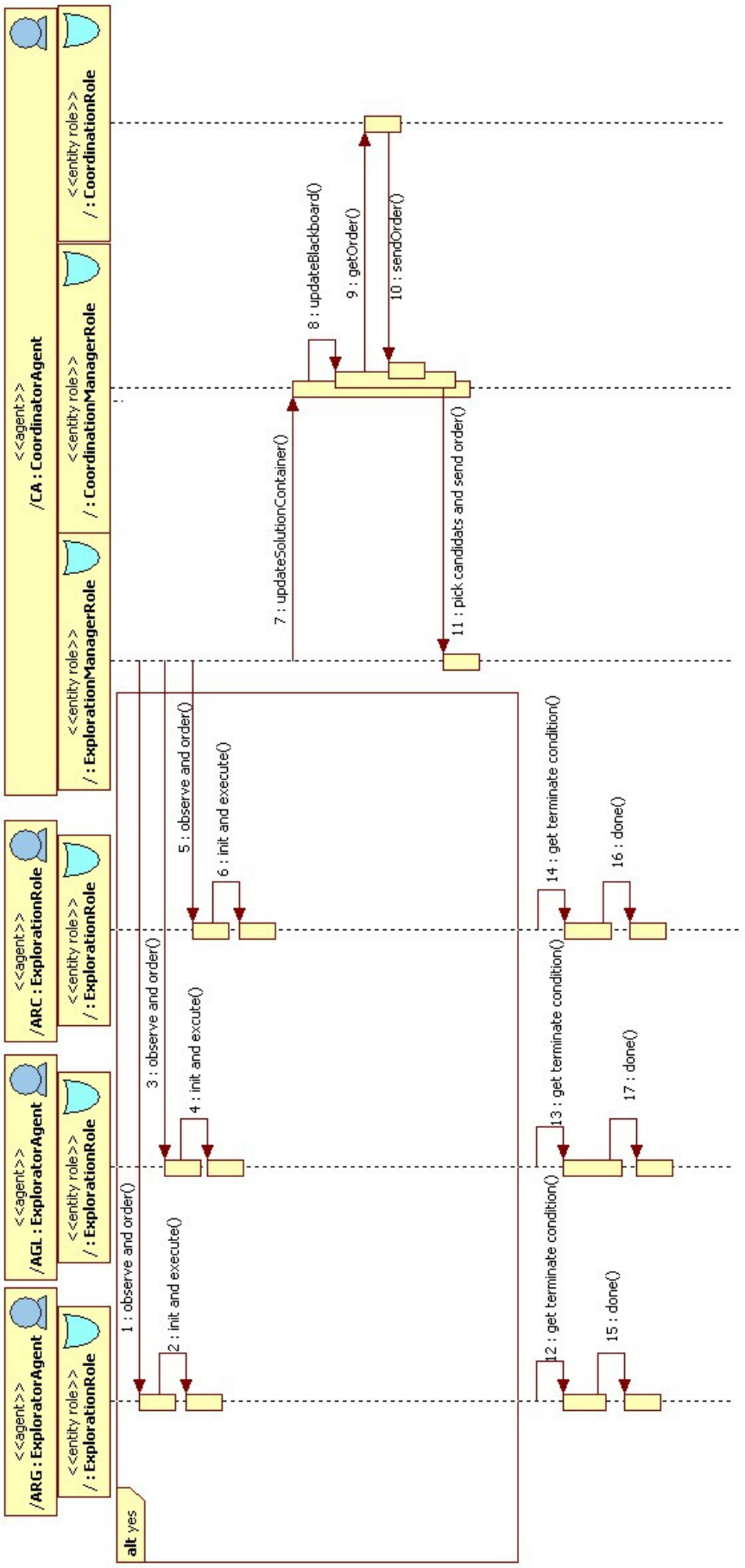

Figure 5.11 – L'interaction entre les agents dans la coopération centralisé dans GRADEA

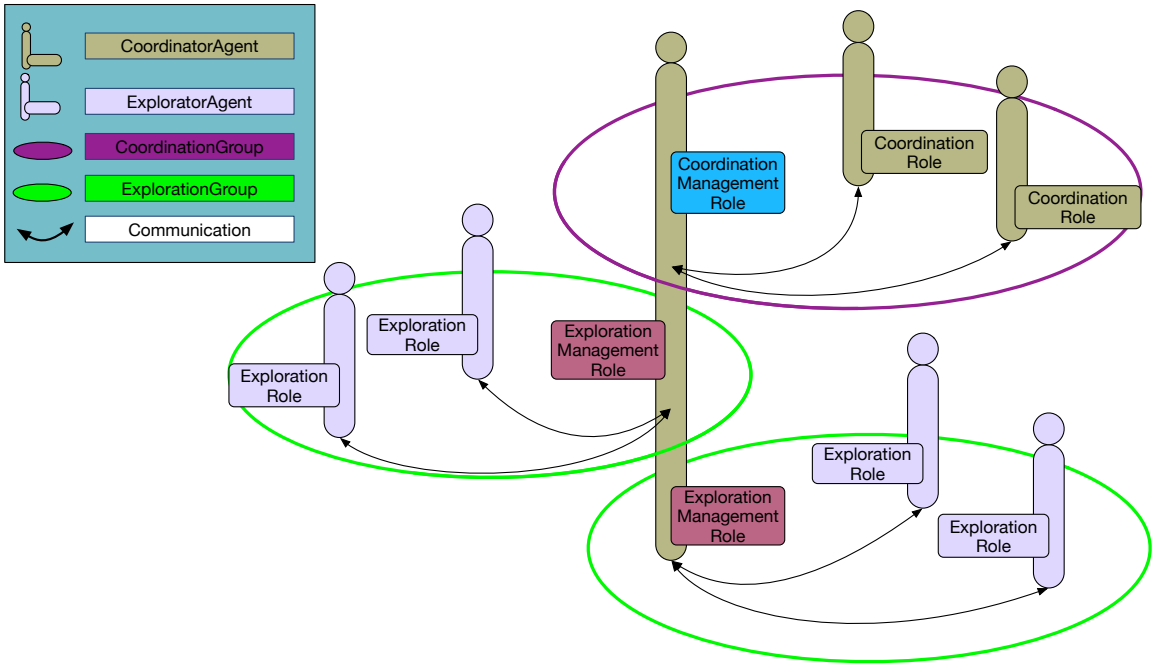

Figure 5.12 – Exemple de modélisation de la coordination centralisée utilisant GRADEA

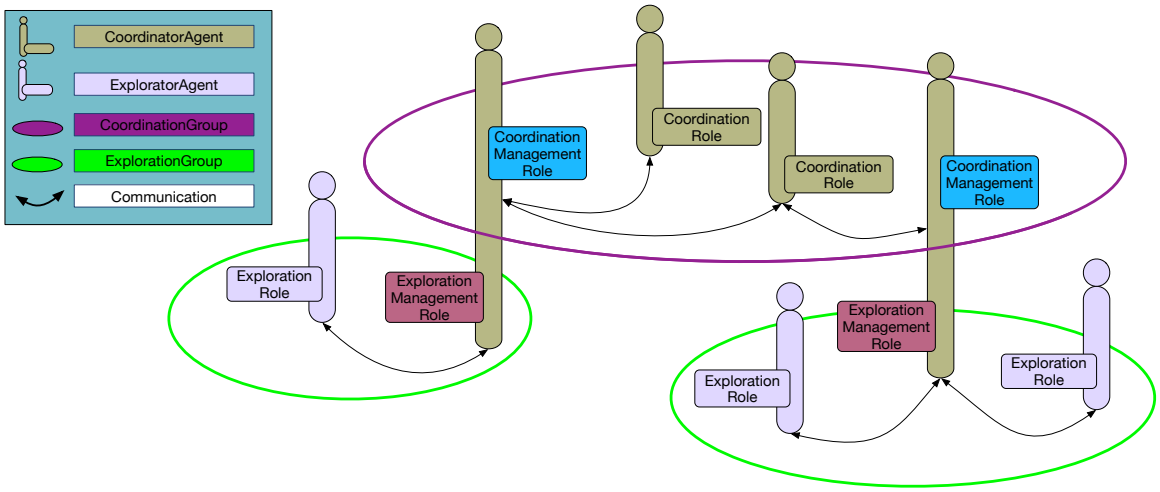

Figure 5.13 – Exemple de modélisation de la coordination décentralisée utilisant GRADEA

Chapitre 6

Plate-forme GRADEA

Sommaire

Ce chapitre présente l'implémentation précise de GRADEA à partir de la description conceptuelles établie dans le chapitre 5. cette dernière se compose d'une architecture logicielle de GRADEA et d'un guide d'implémentation des nouveaux modules dans GRADEA. L'architecture logicielle repose sur une architecture modulaire des stratégies d'exploration et de calcul. Après ça, on explique comment intégrer les nouveaux modules d'exploration et modules de calcul sur GRADEA. Le chapitre comporte trois sections : la modularisation dans GRADEA, l'infrastructure logicielle de GRADEA et l'utilisation de GRADEA.

6.1 Une plate-forme modulaire

6.1.1 Une architecture distribuée à base de composants

La caratéristique la plus importante de GRADEA est la modularité. Cette modularité est permise grâce à une architecture adaptée présentée dans la figure 6.1.

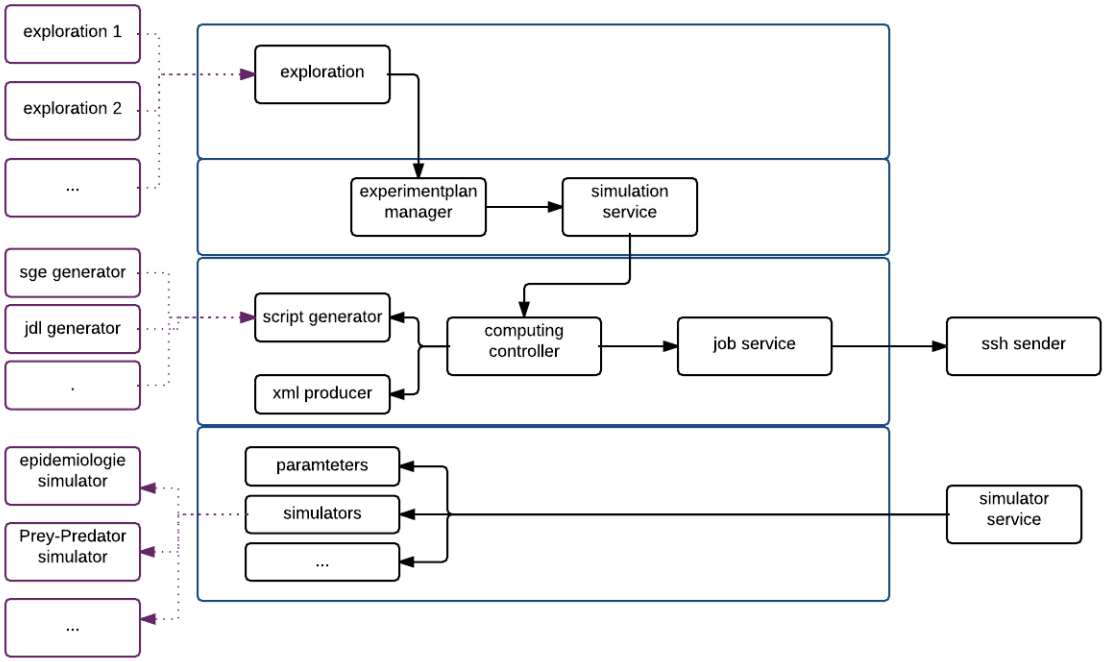

Figure 6.1 – Les détails des modules dans GRADEA

— le module «Experimentplan Manager »génère les plans d'expériences.

— le module «Simulation Service»permet de lancer, arrêter, supprimer une simulation et d'en consulter l'état.

— le module «Computing Controle»se compose des services Script Generator et XML Producer. XML Producer génère les fichiers de données XML qui décrive toutes les simulations possibles. Script Generator est responsable de produire le fichier de SGE, PBS, LSF ou d'autre fichier dépendant de la plate-forme physique de calcul intensif.

— le module «Job Service »est une interface de base niveau qui permet la gestion des jobs lancés sur la plate-forme de calcul intensif. «Job Service »est une part à niveau bas de «Simulation Service»

— le module «SSH Sender »assure la connexion entre le serveur d'application et cluster (à distance) pour envoyer les fichiers, soumettre les jobs sur le cluster, suivre l'état des jobs et recueillir les données sur le serveur d'application.

— le module «Simulator Service »permet l'upload des simulateurs exécutables sur le cluster pour une utilisation ultérieur.

— le module «Exploration »d'implémente les stratégies de l'exploration coopérative dans GRADEA que l'on a discuté dans le chapitre 4 et 5. L'implémentation du module est

représenté plus détail dans la section suivant.

De cette façon, l'utilisateur va suivre le workflow suivant pour utiliser GRADEA :

— utiliser un simulateur existant ou ajouter un nouveau simulateur en utilisant le "simulator service". Il se compose d'upload la source exécutable du simulateur, déclarer et sauvegarder les paramètres d'entrée et les paramètres de sortie du modèle, récupérer les donées dans la base de données.
— choisir les sources de calcul ou ajouter de nouvelles sources de calcul en utilisant les sous services du "computing controller" : implémenter l'interface "job service", définir le la générateur du script en utilisant "script generator" et utiliser "xml producer" pour générer les fichiers de paramètres.
— choisir un stratégie d'exploration existant ou intégrer les nouveaux stratégies d'exploration.

6.1.2 Ajouter de nouveaux simulateurs

Le module "Simulator Service" fournit une interface d'implémentation pour ajouter les simulateurs sur le serveur et aussi que sauvegarder les informations des paramètres dans une base de données.

— *uploadSimulator* permet de transférer un simulateur à un répertoire. Après avoir utilisé SCP pour envoyer le simulateur sur le cluster, qu'il faut mettre à jour la base de données.
— *uploadSimulatorFolder* est utilisé dans le cas où le simulateur se compose de plusieurs fichiers.
— *getSimulator* permet de consulter le simulateur disponible. Cette méthode liste tous les simulateurs disponibles pour un utilisateur.

L'utilisateur peut choisir le fichier qui contient l'exécutable pour upload sur cluster (un fichier ou un zip, si il est un zip, le serveur d'applications va décompresser sur le serveur). Il décrit ensuite les paramètres et les sorties du simulateur. Ses paramètres et ses sorties seront sauvegardés dans la base de données. Sur le cluster, les plates-formes prises en charge sont installés en tant que bibliothèques standard à savoir dans le répertoire du logiciel. Une fois que le fichier de modèle ou le simulateur est téléchargé sur le serveur d'applications, il est dans le répertoire avec les bibliothèques standard du plates-forme correspondant pour fournir à l'utilisateur un environnement d'exécution propre.

6.1.3 Ajouter de nouvelles ressources de calcul

Pour ajouter une nouvelle ressource de calcul, par exemple un cluster utilisant PBS au lieu de SGE (vu dans la partie 4.3.2), il faut réaliser les deux étapes suivantes :

1. Implémenter un module héritant de "Script Generator" afin de construire les scripts PBS/Torque comme ceux présentés dans la (figure 6.2) :
2. Implémenter un module héritant de "Job Service" et leur fonctions afin de tenir compte des commande de PBS/Torque. Les fonctions à implémenter sont : (1) soumettre un job. cette méthode prend en paramètre le chemin du script en format dessus et retourne l'identifiant du job. (2) soumettre un job avec un script sur une machine locale. (3) qui extrait et retourne la liste de tous les identifiants des jobs. (4) retourner l'état d'un job. (5) retourner l'état d'un ensemble de sous-Job (6) supprimer un Job en cours d'exécution.

```
### PBS/Torque
#!/bin/bash -l
#PBS -q normal2h
#PBS -t 1-100 -tc 20
#PBS -o \$PBS_JOB_NAME.\$PBS_JOB_ID.out
#PBS -e \$PBS_JOB_NAME.\$PBS_JOB_ID.err
./epis input\$PBS_ARRAYID.xml output\$PBS_ARRAYID.xml


### SGE
#!/bin/bash -l
#$ -q normal2h
#$ -t 1-100 -tc 20
#$ -o \$JOB_NAME.\$JOB_ID.out
#$ -e \$JOB_NAME.\$JOB_ID.err
./epis input\$SGE_TASK_ID.xml output\$SGE_TASK_ID.xml
```

Figure 6.2 – Le script pour le système PBS/Torque

6.2 Implémentation d'un module d'Exploration

Cette couche implémente le système multi-agents décrit dans le chapitre précédent (5) permettant de guider l'exploration. Le SMA est en étroite interaction avec le *module de contrôle des expérimentations* («Experimentplan Manager »dans la figure 6.1) à la fois pour ordonner l'exécution des simulations et pour recevoir les résultats des simulations antérieures.

Nous vous proposons dans cette partie une architecture des agents que GRADEA a utilisé pour concevoir les agents, une architecture logicielle modulaire pour implémenter les agents et une architecture des services pour la communication.

6.2.1 Architecture des agents

GRADEA repose sur quatre niveaux organisationnelles 6.3.

— Le niveau *Organisation* correspondant aux différents groupes pour lesquels peuvent participer les agents. Il y a trois groupes dérivant directement du méta-modèle : DirectoryGroup, ExplorationGroup et CoordinationGroup.

— Le niveau *Communication* contient les protocoles de communication entre les agents. Il se compose des mécanismes d'enregistrement, de partage des connaissances, d'envoi des demandes d'exploration et des demandes de changement de configurations.

— Le niveau *Comportement* contient une implémentation du comportement pour les différents rôles : ExplorationRole, CoordinationRole, ExplorationManagementRole et CoordinationManagementRole.

— Le niveau *Environnement* contient l'espace de solution, l'espace d'état des explorations et l'espace des ressources de calcul. L'espace des solutions est pour générer les jeux de paramètres. L'espace d'état des explorations se compose des configurations actuelles de l'algorithme, des meilleures solutions trouvées, des régions (ou points) prometteuses et des régions prohibitives qu'un agent de coopération peut utiliser pour percevoir

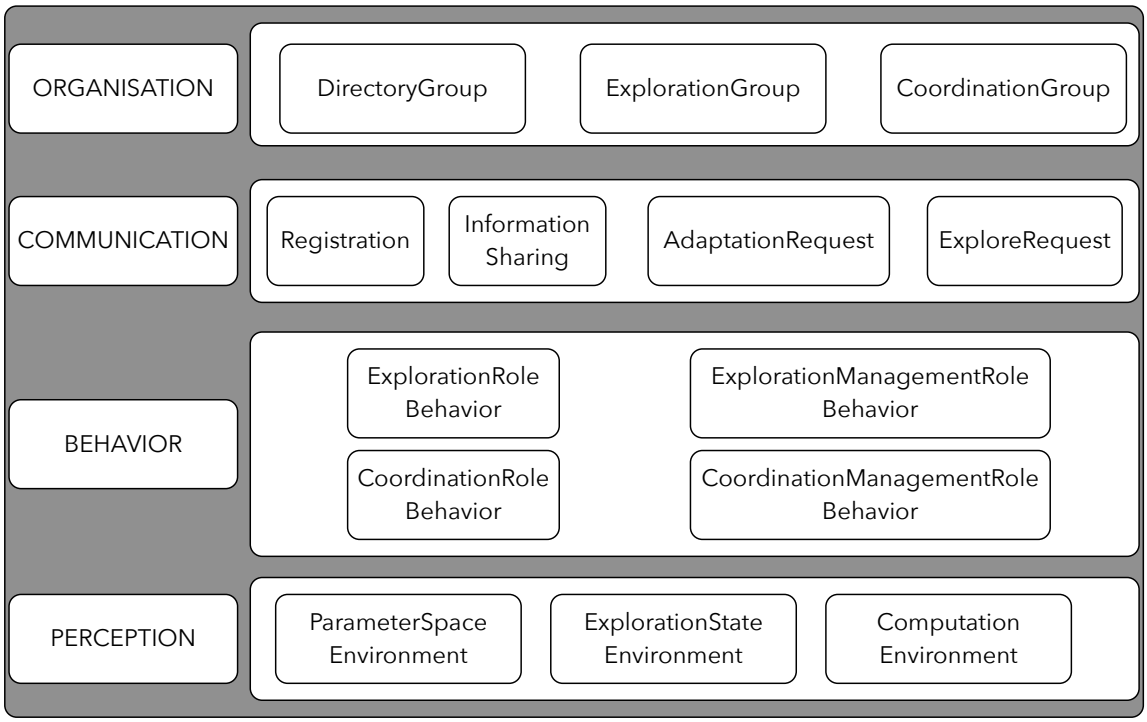

Figure 6.3 – Les couches en terme des fonctionnements

leur environnements de travail (l'espace de recherche). L'espace de ressources de calcul caractérise les configurations qui définissent la capacité de calculs dans laquelle GRADEA est lancé.

6.2.2 Architecture logicielle modulaire

A partir de la vue conceptuelle, on traduit naturellement une structure modulaire du SMA. On peut voir qu'il y a 8 modules dans GRADEA 6.4.

— Le module *AGENT* : ce module contient les implémentations de tous les agents. Il contient une classe abstraite qui est rafinée par les différentes implémentations d'agents. Elle contient un attribut pour représenter l'identifiant unique de l'agent et se distinguer de l'ensemble dans le système entier. La valeur de cet attribut doit être attribué par un mécanisme lors de l'enregistrement. Cet agent contient un écouteur pour recevoir de façons asynchrone les informations envoyées par les autres agents.

— Le module *EXPLORATOR_AGENT* : ce module contient l'implémentation de l'agent d'exploration. Il contient les paramètres de l'algorithme et les paramètres du problème pour définir son comportement.

— Le module *COORDINATOR_AGENT* : ce module contient l'implémentation de l'agent coordination. Il est constitué par les services d'exploration et les files de messages en provenance des agents d'exploration.

— Le module *MESSAGE* : ce module contient l'implémentation des messages. Un message se compose de l'identifiant du message, de l'identifiant de l'agent émetteur et de l'identifiant de l'agent de récepteur. Différentes implémentations de message sont

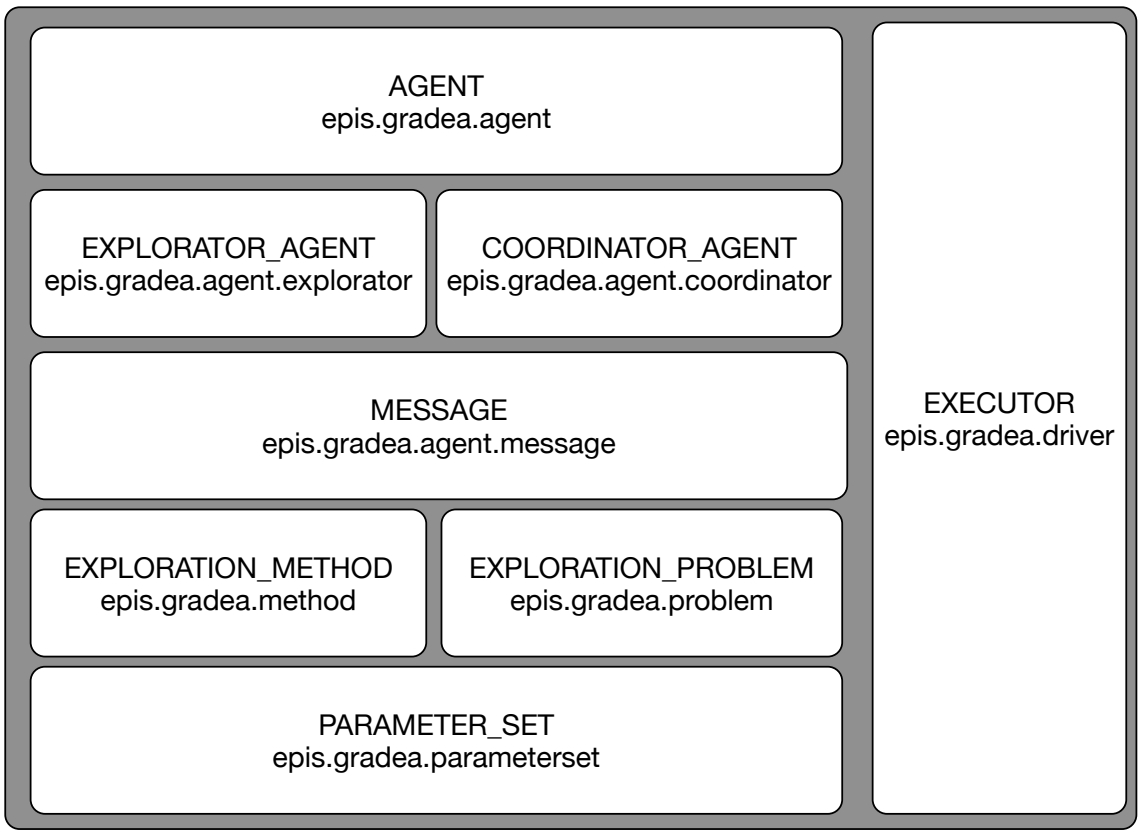

Figure 6.4 – Les couches en terme des implémentions

nécessaires pour préciser le type de message.

— Le module *EXPLORATION_METHOD* : ce module contient l'implémentation des opérateurs d'exploration d'un algorithme d'exploration. Un algorithme va être attribué aux agents d'exploration pour préciser les services que cet agent fournit.

— Le module *EXPLORATION_PROBLEM* : ce module contient l'implémentation pour retirer les informations du simulateur complexe que l'on veut explorer. Il va être attribué aux agents d'exploration et aux agents de coordination pour préciser le problème qu'il faut résoudre.

— Le module *SOLUTION* : ce module contient l'implémentation des solutions candidates d'un problème de résolution. Chaque solution candidate est liée à un jeu de paramètres.

— Le module *EXECUTOR* : ce module contient l'implémentation des services d'exécution. Il lance les exécutions des agents et les exécutions des évaluations d'un jeu de paramètres.

6.2.3 Algorithme de découverte des services

Le cycle de vie des services d'exploration sont gérées en se basant sur la conception d"une architecture orientée services [BHM+]. Il se compose de trois grands rôles : le rôle d'un fournisseur de services, le rôle d'un consommateur de service (demandeur), et le rôle d'un répertoire de service.

— Le fournisseur de services permet l'accès aux services, crée une description d'un service et le publie au répertoire des service.

— Le répertoire de services accueille les descriptions des différents services. Il est responsable de la liaison d'un demandeur à un fournisseur de services.

— Le demandeur de service est responsable de la découverte d'un service en cherchant dans les descriptions des services fournis par le répertoire de services. Un demandeur est également responsable de la liaison à des services fournis par le fournisseur de services.

Ce triptyque montre qu'il existe un besoin d'un moyen de se rapprocher des fournisseurs et des consommateurs. Pour cette raison, dès qu'un fournisseur commence à fournir un service, il enregistre le service à un endroit connu (un répertoire des services).Quand un consommateur a besoin d'un service, il interroge ce répertoire afin de trouver des fournisseurs assurant le service requis.

Lors de l'exécution de GRADEA, il y a toujours un agent DirectoryAgent jouant le rôle DirectoryRole qui est responsable de la gestion des services d'exploration (i.e algorithme d"exploration) pour que les autres agents puissent les trouver. Un agent jouant le rôle d'exploration va inscrire au agent DirectoryAgent pour recevoir l'information de son identifiant, sa boîte des messages (unique pour se distinguer), décrire au DirectoryAgent le service d'exploration qu'il fournit aux autres agents.

6.3 Implémentation d'un algorithme exploration coopératif sur GRADEA

6.3.1 Implémentation d'un algorithme d'exploration

Un algorithme d'exploration dans GRADEA doit implémenter l'interface *Algorithm* et une classe d'abstraite *AbstractState* comme dans le diagramme UML 6.5. La classe d'abstraite *AbstractState* doit utiliser les classes suivantes : Initializer, Evaluator, Analyzer et Experimentplan (dans la figure 6.6). Initializer vise à initialiser les solution candidates et les paramètres de l'algorithme. Evaluator vise à evaluer les jeux de paramètres. Analyzer vise à analyser l'état actuel de l'algorithme. Experimentplan sert à définir les jeux de paramètres d'un simulateur précis.

Un jeu de paramètres est représenté par la classe ParameterSet pendant que JobCandidate est responsable de gérer un jeu de paramètres. Ce dernier est intégré à un objet Evaluator pour faire l'évaluation (6.7). Différent types de paramètres sont présentés par la classe Species. Le classe Fitness implémentant la fonction objectif a pour but de calculer la performance d'un jeu de paramètres à partir des résultats obtenus après le calcul.

6.3.2 Déploiement d'un algorithme d'exploration dans GRADEA

Pour que les agents puissent envoyer et recevoir des messages entre eux, il faut que un «Algorithm »soit appelé dans un «ExplorationServiceOptimization »comme dans le diagramme 6.8. Ce diagramme représente le diagramme classe UML des agents dans le plateforme GRADEA.

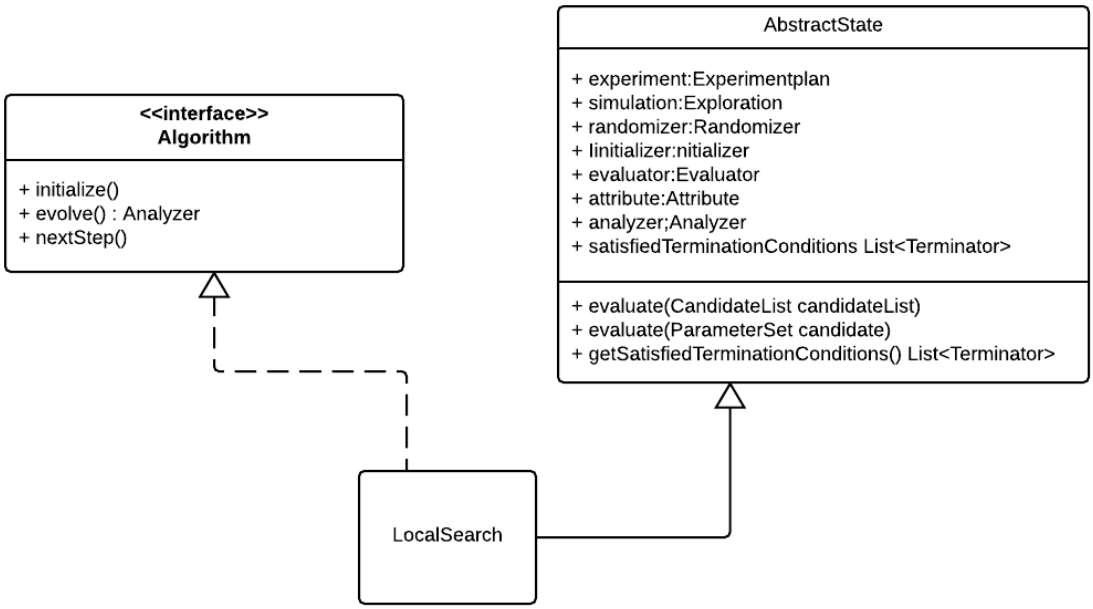

Figure 6.5 – Le diagramme classe de l'algorithme d'exploration dans GRADEA

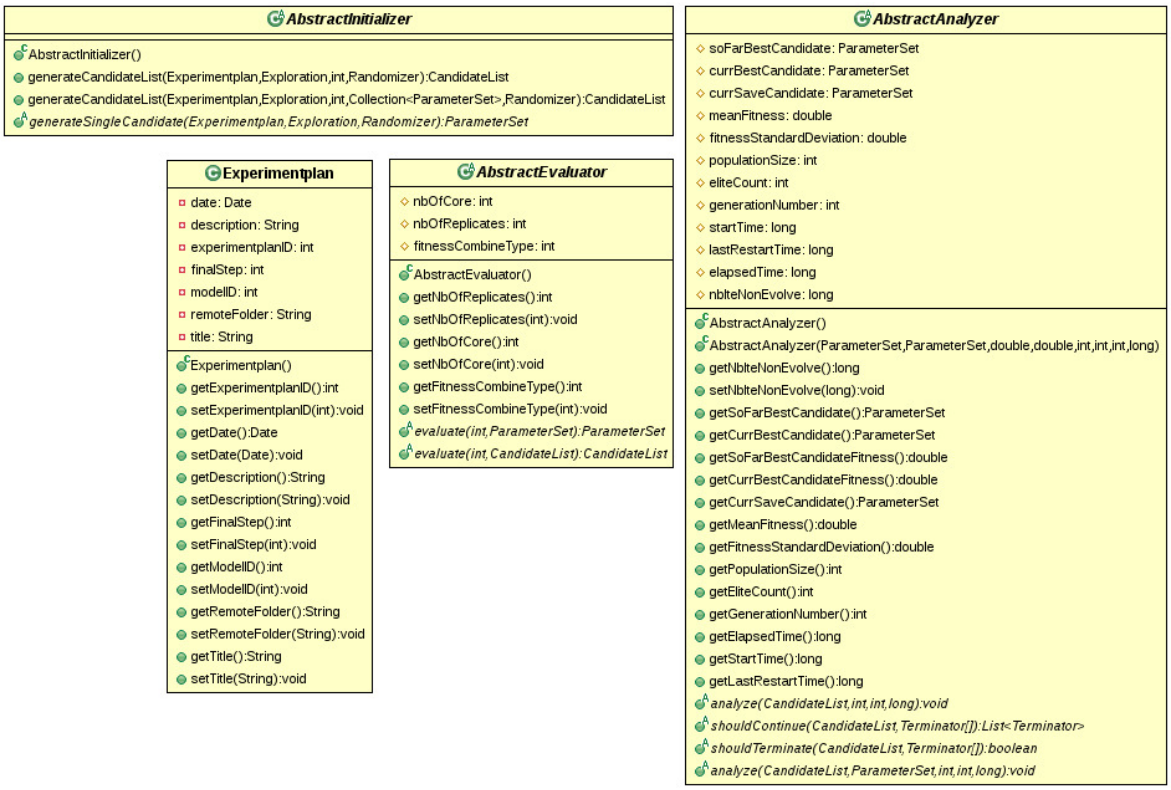

Figure 6.6 – Le diagramme classe des abstractions de Initializer, Evaluator, Analyzer, Experimentplan

A partir de l'abstraction Agent, on implémente deux classes : l'emetteur (AgentSender) et le récepteur (AgentReceiver). Deux interfaces indépendantes fournissent une infrastructure complète permettant la communication entre les agents.

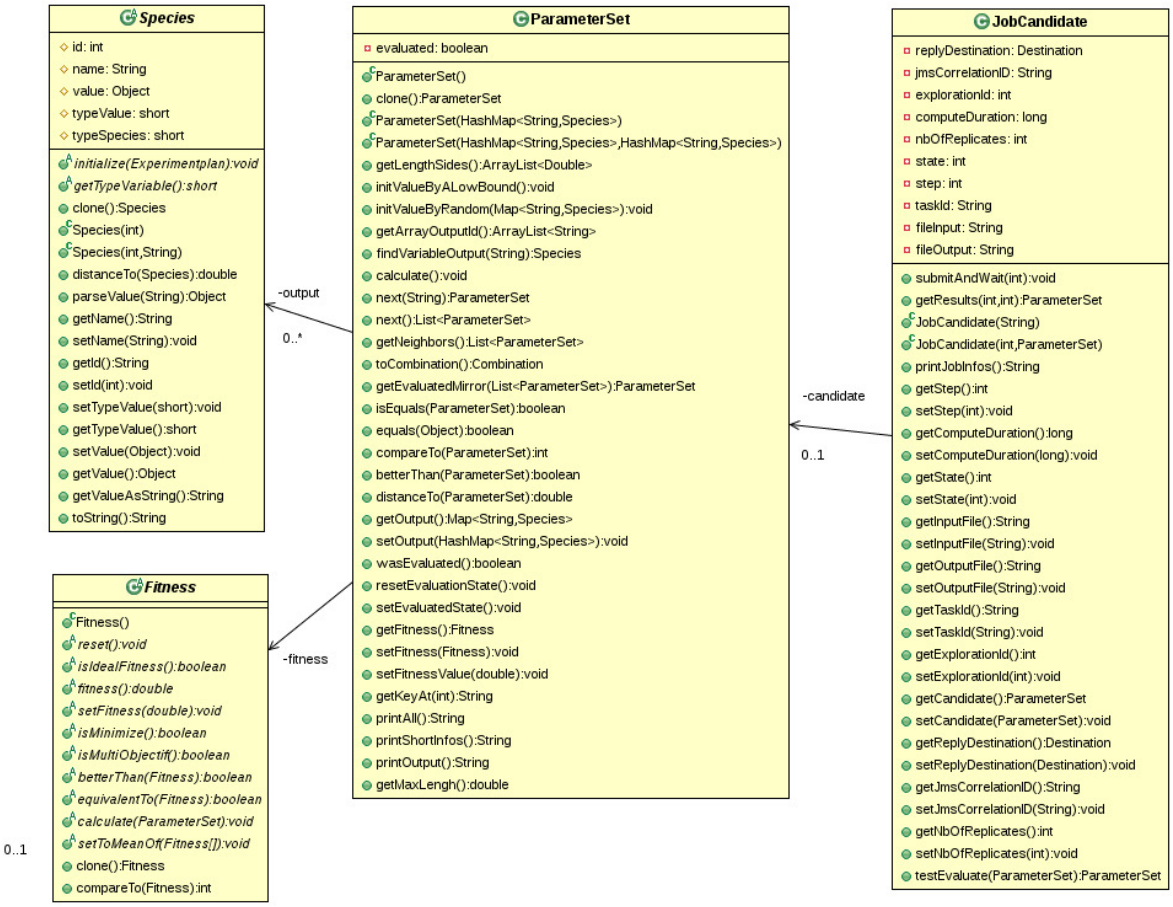

Figure 6.7 – Le diagramme classe des candidats de solution

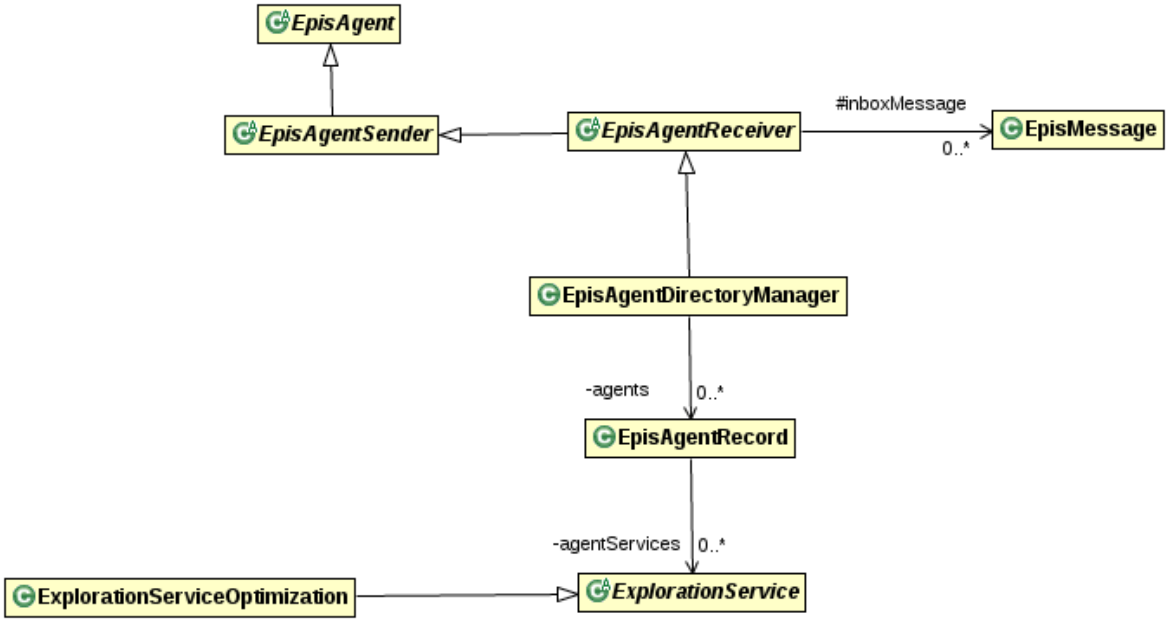

Figure 6.8 – Le diagramme classe des agents de GRADEA

Les agents envoient les messages entre eux. Chaque message contient un ensemble de jeux de paramètres ParameterSet ou les commandes.

6.3.3 Echange des connaissances via messages

Le diagramme 6.9 représente le diagrame de classes UML des messages dans le plateforme GRADEA.

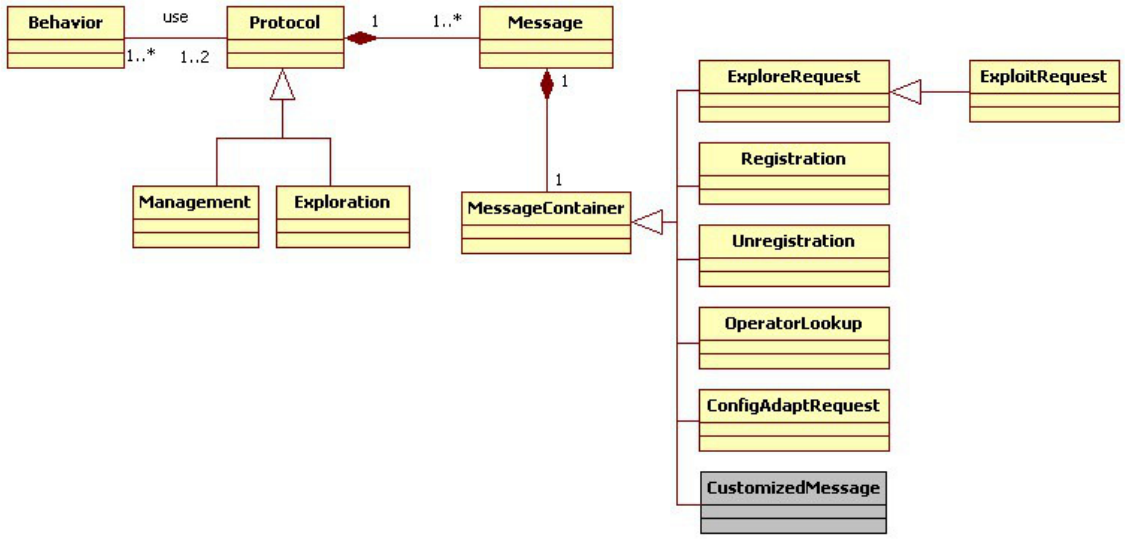

Figure 6.9 – Le diagramme classe des messages de GRADEA

La communication entre les agents est basée sur le message. Deux protocoles d'interaction différents sont spécifiés dans GRADEA : les messages d'enregistrement (Directory) et les messages d'exploration (Exploration).

Une classe de protocole est composé d'au moins un message. Le contenu d'un message est encapsulé dans la classe MessageContainer, qui peut contenir les demandes d'enregistrement, de recherche des services, d'adaptation des paramètres de configurations attachées ou d'adaptation des paramètres d'exploration d'un individu ou d'une région (avec un ensemble des individues attachées). Un protocole est utilisé par un rôle, c'est pourquoi il existe un lien entre le Protocole et les classes de comportement.

6.3.4 Exemple d'implémentation d'un algorithme évolutionnaire dans GRADEA

Le diagramme 6.10 représente le diagrame de classes UML des opérateurs de l'algorithme évolutionnaire dans la plateforme GRADEA.

La classe de l'opérateur représente une opération spécifiée dans l'algorithme d'optimisation. Il est le noyau du comportement ExecutionBehavior et l'élément manipulable du comportement. L'opérateur peut représenter un algorithme, un ensemble d'algorithmes ou un des opérateurs au sein d'une métaheuristique unique. Son utilisation est simple tout en étant expressif. La logique principale est basé sur l'abstraction d'un processus de transformation dans lequel il dispose une entrée, dans ce cas, la liste des solutions du problème à résoudre, un processus de transformation représentée par l'opérateur et une sortie représentée par la liste des solutions transformées. L'opérateur est alors une classe abstraite qui fournit une variété de fonctionnalités pour l'utilisateur. Ce dernier doit néanmoins être capable d'étendre ses fonctionnalités avec son algorithme doué d'une logique spécifique. L'opérateur dispose de

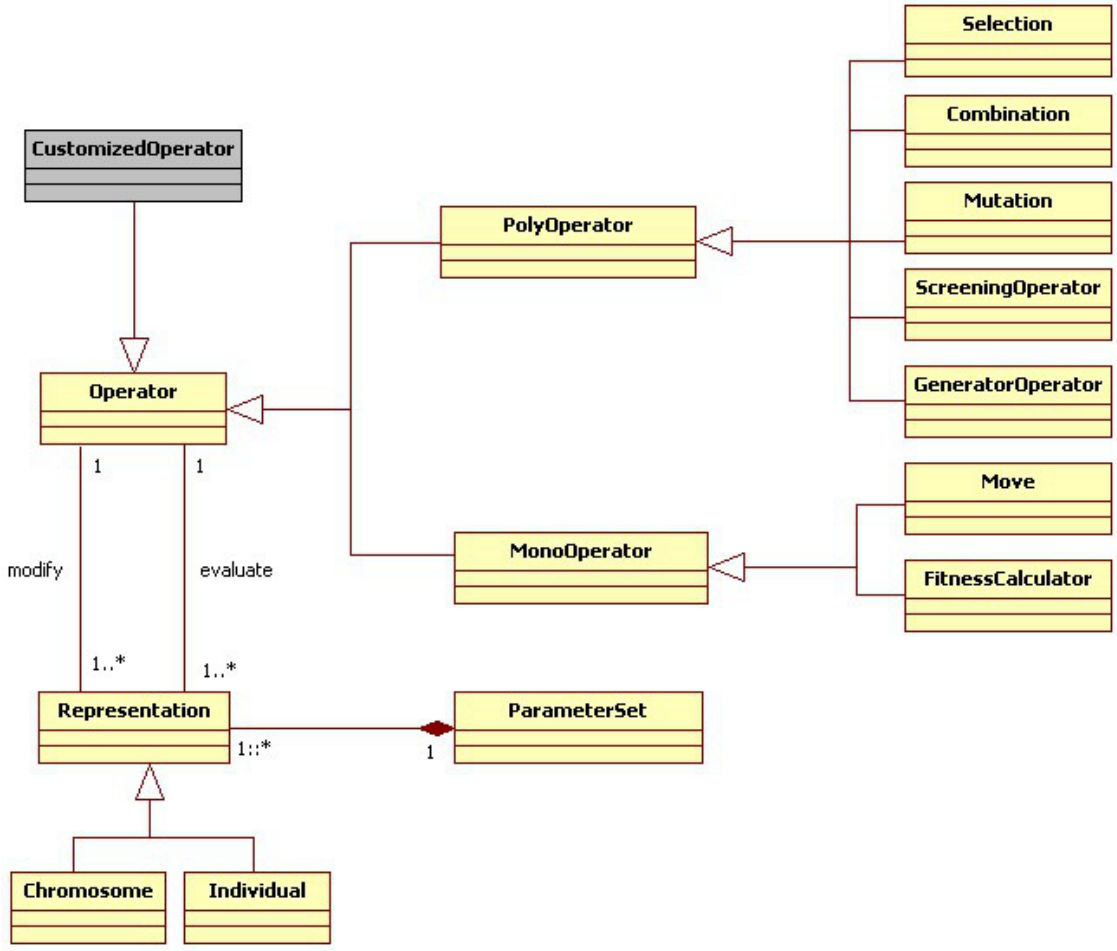

Figure 6.10 – Le diagramme classe des opérateurs de GRADEA

trois méthodes abstraites sur le tableau 6.1 :

capacité	description
execute(solution*)	Cette capacité est appelée par l'opérateur déléguant l'exécution à un prototype de computation qui peut effectuer la tâche
notifyCompletion(obj)	Cette capacité est appelée par une observation à chaque fois ou l'exécution d'un opérateur est terminée.
getResults()	Cette capacité renvoie les résultats de l'exécution si elles sont disponibles.

Table 6.1 – Les fonctionnalités de la classe Operator

Un jeu de paramètres dans GRADEA est représenté par la classe ParameterSet. Dépendent au problème de résolution et le type des opérateurs spécifiques, il faut encoder le ParameterSet en Representation pour que les algorithmes peuvent lire. Par exemple, pour les opérateurs de l'algorithme évolutionnaire, il faut encoder en Chromosome. Pour les recherche locales comme la recherche tabou, il faut encoder en Individual. Un ensemble de Representation est l'entrée et la sortie d'un opérateur.

La classe MonoOperator est une classe héritée d'Operator. Le MonoOperator est un opérateur qui effectue une action sur chaque objet dans la liste d'entrée de solution. D'une part, ce

type d'opérateur est utile pour la mise en œuvre des opérateurs d'évaluation de la fonction objectif (la classe héritée FitnessCalculator) qui est effectuée sur chaque solution séparément quand l'on veut distribuer la calcul de façons asynchrone sur différents ressources de calcul. Selon les paramètres de configuration des services de calcul, le MonoOperator délègue le calcul de l'évaluation aux services de calcul. Cette action est gérée par le comportement ComputingDelegateBehavior. Plusieurs possibilités des services de calcul sont le calcul local, le multithreading ou le calcul intensif. La raison de l'existence de MonoOperator est motivée par le fait de fournir une grande flexibilité de calcul. D'autre part, la mise en œuvre de cet opérateur est nécessaire pour les opérateurs de recherche locale afin de raffiner une solution. Ces types d'opérateur demandent que l'entrée et la sortie soit une seule solution candidate. Sa mise en œuvre est simple, car il est tout simplement responsable de diriger un objet de solution comme une entrée à la méthode execute() et d'attendre la méthode le résultat de notifyComplement() avant de récupérer l'objet de sortie en appelant getResult().

Le PolyOperator est un opérateur qui effectue une action sur une liste de solutions. Ce type d'opérateur est utile pour la mise en œuvre des opérateurs tels que la sélection, les opérateurs qui définissent la structure algorithmique de métaheuristiques (mutation, recombinaison), les opérateurs d'initialisation des candidats. Sa mise en œuvre est similaire MonoOperator sauf que l'entrée et la sortie est la liste des solutions. D'autre part, ce type d'opérateur est utile pour la mise en œuvre des opérateurs d'évaluation de la fonction objectif sur un ensemble de solutions quand l'on fait le calcul centralisée de façons synchrone l'un après l'autre.

Discussion

Ce chapitre décrit l'architecture de la plate-forme GRADEA et sa mise en œuvre en trois couches : la couche de Calcul, la couche de Soumission et la couche d'Exploration.

GRADEA propose une architecture suffisante pour combiner des algorithmes d'exploration. L'utilisateur établit alors une politique d'exploration par la définition d'un système multi-agents combinant comportement d'exploration et comportement de coordination. Ainsi, le système multi-agents est l'ordonateur d'un grand nombre de simulations qui sont exécutés sur les nœuds d'un cluster.

La plate-forme GRADEA assure la gestion d'un système multi-agents et propose des services de bas niveau : perception, communication et organisation dédiés au domaine de l'exploration de modèle. En outre, elle repose sur un moteur d'inférence qui gère le cycle de vie de l'agent et l'activation des différents modules internes qui le compose : les agents, les protocoles d'interaction, les comportements, les opérateurs d'exploration, les algorithmes et les problèmes de résolution.

Ainsi GRADEA est un environnement d'exploration de modèle complexe fortement modulaire dans la mesure où il permet de définir, au cas par cas, des stratégies d'exploration en : (i) combinant des algorithmes d'exploration, (ii) et en facilitant l'ajout de nouveaux algorithmes d'exploration.

Dans le chapitre suivant, nous présentons un cas d'application de la plate-forme GRADEA tiré du domaine de l'épidémiologie. Ce cas d'application nous permettra de mettre en avant l'intéret de la plate-forme vis à vis d'une problématique réelle. D'autre part, nous en évaluerons ses performances.

Chapitre 7

Application GRADEA : l'exploration du modèle de vaccination de la maladie rougeole au Vietnam

Sommaire

Dans ce chapitre, nous utilisons GRADEA pour évaluer le modèle de vaccination de la maladie rougeole au Vietnam et aussi que nous utilisons ce modèle pour évaluer la performance de l'exploration coopérative dans GRADEA. Ce chapitre se compose trois parties : La représentation du modèle de vaccination de la maladie rougeole au Vietnam, les configurations experimentales choisies et les résultats de l'expérimentation.

7.1 Modèle de vaccination de la maladie rougeole au Vietnam

Dans le chapitre 1, nous avons représenté quelques modèles pour étudier la dynamique de système complexe et l'usage dans la prise de décision. Cette section, nous concentrons sur un modèle précis en épidémiologie : Modèle de vaccination de la maladie rougeole au Vietnam.

7.1.1 Enjeux du modèle

Afin d'illustrer notre travail, nous avons choisi de nous intéresser à une problématique concrète et d'actualité : la propagation d'une épidémie en Asie du Sud Est en vue d'identifier des stratégies de vaccination de la rougeole. Dans ce cadre, un modèle stochastique a été développé à partir du modèle SEIR [KR08] en vue d'évaluer des stratégies de vaccination.

L'idée principale de modélisation de la propagation de maladie est que la population se décompose en quatre catégories libellées "S", "I", "E", et "R", autrement dit, des compartiments. Dans ce modèle, les compartiments interagissent entre eux selon certaines règles. Dans le modèle SIER, il y a les groupes suivants :

— Susceptible (S) : contient les individus sains, n'ayant pas encore été infectées mais pouvant potentiellement être infectés par la maladie. Si un nouveau individu est né, il est ajouté dans ce compartiment.
— Infectieux (I) : ce compartiment représente ceux qui, dans la population, sont, non seulement déjà infectés, mais également en pouvant propager la maladie.
— Exposé (E) : contient les individus exposés à la maladie mais n'en subissant pas les effets et ne sont pas immédiatement capables d'en contaminer d'autres.
— Guéri (R) : (comme «recovered»en anglais) après une période, les infectés guérissent et se déplacent dans la catégorie des guéris. Les individus géris peuvent acquérir une immunité temporaire ou totale.
— Vacciné (V) : contient les individus vaccinés dans la campagne de vaccination. Ils peuvent acquérir une immunité permanente. On fait en sorte que le nombre d'individus vaccinés va dans un compartiment qui s'appelle V pour vacciné et pas dans R, car il faut que l'on garde dans R que ceux qui viennent de I par l'immunité.
— Mort (X) : Mort comptabilise tous les morts depuis le début de la simulation. Il est utilisé afin de mettre en évidence ce phénomène lorsque ceci est jugé nécessaire.
— Nombre total (N) : ce compartiment regroupe plusieurs compartiments. Selon le contexte, la population dans son ensemble est supposée constante ou dynamique. On choisira une population dynamique si l'on veut étudier une succession d'épidémies sur le long terme. Il est important de complexifier le modèle en y incluant la démographie et par l'ajout de taux de natalité et de mortalité. Par exemple, pour une maladie sans symptôme apparent, le nombre d'individus observés au total en un temps donné t serait $N(t) = S(t) + I(t) + E(t) + R(t) + V(t)$.

7.1.2 L'implémentation du modèle

Ce système est décrit par un ensemble de paramètres spécifiques qui sont attribués à des taux correspondants à des situations fréquentes :

— β : taux d'infection (i.e. $S \to I$ ou $S \to E$).
— θ : taux de développement de la maladie (i.e. $E \to I$).

Événement	Taux	Description
(E1) Naissance des S	$S \to S + 1$	μN
(E2) Décès des S	$S \to S - 1$	μS
(E3) Infection	$S \to S - 1$ & $E \to E + 1$	βS
(E4) Décès des E	$E \to E - 1$	μE
(E5) Fin de la phase latente	$E \to E - 1$ & $I \to I + 1$	θE
(E6) Décès des I	$I \to I - 1$	μI
(E7) Fin de la phase infectieux	$I \to I - 1$ & $R \to R + 1$	νI
(E8) Décès des R	$R \to R - 1$	μR

Table 7.1 – Les événements de transmission d'individus entre les compartiments

— ν : taux de guérison avec gain d'immunité (i.e. $I \to R$).
— μ : taux de natalité/mortalité (e.g. $S, E, I, R \to X$ ou $S \to S$)

Ce modèle est mis en oeuvre en utilisant le modèle spatialement explicite avec le processus stochastique pour exprimer l'évolution et la propagation de la maladie en temps continu à travers différents niveaux spatiaux. La version stochastique de SEIR est définie par les événements et les taux dans le tableau 7.1 (avec N = S + E + I + R + V) : .

A partir des conditions initiales S(0), E(0), I(0) et R(0), on simule le processus en réitérant l'algorithme suivants :

— Calculer le taux total des événements $f_{sum} = f(S, E, I, R, \beta, \theta, \nu, \mu)$
— Régler l'incrément de temps $\delta(t) = f(f_{sum}, \delta t_{min})$
— Attribuer un nombre aléatoire à partir de l'intervalle $y \sim U(0, 1)$
— Choisir un événement :
  — (E1) if $y \leq \mu N \delta t$
  — (E2) else if $y \leq (\mu N + \mu S)\delta t$
  — (E3) else if $y \leq (\mu N + \mu S + \beta I)\delta t$
  — (E4) else if $y \leq (\mu N + \mu S + \beta I + \mu E)\delta t$
  — (E5) else if $y \leq (\mu N + \mu S + \beta I + \mu E + \theta E)\delta t$
  — (E6) else if $y \leq (\mu N + \mu S + \beta I + \mu E + \theta E + \mu I)\delta t$
  — (E7) else if $y \leq (\mu N + \mu S + \beta I + \mu E + \theta E + \mu I * \nu I)\delta t$
  — (E8) else if $y \leq (\mu N + \mu S + \beta I + \mu E + \theta E + \mu I * \nu I + \mu R)\delta t$
  — Aucun événement se produit autrement
— Mettre à jours (S(t), E(t), I(t) & R(t)) et le temps $t = t + \delta t$

Ce modèle reproduit la propagation et l'évolution de la maladie à partir d'un des objectifs initiaux de l'épidémiologie. Dans l'étape suivante, on modélise le campagne de vaccination.

7.1.3 Les principaux scénario de vaccination établis par les experts du domaine

La vaccination peut être appliquée de différentes manières pour réduire le taux de reproduction de la maladie. Nous considérons deux types : la vaccination de masse et la vaccination par pulsations (cf [CC09]). Ces deux aspects sont expliquées dans la suite de la section.

La vaccination de masse implique systématiquement une vaccination avec une couverture vaccinale minimale de la population. Par exemple, supposons que nous visons à vacciner une population pour la rougeole, nous pourrions vacciner chaque nouveau-né quelques mois après

leur naissance. Nous choisissons un modèle simple avec compartiments S, E, I, R et V. V est la population vaccinée. Nous supposons que la vaccination est continue avec le taux Θ (l'individu est vacciné après $1/\Theta$ unités de temps). Dans les pays occidentaux, la vaccination de masse est assez efficace car certaines maladie a pu être éradiquer du territoire. Malheureusement, cette politique est très coûteuse et est logistiquement lourde à réaliser à grand échelle, surtout dans les pays émergent.

La vaccination par pulsations implique une campagne pour vacciner une proportion donnée de la population vulnérable après certains intervalles de temps, avec une longueur de campagne suffisamment courte pour maintenir dynamiquement la proportion de susceptibles dans la population. Avec l'exemple de la rougeole, nous pourrions organiser une campagne tous les T ans, par exemple, et vacciner une proportion donnée (idéalement tous) des enfants de moins de quatre ans. Une campagne de quelques jours ou quelques semaines à une temporalité négligeable vis à vis de l'intervalle de quatre ans entre les phases de vaccination. Cependant, entre chaque campage de vaccination, la proportion de susceptibles dans la population augmente le taux de natalité jusqu'à la campagne suivante dans des proportion acceptable.

L'idée principale de la vaccination par pulsations est de réduire la population sensible autant que possible en trouvant une périodicité optimale de vaccination T. Suite au compte-rendu [CC09], les résultats théoriques disent que cette politique vaccinale est moins onéreuse mais elle est aussi moins efficace que la vaccination de masse. Un autre avantage de cette politique de vaccination est sa facilité à mettre en œuvre logistiquement, surtout dans le pays émergents comme Vietnam.

Dans la section suivante, on va discuter autour des choix experimentaux du modèle et de la combinaison d'algorithmes d'exploration pour l'explorer.

7.2 Choix experimentaux

7.2.1 Stratégie de vaccination

Ce modèle considère une politique de vaccination par pulsations, c'est à dire par des campagnes ciblées (spatialement et temporellement) de vaccination. Ces campagnes sont déclenchées par le nombre d'individus infectés (I) et le pourcentage de population vaccinée.

Quand ce nombre atteint un seuil, une campagne sanitaire est menée visant à vacciner une partie des susceptibles (S). La politique est définie selon trois paramètres (w_1, w_2, w_3). La politique de vaccination est comme suivante :

$$\boxed{si \quad w_1 x(I) + w_2 x(E) > \alpha * w_3, \quad faire \quad vacciner \quad 80\% \quad de \quad (S)}$$

- (S) : nombre d'individus dans le compartiment susceptible.
- (I) : nombre d'individus dans le compartiment infectieux.
- (E) : nombre d'individus dans le compartiment exposé.
- α : la constante qui équilibre S(0), E(0), I(0) et R(0).

Compte tenu du nombre de stratégies spatiales de vaccination possibles, une exploration complète du modèle est impossible d'où la nécessité d'utiliser des approches comme GRADEA. L'enjeu est de minimiser l'espace des paramètres à explorer, noté (w_1, w_2, w_3), tout en conservant des résultats satisfaisants.

Pour explorer le modèle, il faut déterminer la fonction objectif qui permet de déterminer les caractéristiques que l'on souhaite atteindre représentative d'une politique efficace de vaccination. Dans notre cas, nous souhaitons minimiser le nombre totale d'infectés et définissons ainsi la fonction objectif utilisée pendant l'exploration du modèle.

$$\boxed{minimize(TotalInfected)}$$

L'exploration devient alors une phase d'optimisation qui vise à trouver un ensemble de paramètres (w_1, w_2, w_3) caractérisant une campagne de vaccination minimisant le nombre total des infectés.

7.2.2 Algorithme d'exploration

Comme nous l'avons vu dans les chapitres précédents, GRADEA est une approche modulaire qui permet de construire des algorithmes hybrides d'exploration, en fonction des besoins et de la question scientifique.

Afin de répondre au mieux à notre problématique de modélisation, nous avons choisi d'associer 3 algorithmes d'exploration : (i) DIRECT (algorithme de recherche par criblage) ; (ii) l'algorithme évolutionnaire (algorithme de recherche globale) ; (iii) et APPS (un algorithme de recherche locale). L'utilisation que font [GK10, HHM07] montre que ces trois algorithmes sont particulièrement adaptés à notre problématique.

A l'image des préconisations de GRADEA, un agent a été créé pour chaque algorithme couplé, l'implémentation des opérateurs de trois algorithmes sur GRADEA est présenté dans la figure 7.1. :

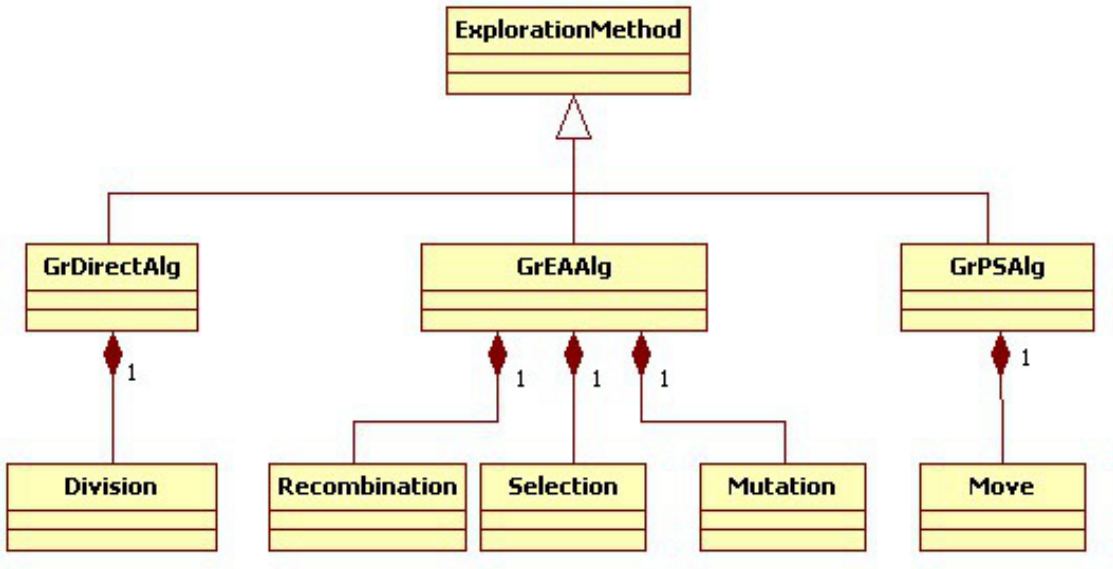

Figure 7.1 – Implémentation des opérateurs de recherche de la stratégie d'exploration choisie sur GRADEA

— *Agent GrDIRECT pour la recherche par criblage :* DIRECT (DIviding RECTangle) [JPS93] est un algorithme déterministe de recherche par motifs pour l'optimisation de fonctions

d'utilité multi-variées. Il considère l'espace des paramètres comme des hyper-rectangles (appelés *boxs*) multidimensionnels et divisibles. DIRECT est composé de 2 principaux opérateurs :

— *la division* pour diviser des boxs potentiellement optimales en trois sous boxs ;
— *la sélection* pour choisir les boxs potentiellement optimales devant être explorées à l'itération suivante.

Chaque *box* est représentée par son centre qui est évalué pour déterminer le prochain découpage.

A cet effet un agent de Recherche par Criblage a été implémenté. Son comportement est animé par une variante de l'algorithme DIRECT proposé dans [GKY08]. Nous avons développé une variante de l'opérateur de division qui s'applique à un ensemble de boxs dès qu'ils sont évaluées au lieu d'attendre tous les boxs sélectionnées. Cela réduit les temps d'attente des évaluations mais peut provoquer un changement d'ordre du partitionnement.

Algorithm 1 GrDIRECT

```
1: boxes ← {initBox}
2: Pqueue ← {GetCenter(initBox)}
3: for each message ∈ messageBox do
4:     sol ← GetSolution(message)
5:     R ← FindContainingBox(sol, boxes)
6:     UpdateCenterPoint(R, sol)
7: end for
8: agrSet ← GetAggressiveSet(boxes)
9: for each R ∈ agrSet do
10:    {i_1, i_2, .., i_m} ←
           OrderRandomLongestSides(R)
11:    R_new ← Divide(R, {i_1, i_2, .., i_m})
12:    boxes ← boxes \ {R} ∪ R_new
13:    for all R' ∈ R_new do
14:        Pqueue ← Pqueue ∪ GetCenter(R')
15:        SendToEvaluate(Pqueue)
16:    end for
17: end for
```

— *Agent GrEA pour la recherche globale :* Pour la recherche globale, nous faisons le choix d'utiliser un algorithme *génétique panmictique* asynchrone (AE) présenté dans [SDV08]. Cet algorithme a été adapté à notre contexte et implémenté dans le comportement d'un Agent de Recherche Globale. Le comportement de GrEA maintient μ jeux de paramètres dans sa population. AE est équipé des opérateurs *Selection, CrossOver (intermediate recombination), Mutation (Bit-Flip Mutation)* dans [Luk13]. Par contre, notre implémentation présente l'avantage de s'exécuter de façon asynchrone. Il n'a pas besoin d'attendre l'évaluation de tous les candidats de la population pour passer à l'itération suivante grâce au mécanisme de vérification continue de l'état des jobs en cours afin d'être informé de leur terminaison. Etant donné que l'expérimentation avec les paramètres de l'algorithme n'est pas notre objectif principal, nous fixons pour cet exemple les paramètres : $\mu = 10$, $\lambda = 20$, la proportion de recombinaison et de mutation est 0.25.
— *Agent GrPS pour la recherche Locale :* L'algorithme intégré pour l'Agent de Recherche Local est GrPS qui est une version de l'algorithme APPS ("Asynchronous Parallel Pattern Search" en anglais) dans [HKT01]. GrPS explore l'espace des paramètres à partir d'un point unique et cherche l'optimum local autour de ce point.

Ainsi trois agents de recherche GrEA, GrPS et GrDIRECT ont été implémentés. Lors de leur chargement, ils s'enregistrent auprès de la liste des services de la plate-forme GRADEA. A

Algorithm 2 GrEA

```
1: μ : maintain μ candidates in Pqueue
2: λ ← μ × k : maximal candidates in Pqueue
3: best ← null
4: for each message ∈ messageBox do
5:     solution ← GetSolution(message)
6:     Pqueue ← Pqueue ∪ {solution}
7:     if best ≠ GetBest(Pqueue) then
8:         best ← GetBest(Pqueue)
9:     end if
10:    if CheckStopCond(Pqueue) = true then
11:        Stop()
12:    end if
13:    if ‖Pqueue‖ > λ then
14:        TruncateSelect(Pqueue, μ)
15:    end if
16:    {P1, P2} ← RouletteWheelSelect(Pqueue)
17:    O ← CrossOver(P1, P2)
18:    O′ ← Mutation(O)
19:    SendToEvaluate(O′)
20:    LocalSearch(O′)
21: end for
```

la charge des stratégies de coopération d'utiliser intelligemment ces nouveaux services.

7.2.3 Stratégie de coopération entre les agents

Les trois agents précédemment définis (Agent GrPS, Agent GrEA et Agent GrDIRECT) sont cadencés selon l'approche GRADEA par deux agents de coordination : Agent Exploration-Requestor et ExploitationRequestor (cf. la figure 7.2). Ce dernier décompose l'exploration du modèle d'épidémiologie en 3 étapes :

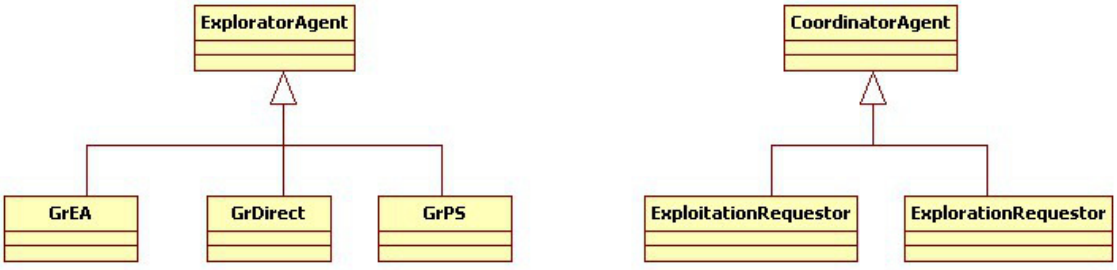

Figure 7.2 – Implémentation des agents de la stratégie d'exploration choisie sur GRADEA

1. Tous les agents s'inscrive auprès du répertoire d'agents pour s'exécuter.
2. L'agent GrDIRECT se lance sans conditions. L'agent GrDIRECT s'arrête lorsque les *box* ont une taille inférieure à un seuil.
3. L'agent GrDIRECT envoie les jeu de paramètres à l'agent GrEA via l'agent de coordination ExplorationRequestor.
4. L'agent GrEA s'achève lorsqu'aucune modification de la population de jeux de paramètre n'évolue pas pendant quelques itérations.
5. L'agent GrEA envoie les jeu de paramètres à l'agent GrPS via l'agent de coordination ExploitationRequestor.

Algorithm 3 GrPS

$\boldsymbol{D} \leftarrow \{d_i : 1 \leq i \leq 2p\}$
: initial set of search directions
$\delta_{init} \leftarrow \text{Normalize}(U_i - L_i),\ 1 \leq i \leq 2p$
$\boldsymbol{\Delta} \leftarrow \{\delta_i : 1 \leq i \leq p \text{ and } \delta_i \leftarrow \delta_{init}\}$
$\delta_{tol} \leftarrow \alpha \times \delta_{init}$
: step length convergence tolerance
$\delta_{min} \leftarrow \beta \times \boldsymbol{\delta}_{init}$ minimum first step length
$\boldsymbol{A} \leftarrow \emptyset$
$best \leftarrow x$
if *messageBox* receives a set of *message* **then**
    $\boldsymbol{X}_{recv} \leftarrow \text{ReceiveSolutions}(\{\text{message}\})$
    $x_{recv} \leftarrow \text{Min}(\boldsymbol{X}_{recv})$
    if f(x_{recv}) < f(best) **then**
        $best \leftarrow x_{recv}$
        $\delta_{recv} \leftarrow \text{Step}(x_{recv})$
        $\boldsymbol{\Delta} \leftarrow \{\delta_i : \delta_i \leftarrow \text{Max}(\delta_{min}, \delta_{recv(i)}),$
        $\forall i \in \{1, \ldots, \|\boldsymbol{D}\|\}\}$
        $\boldsymbol{A} \leftarrow \emptyset$
    else
        for each $x_{recv} \in \boldsymbol{X}_{recv}$ &
        $\text{Parent}(x_{recv}) = best$ **do**
            $\boldsymbol{I} \leftarrow \boldsymbol{I} \cup \{\text{Direction}(x_{recv})\}$
        end for
        for all $i \in \{1, \ldots, \|\boldsymbol{D}\|\} \setminus \boldsymbol{I}$ **do**
            $\delta_i \leftarrow \delta_i \times rand()$
        end for
        $\boldsymbol{K} \leftarrow \{i : 1 \leq i \leq \|\boldsymbol{D}\|, \delta_i < \delta_{tol}\ \}$
        $\boldsymbol{A} \leftarrow \boldsymbol{K} \setminus \boldsymbol{I}$
    end if
    if $\delta_i < \delta_{tol}\ \forall i \in \{1, \ldots, \|\boldsymbol{D}\|\}$ **then**
        Terminate()
    end if
    for all $i \in \{1, \ldots, \|\boldsymbol{D}\|\} \setminus \boldsymbol{A}$ **do**
        $y \leftarrow x_{best(i)} + \delta_i \times d_i$
        $\text{Parent}(y) \leftarrow x_{best}$
        $\text{Direction}(y) \leftarrow i$
        $\text{Step}(y) \leftarrow \delta_i$
        $\boldsymbol{Y} \leftarrow \boldsymbol{Y} \cup y$
    end for
    SendToEvaluate($\boldsymbol{Y}$)
end if

6. L'agent GrPS s'exécute tant que la fonction d'utilité n'attend pas un seuil avec une précision établie.
7. Agent GrPS et Agent GrDIRECT partage son meilleur jeu de paramètres trouvé à l'agent GrEA.

Dans ce contexte, l'agent Coordinateur est seulement un agent de transmission. Mais dans l'avenir, nous envisageons de lui conférer des capacités cognitives d'analyse de l'espace de paramètres afin de mieux orienter l'exploration par une utilisation parallèle des différentes méthodes de recherche.

Nombre de noeuds	Nombre de coeurs	Mémoire (To)	Puissance (Gflops) CPU	Puissance (Gflops) GPU	Consommation électrique (W)
135	1516	5.5508	24715	8253	29790

Table 7.2 – La puissance du mésocentre

7.2.4 Supports d'exécution

Nous utilisons le cluster du mésocentre de l'Université de Franche-Comté pour faire notre expérimentation [1]. Le mésocentre est un service de l'Université de Franche-Comté en partenariat avec l'Université Technologique de Belfort-Montbéliard, l'École Supérieure Nationale de Mécanique et des Microtechniques.

La ressource de calcul du mésocentre se sépare en deux clusters : mesocomte et lumière dont la puissance globale est dans le tableau 7.2. Notre travail est réalisé avec le cluster lumière dans lequel les jobs sont lancés dans la file d'attente all.q qui permet de soumettre les tâches en tableau de tâches.

L'architecture générale pour s'éxecuter GRADEA est présenté dans le figure 7.3

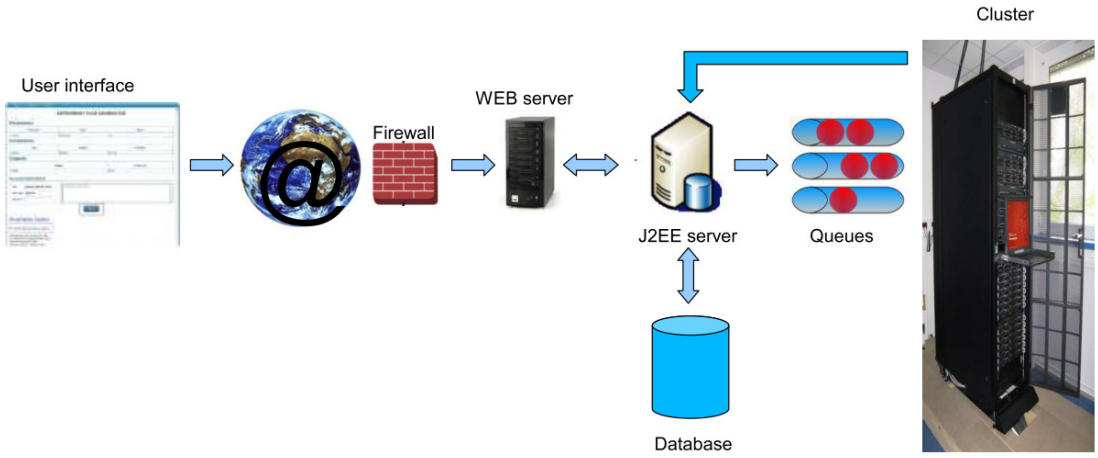

Figure 7.3 – L'archirecture matérielle de l'execution de GRADEA

Le serveur d'application (Jonas) est au cœur du cadre. Il est en charge d'ajouter les nouveaux simulateur, de l'enregistrement les données sur la base de données, de lancer les simulations sur le cluster, de recueillir les résultats du simulateur. Il est aussi en charge de lancer les fonctions de GRADEA.

7.3 Études des performances avec deux fonctions de benchmark

Nous testons la performance des différents stratégies d'exploration de GRADEA avec deux fonctions benchmark de différents niveaux de difficulté : (1) la fonction Branin dans [BJ72] avec un minimum global sans minimum local et (2) la fonction Goldstein-Price dans [GP71]

1. (site web de mésocentre "http ://meso.univ-fcomte.fr")

avec un minimum global et quelques minimums locaux. La forme de l'espace de paramètres de ces deux fonctions sont consultable à l'adresse `http://www-optima.amp.i.kyoto-u.ac.jp/member/student/hedar/Hedar_files/TestGO_files/Page1760.htm` pour Branin et `http://www-optima.amp.i.kyoto-u.ac.jp/member/student/hedar/Hedar_files/TestGO_files/Page913.htm` pour Goldstein-Price.

7.3.1 Protocole expérimental

La description de fonction Branin est ci-dessous :

- Nombre de variables : 2
- $f(x_1, x_2) = (x_2 - (5/4\pi^2)x_1^2 + (5/\pi)x_1 - 6)^2 + 10(1 - (1/8\pi))\cos(x_1) + 10$
- Domaine de recherche des variables : $-5 < x_1 < 10$, $0 < x_2 < 15$.
- Minimum local : il n'y a pas de minimum local
- Minimum global : $(x_1^*, x_2^*) = (-\pi, 12.275), (\pi, 2.275), (9.42478, 2.475)$;
  $f(x_1, x_2^*) = 0.397887$

Le description de fonction Goldstein-Price est ci-dessous :

- Nombre de variables : 2
- $f(x_1, x_2) = (1 + (x_1 + x_2 + 1)^2(19 - 14x_1 + 13x_1^2 - 14x_2) + 6x_1x_2 + 3x_2^2))(30 + (2x_1 - 3x_2)^2(18 - 32x_1 + 12x_1^2 - 48x_2 - 36x_1x_2 + 27x_2^2))$
- Domaine de recherche des variables : $-2 < x_i < 2$, $i = 1, 2$..
- Minimum local : il y a quelques minimums locaux
- Minimum global : $(x_1^*, x_2^*) = (0, 1)$;
  $f(x_1, x_2^*) = 3$

Nous recherchons le minimum global de ces deux fonctions avec une précision de 0,01. Le temps de calcul d'une fonction est allongé de 5 seconds afin d'assurer un de temps de calcul significatif. A chaque exécution, une fonction est répété 5 fois pour éviter les erreurs. Une telle exécution avec le nombre précis de réplications d'un même jeu valeur de paramètres qui s'appelle une unité d'évaluation ou évaluation dans le reste du document.

Les simulations sont soumisses dans une file d'attente pour laquelle sera affectée 8, 16 et 32 coeurs, cadencés à 2.4 Ghz, et doté de 64go de mémoire vive. Nous testons ces différentes configurations afin d'évaluer les performances et la monté en charge du système GRADEA en fonction de sa sollicitation. Les compositions en mode parallèle que nous avons implémenté sur GRADEA se composent : GrEA (E), GrEA + GrDIRECT (ED), GrEA + GrDIRECT + GrPS (EDP). La condition d'arrêt de GrDIRECT est la $taille\_box < 0.01$. La condition d'arrêt de GrEA est le nombre d'itération sans évolution de la qualité du paramètres $nb\_ite\_sans\_modification > 100$. Pour GrPS, la $granulation < 0.01$.

7.3.2 Résultats obtenus

La figure 7.4, 7.5 et 7.6 montre la performance des stratégies de recherche sur la fonction Branin : le temps de calcul, le taux où l'optimal global n'est pas trouvé avec une précision et le nombre d'évaluations.

Pour diminuer l'influence de la stochasticité dans les algorithmes d'exploration, nous avons lancé 7 réplications d'une expérimentation.

Au début, on compare le temps de calcul moyen d'une même stratégie d'exploration en

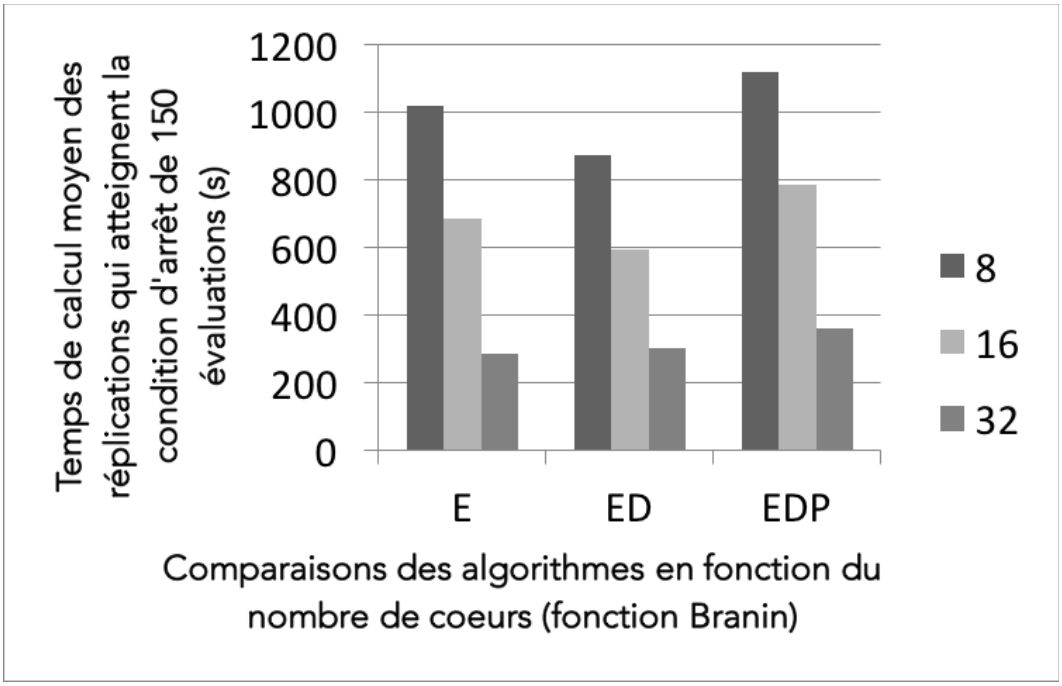

Figure 7.4 – temps de calcul moyen des réplications qui atteignent la condition d'arrêt de 150 évaluations de la fonction Branin(T(s))

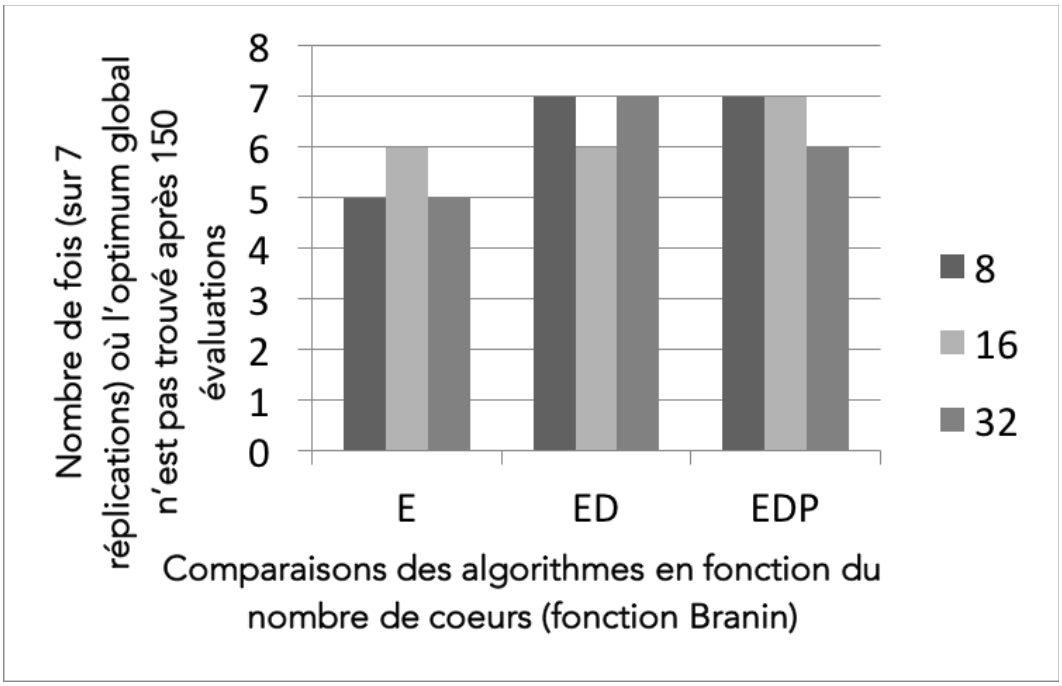

Figure 7.5 – nombre fois (sur 7 réplications) où l'optimal global n'est pas trouvé après 150 évaluations de la fonction Branin(T(s))

fonction du nombre de cœurs différents. Le calcul moyen ne compte que les expérimentations qui atteignent la condition d'arrêt de 150 évaluations pour s'assurer que la comparaison est avec des expérimentation de même précision. Le nombre de fois (sur 7 réplications) où l'optimal global n'est pas trouvé après 150 évaluations de la fonction donne une évaluation de l'algorithme. Ces valeurs peuvent être trouvées dans la figure 7.5. Les cas qui ne atteignent pas la précision sont les cas où est trouvé l'optimal globale avant d'atteindre 150 évaluations.

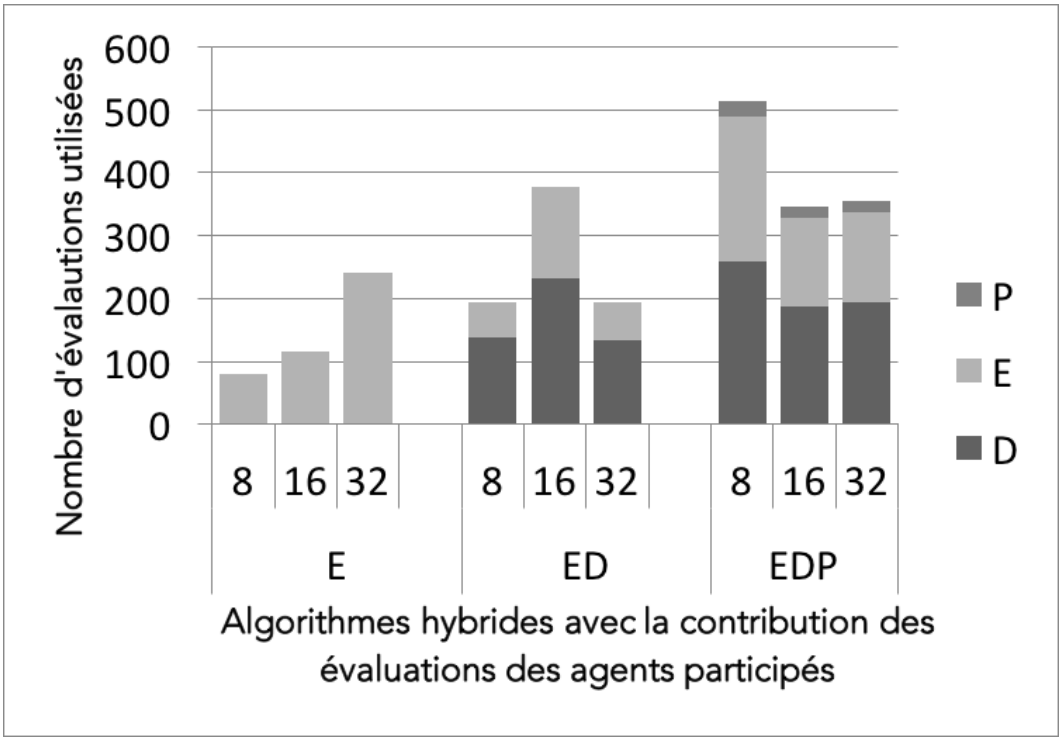

Figure 7.6 – nombre d'évaluations cumulé de la fonction Branin(T(s))

Pour la fonction Brannin qui n'a pas d'optimum local, il est facile pour un algorithme évolutionnaire sans être combiné de trouver l'optimum global. La figure 7.5 montre qu'il n'a pas besoin d'être guidé par les autres algorithmes pour trouver l'optimum local après avoir atteint la condition d'arrêt de 150 évaluations. Il y a totalement 3 cas (réplication) dans lesquels le stratégie E a trouvé l'optimal global tandis que ED, EDP requièrent plus nombreux d'évaluations pour trouver l'optimum global (il y a un seul réplication dans lequel ED et EDP trouve l'optimum global. Cette figure montre aussi que l'augmentation de nombre de cœurs ne change pas l'opportunité de trouver l'optimum global avec la même précision.

La figure 7.4 montre l'efficacité de temps de calcul en fonction du nombre de cœurs. La figure visualise le temps de calcul moyen des réplications avec 150 évaluations pour chacune d'elles. Dans tous les trois stratégies d'exploration E, ED et EDP, le temps de calcul diminue très vite grâce à l'augmentation de nombre de cœurs. Dans le cas d'une stratégie d'exploration E : le temps diminue 33% pour 16 cœurs et 72% pour 32 cœurs. Dans le cas du stratégie d'exploration ED : le temps diminue 32% pour 16 cœurs et 65% pour 32 cœurs. Dans le cas du stratégie d'exploration EDP : le temps diminue 30% pour 16 cœurs et 68% pour 32 cœurs par rapport les expérimentations avec 8 cœurs.

Afin de conclure sur l'expérimentation du calcul intensif et de l'association des algorithmes de la fonction Brannin, on peut dire que l'augmentation de nombre de cœurs de calcul réduit de manière significative les temps de calcul. Cependant, l'efficacité de l'association des différents algorithmes d'exploration sur un problème qui n'a pas d'optimums locaux, n'est pas montrée.

Pour la convergence, on s'intéresse à une expérimentation précise en tant qu'exemple. On peut voir qu'avec la fonction Branin sans minimum local, l'algorithme EA fonctionne très bien par lui-même. Le nombre d'évaluations est moins grand que pour les autres. On peut comprendre que dans ce cas, ED et EDP a perdu beaucoup de temps pour couvrir spécialement l'espace de paramètres. On peut voir le grand nombre d'évaluations de l'algorithme GrDIRECT (D) qui cause cette situation. La figure 7.6 indique que le nombre d'évaluations de l'algorithme

GrDIRECT est plus grand que le nombre d'évaluations de l'algorithme GrEA. Le comportement de l'algorithme GrDIRECT est de diviser l'espace de paramètres et explorer partout dans l'espace de paramètres. Dans deux cas d'étude ED-8 et ED-32, on voit que l'algorithme GrDIRECT a guidé l'algorithme GrEA pour couvrir l'espace de paramètres alors que cette fonction Branin ne contient pas d'optimum local. La convergence de GrEA est plus lente.

On étudie, dans cette partie, la différence quand on applique les mêmes stratégies de recherche sur la fonction Goldstein-Price qui contient quelques minimums locaux. La figure 7.7, 7.8 et 7.9 montrent les performances des stratégies de recherche sur la fonction Goldstein-Price : le temps de calcul, le taux où l'optimal global n'est pas trouvé avec une précision et le nombre d'évaluations de cette fonction.

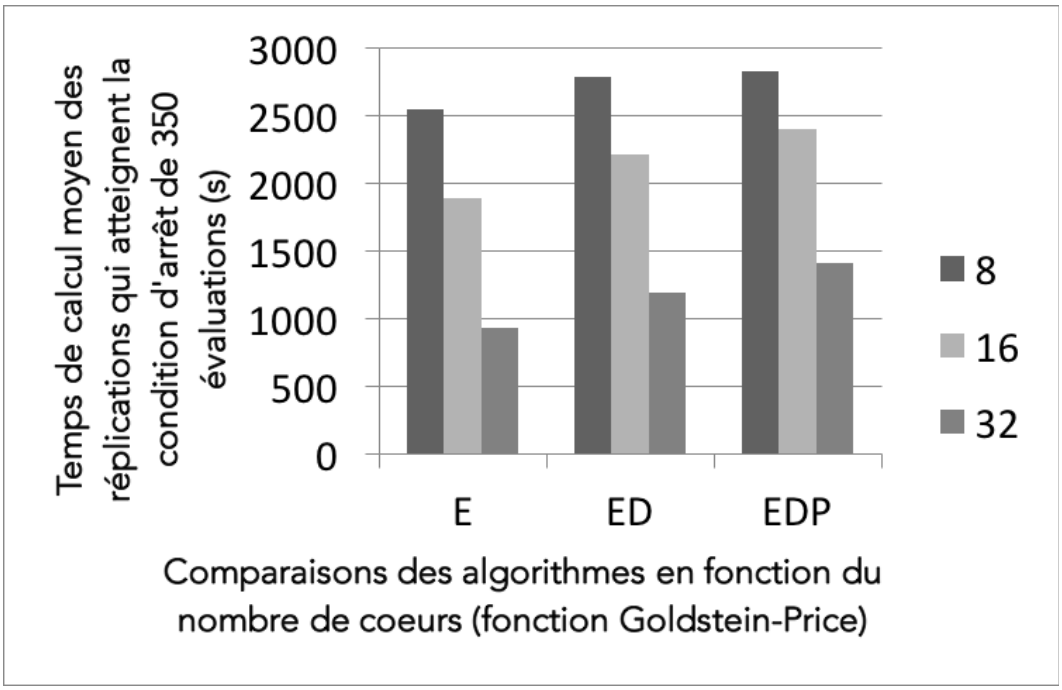

Figure 7.7 – temps de calcul moyen des réplications qui atteignent la condition d'arrêt de 150 évaluations de la fonction Goldstein-Price (T(s))

On réalise les analyses de ces deux graphes de la même manière que pour la fonction de Branin afin d'être en mesure de comparer les résultats.

Avec cette fonction, nous avons également lancé 7 réplications d'une expérimentation.

On compare le temps de calcul moyen en fonction du nombre de cœurs différents. La précision (la condition d'arrêt) dans ce cas est 350 évaluations. On distingue les cas où l'optimal global n'est pas trouvé. Ces valeurs sont indiquées dans la figure 7.8. Les cas qui n'atteignent pas la précision sont les cas où est trouvés l'optimal global avant d'atteindre 350 évaluations.

Pour la fonction Goldstein-Price qui contient les optimums locaux, met en difficultée un algorithme évolutionnaire non combiné pour de trouver l'optimum global. La figure 7.8 montre que tous les 21 fois, la stratégie E (seulement algorithme évolutionnaire) ne peut pas trouver l'optimum globale après 350 évaluations. En associant avec l'algorithme D, le stratégie ED trouve l'optimum global dans 4 fois (2 fois avec 8 coeurs et 2 fois avec 16 coeurs). Le meilleur cas est le stratégie EDP, il trouve l'optimum local dans 11 fois (4 avec 8 coeurs, 3 fois avec 16 coeurs et 4 fois avec 32 coeurs).

La figure 7.7 montre l'efficacité de temps de calcul en fonction du nombre de cœurs. La figure

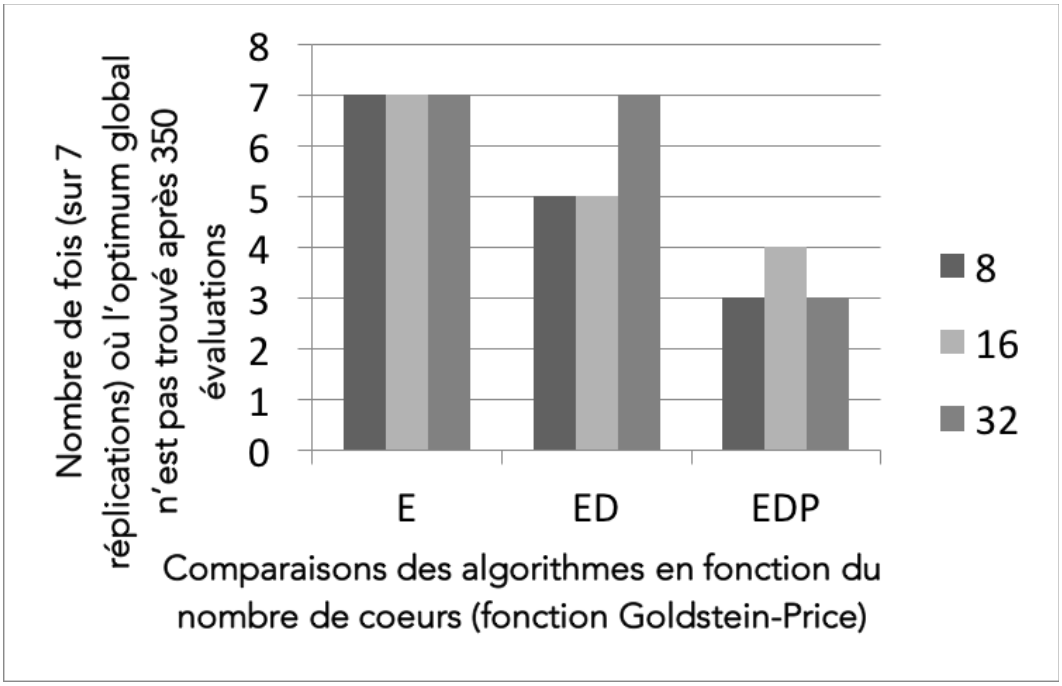

Figure 7.8 – nombre fois (sur 7 réplications) où l'optimal global n'est pas trouvé après 150 évaluations de la fonction Goldstein-Price (T(s))

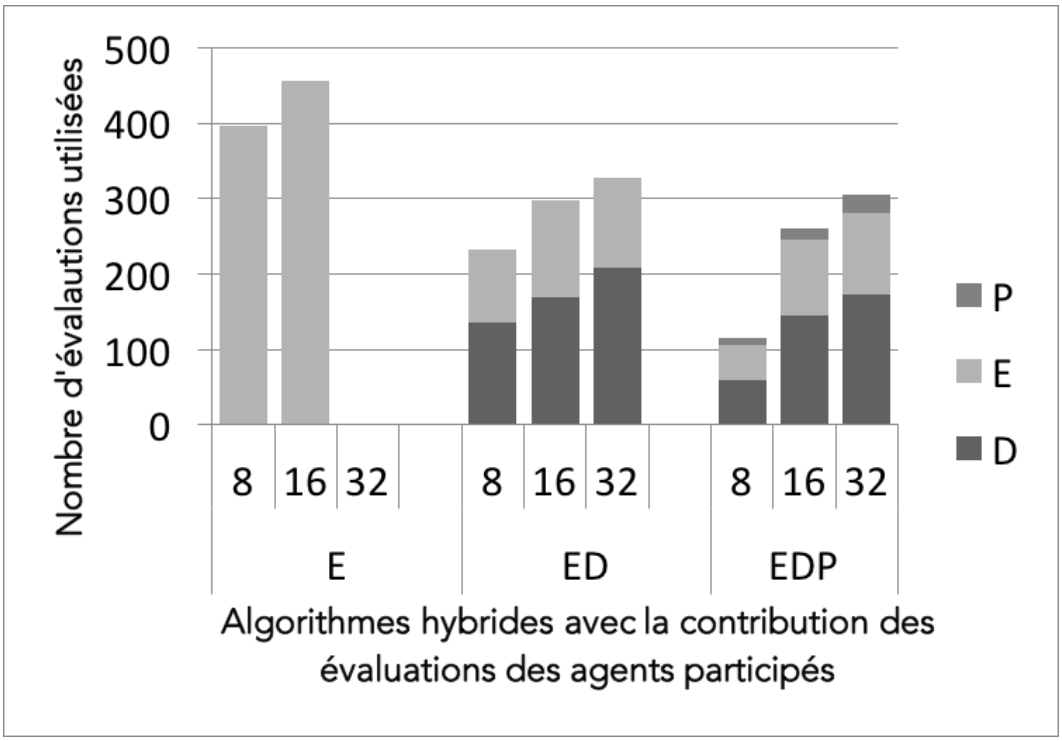

Figure 7.9 – nombre d'évaluations cumulé de la fonction Goldstein-Price(T(s))

visualise le temps de calcul moyen des réplications avec 350 évaluations de chacune d'elles. Dans le cas d'une stratégie d'exploration E : le temps diminue 26% pour 16 cœurs et 64% pour 32 cœurs. Dans le cas du stratégie d'exploration ED : le temps diminue 20% pour 16 cœurs et 57% pour 32 cœurs. Dans le cas du stratégie d'exploration EDP : le temps diminue 15% pour 16 cœurs et 50% pour 32 cœurs par rapport les expérimentations avec 8 cœurs.

En conclusion, on peut dire que les algorithmes savent tirer de nombre de cœurs de calcul. On montre aussi l'efficacité de l'association des différents algorithmes d'exploration pour un même problème contenant des optimums locaux.

En ce qui concerne la convergence, sur une expérimentation précis dans la figure 7.9 on note une différence notable avec les résultats de la fonction Branin. On voit qu'avec la fonction Goldstein-Price, il y a quelques minimums locaux, l'algorithme EA ne fonctionne pas correctement quand il est utilisé seul. Le nombre d'évaluations est plus grand qu'avec d'autres approches. Le pire cas est quand E-32 ne peut pas trouver l'optimum globale après avoir évalué 1000 jeux de paramètres parce que le fonction Goldstein-Price est plus difficile à explorer que Branin. Parfois, l'algorithme GrEA ne peut pas trouver l'optimum global si il n'est pas guidé par d'autres algorithmes comme GrDIRECT ou GrPS.

De plus, on voit clairement que ED est mieux que E et EDP est mieux que ED. Dans ce cas, on peut voir aussi que l'algorithme GrDIRECT (D) guide ED et EDP de dépasser les minimums locaux. On peut voir aussi que l'algorithme P guide EDP de dépasser facilement les minimums locaux.

Après avoir étudié GRADEA avec deux cas d'étude, on peut voir l'application prometteuse de ces stratégies d'exploration sur différents problèmes. On va appliquer ces stratégies d'exploration au domaine de l'épidémiologique dans la section suivante.

7.4 Études des performances avec le modèle épidémiologique

Nous suivrons donc pas à pas l'approche GRADEA en présentant les algorithmes d'exploration mis en place, et la stratégie d'exploration dans un second temps. Dans un dernier temps, nous ferons une analyse des résultats.

7.4.1 Protocole expérimental

Nous recherchons à réduire le nombre d'infections sur une population d'un million d'individus durant une période de 5 ans avec 600 000 doses disponibles. L'évaluation de la performance de GRADEA sera réalisée au travers de l'exploration des paramètres (w_1, w_2, w_3) dont le domaine est compris entre 0 et 1 avec une précision de 0,01 et réaliser ainsi 1000 réplication du modèle SEIR.

GRADEA sera en mesure de lancer les simulations sur un cluster basé sur SGE, système de soumission supporté par la version courante de GRADEA. Chaque simulation est ainsi soumise dans une file d'attente spécifique du cluster et répliquée 3000 fois : une telle simulation avec le nombre précis de réplications d'un même jeu de valeurs de paramètres qui s'appelle une unité d'évaluation ou évaluation dans le reste du document.

Les simulations soumissent dans une file d'attente pour laquelle sera affectée 8, 16 et 32 coeurs, cadencés à 2.4 Ghz, et doté de 64go de mémoire vive. Nous testons ces différentes configurations afin d'évaluer les performances et la montée en charge du système GRADEA en fonction de sa sollicitation.

Nous avons testé différentes configuration d'utilisation de GRADEA. Les combinaisons en mode parallèle que nous avons implémentées sur GRADEA se composent : GrEA (E), GrEA + GrDIRECT (ED), GrEA + GrDIRECT + GrPS (EDP). La condition d'arrêt de GrDIRECT

Table 7.3 – Les paramètres de l'expérimentation

Modèle SEIR stochastique	la population	1000000 individus
	la période de simulation	5 ans
	le nombre de doses de vaccin	600000
Paramètres d'exploration	les paramètres d'entrée	w1, w2, w3
	domaine des paramètres	[0, 1], précision 0.01
	fonction objectif	minimiser le nombre d'infectés
	nombre de réplications	1000 répétitions
Conditions d'arrêt de la stratégie coopérative	GrDIRECT	taille_box <0.01
	GrEA	nb_ite_sans_modification >100
	GrPS	granularité <0.01
Cluster	nombre de coeurs	8, 16 et 32
	cadencement	2.4 Ghz
	mémoire vive	64go

est la $taille\_box < 0.01$. La condition d'arrêt de GrEA est le nombre d'itération sans évolution de la qualité du paramètres $nb\_ite\_sans\_modification > 100$, pour GrPS, la $granulation < 0.01$.

Tous les conditions de l'exploration sont exprimées dans le tableau 7.3 :

7.4.2 Résultats obtenus

L'approche GRADEA est évaluée par deux métriques : le résultat thématique qui montre la capacité de convergence et les performances qui donnent une indication sur la scalabilité.

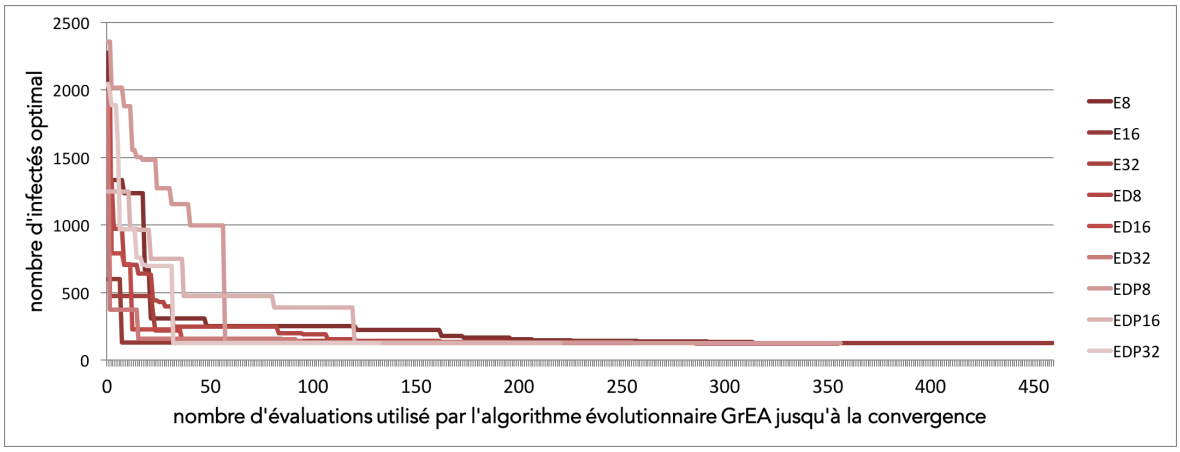

Figure 7.10 – convergence des différents stratégies réalisées sur 8, 16 et 32 coeurs de calcul

La figure 7.10 affiche une convergence de E, ED et EDP réalisée sur 8, 16 et 32 cœurs de calcul. Dans cette figure pour regarder la convergence, nous avons choisi de n'afficher que le nombre d'évaluations de l'algorithme évolutionnaire GrEA dans sa combinaison avec les autres algorithmes (GrPS et GrDIRECT). Nous voulons étudier l'importance des algorithmes GrPS et GrDIRECT pour améliorer le comportement de l'algorithme GrEA.

Dans la figure 7.10, la forme de la convergence a montré que dans la combinaison de l'EDP, GrEA nécessite moins d'évaluations (et donc de simulations) que celui de l'ED. GrEA de l'ED nécessite moins évaluations que celui de la combinaison E. Ca veut dire que dans ces

cas : le rôle des algorithmes d'assistance est très important. D'autre part, la convergence des hybrides est plus régulière. Cette dernière est utile en cas d'exploration du comportement du modèle plutôt que trouver une solution optimale.

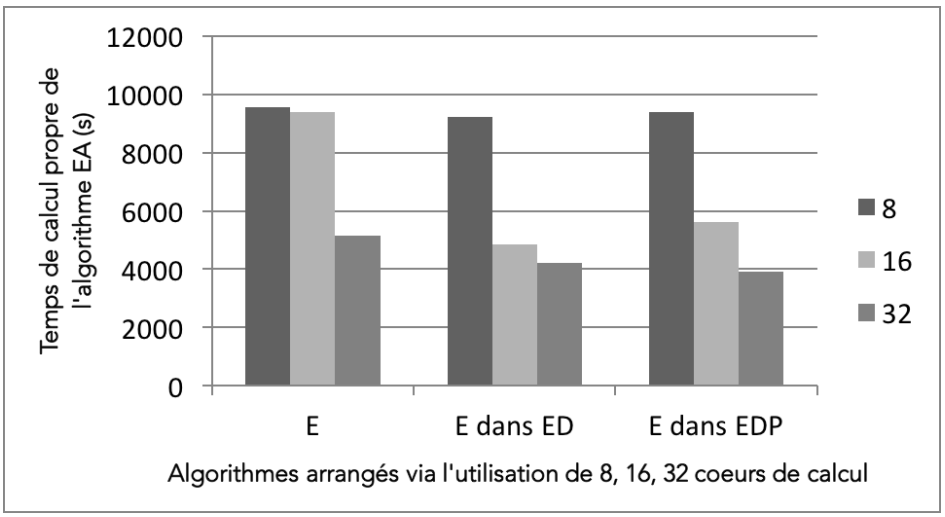

Figure 7.11 – temps de calcul (T(s))

La figure 7.11 affiche le temps de calcul consommé par l'algorithme évolutionnaire GrEA (E) lorsqu'il est jumelé avec les deux algorithmes GrDIRECT et GrPS. Quand le nombre de cœur augmente de 8, 16 et 32 dans les trois cas, le temps de calcul est réduit très rapide.

Les résultats présentés par la figure 7.12 montrent l'efficacité de GRADEA et l'intérêt de combiner des algorithmes d'exploration. En effet, l'algorithme EDP se termine avec moins d'évaluations que ED tout en conservant une précision dans ses résultats. Pour cela, il tire le meilleur des trois algorithmes d'exploration. E semble plus performant que ED et EDP dans ce cas d'étude précis.

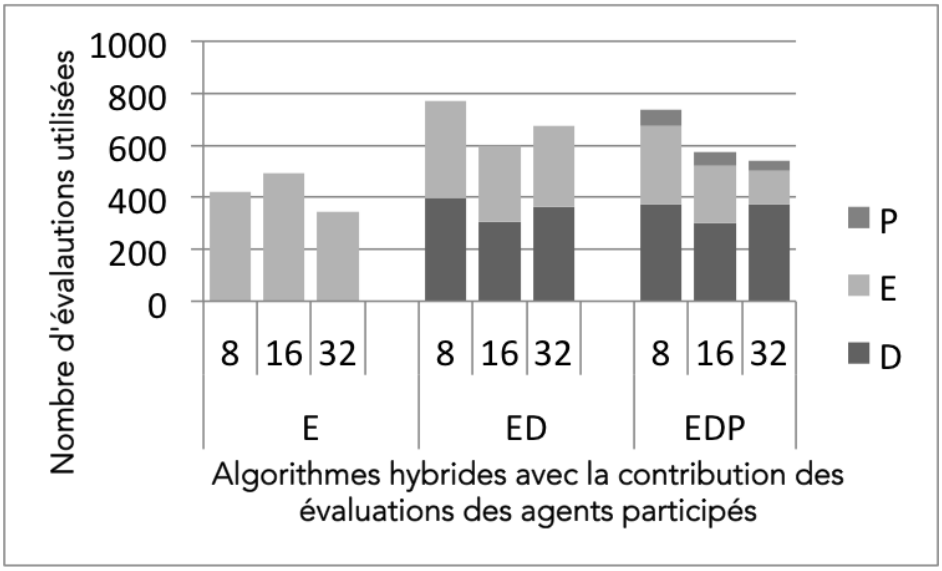

Figure 7.12 – nombre d'évaluations (nb_{eval}) (T(ms)/nb_{eval})

Malgré un temps d'exécution plus lent que ED, l'algorithme hybride EDP donne en contrepartie des résultats plus précis. Ces temps de calcul et cette précision est le fruit de l'algorithme de recherche local (GrPS), dernière étape de la stratégie implémentée dans EDP.

Cet algorithme raffine la solution mais ne soumet pas suffisamment d'évaluations dans la file d'attente. En effet il génère au plus de 2n (n=3) évaluations à chaque itération. Une des améliorations possibles viserait à distribuer plusieurs exécutions de GrPS, en le considérant, dans la stratégie d'exploration, comme une partie de l'algorithme évolutionnaire.

7.4.3 Résultats thématiques

L'objectif principale de cette expérimentation est de tester la performance de la coordination entre plusieurs algorithmes d'exploration. Cependant, nous avons aussi donner quelques lignes sur les résultats thématiques.

La politique de vaccination avec trois paramètres (w_1, w_2, w_3) :

$$\boxed{si \quad w_1x(I) + w_2x(E) > \alpha * w_3, \quad faire \quad vacciner \quad 80\% \quad de \quad (S)}$$

Les résultats montrent que quand $w1$ et $w2$ convergent aux bornes supérieures (valeur 1) et $w3$ converge à borne inférieure (valeur 0), ça veut dire que on va vacciner tous les individus infectés, on va obtenir le nombre minimale des individus d'infection durant toute le campagne de vaccination.

Dans l'avenir, on peut explorer un autre stratégie d'exploration tel que : le pourcentage des individus pour vacciner ou un autre fonction de vaccination par exemple : $w_0x(S) + w_1x(I) + w_2x(E) + w_3x(R) > \alpha * w_4$.

Discussion

Le protocole d'expérimentation sur 8, 16, 32 cœurs avec la fonction Branin et la fonction Goldstein-Price permet d'étudier notre proposition dans les deux aspects : l'efficacité du calcul intensif et l'efficacité de l'association des algorithme d'exploration sur des stratégies d'exploration différentes E, ED et EDP.

Les résultats montrent toujours que notre proposition profite bien des systèmes de calcul intensif. Le temps de calcul diminue très vite quand le nombre de cœurs augmente. L'association des algorithmes dans les stratégies différents ne montre pas toujours son efficacité. Pour les fonctions de l'espace de paramètres plus simple, l'association n'est pas nécessaire et parfois requière plus d'évaluations que nécessaire. Par contre, pour des fonctions avec un espace de paramètres plus compliqué (avec quelques optimums locaux, par exemple), l'association est nécessaire pour trouver l'optimum globale.

Dans ce chapitre, nous avons appliqué l'approche GRADEA avec un modèle épidémiologique s'intéressant à la maladie rougeole au Vietnam. En pratique, la vaccination en masse est difficile à réaliser quand la superficie et la population à vacciner sont importantes. Une politique de vaccination par pulsations est donc souhaitable dans un pays en développement comme Vietnam. C'est pourquoi, on a besoin de trouver une politique efficace parmi l'ensemble des possibilités.

Dans la dernière partie, on a expérimenté le protocole expérimental avec le modèle épidémiologique. Une politique de vaccination efficace est trouvée, la coordination des algorithme d'exploration s'avère une approche performante.

Conclusion et Perspective

Conclusion

Dans ce manuscrit, nous nous intéressons à l'exploration de simulateurs complexes représentatifs de phénomènes réels. L'exploration reprose sur un parcours adapté de l'espace des paramètres afin de répondre à plusieurs objectifs : la validation, l'optimisation, la prise de décision. Ce processus vise à trouver un ou des jeux de paramètres satisfaisants en exécutant le nombre minimal possible de simulations nécessaires. Cette phase d'exploration est très importante comme nous l'avons montré mais aussi très difficile.

L'exploration d'un simulateur complexe nécessite la mise en place d'un algorithme d'exploration adapté et singulier. En effet, la nature et la texture de l'espace des paramètres sont fortement conditionnées par le système complexe étudié et la question scientifique. L'approche que nous avons choisie pour résoudre ce problème est que considérer ce problème comme un problème d'optimisation. Nous traitons un simulateur complexe comme une boîte noire définie par ses entrées (jeux de paramètres) et ses résultats. Cette boîte noire est manipulée et observée par une plate-forme appelée GRADEA. Cette plate-forme tire partie des systèmes de calcul haute performance et d'une combinaisons d'algorithmes d'exploration choisie au cas par cas par l'utilisateur.

GRADEA repose sur une architecture multi-agents où chaque agent est un algorithme d'exploration autonome qui collabore avec d'autres agents algorithme, sous le couvert de coordinateurs. Grâce à cette approche, l'algorithme d'exploration devient interchangeable et combinable, en fonction des cas d'étude et sans modification de l'architecture existante.

Cette architecture permet de résoudre deux problèmes : la modularité des algorithmes d'exploration et la coordination des algorithmes d'exploration. En effet, GRADEA fournit un outil pour la conception des stratégies de coordination et les représente sous la forme d'un systèmes multi-agents. Des agents implémentant des algorithmes usuels sont nativement proposés (DIRECT, Algorithme Evolutionaire...), mais l'utilisateur est en mesure d'en ajouter selon ses besoins. C'est agents réutilisables sont activés pour fonctionner indépendamment ou de façon coordonnée.

En outre, GRADEA propose une infrastructure modulaire afin de faciliter la conception de stratégies combinées d'exploration. Les stratégies ainsi conçues peuvent embarquées des politiques issues de Intelligence Artificielle, tel que l'apprentissage par renforcement. L'infrastructure permet ainsi de définir les politiques de collecte des informations fournies par les algorithmes de recherche, d'échanger l'information, de définir les phases de coopération ou de compétition et d'adapter les paramètres de chaque algorithme à partir des informations collectées.

GRADEA repose sur une architecture qui permet la modularité des calculs. Cette architecture est définie par un ensemble de formalismes génériques et d'interfaces d'implémentation pour faciliter l'intégration des nouveaux simulateurs ou permettre la connexion à de nouveaux systèmes de calcul. Les simulateurs complexes peuvent être intégrés dans GRADEA grâce à ces interfaces unique.

En couplant à un serveur d'application JEE et un cluster, GRADEA permet de déployer de nombreuses simulations sur des moyens de calcul intensif. La coordination de l'exploration est réalisé au sein du serveur d'application. Cette infrastructure logicielle a été déployée sur un serveur JEE JONAS jumelé au mésocentre de calcul de l'université de Franche-Comté (France, l'Université de Franche Comté).

Pour évaluer la performance de GRADEA, Nous tirons partie de notre expérience en modélisation de phénomènes de propagation de maladie rougeole au Vietnam. A autre côté, nous avons choisi d'implémenter l'algorithme DIRECT pour la recherche par criblage, l'algorithme génétique panmitique asynchrone pour la recherche globale et l'algorithme de recherche en patron (PS) pour la recherche locale. L'objectif est pour comparer la performance de différents associations des algorithmes entre la recherche par criblage, la recherche locale ; et la recherche globale.

Les résultats des expériences numériques ont montré les performances intéressante de GRADEA et sa scalabilité. L'exploration coopérative à base d'agents est une approche prometteuse et efficace pour résoudre des problèmes d'optimisation sophistiquées de manière adaptative et distribuée. Dans le cas le plus désastreux, la coordination est au moins aussi rapide que l'algorithme dans sa version isolée.

Perspective

Exécution distribuée de GRADEA

En revanche, le principal inconvénient de cette approche est l'augmentation du nombre d'évaluations, qui est implicitement causé par l'exécution simultanée de plusieurs méthodes. Cette situation peut causer le bottleneck. Dans l'avenir, il semble intéressant de pouvoir tester l'exécution des agents en parallèle (par exemple l'utilisation d'une plate-forme multi-agent parallèle) pour tirer le meilleur partie des ressources de calcul intensif. De cette façon, la parallélisation est stratifiée sur plusieurs niveaux : niveau agent, niveau de l'algorithme lui-même et niveau des évaluations de la solution. D'autre part, on peut intégrer GRADEA avec les autres systèmes de calcul intensif tels que la grille de calcul ou la grille volontaire.

L'évolutivité des solveurs pendant l'exécution

La performance d'un coordination dépend de l'ensemble des algorithmes disponibles. Par conséquent, même si une approche de coordination est efficace pour explorer plus efficacement un problème, il faut presque faire à la main le choix des algorithmes et leur paramètres de configuration pour les associer dès le début. Il serait alors intéressant de fournir une opportunité d'adaptation automatique faisant le choix de quelques algorithmes ou différents paramètres de configuration pour un problème spécifique. Une combinaison adaptée autonome des algorithmes peut alors être utilisée comme un outil d'exploration coopérative entièrement automatisée pour résoudre des problèmes de manière plus efficace. Pour faire cela, au cours de

l'exécution de l'algorithme, pléthore d'information peut être suivie. L'idée est telle que nous allons suivre toutes les informations d'exécution des algorithmes ou une algorithme spécifique pour choisir la meilleure combination.

Caractérisation du paysage adaptatif (fitness landscapes)

En amont de GRADEA, nous envisageons de proposer des mécanismes de clustering qui permettront d'identifier la nature de l'espace des paramètres d'un modèle complexe ou le paysage adaptatif. Le paysage adaptatif est utilisé pour étudier la relation entre les résultats avec l'espace de recherche. Cet outil facilitera le choix des algorithmes d'exploration à utiliser pour étudier un phénomène réel. Par exemple, on pourrait être en mesure d'estimer le nombre d'optimum locaux présents dans l'espace de résultats, et la taille de bassins pour chacun de ces optimums. A cause de chaque algorithme fonctionne bien avec un type de paysage adaptatif, on peut fournir un profile des algorithmes et choisir un algorithme dépendant un paysage séparée. D'autre part, il faut garder le containeur de solutions assez diverse qui est responsable de fournir de solutions de candidats à agent particulier de l'algorithme. Il pourrait maintenir la population suffisamment diversifiés en sélectionnant les candidates pour envoyer.

Apprentissage hors ligne

Grâce au comportement d'exploration des agents dans un système hybride, on peut récupérer les performances des algorithmes d'exploration et leur meilleures configurations sur un problème précis et sauvegarder les résultats d'exploration stockés dans la base de donnée. Ces résultats peuvent être réutilisé dans le prochain fois d'exploration d'un modèle de même problème. De cette façon, on peut initialiser la choix d'association des méthodes d'exploration parmi plus nombre des algorithmes d'exploration existants à partir des données sauvegardées hors ligne. Cela augmente l'efficacité, l'évolutivité d'une stratégie coopérative et diminue de la complexité de mise en œuvre des nouvelles approches hybrides.

Exploration coopérative pour résolution multi-objectifs

Le problème présenté dans ce travail est mono-objectif. On considère souvent qu'un objectif simple est combiné par plusieurs objectifs en une seule fonction objectif de somme pondérée. Toutefois, Il existe également des recherches multi objectifs. Au lieu de choisir un seul comportement, les modélisateurs choisissent plusieurs comportements, et conçoivent plusieurs fonctions objectifs qui quantifient ces comportements. Par conséquent, nous supposerons qu'il est possible d'utiliser la puissance de l'exploration coopérative pour résoudre ce type de problèmes plus efficacement.

Cas d'étude supplémentaire

Chaque cas d'étude fourni de nouvelles informations et des perspectives sur l'exploration coopérative des simulations de système complexe. Il permet également de mesurer l'efficacité des différentes associations des algorithmes sur différents problèmes. Alors, pour mieux apprendre, des études de cas supplémentaires sont également susceptibles d'être utils et instructifs. Dans ce cas, comme on a présenté dans le chapitre 1, il y a deux autres modèles :

l'étude de comportements de la mobilité urbaine et le mouvement dans les ville et la modélisation et simulation des socio-agro-hydrosystèmes sont autant de sujet intéressant. Pour chacun d'eux, il faudra trouver une association efficace des algorithme d'exploration.

Bibliographie

[ABE+09a] David Abramson, Blair Bethwaite, Colin Enticott, Slavisa Garic et Tom Peachey : Parameter space exploration using scientific workflows. *In Proceedings of the 9th International Conference on Computational Science (ICCS '09)*, pages 104–113. Springer, 2009.

[ABE+09b] David Abramson, Blair Bethwaite, Colin Enticott, Slavisa Garic et Tom Peachey : Parameter space exploration using scientific workflows. *In Computational Science–ICCS 2009*, pages 104–113. Springer, 2009.

[ABE+11] David Abramson, Blair Bethwaite, Colin Enticott, Slavisa Garic et Tom Peachey : Parameter exploration in science and engineering using many-task computing. *Parallel and Distributed Systems, IEEE Transactions on*, 22(6):960–973, 2011.

[AP06] Frédéric Amblard et Denis Phan : Modélisation et simulation multi-agents. 2006.

[AT02] Enrique Alba et Marco Tomassini : Parallelism and evolutionary algorithms. *Evolutionary Computation, IEEE Transactions on*, 6(5):443–462, 2002.

[AT04] Mike Ault et Madhu Tumma : *Oracle 10G Grid & Real Application Clusters : Oracle 10G Grid Computing With Rac*. Rampant Techpress, 2004.

[Aum07] Craig A. Aumann : A methodology for developing simulation models of complex systems. *Ecological Modelling*, 202(3–4):385 – 396, 2007.

[Ayd07] Mehmet E Aydin : Metaheuristic agent teams for job shop scheduling problems. *In Holonic and Multi-Agent Systems for Manufacturing*, pages 185–194. Springer, 2007.

[Aza99] F. Azadivar : Simulation optimization methodologies. *In Simulation Conference Proceedings, 1999 Winter*, volume 1, pages 93–100 vol.1, 1999.

[Bak00] Mark Baker : Cluster computing white paper. *arXiv preprint cs/0004014*, 2000.

[BB03] PM Boulanger et T Bréchet : Modélisation et aide à la décision pour un développement durable : état de l'art et perspectives. *Rapport final au SPP Politiques scientifiques (AS/F5/01). Bruxelles (Belgique) : Institut pour un développement durable*, 2003.

[BBMC+10] Arnaud Banos, Annabelle Boffet-Mas, Sonia Chardonnel, Christophe Lang, Nicolas Marilleau et Thomas Thévenin : MIRO : des trajectoires individuelles à la ville en mouvement. *In* Antoni Jean-Philippe, éditeur : *Modéliser la ville : formes urbaines et politiques de transport*, pages 216–245. Economica, 2010. 08 08.

[BCC+11] Eric Blanchart, Christophe Cambier, C. Canape, Benoit Gaudou, The-Nhan Ho, Tuong-Vinh Ho, Christophe Lang, Fabien Michel, Nicolas Marilleau et Laurent Philippe : EPIS : A grid platform to ease and optimize

multi-agent simulators running. *In Advances on Practical Applications of Agents and Multiagent Systems (PAAMS)*, volume 88, pages 129–134. Springer, 2011.

[BFH03] Fran BERMAN, Geoffrey FOX et Anthony JG HEY : *Grid computing : making the global infrastructure a reality*, volume 2. John Wiley and sons, 2003.

[BHM+] D BOOTH, H HAAS, F MCCABE, E NEWCOMER, M CHAMPION et C FERRIS : Orchard. d. ?web services architecture, w3c working draft 8 august 2003 ?

[BHPT98] Vincent BACHELET, Zouhir HAFIDI, Philippe PREUX et El-Ghazali TALBI : Vers la coopération des métaheuristiques. *Calculateurs parallèles, réseaux et systèmes répartis*, 9(2):211–223, 1998.

[BJ72] Franklin H BRANIN JR : Widely convergent method for finding multiple solutions of simultaneous nonlinear equations. *IBM Journal of Research and Development*, 16(5):504–522, 1972.

[BKB96] Benjamín BARÁN, Eugenius KASZKUREWICZ et Amit BHAYA : Parallel asynchronous team algorithms : Convergence and performance analysis. *Parallel and Distributed Systems, IEEE Transactions on*, 7(7):677–688, 1996.

[BPR01] Fabio BELLIFEMINE, Agostino POGGI et Giovanni RIMASSA : Developing multi-agent systems with jade. *In Intelligent Agents VII Agent Theories Architectures and Languages*, pages 89–103. Springer, 2001.

[BT00] Vincent BACHELET et E TALBI : Cosearch : a co-evolutionary metaheuristic. *In Evolutionary Computation, 2000. Proceedings of the 2000 Congress on*, volume 2, pages 1550–1557. IEEE, 2000.

[CBP+09] David CHAVALARIAS, Paul BOURGINE, Edith PERRIER, Fréderic AMBLARD, François ARLABOSSE, Pierre AUGER, Jean-Bernard BAILLON, Olivier BARRETEAU, Pierre BAUDOT, Elisabeth BOUCHAUD, Soufian BEN AMOR, Hugues BERRY, Cyrille BERTELLE, Marc BERTHOD, Guillaume BESLON, Giulio BIROLI, Daniel BONAMY, Daniele BOURCIER, Nicolas BRODU, Marc BUI, Yves BURNOD, Bertrand CHAPRON, Catherine CHRISTOPHE, Bruno CLÉMENT, Jean-Louis COATRIEUX, Jean-Philippe COINTET, Valérie DAGRAIN, Katia DAUCHOT, Olivier DAUCHOT, François DAVIAUD, Silvia DE MONTE, Guillaume DEFFUANT, Pierre DEGOND, Jean-Paul DELAHAYE, René DOURSAT, Francesco D'OVIDIO, A. DUBOIS, Marc, Berengère DUBRUELLE, Marie DUTREIX, Robert FAIVRE, Emmanuel FARGE, Patrick FLANDRIN, Sara FRANCESCHELLI, Cédric GAUCHEREL, Jean-Pierre GAUDIN, Michael GHIL, Jean-Louis GIAVITTO, Francesco GINELLI, Vincent GINOT, François HOULLIER, Bernard HUBERT, Pablo JENSEN, Ludovic JULLIEN, Zoi KAPOULA, Daniel KROB, François LADIEU, Gabriel LANG, Chrsitophe LAVELLE, André LE BIVIC, Jean-Pierre LECA, Christophe LECERF, Pierre LEGRAIN, Denis L'HÔTE, Maud LOIREAU, Jean-Francois MANGIN, Olivier MONGA, Michel MORVAN, Jean-Pierre MULLER, Ioan NEGRUTIU, Nadine PEYREIRAS, Denise PUMAIN, Ovidiu RADULESCU, Jean SALLANTIN, Eric SANCHIS, Daniel SCHERTZER, Marc SCHOENAUER, Michèle SEBAG, Eric SIMONET, Adrien SIX, Fabien TARISSAN et Patrick VINCENT : French Roadmap for complex Systems 2008-2009, mars 2009. This second issue of the French Complex Systems Roadmap by the French National Network for Complex SYstems (http ://RNSC.fr) and the Paris Ile-de-France Complex Systems Institute (http ://iscpif.fr) is the outcome of the "Entretiens de Cargèse 2008", an interdisciplinary brainstorming session organized over one week in 2008, jointly by RNSC, ISC-PIF and IXXI. It capitalizes on the first roadmap and gathers contributions of more than 70 scientists from major French institutions.

[CC09] Marc CHOISY et B CAZELLES : Conséquences des dynamiques épidémiques en santé publique : rôle des modèles mathématiques. 2009.

[CDFD10] Florent CHUFFART, Nicolas DUMOULIN, Thierry FAURE et Guillaume DEFFUANT : Simexplorer : Programming experimental designs on models and managing quality of modelling process. *International Journal of Agricultural and Environmental Information Systems (IJAEIS)*, 1(1):55–68, 2010.

[CDFD12] Florent CHUFFART, Nicolas DUMOULIN, Thierry FAURE et Guillaume DEFFUANT : Simexplorer : Programming experimental designs on models and managing quality of modelling process. *New Technologies for Constructing Complex Agricultural and Environmental Systems*, page 19, 2012.

[CGB+12] Rosaria CONTE, Nigel GILBERT, Guilia BONELLI, Claudio CIOFFI-REVILLA, Guillaume DEFFUANT, Janos KERTESZ, Vittorio LORETO, Suzy MOAT, J-P NADAL, Anxo SANCHEZ *et al.* : Manifesto of computational social science. *The European Physical Journal Special Topics*, 214(1):325–346, 2012.

[CGH94] Lyndon CLARKE, Ian GLENDINNING et Rolf HEMPEL : The mpi message passing interface standard. *In Programming environments for massively parallel distributed systems*, pages 213–218. Springer, 1994.

[CH06] Benoît CALVEZ et Guillaume HUTZLER : Automatic tuning of agent-based models using genetic algorithms. *In Proceedings of the 6th international conference on Multi-Agent-Based Simulation*, MABS'05, pages 41–57. Springer, 2006.

[CH+07] Benoît CALVEZ, Guillaume HUTZLER *et al.* : Adaptative dichotomic optimization : a new method for the calibration of agent-based models. *In A. Tanguy C. Bertelle J. Sklenar et G. Fortino, éditeurs, Proceedings of the 2007 European Simulation and Modelling Conference (ESM'07)*, pages 415–419, 2007.

[CMT04] Sébastien CAHON, Nordine MELAB et E-G TALBI : Paradiseo : A framework for the reusable design of parallel and distributed metaheuristics. *Journal of Heuristics*, 10(3):357–380, 2004.

[Cor91] Daniel D CORKILL : Blackboard systems. *AI expert*, 6(9):40–47, 1991.

[CQ13] Philippe CAILLOU et Javier Gil QUIJANO : Description automatique de dynamiques de groupes dans des simulations à base d'agents. simanalyzer : un outil générique pour l'analyse de simulations. *Revue d'Intelligence Artificielle*, 27(6):739–764, 2013.

[CT03] Teodor Gabriel CRAINIC et Michel TOULOUSE : *Parallel strategies for metaheuristics*. Springer, 2003.

[CT05] A Colin CAMERON et Pravin K TRIVEDI : *Microeconometrics : methods and applications*. Cambridge university press, 2005.

[CT07] Radovan CERVENKA et Ivan TRENCANSKY : *The Agent Modeling Language-AML : A Comprehensive Approach to Modeling Multi-Agent Systems*. Springer Science & Business Media, 2007.

[CT08] Teodor Gabriel CRAINIC et Michel TOULOUSE : *Explicit and emergent cooperation schemes for search algorithms*. Springer, 2008.

[CT10] Teodor Gabriel CRAINIC et Michel TOULOUSE : Parallel meta-heuristics. *In Handbook of metaheuristics*, pages 497–541. Springer, 2010.

[DBB10] Grégoire DANOY, Pascal BOUVRY et Olivier BOISSIER : A multi-agent organizational framework for coevolutionary optimization. *In Transactions on Petri nets and other models of concurrency IV*, pages 199–224. Springer, 2010.

[DBC+15] Guillaume DEFFUANT, Arnaud BANOS, David CHAVALARIAS, Cyrille BERTELLE, Nicolas BRODU, Pablo JENSEN, Annick LESNE, Jean-Pierre MÜLLER, Édith PERRIER et Franck VARENNE : Visions de la complexité. le démon de laplace dans tous ses états. *Natures Sciences Sociétés*, 23(1):42–53, 2015.

[DBQ12] Raphaël DUBOZ, Bruno BONTÉ et Gauthier QUESNEL : Vers une spécification des modèles de simulation de systèmes complexes. *Stud. Inform. Univ.*, 10(1):7–37, 2012.

[Dem95] Yves DEMAZEAU : From interactions to collective behaviour in agent-based systems. *In In : Proceedings of the 1st. European Conference on Cognitive Science. Saint-Malo*, pages 117–132, 1995.

[DGST09] Ewa DEELMAN, Dennis GANNON, Matthew SHIELDS et Ian TAYLOR : Workflows and e-science : An overview of workflow system features and capabilities. *Future Generation Computer Systems*, 25(5):528–540, 2009.

[DL06] Eric DAUDÉ et Patrice LANGLOIS : *Modélisation et simulation multi-agents application pour les Sciences de l'Homme et de la Société*, chapitre Comparaison de trois implémentations du modèle de Schelling, pages 411–441. Hermès, Paris, 2006.

[DM98] Leonardo DAGUM et Ramesh MENON : Openmp : an industry standard api for shared-memory programming. *Computational Science & Engineering, IEEE*, 5(1):46–55, 1998.

[DMBP11] Stéphane DONCIEUX, Jean-Baptiste MOURET, Nicolas BREDECHE et Vincent PADOIS : Evolutionary robotics : Exploring new horizons. *In New horizons in evolutionary robotics*, pages 3–25. Springer, 2011.

[DPL+] Frédéric DADEAU, Fabien PEUREUX, Bruno LEGEARD, Régis TISSOT, Jacques JULLIAND, Pierre-Alain MASSON et Fabrice BOUQUET : Model-based testing for embedded systems.

[DSV08] Travis DESELL, Boleslaw SZYMANSKI et Carlos VARELA : Asynchronous genetic search for scientific modeling on large-scale heterogeneous environments. *In Parallel and Distributed Processing, 2008. IPDPS 2008. IEEE International Symposium on*, pages 1–12. IEEE, 2008.

[DVM03] Alexis DROGOUL, Diane VANBERGUE et Thomas MEURISSE : Multi-agent based simulation : Where are the agents ? *In Proceedings of the 3rd International Conference on Multi-agent-based Simulation II*, MABS'02, pages 1–15, Berlin, Heidelberg, 2003. Springer-Verlag.

[EGGA+04] Tarek EL-GHAZAWI, Kris GAJ, Nikitas ALEXANDRIDIS, Frederic VROMAN, Nguyen NGUYEN, Jacek R RADZIKOWSKI, Preeyapong SAMIPAGDI et Suboh A SUBOH : A performance study of job management systems. *Concurrency and Computation : Practice and Experience*, 16(13):1229–1246, 2004.

[EVBM09] Jose A. EGEA, Emmanuel VAZQUEZ, Julio R. BANGA et Rafael MARTÍ : Improved scatter search for the global optimization of computationally expensive dynamic models. *J. of Global Optimization*, 43(2-3):175–190, mars 2009.

[FBB+16] Pierre FOSSET, Arnaud BANOS, Elise BECK, Sonia CHARDONNEL, Christophe LANG, Nicolas MARILLEAU, Arnaud PIOMBINI, Thomas LEYSENS, Alexis CONESA et Isabelle ANDRE-POYAUD : Exploring intra-urban accessibility and impacts of pollution policies with an agent-based simulation platform : Gamirod. *Systems*, 4(1):5, 2016.

[FGM04] Jacques FERBER, Olivier GUTKNECHT et Fabien MICHEL : From agents to organizations : An organizational view of multi-agent systems. *In Agent-Oriented Software Engineering IV*, pages 214–230. Springer, 2004.

[FIM+13] Robert FAIVRE, Bertrand IOOSS, Stéphanie MAHÉVAS, David MAKOWSKI et Hervé MONOD : *Analyse de sensibilité et exploration de modèles : application aux sciences de la nature et de l'environnement.* Editions Quae, 2013.

[Fis95] Paul A. FISHWICK : *Simulation Model Design and Execution : Building Digital Worlds*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 1st édition, 1995.

[FK97] Ian FOSTER et Carl KESSELMAN : Globus : A metacomputing infrastructure toolkit. *International Journal of High Performance Computing Applications*, 11(2):115–128, 1997.

[FKP05] Manuel FEHLER, Franziska KLÜGL et Frank PUPPE : Techniques for analysis and calibration of multi-agent simulations. *In Proceedings of the 5th International Conference on Engineering Societies in the Agents World*, ESAW'04, pages 305–321, Berlin, Heidelberg, 2005. Springer-Verlag.

[GDZ13] Arnaud GRIGNARD, Alexis DROGOUL et Jean-Daniel ZUCKER : Online analysis and visualization of agent based models. *In Computational Science and Its Applications–ICCSA 2013*, pages 662–672. Springer, 2013.

[Gen01] Wolfgang GENTZSCH : Sun grid engine : Towards creating a compute power grid. *In Cluster Computing and the Grid, 2001. Proceedings. First IEEE/ACM International Symposium on*, pages 35–36. IEEE, 2001.

[GK10] Joshua D. GRIFFIN et Tamara G. KOLDA : Asynchronous parallel hybrid optimization combining direct and gss. *Optimization Methods and Software*, 25(5):797–817, 2010.

[GKY08] Noam GOLDBERG, Tamara G. KOLDA et Ann S. YOSHIMURA : Concurrent optimization with duet : Direct using external trial points. Rapport technique, Sandia National Laboratories, 2008.

[Gou84] Christian GOURIEROUX : *Econométrie des variables qualitatives*. Economica, 1984.

[GP71] AA GOLDSTEIN et JF PRICE : On descent from local minima. *Mathematics of Computation*, 25(115):569–574, 1971.

[GSBT+13] Benoit GAUDOU, Christophe SIBERTIN-BLANC, Olivier THEROND, Frédéric AMBLARD, Jean-Paul ARCANGELI, Maud BALESTRAT, Marie-Hélène CHARRON-MOIREZ, Etienne GONDET, Yi HONG, Thomas LOUAIL *et al.* : The maelia multi-agent platform for integrated assessment of low-water management issues. *In MABS, Multi-Agent-Based Simulation XIV-International Workshop (to appear, 2013)*, 2013.

[GTG+13] Arnaud GRIGNARD, Patrick TAILLANDIER, Benoit GAUDOU, Duc An VO, Nghi Quang HUYNH et Alexis DROGOUL : Gama 1.6 : Advancing the art of complex agent-based modeling and simulation. *In PRIMA 2013 : Principles and Practice of Multi-Agent Systems*, pages 117–131. Springer, 2013.

[HGH99] Jin-Kao HAO, Philippe GALINIER et Michel HABIB : Métaheuristiques pour l'optimisation combinatoire et l'affectation sous contraintes. *Revue d'intelligence artificielle*, 13(2):283–324, 1999.

[HH06] Shun-Fa HWANG et Rong-Song HE : A hybrid real-parameter genetic algorithm for function optimization. *Advanced Engineering Informatics*, 20(1):7–21, 2006.

[HHM07] S. HIWA, T. HIROYASU et M. MIKI : Hybrid optimization using direct, ga, and sqp for global exploration. *In Evolutionary Computation, 2007. CEC 2007. IEEE Congress on*, pages 1709–1716, 2007.

[HHML07] J HERRERA, E HUEDO, R MONTERO et I LLORENTE : Gridway drmaa 1.0 implementation-experience report. *In Document GFD. E-104, DRMAA Working Group–Open Grid Forum*, 2007.

[Hil93] David R. C. HILL : *Analyse orientée objets et modélisation par simulation*. Addison-Wesley France, Paris, 1993. Sur la page de couv. : Avec un logiciel d'animation livré sous disquette.

[HKT01] Patricia D. HOUGH, Tamara G. KOLDA et Virginia J. TORCZON : Asynchronous Parallel Pattern Search for Nonlinear Optimization. *SIAM Journal on Scientific Computing*, 23(1):134–156, janvier 2001.

[HMP+13] The-Nhan HO, Nicolas MARILLEAU, Laurent PHILIPPE, Hong-Quang NGUYEN et Jean-Daniel ZUCKER : A grid-based multistage algorithm for parameter simulation-optimization of complex system. *In Computing and Communication Technologies, Research, Innovation, and Vision for the Future (RIVF), 2013 IEEE RIVF International Conference on*, pages 221–226, Nov 2013.

[JFT05] Akbar A JAVADI, Raziyeh FARMANI et Teng Puay TAN : A hybrid intelligent genetic algorithm. *Advanced Engineering Informatics*, 19(4):255–262, 2005.

[Joh08] Jeffrey JOHNSON : Science and policy in designing complex futures. *Futures*, 40(6):520 – 536, 2008. Design out of Complexity.

[JPS93] D.R. JONES, C.D. PERTTUNEN et B.E. STUCKMAN : Lipschitzian optimization without the lipschitz constant. *Journal of Optimization Theory and Applications*, 79(1):157–181, 1993.

[KF99] C KESSELMAN et I FOSTER : The grid : blueprint for a future computing infrastructure. *chapter2 Morgan Kaufmann Publication*, 1999.

[KLT03] Tamara G KOLDA, Robert Michael LEWIS et Virginia TORCZON : Optimization by direct search : New perspectives on some classical and modern methods. *SIAM review*, 45(3):385–482, 2003.

[KM32] William O KERMACK et Anderson G MCKENDRICK : Contributions to the mathematical theory of epidemics. ii. the problem of endemicity. *Proceedings of the Royal society of London. Series A*, 138(834):55–83, 1932.

[Kor09] Daniel KORNHAUSER : *Representation and parameter space visualization of agent based models*. Thèse de doctorat, Northwestern University, 2009.

[KR08] Matthew James KEELING et Pejman ROHANI : *Modeling infectious diseases in humans and animals*. Princeton University Press, Princeton, 2008.

[KS05] N. KRASNOGOR et J. SMITH : A tutorial for competent memetic algorithms : model, taxonomy, and design issues. *Evolutionary Computation, IEEE Transactions on*, 9(5):474–488, 2005.

[KSLC05] Jack P. C. KLEIJNEN, Susan M. SANCHEZ, Thomas W. LUCAS et Thomas M. CIOPPA : State-of-the-art review : A user's guide to the brave new world of designing simulation experiments. *INFORMS J. on Computing*, 17(3):263–289, juillet 2005.

[LACJ03] MH LYONS, Iqbal ADJALI, David COLLINGS et KO JENSEN : Complex systems models for strategic decision making. *BT Technology Journal*, 21(2):11–27, 2003.

[Lem09] Christiane LEMIEUX : *Monte Carlo and Quasi-Monte Carlo Sampling*, volume 20. Springer, 2009.

[LEP+06] Erwin LAURE, A EDLUND, F PACINI, P BUNCIC, M BARROSO, A DI MEGLIO, F PRELZ, A FROHNER, O MULMO, A KRENEK *et al.* : Programming the grid with glite. Rapport technique, 2006.

[LMSB+14] Romain LARDY, Pierre MAZZEGA, Christophe SIBERTIN-BLANC, Yves AUDA, José-Miguel SANCHEZ-PEREZ, Sabine SAUVAGE et Olivier THEROND : Calibration of simulation platforms including highly interweaved processes : the maelia multi-agent platform. *In Proceedings of the 7th International Congress on Environmental Modelling and Software*, June 15-19, Berlin, Heidelberg, 2014.

[Luk13] Sean LUKE : *Essentials of Metaheuristics*. Lulu, second édition, 2013. Available for free at http ://cs.gmu.edu/∼sean/book/metaheuristics/.

[LW] Yvan LABICHE et Gabriel WAINER : Towards the verification and validation of devs models. *In in Proceedings of 1st Open International Conference on Modeling and Simulation, 2005*, pages 295–305.

[Mai08] A MAIER : Ganga—a job management and optimising tool. *In Journal of Physics : Conference Series*, volume 119, page 072021. IOP Publishing, 2008.

[Mar06] Nicolas MARILLEAU : Méthodologie, formalismes et outils de modélisation-simulation pour l'étude des systèmes complexes : application à la mobilité géographique. *These de doctorat, Université de Franche-Comté*, 2006.

[Mat] Philippe MATHIEU.

[MCK08a] David MEIGNAN, J-C CREPUT et Abderrafiâa KOUKAM : A coalition-based metaheuristic for the vehicle routing problem. *In Evolutionary Computation, 2008. CEC 2008.(IEEE World Congress on Computational Intelligence). IEEE Congress on*, pages 1176–1182. IEEE, 2008.

[MCK08b] David MEIGNAN, Jean-Charles CRÉPUT et Abderrafiâa KOUKAM : An organizational view of metaheuristics. *In First International Workshop on Optimisation in Multi-Agent Systems, AAMAS*, volume 8, pages 77–85, 2008.

[MCM04] Pablo MOSCATO, Carlos COTTA et Alexandre MENDES : Memetic algorithms. *In New optimization techniques in engineering*, pages 53–85. Springer, 2004.

[MdW04] Christopher MAGEE et Olivier de WECK : Complex system classification. 2004.

[Min65] Marvin MINSKY : Matter, mind and models. 1965.

[MKC10] David MEIGNAN, Abderrafiaa KOUKAM et Jean-Charles CREPUT : Coalition-based metaheuristic : a self-adaptive metaheuristic using reinforcement learning and mimetism. *Journal of Heuristics*, 16(6):859–879, 2010.

[MM02] Rémi MUNOS et Andrew MOORE : Variable resolution discretization in optimal control. *Machine learning*, 49(2-3):291–323, 2002.

[Moi06] Alexandre MOINE : Le territoire comme un système complexe : un concept opératoire pour l'aménagement et la géographie. *L'Espace géographique*, (2): 115–132, 2006.

[Mon] Hervé MONOD : Plans d'expérience pour le criblage et l'analyse de sensibilit é.

[Mon08] Thomas MONCION : *Modélisation de la complexité et de la dynamique des simulations multi-agents : application pour l'analyse des phénomènes émergents.* Thèse de doctorat, 2008. 2008EVRY0038.

[Mor09] Gildas MORVAN : *Approche multi-agents d'un système d'aide à la décision en environnement dynamique incertain-Application à l'entomologie médico-légale.* Thèse de doctorat, Artois, 2009.

[MR04] Michela MILANO et Andrea ROLI : Magma : a multiagent architecture for metaheuristics. *Systems, Man, and Cybernetics, Part B : Cybernetics, IEEE Transactions on*, 34(2):925–941, 2004.

[MRS03] Philippe MATHIEU, Jean-Christophe ROUTIER et Yann SECQ : Rio : Roles, interactions and organizations. *In Multi-Agent Systems and Applications III*, pages 147–157. Springer, 2003.

[NCR+02] Edgar NODA, André LV COELHO, Ivan LM RICARTE, Akebo YAMAKAMI et Alex A FREITAS : Devising adaptive migration policies for cooperative distributed genetic algorithms. *In Systems, Man and Cybernetics, 2002 IEEE International Conference on*, volume 6, pages 6–pp. IEEE, 2002.

[Neu04] Arnold NEUMAIER : Complete search in continuous global optimization and constraint satisfaction. *Acta numerica*, 13:271–369, 2004.

[Ngu13] Trong Khanh NGUYEN : *Elaboration d'un processus et conception d'outils pour introduire la collaboration dans la simulation.* Thèse de doctorat, Université Pierre et Marie Curie-Paris 6, 2013.

[NLLA08] Antonio J NEBRO, Gabriel LUQUE, Francisco LUNA et Enrique ALBA : Dna fragment assembly using a grid-based genetic algorithm. *Computers & Operations Research*, 35(9):2776–2790, 2008.

[OB04] Godfrey C ONWUBOLU et BV BABU : *New optimization techniques in engineering*, volume 141. Springer, 2004.

[OL96] Ibrahim H OSMAN et Gilbert LAPORTE : Metaheuristics : A bibliography. *Annals of Operations Research*, 63(5):511–623, 1996.

[Óla06] Sigurdur ÓLAFSSON : Chapter 21 metaheuristics. *In* Shane G. HENDERSON et Barry L. NELSON, éditeurs : *Simulation*, volume 13 de *Handbooks in Operations Research and Management Science*, pages 633 – 654. Elsevier, 2006.

[OP10] Djamila OUELHADJ et Sanja PETROVIC : A cooperative hyper-heuristic search framework. *Journal of Heuristics*, 16(6):835–857, 2010.

[Par96] Jan PAREDIS : Coevolutionary life-time learning. *In Parallel Problem Solving from Nature—PPSN IV*, pages 72–80. Springer, 1996.

[Par11] Lael PARROTT : Hybrid modelling of complex ecological systems for decision support : Recent successes and future perspectives. *Ecological Informatics*, 6(1): 44–49, 2011.

[Pea09] Judea PEARL : *Causality : Models, Reasoning and Inference.* Cambridge University Press, New York, NY, USA, 2nd édition, 2009.

[Pre09] Vincent PRETRE : *Génération automatique de tests à partir de modèle formel pour les applications de type web services.* Thèse de doctorat, Université de Franche-Comté, 2009.

[PS07] Daniel J POWER et Ramesh SHARDA : Model-driven decision support systems : Concepts and research directions. *Decision Support Systems*, 43(3):1044–1061, 2007.

[Pum03] Denise PUMAIN : Une approche de la complexité en géographie. *Géocarrefour*, 78(1):25–31, 2003.

[RCL+10] Romain REUILLON, Florent CHUFFART, Mathieu LECLAIRE, Thierry FAURE, Nicolas DUMOULIN et David R. C. HILL : Declarative task delegation in OpenMOLE. *In Proceedings of the 2010 International Conference on High Performance Computing & Simulation, (HPCS)*, volume 3, pages 55–62. IEEE, 2010.

[Rey10] Sylvain REYNAUD : Uniform access to heterogeneous grid infrastructures with jsaga. *In Production grids in Asia*, pages 185–196. Springer, 2010.

[Rio09] François RIOUX : *Conception et mise en oeuvre de Multichronia, un cadre conceptuel de simulation visuelle interactive.* Thèse de doctorat, 2009.

[Rou07] William B. ROUSE : Complex engineered, organizational and natural systems : Issues underlying the complexity of systems and fundamental research needed to address these issues contract : National science foundation ; grant number 0538768. *Syst. Eng.*, 10(3):260–271, juin 2007.

[Roy96] B. ROY : *Multicriteria Methodology for Decision Aiding.* Kluwer Academic, Dordrecht, 1996.

[SB99] Richard S SUTTON et Andrew G BARTO : Reinforcement learning : An introduction. *Robotica*, 17(2):229–235, 1999.

[Sch69] Thomas C SCHELLING : Models of segregation. *The American Economic Review*, pages 488–493, 1969.

[SDV08] Boleslaw K SZYMANSKI, Travis DESELL et Carlos VARELA : The effects of heterogeneity on asynchronous panmictic genetic search. *In Parallel Processing and Applied Mathematics*, pages 457–468. Springer, 2008.

[Sim62] Herbert A. SIMON : The architecture of complexity. *In Proceedings of the American Philosophical Society*, pages 467–482, 1962.

[SK98] Hidefumi SAWAI et Sachio KIZU : Parameter-free genetic algorithm inspired by disparity theory of evolution. *In* AgostonE. EIBEN, Thomas BACK, Marc SCHOENAUER et Hans-Paul SCHWEFEL, éditeurs : *Parallel Problem Solving from Nature — PPSN V*, volume 1498 de *Lecture Notes in Computer Science*, pages 702–711. Springer Berlin Heidelberg, 1998.

[Spe08] J. Michael SPECTOR : Cognition and learning in the digital age : Promising research and practice. *Computers in Human Behavior*, 24(2):249 – 262, 2008. Part Special Issue : Cognition and Exploratory Learning in Digital Age.

[SRC11] Clara SCHMITT et Sébastien REY COYREHOURCQ : Exploring geographical agent based models : a new protocol using automated procedures. *In European Colloquium on Quantitative and Theoritical Geography*, Athène, Grèce, 2011.

[Sto11] Forrest J STONEDAHL : *Genetic Algorithms for the Exploration of Parameter Spaces in Agent-Based Models*. Thèse de doctorat, NORTHWESTERN UNIVERSITY, 2011.

[SW10] Forrest STONEDAHL et Uri WILENSKY : Finding forms of flocking : Evolutionary search in abm parameter spaces. *In In Proceedings of the MABS workshop at the 9th international conference on Autonomous Agents and Multi-Agent Systems*, 2010.

[SZS09] Wanfeng SHANG, Shengdun ZHAO et Yajing SHEN : A flexible tolerance genetic algorithm for optimal problems with nonlinear equality constraints. *Advanced Engineering Informatics*, 23(3):253–264, 2009.

[TA+11] Patrick TAILLANDIER, Frédéric AMBLARD *et al.* : Cartography of multi-agent model parameter space through a reactive dicotomous approach. *In European Simulation and Modelling Conference*, pages 38–42, 2011.

[Tal02] E-G TALBI : A taxonomy of hybrid metaheuristics. *Journal of heuristics*, 8(5):541–564, 2002.

[Tar05] Albert TARANTOLA : *Inverse problem theory and methods for model parameter estimation*. siam, 2005.

[TB06a] El-Ghazali TALBI et Vincent BACHELET : Cosearch : A parallel cooperative metaheuristic. *Journal of Mathematical Modelling and Algorithms*, 5(1):5–22, 2006.

[TB06b] El-Ghazali TALBI et Vincent BACHELET : Cosearch : A parallel cooperative metaheuristic. *Journal of Mathematical Modelling and Algorithms*, 5(1):5–22, 2006.

[TCM+06] William M. TROCHIM, Derek A. CABRERA, Bobby MILSTEIN, Richard S. GALLAGHER et Scott J. LEISCHOW : Practical Challenges of Systems Thinking and Modeling in Public Health. *Am J Public Health*, 96(3):538–546, mars 2006.

[TCS99] Michel TOULOUSE, Teodor Gabriel CRAINIC et Brunilde SANSÓ : An experimental study of systemic behavior of cooperative search algorithms. *In Meta-Heuristics*, pages 373–392. Springer, 1999.

[TDZ08] Jean-Pierre TREUIL, Alexis DROGOUL et Jean-Daniel ZUCKER : Modélisation et simulation à base d'agents. *Exemples commentés, outils informatiques et questions théoriques. Dunod*, 2008.

[TTL05] Douglas THAIN, Todd TANNENBAUM et Miron LIVNY : Distributed computing in practice : The condor experience. *Concurrency and Computation : Practice and Experience*, 17(2-4):323–356, 2005.

[ULB+15] Mark UTTING, Bruno LEGEARD, Fabrice BOUQUET, Elizabeta FOURNERET, Fabien PEUREUX et Alexandre VERNOTTE : Recent advances in model-based testing. *In Advances in Computers*. Elsevier, 2015.

[Wil99] Uri WILENSKY : {NetLogo}. 1999.

[WM97] David H WOLPERT et William G MACREADY : No free lunch theorems for optimization. *Evolutionary Computation, IEEE Transactions on*, 1(1):67–82, 1997.

[XNH10] Jie XU, Barry L NELSON et JEFF HONG : Industrial strength compass : A comprehensive algorithm and software for optimization via simulation. *ACM Transactions on Modeling and Computer Simulation (TOMACS)*, 20(1):3, 2010.