Génération d'algorithmes de recherche locale

Vincent Hénaux

▶ **To cite this version:**

Vincent Hénaux. Génération d'algorithmes de recherche locale. Informatique et langage [cs.CL]. Université d'Angers, 2021. Français. NNT : 2021ANGE0026 . tel-03591753

HAL Id: tel-03591753

https://theses.hal.science/tel-03591753v1

Submitted on 28 Feb 2022

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

DOCTORAT BRETAGNE LOIRE / MATHSTIC

université angers

THÈSE DE DOCTORAT

DE L'UNIVERSITÉ D'ANGERS
COMUE UNIVERSITÉ BRETAGNE LOIRE

ÉCOLE DOCTORALE N° 601
Mathématiques et Sciences et Technologies de l'Information et de la Communication
Spécialité : *Informatique*

Par
Vincent HÉNAUX

Génération d'algorithmes de recherche locale

Thèse présentée et soutenue à l'université d'Angers - UFR Sciences - L004, le 18/11/2021
Unité de recherche : LERIA
Thèse N° :

Rapporteurs avant soutenance :

Cyril Fonlupt	Professeur des universités à l'université du Littoral – Côte d'Opale
Nouredine Melab	Professeur des universités à l'université Lille 1

Composition du Jury :

Présidente :	Évelyne Lutton	Directrice de recherche à l'INRAE dpt MIA
Examinateurs :	David Lesaint	Professeur des universités à l'université d'Angers
Dir. de thèse :	Adrien Goëffon	Maître de conférence HDR à l'université d'Angers
Co-dir. de thèse :	Frédéric Saubion	Professeur des universités à l'université d'Angers

RÉSUMÉ

Résoudre un problème d'optimisation consiste à en trouver les meilleures solutions possibles. Pour y parvenir, une approche commune est d'utiliser des algorithmes spécifiques, en général conçus pour des classes de problèmes précises. Cette approche souffre néanmoins de deux désavantages. D'abord à chaque nouveau type de problème, un nouvel algorithme doit souvent être défini, ce qui est un processus long, nécessitant une connaissance des propriétés du problème en question. Ensuite, si ces algorithmes ne sont testés que sur certaines instances du problème, il est possible qu'ils s'avèrent trop spécifiques et donc finalement moins performants sur l'ensemble des instances de la classe. Dans ce travail de thèse, nous explorons la possibilité de générer automatiquement des algorithmes d'optimisation pour un problème donné. Le processus de génération reste suffisamment générique tandis que les algorithmes ainsi produits peuvent être très spécifiques afin d'être les plus efficaces possibles.

Plus précisément, nous faisons évoluer de simples algorithmes de recherche par voisinage via les fonctions d'évaluation qu'ils utilisent pour explorer l'espace des solutions du problème. Le processus évolutionnaire permet implicitement d'adapter le paysage de recherche à la stratégie de résolution basique, tout en conservant une cohérence avec la fonction objectif initiale du problème à résoudre. Ce processus de génération est testé sur deux classes de problèmes dont les difficultés sont très différentes, et obtient des résultats encourageants. Cette expérimentation est complétée par une analyse du processus de génération et des algorithmes ainsi générés.

Solving an optimization problem is about finding the best possible solutions. To achieve this, a common approach is to use specific algorithms, usually designed for specific classes of problems. However, this approach suffers from two disadvantages. First with each new type of problem, a new algorithm often has to be defined, which is a lengthy process, requiring knowledge of the properties of the problem in question. Then, if these algorithms are only tested on certain instances of the problem, it is possible that they turn out to be too specific and therefore ultimately less efficient on all the instances of the class. In this thesis work, we explore the possibility of automatically generating optimization algorithms for a given problem. The generation process remains sufficiently generic while the algorithms thus produced can be very specific in order to be as efficient as possible.

More precisely, we develop simple neighborhood search algorithms via the evaluation func-

tions that they use to explore the problem solution space. The evolutionary process implicitly makes it possible to adapt the search landscape to the basic solution strategy, while maintaining consistency with the initial objective function of the problem to be solved. This generation process is tested on two classes of problems whose difficulties are very different, and obtains encouraging results. This experiment is completed by an analysis of the generation process and of the algorithms thus generated.

Sommaire

Introduction **19**

1 Contexte et état de l'art **23**

1.1 Problèmes d'optimisation . . . 23
1.1.1 Problèmes étudiés . . . 25
Problème NK . . . 25
Problème MAX-SAT . . . 27
Problèmes boîte noire . . . 28
1.2 Méthodes de résolution . . . 29
1.2.1 Résolution exacte . . . 29
1.2.2 Résolution approchée . . . 31
1.2.3 Métaheuristiques . . . 31
Méthodes à base de voisinage . . . 32
Recherche locale . . . 33
Hill climber . . . 35
Règles pivots . . . 35
Hill climber, performances et caractéristiques . . . 36
Recherche locale itérée . . . 37
Recherche locale guidée . . . 37
Recherche à voisinage variable . . . 38
Algorithmes génétiques . . . 39
Stratégies d'évolution . . . 41
Programmation génétique . . . 42
1.3 Apprentissage pour l'optimisation . . . 43
1.3.1 Modèles surrogate . . . 43
1.3.2 Apprentissage de fonction d'évaluation . . . 44
1.3.3 Configuration automatique de paramètres . . . 45
1.4 Paysage de fitness . . . 46
1.4.1 Épistasie et rugosité . . . 47
1.4.2 Bassins d'attraction . . . 48
1.4.3 Réseaux d'optima locaux . . . 48

1.4.4 Visualisation des paysages de fitness 49

2 Motivations **53**
2.1 Rechercher une solution ou rechercher un algorithme 53
2.2 Objectif 55
2.2.1 Vision d'un point de vue configuration automatique 55
2.2.2 Vision d'un point de vue paysage de fitness 56
2.2.3 Problématique 57
2.3 Plan du manuscrit 58

3 Génération d'algorithmes **61**
3.1 Modèles d'algorithmes 61
3.1.1 Recherche locale stochastique avec montée stricte 61
3.1.2 Recherche locale déterministe de montée stricte à départ unique fixé . . . 62
3.1.3 Recherche locale déterministe de montée stricte à départs multiples fixés . 64
3.2 Modèles de fonction d'évaluation 64
3.2.1 Modèle NK 65
3.2.2 Modèle MAX-3SAT 65
3.2.3 Modèle de somme de fonctions pseudo-booléennes 66
3.2.4 Mutation de fonction d'évaluation 69
3.3 Sélection de mutations d'algorithmes 70
3.3.1 Sélection d'algorithmes déterministes 71
3.3.2 Sélection d'algorithmes stochastiques 71
Dominance statistique 72
Distribution uniforme de budget 72
Sélection UCB 73
Sélection SoftMax 73
Sélection par tournoi 74
3.4 Évolution d'algorithmes 74
3.4.1 Fonction complémentaire 74
3.4.2 (1+1)-ES 75
3.4.3 (1,λ)-ES 77

4 Résultats expérimentaux **81**
4.1 Fonctions complémentaires et preuve de concept 81
4.1.1 Protocole expérimental 81
4.1.2 Résultats 83
4.2 Fonctions alternatives 84

4.2.1 Protocole expérimental . . . 84
4.2.2 Résultats . . . 84
4.3 Génération d'algorithmes stochastiques . . . 89
4.3.1 Protocole expérimental . . . 89
4.3.2 Résultats . . . 91
4.4 Génération d'algorithmes déterministes . . . 92
4.4.1 Protocole expérimental . . . 94
4.4.2 Résultats . . . 96
4.5 Sélection d'algorithmes stochastiques . . . 96
4.5.1 Méthodes de sélection . . . 98
4.5.2 Protocole expérimental . . . 101
4.5.3 Résultats . . . 103
4.6 Expérimentation étendue . . . 104
4.6.1 Protocole expérimental . . . 106
Description des composants . . . 106
Paramétrage des algorithmes de génération . . . 107
Protocole de génération . . . 111
Algorithmes de référence . . . 111
4.6.2 Résultats . . . 112

5 Analyses comportementales 117

5.1 Analyses des composants de la génération d'algorithmes . . . 117
5.1.1 Effets des mutations . . . 117
Protocole expérimental . . . 118
Résultats . . . 118
5.1.2 Effets de la précision de l'évaluation . . . 119
Protocole expérimental . . . 119
Résultats . . . 120
5.1.3 Pertinence des modèles de fonction . . . 121
Protocole expérimental . . . 121
Effet de la neutralité du modèle . . . 121
Comparaison des modèles . . . 122
Résultats . . . 122
Effet de la neutralité du modèle . . . 122
Comparaison des modèles . . . 122
5.1.4 Comparaison des algorithmes déterministes et stochastiques . . . 124
Protocole expérimental . . . 124
Résultats . . . 124

5.1.5 Effet de la solution de départ pour les algorithmes déterministes à point de départ unique fixé . . . 127
Protocole expérimental . . . 127
Résultats . . . 127
5.2 Analyses des fonctions et algorithmes générés . . . 128
5.2.1 Apparition, évolution, et destruction des optima locaux dans les fonctions générées . . . 129
Protocole expérimental . . . 129
Résultats . . . 129
5.2.2 Évolution des algorithmes déterministes . . . 131
Protocole expérimental . . . 131
Résultats . . . 132
5.2.3 Épistasie des fonctions d'évaluation générées . . . 134
Protocole expérimental . . . 135
Résultats . . . 135
5.2.4 Interactions entre variables . . . 136
Protocole expérimental . . . 136
Résultats . . . 138
5.2.5 Interactions entre sous-fonctions . . . 138
Protocole expérimental . . . 140
Résultats . . . 140
5.2.6 Visualisation des fonctions d'évaluation . . . 141
Algorithme de placement de sommets de l'hypercube sur le plan . . . 142
Exemples de visualisation de différentes fonctions d'évaluation . . . 143

Conclusion **147**

Annexe **151**

Bibliographie **165**

Introduction

Un problème d'optimisation combinatoire est défini par un ensemble de solutions, vérifiant d'éventuelles contraintes du problème, ainsi qu'une fonction permettant d'évaluer ces solutions vis à vis d'un objectif à atteindre. Résoudre un tel problème consiste alors à trouver une solution optimale relativement à cette fonction objectif. Une fois que l'on dispose d'algorithmes d'optimisation performants pour accomplir cette tâche, analyser les problèmes ainsi que ces méthodes de résolution constitue un axe de recherche qui s'appuie notamment sur le concept de paysages de fitness. Ce concept a d'abord été proposé en biologie par Wright [1] pour l'étude des effets de processus de d'évolution comme la sélection ou la mutation. Il a été repris ensuite pour l'analyse des méthodes de résolution évolutionnaires. En optimisation combinatoire, un paysage de fitness est défini par un ensemble discret de solutions, une fonction d'évaluation qui associe un score à chaque solution et une fonction/relation de voisinage qui lie les solutions proches entre elles. Ces différents composants déterminent les caractéristiques du paysage qui peuvent être alors décrites en utilisant des analogies topographiques (plateaux, vallées...). L'analyse de ces paysages de fitness permet entre autres de s'intéresser à la présence d'optima locaux, c'est à dire de solutions dont les solutions voisines ont des valeurs inférieures (dans le cas de problèmes de maximisation). L'objectif de ces analyses est à la fois de mieux comprendre les algorithmes et leur comportement relativement aux problèmes à résoudre, mais aussi de proposer des améliorations de ces algorithmes en utilisant les connaissances ainsi acquises. Les travaux de recherche menés au cours de cette thèse se situent dans ce contexte d'amélioration de méthodes génériques de résolution de problèmes combinatoires.

Dans notre approche, les algorithmes vont être construits spécifiquement pour certaines classes de problèmes afin de pouvoir exploiter leurs structures propres. Notre postulat est de considérer que bien souvent les fonctions objectif initiales ne peuvent être exploitées facilement par des méthodes simples d'optimisation telles qu'une recherche locale qui procéderait par recherches successives d'une solution améliorante dans un voisinage basique. Ces fonctions objectif sont en effet définies en priorité pour évaluer les solutions mais ne sont pas nécessairement adaptées aux propriétés des méthodes qui les utilisent, notamment au sein des paysages de recherche qu'elles induisent. Typiquement, deux solutions proches relativement à leur encodage peuvent avoir des évaluations très différentes et inversement. Dès lors, plusieurs pistes s'offrent à nous pour pallier cet écueil. Il est par exemple possible d'adapter un algorithme au problème par

le biais de paramètres et heuristiques ad hoc. Il est également possible de trouver un meilleur codage des solutions et/ou des contraintes du problème et modifier ainsi la fonction objectif. De telles approches sont malheureusement souvent spécifiques à un problème donné et donnent alors lieu à des algorithmes complexes et peu compréhensibles quant à leur succès pour résoudre les problèmes. Dans cette thèse, nous choisissons de conserver la simplicité de la méthode de résolution mais d'adapter automatiquement le paysage de recherche des problèmes en modifiant les fonctions d'évaluation des solutions. Ainsi, nous proposons un moyen de faire évoluer des paysages de recherche pour les rendre plus simples à exploiter et donc les rendre plus faciles à optimiser avec des algorithmes simples et génériques.

L'objectif de cette thèse est de proposer une méthode de génération d'algorithmes d'optimisation, et plus précisément, d'algorithmes de recherche locale de montée stricte (hill climbers). La navigation dans le paysage de fitness s'opère par améliorations successives d'une solution courante en recherchant une meilleure solution dans son voisinage et en itérant ce procédé tant que cela est possible. Ainsi, un tel algorithme retourne une solution optimale localement (un optimum local) du paysage que l'on espère être un optimum global du problème (ou proche d'un optimum global relativement à la fonction objectif). L'algorithme est complètement guidé dans le paysage de fitness par la fonction d'évaluation qu'il utilise et qui est généralement la fonction objectif du problème initial. Malheureusement, la présence d'optima locaux peut nuire à l'efficacité de cet algorithme simple et l'empêcher d'atteindre un optimum global. Une façon de modifier le paysage est de modifier la fonction d'évaluation (de fitness) pour que l'algorithme atteigne plus facilement un optimum global. La fonction d'évaluation peut être vue comme l'un des composants essentiels de l'algorithme. Toutefois, il conviendra de conserver une certaine cohérence entre les nouvelles fonctions modifiées et la fonction objectif initiale dont on recherche un optimum global.

Nous proposons alors d'utiliser un processus évolutionnaire pour générer de nouvelles fonctions d'évaluation qui permettront à un hill climber simple d'atteindre de meilleures solutions. Ceci peut donc être vu comme une manière de faire évoluer le paysage de fitness via sa fonction d'évaluation pour le rendre plus exploitable par notre algorithme de résolution. Naturellement, il s'agit finalement de transposer la résolution du problème dans l'espace de recherche initial à la recherche d'une meilleure fonction d'évaluation dans l'espace des paysages de fitness ainsi générés (ou plus sobrement l'espace des fonctions d'évaluation). Toutefois, notre approche va consister à considérer des modèles de fonctions d'évaluation génériques qui pourront être différents de la modélisation initiale de la fonction objectif. En ce sens, notre objectif est réellement d'analyser la généricité d'une telle approche par rapport au développement de techniques ad hoc ou à l'optimisation d'algorithmes via leurs paramètres classiques. Pour valider nos propositions, nous utiliserons plusieurs problèmes de référence afin d'évaluer nos algorithmes de génération : les modèles NK dont la difficulté est paramétrable, et le problème MAX-SAT qui présente des

propriétés différentes.

Le chapitre 1 s'organise en quatre parties. La première partie définit plus formellement les problèmes d'optimisation ainsi qu'un exemple d'un tel problème afin d'apprécier plus concrètement les difficultés que les algorithmes d'optimisation peuvent rencontre. La seconde partie aborde la résolution de problèmes d'optimisation qu'elle soit exacte ou approchée ainsi qu'un certain nombre d'algorithmes tels que les hill climbers ou les stratégies d'évolution. La troisième partie se penche sur les algorithmes et les approches connexes à la notre, à savoir les modèles surrogate, l'apprentissage de fonction d'évaluation et la configuration automatique de paramètres. Enfin, la quatrième partie présente et définit le concept de paysage de fitness ainsi que certaines caractéristiques associées comme l'épistasie, la rugosité et les bassins d'attraction.

Le chapitre 2 fait l'interface entre le contexte présenté et les contributions de la thèse en abordant les motivations qui ont menées à ce travail, ainsi que des différences notables avec les travaux antérieurs. Nous définissons également notre objectif plus formellement.

Le chapitre 3 présente l'ensemble des processus et des différents composants nécessaires à la génération de nouvelles fonctoins d'évaluation. Nous présentons d'abord les modèles d'algorithmes puis les différents modèles nécessaires à l'évolution des fonctions d'évaluation dans les paysages de fitness. Nous détaillons les algorithmes évolutionnaires utilisés ainsi que le processus de sélection mis en oeuvre au sein de ces méthodes.

Le chapitre 4 présente l'ensemble des expérimentations réalisées au cours de la thèse dans un ordre chronologique afin de mieux rendre compte du cheminement suivi. La finalité de ce chapitre est de proposer une version de l'algorithme de génération qui pourra générer des algorithmes efficaces pour un ensemble varié d'instances de problèmes d'optimisation. La dernière section de ce chapitre ainsi alors l'algorithme le plus abouti.

Le chapitre 5 propose différentes analyses comportementales. Il est organisé deux parties. La première présente l'ensemble des analyses qui ont trait aux composants de l'algorithme de génération, comme l'analyse des effets des mutations ou la comparaison des algorithmes stochastiques et déterministes. La seconde partie présente les analyses qui s'intéressent aux fonctions et algorithmes générés, notamment leur processus d'évolution mais aussi les structures que l'on peut retrouver ou non dans les fonctions d'évaluation.

Enfin, nous concluons ce manuscrit en résumant les différents apports de nos travaux pour la génération d'algorithmes ainsi que pour l'évolution de fonctions. Nous identifions ensuite quelques pistes de recherche pour notre approche.

Chapitre 1

Contexte et état de l'art

Le travail de thèse que nous présentons ici s'inscrit dans le contexte de l'optimisation combinatoire et plus précisément des techniques génériques de résolution approchée issues des métaheuristiques, du calcul évolutionnaire et des paysages adaptifs. Ce chapitre est dédié à la présentation de ce contexte et des principales notions utilisées dans la suite du manuscrit. Il s'organise d'abord en rappelant quelques notions de base relatives aux problèmes d'optimisation — en particulier ceux qui sont considérés dans nos travaux —, les principaux algorithmes de recherche basés sur le concept de voisinage, ainsi qu'un propos introductif sur les paysages de fitness.

1.1 Problèmes d'optimisation

Résoudre un problème d'optimisation consiste à déterminer un meilleur élément (ou tous les meilleurs éléments) d'un ensemble de solutions $\mathcal{X}$ selon un ou plusieurs critères. Un critère est habituellement modélisé par une fonction objectif f_{obj} qui associe un score à chaque élément de $\mathcal{X}$. Ici, nous nous concentrons sur les problèmes qui ne possèdent qu'un seul objectif, dits *mono-objectif*. Chaque élément de $\mathcal{X}$ représentant une solution du problème d'optimisation, le but est de trouver la meilleure solution selon f_{obj}, c'est-à-dire une solution dont le score est optimal.

Définition 1 ***Problème d'optimisation (mono-objectif)*** **:** *Soit un ensemble de solutions $\mathcal{X}$ appelé espace de recherche, et une fonction objectif $f_{\text{obj}} : \mathcal{X} \mapsto \mathbb{K}$ qui associe à chaque solution de l'ensemble $\mathcal{X}$ un score défini sur un ensemble $\mathbb{K}$ totalement ordonné. Un problème d'optimisation, décrit par un couple $(\mathcal{X}, f_{\text{obj}})$, consiste à déterminer $s \in \mathcal{X}$ de manière à maximiser ou minimiser $f_{\text{obj}}(x)$.*

Définition 2 ***Solution optimale*** **:** *Étant donné un problème d'optimisation $(\mathcal{X}, f_{\text{obj}})$ et un type de critère (max ou min), une solution optimale est une solution $s^* \in \mathcal{X}$ qui optimise la*

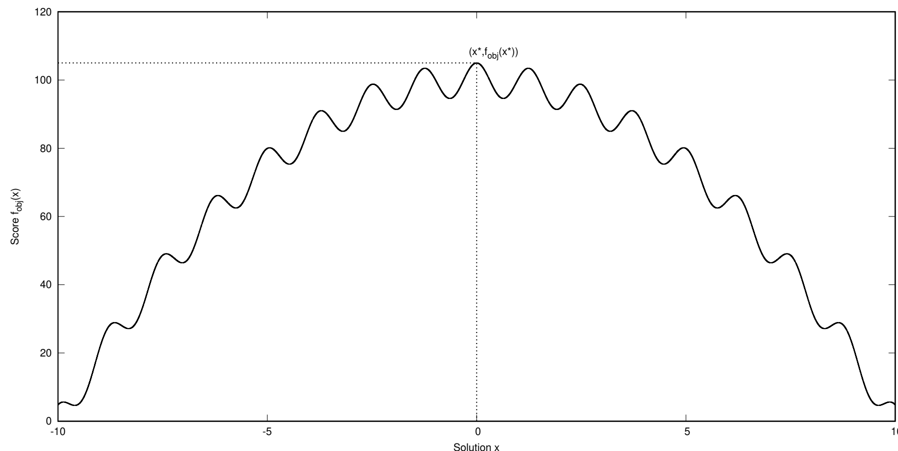
fonction objectif :

- *pour un problème de maximisation* $s^* = \text{argmax}_{s \in \mathcal{X}} f_{\text{obj}}(s)$*, c'est-à-dire une solution* $s^* \in \mathcal{X}$ *telle que* $\forall s \in \mathcal{X}, f_{\text{obj}}(s) \leq f_{\text{obj}}(s^*)$ *;*
- *pour un problème de minimisation* $s^* = \text{argmin}_{s \in \mathcal{X}} f_{\text{obj}}(s)$*, c'est-à-dire une solution* $s^* \in \mathcal{X}$ *telle que* $\forall s \in \mathcal{X}, f_{\text{obj}}(s) \geq f_{\text{obj}}(s^*)$*.*

Naturellement, à chaque problème de minimisation correspond un problème de maximisation équivalent. Tout problème de minimisation $(\mathcal{X}, f_{\text{obj}})$ peut en effet être transformé en un problème de maximisation $(\mathcal{X}, -f_{\text{obj}})$ et traité en tant que tel. Comme les algorithmes appelés *climbers* (littéralement *grimpeurs*) et les paysages de fitness constituent deux des éléments centraux des contributions de cette thèse, induisant dans les deux cas des notions d'ascension et d'adaptation, donc de recherche des solutions les plus hautes, un problème d'optimisation désignera dans la suite du manuscrit le problème de maximisation associé. Notons qu'*a contrario*, c'est parfois le critère de minimisation qui est fixé par convention, en particulier en optimisation continue, et les terminologies peuvent alors varier (algorithme de descente par exemple).

La figure 1.1 présente un exemple de problème d'optimisation, avec la solution optimale associée. Résoudre le problème consiste à déterminer cette solution.

FIGURE 1.1 – Représentation visuelle d'un problème d'optimisation continue, avec l'espace de recherche $\mathcal{X} = [-10, 10]$ et le graphe de la fonction objectif $f_{\text{obj}}(x) = -x^2 + 5\cos 5x + 100$ à maximiser. $x^* = 0$ est une solution optimale, car $\forall x \in \mathcal{X}, f_{\text{obj}}(x) \leq f_{\text{obj}}(x^*)$.



Les problèmes d'optimisation sont généralement rassemblés en classes, selon des caractéristiques communes liées à l'espace de recherche ou à la forme de la fonction objectif. Une classe de problème peut ainsi être aussi large que la définition d'un espace de recherche particulier (comme l'optimisation de fonctions continues de variables réelles, les problèmes d'optimisation

pseudo-booléens, ou les problèmes d'optimisation dans les graphes), mais se restreint communément à un modèle plus ou moins spécifique d'une fonction objectif paramétrique (par exemple, les problèmes NK présentés ci-après). Selon le contexte, un problème peut aussi bien dénoter une *classe* de problèmes — un ensemble d'*instances de problèmes* pouvant être défini en intention au moyen d'un modèle — ou bien dénoter une instance en particulier.

Ce travail de thèse s'inscrit dans le domaine de l'optimisation combinatoire. Nous nous consacrons ainsi aux problématiques de résolution propres aux problèmes d'optimisation combinatoire, dont l'espace de recherche $\mathcal{X}$ est discret, c'est-à-dire en bijection avec une partie finie de $\mathbb{N}$. Plus particulièrement, nous nous concentrerons dans la suite du manuscrit sur des problèmes pseudo-booléens, dont la fonction objectif est de la forme $f_{\text{obj}} : \{0,1\}^n \to \mathbb{R}$ (ou $\mathbb{N}$). n dénote la taille du problème. L'espace de recherche est de cardinalité 2^n, donc exponentielle en la taille du problème.

1.1.1 Problèmes étudiés

Le travail décrit dans les prochains chapitres de ce manuscrit s'appuie sur une conséquente partie expérimentale, d'où la nécessité de restreindre les classes de problèmes étudiées. Nous avons ainsi privilégié les problèmes NK et MAX-SAT, que nous considérons comme deux problèmes fondamentaux de l'optimisation évolutionnaire.

Les fonctions NK ont la particularité d'être paramétrables et de constituer des benchmarks pour étudier les liens entre les caractéristiques d'une instance et le comportement d'un algorithme de recherche locale ou évolutionnaire. Le problème MAX-SAT généralise le problème de satisfiabilité, et consiste à maximiser le nombre de clauses satisfaites d'une formule booléenne en forme normale conjonctive. Les fonctions MAX-SAT partagent certaines propriétés avec les fonctions NK (en particulier l'interaction entre certaines variables) tout en exhibant des difficultés spécifiques, dues notamment à une discrétisation moins fine du codomaine de la fonction objectif. Ces deux modèles de fonctions booléennes sont classiquement les plus étudiées en optimisation évolutionnaire lorsqu'il s'agit de construire des paysages de fitness ayant les propriétés souhaitées afin d'analyser le comportement d'algorithmes de recherche [2, 3, 4]. La suite de cette section présente plus précisément ces deux problèmes étudiés.

Problème NK

Les problèmes NK [5] sont définis par deux matrices dont la taille est fonction de deux paramètres N et K. N est la taille du problème, donc la longueur de la chaîne de bits représentant une solution. Chacune des N variables du problème intéragit avec au moins K autres variables. Ces interactions définissent des dépendances entre elles pour le calcul du score des solutions.

La fonction objectif d'une instance NK est composée de N sous-fonctions pseudo-booléennes. Chaque sous-fonction fait appel à $K+1$ bits de la solution (dans la définition générale du problème, au bit du numéro de la sous-fonction, puis K autres bits distincts) et associe à chaque combinaison de valeurs une contribution réelle entre 0 et 1. Le paramètre K représente ainsi le niveau d'épistasie du paysage associé (voir section 1.4.1) et par extension la difficulté d'une instance à être résolue au moyen d'un algorithme de recherche.

Définition 3 ***Problème NK*** **:** *Soit N fonctions pseudo-booléennes $\gamma_1, \ldots, \gamma_N$ où chacune est définie, en extension, sur $K+1$ variables booléennes ($\gamma_i : \{0,1\}^{K+1} \rightarrow [0;1]$). Le score NK d'une solution $s = (x_i)_{1 \leq i \leq N} \in \{0,1\}^N$ est égal à $f_{\mathrm{NK}}(s) = \frac{1}{N} \sum_{i=1}^{N} \gamma_i(x_i, l_{i1}, \ldots, l_{iK})$, avec l_{ij} la j-ème variable reliée à x_i par γ_i. Résoudre une instance de problème NK consiste donc à calculer* $\mathrm{argmax}_{s \in \{0,1\}^N} f_{\mathrm{NK}}(s)$.

Le tableau 1.1 présente un exemple d'instance NK de paramètres $N = 8$ et $K = 2$ et le calcul du score d'une solution. D'après la matrice des contributions, la première sous-fonction de cette instance renverra une contribution de 0.136 si les bits 1, 2 et 6 de la solution évaluée sont tous fixés à vrai (111). L'évaluation d'une solution est la moyenne des contributions renvoyées par les N sous-fonctions. Le calcul du score de la solution exemple 01000101 correspond donc à la moyenne des contributions indiquées en gras ; on a $f_{\mathrm{obj}}(01000101) = 4.652/8 = 0.5815$. Chaque ligne représente $K+1$ variables en interaction et la sous-fonction pseudo-booléenne complète associée. On remarque bien que la ligne i lie le bit numéro i à K autres bits distincts. Cet entrelacement des variables est là où réside le potentiel de difficulté des instances. En effet, plus une variable est liée à un nombre important d'autres variables (variables distinctes se retrouvant dans la définition des sous-fonctions dans lesquelles elle apparaît), plus l'influence sur le score de la modification de sa valeur sera dépendante du contexte, c'est-à-dire de la valeur des variables inchangées, entraînant de fait une décorrélation entre le score obtenu pour une solution et sa distance à la solution optimale.

Le paramètre K est nécessairement compris entre 0 et $N-1$. Dans le cas $K = 0$, le problème résultant est unimodal et donc ne pose pas de difficulté à être optimisé. À l'opposé, dans le cas $K = N-1$, tous les scores des solutions sont indépendants ; aucune solution ne partage dans ce cas d'information (via une contribution) avec une autre dans l'espace de recherche. Ainsi, pour ce cas particulier, il devient inutile d'utiliser des méthodes de résolution qui tendent à exploiter la fonction objectif et une stratégie d'échantillonnage purement exploratoire sera ainsi au moins aussi efficace que toute autre stratégie.

Table 1.1 – Exemple d'instance NK ($N = 8$,$K = 2$)

Liens			Contributions au score							
1	2	3	000	001	010	011	100	101	110	111
1	2	6	0.024	0.337	0.079	**0.428**	0.697	0.986	0.976	0.136
2	6	7	0.150	0.019	0.546	0.171	0.534	0.234	**0.153**	0.173
3	6	8	0.380	0.825	0.283	**0.876**	1.000	0.663	0.838	0.139
4	5	6	0.675	**0.895**	0.584	0.959	0.135	0.143	0.943	0.074
2	4	5	0.568	0.032	0.549	0.627	**0.959**	0.272	0.033	0.519
3	5	6	0.552	**0.250**	0.718	0.313	0.542	0.350	0.017	0.675
1	2	7	0.112	0.818	**0.344**	0.905	0.018	0.006	0.307	0.995
5	6	8	0.306	0.880	0.129	**0.747**	0.690	0.021	0.902	0.098

Problème MAX-SAT

Le problème MAX-SAT est une extension du problème de satisfiabilité booléenne SAT [6] tiré de la logique propositionnelle. Pour toute formule, calculer sa satisfiabilité revient soit à mettre en lumière une combinaison de valeurs à affecter aux variables présentes dans la formule telle que la valeur de vérité de cette formule évaluée soit *vrai*, soit à prouver l'inexistence d'une telle combinaison. Le problème SAT se restreint aux formules exprimées sous forme normale conjonctive, c'est-à-dire comme une conjonction de disjonctions de littéraux, un littéral étant l'expression d'une variable propositionnelle ou de sa négation. De plus, s'il est possible de transformer n'importe quelle formule en forme normale conjonctive, toutes les transformations ne sont pas polynomiales et donc peuvent constituer une limite aux problèmes pouvant être résolus. Une instance d'un tel problème est définie par un nombre N de variables, un nombre M de clauses (disjonctions de littéraux), et par la description de ces clauses. La figure 1.2 présente un exemple d'instance.

Définition 4 *Problème SAT* : *Soit une formule propositionnelle sous forme normale conjonctive $\mathcal{F} = p_1 \wedge p_2 \wedge \cdots \wedge p_M$ où chaque proposition p_j est une disjonction de littéraux. L'ensemble des affectations possibles d'une formule composée de N variables est $\mathcal{X} := \{0,1\}^N$. L'objectif pour ce problème est alors de trouver une solution s^* qui satisfait $\mathcal{F}$ (on notera $\mathcal{F}(s^*) = 1$), et donc qui satisfait toutes les propositions p_j : $\forall j \in [\![1; M]\!], p_i(s^*) = 1$.*

La variante MAX-SAT [6] constitue le problème d'optimisation pseudo-booléen associé, où il s'agit de déterminer une affectation de valeurs aux variables qui maximise le nombre de clauses vérifiées d'une formule en forme normale conjonctive. Naturellement, la fonction MAX-SAT peut être utilisée comme fonction d'évaluation pour résoudre de manière approchée le problème

FIGURE 1.2 – Exemple d'instance SAT ($N = 8, M = 5$)

Conjonction générale	Disjonctions	Notation simplifiée
$\bigwedge$	$(x_1 \vee x_2 \vee \neg x_3 \vee \neg x_6 \vee \neg x_7)$	1 2 -3 -6 -7
	$(\neg x_2 \vee x_4 \vee x_5 \vee x_7)$	-2 4 5 7
	$(\neg x_3 \vee x_4 \vee \neg x_5 \vee x_8)$	-3 4 -5 8
	$(x_1 \vee x_2 \vee \neg x_3 \vee \neg x_5 \vee x_6 \vee \neg x_7)$	1 2 -3 -5 6 -7
	$(\neg x_4 \vee x_5 \vee \neg x_6)$	-4 5 -6

SAT [7].

Définition 5 ***Problème MAX-SAT*** **:** *Le problème MAX-SAT consiste à déterminer une affectation de valeurs aux variables d'une formule propositionnelle sous forme normale conjonctive, telle que le nombre de clauses satisfaites soit maximal. Résoudre une instance de problème à N variables et M clauses, de la forme* $\mathcal{F} = p_1 \wedge p_2 \wedge \cdots \wedge p_M$, *consiste donc à déterminer* $s^* = \text{argmax}_{s \in \{0,1\}^N} \sum_{i=1}^{M} p_i(s)$.

Même s'il s'agit d'un problème NP-difficile — en l'occurrence SAT est le problème NP-complet de référence [8] —, beaucoup d'instances de problème SAT/MAX-SAT peuvent aujourd'hui être efficacement résolues au moyen d'algorithmes exacts de complexité certes non polynomiale mais réduite [9]. Des techniques approchées telles qu'évolutionnaires seront néanmoins capables de mieux passer à l'échelle et de contourner les difficultés de résolution de certaines classes d'instances, bien qu'elles ne permettent pas dans le cas général de garantir le preuve de l'inexistence d'une solution (pour SAT) ou d'optimalité (pour MAX-SAT).

Problèmes boîte noire

L'optimisation boîte noire (*black box optimization*, BBO) [10], est une branche de l'optimisation dont la spécificité est que le problème à résoudre n'est connu qu'à travers les évaluations des solutions, c'est-à-dire que la description du problème — et *a fortiori* sa structure — n'est pas accessible à l'algorithme d'optimisation. L'optimisation boîte noire est notamment rendue nécessaire dans des contextes de simulation de fonctions complexes ou coûteuses, et plus généralement pour l'analyse des performances d'algorithmes de recherche. Par exemple, pour optimiser une fonction f polynomiale telle que $f(x) = -x^2$, un algorithme de résolution boîte noire n'ayant pas accès à la définition de la fonction ne pourra pas résoudre analytiquement le problème (ici dériver la fonction) mais devra identifier la solution optimale $x^* = 0$ au moyen d'une démarche plus expérimentale, par échantillonnage. De plus, même si la fonction est unimodale et donc dont l'optimum est simple à accéder y compris par un algorithme de résolution boîte noire, il ne

pourra vérifier que celle-ci est optimale à moins de parcourir l'intégralité de l'espace de recherche (possible seulement si l'espace de recherche est discret).

1.2 Méthodes de résolution

Cette section est dédiée aux différentes méthodes de résolution utilisées en optimisation combinatoire. Nous évoquons dans un premier temps le principe général de la résolution exacte et de ses problématiques, puis mettons l'accent sur les techniques approchées et plus spécifiquement les stratégies de type hill climbing qui font l'objet de ce travail de thèse.

1.2.1 Résolution exacte

Étant donné un problème d'optimisation combinatoire, un algorithme d'optimisation exact a pour objectif de calculer une solution optimale, ou l'ensemble des solutions optimales (voir définition 6), de n'importe quelle instance de ce problème.

Définition 6 ***Algorithme exact* :** *Soit un problème d'optimisation combinatoire composé d'un ensemble d'instances $\mathcal{I}$. Un algorithme de résolution exacte $\mathcal{A}$ appliqué à toute instance $(\mathcal{X}, f_{\text{obj}}) \in \mathcal{I}$ renvoie un ensemble $S^* \subseteq \mathcal{X}$ de solutions optimales au sens de f_{obj}, tel que $\forall s^* \in S^*, \forall s \in \mathcal{X} \setminus S^*, f_{\text{obj}}(s) < f_{\text{obj}}(s^*)$ et $\forall s^*, s^{*\prime} \in S^*, f_{\text{obj}}(s^*) = f_{\text{obj}}(s^{*\prime})$.*

L'algorithme exact le plus simple à concevoir et déclinable à tout problème, y compris boîte noire, est l'*exploration exhaustive*. Il s'agit de parcourir et d'évaluer l'intégralité des solutions de l'espace de recherche $\mathcal{X}$, et donc de renvoyer les meilleures solutions rencontrées, nécessairement optimales. Appliqués à des problèmes réellement *combinatoires*, ayant une taille d'espace de recherche non polynomiale par rapport à la taille de l'instance, de tels algorithmes sont naturellement trop lents à l'exécution dans le cas général et ne permettent ainsi pas d'assurer le calcul d'au moins une solution optimale en temps raisonnable. L'algorithme 1 présente un algorithme à parcours exhaustif qui évalue chaque solution de l'espace de recherche.

Les méthodes de résolution exactes plus efficaces limitent le nombre de solutions à évaluer. Pour ce faire, elles reposent sur des preuves qu'il n'existe pas d'optimum global dans les solutions non évaluées. Ces algorithmes généralement constructifs couvrent l'espace de recherche de manière à prouver que l'optimum ne peut être une solution qui n'a pas été évaluée intégralement. Nous pouvons citer les algorithmes de branch and bound [11], ainsi que la méthode des plans sécants avec les coupes de Gomory [12, 13], qui sont des algorithmes exacts et non exhaustifs. Le premier divise l'espace de recherche, générant de fait plusieurs branches de l'espace de recherche. Ainsi, si une branche contient des informations contradictoires ou suffisantes

Algorithme 1 Algorithme exhaustif

Entrée : un espace de recherche $\mathcal{X}$, une fonction objectif f_{obj}
Sortie : l'ensemble des solutions optimales de f_{obj}
1: $S^* \leftarrow \{\}$
2: **pour chaque** $s \in \mathcal{X}$ **faire**
3: **si** $|S^*| = 0$ **ou** $f_{\text{obj}}(s) > f_{\text{obj}}(S^*_0)$ **alors**
4: $S^* \leftarrow \{s\}$
5: **sinon si** $f_{\text{obj}}(s) = f_{\text{obj}}(S^*_0)$ **alors**
6: $S^* \leftarrow S^* \cup \{s\}$
7: **fin si**
8: **fin pour**
9: **retourne** S^*

pour savoir que l'optimum ne s'y trouve pas, l'algorithme n'évaluera pas les solutions de cette branche, limitant l'espace à parcourir et donc le temps d'exécution. Le second est utilisé pour la résolution de problèmes linéaires en nombres entiers et ajoute de nouvelles contraintes au problème donné. Ces contraintes doivent laisser les solutions optimales entières inchangées tout en étant incompatibles avec au moins un optimum global du problème initial. L'ajout de contrainte s'arrête lorsque l'optimum du nouveau problème est une solution entière. En associant ces deux approches en un seul algorithme de branch and cut [14, 15], il est possible de réduire encore l'espace des solutions à évaluer et donc le temps d'exécution. Au-delà de ces approches, la résolution de problèmes par des algorithmes exacts constitue en soit un domaine de recherche, avec un nombre important d'algorithmes variés [16], repoussant de plus en plus le nombre de solutions à évaluer pour un problème donné.

Cependant, s'il est évident que ces algorithmes peuvent résoudre des problèmes d'optimisation bien plus grands que pour les méthodes exhaustives, en admettant la conjecture P$\neq$NP les algorithmes de résolution exacte auront nécessairement une complexité de la forme $\mathcal{O}(x^{yN})$ avec $x \in]1; +\infty[$ et $y \in \mathbb{R}^*_+$, où N est la taille du problème, pour résoudre un problème NP-difficile. En d'autres termes, la résolution exacte d'un tel problème demande un nombre exponentiel d'opérations indexé sur la taille du problème N, et par conséquent il existera toujours des problèmes trop grands pour être résolus en temps raisonnable par un algorithme exact. C'est la limite du passage à l'échelle ; limite constamment repoussée grâce aux progrès algorithmiques et matériels, mais qui ne pourra pas disparaître. Enfin, c'est l'existence même de cette limite qui justifie parallèlement d'améliorer les connaissances dans le domaine des stratégies de résolution approchée.

1.2.2 Résolution approchée

Contrairement aux méthodes exactes, un algorithme de résolution approchée ne garantit pas dans le cas général de renvoyer une solution optimale d'un problème d'optimisation. De fait, il n'est plus question que d'un unique objectif consistant à trouver une solution optimale, mais d'un objectif secondaire consistant à trouver la solution s' dont l'évaluation par f_{obj} est la plus haute possible. Si le temps d'exécution est le principal point de comparaison pour les méthodes exactes, dans le cas de méthodes approchées nous nous intéressons principalement à la qualité de la solution calculée, ou plus généralement l'espérance de la qualité de la solution car la plupart des stratégies utilisées en optimisation évolutionnaire sont stochastiques. Le temps de calcul est naturellement un aspect important, qui peut être ici considéré en tant que budget d'un algorithme de recherche, ou bien comme second critère d'évaluation.

Parmi les méthodes approchées, les algorithmes de recherche échantillonnent l'espace de recherche selon une stratégie propre et les informations collectées au cours de la recherche. Étant donné un budget alloué en termes de nombre de solutions évaluées, la problématique consiste à maximiser la meilleure solution de l'échantillonnage. Contrairement à d'autres méthodes comme les heuristiques constructives, ces algorithmes de recherche ont l'avantage de fournir à tout temps de l'exécution une solution approchée, qui peut ensuite être améliorée en prolongeant la recherche. C'est ainsi qu'une comparaison efficace de stratégies de recherche peut consister à comparer l'espérance du score de la solution atteinte, étant donné un budget de recherche fixé consistant à un nombre d'appels à la fonction objectif f_{obj}.

La suite de cette section évoque les principales stratégies génériques de résolution approchée utilisées en optimisation combinatoire.

1.2.3 Métaheuristiques

Les métaheuristiques sont des algorithmes d'optimisation approchés dont le but est de trouver la meilleure solution possible pour une instance d'un problème donné. La majorité des métaheuristiques sont des processus stochastiques et itératifs : une suite d'instructions randomisées se répète jusqu'à ce que le critère d'arrêt soit atteint. Les métaheuristiques sont des méthodes génériques dans le sens où elles offrent un haut niveau d'abstraction sans faire appel à des données spécifiques à un problème, et sont donc déclinables sur un large panel de problèmes. Une stratégie de recherche générique efficace doit pouvoir retourner de bonnes solutions en moyenne lorsqu'appliquées à différentes classes de problèmes, mais naturellement et d'après le *No Free Lunch theorem* [17], des heuristiques spécifiques pourront toujours être plus efficaces sur des problèmes particuliers. Pour résoudre des problèmes d'optimisation complexes tels que ceux considérés habituellement dans le domaine de la recherche opérationnelle, il est commun

d'utiliser une ou plusieurs métaheuristiques comme schéma algorithmique, puis de sophistiquer l'algorithme en utilisant les propriétés et la structure du problème, voire en définissant des opérateurs originaux appropriés. Il existe un grand nombre de métaheuristiques [18, 19, 20], la plupart étant des métaphores inspirées par la nature [21] — avec les limites qui en découlent [22, 23] voire l'absence de réelle plus-value par rapport aux métaheuristiques originelles [24, 25, 26]. Les métaheuristiques les plus connues restent le recuit simulé [27, 28, 29], la recherche tabou [30] et les colonies de fourmis [31].

Méthodes à base de voisinage

Une grande classe de métaheuristiques définit des modèles d'algorithmes de recherche autour du concept de déplacement d'une solution courante dans l'espace de recherche, assimilant l'échantillonnage produit par un tel algorithme à la description d'une trajectoire dans cet espace. Ceci se conçoit au moyen de la notion fondamentale de *voisinage*, permettant de de structurer l'espace de recherche et de définir une mesure de distance entre solutions.

Définition 7 ***Fonction de voisinage*** **:** *Une fonction de voisinage $\mathcal{N} \to 2^{\mathcal{X}}$ associe à toute solution de $\mathcal{X}$ un ensemble de solutions dites* voisines*. Par extension, un voisinage régulier de taille n est une fonction de la forme $\mathcal{N} \to \mathcal{X}^n$ vérifiant les propriétés suivantes :*

1. $\forall x \in \mathcal{X}, x \notin \mathcal{N}(x)$
2. $\forall x \in \mathcal{X}, [y \in \mathcal{N}(x) \Rightarrow x \in \mathcal{N}(y)]$
3. *Le graphe de transition $(\mathcal{X}, \mathcal{N}[\mathcal{X}])$, avec $\mathcal{N}[\mathcal{X}] = \{(x,y) \in \mathcal{X}^2 : y \in \mathcal{N}(x)\}$, est connexe.*

Le nombre de solutions voisines renvoyées pour un problème donné est propre à chaque fonction de voisinage, et influence le comportement des méthodes de résolution. La taille du voisinage détermine en effet les possibilités d'exploration de la stratégie. Cependant, évaluer l'ensemble d'un voisinage de grande taille demande des ressources pouvant être incompatibles avec le principe même de ces heuristiques, où la réduction du temps de calcul est un facteur important.

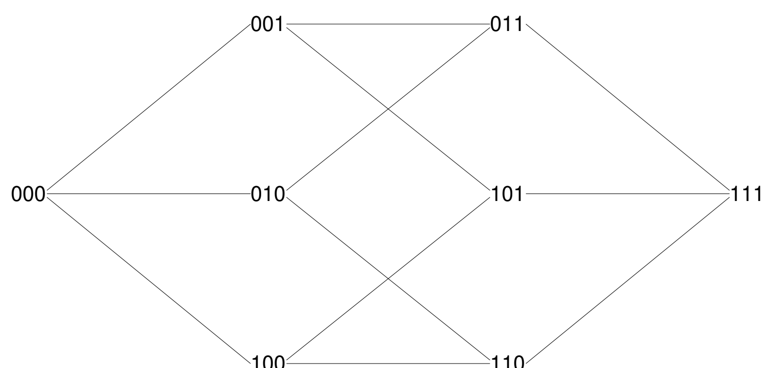
Certains espaces de recherche classiques sont bien associés à des fonctions de voisinages naturelles, comme c'est le cas du voisinage *1-Hamming distance* (communément appelé voisinage 1-flip ou bit-flip) pour les problèmes pseudo-booléens. Dans ce cas, deux solutions s_1 et s_2 de $\{0,1\}^n$ sont voisines si et seulement si leur distance de Hamming est 1 (voir définition 8). Il existe en revanche plusieurs voisinages naturels que l'on peut associer aux problèmes à base de permutations (en particulier insertion, échange ou inversion) et ceux-ci seront plus ou moins adaptés selon la classe du problème, c'est-à-dire le modèle de fonction objectif et les propriétés

inhérentes à celle-ci.

Définition 8 ***Voisinage 1-flip* :** *Soit n une taille de problème pseudo-booléen, et $\mathcal{X} = \{0,1\}^n$ l'espace de recherche associé. La fonction $\mathcal{N} : \mathcal{X} \to 2^{\mathcal{X}}$ décrivant le voisinage 1-flip est tel que $\forall s \in \mathcal{X}, \mathcal{N}(s) = \{s' \in \mathcal{X} : \exists i \in [\![1;n]\!], (s_i \neq s'_i) \text{ et } (\forall j \neq i, s_j = s'_j)\}$. La taille du voisinage est n : $\forall s \in \mathcal{X}, |\mathcal{N}(s)| = n$.*

Il s'agit du voisinage que nous utiliserons pour structurer les problèmes pseudo-booléens étudiés (NK et MAX-SAT). Les méthodes à base de voisinage appliqués sur ces problèmes seront définies à partir du voisinage 1-flip. La figure 1.3 présente un exemple de voisinage sur l'espace de recherche $\mathcal{X} = \{0,1\}^N$ composé des chaînes de bits de longueur N (avec ici $N = 3$). La fonction de voisinage utilisée associe à chaque solution de $\mathcal{X}$ les N solutions situées à une distance de Hamming de 1.

FIGURE 1.3 – Exemple de relation de voisinage pour l'espace de recherche $\{0,1\}^3$



Un algorithme qui se déplace de solution en solution dans l'espace de recherche au moyen d'une relation de voisinage et d'une stratégie d'exploration est une recherche locale. Les recherches locales décrivent ainsi des trajectoires de recherche dans le but d'atteindre la meilleure solution possible.

Recherche locale

L'approche derrière les algorithmes de recherche locale consiste à atteindre des solutions de bonne qualité en améliorant itérativement la solution courante au moyen de déplacements locaux dans l'espace de recherche. Les recherches locales sont des algorithmes d'optimisation approchés dont les composants principaux peuvent être définis et réduits à un quadruplet $(\mathcal{X}, \mathcal{N}, f, \mathcal{H})$, à

savoir :

— Un espace de recherche $\mathcal{X}$ contenant toutes les solutions valides du problème proposé.

— Une fonction de voisinage $\mathcal{N} : \mathcal{X} \rightarrow 2^{\mathcal{X}}$ décrivant les mouvements possibles entre solutions.

— Une fonction d'évaluation des solutions $f : \mathcal{X} \rightarrow \mathbb{R}$ qui associe à chaque solution un score. f peut être la fonction objectif du problème d'optimisation à résoudre, ou une variation de celle-ci qui rend compte de la qualité des solutions et de la pertinence de leurs propriétés pour l'algorithme de recherche.

— Une heuristique de mouvement $\mathcal{H}$, que l'on assimilera à un opérateur de sélection de solution voisine. Cette heuristique peut être stochastique et applique une stratégie basée sur l'évaluation des voisins et d'éventuels paramètres additionnels propres aux algorithmes de recherche, comme la prise en compte de l'historique des précédentes solutions évaluées.

Ces composants interagissent de manière à atteindre au cours de cet échantillonnage la meilleure solution possible. Tout d'abord, une solution de départ est choisie, usuellement de manière aléatoire ou alternativement au moyen d'une heuristique constructive, pour constituer la première solution courante de la recherche. Ensuite, et fonction du mécanisme induit par $\mathcal{H}$, l'algorithme évalue tout ou partie du voisinage de la solution courante puis sélectionne une de ces solutions qui devient la solution courante de la prochaine itération de la recherche. Lorsque qu'un critère de fin est atteint, la recherche locale retourne la meilleure solution évaluée au cours de la recherche. Lorsque la fonction d'évaluation diffère de la fonction objectif, l'algorithme peut inclure des calculs de score supplémentaires au sens de la fonction objectif (par exemple restreints aux solutions courante) et retourne au final la meilleure d'entre elles. L'algorithme 2 montre la structure générale d'une recherche locale.

Algorithme 2 Structure générale d'une recherche locale

Entrée : un problème d'optimisation $(\mathcal{X}, f_{\text{obj}})$, une fonction d'évaluation f, une fonction de voisinage $\mathcal{N}$, une heuristique de mouvement $\mathcal{H}$, un critère d'arrêt c

Sortie : une solution s optimisée via la recherche locale $(\mathcal{X}, \mathcal{N}, f, \mathcal{H})$

1: $s \leftarrow$solution_aléatoire()
2: $s^* \leftarrow s$
3: **tant que** $\neg c$ **faire**
4: $\quad s \leftarrow \mathcal{H}_f(\mathcal{N}(s))$
5: $\quad$ **si** $f_{\text{obj}}(s) > f_{\text{obj}}(s^*)$ **alors**
6: $\quad\quad s^* \leftarrow s$
7: $\quad$ **fin si**
8: **fin tant que**
9: **retourne** s^*

De cette structure de base, différents algorithmes de recherche locale peuvent être construits. Si chaque élément du quadruplet $(\mathcal{X}, \mathcal{N}, f, \mathcal{H})$ peut comporter des spécificités propres pour

définir une recherche locale unique, c'est usuellement l'heuristique de mouvement qui concentre le plus d'attention. Dans la suite de cette section, nous présentons diverses heuristiques de recherches locales parmi les plus classiques.

Hill climber

Les recherches locales dites de *hill climbing* (appelées alors *hill climbers* ou simplement *climbers*) sont des algorithmes qui s'apparentent aux descentes de gradient adaptées à un contexte d'optimisation combinatoire. Ces algorithmes sont des méthodes de forte exploitation, dans le sens où l'heuristique de mouvement sélectionnera systématiquement un voisin de meilleur score. En conséquence, aucune phase de la recherche n'est consacrée à l'exploration de l'espace de recherche via des mécanismes perturbatifs, comme c'est le cas de recherches plus sophistiquées. Les climbers se distinguent ensuite par la règle pivot [32] qu'elles utilisent pour choisir entre plusieurs solutions voisines améliorant la solution courante. Un climber dit *strict* ne pourra améliorer la solution courante que si celle-ci possède au moins un voisin strictement améliorant. L'effet est que ce type de recherche locale bloquera inévitablement sur une solution qui ne possède aucun voisin améliorant, appelée *optimum local*.

Définition 9 ***Optimum local*** **:** *Soit un hill climber* $(\mathcal{X}, \mathcal{N}, f, \mathcal{H})$*. Un optimum local au sens de* f *est une solution* $s \in \mathcal{X}$ *telle que* $\forall s' \in \mathcal{N}(s), f(s) \geq f(s')$*.*

Définition 10 ***Voisinage améliorant*** **:** *Étant donné une solution* $s \in \mathcal{X}$*, une fonction de voisinage* $\mathcal{N}$ *et une fonction d'évaluation* f*, le voisinage améliorant de* s *est l'ensemble* $N^*(s) = \{s' \in \mathcal{N}(s) : f(s') > f(s)\}$*. Si* $N^*(s) = \emptyset$*, alors* s *est un optimum local.*

Règles pivots Une règle pivot d'un hill climber est une heuristique de choix de voisin restreint aux solutions de N^*. Les règles les plus communément utilisées sont celles du *meilleur améliorant* et du *premier améliorant*. Le meilleur améliorant sélectionne le meilleur voisin de la solution courante, donc résout à chaque itération le sous-problème $\text{argmax}_{s' \in N^*(s)} f_{\text{obj}}(s')$ au moyen généralement d'un algorithme d'exploration exhaustive (appliqué cette fois à un ensemble de taille polynomiale en la taille du problème, en supposant que le voisinage l'est). Ainsi, à chaque itération, l'heuristique fait le meilleur choix local à partir de l'information disponible. Au contraire, le premier améliorant sélectionne le premier voisin évalué dont le score améliore la solution courante. Cette règle pivot se comporte de la même façon qu'une heuristique qui sélectionnerait aléatoirement une solution du voisinage améliorant. De par la dimension stochastique de cette sélection il n'est pas nécessaire d'évaluer systématiquement tout le voisinage des solutions courantes, et donc d'économiser du budget d'appels à la fonction objectif ou de temps

d'exécution. L'algorithme 3 présente une instanciation de l'algorithme général d'une recherche locale (algorithme 2) en un hill climber de type premier améliorant. La boucle *tant que* de l'algorithme, de la ligne 3 à 7, correspond à la règle pivot du hill climber et donc à l'heuristique de choix $\mathcal{H}$ de la recherche locale. Le critère d'arrêt est représenté par la condition $N^*(s) = \emptyset$; la recherche s'arrête donc une fois un optimum local atteint. Cet algorithme décrit la logique de navigation dans l'espace de recherche structuré par le voisinage, et non son fonctionnement opérationnel qui peut être optimisé de manière à réduire le nombre d'opérations effectuées.

Algorithme 3 Hill climber premier améliorant

Entrée : une fonction d'évaluation f, une relation de voisinage $\mathcal{N}$
Sortie : un optimum local s de f
1: $s \leftarrow$solution_aléatoire()
2: **tant que** $N^* \neq \emptyset$ **faire**
3: **pour chaque** $s' \in \mathcal{N}(s)$ **faire**
4: **si** $f_{\text{obj}}(s) < f_{\text{obj}}(s')$ **alors**
5: $s \leftarrow s'$
6: **break**
7: **fin si**
8: **fin pour**
9: **fin tant que**
10: **retourne** s

Hill climber, performances et caractéristiques Chaque règle pivot a un impact important sur la trajectoire de la recherche locale. Ces différences ont été étudiées notamment d'un point de vue des solutions retournées, et donc des performances induites par les règles pivots [33, 34, 35], mais aussi selon la manière dont les différents hill climbers se comportent en explorant l'espace de recherche [36].

Ces hill climbers diffèrent dans le nombre d'itérations nécessaires pour atteindre un optimum local. En effet, l'heuristique du meilleur améliorant converge en moyenne en moins de pas vers un optimum local que l'heuristique du premier améliorant de par les gains itérés plus importants [33] (trajectoires plus courtes mais plus ascensionnelles) ; cependant elle nécessite généralement d'évaluer davantage de solutions puisque les voisinages des solutions courantes sont explorés exhaustivement. La qualité de l'optimum atteint en fonction de la stratégie choisie dépend en partie de la structure du problème [33, 34, 35]. Certaines instances profitent d'optimisations plus longues, et donc plus progressives. C'est ainsi que la règle pivot du moins bon améliorant [37] — consistant à sélectionner plutôt que la solution $\text{argmax}_{s' \in N^*(s)} f_{\text{obj}}(s')$ de la règle du meilleur améliorant, la solution $\text{argmin}_{s' \in N^*(s)} f_{\text{obj}}(s')$ — qui adopte une ascension plus progressive que par premier améliorant, permet sur certains types d'instances plus difficiles d'atteindre de meilleures solutions tout en se restreignant à ne sélectionner que des voisins stric-

tement améliorants. Des variantes de cette règle pivot se sont révélées plus efficaces en termes de performances et de nombre d'appels à la fonction objectif [38] sur certaines instances, comparée aux deux règles pivots plus conventionnelles (premier et meilleur améliorant).

Recherche locale itérée

Une recherche locale itérée (ILS, *Iterated Local Search* [39]) alterne des phases d'intensification de la recherche — au moyen de recherches de type hill climbing présentées ci-avant — avec des phases perturbatives. Il s'agit d'un schéma général et répété de stratégie de recherche permettant de combiner et paramétrer efficacement exploration et exploitation. De nombreux algorithmes d'optimisation, y compris développés au cours des années 1980, suivent un tel principe pouvant se décliner sous de nombreuses formes selon le type d'approche et le problème à résoudre [40, 41, 42, 43, 44, 45].

Les ILS fonctionnent par cycles de deux étapes. Tout d'abord, l'on optimise la solution courante via la recherche choisie. Une fois le critère d'arrêt atteint par cette dernière, un nombre de mouvements n aléatoires dans l'espace de recherche à l'aide du voisinage $\mathcal{N}$ à partir de la dernière solution retournée sont effectués. n doit être suffisamment grand pour ne pas se bloquer dans des régions sous-optimales de l'espace de recherche [46], mais assez petit pour bénéficier des propriétés intéressantes de la solution courante. Ce paramètre est généralement déterminé empiriquement en suivant l'objectif du paramétrage le plus efficace. Le nombre de cycles à effectuer dépend davantage des contraintes de budget qui s'appliquent à la résolution. L'algorithme 4 présente une ILS basé sur un hill climber.

Les recherches locales itérées ont pour but de naviguer entre les optima locaux. La perturbation doit alors être assez importante pour s'éloigner de l'optimum courant pour en visiter de nouveaux, tout en restant modérée afin de ne pas s'éloigner des régions favorables de l'espace de recherche.

Recherche locale guidée

Les recherches locales guidées [47] sont la version étendue de l'architecture GENET [48] qui se concentre sur les problèmes de satisfaction de contraintes. Une recherche locale guidée fonctionne de la même manière qu'une ILS dans le sens où une recherche locale est répétée au sein d'une metaheuristique. Ce qui différencie ces deux méthodes est le processus qui s'opère entre deux recherches locales. Les ILS détériorent les solutions avec un certain nombre de mouvements aléatoires, tandis que la recherche locale guidée choisit aléatoirement une nouvelle solution de départ et change la fonction d'évaluation que la prochaine recherche locale utilisera.

Algorithme 4 Recherche locale itérée

Entrée : une fonction d'évaluation f, une relation de voisinage $\mathcal{N}$, un critère d'arrêt c, un nombre de perturbations n

Sortie : un optimum local s de f

1: $s \leftarrow$solution_aléatoire()
2: $s^* \leftarrow s$
3: **tant que** $\neg c$ **faire**
4: $s \leftarrow$ hill_climber($f, \mathcal{N}, s$)
5: **si** $f_{\text{obj}}(s^*) < f_{\text{obj}}(s)$ **alors**
6: $s^* \leftarrow s$
7: **fin si**
8: **pour** $i = 1 \rightarrow n$ **faire**
9: $s \leftarrow$solution_aléatoire($\mathcal{N}(s)$)
10: **fin pour**
11: **fin tant que**
12: **retourne** s

La fonction d'évaluation employée en début de recherche est la fonction objectif du problème. Ensuite, au cours de la recherche, la fonction d'évaluation module l'évaluation de la fonction objectif via l'utilisation de pénalités qui s'appliquent sur la structure de la solution évaluée. Par exemple, pour le problème du voyageur de commerce, une pénalité peut s'appliquer sur l'enchaînement direct entre deux villes. Ainsi, les pénalités préviennent la recherche de se bloquer dans des régions sous-optimales de l'espace de recherche via la modulation de l'évaluation de certaines caractéristiques de la solution. Du point de vue de la structure de la recherche, les pénalités doivent servir à supprimer les pics non optimaux de la fonction objectif, à savoir les optima locaux. En réduisant suffisamment le nombre d'optima locaux, et sous l'hypothèse que l'optimum global reste un optimum tout au long de la recherche, alors trouver ce dernier devient de plus en plus simple à mesure que les pénalités s'ajoutent.

Il est néanmoins à noter que pour constituer un système de pénalités efficace, avoir des pénalités les plus spécifiques possibles au problème à résoudre est nécessaire. Si cela rend cette métaheuristique plus efficace dans sa recherche, cela la rend aussi plus difficile à mettre en place à cause de cet aspect supplémentaire à développer (et à choisir) spécifiquement pour la recherche locale guidée.

Recherche à voisinage variable

La recherche à voisinage variable [49, 50] suit le fonctionnement classique d'une simple recherche locale de type hill climber, à la différence qu'à intervalle régulier, ou lorsqu'un optimum local est rencontré, le voisinage utilisé change. Bien qu'il s'agisse d'un schéma général d'algo-

rithme et donc que des sophistications sont toujours possibles, l'usage est d'alterner des voisinages selon un schéma prédéfini. Ces différents voisinages peuvent être disjoints ou au contraire inclus les uns dans les autres ; dans ce second cas de figure la recherche débute avec le voisinage le plus petit et s'élargit lorsqu'un optimum local au sens du voisinage courant est atteint. Dans le cas de voisinage disjoints ou partageant peu de solutions en commun, une telle recherche permet de combiner plusieurs voisinages et d'avoir virtuellement un plus grand voisinage que le plus grand voisinage utilisé effectivement au cours de la recherche. Un plus grand voisinage implique plus de liens entre les solutions ce qui rend plus difficile l'apparition de solutions sans voisin améliorant, donc d'optima locaux. Puisque la difficulté d'une instance est liée au nombre d'optima locaux dans la fonction objectif, réduire ce nombre revient à faciliter l'optimisation, et à augmenter la qualité des solutions renvoyées. La recherche à voisinage variable bénéficie de ces avantages tout en limitant les désavantages d'avoir un grand voisinage, c'est-à-dire le temps d'exécution accru pour les algorithmes qui utilisent de grands voisinages. En effet, chaque voisinage est considéré seul, le nombre de voisins maximum à générer pour une itération reste donc faible. C'est pourquoi usuellement la descente à voisinage variable utilise comme heuristique de choix de voisin un climber de type meilleur améliorant.

Algorithmes génétiques

Les algorithmes génétiques diffèrent des metaheuristiques présentées jusqu'alors dans le fait qu'ils font appel à une population de solutions plutôt qu'à une seule solution courante pour générer les futures potentielles solutions courantes. Ces algorithmes ont d'abord été proposés par Holland [51] dans un but d'analyse des processus adaptatifs. Cependant, ils sont depuis utilisés comme technique d'optimisation. Utiliser une population de solutions plutôt qu'une solution unique nécessite de définir des outils capables de travailler sur une population. Ainsi, en ajout de la sélection de voisins pour composer la population de l'itération suivante, il est nécessaire de définir un mode de sélection des solutions courantes à partir desquelles générer les solutions voisines. Il est à noter que généralement, dans un contexte d'algorithmes génétiques, ce n'est pas une fonction de voisinage qui est utilisée, mais une fonction de mutation stochastique ; il est donc à propos de parler de mutants plutôt que de voisins. Enfin travailler sur une population rend possible l'utilisation d'un nouvel opérateur : la recombinaison. Cette fonction génère une solution à partir de plusieurs autres de la population. Par exemple, dans le cas d'un espace de recherche binaire $\mathcal{X} = \{0,1\}^n$, l'opérateur de recombinaison (ou *croisement*) de deux solutions $s_1, s_2 \in \mathcal{X}$ pourra consister à générer une solution dont chaque valeur de bit provient de l'une des deux solutions parentes, c'est-à-dire $s' \in \mathcal{X}$ telle que $\forall i \in [\![1; N]\!], s'_i \in \{s_{1i}, x_{2i}\}$.

Un algorithme génétique peut être défini par un sextuplet $(\mathcal{X}, f, \mu, \rho, \alpha, \omega)$, avec :

— $\mathcal{X}$ l'espace de recherche du problème à résoudre.

— $f : \mathcal{X} \rightarrow \mathbb{R}$ une fonction d'évaluation

— $\mu : \mathcal{X} \rightarrow \mathcal{X}$ un opérateur de mutation (stochastique)

— $\rho : \mathcal{X}^n \rightarrow \mathcal{X}$ un opérateur de recombinaison n-aire (stochastique)

— α une fonction de sélection de la population courante, pour générer la population de l'itération suivante via μ et ρ.

— ω : Une fonction de sélection des solutions générées, afin de composer la nouvelle population dans le cas où le nombre d'individus générés est plus grand que la taille de la population.

Les fonctions α et ω remplacent l'heuristique de choix $\mathcal{H}$ présente dans les algorithmes de recherche locale. Généralement seule la fonction α est utilisée car il est d'usage de générer autant de solutions que de solutions dans la population courante, rendant une sélection ω inutile. De fait, la définition d'un algorithme génétique devient plus simple et la pression de sélection repose donc intégralement sur la sélection α.

L'algorithme 5 présente la structure globale d'un algorithme génétique. On commence avec une population d'individus aléatoires. À chaque itération, une nouvelle population de taille P est générée à partir d'un sous-ensemble d'individus $\alpha(S)$ avec les fonctions μ et ρ. Le processus de génération d'une nouvelle population n'est pas explicité ici car la façon de générer cette population est libre.

Algorithme 5 Algorithme génétique

Entrée : une fonction d'évaluation f, une fonction de mutation μ, un opérateur de recombinaison ρ, une fonction de sélection α, une taille de population P
Sortie : une solution s de f

1: $S \leftarrow \{\text{solution_aléatoire}() | i \in [\![1; P]\!]\}$
2: $s^* \leftarrow \text{meilleur}_f(S)$
3: **tant que** $\neg c$ **faire**
4: $\quad S \leftarrow \text{nouvelle_population}(\alpha(S), \mu, \rho, P)$
5: $\quad s^* \leftarrow \text{meilleur}_f(S \cup \{s^*\})$
6: **fin tant que**
7: **retourne** s^*

L'introduction des fonctions de recombinaison grâce à la population de solutions permet d'exploiter bien plus les structures présentes dans la fonction d'évaluation f qu'une fonction de voisinage ou de mutation. En effet, les valeurs de variables en interaction d'une solution peuvent être transmises simultanément dans la solution recombinée, à plus forte raison lorsque l'on utilise des fonctions de recombinaison spécifiques basés sur les propriétés et la connaissance des problèmes étudiés [52]. Une difficulté dans le paramétrage des algorithmes génétiques réside

dans la gestion de la diversité des individus au sein de la population (pour garantir l'efficacité exploratoire de l'opérateur de recombinaison) et sa convergence.

Stratégies d'évolution

Les stratégies d'évolution sont un cadre de résolution de problèmes issu de l'optimisation continue. Ces stratégies sont similaires aux algorithmes génétiques dans le sens où ce sont des méthodes de résolution approchée, évolutionnaires et, dans le cas général, à population. Dans ces deux approches, il s'agit de faire évoluer une population d'individus au moyen d'opérateurs de mutation, de recombinaison et de sélection.

Ce cadre à d'abord été développé pour les problèmes d'optimisation continus à travers les publications de Holland [51], Bremermann [53], Rechenberg [54] et Schwefel [55]. Ces travaux ont débouché sur des méthodes de résolution robustes comme le schéma général ($\mu_{/\rho} \overset{+}{,} \lambda$)-ES, la stratégie CMA-ES [56] ou la règle du 1/5 [54], stipulant qu'une mutation sur cinq en moyenne doit être améliorante pour exploiter de la manière la plus efficace la fonction objectif.

> *"Le ratio des mutations améliorantes sur toutes les mutations devrait être $\frac{1}{5}$. S'il est plus grand que $\frac{1}{5}$, la variance de la mutation doit être augmenté ; au contraire s'il est plus petit, la variance doit être diminuée."* [54, 57]

Des travaux pionniers sur les stratégies d'évolution jusqu'aux développements plus récents en optimisation boîte noire, une partie des connaissances comme cette règle du 1/5 se base sur des fondements et propriétés (comme la continuité de la fonction objectif) qui ne correspondent pas nécessairement avec les problématiques de l'optimisation combinatoire [57]. Dans la suite du manuscrit, les descriptions et considérations relatives aux stratégies d'évolution s'entendent justement dans un contexte discret.

Une stratégie d'évolution se définit par une expression de la forme ($\mu_{/\rho} \sigma \lambda$)-ES. Le suffixe "-ES" indique que l'on parle d'une stratégie d'évolution (Evolution Strategy), μ est le nombre de solutions courantes, λ le nombre de nouvelles solutions générées à chaque itération, en tout ou partie depuis les solutions courantes, ρ le nombre de solutions parentes utilisées pour la recombinaison (paramètre absent s'il n'y a pas de recombinaisons) et σ l'opérateur définissant la population de solution dans laquelle les solutions courantes de l'itération suivante seront choisies. L'opérateur σ est choisi parmi "," et "+". La stratégie dite *virgule* indique que les solutions sélectionnées pour la prochaine itération devront faire partie des λ solutions générées. Au contraire, la stratégie *plus* définit la population de solutions pouvant être sélectionnées comme l'union des ensembles des μ solutions courantes et des λ solutions générées.

Le tableau 1.2 présente notamment les les analogies existant entre les stratégies d'évolution à solution unique ($\mu = 1$) et les recherches locales. La différence éventuelle concerne l'opérateur de mutation qui peut affecter la solution courante selon diverses lois de probabilités, tandis qu'une fonction de voisinage employée dans un algorithme de recherche locale propose à l'heuristique de choix un nombre restreint de solutions candidates.

TABLE 1.2 – Catégories notables de stratégies d'évolution, et leurs applications

Stratégies	Applications
(1,λ)-ES	Cette stratégie d'évolution à solution unique se déplace plus fréquemment dans l'espace de recherche, à raison d'un pas toutes les λ évaluations. En considérant un opérateur de voisinage comme un opérateur de mutation, cette stratégie associe certaines recherches locales : marche aléatoire ($\lambda = 1$), marche par échantillonnage [58] recherche tabou [30].
(1+λ)-ES	La composante exploratoire de cette stratégie à sélection élitiste est intégralement déterminée par l'opérateur de mutation. Dans le cas où celle-ci s'opère au moyen d'une fonction de voisinage, il s'agit d'une marche adaptative assimilée à un hill climbing : premier améliorant si $\lambda = 1$, meilleur améliorant si $\lambda = n$ (avec n la taille du voisinage)
($\mu_{[/\rho]}\sigma\lambda$)-ES	Les méthodes d'optimisation à population sont représentées par cette expression. Les algorithmes à population est une catégorie large incluant notamment les algorithmes génétiques (stationnaires si $\lambda = 1$) et la programmation génétique.

Programmation génétique

La programmation génétique [59] est largement inspirée des algorithmes génétiques dans son fonctionnement et dans les procédés utilisés au sein de la recherche. Au-delà de ces similitudes que l'on détaillera plus tard, les algorithmes génétiques sont usuellement utilisés comme des algorithmes d'optimisation, c'est-à-dire qu'un individu de la recherche est une solution du problème à résoudre. Au contraire, la programmation génétique est utilisée pour générer des fonctions ou des algorithmes, et donc sera classé comme un algorithme d'apprentissage plutôt que d'optimisation.

Tout comme les algorithmes génétiques, la programmation génétique utilise des fonctions pour travailler sur les individus de sa population afin d'en générer une nouvelle pour l'itération suivante. Une fonction de mutation et une autre de recombinaison permettent la génération de nouveaux individus, et une fonction de sélection dans la population courante, ainsi qu'une

autre pour sélectionner les individus nouvellement générés qui feront partie de la population à l'itération suivante sont nécessaires pour l'établissement d'une pression évolutive de sélection.

Ensuite, la programmation génétique ne travaille pas directement sur les solutions de l'espace de recherche, plutôt elle cherche à générer soit des fonctions qui doivent aider à l'optimisation de ces solutions, soit des algorithmes complets d'optimisation. Pour ce faire, une représentation des individus par des arbres est utilisée. Les nœuds sont des opérateurs et les feuilles des constantes ou des variables. Les opérateurs et valeurs utilisés seront différents en fonction du but à atteindre. Ainsi, si l'on veut générer une fonction polynomiale, les opérateurs +,-,*,/,ˆ seront de bons candidats à la création de telles fonctions, alors que si l'objectif est de créer des algorithmes, des blocs de conditions, et de boucles seront utiles. Le choix des opérateurs se fait donc en fonction du problème et de l'objectif que l'on se fixe. Les valeurs, elles, sont conditionnées aux domaines de définition des opérateurs utilisés. Ainsi, si l'on veut générer des fonctions numériques les constantes et variables le seront numériques, tandis que si l'on cherche à générer un algorithme de plus haut niveau, les feuilles des arbres des individus seront des instructions.

Les opérateurs de mutation et de recombinaison auront un effet direct sur l'arbre d'un individu. Une mutation pourra changer un opérateur par un autre, tronquer une partie de l'arbre, changer la valeur d'une constante, *etc.* La recombinaison pourra quant à elle sélectionner une partie d'un parent et l'ajouter aléatoirement dans l'arbre de l'autre parent pour créer un nouvel individu. La spécificité des opérateurs utilisés au problème et à l'objectif à atteindre est certainement une limite de cette approche en plus du temps d'exécution long lié à la représentation en arbre des individus. Cependant, cela n'impute en rien son efficacité notamment sur des problématiques d'adaptation à une courbe [60].

1.3 Apprentissage pour l'optimisation

En intelligence computationnelle, de nombreuses approches combinent désormais des techniques avancées d'optimisation évolutionnaire et d'apprentissage automatique, que ce soit pour rendre les algorithmes de recherche plus autonomes ou auto-adaptatifs, pour déterminer la meilleure stratégie à appliquer à tel ou tel type de problème, ou encore pour aider au paramétrage d'algorithmes. Il s'agit d'un champ d'étude très vaste, dont nous listons ci-après quelques concepts reliés aux contributions de cette thèse.

1.3.1 Modèles surrogate

Les modèles surrogate (ou de substitution) construisent une fonction imitant le comportement de la fonction objectif de manière à la remplacer progressivement. Un tel objectif peut être

rapproché d'outils de l'analyse mathématique comme les séries de Fourier [61]. Ces dernières, limitées aux fonctions périodiques, sont des sommes infinies de fonctions trigonométriques ; ainsi, la fonction de remplacement devient de plus en plus précise avec le nombre d'harmoniques qui composent la fonction.

Les modèles surrogate sont construits soit à base de sommes infinies de sous-fonctions [62], soit via un échantillonnage de la fonction objectif [63] qui sert de base à la génération du modèle. Un axe de recherche sur les surrogate a été d'utiliser cet échantillonnage, puis les recommandations du modèle, comme guide pour de l'optimisation [64, 65]. Dans ce contexte, le modèle est initialisé avec un faible nombre de points, puis on ajoute le point recommandé par ce modèle à l'échantillon, et enfin on construit un nouveau modèle à partir de cet échantillonnage augmenté. Si ces méthodes arrivent à des résultats intéressants, c'est la capacité des surrogate à remplacer des fonctions plus complexes qui est la plus utile pour les domaines dont les fonctions à évaluer sont coûteuses, notamment en aérospatial [66, 65] ou en chimie quantique [67] dont les fonctions objectif consistent en de longues simulations. Ainsi, pour ces usages, les modèles surrogate sont particulièrement adaptés, dans le sens où ils répondent exactement aux besoins du domaine. Il reste à noter que les surrogates sont des modèles boîte noire, et donc ne requièrent aucune information spécifique sur la fonction objectif.

Néanmoins, certaines caractéristiques des surrogates peuvent être soit inutiles, soit contre-productives dans des applications spécifiques. Ainsi, dans un contexte d'optimisation pure, où seule la solution renvoyée n'a d'importance, alors les valeurs associées à chaque solution de l'espace de recherche par le modèle ne sont pas nécessairement pertinentes. Donc l'effort d'imiter la fonction objectif sur ce point n'est pas nécessaire. De plus, cet objectif de suivre au maximum la fonction initiale a pour conséquence de reproduire les structures de celle-ci, et ce même si ces structures imposent une détérioration de la recherche, principalement causée par l'interaction entre variables.

1.3.2 Apprentissage de fonction d'évaluation

L'apprentissage de fonctions a été grandement étudié, ne serait-ce qu'à travers la génération de modèles de substitution, ou dans des systèmes particuliers comme les systèmes embarqués avec les travaux de Kaelbling [68]. Plus précisément cette idée a aussi été proposée spécifiquement pour les algorithmes de recherche locale par Boyan et Moore [69, 70]. La façon dont fonctionne ce dernier algorithme, pour apprendre une fonction objectif, est de faire évoluer une fonction pour qu'elle prédise le résultat d'une recherche locale sur la fonction objectif. Pour ce faire, une fonction est d'abord construite aléatoirement. Ensuite, une recherche locale est effectuée avec cette fonction comme fonction d'évaluation, et avec une solution de départ fixe. La solution retournée par cette recherche locale est ensuite utilisée comme solution de départ pour

une seconde recherche locale, cette fois en utilisant la fonction objectif. La seconde recherche renseigne alors l'algorithme sur la qualité de la fonction générée et peut la modifier en conséquence. Enfin, lorsque la seconde recherche ne modifie plus la solution retournée par la première recherche, c'est-à-dire lorsque l'optimum local atteint dans le paysage de la fonction générée est aussi un optimum de la fonction objectif, la solution de départ est modifiée pour continuer l'apprentissage de la fonction jusqu'à épuisement du budget. Enfin la meilleure combinaison solution de départ/fonction d'évaluation est retournée. Ce couple peut ensuite être utilisé dans un algorithme de recherche locale pour retrouver la meilleure solution rencontrée au cours de la génération de la fonction.

Cette méthode de génération se montre efficace pour trouver de bonnes solutions. Cependant, si la fonction retournée présente des caractéristiques communes avec la fonction objectif (a minima un second optimum local), aucune garantie ne peut être énoncée quant à sa qualité générale. Ainsi, une fonction générée par cet algorithme doit être utilisée avec sa solution de départ associée.

Pour surmonter cette limitation, il faut être capable d'évaluer des algorithmes stochastiques et de les comparer. En ce sens les travaux de Whitley sur l'évaluation des algorithmes évolutionnaires [71] peuvent être utiles à la génération d'algorithmes, à travers leur évaluation. L'étude sur les tournois comme méthode d'évaluation et de sélection [72] montre la possibilité et l'efficacité d'utiliser un tournoi dans la sélection d'algorithmes stochastique, ce qui peut être intéressant à intégrer dans un processus de génération d'algorithmes.

1.3.3 Configuration automatique de paramètres

Dans un contexte de problème de sélection d'algorithme [73, 74], de configuration et de réglage automatique des paramètres [75, 76, 77], ou de recherche autonome [78], l'objectif est d'utiliser un algorithme B pour chercher, sélectionner, construire, ou paramétrer un algorithme cible $A \in \mathcal{A}$ afin que ce dernier atteigne une performance maximale d'après un critère p sur un ensemble d'instances $\mathcal{J}$ de manière à maximiser $p(A, \mathcal{J})$. L'algorithme B peut être considéré comme appartenant à l'ensemble des hyperheuristiques [79], alors que l'algorithme A est un algorithme sélectionné depuis un sous-ensemble $\mathcal{A}' \subseteq \mathcal{A}$ d'algorithmes d'optimisation comme des heuristiques ou des métaheuristiques instanciées potentiellement efficaces sur les instances $\mathcal{J}$ concernées.

En général l'approche par portfolios [73, 74], pour un ensemble d'algorithmes $\mathcal{A}'$ donné, a pour but de trouver l'algorithme A^* tel que $\text{argmax}_{A^* \in \mathcal{A}'} p(A, \mathcal{J}')$, $\mathcal{J}' \subset \mathcal{J}$. Les algorithmes de l'ensemble $\mathcal{A}'$ peuvent différer par leur composants ou par leur valeurs de paramètres. Pour la configuration automatique par instance [80], pour un ensemble d'instances $\mathcal{J}' = \bigcup_{i=1}^{n} \mathcal{I}_i$, l'ob-

jectif est alors de trouver une séquence d'algorithmes $(A_i)_{1\leq i\leq n}$ qui maximise $\sum_{i=1}^{n} p(A_i, \{I_i\})$.

Une autre approche est de paramétrer automatiquement un algorithme donné pour un ensemble d'instances. Le package irace [77] effectue exactement cette tâche. Étant donné un algorithme à paramétrer, un intervalle de valeurs possibles pour chaque paramètre encore non fixé, et un ensemble d'instances que l'algorithme doit résoudre, irace propose une combinaison de paramètres pour l'ensemble d'instances avec un budget restreint. Irace utilise un algorithme d'optimisation pour trouver les meilleurs paramètres. Grossièrement, un paramétrage est une solution, et l'évaluation de celle-ci se fait en estimant l'espérance de l'algorithme associé pour un budget donné en l'exécutant plusieurs fois. Et dans le détail, il s'accommode d'hypothèses simplifiant sa recherche et économisant son budget alloué. Tout d'abord, afin de ne pas utiliser du budget inutilement sur des paramétrages de mauvaise qualité, notamment au début de la recherche, le budget pour chaque exécution de l'algorithme solution est faible. L'hypothèse associée à cette économie de budget est qu'à budget croissant la qualité des solutions retournées augmente. Cette hypothèse est raisonnable dans le sens où elle est vérifiée dans la très grande majorité des cas — par exemple, pour tous les algorithmes présentés jusqu'ici cette hypothèse est vraie. Ensuite, une autre hypothèse utilisée est que des paramétrages similaires auront des résultats proches.

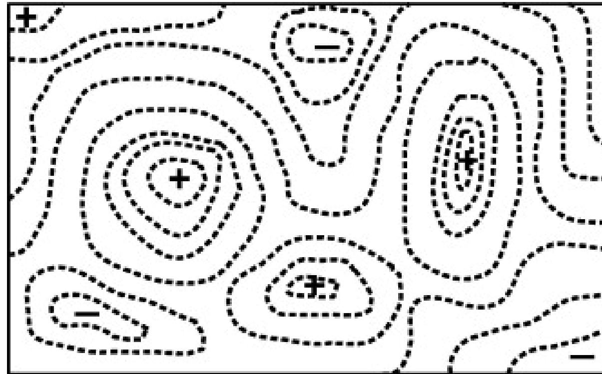
1.4 Paysage de fitness

Un paysage de fitness est un concept hérité de la biologie théorique, plus précisément de l'étude de l'influence du génome et de ses variations sur sa capacité à se transmettre. C'est le biologiste Wright [1] qui a d'abord proposé cette représentation du génotype et du phénotype dans le but d'expliquer les effets et les rôles qu'ont dans un processus évolutif la mutation, la diversité, le croisement et la sélection. Si le concept de paysage a d'abord été pensé pour l'évolution des espèces d'un point de vue biologique, il a été repris pour l'étude des algorithmes évolutionnaires. La figure 1.4 est une adaptation de la représentation graphique de ces paysages de Wright proposée par Malan [81].

Définition 11 ***Paysage de fitness* :** *Un paysage de fitness combinatoire est décrit par un triplet* $(\mathcal{X}, \mathcal{N}, f)$*.* $\mathcal{X}$ *est l'ensemble des points du paysage appelé espace de recherche.* $\mathcal{N} : \mathcal{X} \to \mathcal{X}^n$ *est la fonction de voisinage qui définit la proximité entre points.* $f : \mathcal{X} \to [0, 1]$ *(ou* $\mathbb{R}$*) est la fonction de fitness définissant la hauteur des différents points de* $\mathcal{X}$*. Un paysage de fitness dérivé d'un problème combinatoire est le triplet* $(\mathcal{X}, \mathcal{N}, f)$ *défini à partir du problème* $(\mathcal{X}, f)$ *et d'une fonction de voisinage* $\mathcal{N} : \mathcal{X} \to 2^{\mathcal{X}}$*.*

Dans ce cadre, l'objectif d'un algorithme d'optimisation est de trouver le point le plus haut

FIGURE 1.4 – *"Adaptation du paysage de fitness de Wright [1], qu'il appelait 'surface de valeurs sélectives'" [81]*



du paysage. Les points hauts, représentés par des "+" sur la figure, sont des points ou ensembles de points tels qu'aucun point adjacent n'est plus haut ; il s'agit ainsi des optima locaux. Un algorithme de recherche locale de type un hill climber appliqué sur un paysage de fitness retournera un de ces points haut.

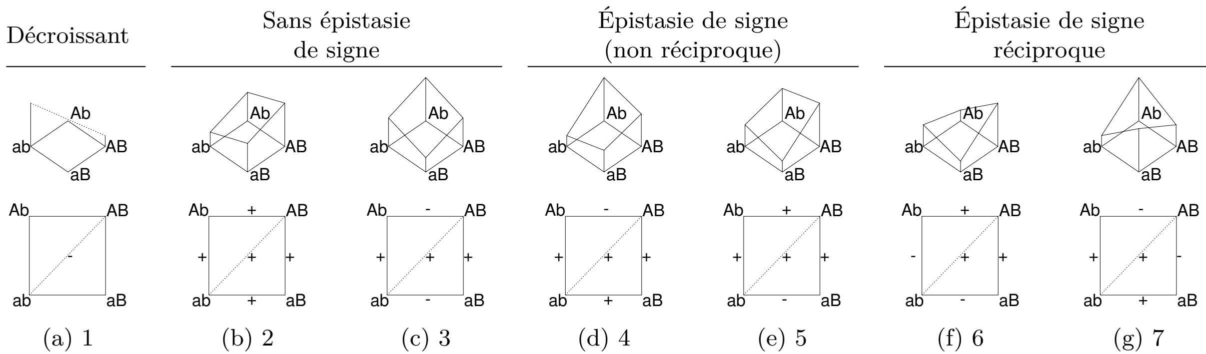
1.4.1 Épistasie et rugosité

L'apparition et la compréhension des causes qui mènent à avoir un grand nombre d'optima sont des axes de recherche importants pour l'amélioration des algorithmes d'optimisation évolutionnaires. En effet, si certaines causes peuvent être évitées ou détectées et contrecarrées alors il sera plus simple d'atteindre de meilleures solutions. L'épistasie, phénomène qui découle de l'interaction entre gènes et ici entre variables, est une condition nécessaire à l'apparition de plusieurs optima [82] et à rendre le problème multimodal. Plus précisément, c'est l'épistasie de signe réciproque qui remplit ce rôle.

Définition 12 *Épistasie* : *Soient deux mutations indépendantes α et β et des versions de gènes, ou valeurs de variables, $\{a, A\}$ et $\{b, B\}$ avec $a \xrightarrow{\alpha} A$ et $b \xrightarrow{\beta} B$. Il existe de l'épistasie de signe réciproque si $f(ab) < f(AB)$, $f(Ab) < f(ab)$ et si $f(aB) < f(ab)$, c'est-à-dire si les deux mutations cumulées sont bénéfiques tandis que chaque mutation prise indépendamment est détériorante.*

La figure 1.5 présente les différentes configurations menant à de l'épistasie. Les différents graphiques se lisent du point "ab" au point "AB". Les signes "+" et "-" correspondent à des améliorations ou détériorations après une mutation. L'épistasie apparaît lorsque les signes d'une même mutation ne sont pas les mêmes en fonction du point de départ de la mutation.

FIGURE 1.5 – Caractérisation de l'épistasie de signe : différentes configurations depuis deux solutions de référence distantes de 2 notées ab et AB.



(a) 1 (b) 2 (c) 3 (d) 4 (e) 5 (f) 6 (g) 7

La rugosité est l'expression de l'épistasie sur le paysage de fitness. Plus les variables des solutions sont liées entre elles, plus le nombre d'optima augmentera, ce qui est directement lié à la rugosité d'un paysage. Ainsi, plus un paysage est rugueux, plus il sera difficile pour un algorithme de recherche de l'exploiter efficacement et donc de trouver de bonnes solutions.

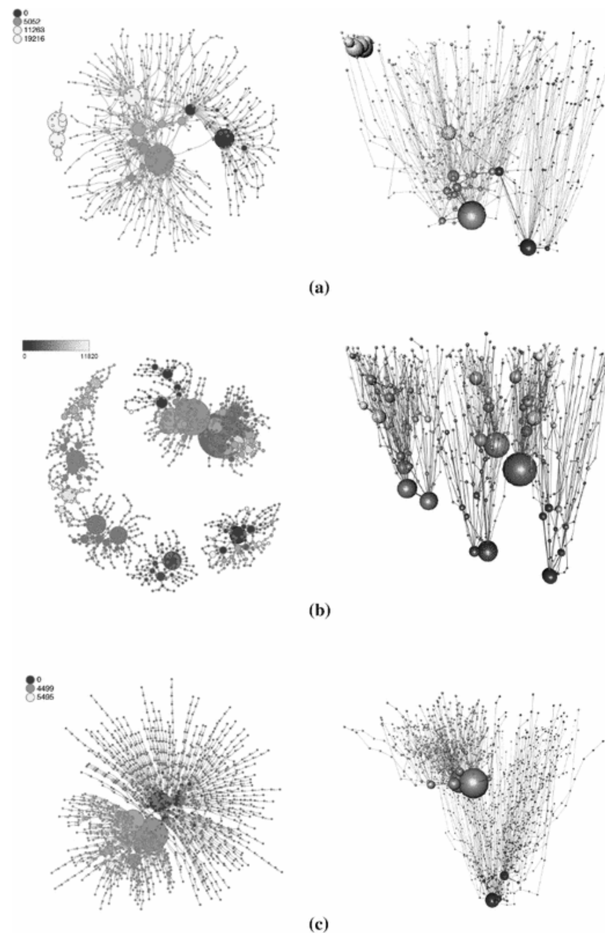
1.4.2 Bassins d'attraction

Les hill climber sont dans leur version classique des algorithmes non déterministes, que ce soit dans le choix de la solution initiale ou, dans le cas de climbers de type premier améliorant, dans l'ordre de parcours du voisinage. Ainsi, un hill climber peut retourner chaque optimum local du paysage avec une certaine probabilité, en fonction de sa règle pivot. Les bassins d'attractions [83] liés à chaque optimum sont les zones du paysage depuis lesquelles un algorithme de recherche locale a le plus de chance d'atteindre ces optimums. Bien que des trajectoires ascendantes peuvent souvent exister entre une proportion non négligeable du paysage et l'optimum global [84], le nombre généralement exponentiel de trajectoires ascendantes empruntables depuis un point de départ fait que l'algorithme pourra avoir tendance à choisir de mauvaises directions et, selon les propriétés du paysage et la règle pivot utilisée, être davantage attiré vers des zones bien spécifiques du paysage. Cela suggère que les heuristiques de choix ont bien plus d'influence sur les solutions retournées que la solution de départ.

1.4.3 Réseaux d'optima locaux

Les réseaux d'optima locaux [85] sont des outils permettant de représenter le comportement d'algorithmes plus sophistiqués comme les ILS qui alternent trajectoires ascendantes au moyen d'un climber et trajectoires perturbatives au moyen d'une marche aléatoire dans le paysage.

FIGURE 1.6 – *"Réseaux d'optima locaux d'une sélection d'instances aléatoires du problème du voyageur de commerce'" [86]*



Un aspect important est donc de déterminer la capacité d'un algorithme à sortir du bassin d'attraction d'un optimum, et plus particulièrement la possibilité qu'il a de sauter d'un bassin particulier à un autre. Un réseau d'optima locaux est un graphe dont les sommets sont les optima rencontrés lors de la recherche et les arêtes entre deux optima représentent une probabilité suffisante qu'il a d'atteindre le second depuis le premier. Grâce à cet outil, il a été possible de montrer que de mauvais choix tôt dans la recherche peut prévenir l'algorithme d'atteindre l'optimum global indépendamment des ressources allouées [46].

1.4.4 Visualisation des paysages de fitness

La visualisation des paysages de fitness peut être un outil puissant dans la compréhension des algorithmes et de leurs comportements. En effet, être capable de voir ce que fait l'algorithme

étudié sur un paysage donné peut permettre d'identifier la cause de l'efficacité ou de l'inefficacité de l'algorithme sur ce problème, et pouvoir ensuite corriger le défaut de l'algorithme. Dans cette optique d'étude des algorithmes ajoutée à l'étude simple des problèmes, une représentation des réseaux d'optima, en se basant sur les travaux de visualisation des graphes [87, 88] a pu mettre en évidence des clusters d'optima impossible à échapper pour certains algorithmes [46].

La figure 1.6 est une visualisation de la structure de problèmes du voyageur de commerce proposée par Ochoa et Veerapen [86]. Nous pouvons notamment y voir des réseaux d'optima indépendants renforçant la difficulté de ces instances [46]. Il reste à noter que ce type de visualisation se limite aux réseaux d'optima locaux, tant il est particulièrement complexe de visualiser un paysage de fitness combinatoire en tant que tel.

Chapitre 2

Motivations

Ce chapitre est dédié à l'introduction des motivations et de l'objectif qui conduisent le travail présenté dans ce manuscrit. Nous évoquons d'abord nos considérations initiales ainsi que leurs liens avec l'état de l'art. Nous énonçons ensuite les objectifs de la thèse selon plusieurs points de vue.

2.1 Rechercher une solution ou rechercher un algorithme

Nous l'avons vu dans le chapitre 1, une approche classique pour optimiser des problèmes de taille trop importante pour être résolus de manière exacte consiste à utiliser des stratégies de résolution générales telles que les métaheuristiques. Ces algorithmes répondent alors à deux objectifs. D'abord, ils doivent évidemment être efficaces sur les problèmes qu'ils doivent résoudre. C'est l'objectif principal pour n'importe quel algorithme d'optimisation par définition. Et ensuite, ils doivent avoir un champ d'action suffisamment large. En effet, un algorithme efficace sur plusieurs classes de problèmes est certainement plus intéressant que l'utilisation d'une multitude d'algorithmes ayant une efficacité éventuellement légèrement supérieure mais sur un nombre très restreint de problèmes et nécessitant une connaissance experte de la structure de ces problèmes (ce qui met une telle démarche algorithmique hors de portée des utilisateurs finaux). Ceci est d'ailleurs dans l'esprit des paradigmes modélisation-résolution intégrés tels que la programmation linéaire ou la programmation par contraintes. Les *No free lunch theorems* pour l'optimisation [17] soutiennent également un peu ce discours qui limite les perspectives de la "sur-spécialisation", même s'il ne s'agit bien sûr que d'un point de vu théorique. Mais finalement pour contrecarrer les *No free lunch theorems* et concevoir des algorithmes encore plus spécifiques et performants que ceux construits pour une classe de problème, il faudrait développer des algorithmes dédiés à chaque instance. Ceci peut prendre un sens si l'on automatise la construction de ces algorithmes. Une telle approche peut alors être vue comme la transformation du problème de recherche d'une solution optimale en un problème de recherche d'un algorithme

qui mène vers une solution optimale.

C'est finalement ce qui arrive dans le contexte de la configuration automatique d'algorithme si l'ensemble d'instances sur lequel doit être configuré cet algorithme n'est composé que d'une seule instance. Un processus est charger de configurer, choisir, ou construire un algorithme cible qui a pour tâche d'optimiser un problème à résoudre. L'algorithme configuré sera alors spécifique pour l'instance à résoudre. Un tel processus de configuration considère un algorithme d'optimisation paramétrable, où l'on peut voir chaque combinaison de valeurs de paramètres comme un comportement différent de l'algorithme. L'objectif pour l'algorithme de configuration est alors de trouver le comportement le plus favorable de l'algorithme d'optimisation cible. Comme nous le verrons dans la prochaine section, notre travail suit globalement cette approche. En revanche, plutôt que de configurer un algorithme complexe, nous chercherons à générer des algorithmes à la fois simples dans leurs mécanismes de recherche mais aussi très spécialisés dans le sens où ils seront dévolus à la résolution d'une instance de problème unique.

La spécificité de chaque algorithme sera induite par la fonction d'évaluation sur laquelle son heuristique de recherche sera basée. La génération d'algorithme s'effectue par évolution de la fonction d'évaluation. Nous pouvons identifier dans la littérature plusieurs approches évoquées dans le chapitre précédent (section 1.3) et pouvant s'apparenter à cette idée. Tout d'abord, la recherche locale guidée modifie la fonction d'évaluation et par conséquent le comportement de l'algorithme guidé par celle-ci ; le but est ici d'orienter différemment la recherche mais pas de générer au final un algorithme d'optimisation. L'apprentissage de fonctions d'évaluation s'apparente davantage au contexte décrit, à savoir la recherche de fonctions d'évaluation plus performantes que la fonction objectif. En particulier, le concept d'apprentissage introduit par Boyan et Moore [70] cherche à générer des fonctions menant vers des solutions de départ favorables à un hill climber. Dans les travaux que nous présentons dans ce manuscrit, nous cherchons au contraire à générer des fonctions guidant un hill climber vers de bonnes solutions de l'espace de recherche. Les finalités de ces objectifs sont similaires car les solutions de départ favorables finissent par être de bonnes solutions de l'espace de recherche. La problématique et le processus d'apprentissage qui en découle restent néanmoins différents.

Enfin, cette approche de génération d'algorithmes spécifiques, notamment via l'évolution de fonctions d'évaluation s'apparente à celle des modèles de substitution, concernant notamment l'objectif de générer une fonction dont qui se substitue à la fonction objectif. À la différence des modèles de substitution, nous ne cherchons pas à répliquer certaines structures locales ni à copier les propriétés de la fonction objectif, mais à trouver une fonction qui idéalement n'aurait qu'à partager l'optimum global de la fonction objectif tout en étant plus simple à optimiser.

2.2 Objectif

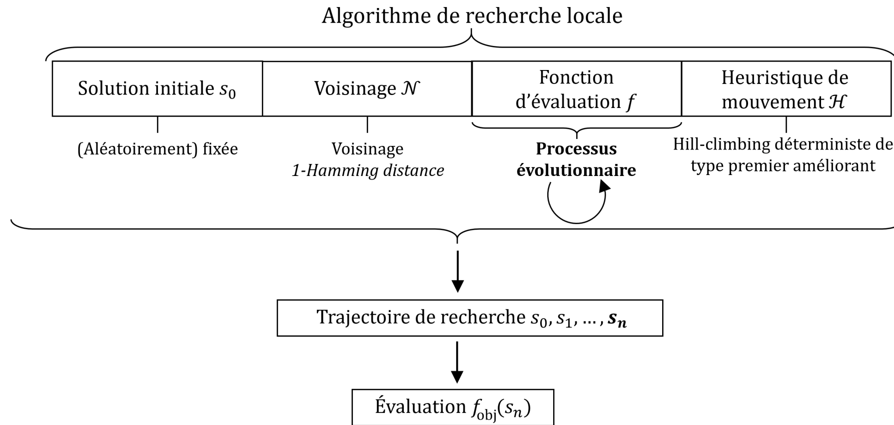
Dans cette section nous présentons plus précisément l'objectif de la thèse. Nous l'expliquerons sous deux points de vue : celui de la configuration automatique, et celui des paysages de fitness. Ensuite, nous définirons plus formellement notre objectif ainsi que les moyens que nous mettons en place pour l'atteindre.

Dans ce travail de thèse, nous nous limitons à la génération d'algorithmes de recherche locale, et plus précisément aux hill climbers. Comme nous l'avons vu, un algorithme de recherche locale est décrit par un voisinage $\mathcal{N}$, une fonction d'évaluation f et une heuristique de mouvement $\mathcal{H}$ (éléments basés sur l'espace de recherche $\mathcal{X}$). Nous ajoutons à ces trois composants la solution initiale qui influence elle aussi le comportement de l'algorithme en définissant le point de départ de l'espace de recherche. La proposition de nouveaux algorithmes se fait souvent à travers l'intégration de nouvelles heuristiques de mouvement, de règles pivots dans le cas des hill climbers (cf. section 1.2.3) ; la modification de l'algorithme via le voisinage a aussi été proposée dans les recherches à voisinage variable (section 1.2.3). De plus, si la solution de départ influence la recherche, celle-ci n'est pas forcément un facteur bloquant ni déterminant [84]. Nous proposons ici de générer des algorithmes dont la spécificité vient de la fonction d'évaluation, en explorant un espace de ces fonctions d'évaluation. La figure 2.1 représente les différents éléments décrivant un algorithme de hill climbing, où nous faisons évoluer la fonction d'évaluation afin de modifier les choix de l'algorithme, ses propriétés, et donc l'algorithme en lui-même. Une exécution d'un tel algorithme nous fournit une trajectoire possible à travers l'espace de recherche qui commence par la solution s_0 et se termine à un optimum local au sens de f (solution s_n). En évaluant s_n, nous évaluons la trajectoire qui nous a menée à cette solution, et donc en partie l'algorithme qui a produit cette trajectoire.

2.2.1 Vision d'un point de vue configuration automatique

Nous l'avons vu dans le chapitre précédent (section 1.3.3), l'objectif de la configuration automatique est de proposer un algorithme B dont le but est de chercher, choisir, ou construire un algorithme A pour optimiser un ensemble d'instances donné. De cet objectif nous pouvons remarquer que dans le cas particulier où l'ensemble d'instances n'en est composé que d'une seule, alors l'objectif d'un algorithme de configuration automatique devient proche du nôtre. En effet, dans ces deux cas la problématique est de produire un algorithme B qui doit générer/choisir un algorithme $A \in \mathcal{A}$ qui maximise un critère de performance p sur une instance $\mathcal{I}$ — en d'autres termes résoudre $\text{argmax}_{A \in \mathcal{A}} p(A, \mathcal{I})$). Si un algorithme de configuration possède évidemment un espace de recherche, c'est essentiellement sur ce point que notre approche en est différente. En effet, l'espace de recherche qui caractérisera notre espace des algorithmes sera celui des fonctions

Figure 2.1 – Génération d'algorithme de recherche locale par évolution de la fonction d'évaluation

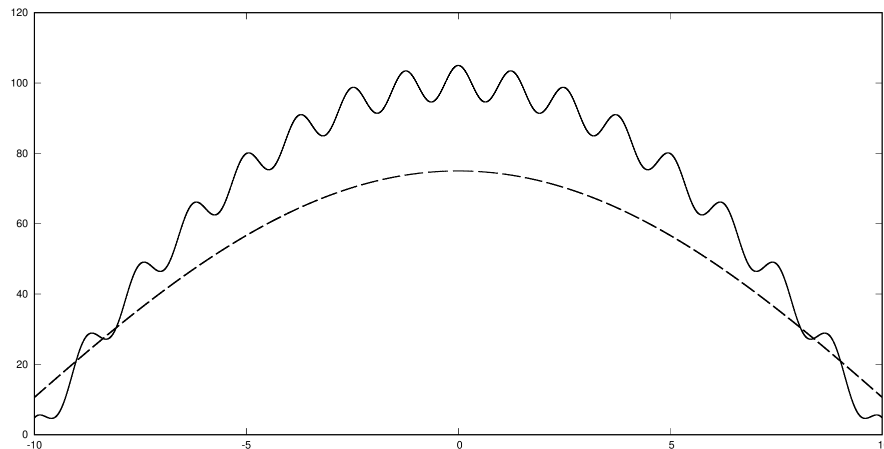


plutôt qu'un ensemble de paramétrages d'algorithme.

2.2.2 Vision d'un point de vue paysage de fitness

Prenons un exemple illustrant l'idée directrice de génération d'algorithmes de recherche locale spécifiques. Soient deux instances $(\mathcal{X}, f_1)$ et $(\mathcal{X}, f_2)$ qui partagent le même espace de recherche ainsi que leur optimum global s^*, c'est-à-dire $\forall s \in \mathcal{X},\ f_1(s) \leq f_1(s^*)$ et $f_2(s) \leq f_2(s^*)$, et soit une fonction de voisinage $\mathcal{N}$ défini sur $\mathcal{X}$. On note $L_1 = (\mathcal{X}, \mathcal{N}, f_1)$ et $L_2 = (\mathcal{X}, \mathcal{N}, f_2)$ les paysages de fitness correspondants aux deux instances définies. Considérons que L_1 est un paysage rugueux, donc que la corrélation entre l'évaluation d'une solution et sa distance à l'optimum est faible, ou que la qualité d'une solution donnée ne peut pas être facilement déduite grâce à un ensemble de couples $(s, f_{\mathrm{obj}}(s))$. En conséquence, une recherche basique comme un hill climber sera inefficace pour trouver de bonnes solutions sur ce paysage. À l'opposé, considérons le paysage L_2 comme un paysage lisse, et donc avec les implications contraires à celles du paysage rugueux. À cet égard, un algorithme d'optimisation simple appliqué au paysage $(\mathcal{X}, \mathcal{N}, f_2)$ peut être approprié pour résoudre l'instance $(\mathcal{X}, f_2)$, cependant ce ne sera pas le cas pour un paysage plus difficile à optimiser comme $(\mathcal{X}, \mathcal{N}, f_1)$. Il ressort de cet exemple qu'il est évident qu'un algorithme simple (comme un hill climber) ne sera pas efficace pour la résolution de n'importe quelle instance de problème. L'approche communément choisie pour améliorer l'algorithme $(\mathcal{X}, \mathcal{N}, f, \mathcal{H})$ est de choisir une heuristique de mouvement $\mathcal{H}$ plus appropriée au paysage étudié. Néanmoins, si l'on peut remarquer qu'un algorithme $(\mathcal{X}, \mathcal{N}, f_1, \mathcal{H})$ peut être un mauvais choix pour résoudre l'instance $(\mathcal{X}, f_1)$, l'algorithme $(\mathcal{X}, \mathcal{N}, f_2, \mathcal{H})$ sera capable d'atteindre cet objectif, puisque les deux fonctions f_1 et f_2 partagent le même optimum global.

Figure 2.2 – Exemple de fonctions partageant le même optimum avec des différences d'exploitabilité évidentes



La figure 2.2 montre, dans un cadre continu pour faciliter la visualisation, deux fonctions dont l'optimum global est le même ($x = 0$) mais dont les exploitabilités sont très différentes. En effet, la fonction qui nous a servi d'exemple dans le chapitre 1 possède plusieurs optima locaux qui sont autant d'obstacles pour un algorithme d'optimisation, et a fortiori pour des hill climbers qui ne peuvent s'en échapper. Au contraire la seconde fonction est parfaitement lisse sur l'espace de recherche $\mathcal{X} = [-10; 10]$. Le seul optimum local de la fonction est aussi son optimum global. Ainsi, un hill climber est parfaitement adapté pour trouver l'optimum de cette fonction. Et puisque les deux fonctions partagent le même optimum global, il peut être approprié d'utiliser la seconde fonction pour trouver l'optimum sur la première. C'est l'objectif que nous avons ici, générer une fonction plus simple à exploiter par un algorithme simple et dont les optima coïncident avec de bonnes solutions de la fonction objectif.

2.2.3 Problématique

Comme rappelé précédemment, résoudre une instance de problème d'optimisation $(\mathcal{X}, f_{\text{obj}})$ consiste à trouver la (ou une) solution s^* telle qu'il n'existe pas de solution avec une évaluation par f_{obj} supérieure — on se place toujours dans un contexte de maximisation. L'objectif d'identifier s^* tout en prouvant la non-existence de meilleures solutions n'étant pas atteignable en temps raisonnable pour toutes les instances, notamment les instances dont les espaces de recherche sont très grands, l'objectif secondaire de l'optimisation approchée consiste simplement à trouver la meilleure solution possible dans un temps de calcul, ou un nombre d'opérations, imparti. Trouver l'optimum global s^* en relâchant la nécessité de preuve coûteuse, ou à défaut une solution proche en termes d'évaluation, reste l'objectif de d'un tel algorithme de recherche ; il

s'agira également de l'objectif d'un algorithme généré pour la résolution d'une instance $(\mathcal{X}, f_{\text{obj}})$.

L'objectif d'un algorithme de génération d'algorithmes sera quant à lui de trouver, dans un espace d'algorithmes $\mathcal{A}$, un algorithme $A \in \mathcal{A}$ dont la l'espérance de la qualité de la solution retournée sera maximale. Optimiser ce critère revient à résoudre ce problème d'optimisation et de se placer dans ce cas dans l'espace de recherche des algorithmes. Dans le cas d'algorithmes de recherche stochastiques, optimiser l'espérance des solutions retournées permet de tendre à générer des algorithmes constants dans les solutions qu'ils renvoient, davantage qu'au moyen d'une évaluation tenant compte des potentielles meilleures solutions renvoyées par un algorithme apte à trouver de très bonnes solutions mais éventuellement de qualité très hétérogènes.

Le point central est ici d'évaluer un algorithme (calculer ou estimer l'espérance de sa performance) sur la seule instance $(\mathcal{X}, f_{\text{obj}})$. Les algorithmes générés seront ainsi spécifiques à cette instance, ajustés uniquement à la fonction objectif de l'instance à résoudre. Le processus de résolution de l'instance $(\mathcal{X}, f_{\text{obj}})$ est donc effectué par l'algorithme de génération d'algorithmes, lui bien générique et qui sera basé sur un autre modèle de représentation.

Enfin, il nous reste à remarquer que les algorithmes générés et les algorithmes d'optimisation classiques ne maximisent pas la même chose, en dépit d'un but commun. En effet, un algorithme d'optimisation telle qu'une métaheuristique M devra maximiser sa performance sur un ensemble d'instances généralement infini $\mathcal{J}$ d'une une classe de problèmes (par exemple l'ensemble des fonctions NK ou l'ensemble des fonctions MAX-SAT), Cette performance de l'algorithme M ne peut être estimée que sur un sous-ensemble d'instances de $\mathcal{J}$, ce qui rend son évaluation sensible à des biais, notamment au sur-apprentissage du sous-ensemble utilisé. Au contraire, la performance d'un algorithme dédié à la résolution d'une unique instance ne sera pas affectée par ces considérations puisque sa conception est justement établie autour d'un sur-apprentissage extrême. Ce changement de paradigme permet l'exploitation de structures au niveau de l'instance et non au niveau de la classe de problème ; si ces structures sont exploitées efficacement, alors la performance des algorithmes générés en sera augmentée.

2.3 Plan du manuscrit

Nous divisons le reste du manuscrit en trois chapitres. Le prochain chapitre présente l'ensemble des méthodes utilisées au cours de la thèse pour la génération d'algorithmes. Le chapitre 4 sera dédié aux différents résultats expérimentaux et aux comparaisons de l'efficacité des composants utilisés. Enfin, le chapitre 5 regroupe les analyses comportementales et paramétriques des algorithmes de génération et des algorithmes générés respectivement présentés et créés alors.Nous choisissons d'organiser ce manuscrit de cette manière plutôt que d'isoler chaque contribution car ces dernières sont particulièrement dépendantes les unes des autres. Cette structure nous permet

ainsi de réduire la répétition d'éléments à travers les différents chapitres.

Chapitre 3

Génération d'algorithmes

Ce chapitre est dédié aux différents algorithmes développés tout au long de la thèse, ainsi qu'à leur fonctionnement. Nous présentons tout d'abord les modèles d'algorithmes sur lesquels se base notre travail, qui définiront donc l'étendue de l'espace de recherche dans lequel nous travaillons.

La génération d'algorithmes telle que proposée dans ce travail repose sur l'évolution de fonctions d'évaluation. Nous présentons donc les modèles de fonction utilisés au cours de la thèse afin de préciser les espaces de fonctions considérés. Pour explorer ces espaces de fonctions, nous nous basons sur des stratégies évolutionnaires dont nous décrivons également, dans ce chapitre, les composants.

3.1 Modèles d'algorithmes

Comme nous l'avons vu dans le chapitre 2, nous allons utiliser l'évolution de fonctions d'évaluation comme moyen de produire des algorithmes de recherche locale plus efficaces. Pour ce faire, nous choisissons comme représentant de ces algorithmes de recherche locale, un algorithme simple de hill climbing. Ce choix est d'abord motivé par la grande utilisation de ces algorithmes — que ce soit en tant que stratégie de résolution rapide ou comme composant d'algorithmes plus sophistiquées —, par leur simplicité, et enfin parce que ces algorithmes convergent rapidement en atteignant un optimum local. Sur ce dernier point, le temps de calcul est un facteur non négligeable dans le sens où l'évaluation de la performance des algorithmes, en particulier stochastiques, nécessiteront de nombreuses exécutions.

3.1.1 Recherche locale stochastique avec montée stricte

La stratégie de hill climber employée ici est la variante de montée stricte avec une heuristique de choix du premier améliorant (voir section 1.2.3), très communément utilisée et implémentée

dans des métaheuristiques comme les ILS (voir section 1.2.3). Sa restriction sur le choix de mouvement parmi les voisins strictement améliorants facilite son étude puisqu'il va systématiquement s'arrêter sur un optimum local. Une autre variante non stricte, dite stochastique, permet au hill climber d'accepter des mouvements neutres dans le but de naviguer sur les plateaux des paysages de fitness. Néanmoins, l'heuristique employée possède un élément stochastique en évaluant le voisinage dans un ordre aléatoire à chaque pas de recherche ; le choix aléatoire de la solution initiale demeurant un autre composant stochastique de l'algorithme. L'algorithme 6 décrit cette stratégie de recherche.

Algorithme 6 Hill climber stochastique

Entrée : une fonction d'évaluation f, une fonction de voisinage $\mathcal{N}$
Sortie : un optimum local s de f

1: $s \leftarrow$solution_aléatoire()
2: **tant que** $\neg$optimum_local(f, s) **faire**
3: **pour chaque** $s' \in \mathcal{N}(s)$ **faire**
4: **si** $f(s') > f(s)$ **alors**
5: $s \leftarrow s'$
6: **break**
7: **fin si**
8: **fin pour**
9: **fin tant que**
10: **retourne** s

La fonction booléenne optimum_local(f, s) renvoie vrai si aucune solution voisine de s ne l'améliore strictement vis-à-vis de f — $\forall s' \in \mathcal{N}(s), f(s') \leq f(s)$ — et renvoie faux dans les autres cas. L'ordre d'exécution du "pour chaque" en ligne 3 n'étant pas défini, c'est l'une des sources du caractère aléatoire de l'algorithme. En effet, entre deux exécutions du même algorithme, avec la même solution de départ, lors d'une itération où plusieurs voisins améliorent la solution courante, les recherches peuvent diverger pour atteindre des optima différents.

Une seule exécution d'un algorithme stochastique donne relativement peu d'information sur son efficacité générale. Plus le nombre d'exécutions est élevé, plus l'évaluation de l'algorithme est précise. Cette dimension est importante pour que la génération d'algorithme puisse comparer différents algorithmes entre eux, tout en conservant un budget de calcul le plus bas possible afin d'effectuer un maximum d'itérations. Nous reviendrons plus longuement sur ce point par la suite.

3.1.2 Recherche locale déterministe de montée stricte à départ unique fixé

Pour réduire le biais identifié précédemment quant à l'aspect stochastique, nous proposons de considérer une seconde variante de hill climber dépourvu de ses composants stochastiques. Dans

cette variante déterministe, la solution de départ est générée en amont mais cette étape n'est plus incluse dans le hill climber. C'est-à-dire que la solution de départ devient une donnée de l'algorithme, sera donc choisie au départ du processus de génération et par conséquent commune à tous les algorithmes générés. L'ordre de parcours des voisins lors de la recherche du premier améliorant ne sera plus aléatoire mais cyclique. Un ordre sera fixé au départ de la génération et sera le même pour chaque algorithme. Il s'agit de la règle originelle d'exploration déterministe (cyclique) du voisinage telle que mentionnée dans [89]. Si la variante de parcours aléatoire a un intérêt dans un contexte de recherche itérée, Hoos et Stützle précisent que les performances moyennes des deux variantes sont potentiellement similaires.

Algorithme 7 Hill climber déterministe

Entrée : une fonction d'évaluation f, une fonction de voisinage $\mathcal{N}$, une solution de départ s_0, un ordre de voisins θ
Sortie : un optimum local s de f

1: $s \leftarrow s_0$
2: **tant que** $\neg$optimum_local(f, s) **faire**
3:     **pour chaque** $s' \in \theta(\mathcal{N}(s))$ **faire**
4:         **si** $f(s') > f(s)$ **alors**
5:             $s \leftarrow s'$
6:             **break**
7:         **fin si**
8:     **fin pour**
9: **fin tant que**
10: **retourne** s

L'algorithme 7 présente la transformation de l'algorithme 6 en une version déterministe avec les modifications décrites plus haut. Deux paramètres s'ajoutent aux deux nécessaires dans la version stochastique : la solution s_0 de départ qui sera constante pour tout le processus de génération, ainsi que l'ordre de visite des voisins.

Grâce à cet algorithme, et en utilisant une solution de départ fixée, pour une fonction d'évaluation donnée, la solution retournée sera unique. En conséquence, les algorithmes générés avec ce modèle pourront être évalués plus simplement (une seule exécution nécessaire) et sans estimation puisque l'algorithme retournera toujours la même solution. Ce lien entre fonctions d'évaluation et solutions de l'espace de recherche montre qu'en utilisant ce modèle d'algorithme, nous transformons le problème d'optimisation dans l'espace de recherche $\mathcal{X}$, en un problème d'optimisation d'une fonction de fitness pour un hill climber.

3.1.3 Recherche locale déterministe de montée stricte à départs multiples fixés

Pour relâcher les fortes contraintes du modèle présenté précédemment, nous proposons d'utiliser exactement le même modèle que celui décrit par l'algorithme 7. Cependant, plutôt que d'utiliser une solution de départ unique pour évaluer l'algorithme dans son ensemble, nous utiliserons un ensemble de solutions de départ afin de pallier les éventuels effets d'une optimisation trop localisée des algorithmes présentés dans la section précédente.

L'algorithme étant toujours déterministe, son évaluation reste simple. Il suffit en effet de calculer la somme des évaluations des solutions retournées pour avoir une évaluation de l'algorithme ($\sum_{s_0 \in \mathcal{X}_0} f_{\text{obj}}(HC(f, \mathcal{N}, s_0, \theta))$, avec $\mathcal{X}_0$ l'ensemble des solutions de départ). La taille de l'ensemble des solutions de départ est un paramètre supplémentaire propre à ce modèle d'algorithme. Il devra être étudié comme tel, en particulier son influence sur les performances des algorithmes générés selon s'il est petit et se rapproche du modèle présenté précédemment, ou grand et s'apparente aux hill climbers stochastiques.

3.2 Modèles de fonction d'évaluation

Maintenant que l'on a défini les composants fixes des algorithmes que l'on générera, il nous reste à définir les parties variables qui seront à la source de l'évolution des hill climbers. Pour ce faire, nous proposons différents modèles de fonctions de fitness. Chaque instance de ces modèles correspond à une fonction de l'espace de recherche des fonctions que l'on pourra explorer lors de la génération d'algorithmes.

Soit une fonction objectif $f_{\text{obj}} : \mathcal{X} \to \mathbb{R}$, où $\mathcal{X}$ est l'espace de recherche du problème à résoudre. Un modèle de fonction $\mathcal{M}$ doit être restreint à des fonctions d'évaluation sur le même espace de recherche $\mathcal{X}$, i.e. $\forall f \in \mathcal{M}, f : \mathcal{X} \to \mathbb{K}$. Le codomaine $\mathbb{K}$ sur lequel les solutions évaluées sont transformées n'est pas nécessairement le même que celui de la fonction objectif de référence. Toutefois, un ordre sur ses éléments doit être défini pour que les hill climbers puissent exploiter ces fonctions. Le nombre de valeurs différentes de $\mathbb{K}$ doit être suffisant. Par exemple si l'on considérait $\mathbb{K} = \{0, 1\}$ pour travailler sur la satisfaction d'une formule propositionnelle, alors presque toutes les solutions du paysage associé seraient des optimums locaux et un algorithme de recherche locale de montée stricte serait bloqué à son point de départ — à moins que le point de départ soit un voisin de la ou de l'une des solutions au problème, où l'algorithme effectuerait dans ce cas un unique pas vers un optimum global. Notons que pour les problèmes pseudo-booléens sur lesquels nous nous concentrons ici, une solution aléatoire est en moyenne à $\frac{N}{2}$ itérations de l'optimum global (N la taille de la chaîne de bits qui composent une solution),

donc un algorithme de recherche locale efficace doit avoir la possibilité d'établir une trajectoire de recherche au moins aussi longue.

Dans ce travail, nous nous limitons à l'étude des problèmes d'optimisation pseudo-booléens. Ainsi, l'espace de recherche des problèmes et l'ensemble de départ des fonctions d'évaluation générées est $\mathcal{X} = \{0,1\}^N$, avec N la taille de l'instance du problème à résoudre. Nous proposons trois modèles de fonction d'évaluation. Les deux premiers sont des adaptations des problèmes que l'on étudie, un modèle NK qui correspond à une version relâchée des problèmes NK présentés au chapitre 1, et un modèle MAX-3SAT qui est un cas particulier de formules pour le problème MAX-SAT dont toutes les clauses sont composées d'exactement trois littéraux (voir section 1.1.1). Le troisième modèle est une somme de fonctions pseudo-booléennes basiques d'arité 2.

3.2.1 Modèle NK

Le premier modèle que l'on présente ici est basé sur les modèles de paysages de fitness NK. Plus précisément, les fonctions créées depuis le modèle NK forment un sous-ensemble des instances possibles NK avec $K = 1$. Ces fonctions sont composées de N fonctions pseudo-booléennes. Chacune de ces sous-fonctions lie deux variables des solutions de l'espace de recherche. L'évaluation d'une solution par une fonction suivant ce modèle est la somme de toutes les évaluations des sous-fonctions qui la composent. Une sous-fonction renvoie une valeur dans l'intervalle $[0;1]$ selon les variables liées par celle-ci (voir section 1.1.1).

Le tableau 3.1 présente un exemple partiel d'une fonction suivant le modèle NK. La première sous-fonction de cet exemple lie les 49e et 72e bits des solutions de $\mathcal{X}$. Évaluer une solution avec ces bits fixés respectivement à 1 et 0 avec cette sous-fonction, la valeur de 0.586768 sera retournée. Pour créer une fonction aléatoire avec ce modèle, nous choisissons chaque variable pour chaque lien aléatoirement entre tous les bits d'une solution, les contributions au score sont des valeurs décimales à six digits choisies aléatoirement dans l'ensemble $\{\frac{k}{1000000} : k \in [\![0;999999]\!]\}$.

Contrairement au problème NK, ce modèle ne requiert pas que le bit i fasse partie des variables liées de la sous-fonction i. C'est en ceci que ce modèle NK est une version relâchée du problème homonyme.

3.2.2 Modèle MAX-3SAT

Le second modèle de fonction d'évaluation se base sur des instances particulières du problème MAX-SAT. Un problème MAX-SAT est constitué de plusieurs clauses qui sont des disjonctions de littéraux. La particularité du modèle MAX-3SAT est que chaque clause comporte trois littéraux exactement (voir tableau 3.2). L'évaluation d'une solution par une fonction suivant le

TABLE 3.1 – Exemple partiel d'une fonction basée sur le modèle NK

Liens		Contributions au score			
Premier	Second	00	01	10	11
49	72	0.411447	0.399435	0.586768	0.063350
56	112	0.620090	0.946004	0.566001	0.971463
126	91	0.958589	0.292135	0.774098	0.983260
121	32	0.685303	0.443777	0.595195	0.950059
⋮	⋮	⋮	⋮	⋮	⋮

TABLE 3.2 – Exemple d'instance MAX-3SAT ($N = 8$, $M = 5$)

Conjonction générale	Clauses (notation simplifiée)
$\wedge$	2 -6 -3
	-7 -2 4
	-5 -3 8
	6 2 1
	-6 5 -4

modèle MAX-3SAT est égale au nombre de clauses satisfaites par la solution. Une fonction MAX-3SAT est composée de M clauses, ainsi pour construire aléatoirement une fonction, chaque littéral de chaque clause est choisi aléatoirement parmi tous les $2N$ littéraux possibles (chaque variable et sa négation).

Dans la suite du manuscrit nous ferons référence au modèle MAX-3SAT comme 3-SAT et MAX-3SAT indifféremment.

3.2.3 Modèle de somme de fonctions pseudo-booléennes

Le modèle de somme de fonctions pseudo-booléennes consiste en une somme pondérée d'opérations booléennes sur les bits des solutions évaluées. Les opérations possibles sont limitées aux opérateurs d'arité 2 tout en pouvant ajouter une négation sur cette opération. Une fonction utilisant ce modèle est composée d'un certain nombre de ces opérations, et chaque opération fait appel à deux bits des solutions. Ensuite, le poids de chaque fonction est compris dans l'intervalle $[\![1; 10]\!]$. Grâce à cet intervalle restreint, chaque sous-fonction lie deux variables à l'instar du modèle NK, mais réduit l'espace de recherche pour l'algorithme de génération en plus de ne faire appel qu'à un seul poids plutôt qu'à quatre contributions différentes. L'ensemble des opérateurs pour construire une sous-fonction est $\{\wedge, \vee, \oplus, \rightarrow\}$. Cet ensemble offre plus de possibilités que

TABLE 3.3 – Exemple d'une fonction basée sur le modèle de somme de fonctions pseudo-booléennes ($N = 8, M = 4$)

Variable id.	Opérateur	Variable id.	Négation	Poids
7	$\rightarrow$	1		7
2	$\vee$	5	$\neg$	6
4	$\oplus$	8	$\neg$	10
6	$\wedge$	3		6

la simple disjonction du modèle MAX-3SAT. De plus, avoir le choix entre plusieurs opérateurs entrave l'apparition de neutralité dans le paysage de la fonction induite par l'opérateur $\vee$ utilisé par le modèle MAX-3SAT. En effet, lorsque les deux parties de l'opérateur $\vee$ sont fixées à 1, un changement de l'une ou l'autre des parties ne sera suffisante pour changer l'évaluation de l'opérateur ; seul un changement des deux parties pourra induire un changement de la valeur retournée par $\vee$. En d'autres termes, dans ce cas précis, la sous-fonction ne donne plus aucune indication quant à la direction prendre pour l'exploration de l'espace de recherche (présence d'un plateau).

Le tableau 3.3 est un exemple d'une fonction basée sur le modèle de somme de fonctions pseudo-booléennes. Prenons comme exemple la seconde opération de la fonction $(6 \times [\neg(x_2 \vee x_5)])$. Cette opération ajoutera à l'évaluation de la solution une contribution de 6 si et seulement si les bits de cette solution 2 et 5 sont fixés à 0. Une fonction basée sur ce modèle est composée de M sous-fonctions. Dans la suite de ce manuscrit nous considérons $M = 2N$, c'est-à-dire le double du nombre de sous-fonctions présentes dans une fonction basée sur le modèle NK. Cette différence est justifiée par la différence d'expressivité des deux modèles (on compense en partie la réduction de la taille de l'espace de recherche des fonctions liée au nombre de poids différents).

Pour construire aléatoirement une fonction suivant ce modèle, nous choisissons chaque élément qui la compose aléatoirement : chaque variable de chaque sous-fonction parmi les N disponibles, chaque opérateur parmi les quatre $\{\wedge, \vee, \oplus, \rightarrow\}$ et la négation avec une chance sur deux. Chaque poids est tiré aléatoirement et uniformément dans l'intervalle $[\![1; 10]\!]$.

Enfin, pour développer sur le nombre limité d'opérateurs utilisés par ce modèle, les 16 opérateurs binaires d'arité 2 peuvent être exprimés avec seulement les quatre opérateurs $\{\wedge, \vee, \oplus, \rightarrow\}$ et leur négation. Le tableau 3.4 présente les opérateurs binaires ainsi que leur évaluation en fonction des valeurs utilisées en entrée ; ensuite, la notation utilisée communément pour décrire chaque opérateur ; enfin, un exemple utilisant le format permis par le modèle pour atteindre l'opérateur décrit montre que l'utilisation d'un ensemble d'opérateurs restreint n'empêche pas l'apparition de structures potentiellement nécessaires à la génération d'algorithmes. Les lignes à fond gris correspondent aux opérateurs impossibles à instancier avec les quatre opérateurs mis

Table 3.4 – Liste des opérateurs binaires, leur évaluation et notation, ainsi qu'une formule au même comportement suivant les contraintes du modèle.

Évaluation de l'opérateur				Notation de l'opérateur	Clause suivant les contraintes
$a = 0$		$a = 1$			
$b = 0$	$b = 1$	$b = 0$	$b = 1$		
0	0	0	0	$\bot$	$a \oplus a$
0	0	0	1	$a \wedge b$	$a \wedge b$
0	0	1	0	$\neg(a \rightarrow b)$	$\neg(a \rightarrow b)$
0	0	1	1	a	$a \wedge a$
0	1	0	0	$\neg(a \leftarrow b)$	$\neg(b \rightarrow a)$
0	1	0	1	b	$b \wedge b$
0	1	1	0	$a \oplus b$	$a \oplus b$
0	1	1	1	$a \vee b$	$a \vee b$
1	0	0	0	$\neg(a \vee b)$	$\neg(a \vee b)$
1	0	0	1	$a \iff b$	$\neg(a \oplus b)$
1	0	1	0	$\neg b$	$\neg(b \wedge b)$
1	0	1	1	$a \leftarrow b$	$b \rightarrow a$
1	1	0	0	$\neg a$	$\neg(a \wedge a)$
1	1	0	1	$a \rightarrow b$	$a \rightarrow b$
1	1	1	0	$\neg(a \wedge b)$	$\neg(a \wedge b)$
1	1	1	1	$\top$	$a \rightarrow a$

à disposition, à moins d'utiliser la même variable pour les deux entrées de l'opérateur. Ceci a pour conséquence de réduire la probabilité d'apparition des opérateurs concernés. Cependant, il convient de noter que pour le premier et dernier opérateur de la liste, cela ne pourra qu'être bénéfique à la génération d'algorithmes tant ils n'affectent en rien l'exécution d'un hill climber de par leur valeur constante.

3.2.4 Mutation de fonction d'évaluation

Nous avons défini plusieurs modèles de fonction d'évaluation ainsi qu'une démarche à suivre pour créer une fonction aléatoire pour chacun de ces modèles. Ces premiers éléments sont nécessaires à l'initialisation de la génération des fonctions d'évaluation. Cependant, pour continuer le processus de génération et pour être capable de faire évoluer les fonctions, il est nécessaire de définir un opérateur de mutation. C'est au moyen de ces transformations que l'algorithme de génération pourra explorer l'espace de recherche des fonctions d'évaluation.

Trois modèles de fonctions ont été décrits, mais plutôt que de créer un processus de mutation pour chacun d'entre eux, nous choisissons de proposer un procédé de mutation général. Pour ce faire, nous définissons une instance d'un problème comme plusieurs listes ordonnées de composants interagissant entre eux :

— Modèle NK : la liste des liens entre variables, puis celle des contributions au score.
— Modèle MAX-3SAT : seule une liste ordonnée de littéraux est nécessaire pour décrire une instance.
— Modèle somme de fonctions pseudo-booléennes : liste des couples de variables utilisées par chaque sous-fonction, liste des opérateurs correspondants, liste des négations (oui/non), et liste des poids associés à chaque sous-fonction (cf. section 3.2.3).

Pour muter une une fonction, chaque élément f_{ij} de chaque liste ordonnée f_i décrivant la fonction f a une probabilité de changer de valeur aléatoirement. Les probabilités de mutation de chaque élément sont spécifiques à chaque type de liste, sous la forme $P_i = \frac{m}{|f_i|}$, où m est le paramètre associé à l'ensemble muté, et $|f_i|$ la taille de la liste f_i. Cette procédure est détaillée dans l'algorithme 8.

Cette procédure de mutation est naturellement stochastique, et son effet dépend essentiellement de l'ensemble de probabilités utilisé. En effet, plus les probabilités sont grandes, plus la fonction mutée différera syntaxiquement de la fonction initiale. Le nombre d'éléments de la fonction mutée suit une loi binomiale de paramètres $(P_i,|f_i|)$, ce qui permet de générer efficacement le nombre d'éléments à muter pour chaque mutation.

Pour une plus grande efficacité et afin de limiter l'ajout de paramètres à l'algorithme de

Algorithme 8 Mutation de fonctions d'évaluation

Entrée : une fonction d'évaluation f, un ensemble d'intervalles I, un ensemble de probabilités P

Sortie : une mutation de f notée f'

```
1: f' ← f
2: pour i = 1 → |f'| faire
3:     pour j = 1 → |f'_i| faire
4:         si rand([0;1]) < P_i alors
5:             f'_ij ← rand(I_i)
6:         fin si
7:     fin pour
8: fin pour
9: retourne f'
```

génération, nous avons proposé un schéma de régulation auto-adaptatif des paramètres P_i. Au départ, nous initialisons l'ensemble de probabilités aux plus faibles valeurs : $P_i = \frac{1}{|f_i|}$. À chaque fin d'itération du processus de génération, la liste contenant les probabilités de mutation des éléments de la fonction sont mises à jour en fonction du nombre d'éléments effectivement mutés. Par exemple, pour une fonction MAX-3SAT contenant 10 clauses, la liste contenant l'unique probabilité de mutation est initialisée à $(\frac{1}{30})$ (3 littéraux pour chacune des 10 clauses). Ensuite, si le générateur suivant la loi binomiale de paramètres $(\frac{1}{30}, 30)$ génère une mutation affectant deux éléments parmi les 30 et que cette mutation est sélectionnée, alors, pour l'itération suivante, la probabilités de mutation évoluera en $(\frac{2}{30})$. Ce mécanisme supprime le besoin de configurer la mutation en fonction du problème que l'on résout en proposant des mutations que l'on suppose adaptées à l'état de la recherche, puisque les probabilités de muter chaque élément se recentrent autour du niveau qui a permis de générer le meilleur mutant de la génération précédente.

3.3 Sélection de mutations d'algorithmes

La question de la sélection du mutant dans un algorithme évolutionnaire est centrale puisque la pression de sélection est l'élément du processus évolutionnaire qui va orienter la recherche vers les meilleurs individus (algorithmes). Ceci repose sur l'évaluation de la performance des algorithmes, qui dépendra elle-même de leur aspect déterministe ou stochastique. Nous divisons ici le problème de sélection en deux sous-problèmes distincts : la sélection d'algorithmes déterministes, puis celle d'algorithmes stochastiques.

3.3.1 Sélection d'algorithmes déterministes

La sélection au sein d'un ensemble d'algorithmes déterministes est facilité par le nombre réduit de solutions renvoyées par ces algorithmes. Les modèles d'algorithmes présentés en section 3.1.2 et 3.1.3 sont les seuls que l'on considère dans ce travail. De plus, il convient de noter que le premier algorithme présenté (section 3.1.2) est un cas particulier du second, où l'ensemble des solutions de départ n'est composé que d'une seule solution. Ainsi, pour la suite de cette section, nous ne considérerons que le problème de sélection d'algorithmes de recherche locale à points de départ multiples.

Considérons un ensemble d'algorithmes déterministes $\mathcal{A}$, un problème d'optimisation $(\mathcal{X}, f_{\text{obj}})$, et un ensemble de solutions de départ $\mathcal{X}_0 \subset \mathcal{X}$. Chaque algorithme $A \in \mathcal{A}$ a une performance propre $\mathbb{E}(f_{\text{obj}}(A(s))|s \in \mathcal{X}_0)$ sur cette instance, avec $A(s)$ représentant l'algorithme de recherche A appliqué depuis le point de départ s. Comme $A(s)$ est déterministe, $f_{\text{obj}}(A(s))$ l'est aussi et l'évaluation de A relativement à l'ensemble $\mathcal{X}_0$ revient à calculer la performance moyenne de A depuis chacun des points de départ, soit $\frac{\sum_{s \in \mathcal{X}_0} f_{\text{obj}}(A(s))}{|\mathcal{X}_0|}$. Le problème de sélection parmi un ensemble d'algorithmes $\mathcal{A}$ dans un contexte déterministe et en fixant un ensemble de points de départ revient alors à déterminer $\text{argmax}_{A \in \mathcal{A}} \sum_{s \in \mathcal{X}_0} f_{\text{obj}}(A(s))$. Ce processus de sélection a une complexité de $\mathcal{O}(|\mathcal{X}_0| \times |\mathcal{A}| \times (\mathcal{O}(f_{\text{obj}}) + \mathcal{O}(A)))$.

3.3.2 Sélection d'algorithmes stochastiques

La sélection d'algorithme parmi un ensemble d'algorithmes stochastiques ne peut se plier aux mêmes règles que la sélection en contexte déterministe. En effet, le nombre de solutions qui peuvent être retournées par un algorithme stochastique est potentiellement très grand et sont dans le cas général associées à différentes probabilités d'obtention. Le problème de la sélection d'algorithmes dans ce contexte est alors un problème de décision avec de l'information imprécise.

La performance d'un algorithme stochastique peut être approchée en moyennant la performance observée sur un certain nombre d'exécutions. Naturellement, plus le nombre d'exécutions sera élevé, plus l'estimation de sa performance sera précise. Un équilibre entre précision de l'évaluation et utilisation des ressources calculatoires doit donc être finement étudié. Une précision trop faible entraînerait une sélection trop aléatoire et donc la génération serait inefficace. À l'inverse, chercher à atteindre une précision très élevée ne donnerait pas nécessairement lieu à des changements très significatifs dans la sélection pour que le budget supplémentaire dépensé à affiner la précision de l'évaluation des algorithmes soit rentable sur le long de la recherche.

Au cours de la thèse, plusieurs méthodes de sélection ont été testées. Nous présentons ici les processus de sélection qui ont montré leur efficacité. La première stratégie d'évaluation se limite

à la question du choix entre deux algorithmes au moyen d'un test statistique. Nous discutons ensuite de méthodes permettant de choisir un algorithme parmi n, en distribuant uniformément le budget permettant de les estimer, ou en appliquant des stratégies de bandits. Enfin, nous présentons un algorithme de sélection basé sur un système de tournoi.

Dominance statistique

Soit deux algorithmes stochastiques A_1 et A_2 appliqués à un problème $(\mathcal{X}, f_{\text{obj}})$. L'ensemble des exécutions de ces algorithmes est un ensemble de solutions $\mathcal{X}_{A_1} \subset \mathcal{X}$ et $\mathcal{X}_{A_2} \subset \mathcal{X}$. Une exécution particulière de ces algorithmes revient à choisir aléatoirement un élément de cet ensemble selon une loi de probabilité inconnue. Le problème de sélection dans ce contexte revient alors à comparer ces deux algorithmes et donc de choisir le meilleur. Pour comparer deux algorithmes il peut être utile d'utiliser un test statistique afin de montrer la supériorité de l'un sur l'autre. Nous proposons ici d'utiliser le test de Wilcoxon-Mann-Whitney [90], test non paramétrique relativement simple à déployer à grande échelle. Ce test est notamment utile pour montrer que deux échantillons de valeurs n'ont pas été générés à partir de la même loi de probabilité. Ici, chaque échantillon est généré avec un des deux algorithmes à évaluer. Ainsi, si le test statistique montre une différence significative entre les échantillons, on peut alors supposer que la différence de performance observée entre les algorithmes à l'origine de ces valeurs est effective.

Pour ce qui est de la fiabilité des conclusions du test statistique, nous considérons une différence entre les algorithmes avec une valeur-p de 0.05, et fixons le nombre d'éléments de chaque échantillon à 100 valeurs afin d'être en mesure d'observer des différences entre algorithmes même relativement proches. Avec une telle valeur-p, on peut s'attendre à une confiance relativement grande dans le résultat du test bien que, puisque ce test sera effectué à chaque itération, les faux positifs seront inévitables et réguliers. Il convient néanmoins de rappeler que les processus évolutionnaires n'ont pas besoin de sélectionner de manière optimale à chaque itération pour être efficaces, mais d'appliquer de manière globale une pression de sélection suffisamment favorable à l'émergence de bonnes solutions.

Distribution uniforme de budget

Considérons n fonctions d'évaluation (désignant n algorithmes de recherche) et un budget B d'appels à la fonction objectif afin d'en sélectionner une. Une façon simple de répartir le budget alloué à la sélection est d'exécuter chaque algorithme $\frac{B}{n}$ fois, et de calculer une estimation de l'espérance de chacun des algorithmes à partir de ces exécutions. Le score d'un algorithme A sera alors la performance moyenne de son exécution, et la fonction sélectionnée sera celle avec le meilleur score. Ainsi, nous respectons le budget alloué tout en sélectionnant la meilleure fonction

selon les informations disponibles. Néanmoins, les algorithmes à comparer étant stochastiques, les scores ne sont pas constants à chaque comparaison pour un même algorithme et donc les hiérarchies entre mêmes algorithmes peuvent varier d'une génération à l'autre. Augmenter le budget alloué pour le calcul d'un score permet d'améliorer la fiabilité de cette sélection. Cependant, pour un budget global fixe (d'appels à la fonction objectif), augmenter le budget de chaque itération implique un nombre d'itérations plus faible. Un choix doit donc être fait autour de ce paramètre entre précision de la sélection et nombre d'itérations.

Sélection UCB

Un ensemble d'algorithmes stochastiques peut être assimilé à un ensemble de variables aléatoires. Sélectionner le meilleur algorithme en utilisant un budget de B appels à la fonction objectif (appliquée uniquement sur les solutions retournées par ces algorithmes) peut être vu comme un de problème bandit manchot. L'ensemble d'algorithmes est considéré comme un ensemble de bras et le budget comme l'horizon de l'algorithme de bandit. La politique de bandit UCB [91] (*Upper-Confidence Bound*) est très utilisée pour ce type de problématiques, en particulier dans des applications de sélection d'opérateurs [92, 93]. Dans notre cas, UCB est utilisé pour affiner l'estimation des algorithmes les plus prometteurs tout en réduisant le nombre d'évaluations des algorithmes donnant des premières estimations médiocres. Pour un horizon B, UCB sélectionne pour chaque appel à f_{obj} un algorithme à exécuter afin d'avoir une estimation plus précise de sa performance (score moyen des solutions retournées par l'algorithme jusqu'alors). L'algorithme sélectionné est toujours celui qui maximise la valeur $\mathbb{E}_t(A) + \sqrt{\frac{2\ln(t)}{t_A}}$, avec t le budget déjà dépensé, t_A le budget dépensé spécifiquement pour l'algorithme A, et $\mathbb{E}_t(A)$ l'espérance de l'algorithme estimée jusqu'ici. Cette formule favorise les algorithmes dont l'estimation actuelle est élevée (premier terme) et ceux dont la précision est faible (second terme), afin de ne pas délaisser les algorithmes dont la mauvaise évaluation serait due à un nombre d'exécutions trop faible et non représentatives. Une fois que le budget est entièrement dépensé, l'algorithme avec la meilleure espérance est sélectionné.

Sélection SoftMax

SoftMax est une autre stratégie de bandits manchots multi-bras, consistant à sélectionner à chaque itération un bras selon une certaine probabilité. Dans notre cas, la stratégie de bandits détermine (comme pour la solution UCB) le choix de l'algorithme à exécuter afin d'affiner l'estimation de sa performance en calculant la valeur objectif de la solution retournée. Pour un ensemble d'algorithmes et un budget (ou horizon) B, la probabilité d'exécution d'un algorithme $A \in \mathcal{A}$ à l'itération t est $\frac{e^{\mathbb{E}_t(A)\ln(t)}}{\sum_{A\in\mathcal{A}} e^{\mathbb{E}_t(A)\ln(t)}}$. De manière similaire à UCB les algorithmes avec

les meilleures estimations et ceux dont les estimations sont basées sur trop peu d'exécutions ont plus de chances d'être exécutées. Ce mécanisme évite d'allouer trop de budget pour affiner l'estimation de la performance d'un algorithme non pertinent.

Sélection par tournoi

Afin d'économiser au maximum le budget pour pouvoir effectuer une recherche longue avec le plus d'itérations possible, nous avons proposé une stratégie alternative permettant d'allouer un minimum de budget aux algorithmes dont les premières estimations sont faibles. Pour cela, nous utilisons un tournoi à n étapes entre les différents algorithmes. À chaque étape, tous les algorithmes encore en lice sont exécutés m fois. Puis, une fraction d'algorithmes dont l'estimation est la plus élevée est conservée pour la prochaine étape du tournoi. L'estimation des algorithmes étant calculée grâce aux exécutions de l'étape en cours ainsi que des étapes précédentes, celle-ci devient de plus en plus précise. Le dernier algorithme restant en lice est celui qui est sélectionné.

3.4 Évolution d'algorithmes

Dans cette thèse, nous proposons de générer des algorithmes de recherche au moyen d'un processus évolutionnaire appliqué sur un espace de fonctions d'évaluation, tout en fixant l'ensemble des autres spécificités et paramètres de l'algorithme (type d'algorithme, voisinage, heuristique de choix).

Dans un premier temps, nous proposons de faire évoluer une fonction d'évaluation directement basée sur la fonction objectif du problème. La fonction objectif sera donc intégrée à l'algorithme généré, le problème consistant à déterminer une fonction complémentaire permettant de réduire les pièges de la fonction d'origine.

Nous considérerons dans un second temps la problématique de recherche d'une fonction d'évaluation alternative sans lien imposé avec la fonction objectif d'origine. Deux stratégies d'évolution seront proposées dans ce cadre : (1+1)-ES et $(1,\lambda)$-ES.

3.4.1 Fonction complémentaire

Le premier algorithme de génération que l'on présente ici a pour but de générer une fonction s'ajoutant à la fonction objectif du problème afin d'améliorer les solutions obtenues via un hill climber. Cet algorithme a d'abord été développé comme une preuve de concept de l'évolution de fonctions et de la génération d'algorithmes. Il s'agit de déterminer un hill climber guidé par une fonction g de la forme $g(x) = f_{\text{obj}}(x) + f'(x)$ en faisant évoluer la fonction complémentaire

f'. L'objectif est de construire un paysage $(\mathcal{X}, \mathcal{N}, g)$ davantage adapté à un hill climber pour résoudre $(\mathcal{X}, f_{\text{obj}})$, que le paysage naturel $(\mathcal{X}, \mathcal{N}, f_{\text{obj}})$.

L'algorithme 9 détaille la procédure que l'on utilise pour générer une fonction complémentaire qui, additionnée à la fonction objectif, définit une fonction alternative pour naviguer dans l'espace de recherche des solutions. Ensuite, à chaque itération, sans limite prédéfinie de nombre d'itérations, une mutation de la fonction complémentaire est générée. Si la mutation améliore la fonction complémentaire courante au sens de la fonction de sélection ω, alors la mutation devient la nouvelle fonction courante pour la prochaine itération. Enfin, si λ mutations non améliorantes sont proposées successivement, alors la recherche s'arrête et retourne la fonction alternative composée de la fonction objectif et de la dernière fonction complémentaire atteinte (la fonction courante).

Algorithme 9 Évolution d'une fonction complémentaire

Entrée : une fonction objectif f_{obj}, un modèle de fonction $\mathcal{M}$, un paramètre de mutation ξ, une fonction de sélection ω, un nombre de mutations non améliorantes maximum λ
Sortie : une fonction alternative de f_{obj} f_{alt}

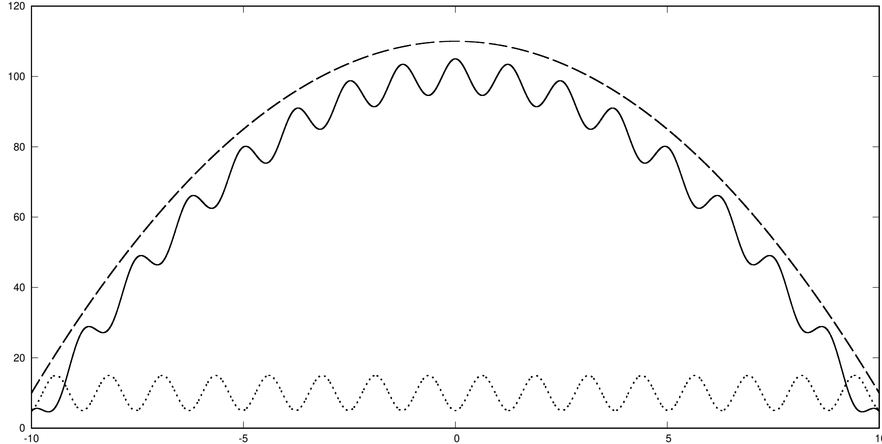
1: $f_{\text{comp}} \leftarrow \text{initialisation}_{\mathcal{M}}()$
2: compteur_mutation $\leftarrow 0$
3: **tant que** compteur_mutation $< \lambda$ **faire**
4: $\quad f'_{\text{comp}} \leftarrow \text{mutation}_{\mathcal{M},\xi}(f_{\text{comp}})$
5: $\quad$ compteur_mutation $\leftarrow$ compteur_mutation+1
6: $\quad$ **si** $\omega(f_{\text{obj}}, \{(f_{\text{comp}} + f_{\text{obj}}), (f'_{\text{comp}} + f_{\text{obj}})\}) = (f'_{\text{comp}} + f_{\text{obj}})$ **alors**
7: $\quad\quad f_{\text{comp}} \leftarrow f'_{\text{comp}}$
8: $\quad\quad$ compteur_mutation$\leftarrow 0$
9: $\quad$ **fin si**
10: **fin tant que**
11: **retourne** $f_{\text{alt}} = f_{\text{obj}} + f_{\text{comp}}$

La figure 3.1 montre, dans un toujours hypothétique cadre continu (plus aisé à visualiser), un exemple de fonction complémentaire permettant de décrire une fonction alternative plus simple à exploiter.

3.4.2 (1+1)-ES

L'évolution de fonctions complémentaires est une première tentative de génération d'algorithmes de recherche locale via un mécanisme d'évolution de fonction d'évaluation, et par extension d'aborder la problématique de résolution du problème initial en explorant un espace de recherche fondamentalement différent (espace des fonctions/des algorithmes, plutôt qu'espace des solutions). Nous proposons maintenant de faire évoluer librement la fonction alternative, sans qu'elle serve à complémenter la fonction objectif initiale. Ainsi, non seulement la structure

FIGURE 3.1 – Représentation visuelle d'un problème d'optimisation continue défini sur l'espace de recherche $\mathcal{X} = [-10, 10]$, et d'une fonction alternative. La ligne continue est le graphe de la fonction objectif $f_{\text{obj}}(x) = -x^2 + 5\cos 5x + 100$ à maximiser. La ligne pointillée (en bas) est le graphe d'une fonction complémentaire $f'(x) = 5\sin 5(x - \pi/2) + 10$. La ligne discontinue (en haut) est la fonction alternative correspondante $g(x) := f_{\text{obj}}(x) + f'(x)$.



de la fonction objectif ne sera pas incluse dans la fonction alternative, mais les climbers guidés par cette fonction ne consommeront pas l'éventuel budget limité d'appels à la fonction objectif.

Cette procédure est présentée dans l'algorithme 10. Celui-ci est une stratégie d'évolution naviguant dans l'espace des fonctions d'évaluation contenues dans le modèle de fonctions $\mathcal{M}$. Cette première configuration (1+1)-ES se base sur un processus évolutionnaire lui-même similaire au hill climber, à la différence près que l'opérateur de mutation évolutionnaire remplace celui de recherche locale reposant sur une fonction de voisinage. On commence d'abord la recherche avec une fonction alternative aléatoire selon le modèle de fonction choisi. Ensuite, à chaque itération, une mutation de la fonction courante est proposée selon le paramètre de mutation ξ. Si la mutation est améliorante au sens de la fonction de sélection ω alors elle devient la fonction courante pour la prochaine itération. Une fois que le budget d'appels à la fonction objectif est dépensé, la dernière fonction courante est retournée.

Les algorithmes 9 et 10 sont très similaires dans les étapes suivies pour générer une fonction. Une différence réside dans la sélection d'une mutation par ω, où à chaque acceptation d'une nouvelle fonction courante le paramètre de mutation est actualisé avec le nombre effectif d'éléments modifiés entre la fonction courante et sa mutation $f_{\text{alt}} \Delta f'_{\text{alt}}$.

Algorithme 10 Stratégie d'évolution (1+1)-ES dans l'espace des fonctions d'évaluation

Entrée : une fonction objectif f_{obj}, un modèle de fonction $\mathcal{M}$, un paramètre de mutation ξ, une fonction de sélection ω, un budget d'appels à f_{obj} B
Sortie : une fonction alternative de f_{obj} f_{alt}
1: $f_{\text{alt}} \leftarrow \text{initialisation}_{\mathcal{M}}()$
2: **tant que** nombre_appels(f_{obj}) $< B$ **faire**
3: $f'_{\text{alt}} \leftarrow \text{mutation}_{\mathcal{M},\xi}(f_{\text{alt}})$
4: **si** $\omega(f_{\text{obj}}, \{f_{\text{alt}}, f'_{\text{alt}}\}) = f'_{\text{alt}}$ **alors**
5: $f_{\text{alt}} \leftarrow f'_{\text{alt}}$
6: $\xi \leftarrow f_{\text{alt}} \Delta f'_{\text{alt}}$
7: **fin si**
8: **fin tant que**
9: **retourne** f_{alt}

3.4.3 (1,λ)-ES

Nous proposons enfin de tester le plein potentiel de notre approche au moyen d'une stratégie d'évolution plus robuste et puissante que la précédente basée sur une simple amélioration itérative des fonctions alternatives.

L'algorithme 11 adapte la stratégie d'évolution (1,λ)-ES à notre problématique, en utilisant les composants présentés précédemment. Le processus de sélection est différent, puisque la stratégie virgule sélectionne le meilleur mutant (parmi λ) même si celui-ci est évalué comme moins bon que l'individu courant (ici la fonction alternative courante). Puisque cette stratégie est non élitiste, une copie de la meilleure fonction évaluée en cours de recherche est gardée en mémoire. Une fois le budget d'appels à la fonction objectif dépensé, la recherche s'arrête et la meilleure fonction rencontrée est retournée. Notons que la mise à jour du paramètre de mutation se fait à chaque itération dans cette stratégie d'évolution plutôt que lorsque la mutation améliore la fonction d'évaluation. De plus, étant donné qu'à chaque itération la fonction courante est remplacée par une mutation, il devient difficile pour la recherche de se bloquer dans une région de l'espace de recherche sous-optimale.

Algorithme 11 Stratégie d'évolution (1,λ)-ES dans l'espace des fonctions d'évaluation

Entrée : une fonction objectif f_{obj}, un modèle de fonction $\mathcal{M}$, un paramètre de mutation ξ, une fonction de sélection ω, un budget d'appels à f_{obj} B, un nombre de mutation par itération λ

Sortie : une fonction alternative de f_{obj} f_{alt}

1: $f_{\text{alt}} \leftarrow \text{initialisation}_{\mathcal{M}}()$
2: $f_{\text{best}} \leftarrow f_{\text{alt}}$
3: **tant que** nombre_appels(f_{obj}) $< B$ **faire**
4:     **pour** $i = 1 \to \lambda$ **faire**
5:         $f'_i \leftarrow \text{mutation}_{\mathcal{M},\xi}(f_{\text{alt}})$
6:     **fin pour**
7:     $\xi \leftarrow f_{\text{alt}} \Delta \omega(f_{\text{obj}}, f')$
8:     $f_{\text{alt}} \leftarrow \omega(f_{\text{obj}}, f')$
9:     $f_{\text{best}} \leftarrow \omega(f_{\text{obj}}, \{f_{\text{alt}}, f_{\text{best}}\})$
10: **fin tant que**
11: **retourne** f_{best}

Chapitre 4

Résultats expérimentaux

Ce chapitre est dédié à l'évaluation des modèles d'algorithmes et de fonctions, des schémas d'évolution et de l'effet des différents paramètres, à travers une succession d'expérimentations.

4.1 Fonctions complémentaires et preuve de concept

Comme spécifié dans le chapitre 3, la génération de fonctions complémentaires doit simplifier le problème de la génération d'algorithmes, tant concernant la difficulté de la génération que d'un point de vue algorithmique. Nous précisons d'abord le protocole expérimental utilisé afin d'obtenir les premiers résultats de génération d'algorithmes qui seront présentés ensuite.

4.1.1 Protocole expérimental

Puisque l'objectif de cette expérience est essentiellement préliminaire afin de mesurer la potentialité de cette approche de génération d'algorithme par évolution de fonction d'évaluation, nous gardons le protocole aussi simple que possible. Nous nous limitons ainsi aux fonctions NK, à la fois pour les instances de problèmes à résoudre et pour le modèle de fonction complémentaire. Les instances (fonctions objectif) utilisées sont 14 fonctions NK, générées à partir des combinaisons de paramètres $N \in \{128, 256\}$ et $K \in \{1, 2, 4, 6, 8, 10, 12\}$. Le modèle de fonction complémentaire utilisera naturellement la taille N de la fonction objectif, mais nous fixons le paramètre K à 1 pour réduire la description des fonctions au maximum et réduire l'espace de recherche des fonctions. Pour générer ces fonctions complémentaires, nous appliquons l'algorithme 9 présenté au chapitre 3. La mutation de fonction suivant le modèle NK nécessite deux paramètres : la probabilité de changer chaque lien, et celle de changer chaque contribution au score. Nous fixons ces paramètres à des valeurs arbitraires que l'on avait déjà testées lors d'essais préliminaires et qui avaient montré des résultats intéressants, à savoir $5/(2N)$ — $N \cdot (K+1) = 2N$ étant la taille de la liste des liens — pour la mutation des liens et 0.05 pour la mutation des

contributions au score. Chacun des éléments de la fonction suivant le modèle NK sera donc modifié avec l'une ou l'autre de ces probabilités. Ensuite, pour sélectionner la nouvelle fonction courante avant la prochaine génération de la stratégie d'évolution de type (1+1), nous utilisons un test de Wilcoxon basé sur 100 appels à la fonction objectif pour chacune des deux fonctions. Les algorithmes étant stochastiques, il est pertinent de réévaluer l'algorithme utilisant la fonction courante (qui a déjà été évalué lors d'au moins la génération précédente) afin de limiter les biais stochastiques qui pourraient bloquer la recherche en cas d'évaluation trop favorable ; bien sûr, des optimisations dans la gestion de ce budget sont envisageables — et seront discutées ultérieurement —, mais ce n'est pas l'objet de cette première expérimentation. Enfin, nous fixons le nombre maximum de mutations non améliorantes consécutives à 1000, critère d'arrêt de l'algorithme.

Table 4.1 – Liste des paramètres et leurs valeurs associées

Paramètre	Valeur utilisée
Fonction objectif	Problème NK
Modèle de fonction complémentaire	Modèle NK
Paramètre de mutation	$(5/(2N),0.05)$
Critère de sélection	Wilcoxon
Critère d'arrêt	1000 échecs successifs

Le tableau 4.1 résume les paramètres nécessaires à l'exécution de l'algorithme de génération de fonctions complémentaires ainsi que les valeurs utilisées pour chacun des paramètres. Comme nous pouvons le voir, tous les paramètres ont des valeurs fixes, donc toutes les ressources peuvent être allouées à la génération de fonctions et non à l'ajustement des paramètres avant l'exécution finale.

Afin de s'assurer de la qualité des fonctions alternatives générées, et donc des hill climbers correspondants, nous comparons ces derniers à des hill climbers guidés par les seules fonctions objectif. Étant donné que les algorithmes que l'on compare sont stochastiques, nous exécutons 100 fois chaque algorithme afin d'avoir une estimation plus précise des performances que l'on peut attendre de ceux-ci. Ainsi, pour la recherche locale guidée par la fonction objectif, les résultats de 100 exécutions sont réduits à leur moyenne. Pour les recherches locales guidées par les fonctions alternatives (une fonction alternative résulte de la somme d'une fonction complémentaire générée et de la fonction objectif), les résultats moyens de 100 algorithmes sont reportés : 10000 recherches locales, chacun des 100 algorithmes générés ayant été rééxécutés 100 fois, postérieurement au processus d'évolution.

4.1.2 Résultats

Le tableau 4.2 montre les résultats obtenus avec le protocole que l'on vient de décrire. La première colonne décrit les différentes instances utilisées pour cette expérimentation. La seconde colonne présente les résultats obtenus par les hill climbers guidés par la fonction objectif. La troisième colonne présente les résultats des recherches locales guidées par les fonctions générées et la fonction objectif. Les valeurs en gras sont les meilleures valeurs pour les instances concernées.

TABLE 4.2 – Comparaison des hill climbers guidés par la fonction objectif seule et avec une fonction complémentaire générée au préalable

Instance	Fonction d'évaluation	
	f_{obj}	$f_{\text{obj}} + f_{\text{comp}}$
$NK_{128,1}$	0.709	**0.723**
$NK_{128,2}$	0.716	**0.745**
$NK_{128,4}$	0.727	**0.765**
$NK_{128,6}$	**0.717**	0.715
$NK_{128,8}$	**0.715**	0.711
$NK_{128,10}$	**0.708**	0.706
$NK_{128,12}$	**0.700**	0.699
$NK_{256,1}$	0.691	**0.705**
$NK_{256,2}$	0.713	**0.735**
$NK_{256,4}$	0.719	**0.729**
$NK_{256,6}$	**0.723**	0.718
$NK_{256,8}$	**0.717**	0.710
$NK_{256,10}$	**0.710**	0.707
$NK_{256,12}$	**0.707**	0.702

Sur le tableau 4.2, nous pouvons observer que pour les instances les moins rugueuses (K≤4) les algorithmes générés sont plus efficaces que l'algorithme initial guidé par la fonction objectif. Pour les instances plus difficiles à résoudre les algorithmes générés obtiennent de moins bon résultats ; cependant, l'écart de performance entre les algorithmes comparés est limité. Il est de prime abord plutôt surprenant de voir que les climbers alternatifs sont plus performants sur des instances plus simples à résoudre par un hill climber utilisant la fonction objectif, et au contraire moins performants sur les instances plus rugueuses, alors qu'au contraire la fonction complémentaire devrait pallier les pièges de la fonction objectif. Comme nous le vérifierons dès la prochaine section où cet écueil sera évité, le fait de combiner une fonction complémentaire de paramètre $K = 1$ avec une fonction objectif de paramètre K très supérieur semble ne pas permettre une complémentarité cohérente, et la stratégie d'évolution utilisée n'a pas non plus été capable de minimiser totalement cette fonction complémentaire en abaissant au maximum ses valeurs de contribution, bien qu'elle s'en soit approchée (les résultats, notamment sur les ins-

tances $N = 128$, sont toujours des proches du climber de référence). Nous considérons néanmoins ces résultats comme très encourageants. La suite des expérimentations est dédiée à la génération indépendante des fonctions alternatives, c'est-à-dire non complémentées par la fonction objectif.

4.2 Fonctions alternatives

4.2.1 Protocole expérimental

À l'instar de la génération de fonctions complémentaires, nous gardons une volonté de conserver un protocole simple. L'objectif est d'abord de proposer un premier algorithme de génération dont les performances sont acceptables.

L'algorithme de génération que l'on utilisera dans ce contexte est l'algorithme 9 présenté dans le chapitre 3 dont la fonction alternative ne sera plus définie comme $f_{\text{alt}} = f_{\text{comp}} + f_{\text{obj}}$ mais seulement comme $f_{\text{alt}} = f_{\text{comp}}$. Toujours pour privilégier la simplicité et donc faciliter la compréhension et l'interprétation des résultats, mais aussi pour conserver les processus qui l'ont rendu efficaces, nous utilisons les mêmes valeurs pour les mêmes paramètres que celles de la génération de fonctions complémentaires (voir tableau 4.1).

Nous travaillons ici encore avec des processus stochastiques. Afin de pouvoir les évaluer correctement nous exécutons plusieurs fois un même algorithme pour calculer une estimation de la qualité des solutions retournées. Dans le cas de l'algorithme de génération, nous générons 100 fonctions alternatives par instance à résoudre. Ensuite, nous effectuons 100 recherches locales (hill climber) guidées par chaque fonction générée. Les 10000 solutions retournées sont alors évaluées par la fonction objectif de l'instance, puis moyennées pour en extraire un score unique. Nous comparons les résultats obtenus d'abord avec le même algorithme de recherche locale de type climber, mais guidé par la fonction objectif. L'évaluation de cet algorithme se présente de la même façon que pour la section précédente. Nous reportons les scores obtenus par l'algorithme de génération de fonctions complémentaires afin de comparer les deux méthodes de génération.

4.2.2 Résultats

Le tableau 4.3 présente les résultats du protocole que l'on vient de décrire. Ce tableau complète en fait le tableau 4.2, en ajoutant la dernière colonne. Pour rappel, la première colonne montre les instances utilisées pour générer de nouvelles fonctions ; la seconde, le score d'un hill climber appliqué à ces instances guidé par la fonction objectif du problème ; la troisième colonne, les résultats obtenus en moyenne par les recherches locales guidées par la fonction objectif et une fonction complémentaire. Enfin, la dernière colonne présente les résultats obtenus en moyenne

par les hill climbers guidés par des fonctions alternatives générées indépendamment de la fonction objectif.

TABLE 4.3 – Comparaison des hill climbers guidés par la fonction objectif seule, avec une fonction complémentaire et par une fonction alternative

Instance	Fonction d'évaluation		
	f_{obj}	$f_{\text{obj}} + f_{\text{comp}}$	f_{alt}
$\text{NK}_{128,1}$	0.709	**0.723**	0.718
$\text{NK}_{128,2}$	0.716	**0.745**	0.726
$\text{NK}_{128,4}$	0.727	**0.765**	0.732
$\text{NK}_{128,6}$	0.717	0.715	**0.728**
$\text{NK}_{128,8}$	0.715	0.711	**0.721**
$\text{NK}_{128,10}$	0.708	0.706	**0.714**
$\text{NK}_{128,12}$	0.700	0.699	**0.708**
$\text{NK}_{256,1}$	0.691	**0.705**	0.703
$\text{NK}_{256,2}$	0.713	**0.735**	0.723
$\text{NK}_{256,4}$	0.719	0.729	**0.731**
$\text{NK}_{256,6}$	0.723	0.718	**0.733**
$\text{NK}_{256,8}$	0.717	0.710	**0.722**
$\text{NK}_{256,10}$	0.710	0.707	**0.720**
$\text{NK}_{256,12}$	0.707	0.702	**0.712**

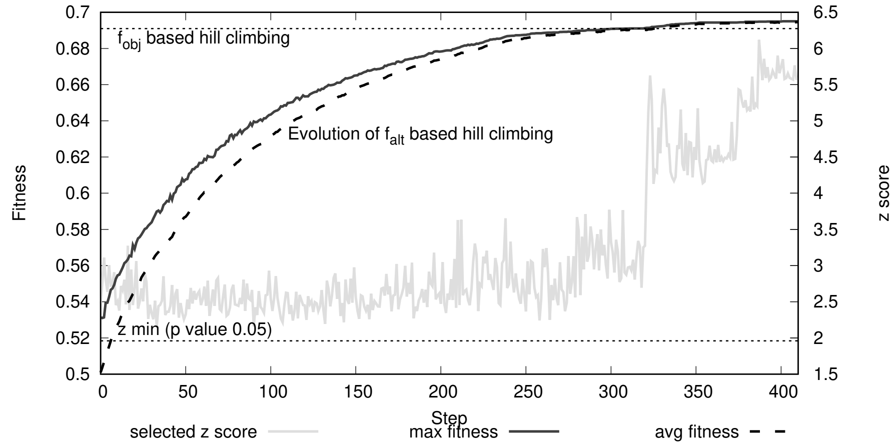
Les valeurs en gras dans le tableau 4.3 sont les meilleurs scores atteints pour une instance donnée. De plus, les valeurs dont la case est grisée sont les scores qui surpassent les scores du hill climber guidé par la fonction objectif. En d'autres termes, ce sont les valeurs qui atteignent l'objectif fixé, à savoir, générer des fonctions d'évaluation qui mènent les recherches locales vers de meilleures solutions que la fonction objectif du problème.

Nous pouvons d'abord remarquer que la génération de fonctions complémentaires garde son avantage sur les instances peu rugueuses. Cependant, ce sont sur les instances difficiles ($K \geq 6$) que s'opèrent les plus grands changements. En effet, la génération de fonctions alternatives indépendantes surpasse les hill climbers de référence, ce que la génération de fonctions complémentaires échouait à atteindre. Enfin, la génération de fonctions alternatives mène vers de meilleures solutions que la fonction objectif et ce pour toutes les instances testées.

Ces valeurs étant des estimations de l'espérance de la performance des différents hill climbers, cela signifie qu'étant donnée une fonction objectif et une fonction alternative générée avec la stratégie d'évolution (1+1) utilisée pour le moment, si une seule recherche locale doit être effectuée, il est probablement plus raisonnable de guider cette recherche par la fonction alternative plutôt que par la fonction objectif pour atteindre la meilleure solution du problème possible.

Afin de comprendre le processus de génération que l'on utilise ici, nous proposons de le repré-

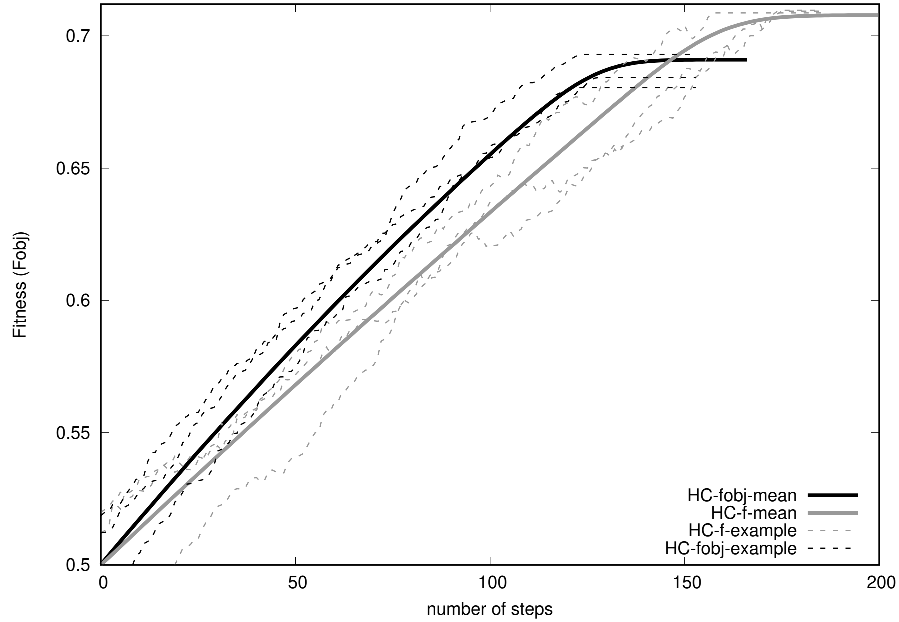
FIGURE 4.1 – Évolution du score de la fonction alternative et du z-score (Wilcoxon) en fonction du nombre d'itération du processus de génération sur l'instance $NK_{256,1}$



senter graphiquement. La figure 4.1 montre l'évolution de la capacité de la fonction alternative à guider l'algorithme de recherche locale vers de bonnes solutions au sens de f_{obj}, en fonction du nombre d'itérations effectuées. Pour évaluer cette évolution, on y ajoute le score moyen obtenu par un hill climber guidé par la fonction objectif. La qualité des solutions renvoyées par la fonction alternative augmente rapidement au début du processus de génération, là où les mutations améliorant la fonction courante sont simples à trouver. En fin de génération, les améliorations sont ténues, mais sont suffisantes pour dépasser le hill climber guidé par f_{obj}. Ensuite, on peut remarquer que les meilleures solutions renvoyées sont toujours supérieures à la moyenne des solutions en début de recherche, mais se confondent en fin de génération. Ceci indique que les optima locaux de f_{alt} se concentrent autour d'une valeur de f_{obj}. Enfin, nous pouvons voir que le z-score reste stable pendant la majorité de la recherche au-dessus de 1.96 (qui est une condition sine qua non), mais augmente à la fin de la recherche. Dans le détail de la génération, sans moyenner plusieurs processus de génération, nous remarquons que le z-score atteint est le maximum possible pour un échantillon de 200 points, c'est-à-dire que la mutation acceptée bat en tout point la fonction courante. Les hypothèses les plus vraisemblables pour expliquer cela est que la mutation proposée possède peu d'optima locaux et que les plus fréquents sont meilleurs que n'importe quel optimum de la fonction courante, ou que la fonction courante renvoie principalement un seul optimum et que la mutation change cet optimum vers une meilleure solution. Ces explications sont cohérentes avec ce que l'on a déjà pu observer du comportement de la fonction alternative en fin de génération, particulièrement avec la conjonction des scores moyens et maximaux.

Pour continuer d'approfondir la compréhension de notre algorithme de génération, nous dé-

FIGURE 4.2 – Exemples et moyennes de recherches locales guidées par la fonction objectif et une fonction alternative sur l'instance $NK_{256,1}$

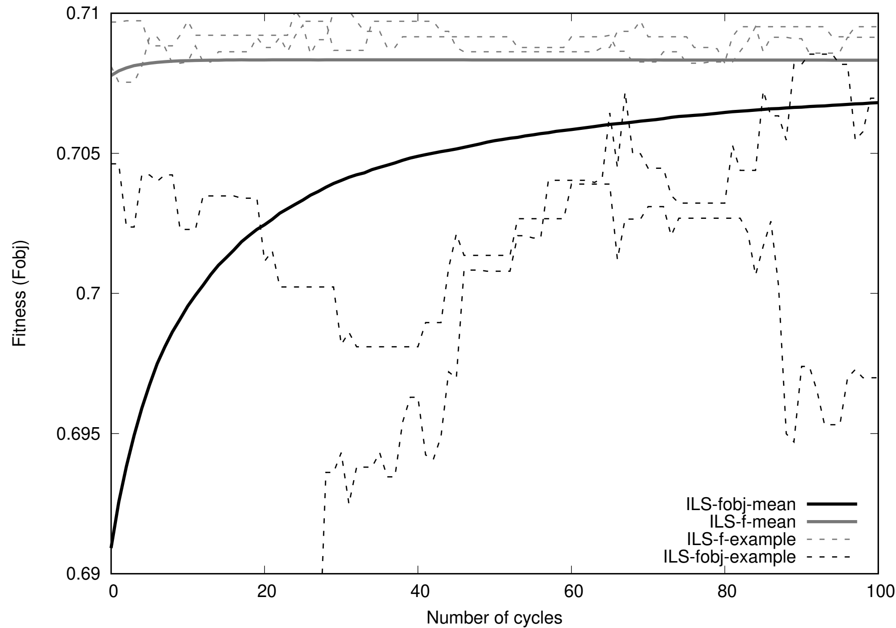


cidons de nous intéresser plus en détail aux algorithmes générés directement. Pour ce faire, nous comparons les recherches locales utilisant la fonction objectif et celles utilisant une fonction alternative générée. La figure 4.2 présente cette comparaison : d'abord avec des exemples d'exécutions uniques de recherches locales guidées par la fonction objectif et la fonction alternative (trois de chaque), puis avec un ensemble d'exécutions moyennées pour en déduire une trajectoire moyenne de la recherche locale selon la fonction d'évaluation utilisée. La fonction alternative exemple utilisée pour construire cette figure a été arbitrairement choisie dans celles générées lors de l'expérience précédente.

Sur cette figure nous pouvons d'abord remarquer que les formes des trajectoires sont similaires, ce qui nous indique que les algorithmes que l'on génère via les fonctions alternatives se comportent en moyenne comme des hill climbers que l'on prend en modèle. Ensuite, les recherches locales guidées par la fonction alternative montent moins rapidement dans le paysage de fitness par rapport à celles guidées par f_{obj}, mais ont un plus grand nombre d'itérations et atteignent finalement de meilleures solutions. En ce sens, ces hill climbers se comportent de manière assez similaire à des hill climbers basés sur la règle pivot du pire améliorant [37]. On doit noter que les exécutions de recherches locales guidées par la fonction alternative ne se comportent pas nécessairement comme des hill climbers dans le paysage de fitness initial. En effet, sur les exemples de trajectoires de f_{alt} certaines itérations détériorent la solution au sens

de f_{obj}, donc ne sont pas des trajectoires empruntables par un hill climber guidé par f_{obj}. Les possibilités offertes par la génération de fonctions alternatives — les trajectoires empruntables par un hill climber ne le sont pas nécessairement par un autre — et la sélection progressive des ensembles de trajectoires les plus pertinents, permettent aux algorithmes de ne pas se bloquer dans les optima sous-optimaux du paysage de fitness initial.

FIGURE 4.3 – Exemples et moyenne de recherches locales itérées guidées par la fonction objectif et une fonction alternative sur l'instance $NK_{256,1}$



Enfin, pour utiliser au maximum l'information contenue dans les fonctions alternatives, nous proposons maintenant de les employer dans un contexte de recherche locale itérée (ILS). Comme nous l'avons vu dans la section 1.2.3 du chapitre 1, les ILS consistent en une alternance d'exploitation de la fonction d'évaluation au moyen d'un hill climber et d'exploration via des mouvements aléatoires. Cet algorithme, du point de vue du paysage de fitness, visite alors les optima locaux à tour de rôle sans pour autant avoir à réinitialiser intégralement la recherche. Sur un temps assez long, cet algorithme visitera tous les optima sous condition que la stratégie de perturbation soit en adéquation avec les propriétés du paysage [46, 94].

La figure 4.3 compare les trajectoires d'ILS obtenues via f_{obj} et f_{alt}. On peut d'abord remarquer qu'en 100 cycles d'ILS sur la fonction objectif, nous n'atteignons pas les résultats obtenus par la fonction alternative avec un seul cycle. Cependant cette efficacité va de pair avec une très grande difficulté pour cette dernière d'améliorer les solutions retournées. En effet, le score en début de recherche est sensiblement le même qu'en fin de recherche. Néanmoins, ce résultat était

à prévoir à la lumière de ce que l'on a pu comprendre de la génération d'algorithmes jusque-là, notamment sur le nombre d'optima locaux des fonctions alternatives. En effet, un faible nombre de ces solutions implique une plus grande efficacité pour atteindre les meilleurs optima, mais aussi une exploration plus difficile voire sans intérêt particulier dans un contexte comme celui des ILS.

Ces différentes expériences et résultats ont été publiés dans [95, 96].

4.3 Génération d'algorithmes stochastiques

Dans cette nouvelle section, nous continuons avec l'expérimentation de génération de recherches locales stochastiques. Cependant, contrairement aux parties précédentes, nous cherchons à générer les meilleurs algorithmes possibles selon les connaissances que l'on a pu acquérir jusqu'ici. L'axe d'amélioration que l'on explore ici est l'algorithme de génération lui-même. Comme pour les sections précédentes, nous présentons d'abord le protocole expérimental utilisé, puis les résultats obtenus.

4.3.1 Protocole expérimental

Afin de mieux juger de l'efficacité de la stratégie d'évolution d'algorithmes, nous étendons notre ensemble d'instances à des tailles d'instances plus grandes : $N \in \{128, 256, 512, 1024\}$ pour la taille des solutions et toujours $K \in \{1, 2, 4, 6, 8, 10, 12\}$ pour le paramètre de rugosité.

Nous appliquons désormais les deux stratégies d'évolution pour échantillonner l'espace de recherche des fonctions d'évaluation décrites au chapitre 3 : (1+1)-ES (algorithme 10) et (1,λ)-ES (algorithme 11). Rappelons que le premier algorithme applique la même stratégie que celui utilisé dans la section précédente (algorithme 9), les seules différences se situant dans le critère d'arrêt (un budget sera alloué) et dans le caractère adaptatif de l'opérateur de mutation. La stratégie d'évolution $(1, \lambda)$-ES possède un paramètre propre λ qui contrôle le nombre de mutants en compétition à chaque itération de la génération. Ce paramètre peut être dépendant de l'instance et a une grande influence sur l'efficacité de la stratégie ; nous choisissons de le paramétrer à partir du résultat de courtes exécutions. Pour chaque instance, l'algorithme de génération est exécuté 10 fois pour chaque valeur possible de λ issues de l'ensemble $\Lambda = \{10k : k \in [\![1; 55]\!]\}$; la valeur de λ qui obtiendra les meilleures fonctions d'évaluation sera sélectionnée pour les exécutions longues.

Hormis le paramètre λ de la stratégie d'évolution $(1, \lambda)$, les paramètres nécessaires pour les deux algorithmes sont les mêmes et seront fixés de manière identique pour faciliter l'interprétation des résultats. Le modèle de fonction utilisé précédemment (fonction NK avec $K = 1$) est

pour le moment conservé. Les probabilités de mutation de chaque élément décrivant une fonction NK (référence à un lien ou contribution au score) sont initialisées au paramètre minimal : $\frac{d_1}{2N}$ et $\frac{d_2}{N \cdot 2^{K+1}}$, avec $d_1 = d_2 = 1$ — il y a respectivement $2N$ références et $N \cdot 2^(K+1)$ et contributions dans la description d'une fonction NK. Les paramètres d_1 et d_2 seront mis à jour de manière adaptative à chaque génération selon le nombre de modifications effectives ayant été effectuées pour générer le mutant sélectionné. La sélection ne sera plus déterminée au moyen d'un test de Wilcoxon mais à partir d'une simple estimation de la performance des algorithmes établie en moyennant les évaluations de chaque exécution de climber. Deux raisons motivent ce choix ; tout d'abord, le test de Wilcoxon permet de discriminer deux algorithmes, ce qui le rend inadapté pour la stratégie d'évolution $(1, \lambda)$; ensuite et comme nous le verrons, les résultats obtenus en utilisant une simple sélection par moyenne sont similaires à ceux basés sur le test de Wilcoxon, alors même que l'emploi du test nécessite un calcul plus complexe. Les moyennes seront calculées sur un ensemble de 100 solutions pour chaque fonction soumise à la sélection ; l'évaluation d'une fonction d'évaluation mutante sera donc la moyenne des valeurs objectif de 100 optimums locaux non nécessairement distincts obtenus au moyen d'un hill climber guidé par cette fonction. Enfin, nous fixons un budget d'appels à la fonction objectif volontairement conséquent (10^8) de manière à laisser l'algorithme de génération converger suffisamment dans l'espace des fonctions. Toutes ces valeurs de paramètres sont rappelées dans le tableau 4.4.

TABLE 4.4 – Liste des paramètres et leurs valeurs associées

Paramètre	Valeur utilisée
Fonction objectif	Problème NK
Modèle de fonction alternative	Modèle NK
Paramètre de mutation	Adaptatif
Critère de sélection	Moyenne uniforme (100)
Budget d'appels à f_{obj}	10^8
λ	$\{10k : k \in [\![1; 55]\!]\}$

Pour chaque instance, les deux stratégies d'évolution sont exécutées 100 fois et fournissent donc 100 fonctions d'évaluation définissant autant d'algorithmes stochastiques de hill climbing. Pour évaluer une stratégie, nous reportons deux valeurs : une estimation de l'espérance de la performance des hill climbers basés sur les fonctions générées (HC), et celle des hill climbers itérés correspondants (RRHC pour *Random Restart Hill Climber*, soit à redémarrage aléatoire). Soit $\mathcal{F} = \{f_1, \ldots, f_{100}\}$ l'ensemble des fonctions retournées lors des 100 exécutions indépendantes d'un algorithme donné sur une instance donnée. Nous estimons le score de chaque fonction f_i, et donc chaque de hill climber, en moyennant la valeur objectif de 100 solutions $x_{i,1}$, …, $x_{i,100}$ retournés par ces derniers (HC), mais aussi en déterminant la valeur objectif maximale (RRHC). Les indicateurs de performance des différentes stratégies s'obtiennent en moyennant

ces valeurs sur les 100 fonctions générées : $\frac{1}{10000}\sum_{i=1}^{100}\sum_{j=1}^{100} f_{\text{obj}}(x_{ij})$ pour la mesure HC, et $\frac{1}{100}\sum_{i=1}^{100}\max_{j=1}^{100} f_{\text{obj}}(x_{ij})$ pour RRHC.

Pour avoir un point de référence en termes de scores atteints, nous comparons les résultats que venons de décrire avec des hill climbers guidés par la fonction objectif. Pour comparer ces hill climbers à l'espérance des solutions renvoyées par les algorithmes générés (mesure HC) nous calculons l'espérance du hill climber guidé par f_{obj}. Nous l'estimons via 10000 exécutions indépendantes de l'algorithme et en faisant la moyenne des scores des solutions renvoyées.

Nous ne perdons finalement pas de vue que dans ce contexte de sur-apprentissage volontaire à une unique instance, la stratégie d'évolution permettant de générer une fonction fait office d'algorithme de résolution. Bien que dans cette expérimentation nous ne la traitons pas comme l'algorithme de recherche à évaluer — la meilleure solution rencontrée au cours du processus n'est pas retournée et l'algorithme généré est réexécuté de manière indépendante pour être évaluation —, nous ajoutons un critère de comparaison secondaire qui est la performance d'un hill climber itéré dans le paysage initial, avec un budget de 10^8 évaluations soit le budget utilisé par la stratégie d'évolution. La valeur reportée est la moyenne sur 100 exécutions des meilleurs optimums rencontrés après une succession de hill climbers effectuée jusqu'à épuisement du budget.

4.3.2 Résultats

Le tableau 4.5 présente les résultats des algorithmes de génération 10 et 11, ainsi que ceux des algorithmes de référence que l'on utilise dans le protocole que l'on a décrit dans la section précédente. La première colonne liste les instances sur lesquelles nous travaillons. Les deuxième et troisième colonnes présentent les résultats obtenus par les algorithmes de référence sur les instances tout juste mentionnées. D'abord, les résultats obtenus en moyenne par un hill climber guidé par la fonction objectif, soit l'espérance de cet algorithme (HC) ; ensuite, ceux du hill climber à redémarrage aléatoire (RRHC) dont le budget d'appels à f_{obj} est le même que celui des algorithmes de génération pour les comparer en tant qu'algorithmes d'optimisation. Les quatre prochaines colonnes présentent les résultats obtenus par les hill climbers générés via les deux algorithmes de génération que l'on étudie ici : d'abord, ceux de la stratégie d'évolution $(1+1)$, puis de celle $(1, \lambda)$. Pour chacune on reporte d'abord l'espérance des hill climbers générés, puis les maximums attendus (RRHC, avec 100 points de départ). Enfin, la dernière colonne donne les résultats de l'expérience préliminaire dont le but était de fixer la valeur du dernier paramètre de la stratégie d'évolution $(1, \lambda)$. Ces valeurs ont été utilisées lors de la génération des hill climbers dont les résultats sont présentés dans les colonnes précédentes. Afin de faciliter la comparaison entre les différentes espérances et maximums des algorithmes de référence et générés nous reportons en gras les meilleures valeurs obtenues à la fois pour le critère HC et

pour le critère RRHC.

Grâce au tableau 4.5 nous pouvons voir que les hill climbers générés par la stratégie d'évolution $(1, \lambda)$ sont en général plus efficaces que les autres. Notamment en moyenne où sur toutes les instances testées ces hill climbers trouvent toujours de meilleures solutions. Il est cependant vrai qu'en utilisant un budget suffisamment important — celui qui a été alloué pour la génération d'une fonction d'évaluation alternative —, le hill climber itéré est parfois plus efficace en tant qu'algorithme d'optimisation. Précisons toutefois que nous ne reportons pas dans cette expérimentation dédiée à la génération de hill climbers, les meilleures solutions évaluées au cours du processus de génération de même budget et qui permettraient de surpasser les résultats du hill climber itéré. Nous nous intéressons ici à l'existence d'algorithmes simples permettant de résoudre des problèmes plus complexes, et à la manière de les générer, plutôt qu'à la recherche d'un algorithme de recherche plus sophistiqué qui naturellement dominerait une recherche locale basique comme étudiée ici. Nous pouvons toutefois noter, pour les instances de type $K \geq 8$, de moins bons ($N = 512$) voire très mauvais ($N = 1024$) résultats pour les fonctions d'évaluations générées par la stratégie $(1 + 1)$. Nous étudierons plus précisément cette exception dans le chapitre suivant propre aux analyses comportementales.

Si nous avons vu qu'il était possible de générer des fonctions d'évaluation alternatives permettant de meilleures trajectoires dans l'espace de recherche qu'au moyen de la fonction objectif, l'une des principales problématiques du processus de génération reste la grande quantité de ressources allouées pour l'évaluation des algorithmes stochastiques, soit 200 appels à f_{obj} par fonction mutante pour la stratégie d'évolution $(1+1)$ et 100λ pour $(1, \lambda)$-ES. Afin de tester une réduction à l'extrême de ces coûts, mais aussi d'expérimenter un autre moyen de générer des algorithmes alternatifs, nous discutons dans la prochaine section de la génération d'algorithmes déterministes, plus simples à évaluer et comparer.

Les résultats présentés dans cette section ont été publiés dans [97].

4.4 Génération d'algorithmes déterministes

Comme nous venons de le voir, cette section est dédiée à la génération d'algorithmes plus simples à évaluer, et donc à sélectionner dans un contexte de processus évolutionnaire. En effet, en réduisant le budget nécessaire à la sélection d'algorithmes nous pourrons effectuer plus d'itérations de génération pour un budget constant, et donc atteindre de meilleurs algorithmes d'optimisation et donc de meilleures solutions. Néanmoins, si les algorithmes que l'on va générer seront plus efficaces de par leur simplicité, nous pouvons nous attendre à des hill climbers si spécialisés qu'ils seront indissociables de leur contexte. Ceci définit une première limite de cette approche d'un point de vue de la génération mais pas en termes d'optimisation. Comme pour

Table 4.5 – Performance des stratégies d'évolution, et des hill climbers générés

Instance	HC	$RRCH_{10^8}$	(1+1)-ES		(1,λ)-ES		λ
			HC	RRHC	HC	RRHC	
$NK_{128,1}$	0.701	0.720	0.716	0.717	**0.724**	**0.725**	20
$NK_{128,2}$	0.707	0.735	0.718	0.719	**0.737**	**0.739**	20
$NK_{128,4}$	0.728	0.783	0.738	0.739	**0.777**	**0.786**	10
$NK_{128,6}$	0.717	**0.777**	0.729	0.730	**0.768**	0.770	20
$NK_{128,8}$	0.718	**0.781**	0.724	0.725	**0.763**	0.766	20
$NK_{128,10}$	0.709	**0.773**	0.718	0.720	**0.756**	0.761	30
$NK_{128,12}$	0.699	**0.764**	0.706	0.710	**0.742**	0.746	40
$NK_{256,1}$	0.707	0.718	0.709	0.710	**0.720**	**0.720**	20
$NK_{256,2}$	0.700	0.729	0.715	0.716	**0.735**	**0.736**	20
$NK_{256,4}$	0.725	0.766	0.733	0.734	**0.766**	**0.767**	20
$NK_{256,6}$	0.725	**0.771**	0.732	0.733	**0.769**	0.770	20
$NK_{256,8}$	0.717	**0.761**	0.723	0.724	**0.757**	0.759	20
$NK_{256,10}$	0.711	**0.755**	0.715	0.717	**0.751**	0.754	20
$NK_{256,12}$	0.705	**0.749**	0.709	0.710	**0.742**	0.744	20
$NK_{512,1}$	0.688	0.697	0.694	0.695	**0.705**	**0.705**	20
$NK_{512,2}$	0.715	0.733	0.718	0.718	**0.739**	**0.739**	20
$NK_{512,4}$	0.721	0.749	0.725	0.725	**0.755**	**0.756**	20
$NK_{512,6}$	0.722	0.753	0.725	0.726	**0.758**	**0.759**	20
$NK_{512,8}$	0.720	0.751	0.710	0.713	**0.754**	**0.755**	20
$NK_{512,10}$	0.716	0.746	0.697	0.701	**0.746**	**0.748**	20
$NK_{512,12}$	0.708	0.739	0.651	0.661	**0.738**	**0.740**	20
$NK_{1024,1}$	0.696	0.705	0.685	0.687	**0.713**	**0.713**	20
$NK_{1024,2}$	0.714	0.729	0.691	0.695	**0.736**	**0.736**	30
$NK_{1024,4}$	0.725	0.745	0.651	0.660	**0.756**	**0.758**	20
$NK_{1024,6}$	0.724	0.745	0.601	0.616	**0.755**	**0.756**	20
$NK_{1024,8}$	0.721	0.743	0.558	0.558	**0.750**	**0.752**	20
$NK_{1024,10}$	0.716	**0.737**	0.528	0.551	**0.730**	0.731	50
$NK_{1024,12}$	0.711	**0.733**	0.516	0.538	**0.724**	0.725	40

les sections précédentes nous commençons par décrire le protocole de l'expérimentation que l'on présente ici, puis les résultats associés.

4.4.1 Protocole expérimental

Jusqu'ici, nous avons généré des hill climbers stochastiques comme nous les avons présentés dans la section 3.1.1. La solution de départ du processus d'optimisation était choisie aléatoirement, puis les voisins de la solution courante étaient aussi présentés dans un ordre aléatoire. Ces éléments stochastiques sont la source de la difficulté d'évaluation des algorithmes, puisqu'une seule exécution de chaque algorithme ne permet pas de les comparer avec une confiance suffisante. Les résultats sont trop aléatoires pour définir si une évaluation est favorable pour un algorithme en raison de sa qualité ou simplement d'une exécution chanceuse ; et ce, même en employant la même solution de départ.

Afin de simplifier le processus de sélection et de supprimer ce biais stochastique, critique dans notre contexte de comparaison d'algorithmes, nous décidons d'ôter la composante aléatoire dans les algorithmes que nous générons. Il s'agit donc d'utiliser à présent un modèle d'algorithme déterministe tel que présenté dans la section 3.1.2. Dans ce cas de figure, chaque fonction d'évaluation donnée est en correspondance avec une unique solution. Un hill climber déterministe A retourne ainsi toujours la même solution et peut ainsi être évalué de manière exacte au moyen d'une unique exécution, y compris si la fonction d'évaluation sous-jacente possède de nombreux optima locaux. On peut alors assimiler un tel hill climber à une fonction HC : $\mathcal{F} \to \mathcal{X}$ faisant correspondre l'espace des fonctions à l'espace des solutions. Il s'agit donc d'une transformation du problème d'optimisation initial, où la recherche de la meilleure solution s'effectue au moyen d'un processus évolutionnaire dans l'espace des fonctions d'évaluation plutôt que dans l'espace des solutions.

Pour tester cette approche, nous utilisons le même ensemble d'instances que dans l'expérience précédente afin de pouvoir comparer simplement les résultats obtenus entre les algorithmes stochastiques et déterministes. De même, nous utilisons les mêmes algorithmes de génération, à savoir les stratégies d'évolution (1+1) et (1,λ), pour les mêmes raisons.

Les différents paramètres utilisés sont indiqués dans le tableau 4.6. Comme précédemment, nous appliquons notre schéma de régulation auto-adaptative du paramètre de mutation, initialisé avec les valeurs les plus basses permises. Le budget disponible est inchangé (10^8 appels à la fonction objectif f_{obj}). La seule différence réside dans le processus de sélection, puisque le fait de générer des algorithmes déterministes permet d'évaluer une seule fois chaque algorithme/fonction et donc d'effectuer davantage de générations pour un budget identique. Concernant enfin le choix du paramètre λ, nous le déterminons pour chaque instance au moyen du protocole évoqué

Table 4.6 – Liste des paramètres et leurs valeurs associées

Paramètre	Valeur utilisée
Fonction objectif	Problème NK
Modèle fonction	Modèle NK
Mutation	(1,1)
Fonction de sélection	Sélection déterministe
Budget d'appels à f_{obj}	10^8
λ	$\{10k \vert k \in [\![1;55]\!]\}$

précédemment : pour chaque $\lambda \in \{10k : k \in [\![1;55]\!]\}$, l'algorithme de génération est exécuté 10 fois avec un budget limité de 2×10^6. La valeur de λ qui donne les meilleures solutions en moyenne est alors utilisée pour les exécutions plus longues.

Pour évaluer les algorithmes d'évolution de hill climbers déterministes, sachant que la génération reste stochastique, nous exécutons 100 fois chaque algorithme sur chaque instance avec les valeurs que nous venons de définir. Nous reportons alors la moyenne des meilleurs hill climbers générés. De par la nature des algorithmes générés, il n'y a pas lieu de considérer les performances de hill climbers itérés ici, puisque la solution de départ est partie intégrante de l'algorithme et que la trajectoire empruntée ensuite est déterminée de manière déterministe.

Enfin, nous utilisons des algorithmes d'optimisation classiques comme référence pour juger de l'efficacité des algorithmes de génération comme algorithmes d'optimisation. D'abord, nous reportons les valeurs déjà utilisées dans la section précédente, c'est-à-dire l'espérance des hill climbers guidés par la fonction objectif, ainsi que celle d'un hill climber itéré à redémarrage aléatoire. À ces algorithmes nous ajoutons les résultats de l'algorithme *Sampled Walk* (SW), une recherche locale plus efficace (autorisant les mouvements détériorants) ayant permis d'établir des premières bornes inférieures sur ces mêmes jeux d'instances [58]. Cet algorithme reprend le mécanisme d'une stratégie d'évolution $(1, \lambda)$ dans un contexte de recherche locale (les mutants sont des voisins). Outre la possibilité de comparer les résultats obtenus par nos climbers avec un algorithme de recherche plus sophistiqué, nous pouvons ici comparer deux stratégies d'évolution (1,λ), l'une (SW) s'appliquant classiquement sur l'espace des solutions (ES) et l'autre sur l'espace des fonction d'évaluation (EFE). L'algorithme SW suit les mêmes règles de paramétrage et d'exécution que celles que nous venons de décrire pour les algorithmes de génération, à savoir le choix du paramètre λ après des exécutions courtes.

4.4.2 Résultats

Le tableau 4.7 présente les résultats obtenus par les algorithmes 10 et 11 appliqués aux hill climbers déterministes (cf. section 3.1.2). La première colonne présente les instances utilisées. Les deuxième et troisièmes colonnes sont les résultats obtenus respectivement par un hill climber et un hill climber itéré guidé par f_{obj}, la valeur reportée étant l'espérance de ces algorithmes sur l'instance concernée. La quatrième colonne fait état des résultats obtenus dans l'espace des solutions (ES) par l'algorithme SW, recherche locale dont les trajectoires sont établies selon les mêmes règles qu'une stratégie d'évolution $(1, \lambda)$, et la cinquième la valeur de λ utilisée et fixée au préalable grâce à un protocole dédié. Les sixième et septième colonnes sont quant à elles les résultats obtenus par les algorithmes de génération (EFE) pour chaque instance. Enfin, la dernière colonne est la valeur de λ utilisée pour l'algorithme $(1, \lambda)$-ES dans l'espace des fonctions d'évaluation. Les valeurs en gras sont les meilleures trouvées pour l'instance concernée à travers l'ensemble des algorithmes utilisés.

Tout d'abord, nous pouvons voir qu'une recherche locale appropriée dans l'espace des solutions (SW) reste la stratégie la plus performante. Néanmoins, nos climbers déterministes générés au moyen d'une stratégie similaire dans l'espace des fonctions d'évaluation obtiennent les mêmes résultats sur les instances les plus simples. De plus, sur le reste des instances la différence entre ces deux approches est souvent réduite (toujours à moins de 0.01) ce qui nous apparaît raisonnable étant donné l'objectif de cette étude : générer un hill climber déterministe capable d'atteindre une bonne solution. Or ici, les solutions retournées sont bien meilleures que les meilleures solutions atteintes par un hill climber stochastique itéré appliqué sur la fonction objectif (colonne RRHC_{10^8}) avec un budget d'appels à la fonction objectif équivalent à celui ayant permis de générer le climber déterministe.

Ensuite, nous pouvons comparer ces résultats aux résultats obtenus lors de la génération d'algorithmes stochastiques. Les performances des algorithmes déterministes sont supérieures, ce qui montre le gain apporté par l'économie de budget dépensé par génération, et indique que simplifier ou contraindre encore le modèle d'algorithme ne freine pas ses capacités potentielles.

Les résultats présentés dans cette section ont été publiés dans [98, 99].

4.5 Sélection d'algorithmes stochastiques

Comme nous l'avons déjà mentionné plusieurs fois jusqu'ici, le problème de sélection est central dans un contexte évolutionnaire. La problématique de la sélection d'algorithmes stochastiques dans un processus où le budget est une ressource cruciale est particulièrement sensible. Renforcer la précision de l'estimation d'un algorithme requiert de l'exécuter un très grand nombre

TABLE 4.7 – Scores moyens des meilleures solutions atteintes lors de la génération d'algorithmes déterministes, comparés aux résultats de recherches locales classiques

Instance	HC	$RRHC_{10^8}$	$(1,\lambda)$-ES [ES]		(1+1)-ES	$(1,\lambda)$-ES [EFE]	
			SW_{10^8}	λ	[EFE]		λ
$NK_{128,1}$	0.701	0.720	**0.725**	30	0.723	**0.725**	100
$NK_{128,2}$	0.707	0.735	**0.742**	50	0.730	**0.742**	110
$NK_{128,4}$	0.728	0.783	**0.795**	40	0.756	**0.795**	90
$NK_{128,6}$	0.717	0.777	**0.799**	60	0.742	0.795	80
$NK_{128,8}$	0.718	0.781	**0.801**	70	0.738	0.795	120
$NK_{128,10}$	0.709	0.773	**0.792**	90	0.730	0.786	150
$NK_{128,12}$	0.699	0.764	**0.781**	100	0.722	0.774	190
$NK_{256,1}$	0.707	0.718	**0.722**	50	0.716	**0.722**	90
$NK_{256,2}$	0.700	0.729	**0.744**	50	0.727	**0.744**	120
$NK_{256,4}$	0.725	0.766	**0.793**	50	0.745	0.789	80
$NK_{256,6}$	0.725	0.771	**0.803**	50	0.745	0.799	110
$NK_{256,8}$	0.717	0.761	**0.795**	90	0.734	0.788	140
$NK_{256,10}$	0.711	0.755	**0.785**	100	0.728	0.780	180
$NK_{256,12}$	0.705	0.749	**0.774**	140	0.720	0.768	200
$NK_{512,1}$	0.688	0.697	**0.709**	80	0.702	0.708	90
$NK_{512,2}$	0.715	0.733	**0.751**	50	0.730	0.750	90
$NK_{512,4}$	0.721	0.749	**0.786**	50	0.738	0.780	90
$NK_{512,6}$	0.722	0.753	**0.797**	60	0.740	0.790	140
$NK_{512,8}$	0.720	0.751	**0.793**	100	0.733	0.783	150
$NK_{512,10}$	0.716	0.746	**0.781**	120	0.727	0.775	260
$NK_{512,12}$	0.708	0.739	**0.771**	160	0.718	0.763	260
$NK_{1024,1}$	0.696	0.705	**0.717**	50	0.708	0.716	90
$NK_{1024,2}$	0.714	0.729	**0.751**	40	0.730	0.750	90
$NK_{1024,4}$	0.725	0.745	**0.787**	60	0.738	0.782	120
$NK_{1024,6}$	0.724	0.745	**0.794**	80	0.737	0.786	150
$NK_{1024,8}$	0.721	0.743	**0.789**	110	0.732	0.779	200
$NK_{1024,10}$	0.716	0.737	**0.776**	210	0.724	0.770	250
$NK_{1024,12}$	0.711	0.733	**0.768**	180	0.718	0.759	300

de fois, ce qui devient vite inadapté en termes de temps de calcul car le nombre d'algorithmes à évaluer au cours du processus évolutionnaire doit également être suffisant. Chaque exécution supplémentaire réduit d'autant le budget pour la génération tout en ayant un poids plus faible dans l'évaluation par rapport aux estimations établies à partir des premières exécutions. Ainsi, la problématique à laquelle nous faisons face est celle d'établir la méthodologie de sélection permettant le meilleur rapport entre précision de sélection et nombre d'exécutions nécessaire. Pour ce faire, nous explorons dans cette section différentes stratégies, les comparons pour finalement choisir la plus appropriée pour l'étude étendue présentée dans la prochaine section.

4.5.1 Méthodes de sélection

L'objectif de cette section est de définir un sous-ensemble de méthodes de sélection d'algorithmes stochastiques. Ce sous-ensemble sera alors à la base d'une analyse plus approfondie afin de déterminer laquelle utiliser dans l'expérimentation étendue. Pour ce faire, nous proposons d'abord quatre catégories d'opérateurs :

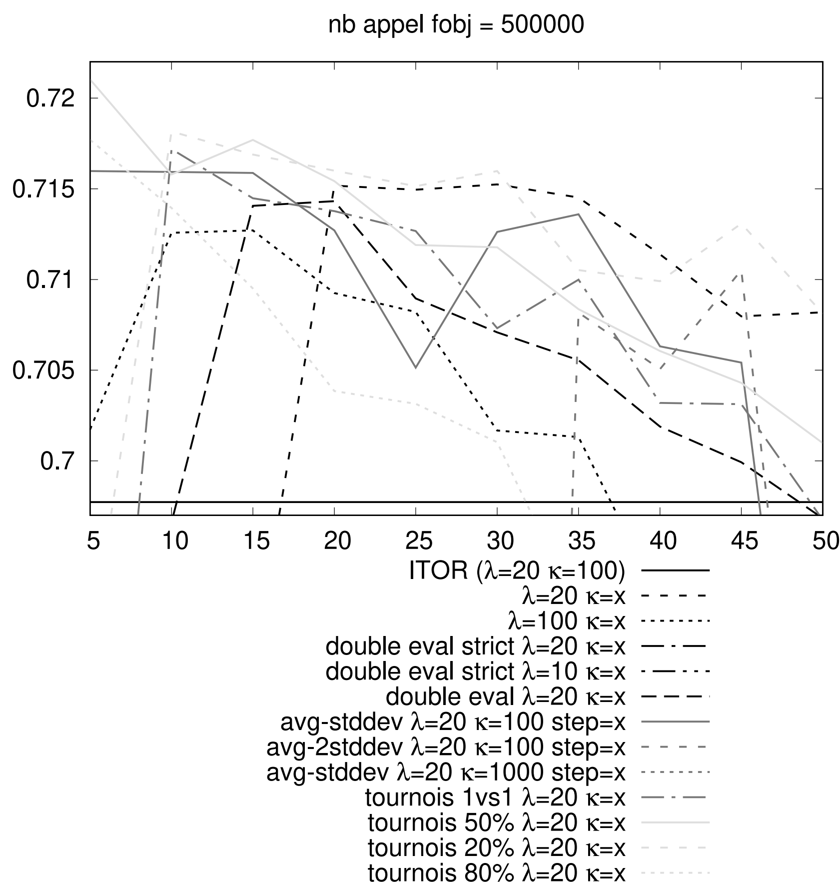
- Une sélection par la moyenne dont le budget est réparti uniformément (cf. section 3.3.2).
- Une sélection par la moyenne répétée afin d'obtenir des résultats plus robustes.
- Une sélection par la moyenne et l'écart type.
- Une sélection par tournois, méthode populaire dans les approches par population[72].

La figure 4.4 présente les résultats obtenus par plusieurs variations de ces catégories. Il s'agit plus précisément d'une comparaison restreinte aux meilleurs résultats obtenus ; nous fournissons en annexe (figure A1) une vue complète de l'étude. Les différentes méthodes de sélection sont testées sur l'instance $NK_{128,1}$ et appliquées à la stratégie (1,λ)-ES avec un budget total de 5×10^5 d'appels à la fonction objectif. L'axe des abscisses donne la valeur du paramétrage utilisé, cette valeur est représentée par un "$\times$" dans la légende de la figure. L'axe des ordonnées représente le score obtenu en moyenne lors de 10 exécutions de l'algorithme pour une valeur de paramètre donnée.

Ensuite, par ordre d'apparition dans la légende les méthodes de sélection sont :

- ITOR ($\lambda = 20, \kappa = 100$) : Cette méthode de sélection est celle utilisée dans la section 4.3, présentée dans la section 3.3.2 et décrite dans notre article publié dans la revue ITOR [97] (*International Transactions in Operational Research*). Chaque algorithme est exécuté 100 fois ; le meilleur algorithme en moyenne est alors sélectionné. C'est donc le minimum à atteindre pour les nouvelles méthodes de sélection pour améliorer le processus de génération en général.

FIGURE 4.4 – Comparaison de 12 fonctions de sélection paramétrables avec celle utilisée en section 4.3 sur l'instance $NK_{128,1}$



— $\lambda = 20, \kappa = x$: Cette méthode est identique à la précédente, avec des nombres d'exécutions plus faible pour chaque algorithme. Le nombre d'exécutions est donné par le paramètre κ. Différentes valeurs entre 5 et 50 (axe des abscisses) ont été testées. Rappelons que diviser de moitié le nombre d'exécutions de chaque algorithme permet de doubler le nombre de générations de la stratégie d'évolution, qui dispose toujours du même budget global d'appels à la fonction objectif.

— $\lambda = 100, \kappa = x$: Méthode identique appliquée sur une stratégie d'évolution paramétrée différemment, avec 100 mutants générés à chaque génération plutôt que 20. On compare également 10 valeurs de κ, entre 5 et 50.

— double eval strict $\lambda = 20, \kappa = x$: Cette méthode procède par évaluations successives et indépendantes de x exécutions de chacun des λ algorithmes, jusqu'à ce qu'un algorithme domine deux évaluations consécutives.

- double eval strict $\lambda = 10, \kappa = x$: Même méthode avec un paramètre λ différent.
- double eval $\lambda = 20, \kappa = x$: Procédure identique en ignorant l'aspect consécutif des deux évaluations : on itère x évaluations des λ mutants, jusqu'à ce qu'un algorithme obtienne pour la seconde fois la meilleure évaluation moyenne.
- avg-stddev $\lambda = 20, \kappa = 100$ step=x : Cette méthode de sélection fonctionne de la même manière que la première décrite, nous sélectionnons le meilleur algorithme sur 100 exécutions. Mais afin d'économiser du budget d'appels à la fonction objectif nous arrêtons d'évaluer les algorithmes dont la différence entre leur moyenne et celle du meilleur algorithme est plus grande que l'écart type de ce dernier.
- avg-2stddev $\lambda = 20, \kappa = 100$ step=x : Celle-ci est identique à la méthode que l'on vient de décrire mais la différence nécessaire pour arrêter d'évaluer un algorithme est de deux écarts types.
- avg-stddev $\lambda = 20, \kappa = 1000$ step=x : De même pour cette méthode où la moyenne est calculée sur 1000 exécutions plutôt que 100.
- tournois 1vs1 $\lambda = 20, \kappa = x$: Ici, nous comparons les algorithmes deux à deux sur des moyennes calculées à partir de κ exécutions par algorithme. Les $\lambda/2$ algorithmes issus de ces comparaisons sont de nouveau comparés pour une seconde étape sur des moyennes calculées à partir de 2κ exécutions (les κ premières, ainsi que κ exécutions supplémentaires). Nous augmentons le nombre d'exécutions pour affiner les estimations moyennes à chaque étape jusqu'à être en mesure de sélectionner un algorithme. À chaque étape, s'il reste un nombre impair d'algorithmes, le plus faible (selon les estimations déjà calculées) est écarté.
- tournois 50% $\lambda = 20, \kappa = x$: Ce tournoi fonctionne de la même façon mais les algorithmes ne sont pas comparés deux à deux, mais l'ensemble des algorithmes sont évalués en même temps. À chaque étape d'évaluation nous gardons les 50% meilleurs des algorithmes (arrondi à l'entier inférieur), jusqu'à ce qu'il n'en reste qu'un.
- tournois 20% $\lambda = 20, \kappa = x$: Même méthode, en ne conservant que 20% des algorithmes à chaque étape (arrondi à l'entier inférieur).
- tournois 80% $\lambda = 20, \kappa = x$: Dans cette configuration 80% des algorithmes restent en lice à chaque étape.

Comme nous pouvons le voir sur la figure 4.4, les sélections par tournoi permettent d'obtenir les meilleurs résultats, en particulier la configuration conservant 50% des algorithmes à chaque étape. Pour la suite des expérimentations, nous gardons parmi ces méthodes de sélection cette stratégie par tournoi ainsi que la simple répartition uniforme du budget qui servira de référence.

Nous avons également expérimenté d'autres alternatives éprouvées dans un contexte de sélection d'opérateurs, Comme nous l'avons vu dans la section 3.3.2, l'évaluation d'un algorithme

stochastique peut être vue comme une variable aléatoire. Nous pouvons ainsi appliquer à notre problème de sélection d'algorithme des algorithmes spécifiquement adaptés aux problèmes de bandits manchots. Nous intégrons aux deux méthodes de sélection préalablement choisies les stratégies UCB et SoftMax que nous avons présentées en section 3.3.2. Ces algorithmes de bandits manchots dépendent d'un horizon correspondant à un budget d'appels aux différents bras, donc ici d'exécutions des différents algorithmes à comparer. Les différentes stratégies seront comparées à budget par itération (une étape de génération de la stratégie d'évolution) identique.

La figure 4.5 présente la comparaison de ces algorithmes selon deux points de vue et sur plusieurs instances. La figure se divise en deux parties. La partie de gauche est similaire à la figure 4.4, à la différence près que l'axe des abscisses représente le budget global par itération (pour comparer l'ensemble des algorithmes) et non le budget dépensé par algorithme. Le budget total de la stratégie d'évolution est fixé à 10^7. Ensuite, la partie de droite présente les exécutions du meilleur paramétrage identifié sur la première partie pour chaque méthode de sélection. La légende indique le paramétrage utilisé. Nous comparons alors les fonctions d'évaluation sur quatre instances NK ($NK_{128,1}$,$NK_{128,12}$,$NK_{512,1}$,$NK_{512,12}$).

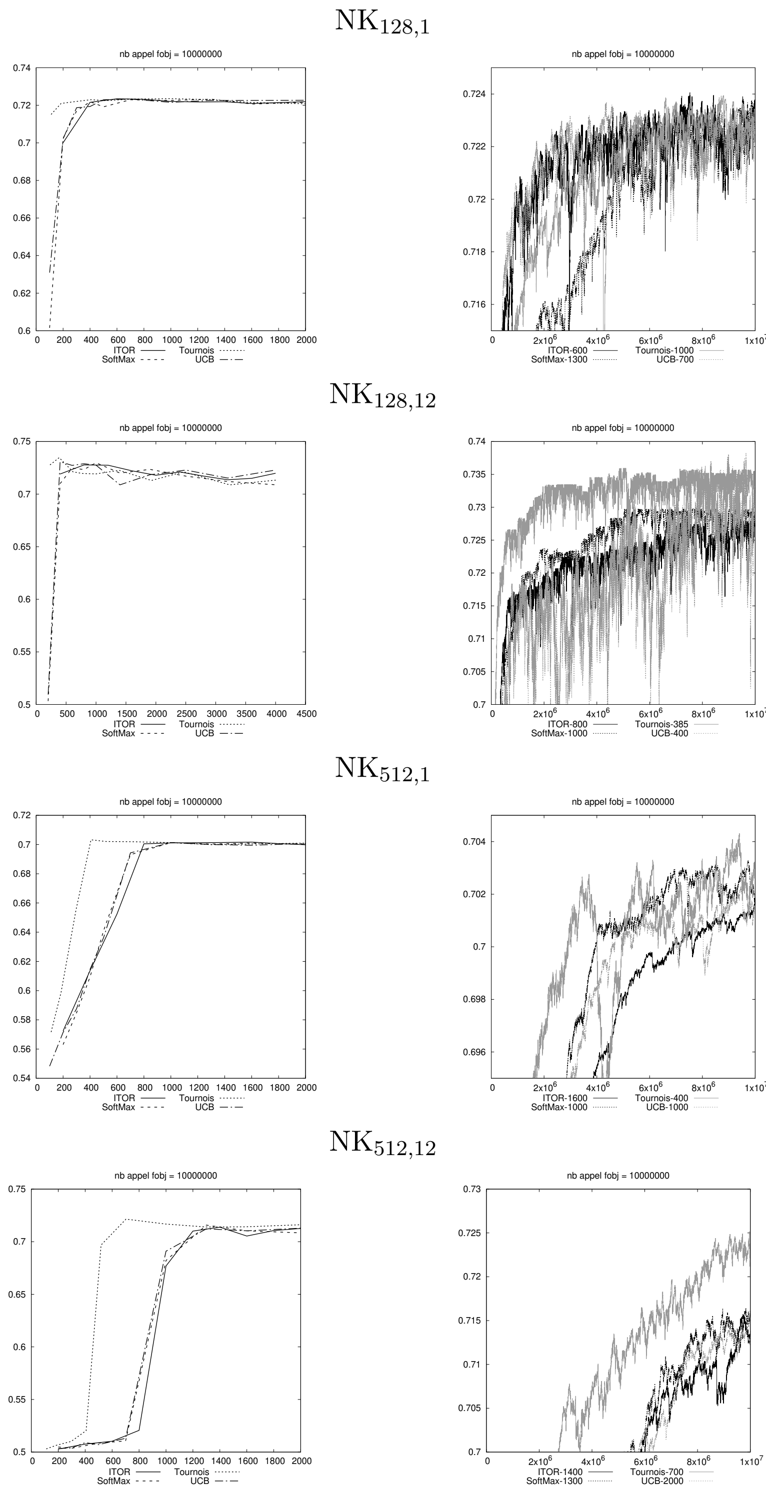
Comme nous pouvons le voir sur la partie de gauche de la figure 4.5, pour l'ensemble des instances, la méthode de sélection par tournoi est la plus robuste. En effet, les résultats atteints avec cette méthode varient moins qu'avec les autres méthodes. De plus, le meilleur paramétrage du tournoi atteint sur chacune des instances les meilleurs résultats. Cependant, en s'attardant sur l'évolution des résultats grâce à la seconde partie, nous pouvons voir que sur la plupart des instances les différentes méthodes ont des résultats proches. Ainsi, il est difficile de justifier la supériorité d'une méthode sur les autres à partir de cette seule expérience. C'est pourquoi, afin d'obtenir des résultats plus probants, nous continuons la comparaison de ces quatre fonctions de sélection avec un protocole plus abouti qui aura pour objectif de déterminer la stratégie à conserver dans la suite des études.

4.5.2 Protocole expérimental

Dans la section précédente nous avons construit un ensemble de méthodes de sélection afin d'en choisir une seule paramétrable pour l'utiliser dans un processus de génération le plus efficace possible. Pour rappel, les fonctions de sélections présentes dans cet ensemble sont :

- Estimation de l'espérance des algorithmes via un budget réparti uniformément.
- Mise en concurrence des algorithmes via un tournois à plusieurs étapes.
- L'algorithme UCB appliqué à la sélection d'algorithmes.
- L'algorithme SoftMax appliqué à la sélection d'algorithmes.

Figure 4.5 – Comparaison de méthodes de sélection issues de la résolution des problèmes de bandits manchots avec les méthodes sélectionnées auparavant



Pour faciliter le paramétrage du tournoi, nous fixons le nombre d'étapes à 3 et établissons un budget global de tournoi (unique paramètre). Le budget est alors équitablement réparti entre les différentes évaluations d'algorithmes nécessaires ($\lambda + \lfloor p\lambda \rfloor + \lfloor p\lambda \rfloor$, où p est la proportion de mutants à garder à chaque étape pour terminer la sélection avec une seule mutation).

Contrairement à la section précédente où l'on jugeait de la pertinence des fonctions de sélection grâce aux résultats atteints sur un certain nombre d'exécutions, nous nous concentrons ici sur l'évaluation des choix spécifiques de chaque méthode. Pour y arriver, nous classons chaque algorithme proposé en fonction de sa qualité puis laissons chaque fonction de sélection choisir un de ces algorithmes. La meilleure méthode de sélection sera alors celle qui choisit en moyenne les meilleurs algorithmes.

Afin de comparer efficacement les différentes stratégies de sélection d'algorithmes, celles-ci seront évaluées simultanément au cours d'une stratégie de génération d'algorithme dont le choix effectif du mutant sera indépendant des stratégies de sélection en comparaison. Plus précisément, nous exécutons l'algorithme de génération d'algorithme (1,λ)-ES sur quatre instances ($NK_{128,1}$,$NK_{128,12}$,$NK_{512,1}$,$NK_{512,12}$). À chaque itération, chacun des $\lambda = 20$ mutants est évalué 1000 fois afin de les classer par performance espérée de manière précise et avec confiance relativement élevée ; le meilleur est alors choisi pour l'itération suivante. Dans le même temps, nous proposons les mêmes mutations à toutes les méthodes de sélection, pour différents budgets (500, 1000 ou 1500 au total par itération) et sauvegardons le classement de la mutation sélectionnée. Nous suivons cette procédure sur 1000 itérations pour avoir un échantillon conséquent et sur différentes qualités d'algorithmes durant le processus d'évolution. Nous calculons alors la moyenne et la médiane des rangs des algorithmes sélectionnés par chacune des méthodes (et plus généralement reportons la répartition des rangs), en comparaison avec la méthode de référence appliquée avec un budget étendu (1000λ appels à la fonction objectif par itération).

4.5.3 Résultats

Le tableau 4.8 compile l'ensemble des résultats obtenus via le protocole que l'on vient de décrire. Du fait du nombre de combinaisons différentes de paramétrages et de la pluralité des résultats retournés pour un paramétrage, les colonnes et les lignes du tableau regroupent quatre critères différents. D'abord, la première ligne indique les instances utilisées, et la seconde le paramètre d'horizon, ou le nombre d'appels à la fonction objectif par itération. Ensuite, les quatre lignes suivantes présentent les rangs moyens des algorithmes sélectionnés (sur $\lambda = 20$) obtenus en moyenne, et les quatre dernières indiquent les rangs médians.

Le tableau 4.8 nous montre que les valeurs médianes sont très souvent identiques, la discrimination des méthodes se faisant alors sur les rangs moyens. Les valeurs moyennes sont d'ailleurs

Table 4.8 – Moyennes et médianes des rangs des mutations sélectionnées par les algorithmes de sélection sur différentes instances et avec différents nombre d'appels à la fonction objectif par itération

Instance		$NK_{128,1}$			$NK_{128,12}$			$NK_{512,1}$			$NK_{512,12}$		
Appels par iteration		500	1000	1500	500	1000	1500	500	1000	1500	500	1000	1500
Rang moyen	Espérance	4.7	5.1	4.5	5.1	5.0	4.9	4.2	3.9	3.4	4.4	3.6	3.2
	Tournois	**4.5**	**5.0**	**4.3**	**5.0**	**4.8**	**4.8**	**3.7**	**3.3**	**2.7**	**3.8**	**3.1**	**2.8**
	UCB	4.6	5.1	4.4	**5.0**	4.9	**4.8**	4.0	3.8	3.2	4.2	3.6	3.1
	SoftMax	4.7	5.1	4.5	5.1	5.0	4.9	4.3	4.0	3.4	4.5	3.6	3.3
Rang médian	Espérance	5	5	4	5	5	5	**3**	3	**2**	4	3	2
	Tournois	5	5	4	5	5	5	**3**	**2**	**2**	**3**	**2**	2
	UCB	5	5	4	5	5	5	**3**	3	**2**	**3**	3	2
	SoftMax	5	5	4	5	5	5	4	3	3	**3**	3	2

sensibles aux extrêmes et soutiennent particulièrement, dans ce contexte, notre objectif. En effet, les méthodes de sélection qui choisissent des algorithmes dont le classement est très bas, même rarement, seront désavantagés. La réciproque est aussi vraie. Ces deux caractéristiques nous permettent alors de choisir la fonction de sélection la plus robuste et la plus efficace.

Nous pouvons alors voir que, en moyenne, le tournoi atteint de meilleurs résultats que les autres méthodes et ce sur toutes les instances et tous les paramétrages. Pour confirmer ces résultats nous pouvons nous intéresser à la répartition des classements des algorithmes sélectionnés par les différentes fonctions de sélection, ce qui est présenté sur la figure 4.6. Chaque ligne de graphique correspond à une instance et chaque colonne à un budget par itération. Les douze graphiques sont aussi disponibles en annexe en plus grand format (figures A2 à A13). Nous pouvons constater que le tournoi sélectionne toujours davantage de fonctions mutantes estimées à la première place du classement, donc les meilleurs algorithmes proposés à la sélection. De plus, cette méthode de sélection par tournoi sélectionne peu de mauvais algorithmes.

L'ensemble des résultats que l'on a pu rassembler dans cette section sont cohérents et désignent la méthode de sélection par tournoi comme la plus pertinente. Notons que les tournois que n'ont pas été paramétrés spécifiquement pour chaque instance ; de meilleurs comportements pourraient donc être observés en réglant différemment les paramètres. Nous optons ainsi pour cette méthode de sélection par tournois pour l'expérimentation de plus grande envergure décrite dans la prochaine section.

4.6 Expérimentation étendue

Dans cette section, nous avons comme objectif de montrer l'efficacité de la génération d'algorithmes selon plusieurs critères. D'abord, sur la capacité de cette approche à générer des

FIGURE 4.6 – Répartition des rangs des algorithmes sélectionnés sur plusieurs instances et différents paramétrages de nombre d'appels à la fonction objectif par itération

Nombre d'appels à f_{obj} par itération = 500	Nombre d'appels à f_{obj} par itération = 1000	Nombre d'appels à f_{obj} par itération = 1500

$\text{NK}_{128,1}$

$\text{NK}_{128,12}$

$\text{NK}_{512,1}$

$\text{NK}_{512,12}$

algorithmes, et donc des solutions, pour différents problèmes d'optimisation. Ensuite, la possibilité de générer ces algorithmes via l'évolution de fonctions d'évaluation avec différents modèles de fonction. Puis, de générer des algorithmes avec différentes propriétés, notamment des algorithmes déterministes et stochastiques. Et enfin, de générer des algorithmes efficaces selon plusieurs critères d'évaluation.

Cette étude est découpée en deux étapes : une première dédiée au paramétrage de la stratégie d'évolution selon les instances, et une seconde à la génération d'algorithmes sur les différentes instances de problème et en utilisant différents modèles de fonction d'évaluation.

4.6.1 Protocole expérimental

Description des composants

Afin de se restreindre à un unique modèles de stratégie d'évolution, nous limitons l'expérimentation à la stratégie (1,λ)-ES qui a retourné invariablement de meilleurs hill climbers stochastiques (section 4.3) et déterministes (section 4.4). La différence d'efficacité des algorithmes générés s'explique par la capacité exploratoire de la stratégie (1,λ)-ES contre (1+1)-ES qui a davantage tendance à rester bloquée sur une fonction d'évaluation.

Afin de tester l'efficacité de la génération d'algorithmes sur plusieurs problèmes dont les structures sont différentes, et donc dont les obstacles à surmonter lors de l'optimisation ne sont pas les mêmes, nous ajoutons à notre benchmark de 28 instances NK un ensemble de 31 instances MAX-3SAT provenant de la bibliothèque SATLIB [100] et possédant des tailles de problème et caractéristiques diverses :

- flat : 8 instances (de taille 90 à 600) issues de problèmes de coloration de graphes planaires dont le nombre chromatique est de 3.
- blocksworld : 4 instances (de taille 48 à 459) issues de problèmes de planification.
- ais : 4 instances (de taille 61 à 265) de problèmes de permutation dont l'objectif est de trouver une suite de nombre dont tous les intervalles sont représentés.
- sw100-8-lp : 5 instances (de taille 500) issues de problèmes de coloration de graphes plus ou moins structurés.
- uf/uuf : 10 instances (de taille 50 et 250) de problèmes MAX-3SAT aléatoires, 5 satisfiables (uf) et 5 non satisfiables (uuf).

Les instances NK permettent tout particulièrement de mesurer l'effet de la rugosité du paysage induit par l'instance et du nombre plus ou moins important d'optimums locaux (paramètre

K). Les paysages MAX-3SAT intègrent une particularité supplémentaire, la neutralité, qui réduit l'information exploitable au niveau local et bloque d'autant plus les hill climbers.

La génération d'algorithmes que nous présentons dans cette thèse se fait à travers l'évolution de fonctions d'évaluation. Dans les sections précédentes nous avons utilisé exclusivement le modèle NK (section 3.2.1 chapitre 3) pour représenter une fonction. Désormais, nous étendons les modèles de fonctions utilisés au modèle MAX-3SAT (section 3.2.2) et au modèle de somme de fonctions pseudo-booléennes (section 3.2.3). Les différentes combinaisons classes de problèmes / modèles de fonctions seront testées.

Nous intégrons également à cette étude les différents modèles de recherches locales. Dans le cas des hill climbers stochastiques (section 3.1.1), les caractéristiques de la fonction d'évaluation générée devront permettre d'établir des trajectoires ascensionnelles de qualité depuis des points de départ aléatoires. Au contraire, le modèle de hill climbers déterministes à départ unique fixé (section 3.1.2) implique de déterminer une fonction d'évaluation permettant de décrire une trajectoire ascensionnelle unique vers une solution la plus haute possible. Enfin, le modèle des hill climbers déterministes à départs multiples fixés combine les spécificités des deux précédents modèles, avec une fonction devant prendre en compte différentes trajectoires mais sans composante stochastique.

Paramétrage des algorithmes de génération

L'objectif est de trouver le paramétrage qui donne les meilleurs résultats sur l'instance la plus difficile. Le tableau 4.9 rappelle l'ensemble des paramètres. Le paramètre propre au hill climber déterministe à départs multiples fixés, à savoir le nombre de solutions de départ, est fixé arbitrairement à 10. Ce tableau donne aussi l'ensemble de valeurs de chaque paramètre. Ces ensembles sont les mêmes pour chacune des versions de l'algorithme de génération, avec une exception pour le nombre de mutations par itération en ce qui concerne le modèle MAX-3SAT pour les hill climbers déterministes et pour la génération de hill climbers stochastiques.

Chaque combinaison de valeurs de paramètres est exécutée 10 fois avec un budget de 10^6 appels à f_{obj} sur chaque instance. Pour cette étape de paramétrage, nous utilisons un sous-ensemble d'instances de l'ensemble que nous avons défini plus haut, composé de 5 instances de chaque problème. Pour chaque combinaison de paramètres, on calcule un score selon la mesure correspondante. Puisque les différentes instances n'ont pas des scores comparables les unes aux autres, nous normalisons les scores obtenus en fonction du meilleur score obtenu sur cette instance. Le meilleur paramétrage pour une instance donnée aura alors un score de 1, et les autres un score compris entre 0 et 1. Cependant, étant donné que les ensembles de valeurs que l'on teste pour les différents paramètres sont de taille restreinte, le paramétrage obtenu pourrait

Table 4.9 – Valeurs possibles des paramètres

Paramètre	Intervalles		
		Modèle MAX-3SAT (HC déterministe)	HC stochastique
Nombre de mutations par itération (λ)	$\{50x \mid x \in [\![1;5]\!]\}$	$\{500x \mid x \in [\![1;5]\!]\}$	$\{10x \mid x \in [\![1;5]\!]\}$
Nombre d'étapes de sélection tournois	$[\![2;4]\!]$		
Nombre d'appels par itération	$\{300x \mid x \in [\![1;10]\!]\}$		
Nombre de clauses (modèle MAX-3SAT)	$\{5x \mid x \in [\![1;6]\!]\}$		

être amélioré avec un choix de valeur plus fin. C'est pourquoi nous décidons de fixer tous les paramètres à l'exception d'un seul, qu'il faudra définir, pour chaque version de l'algorithme de génération (une version étant une combinaison de problème étudié, modèle de fonction, et modèle d'algorithme). Ainsi, pour les versions d'algorithmes qui génèrent des algorithmes déterministes avec les modèles de fonctions NK ou somme de fonctions pseudo-booléennes, seul le nombre de mutations par itération doit être fixé. Pour générer les algorithmes les plus spécifiques possibles aux instances concernées nous choisirons une valeur de ce paramètre par instance. Pour les autres versions de l'algorithme de génération, tout comme les versions que l'on vient de voir, nous laissons un paramètre à ajuster en fonction de chaque instance ; on peut d'ailleurs considérer cette étape préliminaire comme intégrée au processus de génération d'un algorithme dédié à l'instance à résoudre. Le choix de ce paramètre dépend des scores introduits ci-avant, selon une procédure que nous proposons de décrire au moyen d'un exemple (problème NK, modèle de fonction MAX-3SAT et modèle d'algorithme déterministe à départ unique fixé).

Le tableau 4.10 présente les scores obtenus par les différents paramétrages sur une instance. Ce tableau a deux dimensions car l'algorithme de génération possède deux paramètres : le nombre de clauses pour le modèle de fonction et de nombre de mutations par itération. Sur ce tableau, une ligne ou une colonne correspond aux scores obtenus pour un paramètre fixé pendant que l'autre varie. Ainsi, pour le paramètre fixé, le meilleur score est précisé dans la ligne/colonne Max du tableau 4.10.

Le tableau 4.11 montre la compilation des résultats obtenus sur la même version d'algorithme que précédemment. Comme nous l'avons vu, nous voulons garantir un certain niveau d'efficacité

TABLE 4.10 – Exemple de résultats pour l'ensemble des paramétrages possibles pour la génération d'algorithmes déterministes pour le problème NK en utilisant le modèle MAX-3SAT

Nombre de clauses	Nombre de mutations 500	1000	1500	2000	2500	Max
5	0.976	0.971	0.973	0.974	0.969	0.976
10	0.987	0.987	0.990	0.984	0.979	0.990
15	0.991	0.988	0.992	0.989	0.979	0.992
20	0.995	0.992	0.997	0.998	0.994	0.998
25	0.997	0.995	0.993	0.992	0.994	0.997
30	1.000	0.998	0.998	0.998	0.998	1.000
Max	1.000	0.998	0.998	0.998	0.998	

TABLE 4.11 – Exemple de résultats compilés pour la génération d'algorithmes déterministes pour le problème NK en utilisant le modèle MAX-3SAT

Nombre de clauses	Nombre de mutations 500	1000	1500	2000	2500	Max
5						0.927
10						0.962
15						0.971
20						0.970
25						0.987
30						0.976
Max	1.000	0.974	0.967	0.933	0.941	

sur l'ensemble des instances que l'on va rencontrer.

En choisissant le meilleur score, nous choisissons le paramétrage dont le pire score sera le meilleur d'entre tous les paramétrages. On garantit ainsi un certain niveau d'efficacité, en se concentrant sur les instances les plus difficiles pour chaque paramétrage. Pour l'exemple du tableau 4.11, le meilleur score est atteint pour le paramétrage Nombre de mutations = 500 avec un score de 1.

Nous effectuons cette étude de paramètres pour chaque version de l'algorithme de génération, ce qui est résumé dans le tableau 4.12. Les trois premières colonnes définissent les différentes versions de l'algorithme. Les cinq colonnes suivantes présentent les différentes valeurs fixées pour chaque paramètre. La dernière colonne montre le score obtenu pour chaque paramétrage choisi. Les cellules du tableau sont grisées (en foncé) lorsque les paramètres n'interviennent pas dans le fonctionnement de la version de l'algorithme de génération en question. Les cellules en gris clair sont les paramètres dont la valeur a été fixée auparavant. Le reste des cellules présentent

Table 4.12 – Ensemble des valeurs de tous les paramètres fixés ou non pour chaque version de l'algorithme de génération

Version de l'algorithme			Paramètres					Score
Problème	Algorithme généré	Modèle de fonction	Nombre de mutations par itération	Nombre d'étapes de sélection tournois	Nombre d'appels par itération	Nombre de clauses (Modèle MAX-3SAT)	Précision	
NK	HC déterministe départ unique	NK	$\{10x \mid x \in [\![5;30]\!]\}$					1.000
		MAX-3SAT	500			$[\![1;30]\!]$		1.000
		FPB	$\{10x \mid x \in [\![5;30]\!]\}$					1.000
	HC déterministe départs multiple	NK	$\{10x \mid x \in [\![5;30]\!]\}$				10	1.000
		MAX-3SAT	500			$[\![1;30]\!]$	10	1.000
		FPB	$\{10x \mid x \in [\![5;30]\!]\}$				10	1.000
	HC stochastique	NK	30	4	$\{100x \mid x \in [\![1;30]\!]\}$			0.988
		MAX-3SAT	$\{5x \mid x \in [\![1;30]\!]\}$	4	1200	30		0.964
		FPB	30	4	$\{100x \mid x \in [\![1;30]\!]\}$			0.974
MAX-SAT	HC déterministe départ unique	NK	$\{10x \mid x \in [\![5;30]\!]\}$					1.000
		MAX-3SAT	$\{100x \mid x \in [\![1;30]\!]\}$			30		0.999
		FPB	$\{10x \mid x \in [\![5;30]\!]\}$					1.000
	HC déterministe départs multiple	NK	$\{10x \mid x \in [\![5;30]\!]\}$				10	1.000
		MAX-3SAT	500			$[\![1;30]\!]$	10	1.000
		FPB	$\{10x \mid x \in [\![5;30]\!]\}$				10	1.000
	HC stochastique	NK	30	3	$\{100x \mid x \in [\![1;30]\!]\}$			0.988
		MAX-3SAT	10	4	$\{100x \mid x \in [\![1;30]\!]\}$	30		0.937
		FPB	50	4	$\{100x \mid x \in [\![1;30]\!]\}$			0.987

les valeurs choisies grâce au protocole. Les paramètres dont la valeur est un ensemble sont ceux dont la valeur sera fixée spécifiquement à chaque instance.

Les paramètres dont la valeur reste à déterminer sont résumés dans le tableau 4.13. Nous remarquons que le nombre d'étapes de sélection du tournoi n'a pas d'ensemble associé. Cela indique que ce paramètre est soit inutilisé par certaines versions de l'algorithme, soit fixé à une valeur précise pour le reste des versions. Ensuite, nous pouvons voir que les ensembles présentés dans le tableau 4.13 contiennent plus de valeurs que ceux du tableau 4.9. Cette extension est prévue dans ce cas de figure où il n'y a plus qu'un seul paramètre à fixer pour chaque version de l'algorithme et sur chaque instance.

Afin de définir les valeurs de paramètre spécifiques à chaque instance, nous utilisons les mesures d'efficacité définies précédemment en fonction du modèle d'algorithme. Ensuite, pour chaque instance et chaque version d'algorithme, nous exécutons 10 fois chaque algorithme avec les valeurs données dans le tableau 4.12 et un budget de 10^6 appels à la fonction objectif et ce, pour chaque valeur possible du tableau 4.13. La valeur de paramètre qui obtient alors le meilleur résultat sur une instance donnée est sélectionnée.

TABLE 4.13 – Intervalles de tous les paramètres avec une spécification pour le nombre de mutations par itérations pour les hill climbers stochastiques et le modèle MAX-3SAT

Paramètre	Intervalles		
		Modèle MAX-3SAT (HC déterministe)	HC stochastique
Nombre de mutations par itération (λ)	$\{10x \mid x \in [\![5;30]\!]\}$	$\{100x \mid x \in [\![1;30]\!]\}$	$\{5x \mid x \in [\![1;30]\!]\}$
Nombre d'étapes de sélection tournois			
Nombre d'appels par itération	$\{100x \mid x \in [\![1;30]\!]\}$		
Nombre de clauses (modèle MAX-3SAT)	$[\![1;30]\!]$		

Protocole de génération

Une fois tous les paramètres définis, nous pouvons exécuter l'algorithme de génération sur toutes les instances en générant des fonctions d'évaluation pour chaque modèle de fonction et chaque modèle d'algorithme. Le nombre d'instances et d'algorithmes étant importants nous exécutons chaque algorithme sur chaque instance seulement 20 fois contre 100 sur les expériences précédentes. Notre algorithme étant robuste, calculer des moyennes sur 20 exécutions au lieu de 100 permet tout de même d'exploiter convenablement les résultats. Ces exécutions auront un budget de 10^8 appels à f_{obj} comme pour les expériences précédentes. Ces 20 exécutions génèrent 20 algorithmes de recherches locales. Ce sont ces algorithmes que nous allons évaluer afin d'évaluer l'algorithme de génération.

Algorithmes de référence

Les différents algorithmes générés seront comparés avec leurs équivalents guidés par la fonction objectif. Nous indiquons aussi une borne inférieure de la valeur de l'optimum attendu pour chaque instance. Ces valeurs représentent le score de l'optimum global pour ces instances MAX-3SAT, dont les solutions optimales sont connues, et sont les scores des meilleures solutions rencontrées au moyen d'une stratégie d'évolution (1,λ) appliquée de manière prolongée à l'espace des solutions $\mathcal{X}$ pour les problèmes NK.

4.6.2 Résultats

Les résultats de cette étude sont regroupés dans les tableaux 4.12, 4.14, et 4.15. Le tableau 4.14 présente les valeurs utilisées spécifiquement pour chaque instance selon la version de l'algorithme de génération utilisée pour le dernier paramètre à fixer. Ce tableau se décompose en deux parties : le problème concerné avec les instances utilisées, puis les valeurs de paramètres correspondant à ces dernières. Les valeurs de paramètres sont organisées en trois séries de trois colonnes pour chaque modèle de fonction puis chaque modèle d'algorithme de recherche locale utilisé. L'algorithme noté "1" correspond au hill climber déterministe à départ unique fixé, l'agorithme "2" au hill climber déterministe à départs multiples fixés et l'algorithme "3" au hill climber stochastique. Les valeurs présentées dans les tableaux 4.12 et 4.14 sont alors utilisées de concert pour l'exécution de l'algorithme de génération.

Le tableau 4.15 présente l'ensemble des résultats obtenus pour chaque instance, chaque version de l'algorithme de génération et chaque mesure. Les deux premières colonnes présentent les problèmes utilisés ainsi que toutes les instances de ces problèmes. Les trois colonnes suivantes présentent les résultats obtenus par les algorithmes de référence ainsi que les bornes correspondant aux scores des meilleures solutions connues. Le reste du tableau est consacré aux résultats obtenus par les algorithmes générés, et d'abord divisé en trois parties qui correspondent chacune à une mesure d'efficacité : meilleure solution renvoyée par un algorithme, moyenne des solutions renvoyées en utilisant le(s) point(s) de départ éventuellement fixé(s) par les climbers, et une estimation de l'espérance des fonctions d'évaluation sur un hill climber stochastique. Ensuite, chaque partie se divise en trois séries de trois colonnes pour chaque modèle de fonction utilisé puis chaque modèle d'algorithme. Le meilleur score obtenu pour une instance sur une mesure est indiqué en gras pour faciliter la lecture du tableau. Précisons que les résultats peuvent être dupliqués sur plusieurs colonnes : première et deuxième partie pour le modèle de climber "1" (le prérequis algorithmique étant un unique point de départ fixé, le climber retournera toujours la même solution qui sera donc également la meilleure), et deuxième et troisième partie pour le modèle de climber "3" (il n'y a pas de point de départ fixé dans l'algorithme de type hill climbers stochastique, nous reportons donc les valeurs obtenues à partir de points de départ aléatoires). Nous laissons volontairement ces colonnes identiques afin de faciliter la lecture comparative des résultats au sein de chaque partie.

Sur le tableau 4.15 nous pouvons voir que la grande majorité des algorithmes générés surpassent l'espérance d'un hill climber guidé par la fonction objectif sur au moins une mesure. Ensuite, une partie des algorithmes générés surpassent les scores atteints par les hill climbers itérés. Et enfin, quelques algorithmes atteignent sur certaines instances les meilleures solutions connues. Cela indique que la génération d'algorithmes peut être un moyen de trouver de bonnes solutions à des problèmes d'optimisation. Cet objectif n'est certes atteint que pour une partie des

algorithmes générés et des instances ; c'est donc un axe d'amélioration possible que de rendre la génération d'algorithme plus robuste sur l'ensemble des instances des problèmes d'optimisation.

Les résultats diffèrent peu entre les problèmes NK et MAX-3SAT, mais les différences entre les versions des algorithmes sont plus prononcées pour les problèmes NK. Nous pouvons voir que l'évolution de fonctions d'évaluation basées sur le modèle de fonction NK est celle qui atteint les meilleurs résultats en général. Le modèle d'algorithme déterministe à départ unique fixé obtient en général les meilleurs scores dans la partie 1 (meilleures solutions rencontrées), et donc naturellement les meilleurs scores dans la partie 2. De manière attendue, le modèle d'algorithme stochastique atteint les meilleures espérances dans la partie 3, où les fonctions d'évaluation générées sont réévaluées dans un contexte stochastique. Il est intéressant de noter sur cette dernière partie du tableau que les fonctions d'évaluation générées pour les climbers déterministes à point de départ unique fixé (modèle d'algorithme "1") ont des optimums locaux en moyenne de très mauvaise qualité, tandis que les fonctions générées pour les mêmes climbers évalués depuis 10 points de départ fixés (modèle d'algorithme "2") ont des qualités d'optimums locaux qui surpassent déjà en moyenne ceux de la fonction objectif d'origine (colonne HC). Les modèles d'algorithmes 2 et 3 parviennent ainsi à faire évoluer suffisamment un paysage de fitness au moyen de trajectoires multiples à points de départ fixés ou non, tandis que le premier modèle ne fait évoluer qu'une trajectoire sans que le paysage de fitness ne soit impacté bénéfiquement par de cette exploitation.

Table 4.14 – Liste des valeurs utilisées pour chaque version de l'algorithme de génération $(1, \lambda)$-ES et chaque instance

Problème	Instance	Valeur de paramètre utilisée								
		Modèle de fonction								
		NK			MAX-3SAT			$\sum f : 2^n \mapsto \mathbb{R}$		
		Modèle d'algorithme								
		1	2	3	1	2	3	1	2	3
NK	128 1	90	70	200	30	16	150	210	70	400
	128 2	100	110	200	27	19	40	150	70	400
	128 4	80	60	200	21	18	25	150	70	300
	128 6	100	50	200	25	19	25	160	60	200
	128 8	110	110	200	19	26	75	140	50	200
	128 10	120	80	300	20	29	50	170	70	200
	128 12	180	100	300	15	30	30	250	70	200
	256 1	110	80	300	25	30	65	120	50	600
	256 2	80	60	200	25	27	10	150	50	800
	256 4	90	50	300	22	25	35	120	80	500
	256 6	130	60	300	24	30	20	150	60	500
	256 8	110	100	600	23	29	10	210	140	400
	256 10	180	70	300	26	28	10	210	50	900
	256 12	210	130	400	28	30	15	230	50	500
	512 1	100	60	600	27	30	10	170	50	1100
	512 2	70	50	500	23	29	15	100	70	900
	512 4	120	60	700	27	30	10	120	60	900
	512 6	120	80	900	26	29	5	180	90	1200
	512 8	180	140	800	27	26	35	160	80	900
	512 10	200	90	900	29	29	125	210	60	1100
	512 12	190	90	1000	22	25	25	280	50	900
	1024 1	100	50	1100	25	29	40	110	50	1400
	1024 2	100	50	1100	29	30	30	100	50	1700
	1024 4	120	70	1700	29	27	5	150	50	2000
	1024 6	130	60	1100	29	27	15	170	70	1800
	1024 8	150	80	1300	29	28	10	190	70	2100
	1024 10	200	70	2300	30	28	15	270	50	1700
	1024 12	180	60	2600	29	30	125	280	50	2800
MAX-SAT	flat30	130	90	200	100	28	100	190	70	1100
	flat50	300	80	200	400	23	500	300	60	1600
	flat75	240	110	300	300	30	400	250	70	2200
	flat100	220	110	500	600	28	700	250	90	2800
	flat125	210	110	500	700	29	800	240	70	3000
	flat150	300	90	600	1100	25	1100	280	80	2800
	flat175	290	70	900	800	25	700	280	70	2500
	flat200	270	70	800	700	29	800	280	50	2800
	anomaly	50	110	200	100	14	100	110	50	1100
	medium	200	170	400	200	19	200	300	100	1700
	bw_large.a	260	50	600	1200	29	1300	290	70	2600
	huge	260	80	600	1000	29	1300	300	80	3000
	ais6	200	60	300	100	17	200	140	60	2400
	ais8	230	170	200	200	29	200	290	90	2900
	ais10	300	180	200	700	29	600	300	110	2800
	ais12	240	100	400	600	28	900	270	70	2800
	sw100-8-lp0	250	90	800	800	29	600	260	60	2600
	sw100-8-lp1	280	180	800	900	29	1200	290	60	3000
	sw100-8-lp2	260	80	800	1300	29	1200	280	60	2500
	sw100-8-lp3	240	110	900	600	29	1700	300	50	2500
	sw100-8-lp4	280	80	900	800	29	800	290	60	2500
	uf50	50	50	400	100	24	100	300	50	500
	uf100	160	60	200	400	16	100	100	80	200
	uf150	210	130	200	200	16	200	230	60	500
	uf200	200	90	200	300	30	500	180	50	600
	uf250	200	110	300	400	28	600	230	90	600
	uuf50	50	50	1100	100	13	200	50	60	400
	uuf100	160	80	200	200	23	200	180	50	400
	uuf150	190	110	200	400	19	300	230	60	500
	uuf200	190	70	300	200	28	400	190	60	500
	uuf250	180	120	300	300	30	1100	180	50	600

TABLE 4.15 – Ensemble des résultats pour chaque version de l'algorithme de génération $(1, \lambda)$-ES sur chaque instance

Problème	Instance	HC	RRHC	Meilleure solution connue	Meilleure solution rencontrée									Meilleure fonction rencontrée (solutions de départ connues)									Meilleure fonction rencontrée (solutions de départ aléatoires)								
					Modèle de fonction																										
					NK			MAX-3SAT			$\sum f : 2^n \mapsto \mathbb{R}$			NK			MAX-3SAT			$\sum f : 2^n \mapsto \mathbb{R}$			NK			MAX-3SAT			$\sum f : 2^n \mapsto \mathbb{R}$		
					Modèle d'algorithme																										
					1	2	3	1	2	3	1	2	3	1	2	3	1	2	3	1	2	3	1	2	3	1	2	3	1	2	3
NK	128 1	0.709	0.720	0.725	**0.725**	**0.725**	**0.725**	**0.725**	0.724	0.724	**0.725**	**0.725**	0.724	**0.725**	**0.725**	0.724	**0.725**	0.724	0.724	**0.725**	0.724	0.722	0.626	0.721	**0.724**	0.629	0.723	**0.724**	0.597	0.707	0.722
	128 2	0.715	0.735	0.742	**0.742**	0.741	0.741	0.741	0.738	0.734	**0.742**	0.740	0.737	**0.742**	0.741	0.741	0.741	0.737	0.732	**0.742**	0.739	0.735	0.598	0.734	**0.741**	0.597	0.736	0.732	0.576	0.719	0.735
	128 4	0.725	0.783	0.796	**0.794**	0.790	0.789	0.789	0.776	0.768	0.793	0.784	0.780	**0.794**	0.788	0.789	0.789	0.776	0.767	0.793	0.784	0.775	0.581	0.770	**0.789**	0.567	0.771	0.767	0.576	0.752	0.775
	128 6	0.719	0.777	0.800	**0.796**	0.793	0.786	0.779	0.765	0.761	0.795	0.780	0.774	**0.796**	0.787	0.784	0.779	0.764	0.760	0.795	0.779	0.768	0.550	0.771	**0.784**	0.533	0.758	0.760	0.551	0.748	0.768
	128 8	0.714	0.781	0.802	**0.794**	0.782	0.786	0.776	0.762	0.753	0.792	0.783	0.774	**0.794**	0.782	0.786	0.776	0.761	0.753	0.792	0.781	0.766	0.542	0.765	**0.786**	0.512	0.754	0.753	0.539	0.754	0.766
	128 10	0.707	0.773	0.794	**0.785**	0.780	0.772	0.774	0.757	0.748	0.782	0.772	0.764	**0.785**	0.779	0.771	0.774	0.756	0.747	0.782	0.771	0.758	0.518	0.764	**0.771**	0.506	0.751	0.747	0.522	0.744	0.758
	128 12	0.702	0.764	0.782	**0.774**	0.766	0.761	0.763	0.746	0.737	0.771	0.759	0.754	**0.774**	0.765	0.761	0.763	0.746	0.736	0.771	0.758	0.746	0.509	0.745	**0.761**	0.503	0.737	0.736	0.520	0.715	0.746
	256 1	0.691	0.718	0.722	**0.722**	**0.722**	**0.722**	0.720	0.720	0.720	**0.722**	0.721	0.719	**0.722**	0.721	**0.722**	0.720	0.719	0.720	**0.722**	0.719	0.717	0.617	0.716	**0.722**	0.597	0.719	0.720	0.602	0.704	0.717
	256 2	0.713	0.729	0.744	0.743	**0.744**	**0.744**	0.740	0.737	0.735	0.742	0.736	0.732	**0.743**	0.742	**0.743**	0.740	0.737	0.735	0.742	0.734	0.729	0.606	0.733	**0.743**	0.577	0.735	0.735	0.562	0.709	0.729
	256 4	0.719	0.766	0.793	**0.790**	0.787	0.780	0.779	0.771	0.768	0.787	0.777	0.767	**0.790**	0.782	0.780	0.779	0.770	0.767	0.787	0.775	0.763	0.579	0.768	**0.780**	0.546	0.768	0.767	0.559	0.739	0.763
	256 6	0.724	0.771	0.805	**0.798**	0.794	0.784	0.780	0.774	0.768	0.795	0.782	0.773	**0.798**	0.787	0.784	0.780	0.774	0.767	0.795	0.779	0.770	0.553	0.766	**0.784**	0.522	0.770	0.767	0.543	0.743	0.770
	256 8	0.717	0.761	0.796	**0.784**	0.781	0.765	0.770	0.765	0.760	**0.784**	0.763	0.761	**0.784**	0.778	0.765	0.770	0.765	0.759	**0.784**	0.763	0.757	0.526	0.756	**0.765**	0.508	0.760	0.759	0.525	0.722	0.757
	256 10	0.712	0.755	0.787	**0.780**	0.772	0.767	0.766	0.759	0.624	0.775	0.759	0.750	**0.780**	0.763	0.767	0.766	0.758	0.542	0.775	0.754	0.748	0.517	0.735	**0.767**	0.504	0.753	0.542	0.520	0.717	0.748
	256 12	0.705	0.749	0.776	**0.768**	0.763	0.753	0.758	0.752	0.604	0.762	0.747	0.747	**0.768**	0.761	0.753	0.758	0.751	0.508	0.762	0.741	0.743	0.512	0.731	**0.753**	0.502	0.744	0.508	0.514	0.704	0.743
	512 1	0.690	0.697	0.709	**0.708**	0.707	0.707	0.704	0.706	0.684	**0.708**	0.705	0.700	**0.708**	0.706	0.707	0.704	0.704	0.667	**0.708**	0.703	0.697	0.603	0.700	**0.707**	0.581	0.703	0.667	0.579	0.687	0.697
	512 2	0.711	0.733	0.751	**0.749**	0.748	0.743	0.740	0.742	0.680	0.746	0.743	0.729	**0.749**	0.745	0.743	0.740	0.740	0.646	0.746	0.741	0.725	0.603	0.736	**0.743**	0.567	0.738	0.646	0.576	0.717	0.725
	512 4	0.724	0.749	0.786	**0.782**	0.776	0.761	0.764	0.764	0.606	0.776	0.766	0.751	**0.782**	0.772	0.761	0.764	0.762	0.550	0.776	0.763	0.747	0.571	0.756	**0.761**	0.533	0.760	0.550	0.565	0.731	0.747
	512 6	0.723	0.753	0.799	**0.789**	0.783	0.760	0.768	0.772	0.580	0.786	0.769	0.756	**0.789**	0.778	0.760	0.768	0.770	0.516	0.786	0.766	0.754	0.546	0.754	**0.760**	0.512	0.765	0.516	0.542	0.733	0.754
	512 8	0.721	0.751	0.794	**0.785**	0.775	0.756	0.763	0.768	0.574	0.774	0.765	0.754	**0.785**	0.772	0.756	0.763	0.763	0.506	0.774	0.761	0.751	0.528	0.748	**0.756**	0.505	0.755	0.506	0.529	0.721	0.751
	512 10	0.715	0.746	0.783	**0.773**	0.768	0.748	0.756	0.758	0.575	0.765	0.753	0.743	**0.773**	0.761	0.748	0.756	0.753	0.502	0.765	0.749	0.741	0.515	0.727	**0.748**	0.502	0.743	0.502	0.515	0.710	0.741
	512 12	0.708	0.739	0.772	**0.756**	0.753	0.740	0.747	0.649	0.574	**0.756**	0.734	0.734	**0.756**	0.744	0.740	0.747	0.613	0.501	0.756	0.727	0.731	0.508	0.712	**0.740**	0.500	0.512	0.501	0.510	0.687	0.731
	1024 1	0.700	0.705	0.717	**0.716**	0.715	0.713	0.708	0.714	0.567	0.714	0.711	0.690	**0.716**	0.713	0.713	0.708	0.712	0.532	0.714	0.710	0.683	0.609	0.706	**0.713**	0.577	0.711	0.532	0.585	0.693	0.683
	1024 2	0.714	0.729	0.752	**0.751**	0.747	0.740	0.737	0.744	0.568	0.745	0.741	0.722	**0.751**	0.745	0.740	0.737	0.742	0.528	0.745	0.739	0.716	0.605	0.734	**0.740**	0.565	0.739	0.528	0.581	0.716	0.716
	1024 4	0.723	0.745	0.788	**0.781**	0.774	0.750	0.758	0.764	0.557	0.777	0.761	0.744	**0.781**	0.770	0.750	0.758	0.761	0.511	0.777	0.757	0.740	0.571	**0.753**	0.750	0.527	**0.753**	0.511	0.559	0.727	0.740
	1024 6	0.726	0.745	0.795	**0.784**	0.772	0.756	0.758	0.761	0.553	0.780	0.763	0.743	**0.784**	0.766	0.756	0.758	0.756	0.504	0.780	0.760	0.738	0.544	0.745	**0.756**	0.508	0.745	0.504	0.541	0.727	0.738
	1024 8	0.721	0.743	0.790	**0.776**	0.768	0.749	0.752	0.757	0.552	0.771	0.756	0.738	**0.776**	0.762	0.749	0.752	0.750	0.503	0.771	0.752	0.735	0.525	0.736	**0.749**	0.504	0.736	0.503	0.529	0.716	0.735
	1024 10	0.716	0.737	0.779	**0.767**	0.749	0.736	0.744	0.642	0.551	0.763	0.737	0.728	**0.767**	0.740	0.736	0.744	0.620	0.500	0.763	0.731	0.724	0.515	0.710	**0.736**	0.500	0.523	0.500	0.516	0.693	0.724
	1024 12	0.711	0.733	0.769	0.748	0.725	0.727	0.736	0.630	0.551	**0.750**	0.724	0.721	0.748	0.715	0.727	0.736	0.606	0.500	**0.750**	0.718	0.719	0.508	0.682	**0.727**	0.500	0.505	0.500	0.511	0.678	0.719
MAX-3SAT	flat30	293.0	300.0	300	**300.0**	**300.0**	**300.0**	**300.0**	299.6	**300.0**	**300.0**	**300.0**	299.8	**300.0**	**300.0**	**300.0**	**300.0**	299.6	**300.0**	**300.0**	299.9	298.5	261.6	298.5	**300.0**	267.3	298.3	**300.0**	238.1	287.4	298.5
	flat50	532.7	541.9	545	**545.0**	**545.0**	**545.0**	544.2	543.7	544.1	544.6	544.6	542.9	**545.0**	544.6	544.8	544.2	543.7	544.1	544.6	542.5	541.2	479.3	541.1	**544.8**	474.0	542.2	544.1	434.2	521.1	541.2
	flat75	820.5	831.8	840	**839.4**	838.9	839.0	836.7	836.7	836.8	837.2	837.8	833.4	**839.4**	837.9	838.9	836.7	836.2	836.0	837.2	834.6	828.9	733.4	832.0	**838.9**	727.6	834.3	836.0	669.9	801.7	828.9
	flat100	1091.0	1104.3	1117	1115.1	**1115.8**	1114.2	1111.4	1112.5	1111.7	1111.1	1112.6	1106.1	**1115.1**	1114.2	1114.1	1111.4	1111.5	1110.2	1111.1	1109.2	1098.7	971.9	1106.3	**1114.1**	954.8	1109.3	1110.2	890.5	1065.6	1098.7
	flat125	1370.8	1385.3	1403	1399.1	**1400.8**	1399.1	1394.2	1397.0	1366.6	1393.2	1396.5	1384.7	**1399.1**	1398.3	**1399.1**	1394.2	1395.0	1345.9	1393.2	1389.7	1372.1	1222.3	1389.1	**1399.1**	1197.1	1391.7	1345.9	1115.2	1333.9	1372.1
	flat150	1641.0	1657.2	1680	1675.5	**1675.6**	1674.0	1668.8	1671.7	1621.9	1668.5	1671.4	1642.3	**1675.5**	1671.5	1673.9	1668.8	1667.7	1593.0	1668.5	1664.4	1614.3	1463.0	1658.5	**1673.9**	1430.8	1662.2	1593.0	1331.8	1598.3	1614.3
	flat175	1906.3	1923.6	1951	**1945.4**	1944.3	1942.9	1935.9	1941.1	1795.3	1936.7	1938.5	1874.7	**1945.4**	1938.9	1942.8	1935.9	1935.9	1712.6	1936.7	1929.5	1829.9	1698.4	1924.4	**1942.8**	1657.9	1929.0	1712.6	1543.0	1850.3	1829.9
	flat200	2184.8	2204.1	2237	**2230.2**	2228.1	2228.3	2217.7	2226.3	2045.4	2221.0	2215.2	2141.2	**2230.2**	2221.9	2228.3	2217.7	2220.3	1944.2	2221.0	2201.1	2090.5	1949.7	2206.3	**2228.3**	1901.1	2212.3	1944.2	1766.0	2109.0	2090.5
	anomaly	252.4	261.0	261	**261.0**	**261.0**	**261.0**	**261.0**	260.3	**261.0**	**261.0**	**261.0**	260.1	**261.0**	**261.0**	260.8	**261.0**	260.3	**261.0**	**261.0**	**261.0**	259.5	232.5	259.1	260.8	242.1	257.4	**261.0**	214.0	255.5	259.5
	medium	938.0	952.2	953	**953.0**	952.4	952.2	952.6	951.1	**953.0**	952.5	951.5	949.5	**953.0**	952.4	952.1	952.6	951.1	**953.0**	952.5	951.1	948.0	872.4	947.8	952.1	875.2	948.4	**953.0**	790.3	924.5	948.0
	bw_large.a	4637.4	4654.7	4675	4668.2	**4670.5**	4668.4	4661.7	4669.1	4650.6	4648.8	4647.2	4499.6	4668.2	**4669.8**	4668.4	4661.7	4664.3	4635.8	4648.8	4628.1	4404.6	4296.3	4657.4	**4668.4**	4258.6	4658.2	4635.8	3835.2	4468.8	4404.6
	huge	7015.6	7032.0	7054	7045.7	7044.4	7046.8	7038.2	**7047.4**	7030.0	7026.4	7023.9	6849.6	7045.7	7036.8	**7046.5**	7038.2	7041.4	7014.4	7026.4	7003.5	6717.5	6547.2	7015.0	**7046.5**	6468.2	7034.5	7014.4	5742.9	6793.9	6717.5
	ais6	578.6	581.0	581	**581.0**	580.8	**581.0**	**581.0**	580.1	580.9	**581.0**	580.6	579.9	**581.0**	580.7	**581.0**	**581.0**	580.1	580.8	**581.0**	579.6	578.4	537.5	579.9	**581.0**	550.8	578.1	580.8	480.4	564.2	578.4
	ais8	1516.8	1519.8	1520	**1519.7**	1519.2	1519.2	1519.2	1518.9	1519.0	1519.1	1518.3	1513.9	**1519.7**	1519.2	1518.7	1519.2	1518.9	1519.0	1519.1	1516.2	1497.9	1424.9	1516.2	1518.7	1436.9	1516.4	**1519.0**	1255.3	1464.3	1497.9
	ais10	3146.7	3150.0	3151	**3150.0**	**3150.0**	3149.8	**3150.0**	3149.9	3149.2	3148.9	3147.7	3118.0	**3150.0**	**3150.0**	3149.4	**3150.0**	3149.7	3149.0	3148.9	3143.0	3026.5	2963.8	3143.0	**3149.4**	2960.2	3148.4	3149.0	2611.0	3026.9	3026.5
	ais12	5660.4	5664.1	5666	5664.1	**5664.5**	5664.1	5664.4	5664.0	5663.3	5662.0	5657.7	5559.5	5664.1	5664.1	5664.0	**5664.4**	5662.2	5662.5	5662.0	5644.6	5395.8	5350.3	5655.1	**5664.0**	5317.7	5660.3	5662.5	4704.4	5436.0	5395.8
	sw100-8-lp0	3082.0	3093.2	3100	3099.3	3099.2	**3099.8**	3098.3	3098.9	2924.6	3093.3	3089.6	2959.1	3099.3	3094.9	**3099.8**	3098.3	3093.7	2774.1	3093.3	3074.8	2863.2	2814.8	3080.5	**3099.8**	2759.4	3087.2	2774.1	2470.7	2947.6	2863.2
	sw100-8-lp1	3080.3	3092.9	3100	3099.7	**3100.0**	3099.7	3098.5	3099.3	3008.2	3093.6	3089.0	2963.3	**3099.7**	3098.3	3099.6	3098.5	3093.9	2932.8	3093.6	3073.5	2868.9	2803.0	3084.9	**3099.6**	2757.3	3087.9	2932.8	2502.8	2949.1	2868.9
	sw100-8-lp2	3077.8	3089.9	3100	3097.2	3096.5	**3099.0**	3096.1	3098.2	3010.3	3089.3	3087.1	2955.3	3097.2	3091.6	**3098.9**	3096.1	3093.1	2933.2	3089.3	3070.7	2857.1	2813.7	3077.8	**3098.9**	2759.6	3086.5	2933.2	2486.9	2942.0	2857.1
	sw100-8-lp3	3074.7	3087.0	3100	3093.7	3096.6	**3097.1**	3091.1	3096.0	3075.6	3086.6	3079.2	2954.0	3093.7	3093.0	**3096.8**	3091.1	3091.0	3058.2	3086.6	3060.9	2857.9	2813.8	3078.5	**3096.8**	2762.6	3084.7	3058.2	2503.4	2935.4	2857.9
	sw100-8-lp4	3071.9	3083.8	3100	3090.9	3091.5	**3094.3**	3087.5	3093.6	2949.1	3082.0	3081.6	2953.7	3090.9	3086.7	**3094.2**	3087.5	3089.1	2825.5	3082.0	3065.3	2858.5	2820.4	3070.2	**3094.2**	2763.5	3082.2	2825.5	2488.6	2942.3	2858.5
	uf50	212.8	217.6	218	**218.0**	**218.0**	**218.0**	**218.0**	217.7	**218.0**	**218.0**	**218.0**	**218.0**	**218.0**	**218.0**	**218.0**	**218.0**	217.7	**218.0**	218.0	218.0	217.0	203.3	216.8	**218.0**	206.7	215.9	**218.0**	199.2	215.4	217.0
	uf100	418.8	428.0	430	429.9	429.9	429.9	429.4	428.1	**430.0**	429.5	429.1	429.0	**429.9**	429.8	429.7	429.4	428.1	429.7	429.5	429.0	426.6	400.7	427.1	**429.7**	399.3	426.7	429.7	394.0	424.9	426.6
	uf150	629.6	642.0	645	**645.0**	644.6	**645.0**	644.0	643.7	644.9	644.9	644.5	644.1	**645.0**	644.6	644.6	644.0	643.6	644.7	644.9	644.3	642.6	598.0	641.5	644.6	595.7	642.2	**644.7**	591.2	637.4	642.6
	uf200	838.0	851.2	860	858.9	858.7	**859.0**	856.7	857.4	857.6	857.7	856.7	856.5	858.9	858.7	**859.0**	856.7	857.4	857.6	857.7	855.7	855.4	793.1	854.7	**859.0**	788.1	856.3	857.6	782.5	847.3	855.4
	uf250	1043.4	1058.0	1065	1064.0	1064.4	**1064.5**	1062.3	1062.5	1062.9	1063.6	1063.8	1062.8	1064.0	**1064.2**	**1064.2**	1062.3	1062.4	1062.7	1063.6	1063.7	1061.8	986.6	1060.2	**1064.2**	972.9	1061.1	1062.7	979.6	1052.7	1061.8
	uuf50	210.9	217.0	217	**217.0**	**217.0**	**217.0**	**217.0**	216.5	**217.0**	**217.0**	**217.0**	**217.0**	**217.0**	**217.0**	**217.0**	**217.0**	216.4	**217.0**	**217.0**	**217.0**	216.4	200.7	216.4	**217.0**	205.6	214.7	**217.0**	201.5	214.3	216.4
	uuf100	419.0	427.2	429	**429.0**	426.8	**429.0**	428.6	427.8	428.5	428.7	428.2	428.1	**429.0**	426.4	428.8	428.6	427.7	428.4	428.7	427.7	426.8	395.2	424.1	**428.8**	398.0	426.2	428.4	393.3	423.6	426.8
	uuf150	631.5	640.7	644	**643.9**	643.4	**643.9**	643.1	642.3	643.2	**643.9**	643.2	643.0	**643.9**	643.4	643.7	643.1	642.3	643.1	**643.9**	643.1	642.1	593.3	639.9	**643.7**	594.8	640.8	643.1	593.1	636.3	642.1
	uuf200	836.3	849.7	858	**857.4**	857.3	856.4	854.8	854.5	855.0	856.3	856.0	855.2	**857.4**	856.5	856.2	854.8	854.5	854.9	856.3	855.2	853.6	791.0	851.1	**856.2**	785.2	853.1	854.9	787.2	847.8	853.6
	uuf250	1041.0	1054.6	1064	1062.7	1062.7	**1062.8**	1059.0	1061.0	1059.8	1061.2	1061.6	1061.0	**1062.7**	1062.6	1062.6	1059.0	1060.8	1059.6	1061.2	1060.4	1059.7	981.7	1056.6	**1062.6**	970.8	1059.7	1059.6	981.4	1051.5	1059.7

Chapitre 5

Analyses comportementales

Ce dernier chapitre dédié à des analyses additionnelles est divisé en deux parties : une première consacrée aux composants de la génération d'algorithmes, et une seconde aux fonctions d'évaluation et algorithmes générés.

5.1 Analyses des composants de la génération d'algorithmes

Tous les algorithmes de génération (voir chapitre 3) que l'on a développés au cours de la thèse reposent sur les mêmes composants. En premier lieu, nous étudierons les effets des mutations sur les solutions renvoyées par les algorithmes générés. Ensuite, nous nous concentrerons sur la précision de l'évaluation des algorithmes dans un contexte de sélection. La sélection en elle-même a déjà été étudiée dans le chapitre 4. Enfin, nous étudierons les modèles d'algorithmes stochastiques et déterministes, avec leurs effets sur les résultats obtenus, notamment l'impact de la solution de départ sur les algorithmes déterministes à départ unique.

5.1.1 Effets des mutations

Le rôle du processus de génération est naturellement d'être en mesure de trouver un algorithme efficace. Les mutations effectuées doivent donc avoir un effet sur le comportement des algorithmes. Afin d'évaluer les effets des mutations, nous proposons d'analyser les solutions renvoyées par les algorithmes générés et par leurs mutations. Pour un algorithme d'optimisation, et a fortiori pour un hill climber, seule la dernière solution atteinte importe. Les solutions renvoyées par les algorithmes générés ont cette particularité d'être des optima locaux de la fonction d'évaluation utilisée pour guider la recherche locale. Notre première analyse consiste à estimer les effets de la mutation des fonctions d'évaluation, plus particulièrement sur les optima locaux de ces dernières.

Protocole expérimental

Nous limitons notre analyse aux modèles de fonction NK — seul modèle parmi les trois proposés qui décrive des paysages de fitness non neutres — et au modèle d'algorithmes stochastiques. Une instance unique — l'instance $NK_{128,1}$ — sera utilisée pour faciliter l'analyse et la visualisation des résultats.

Nous nous intéressons aux effets de la mutation tout au long de la recherche. Pour ce faire, nous utilisons l'algorithme de génération (1,λ)-ES avec les paramètres de la section 4.3 pour 10 exécutions indépendantes. Pour chaque exécution, nous extrayons 10 fonctions prises à différents stades du processus évolutionnaire. Soit f_j^i la j-ème fonction retournée de l'exécution i. La fonction f_1^i correspond à la fonction aléatoire initiale de l'exécution i, et la fonction f_{10}^i à la meilleure fonction trouvée lors du processus d'évolution. Les fonctions intermédiaires f_j^i correspondent aux meilleures fonctions alors atteintes lors de l'exécution i après $\lfloor \frac{(j-1)}{9} b_i \rfloor$ générations, b_i étant le nombre de générations nécessaires pour trouver la meilleure fonction f_{10}^i.

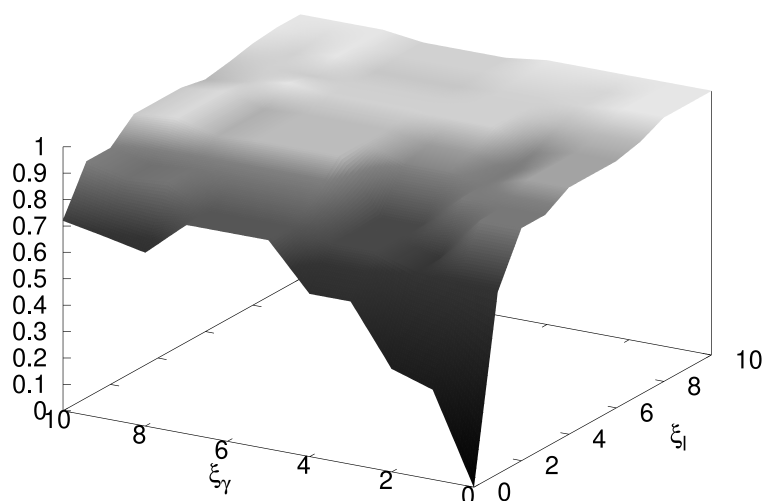
Sur chacune des 100 fonctions que l'on extrait des exécutions de l'algorithme de génération nous itérons un hill climber pour identifier 10^6 optima locaux non nécessairement distincts. La mutation des modèles NK requiert deux paramètres : ξ_γ pour la mutation de la matrice des contributions au score et ξ_l pour la mutation des liens. Nous ajustons dans cette analyse la signification de ces paramètres de mutation afin qu'ils ne soient plus associés à une probabilité de changement de chaque valeur mais à un nombre de changements effectifs ; par exemple $\xi_\gamma = 1$ et $\xi_l = 0$ consiste à modifier aléatoirement une valeur de contribution et aucune référence à un lien). Pour chaque couple de valeur (ξ_γ, ξ_l), 100 mutations sont générées à partir de chacune des fonctions extraites. Nous déterminons alors le nombre d'optima locaux de f_j^i qui ne sont plus optimaux après mutation, ce qui quantifiera l'effet de cette mutation sur le hill climber associé.

Résultats

La figure 5.1 présente les résultats obtenus par le protocole décrit ci-avant sur l'ensemble des 100 fonctions f_j^i extraites à différents stades de différentes exécutions. Nous pouvons laisser le cas particulier $(\xi_\gamma, \xi_l) = (0, 0)$, où la fonction mutée est la même que la fonction initiale. Pour les autres valeurs de paramètres les effets mesurés vont de 34% pour le paramètre $(1, 0)$ jusqu'à près de 100% pour les paramètres où $\xi_l \geq 9$.

La figure 5.2 présente les mêmes résultats mais pour certains stades d'évolution des fonctions d'évaluation (aléatoire, intermédiaire et évoluée respectivement sur les figures 5.2a, 5.2b et 5.2c). Nous pouvons voir que les effets de la mutation sont similaires indépendamment de la qualité des fonctions que l'on doit muter. Les figures A14 et A15 de l'annexe présentent l'ensemble des

FIGURE 5.1 – Proportion des optima changés (fonctions $NK_{128,1}$)



résultats.

5.1.2 Effets de la précision de l'évaluation

Dans cette section, nous voulons étudier les effets de la précision de l'évaluation sur les performances globales de l'algorithme de génération. Il existe en effet un compromis à trouver entre la précision de l'évaluation de chaque fonction et le budget total alloué pour la génération de l'algorithme.

Protocole expérimental

Nous utilisons ici la méthode de sélection des mutations basée sur un budget d'appels à la fonction objectif uniformément distribué entre les mutants. Chaque climber utilisant une fonction d'évaluation mutante est exécuté le même nombre de fois ; la fonction sélectionnée est celle dont les optima locaux atteints (au sens de la fonction d'évaluation) par les climbers ont en moyenne la meilleure valeur objectif (au sens donc de la fonction objectif).

Afin de montrer les effets bénéfiques d'une plus grande précision d'évaluation, nous utilisons la stratégie d'évolution (1+1) uniquement basée sur des comparaisons successives entre fonction courante et fonction mutée. Les instances et les valeurs de paramètres utilisées ici sont celles

FIGURE 5.2 – Proportion des optima changés, pour différents stades d'évolution des fonctions d'évaluation

(a) f_1^i (b) f_6^i (c) f_{10}^i

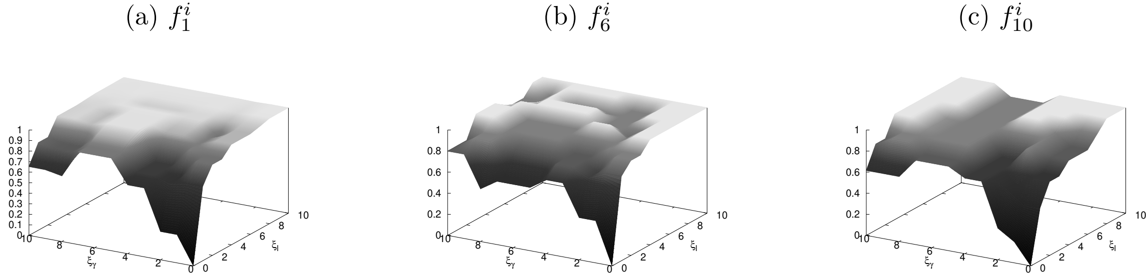


TABLE 5.1 – Performances estimées des fonctions générées avec la stratégie (1+1)-ES sur 28 instances en utilisant deux précisions d'évaluation différentes ($\kappa = 100$ et $\kappa = 1000$)

N	κ	K						
		1	2	4	6	8	10	12
128	100	**0.716**	**0.718**	**0.738**	**0.729**	**0.724**	**0.718**	**0.706**
	1000	0.714	0.717	0.737	0.723	**0.724**	0.714	0.704
256	100	**0.709**	**0.715**	**0.733**	**0.732**	**0.723**	**0.715**	**0.709**
	1000	0.702	0.710	0.725	0.727	0.719	0.710	0.707
512	100	**0.694**	**0.718**	**0.725**	**0.725**	0.710	0.697	0.651
	1000	0.692	0.715	0.718	0.721	**0.719**	**0.710**	**0.708**
1024	100	0.685	0.691	0.651	0.601	0.558	0.528	0.516
	1000	**0.699**	**0.713**	**0.721**	**0.719**	**0.711**	**0.705**	**0.697**

décrites dans le protocole de la section 4.3.

Nous comparons les climbers obtenus par la stratégie en employant deux paramètres de précisions : 100 exécutions par algorithme (voir section 4.3), puis 1000 exécutions. Comme les comparaisons se font à budget constant (10^8 appels à f_{obj}), la stratégie basée sur une évaluation moins précise est en contrepartie appliquée sur 10 fois plus de générations.

Résultats

Le tableau 5.1 présente l'ensemble des résultats obtenus pour chacune des 28 instances de problème NK.

Pour ces instances NK de taille $N \in \{128, 256\}$, nous remarquons que l'évaluation la plus

précise mène à de moins bons résultats que l'évaluation moins coûteuse. Il convient tout de même de noter que la différence entre les résultats des deux évaluations est faible, avec une différence maximale de 0.007. Ensuite, nous pouvons montrer les effets bénéfiques d'une évaluation plus précise. En effet, notamment pour les instances où N$\in \{512, 1024\}$, l'évaluation la moins précise ne permet pas de sélectionner efficacement de bonnes fonctions, ce qui mène à des résultats s'approchant d'une recherche aléatoire pour les instances les plus difficiles. La précision de l'évaluation doit donc être choisie avec soin en fonction des propriétés de l'instance mais dépend également du type de stratégie d'évolution et de la méthode de sélection, comme on a pu le voir au chapitre précédent avec les expérimentations de la stratégie (1,λ)-ES.

5.1.3 Pertinence des modèles de fonction

Les modèles utilisés pour construire les fonctions d'évaluation sont au cœur du processus d'évolution des algorithmes. Nous nous intéressons donc maintenant à l'influence des différents modèles sur les algorithmes générés.

Protocole expérimental

Effet de la neutralité du modèle Dans un premier temps, nous nous intéressons à l'effet de la neutralité induite par les modèles de fonction sur les performances des algorithmes générés. Pour cela, nous comparons les résultats obtenus sur deux instances du problème NK et avec des modèles de fonction MAX-3SAT dont les paysages de fitness possèdent un certain niveau de neutralité.

On a pu constater que le modèle MAX-3SAT rencontrait des difficultés à trouver de bons algorithmes sur certaines instances NK (voir tableau 4.15). Dans la présente analyse, nous faisons figurer deux instances, une qui n'a pas posé de difficultés particulières à la génération ($NK_{128,12}$), et une autre où la génération n'a pas su trouver de bons algorithmes ($NK_{1024,12}$). Pour générer de nouveaux algorithmes, nous utilisons la stratégie $(1, \lambda)$-ES et le modèle des hill climbers déterministes à points de départ multiples (voir section section 4.6 pour le paramétrage). Le seul paramètre que nous faisons varier ici est le nombre de clauses utilisées par le modèle de fonction d'évaluation. En effet, le nombre de clauses a un impact sur la neutralité du paysage de fitness décrit au moyen d'une fonction MAX-3SAT. Plus la fonction comporte de clauses, moins le paysage possède de neutralité.

Nous générons différentes fonctions avec différents nombres de clauses entre $5N$ et $100N$ (N est la taille du problème). Nous évaluons alors la fonction retournée à partir des solutions de départ de ces algorithmes déterministes, puis la neutralité des modèles en estimant la proportion de voisins neutres par échantillonnage de l'espace de recherche.

Comparaison des modèles Dans cette seconde étape, nous utilisons l'instance $\mathrm{NK}_{128,1}$ sur laquelle aucun modèle n'a rencontré de difficulté particulière. Nous générons cette fois des algorithmes stochastiques (voir section 4.6 pour le paramétrage), avec comme objectif de visualiser l'évolution de la qualité des algorithmes sélectionnés tout au long de la recherche ainsi que le nombre moyen d'itérations effectuées par les hill climbers guidés par ces fonctions générées.

Résultats

Effet de la neutralité du modèle La figure 5.3 permet de comparer l'efficacité de différents modèles de fonctions MAX-3SAT utilisant un nombre de clauses différentes (axe des abscisses, en fonction de N) pour résoudre deux instances NK. La neutralité comparée des fonctions est indiquée en pointillé. Les taux de neutralité (indiqués sur l'axe de droite) sont très élevés et négativement corrélés à la qualité des fonctions d'évaluation.

Nous pouvons remarquer que certaines générations atteignent de bons algorithmes sur l'instance $\mathrm{NK}_{128,1}$, notamment à partir du nombre de clauses 40×128. Au contraire, pour l'instance $\mathrm{NK}_{1024,12}$ aucune valeur de nombre de clauses n'a été suffisant pour accéder à de bonnes fonctions. Or pendant que la neutralité baisse légèrement à mesure que l'on ajoute des clauses pour l'instance $\mathrm{NK}_{128,1}$, la neutralité pour l'instance $\mathrm{NK}_{1024,12}$ est au départ plus haute pour le même nombre de clauses par rapport à la taille du problème, et ne semble pas diminuer avec des clauses supplémentaires. Un tel taux de neutralité indique que le paysage est composé essentiellement de grands plateaux sur lesquels les hill climbers vont rapidement se bloquer. Même si la capacité de marche du $(1, \lambda)$-ES permet habituellement de contourner la problématique des optimums locaux en modifiant la fonction et donc le paysage, elle se révèle impuissante pour affecter suffisamment les paysages neutres induits par ce modèle, ce qui explique la faible efficacité du modèle MAX-3SAT sur les instances de grande taille.

Comparaison des modèles La figure 5.4 compare l'efficacité des algorithmes en cours de recherche selon les modèles, et la longueur de leurs trajectoires. Nous pouvons y voir la supériorité du modèle NK sur les deux autres. Le modèle MAX-3SAT et celui de somme de fonctions pseudo-booléennes obtiennent des résultats similaires en termes de score atteint. Le modèle MAX-3SAT semble ne pas réussir à se maintenir dans des régions favorables du paysage de fitness. La longueur des trajectoires des hill climbers est positivement corrélée à leur évaluation. Ceci nous montre l'importance de la flexibilité des modèles ; en effet, un modèle dont il plus simple d'allonger les exécutions de hill climbers associées pourra plus facilement atteindre de bonnes solutions lointaines et donc améliorer la qualité du hill climber.

FIGURE 5.3 – Scores atteints et neutralité des modèles en fonction du nombre de clauses qui les composent

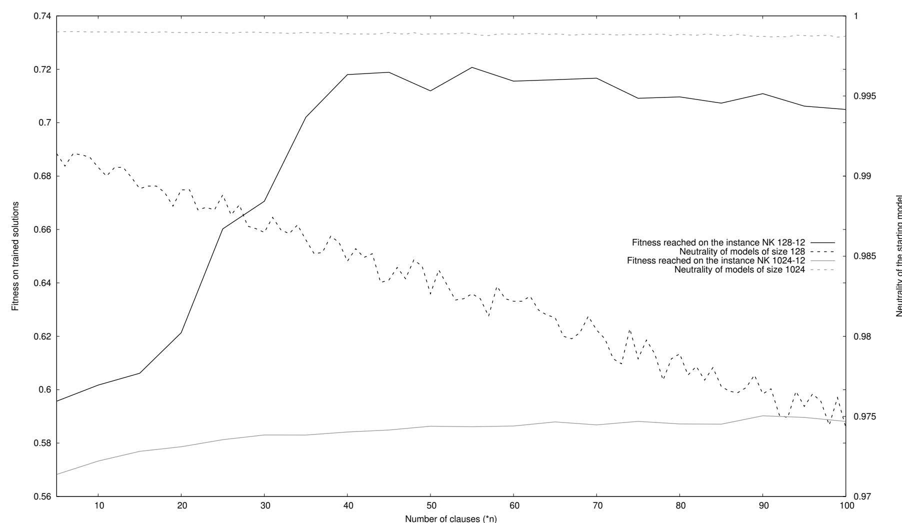
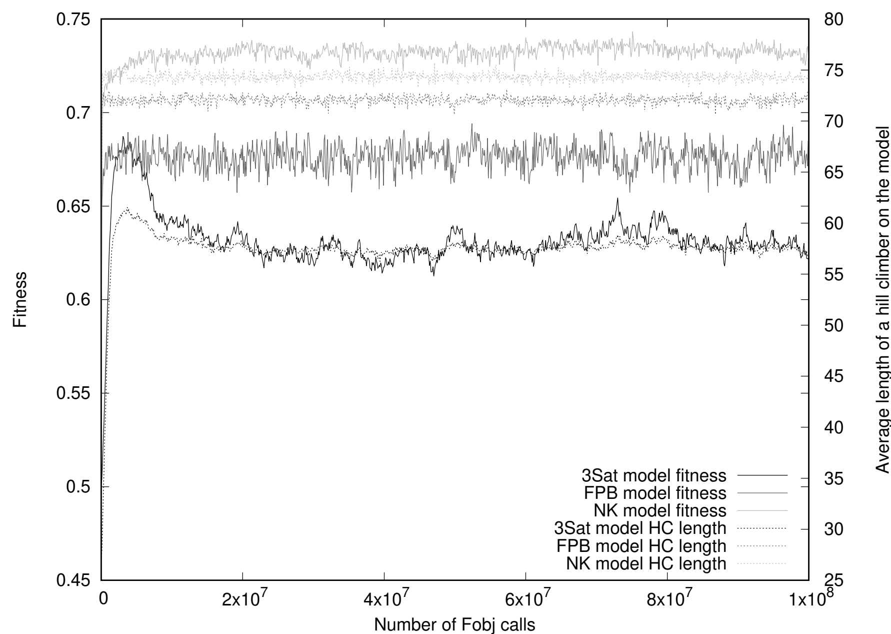


FIGURE 5.4 – Évolution des scores et du nombre d'itération des hill climbers générés au cours de leur génération



5.1.4 Comparaison des algorithmes déterministes et stochastiques

Protocole expérimental

Nous nous intéressons ici à l'aspect déterministe ou stochastique des algorithmes générés. Le modèle déterministe à départs multiples fixés a ceci de particulier que plus il y a de solutions dans son ensemble de départ, plus la stratégie d'évolution affectera le paysage dans son ensemble, à la manière de l'algorithme stochastique. Dans cette étude, nous mesurons l'effet de la variation de la taille de l'ensemble de solutions de départ afin d'analyser ces deux modèles d'évolution des climbers au moyen de la fonction d'évaluation : axée paysage d'une part, et trajectoire d'autre part. L'évolution du paysage au moyen de la fonction d'évaluation prend tout son sens avec la génération d'algorithmes stochastiques. La présente analyse des algorithmes déterministes à départs multiples fixés permet alors d'établir une analyse comparative entre les modèles d'algorithmes déterministe et stochastique.

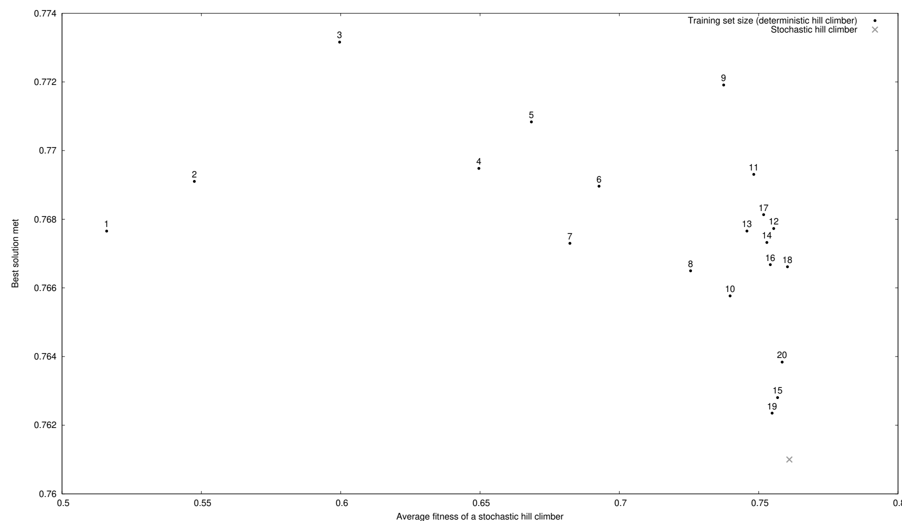
Nous proposons deux protocoles. Le premier s'effectue sur l'instance $NK_{128,12}$ et reprend les paramètres utilisés dans l'expérimentation étendue (voir section 4.6), à l'exception du paramètre de la taille de l'ensemble de départ que nous faisons varier entre 1 et 20. Nous représentons alors graphiquement les moyennes des résultats obtenus par les algorithmes générés. Le premier axe représentera les scores des meilleures solutions rencontrées, et le second le score moyen des solutions renvoyées par les algorithmes générés à partir de solutions aléatoires.

Pour le deuxième protocole, nous nous intéressons au potentiel effet négatif de la taille de l'ensemble de départ sur les performances obtenues par les algorithmes générés. Pour ce faire nous générons de nouveau plusieurs algorithmes avec différentes tailles d'ensembles de départ. Nous utilisons deux instances NK ($NK_{128,12}$ et $NK_{1024,12}$) et l'instance MAX-SAT sw100-8-lp0. Pour les instances NK nous employons le modèle MAX-3SAT qui était le seul à avoir des difficultés à générer de bons algorithmes sur les problèmes NK de grande taille. Pour l'instance MAX-SAT nous utilisons le modèle de somme de fonctions pseudo-booléennes qui trouvait les mêmes difficultés que le modèle MAX-3SAT sur certaines instances MAX-SAT. Pour le reste nous utilisons les mêmes paramètres qu'en section 4.6. Enfin, nous représentons graphiquement les scores obtenus à partir des solutions de départ de l'ensemble $\mathcal{X}_0$ des différents algorithmes en fonction du nombre de solutions de départ.

Résultats

Sur la figure 5.5, où chaque point représente une taille d'ensemble de départ et indique la moyenne des scores des algorithmes déterministes correspondants (meilleure solution avec points de départs fixés sur l'axe des ordonnées, performance d'un hill-climber stochastique sur

FIGURE 5.5 – Performance comparée des hill climbers déterministes à points de départ multiples selon leurs meilleurs scores obtenus (en ordonnée) et climbers stochastiques utilisant les fonctions d'évaluation associées à partir de points de départ aléatoires (en abscisse)



le paysage associé sur l'axe des abscisses), nous pouvons constater qu'une plus grande taille d'ensemble de départ ne mène pas nécessairement à de moins bonnes solutions rencontrées. En effet, les ensembles de taille 2 à 6 ont tous rencontrés de meilleures solutions que celui dont la taille est de 1 (correspondant au hill climber déterministe à point de départ unique fixé). Ceci peut être expliqué par le fait qu'un plus grand nombre de points implique une exploration plus importante de l'espace de recherche, ce qui aide la recherche. Néanmoins, cette aide reste limitée car pour les plus grandes tailles d'ensemble les meilleures solutions trouvées deviennent moins bonnes. En parallèle, l'augmentation du nombre de solutions de départ est bénéfique pour la construction d'un algorithme robuste. En effet, avec plus de solutions de départ nous avons de meilleurs résultats à partir de solutions aléatoires. Nous pouvons d'ailleurs remarquer que certaines tailles d'ensemble dominent la génération d'algorithmes stochastique. C'est à dire, obtiennent de meilleurs résultats en moyenne sur des solutions aléatoires, tout en rencontrant de meilleures solutions au cours de la génération.

Les figures 5.6 et 5.7 comparent les scores obtenus par les hill climbers déterministes à points de départ multiples fixés, respectivement sur deux instances NK difficiles ($NK_{128,12}$ et $NK_{1024,12}$) et sur l'instance MAX-SAT sw100-8-lp0. Nous pouvons visualiser l'effet de la taille de l'ensemble de points de départ, différent selon les instances. Pour l'instance NK la plus petite ($NK_{128,12}$), la taille de l'ensemble n'affecte pas les résultats. Pour l'instance de grande taille, il apparaît clairement qu'un nombre plus important de solutions de départ détériore les performances des algorithmes générés, phénomène que l'on constate également sur l'instance MAX-SAT avec une

FIGURE 5.6 – Scores obtenus en fonction du nombre de solutions de départ sur les problèmes $NK_{128,1}$ et $NK_{1024,1}$

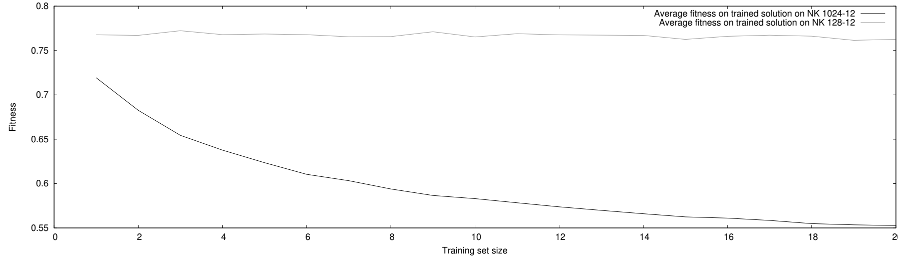
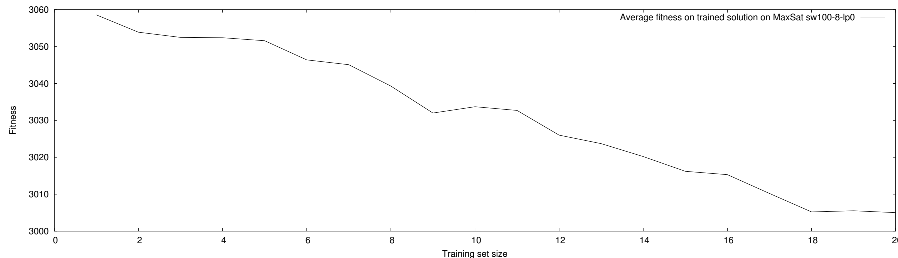


FIGURE 5.7 – Scores obtenus en fonction du nombre de solutions de départ sur le problème sw100-8-lp0



détérioration plus progressive. Il est en fait probable que ce problème ne soit qu'un manque de budget pour la convergence de la génération, bien que celui qui était alloué dans le cadre de cette expérience était déjà conséquent. Ainsi, le budget nécessaire pour la convergence de l'algorithme est sans doute trop grand pour générer des algorithmes en temps raisonnable. Nous pouvons voir que tous les modèles ne sont pas équivalents sur ces problématiques. Il apparaît, de fait, que de la neutralité dans le paysage généré est un obstacle à l'évolution des fonctions. Que ce soit un problème dans la mutation de ces modèles ou un problème d'évaluation des plateaux, il reste clair que si la neutralité dans les modèles doit être évitée dans la mesure du possible.

Pour revenir sur la comparaison des algorithmes déterministes et stochastiques, les résultats que l'on obtient avec nos différents protocoles nous montrent d'abord le spectre des différents algorithmes possibles à générer avec chacun leurs avantages et défauts. La stratégie de génération employée permet d'identifier un algorithme déterministe très simples dont les résultats surpassent en tous points ceux des algorithmes stochastiques. Nous avons également pu voir que le modèle d'algorithmes avec plusieurs points de départ pouvait être bénéfique à la génération d'algorithmes déterministes et, en analysant leurs performances à partir de points de départ aléatoires, permettent d'obtenir des fonctions d'évaluation aux propriétés proches de celles générées au moyen d'un modèle d'algorithme stochastique.

5.1.5 Effet de la solution de départ pour les algorithmes déterministes à point de départ unique fixé

Dans cette section, nous analysons le modèle des algorithmes déterministes à départ unique et plus précisément sur l'effet de la solution de départ sur les propriétés des algorithmes générés.

Protocole expérimental

Ici, nous voulons évaluer la capacité de l'algorithme de génération à trouver des algorithmes déterministes en fonction de la solution de départ de ceux-ci, ainsi que son effet sur leurs propriétés. Pour ce faire, nous avons besoin de connaître l'optimum global du problème. Ainsi, nous décidons d'effectuer ce protocole sur l'instance $\text{NK}_{128,1}$ qui est non seulement la plus simple des problèmes NK, mais aussi une des seules parmi les instances NK utilisées dont nous soyons très confiant sur l'optimalité d'une solution. L'algorithme de génération utilisé est la stratégie (1,λ)-ES appliquée à la génération d'algorithmes déterministes (voir section 4.4 pour le paramétrage employé).

Nous générons plusieurs algorithmes avec différentes solutions de départ, que nous regroupons de départ en trois catégories. D'abord l'optimum global comme solution de départ noté s^*. Ensuite, une solution aléatoire comme nous le faisions déjà jusqu'ici. Enfin l'opposé (ou complémentaire) de l'optimum, noté $\overline{s^*}$, obtenu en flippant tous les bits de s^* ; ce point opposé du paysage est ainsi censé être particulièrement difficile pour débuter une trajectoire de hill climbing. Pour chaque algorithme déterministe généré avec chacun de ces différents points de départ, nous reportons sa qualité (la valeur objectif de la solution retournée), le nombre d'itérations effectuées par le hill climber généré, mais aussi ces mêmes informations dans le cas du même climber, utilisant la même fonction mais partant d'une solution aléatoire.

Résultats

Les résultats sont reportés dans le tableau 5.2. Les trois dernières colonnes contiennent les résultats obtenus avec les algorithmes générés selon la solution de départ utilisée (s^*, aléatoire, $\overline{s^*}$). Les algorithmes sont alors évalués soit depuis la solution fixée initialement comme composant du hill climber, soit depuis une solution aléatoire.

Le premier résultat que l'on peut remarquer est que peu importe la solution de départ, si l'algorithme généré est utilisé avec celle-ci, l'optimum global est toujours atteint. De fait, la solution de départ n'affecte donc pas la possibilité de générer de bons algorithmes. Et ce, même s'il est nécessaire de traverser intégralement l'espace de recherche, ou au contraire ne pas bouger depuis la solution de départ. Dans ce dernier cas de figure, la stratégie d'évolution doit construire

Table 5.2 – Résultats sur l'instance $NK_{128,1}$ avec différentes solutions de départ lors de la génération et de l'évaluation

solution de départ pour HC(f)		s_0 (paramètre de l'algorithme)		
		s^*	aléatoire	$\overline{s^*}$
s_0	score	0.725	0.725	0.725
	itérations	0	71	128
aléatoire	score	0.569	0.626	0.710
	itérations	50	58	73

une fonction d'évaluation telle que la solution de départ fixée devienne un optimum local.

Il est particulièrement intéressant de comparer les scores indiqués dans la partie inférieure, lorsque les algorithmes générés sont ensuite rééxécutés depuis des solutions aléatoires. En effet, plus la solution de départ fixée lors du processus évolutif est éloignée de la solution optimale, plus les solutions issues de nouvelles trajectoires définies depuis des solutions aléatoires ont une valeur objectif élevée, et des trajectoires longues. Cela indique que les fonctions obtenues depuis la génération de hill climber déterministe dont le point de départ fixé est plus éloigné que le point d'arrivée espéré (et atteint) ont un paysage associé plus exploitable dans son ensemble.

Exploiter le paysage induit par la fonction générée est une propriété attendue lorsque l'évaluation du paysage, et donc le processus évolutionnaire, se fait au moyen du modèle d'algorithmes stochastiques. Or même l'emploi du modèle déterministe et d'un point de départ fixé même unique permet de façonner un paysage suffisamment exploitable à un algorithme stochastique, du moment que ce point de départ soit suffisamment éloigné des zones d'intérêt du paysage originel décrit par la fonction objectif.

5.2 Analyses des fonctions et algorithmes générés

Cette seconde partie d'analyse est dédiée à l'étude plus fine des fonctions générées et du comportement des algorithmes associés. Une nouvelle fois, nous présentons plusieurs protocoles dédiés à chaque fois à un point d'analyse particulier : évolution des optima locaux puis des trajectoires d'algorithmes au cours de la génération, épistasie comparée des fonctions d'évaluation et fonctions objectif, interactions comparées entre variables et contribution des sous-fonctions NK. Enfin, nous tenterons de visualiser les différentes fonctions et comportement d'algorithmes à partir de trajectoires.

5.2.1 Apparition, évolution, et destruction des optima locaux dans les fonctions générées

Dans cette section, nous nous concentrons sur l'évolution des optima locaux dans les fonctions générées au cours du processus évolutionnaire.

Protocole expérimental

Nous considérons de nouveau l'instance $\mathrm{NK}_{128,1}$ pour sa plus grande simplicité de résolution, afin de garder raisonnables les ressources nécessaires pour cette étude. L'évolution des fonctions/algorithmes est étudiée à travers leur génération par la stratégie d'évolution $(1, \lambda)$ avec les paramètres utilisés dans la section 4.3. Soient $f_0, \dots f_{100}$ les fonctions courantes extraites du processus de génération, telles que la fonction f_i correspond à la fonction courante à l'instant où $\frac{i}{100}B$ appels à la fonction objectif ont été effectués, B étant le budget total alloué pour la génération. Dans un premier temps, nous estimons le nombre d'optima locaux sur chaque fonction. Pour ce faire, nous exécutons 10^5 hill climbers sur chaque fonction d'évaluation, puis dénombrons le nombre d'optima distincts.

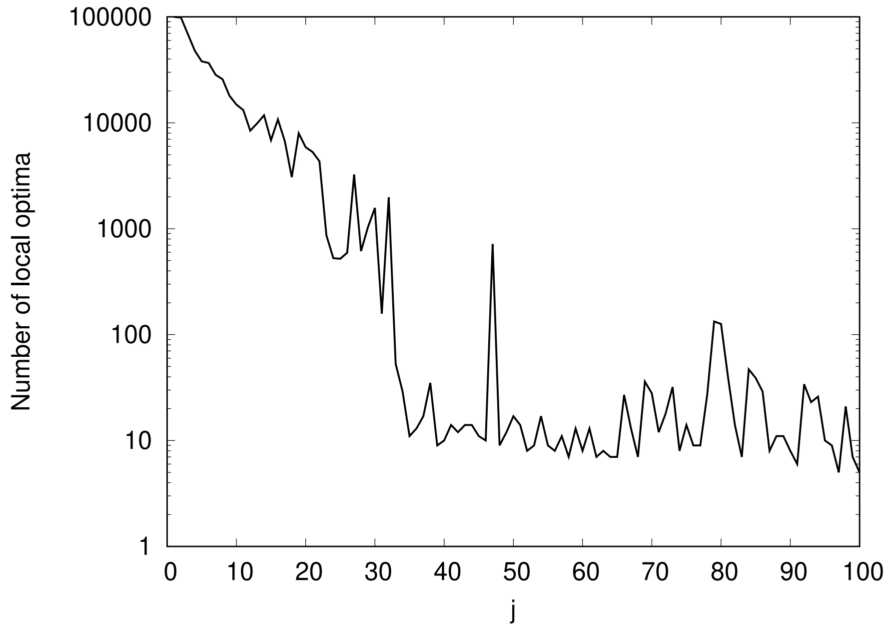
Parallèlement, nous collectons les informations issues de 100 exécutions d'un hill climber — classiquement effectuées au cours du processus de génération — sur chaque fonction courante : valeur objectif (fitness) des différents optima retournés, et fréquence d'apparition de chaque optimum.

Résultats

La figure 5.8 permet de suivre l'évolution du nombre d'optima locaux de la fonction d'évaluation courante au cours du processus de génération. L'axe des abscisses représente le budget dépensé pour la génération en pourcentage, l'axe des ordonnées le nombre d'optima différents obtenus à partir de 100000 exécutions d'un hill climber stochastique. Nous constatons clairement que le nombre d'optima locaux des fonctions d'évaluation chute au cours de la recherche puis se stabilise à quelques dizaines d'optimums au maximum. Ceci indique que la génération d'algorithmes tend à créer des paysages lisses, donc des fonctions d'évaluation simple à exploiter.

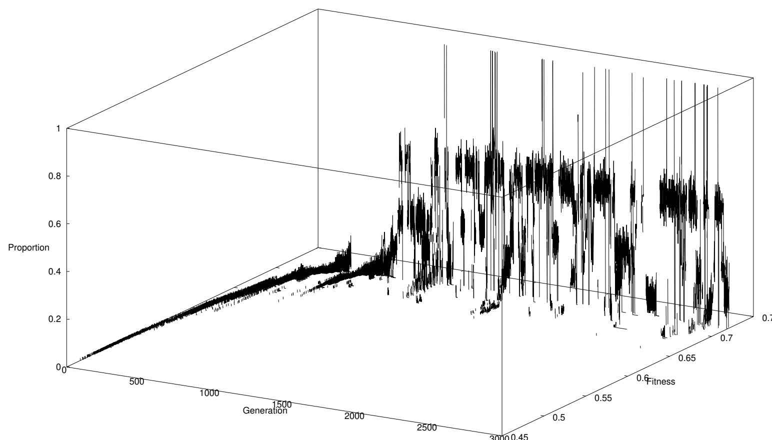
La figure 5.9 reporte l'évolution de la qualité de la fonction courante au moyen de la valeur objectif des optimums locaux calculés, mais aussi de l'apparition prolongée ou non des optimums d'une génération à l'autre. L'axe noté "Génération" est le nombre d'itérations passé pour la génération. L'axe noté "Fitness" représente la valeur objectif de chaque optimum, que l'on peut discriminer plus aisément au moyen de l'échelle de couleur. Enfin, l'axe noté "Proportion" est la fréquence de chaque optimum à une itération de la génération donnée. Pour analyser cette

FIGURE 5.8 – Évolution du nombre d'optima locaux de la fonction générée sur l'instance $NK_{128,1}$ calculé à partir de 100000 hill climbers



figure, il est important de bien comprendre comment elle a été construite. En effet, s'il semble n'y avoir qu'une seule courbe, en réalité à chaque optimum retourné dans la génération correspond une courbe. La figure A16 de l'annexe qui ne montre qu'une partie de la figure 5.9 montre bien cette multitude de courbes. Pour un optimum s retourné pour la première fois à l'itération i, le premier point de sa courbe commencera sur l'axe "Génération" à i itérations à la fitness $f_{\text{obj}}(s)$ et à la proportion à laquelle il est apparu. Le second point se placera de la même façon à l'itération $i + 1$. Lorsqu'un optimum n'est pas retourné à une itération et qu'il est encore un optimum local pour la fonction d'évaluation courante, la courbe continue à l'itération en question avec une proportion de 0 et la même fitness. La courbe s'arrête lorsque la solution n'est plus un optimum local pour la fonction d'évaluation courante. La figure est ainsi formée par l'ensemble de ces courbes représentant la durée de vie des optima. Nous pouvons voir qu'au début de la génération la fitness des optima augmente rapidement avec une proportion faible. Ensuite, quand l'amélioration de la fitness est plus difficile le nombre d'optima diminue, leur proportion d'apparition augmente et ils ont tendance à demeurer optimaux sur davantage d'itérations.

Figure 5.9 – Évolution des optima locaux de la fonction générée sur l'instance $NK_{128,1}$ en fonction du nombre d'itération de la génération, de la fitness des optima et de leur proportion d'apparition



5.2.2 Évolution des algorithmes déterministes

Nous venons de voir l'évolution des optima locaux dans les algorithmes stochastiques. Ici, nous nous intéressons à l'évolution des algorithmes déterministes, et à leur capacité à trouver l'optimum global de la fonction objectif.

Protocole expérimental

Ici, comme pour l'analyse de l'effet de la solution de départ sur les algorithmes déterministes, nous avons besoin de connaître l'optimum global de la fonction objectif ; nous poursuivons ainsi notre analyse sur l'instance $NK_{128,1}$. Un hill climber déterministe A produit une unique trajectoire $s_0, s_1, \ldots, s_{n-1}, s_n$, suite de solutions connectées de l'espace de recherche, puis retourne la dernière solution ($A(s_0) = s_n$). Nous cherchons ici à représenter ces trajectoires afin d'appréhender leur évolution, et donc le comportement de l'algorithme de génération (stratégie d'évolution (1,λ) avec les paramètres utilisés en section 4.4).

Nous proposons deux représentations de ces trajectoires. La première se concentre sur l'atteinte de l'optimum et sur l'évolution des trajectoires pour y arriver. Nous représentons une trajectoire comme une courbe où les points adjacents sont reliés. Chaque solution s_i de la trajectoire est représentée graphiquement par le point $(i, s_i \Delta s^*)$, $s_i \Delta s^*$ représentant la distance de Hamming entre s_i et l'optimum global s^*, c'est-à-dire la longueur de la plus courte trajectoire

permettant d'y accéder depuis s_i. Nous reportons alors plusieurs trajectoires, atteintes à différents moments de la génération. Cela nous permet d'étudier l'évolution des trajectoires depuis un point de départ fixé. Celui-ci peut être une solution aléatoire, ou bien la solution $\overline{s^*}$ opposée à l'optimum global.

La seconde représentation des trajectoires se concentre sur la comparaison des trajectoires générées à des trajectoires que l'on peut suivre avec un hill climber guidé par la fonction objectif. Pour cela, chaque solution s_i d'une trajectoire sera représentée graphiquement par le point $(s_i) = (s_i \Delta s^*, f_{\text{obj}}(s_i))$ et relié comme pour la représentation précédente aux points représentant s_{i-1} et s_{i+1}. Nous comparons alors les trajectoires de plusieurs exemples d'algorithmes déterministes générés à partir du point de départ $\overline{s^*}$, et de trajectoires de hill climbers stochastiques guidés par la fonction objectif, partant de $\overline{s^*}$ ou de points de départ aléatoires.

Résultats

Les figures 5.10 et 5.11 sont les résultats de la première représentation décrite. Les nombres indiqués correspondent au nombre d'améliorations ayant précédé la sélection de la fonction d'évaluation décrivant cette trajectoire (0 pour la trajectoire de départ, 5 pour la trajectoire menant à la cinquième amélioration enregistrée au cours du processus, *etc.*) De manière évidente les meilleures trajectoires sont celles qui s'arrêtent à l'optimum global (point d'ordonnée 0). Ici, une trajectoire parfaite, obtenue sans flipper deux fois le même bit, est une trajectoire en ligne droite entre la solution de départ et l'optimum. On peut remarquer qu'une telle trajectoire a été trouvée en partant de l'opposé de l'optimum (figure 5.10).

Une telle visualisation dans le cas d'un point de départ fixé aléatoirement (figure 5.11) montre naturellement les trajectoires différemment. En effet, la séquence de bits mutés dans les premiers pas de recherche, en particulier concernant la première trajectoire aléatoire (trajectoire 0), ne correspond pas une trajectoire directe vers l'optimum global, alors que c'est forcément le cas depuis $\overline{s^*}$ jusqu'à ce qu'un bit soit flippé une seconde fois. Le caractère déterministe du hill climber avec utilisation d'un ordre de parcours des voisins permet de prolonger ce phénomène. On observe d'autant plus sur la figure 5.11 que l'algorithme évolue progressivement de manière à emprunter une trajectoire quasi-directe vers l'optimum, en prolongeant la longueur de la trajectoire initiale que d'une dizaine de pas.

La figure 5.12 offre une représentation alternative de ces trajectoires. En comparant les trajectoires des hill climbers guidés par la fonction objectif et ceux des climbers générés, nous pouvons identifier des comportements très différents. Tout d'abord, la solution de départ influence notablement les hill climbers guidés par f_{obj}, qui sont moins efficaces depuis $\overline{s^*}$ (l'opposé de l'optimum, trajectoires représentées en pointillé) que depuis une solution aléatoire (trajec-

FIGURE 5.10 – Trajectoires des fonctions intermédiaires et finales générées pour l'instance $NK_{128,1}$ (avec la solution de départ l'opposé de l'optimum $\overline{s^*}$) projetées sur la distance de Hamming de l'optimum s^*

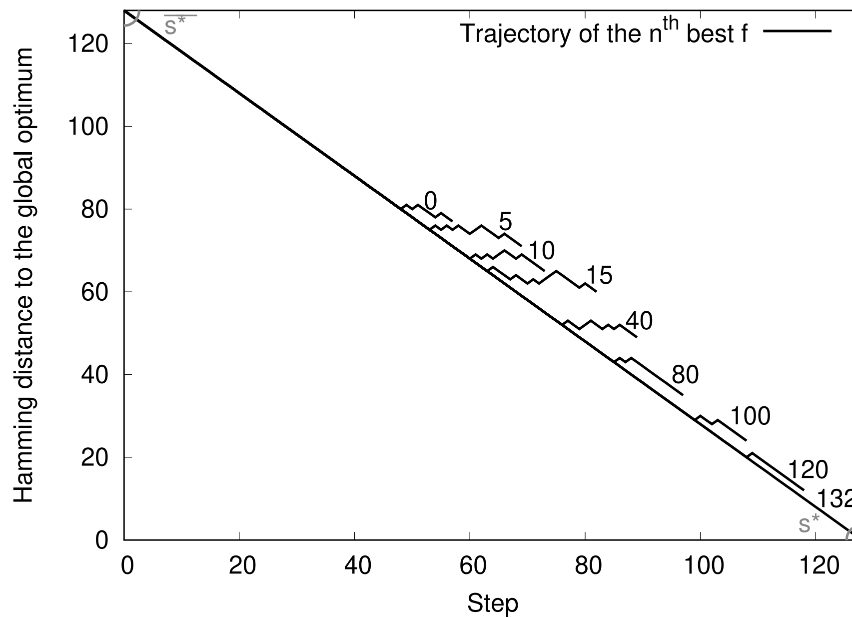


FIGURE 5.11 – Trajectoires des fonctions intermédiaires et finales générées pour l'instance $NK_{128,1}$ (avec une solution de départ aléatoire) projetées sur la distance de Hamming de l'optimum s^*

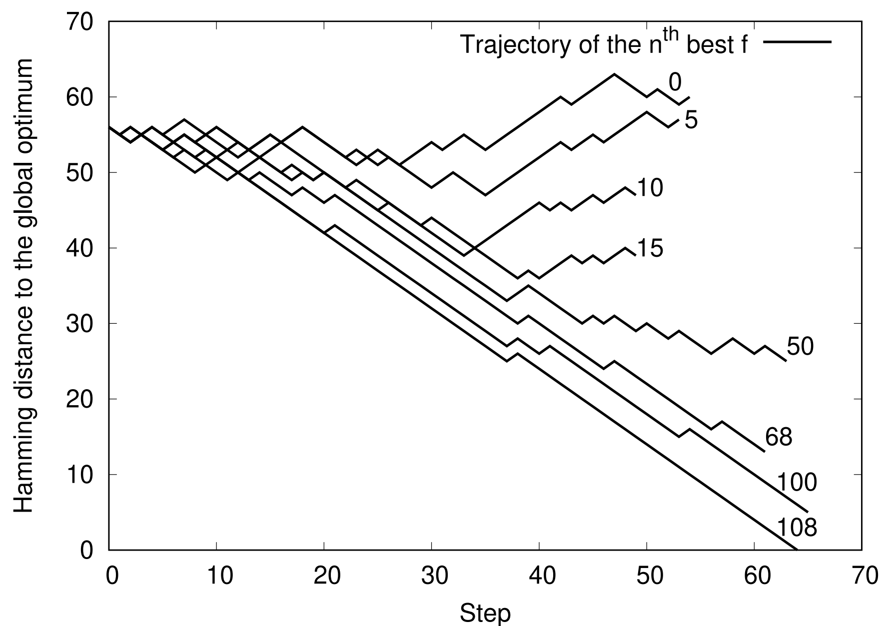
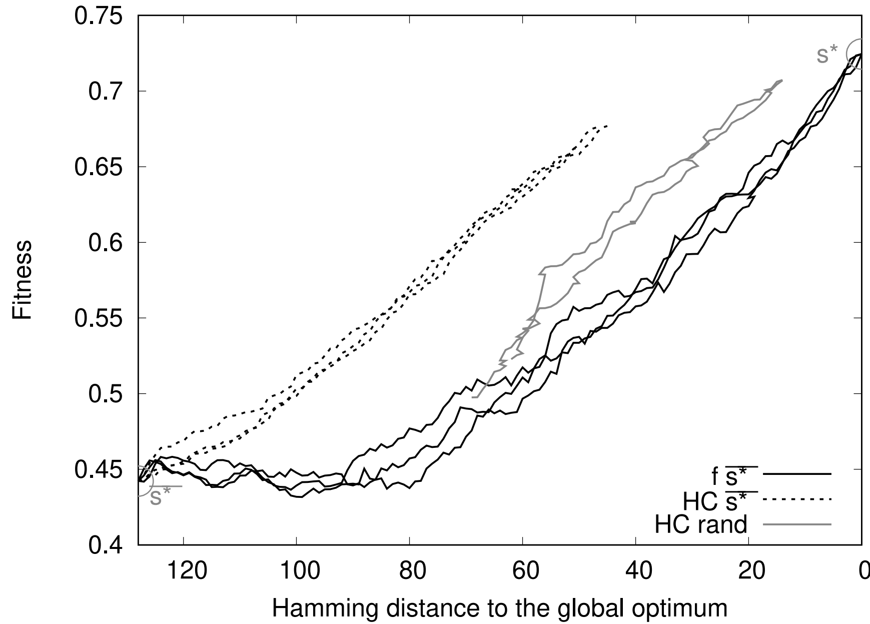


FIGURE 5.12 – Distance à l'optimum global s^* des fonctions générées (projetées sur f_{obj}) comparées aux trajectoires de hill climbers guidés par f_{obj} sur l'instance $\text{NK}_{128,1}$

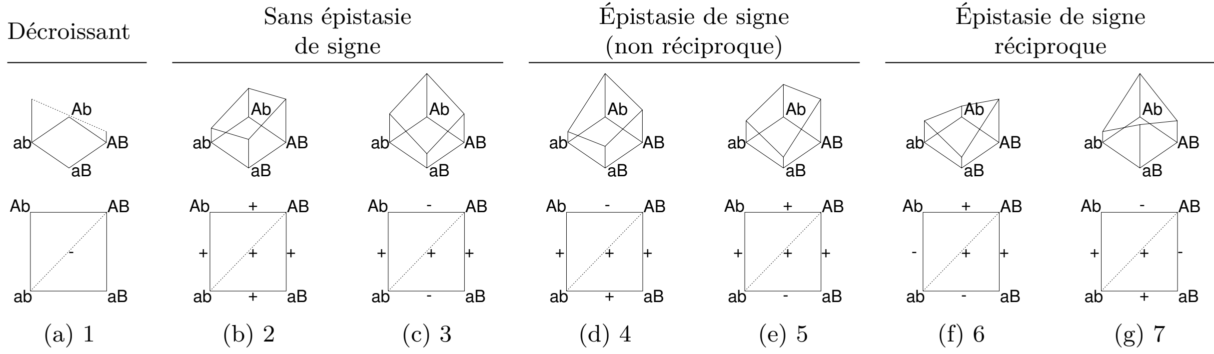


toires claires). Par définition, un hill climber guidé par la fonction objectif ne peut que grimper dans le paysage décrit par cette fonction, avec des trajectoires à la fitness croissante ; ce n'est pas le cas des trajectoires de climbers générés qui dont les trajectoires (en noir) peuvent décroître en termes de valeur objectif. C'est l'effet de l'épistasie de signe qui induit le fait que monter dans le paysage n'implique pas nécessairement de s'approcher de l'optimum. Nous pouvons de nouveau le voir sur les premières itérations des algorithmes générés où nous nous approchons de l'optimum sans augmenter la fitness des solutions. Au contraire, les hill climbers guidés par la fonction objectif montent dès les premières itérations mais s'arrêtent dans des régions sous-optimales. Les retours en arrière sur les trajectoires représentent un éloignement vis-à-vis de s^*, c'est-à-dire le flip d'un bit dont la valeur était celle de l'optimum.

5.2.3 Épistasie des fonctions d'évaluation générées

Dans cette section, nous analysons les types d'épistasie que l'on observe à différentes hauteurs dans les paysages alternatifs.

Figure 5.13 – Caractérisation de l'épistasie de signe : différentes configurations depuis deux solutions de référence distantes de 2 notées ab et AB.



Protocole expérimental

Cette expérience repose sur une analyse exhaustive des solutions de l'espace de recherche ainsi que de leurs voisines. C'est pourquoi nous limitons cette analyse à des paysages de taille réduite (instances NK de taille $N = 24$), dont l'espace de recherche est énumérable ($2^2$4 solutions). Cela permet également de considérer des instances NK au fort ratio K/N et donc extrêmement rugueux, possibilité réduite avec de plus fortes valeurs de N puisque la matrice de contribution, composée de N fonctions pseudo-booléennes à $K+1$ variables décrites en extension, est de taille exponentielle en fonction de K. Trois instances sont ici utilisées : $\text{NK}_{24,1}$, $\text{NK}_{24,12}$ et $\text{NK}_{24,22}$.

Afin d'analyser l'épistasie des fonctions objectif et fonctions générées, nous parcourons l'ensemble de l'espace de recherche. Nous déterminons pour chaque solution s_{ab} puis chaque combinaison $(s_{ab}, s_{Ab}, s_{aB}, s_{AB})$ telle que $s_{ab}\Delta s_{Ab} = s_{ab}\Delta s_{Ba} = 1$ et $s_{ab}\Delta s_{AB} = s_{aB}\Delta s_{Ba} = 2$, la configuration en termes d'épistasie et l'associons à la valeur de fitness de s_{ab}. La figure 5.13 figurait déjà dans le premier chapitre (figure 1.5) ; elle permet de rappeler notre numérotation des différentes configurations d'épistasie de signe qui sera utilisée dans cette analyse. Nous visualisons ensuite la répartition de l'épistasie dans le paysage dans un contexte d'ascension, en fonction de la fitness de la solution de référence (par paliers de 0.01). Les fonctions générées l'ont été avec l'algorithme $(1, \lambda)$-ES paramétré comme précédemment (voir section 4.3) ; il s'agit donc de fonctions NK (relâchées sur le critère des liens) de paramètres $N = 24$ et $K = 1$.

Résultats

La figure 5.14 permet de visualiser l'épistasie des différents paysages : ceux des fonctions objectif (figures 5.14a, 5.14c et 5.14e) et des fonctions alternatives générées respectives (figures 5.14b, 5.14d et 5.14f). Tous les graphiques se composent de la même manière. Chaque niveau

de gris correspond à un type d'épistasie, dont les numéro font références aux catégories de la figure 5.13. Les histogrammes ne représentent que les cas où les mutations sont favorables afin d'identifier les difficultés spécifiques aux hill climbers. Les courbes indiquent globalement les proportions d'épistasie de signe réciproque (6+7), non réciproque (4+5) et d'absence d'épistasie de signe (2+3). La courbe 1 reporte la proportion de solutions non améliorantes situées à une distance de 2 ; pour celles-ci, l'épistasie a été caractérisée dans l'autre sens (depuis s_{AB}).

En premier lieu, nous pouvons visualiser la difficulté des instances que l'on étudie. En effet, nous pouvons remarquer que l'épistasie de signe réciproque se concentre proportionnellement sur les très bonnes solutions, y compris sur un paysage plutôt lisse comme le $\text{NK}_{24,1}$ où environ 40% des double-mutations améliorantes (dans l'espace des solutions) présentent de l'épistasie. Comme attendu, plus le paramètre K augmente plus l'épistasie est importante dans tout le paysage, ce qui implique une plus grande difficulté à naviguer dans le paysage et donc à trouver la solution optimale.

Les fonctions alternatives (à droite sur la figure) ont des caractéristiques différentes des fonctions objectif en termes d'épistasie, et très proches les unes des autres. Tout d'abord, le modèle NK n'utilise qu'une valeur de paramètre $K = 1$, ce qui limite naturellement l'épistasie dans les paysages générés. Cependant, on remarque que l'épistasie est plus faible pour les fonction générées que pour la fonction objectif $\text{NK}_{24,1}$. Cette observation s'inscrit en cohérence avec les analyses précédentes, concernant la réduction des optima locaux et donc de lissage du paysage associé.

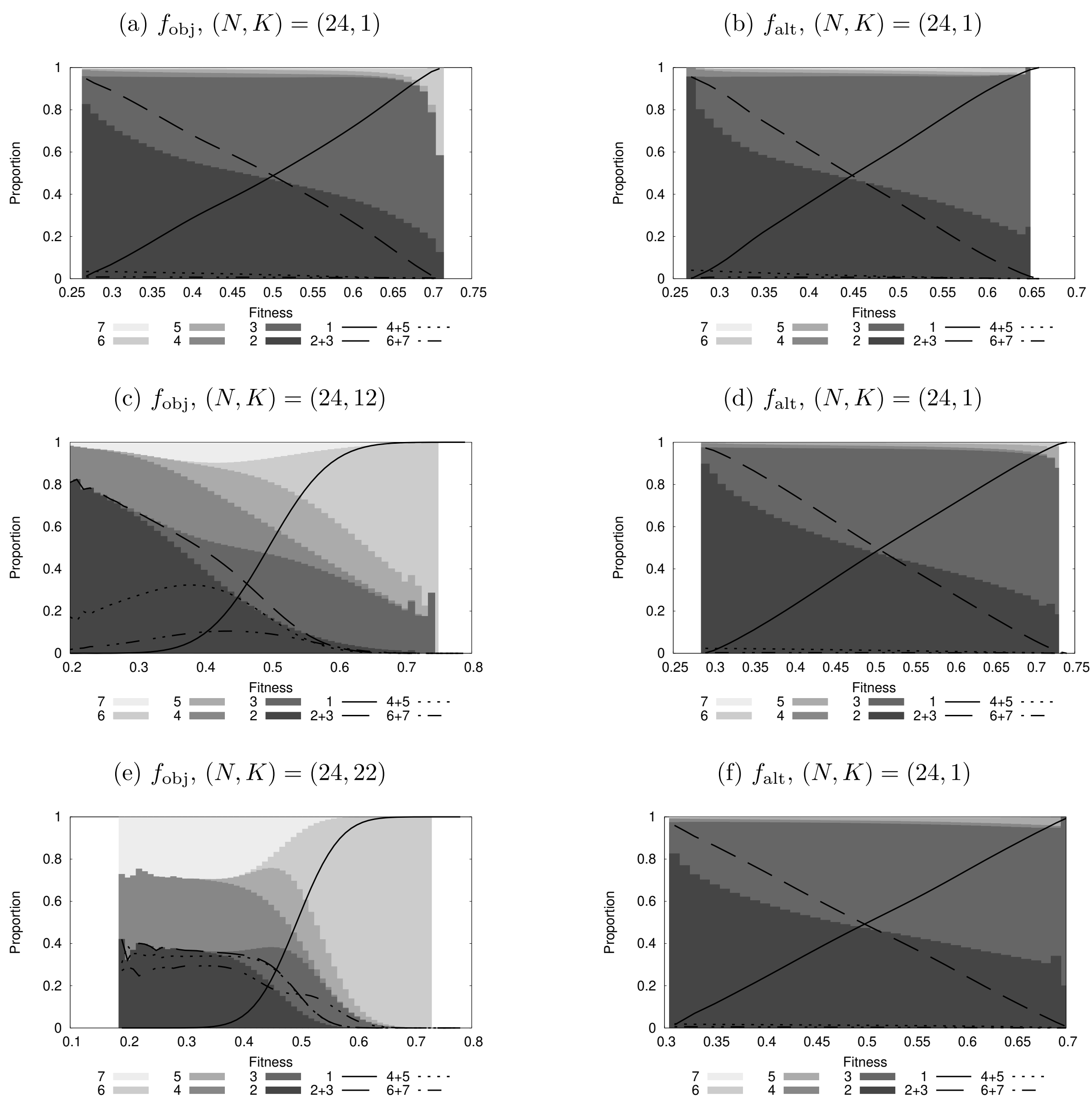
5.2.4 Interactions entre variables

Chacun des modèles que l'on a proposés lie plusieurs variables de la solution entre elles pour en définir une évaluation. Ces liens sont nécessaires pour pouvoir modéliser des fonctions d'évaluation multimodales. Dans le cas contraire, placer le processus d'évolution dans l'espace des fonctions reviendrait essentiellement au même que d'agir dans l'espace des solutions. Nous analysons à présent l'aspect syntaxique des fonctions évoluées, c'est-à-dire la description des fonctions NK issues du processus évolutionnaire. Plus particulièrement, nous comparons les liens entre variables décrits par la fonction alternative avec ceux décrits par la fonction objectif.

Protocole expérimental

Cette étude consiste notamment à visualiser les liens décrits par les fonctions NK. De par son nombre réduit d'interactions entre variables (24 liens) et sa structure simple, utiliser l'instance $\text{NK}_{24,1}$ permettra de rendre lisibles les résultats et faciliter leur interprétation.

FIGURE 5.14 – Répartition de l'épistasie de signe dans des paysages NK aléatoires et issus de fonctions alternatives



Toujours au moyen de la stratégie (1,λ)-ES aux paramètres inchangés, nous générons 100 fonctions d'évaluation alternatives. Nous comparons alors les liens décrits par ces différentes fonctions ainsi que par la fonction objectif $NK_{24,1}$ qui a servi à guider le processus d'évolution. Rappelons que les climbers guidés par les 100 fonctions alternatives parviennent alors à résoudre efficacement l'instance de problème $NK_{24,1}$. Nous empruntons alors à la biologie un modèle de représentation des interactions entre gènes. Les gènes (dans notre cas, les variables) sont représentés en cercle, et sont reliés si une interaction est détectée entre deux gènes (un lien apparaît entre deux variables).

Résultats

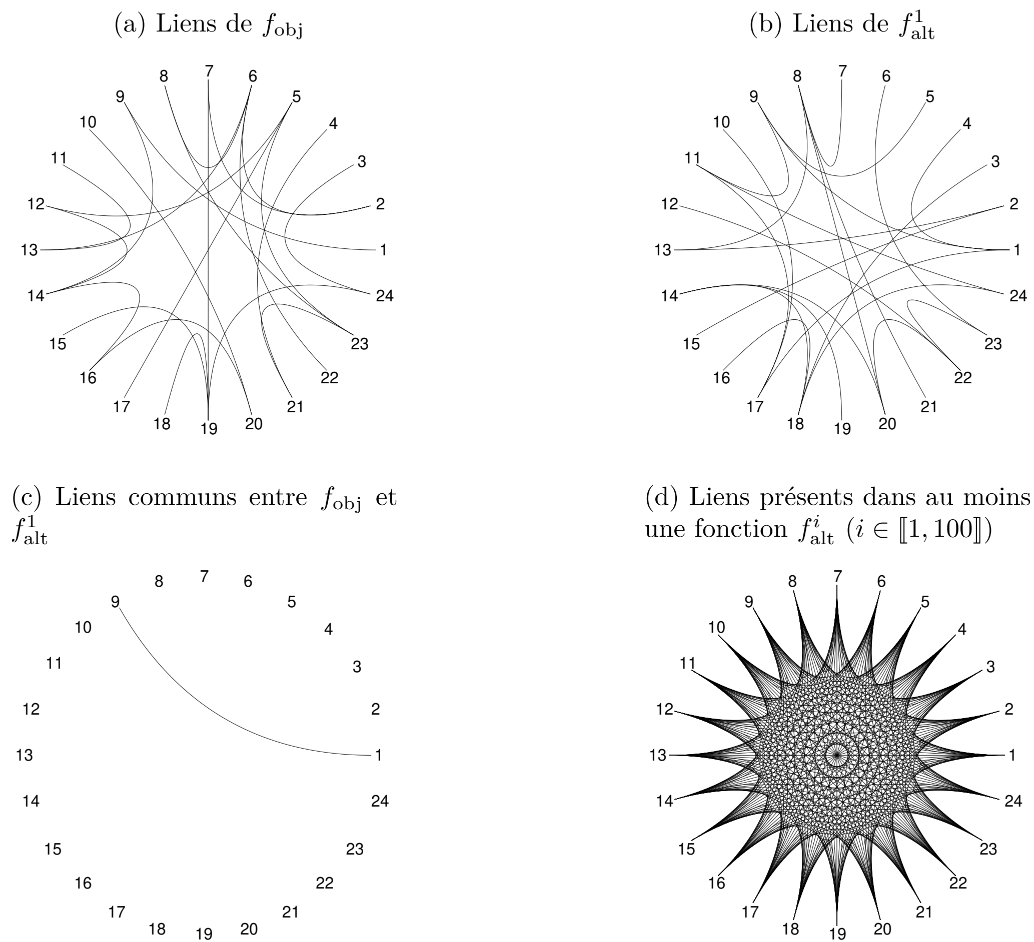
La figure 5.15 représente les liens des fonctions objectif et générées. Les liens de la fonction objectif $NK_{24,1}$ sont présentés figure 5.15a; la figure 5.15b montre ceux d'une fonction d'évaluation générée choisie arbitrairement (première fonction alternative). Pour faciliter leur comparaison la figure 5.15c présente les liens en commun entre ces deux dernières fonctions, et on peut constater qu'un seul un lien est commun entre les deux. Enfin, la figure 5.15d regroupe l'ensemble des liens apparaissant dans au moins l'une des 100 fonctions alternatives. En fait, il existe au moins une fonction générée dans laquelle un couple donné de variables se trouve en interaction ; ce dernier diagramme représente donc un graphe complet. Cela indique que la stratégie d'évolution n'a pas à retrouver la fonction objectif ni ses interactions entre variables, et rappelle qu'une fonction objectif ne décrit pas un optimum global mais une échelle de valeur entre les différentes solutions, et qu'il n'est pas toujours pertinent de chercher à exploiter, en particulier localement, le paysage induit par la description de l'instance.

Pour compléter le propos, nous reportons sur la figure 5.16 la distribution des liens en fonction de leur fréquence d'apparition dans les fonctions générées. Si l'établissement des liens est totalement arbitraire, alors leur nombre d'apparition dans des fonctions d'évaluation doit suivre une distribution binomiale (représentée également sur la figure), ce qui semble être approximativement le cas à partir de l'étude de 100 fonctions. Nous pouvons en conclure qu'il n'y a pas de raison particulière à lier certaines variables entre elles plutôt que d'autres ; il s'agit seulement d'un outil de description façonnant la structure du paysage. De manière évidente, il est possible de construire une fonction d'évaluation efficace à partir de n'importe quel lien entre variables, tant que le modèle permet d'enrichir la définition de cette fonction.

5.2.5 Interactions entre sous-fonctions

Une fonction alternative décrite au moyen des différents modèles utilisés (NK avec $K = 1$, MAX-3SAT, somme pondérée) est composée d'un ensemble de sous-fonctions booléennes ou

FIGURE 5.15 – Comparaison des liens entre variables dans des fonctions $\mathrm{NK}_{24,1}$ objectif f_{obj}, et générées f^{i}_{alt}.



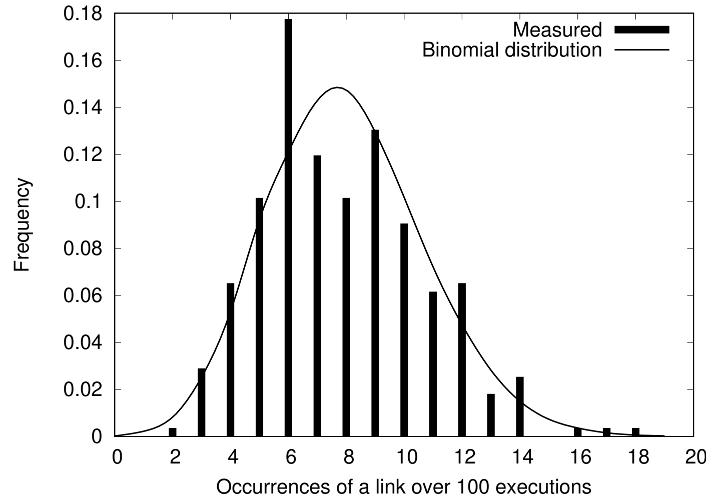
(a) Liens de f_{obj}

(b) Liens de f^{1}_{alt}

(c) Liens communs entre f_{obj} et f^{1}_{alt}

(d) Liens présents dans au moins une fonction f^{i}_{alt} ($i \in [\![1, 100]\!]$)

Figure 5.16 – Distribution observée des liens entre variables sur 100 fonctions $NK_{24,1}$ générée



pseudo-booléennes d'ordre 2 ou 3. Après avoir analysé l'effet des interactions entre variables avec comme cas d'étude le modèle NK, nous évaluons à présent l'impact des sous-fonctions et leurs interactions.

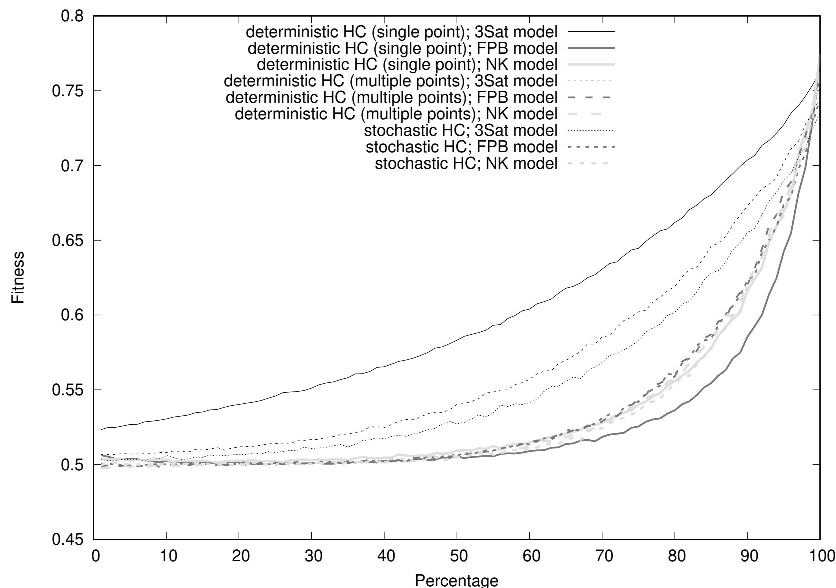
Protocole expérimental

Pour estimer l'impact combiné des sous-fonctions composant une fonction alternative, nous mesurons l'effet de la suppression d'une fraction d'entre elles sur les performances du hill climber associé. En prolongement de l'expérimentation étendue (section 4.6), nous sélectionnons arbitrairement une fonction générée pour chaque combinaison de fonction objectif (parmi les instances $NK_{128,12}$ et MAX-SAT uf250), modèle de fonction (NK, MAX-3SAT, somme de fonctions booléennes) et d'algorithme de hill climbing (déterministe à point de départ unique fixé, déterministe à points de départ multiples fixés, stochastique). Nous comparons alors les résultats des algorithmes obtenus avec différentes fractions de chaque modèle.

Résultats

Les figures 5.17 et 5.18 comparent l'efficacité des différents hill climbers (pour chacune des 9 variantes et respectivement appliqués aux instances $NK_{128,12}$ et MAX-SAT uf250) en fonction de la proportion de sous-fonctions utilisées parmi les sous-fonctions de la fonction alternative générée. L'axe des abscisses représente la proportion conservée de la fonction d'évaluation initiale, et l'axe des ordonnées les scores moyens obtenus par les hill climbers guidés par ces fonctions

Figure 5.17 – Fitness moyenne des solutions retournées à partir de fonctions d'évaluation partielles sur l'instance $NK_{128,12}$

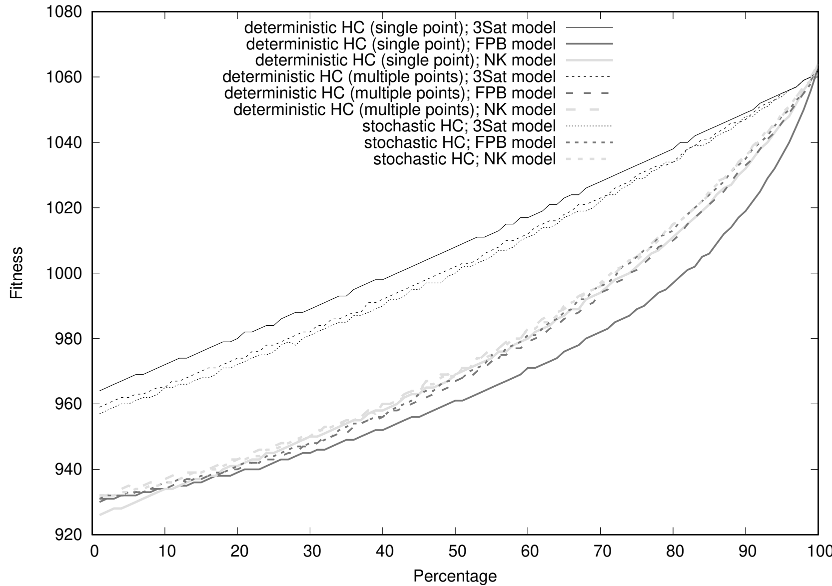


partielles. Tout d'abord, nous pouvons vérifier que les scores sont tous dans la même zone haute lorsque la fonction d'évaluation est utilisée dans son intégralité, signifiant que nous avons pu générer des climbers efficaces, quel que soit le modèle de fonction et d'algorithme employé. Ensuite, nous pouvons remarquer que le modèle MAX-3SAT est beaucoup plus robuste dans cet exercice que les autres. En effet, indépendamment des modèles d'algorithmes, c'est ce modèle de fonction qui obtient les meilleurs résultats avec des modèles partiels. Cela est particulièrement notable sur l'instance u250 où l'augmentation du score est linéaire en fonction du nombre de clauses de la fonction alternative utilisées pour l'évaluation. Sur cette instance et ce modèle précis l'interaction entre sous-fonctions (clauses) n'entre pas en jeu, contrairement aux autres combinaisons d'instance à résoudre et de modèle de fonction utilisé, à des degrés différents.

5.2.6 Visualisation des fonctions d'évaluation

Pour conclure les analyses comportementales et le dernier chapitre de ce manuscrit, nous voulons proposer une méthode de visualisation des fonctions d'évaluation générées ou associées à un problème donné, afin de mieux comprendre non seulement le comportement des fonction individuelles mais aussi leur processus de génération. La visualisation de telles fonctions et paysages combinatoires est une problématique très complexe. Il existe des outils de visualisation déjà développés pour les paysages de fitness [88, 87], basés sur les réseaux d'optima locaux et

FIGURE 5.18 – Fitness moyenne des solutions retournées à partir de fonctions d'évaluation partielles sur l'instance MAX-SAT uf250



l'étude des perturbations de recherches locales itérées. Ici, nous ne cherchons pas à identifier sur un graphe les difficultés globales de l'instance, mais à visualiser des trajectoires empruntées par des algorithmes différents. Nous cherchons donc à projeter toutes les solutions d'un ensemble de trajectoire sur un plan — et pas uniquement des solutions particulières — de manière à préserver au maximum les distances relatives entre les solutions.

Nous organisons alors cette section en deux parties. La première partie présente l'algorithme de placement des solutions à représenter sur le plan, et la seconde différents exemples de visualisation de fonctions et trajectoires.

Algorithme de placement de sommets de l'hypercube sur le plan

Considérons le problème d'optimisation $(\mathcal{X}, f_{\text{obj}})$, un voisinage $\mathcal{N}$, et n trajectoires de recherche dans le paysage $(\mathcal{X}, \mathcal{N}, f_{\text{obj}})$ à représenter. Chaque trajectoire est composée de plusieurs solutions $s \in \mathcal{X}$. Soit une trajectoire $s_1, s_2, \ldots, s_m$, avec $\forall i \in [\![2; m]\!], s_{i-1} \in \mathcal{N}(s_i)$. La problématique consiste à représenter le plus fidèlement possible les n trajectoires sur le plan. Pour ce faire, les solutions apparaissant dans au moins une trajectoire doivent être placées de manière à ce que les distances euclidiennes entre les solutions rendent compte de leurs distances respectives dans le paysage (dans notre cas d'étude, les distances de Hamming). Nous considérons maintenant deux matrices : une matrice des distances de Hamming entre couples de solutions, et une matrice

des distances euclidiennes entre couples de points, où chaque point correspond à une solution. Notre problème est alors de trouver un placement des points du plan pour que la matrice des distances correspondante se rapproche au maximum de la matrice des distances de Hamming.

La résolution approchée de ce problème est effectuée au moyen d'un algorithme glouton plaçant les solutions sur le plan une par une selon l'ordre initial des trajectoires, en ignorant les solutions déjà rencontrées. Le placement d'un point s'effectue au moyen d'une fonction de placement qui initialise le point à placer sur le point précédent de la trajectoire, puis est déplacé séquentiellement à la manière d'une recherche locale naviguant à chaque mouvement parmi quatre directions. Le placement final du point devra néanmoins se trouver dans une limite autour du placement du point précédent afin que la visualisation ne comporte pas de grands sauts dans les trajectoires. La fonction d'évaluation d'un point est donnée par $f(p_k) = \sum_{i=1}^{k-1} |d(p_k, p_i) - s_k \Delta s_i|$, avec s_k la k-ième solution à représenter, p_k le point de coordonnées à évaluer et $d(\cdot, \cdot)$ la distance euclidienne.

Une fois que tous les points sont placés, nous dessinons en trois dimensions les trajectoires que nous voulons visualiser. Les deux premières dimensions correspondent aux placements calculés par l'algorithme que nous venons de décrire, et la troisième correspond à l'évaluation de la solution associée au point par f_{obj}. Une fois que toutes les coordonnées (bidimensionnelles) des points ont été déterminées, les trajectoires peuvent être représentées en trois dimensions à partir de la position des solutions projetées sur un plan et leur hauteur dans le paysage donnée par la fonction objectif f_{obj}.

Exemples de visualisation de différentes fonctions d'évaluation

La figure 5.19 présente un exemple de visualisation pour le problème $\text{NK}_{128,1}$ établie à partir de trajectoires de hill climbers stochastiques guidés soit par la fonction objectif soit par une fonction alternative. Nous pouvons voir que les trajectoires issues de la fonction générée arrivent à de meilleures solutions que celles issues de la fonction objectif, mais aussi que les optima locaux de la fonction objectif sont proches les uns des autres. Cette caractéristique indique que le paysage associé au problème est relativement lisse et donc plutôt simple à optimiser. Nous présentons dans l'annexe plusieurs stades d'évolution de la fonction d'évaluation de la figure A17 à la figure A20.

La figure 5.20 représente les trajectoires d'un hill climber déterministe à points de départ multiples fixés généré pour l'instance $\text{NK}_{128,12}$. Contrairement au cas précédent, nous remarquons que les trajectoires sont globalement améliorantes au sens de f_{obj} uniquement sur les dernières itérations. C'est un comportement qui pouvait être attendu pour ce type de problèmes dont la corrélation entre la fitness des solution et leur distance à l'optimum est faible.

FIGURE 5.19 – Visualisation de hill climbers guidés par la fonction objectif (grises) et par une fonction générée (noire), f_{obj}=$\text{NK}_{128,1}$

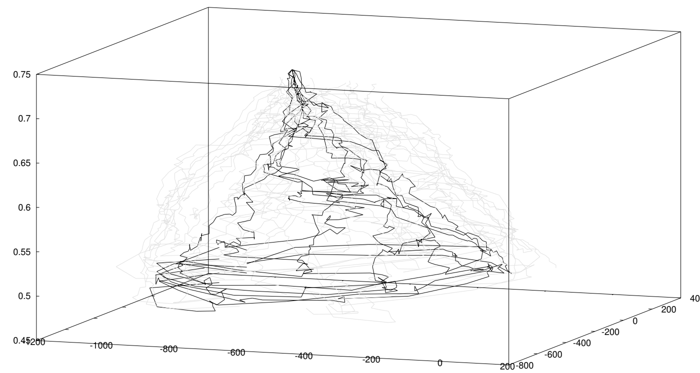


FIGURE 5.20 – Visualisation de hill climbers guidés par une fonction générée, f_{obj}=$\text{NK}_{128,12}$

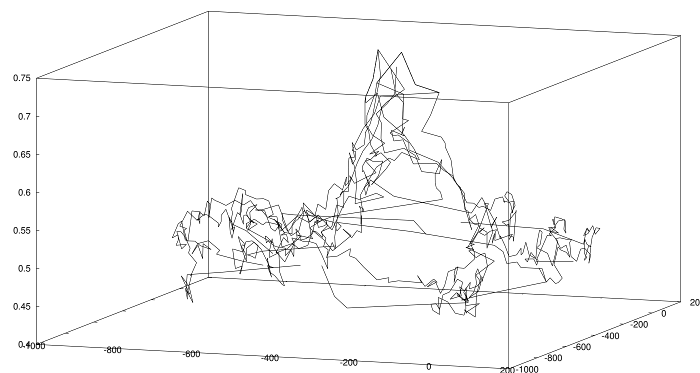
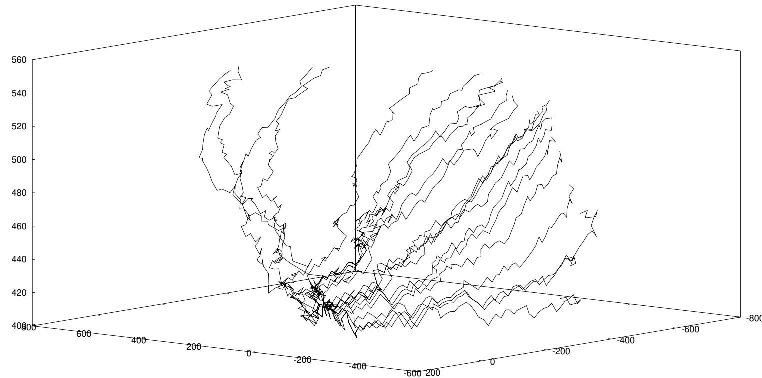


FIGURE 5.21 – Visualisation de hill climbers guidés par les fonctions générées, f_{obj}=flat-50.cnf



Enfin, la figure 5.21 regroupe l'ensemble des fonctions améliorantes générées au cours d'un processus d'évolution pour un hill climber déterministe à point de départ unique, sur l'instance MAX-SAT ais8. Cela permet d'illustrer l'effet de l'évolution de la fonction d'évaluation sur la trajectoire générée par le climber, qui par définition début toujours du même point et parcourt les variables selon un même ordre préétabli. Les zones explorées par les différentes fonctions sont différentes et mènent à de meilleures solution au cours de la recherche.

Conclusion

Dans cette thèse, nous avons proposé une approche de génération automatique d'algorithmes pour la résolution de problèmes d'optimisation. L'objectif était de mettre en œuvre une stratégie d'évolution des fonctions d'évaluation utilisées par des algorithmes de recherche locale afin de guider leur exploration de l'espace des solutions. Une telle approche s'appuie sur des considérations liées à une meilleure adéquation entre le paysage de fitness des problèmes, induit par le mécanisme d'évaluation des solutions, et les heuristiques de mouvement classiquement utilisées par les stratégies de recherche locale. Pour valider cette idée, nous nous sommes concentrés sur des algorithmes très simples afin de mettre en valeur tous les bénéfices qui pourraient être tirés de l'utilisation de fonctions d'évaluation différentes des fonctions objectifs initiales des problèmes. Nous avons ainsi défini un protocole expérimental progressif et approfondi qui nous a permis d'améliorer incrémentalement les résultats jusqu'à l'obtention d'algorithmes compétitifs sur des jeux de tests classiques. Mais au-delà de l'efficacité opérationnelle, notre travail a surtout pour but de mieux comprendre les mécanismes profonds de la résolution de problèmes combinatoires au moyen de processus évolutionnaires.

Dans le chapitre 1 nous avons introduit l'ensemble des concepts nécessaires incluant la définition des problèmes d'optimisation auquel nous nous intéressons ainsi que les paysages de fitness. Nous avons aussi présenté un certain nombre d'algorithmes dont nous avions besoin ou dont l'approche est connexe à la notre.

Le chapitre 2 présente les motivations de cette thèse ainsi nos objectifs.

Le chapitre 3 décrit les composants de nos algorithmes de génération ainsi que le processus d'amélioration progressive que nous avons suivi. Le chapitre 4 est consacré aux différents résultats obtenus pendant la thèse avec différentes combinaisons de composants et d'algorithmes. Les résultats obtenus sont encourageant dans le sens où l'on a pu démontrer une certaine robustesse de l'algorithme au niveau des modèles de fonction, d'algorithmes et des problèmes étudiés. Enfin, dans le chapitre 5 nous proposons différentes analyses comportementales. Grâce à celle-ci nous avons pu montrer des résultats intéressants sur la génération d'algorithmes. Sur les fonctions générées, nous avons pu montrer la différence d'épistasie exprimée entre les fonctions générées et les fonctions objectifs. Nous avons proposé une nouvelle méthode de visualisation de ces fonctions.

En termes de perspectives de recherche nous pouvons donner plusieurs pistes. D'abord, et parce que notre algorithme de génération est avant tout un algorithme d'optimisation, nous pouvons continuer à améliorer cet algorithme afin de générer des méthodes de résolution de plus en plus efficaces en un temps plus court. De même, pour les composants de cette méthode de génération, de meilleures versions peuvent être envisagées. Par exemple, nous pourrions décider de générer des algorithmes plus complexes que des hill climbers. Un composant de la génération qui mérite plus d'attention est le modèle des fonctions d'évaluation. En effet, ces modèles pourraient faire l'objet d'études théoriques et pratiques afin de définir un modèle plus évolué qu'une simple instance de problème, et surtout développer un modèle de fonction avec une fonction de mutation plus spécifique que celle que nous avons pu présenter dans ce manuscrit. Enfin, il peut être intéressant d'étendre la génération d'algorithmes à d'autres classes de problèmes, comme les problèmes de permutation, et d'établir à cet effet des modèles de fonction appropriés.

Annexe

Figure A1 – Comparaison complète de 12 fonctions de sélection paramétrables avec celle utilisée en section 4.3 sur l'instance $NK_{128,1}$

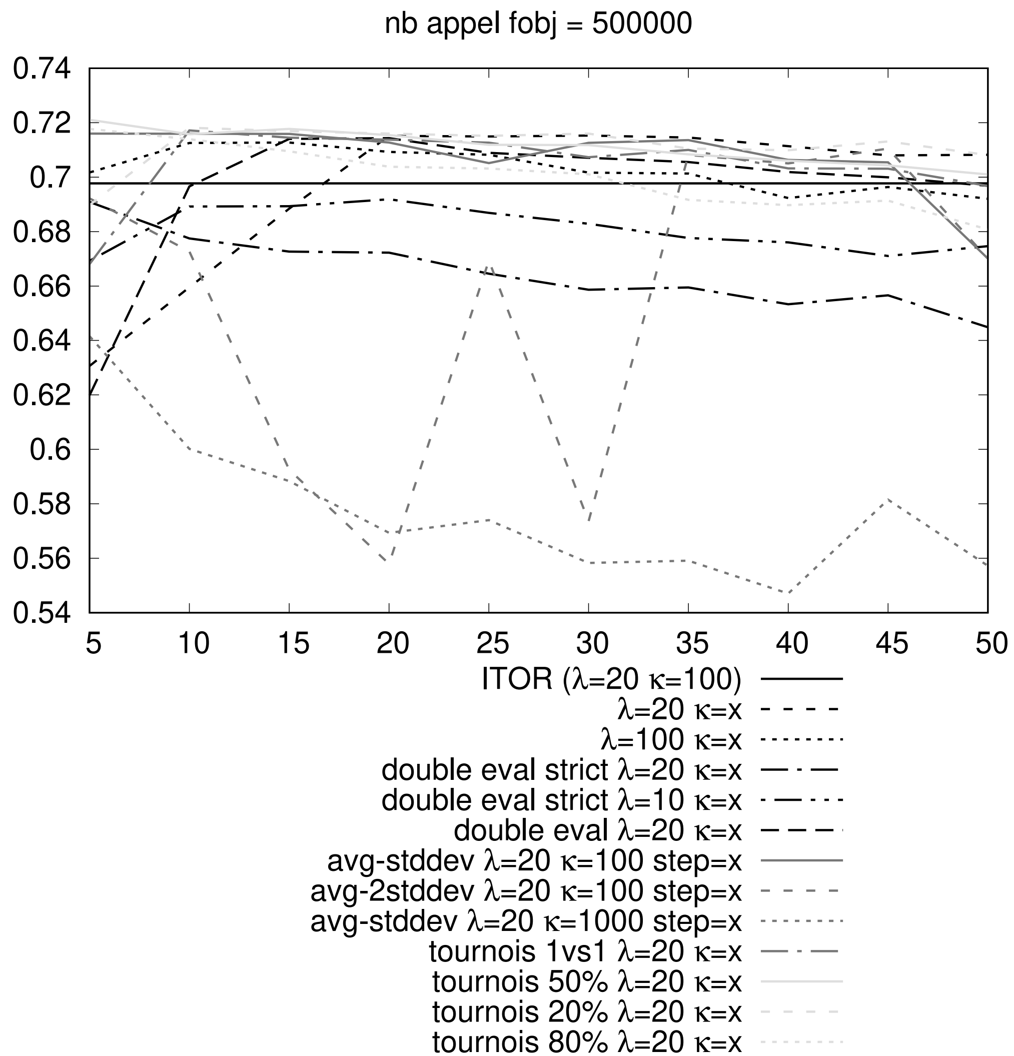


Figure A2 – Répartition des rangs des algorithmes sélectionnés sur l'instance $NK_{128,1}$ et avec le paramétrage de nombre d'appels à f_{obj} par itération = 500

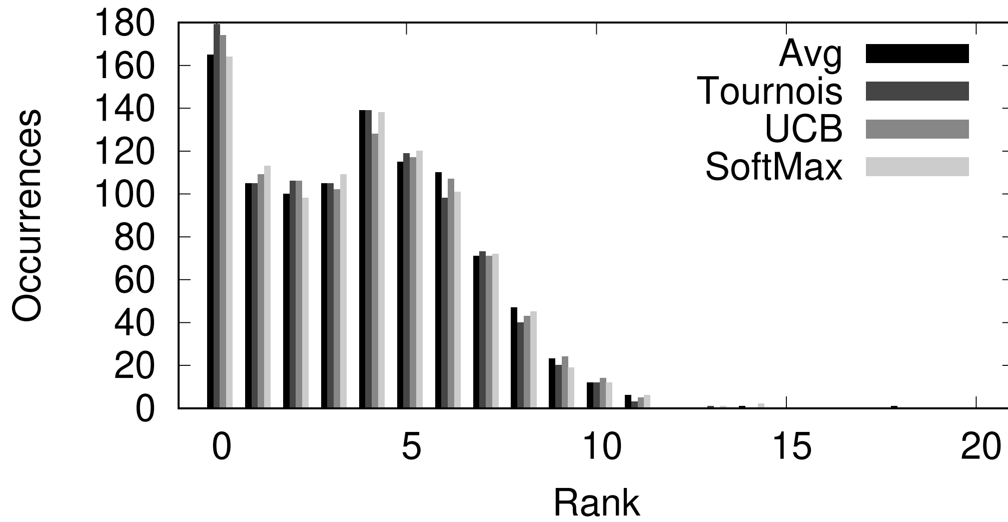


Figure A3 – Répartition des rangs des algorithmes sélectionnés sur l'instance $NK_{128,1}$ et avec le paramétrage de nombre d'appels à f_{obj} par itération = 1000

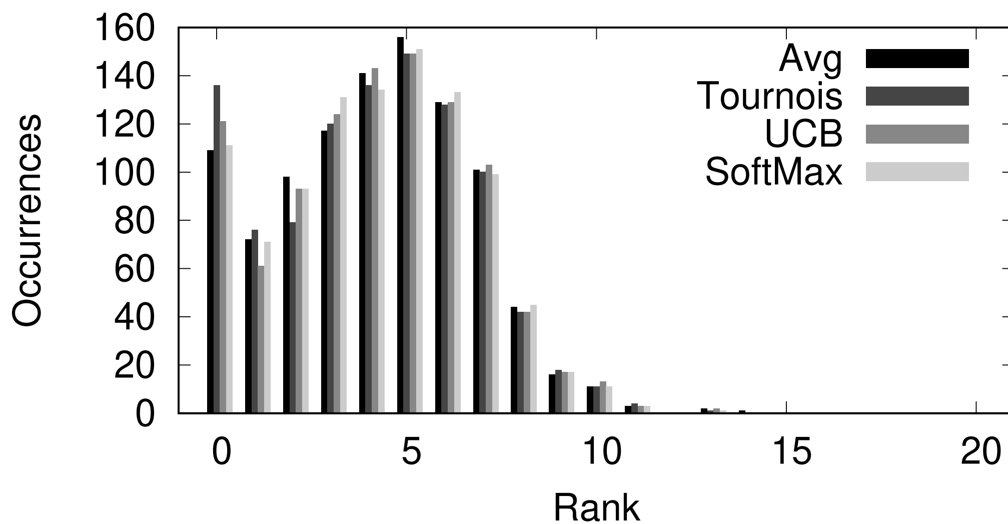


Figure A4 – Répartition des rangs des algorithmes sélectionnés sur l'instance $NK_{128,1}$ et avec le paramétrage de nombre d'appels à f_{obj} par itération = 1500

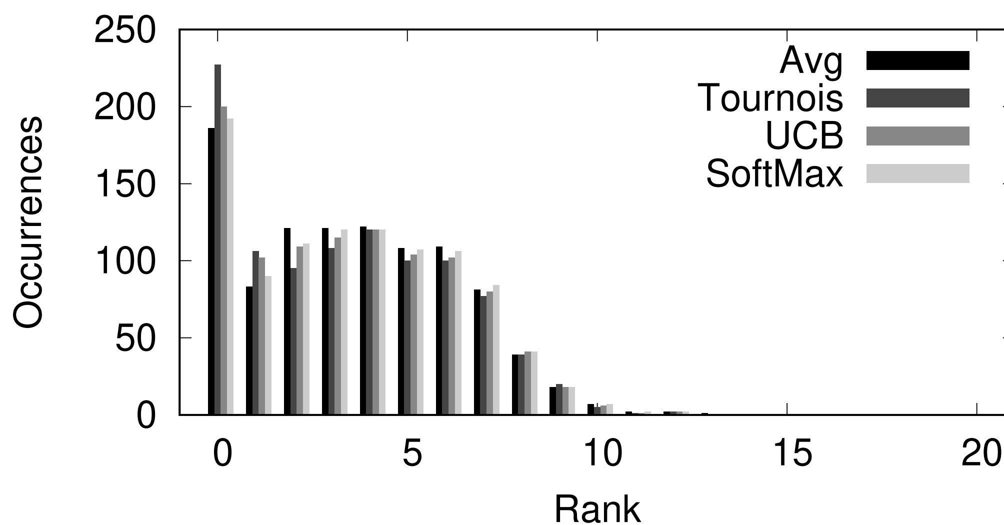


Figure A5 – Répartition des rangs des algorithmes sélectionnés sur l'instance $NK_{128,12}$ et avec le paramétrage de nombre d'appels à f_{obj} par itération = 500

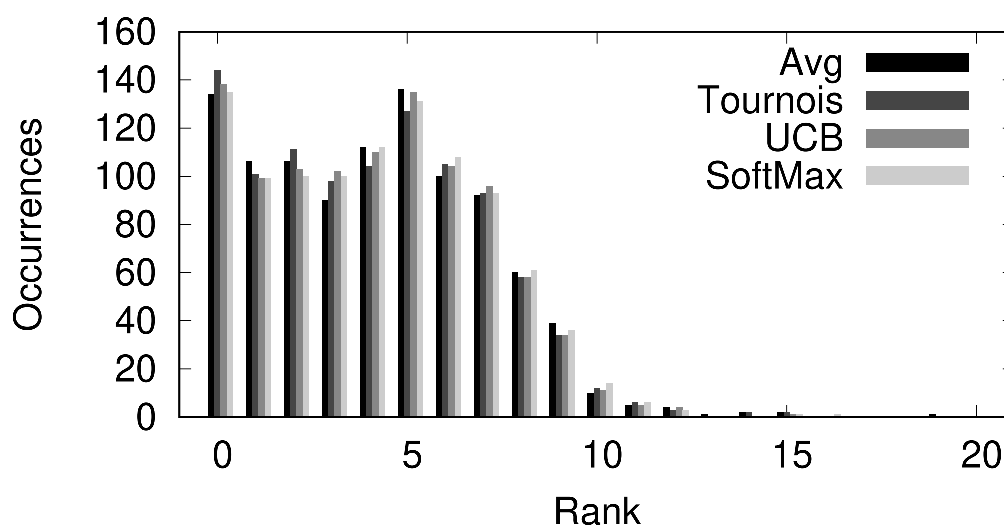


FIGURE A6 – Répartition des rangs des algorithmes sélectionnés sur l'instance $NK_{128,12}$ et avec le paramétrage de nombre d'appels à f_{obj} par itération = 1000

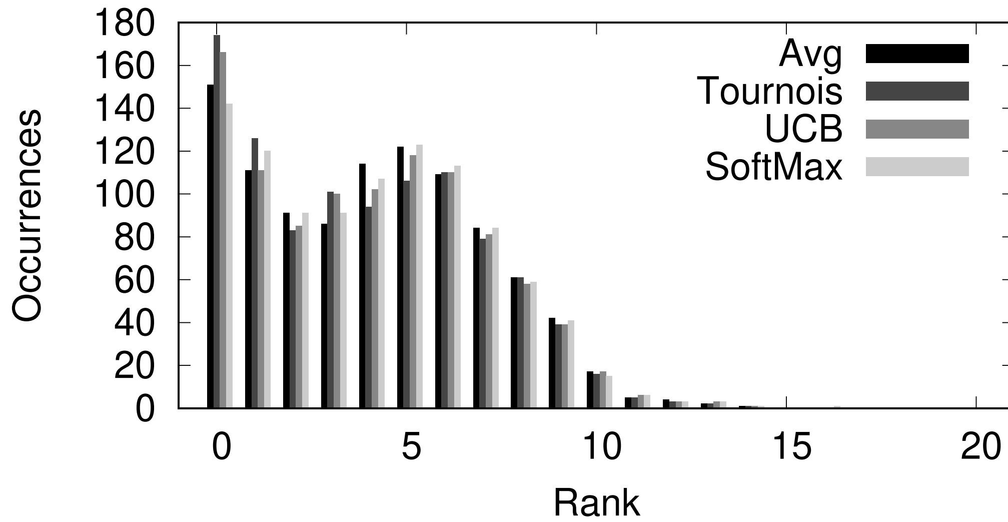


FIGURE A7 – Répartition des rangs des algorithmes sélectionnés sur l'instance $NK_{128,12}$ et avec le paramétrage de nombre d'appels à f_{obj} par itération = 1500

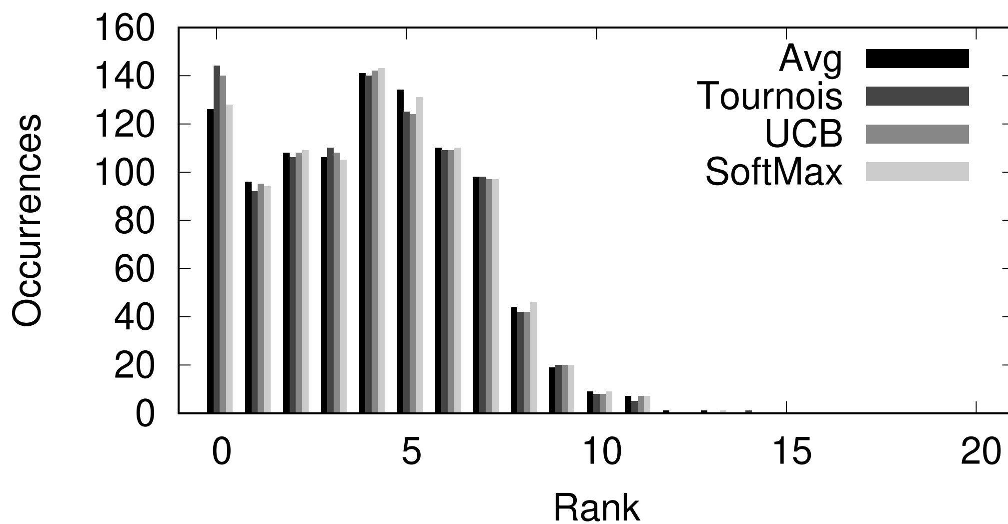


FIGURE A8 – Répartition des rangs des algorithmes sélectionnés sur l'instance $NK_{512,1}$ et avec le paramétrage de nombre d'appels à f_{obj} par itération = 500

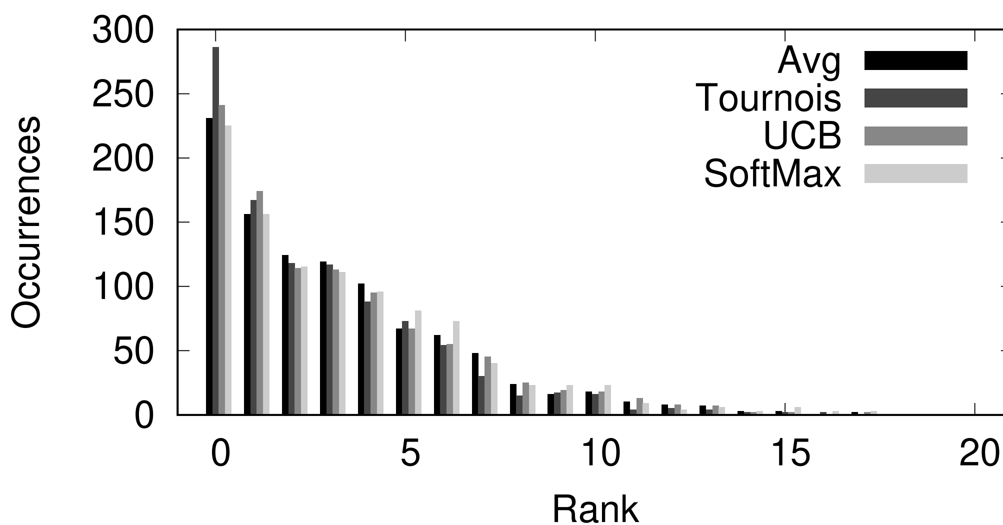


FIGURE A9 – Répartition des rangs des algorithmes sélectionnés sur l'instance $NK_{512,1}$ et avec le paramétrage de nombre d'appels à f_{obj} par itération = 1000

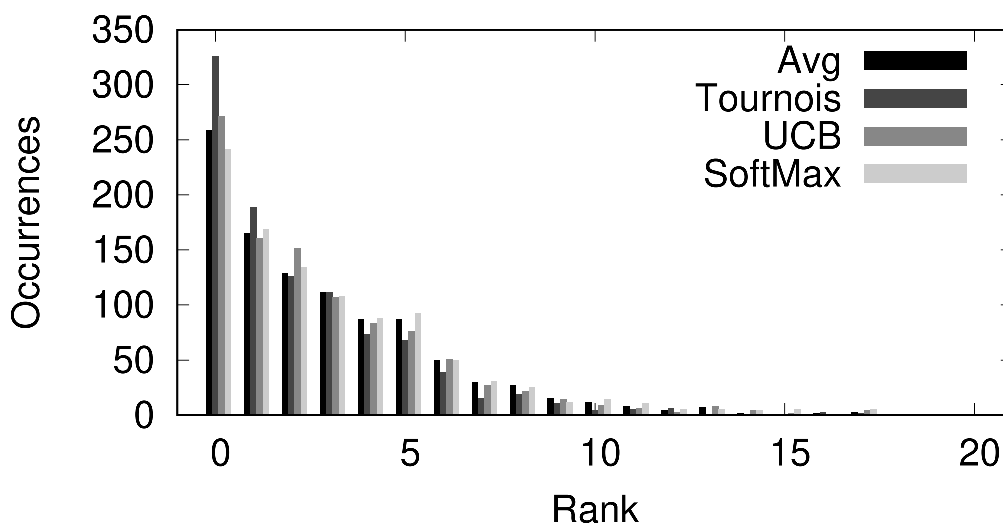


FIGURE A10 – Répartition des rangs des algorithmes sélectionnés sur l'instance $NK_{512,1}$ et avec le paramétrage de nombre d'appels à f_{obj} par itération = 1500

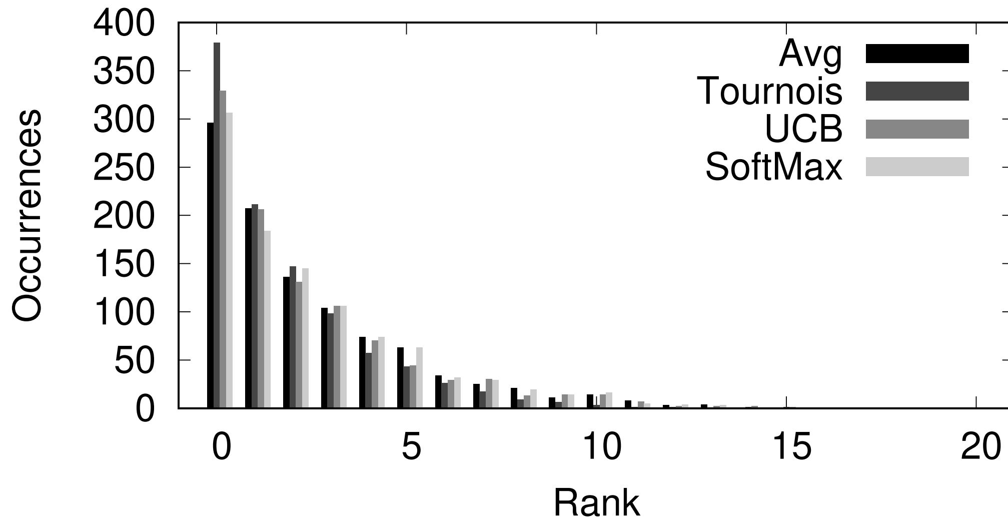


FIGURE A11 – Répartition des rangs des algorithmes sélectionnés sur l'instance $NK_{512,12}$ et avec le paramétrage de nombre d'appels à f_{obj} par itération = 500

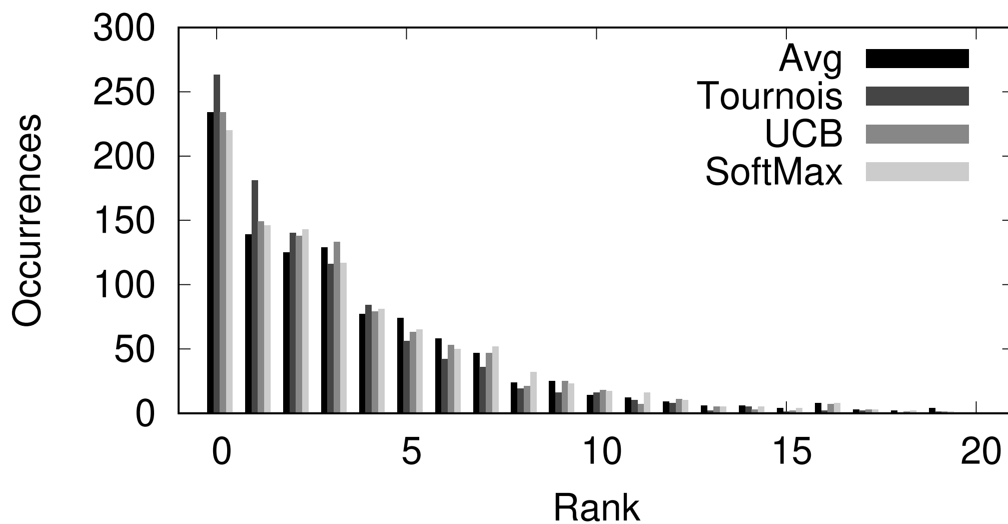


FIGURE A12 – Répartition des rangs des algorithmes sélectionnés sur l'instance $NK_{512,12}$ et avec le paramétrage de nombre d'appels à f_{obj} par itération = 1000

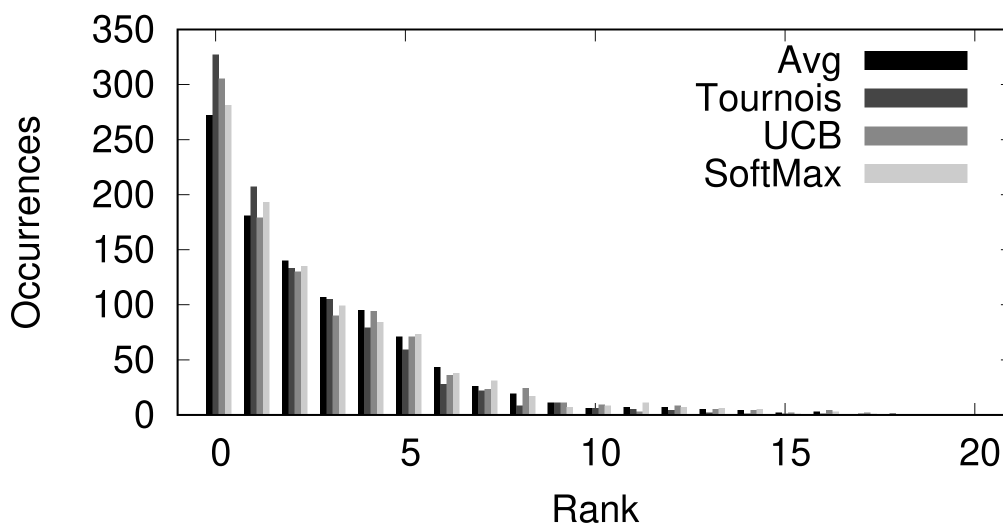


FIGURE A13 – Répartition des rangs des algorithmes sélectionnés sur l'instance $NK_{512,12}$ et avec le paramétrage de nombre d'appels à f_{obj} par itération = 1500

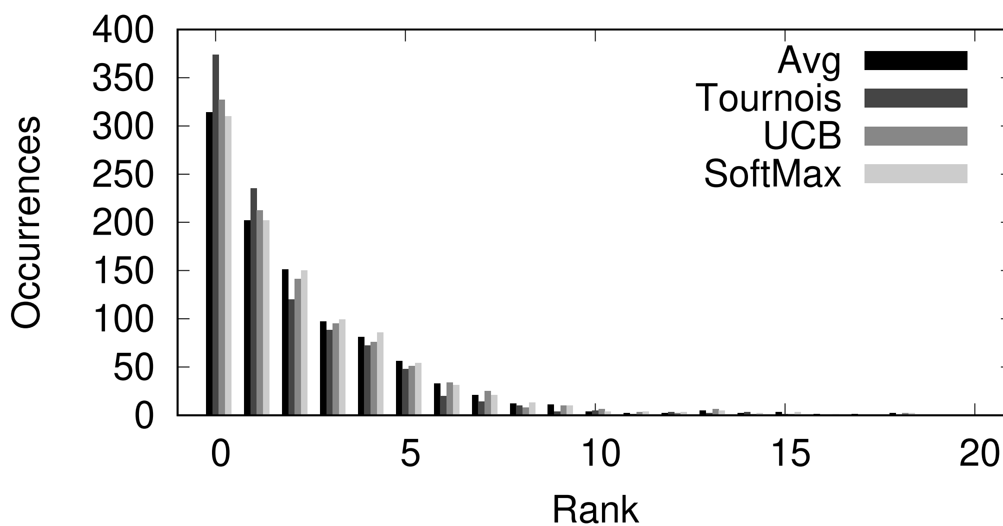


Figure A14 – Proportion des optima changés, pour différents stades d'évolution des fonctions d'évaluation (de f_1^i à f_6^i)

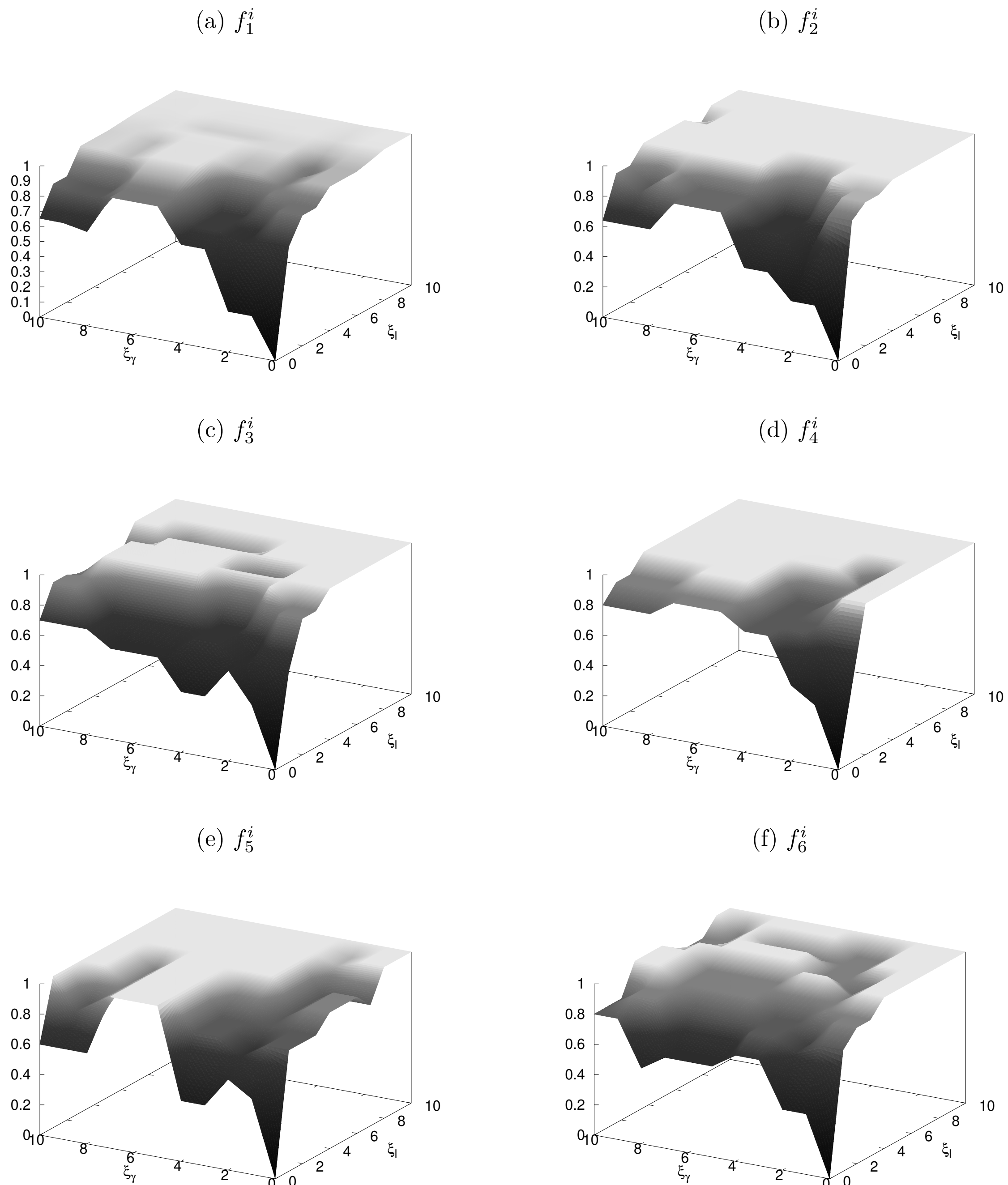
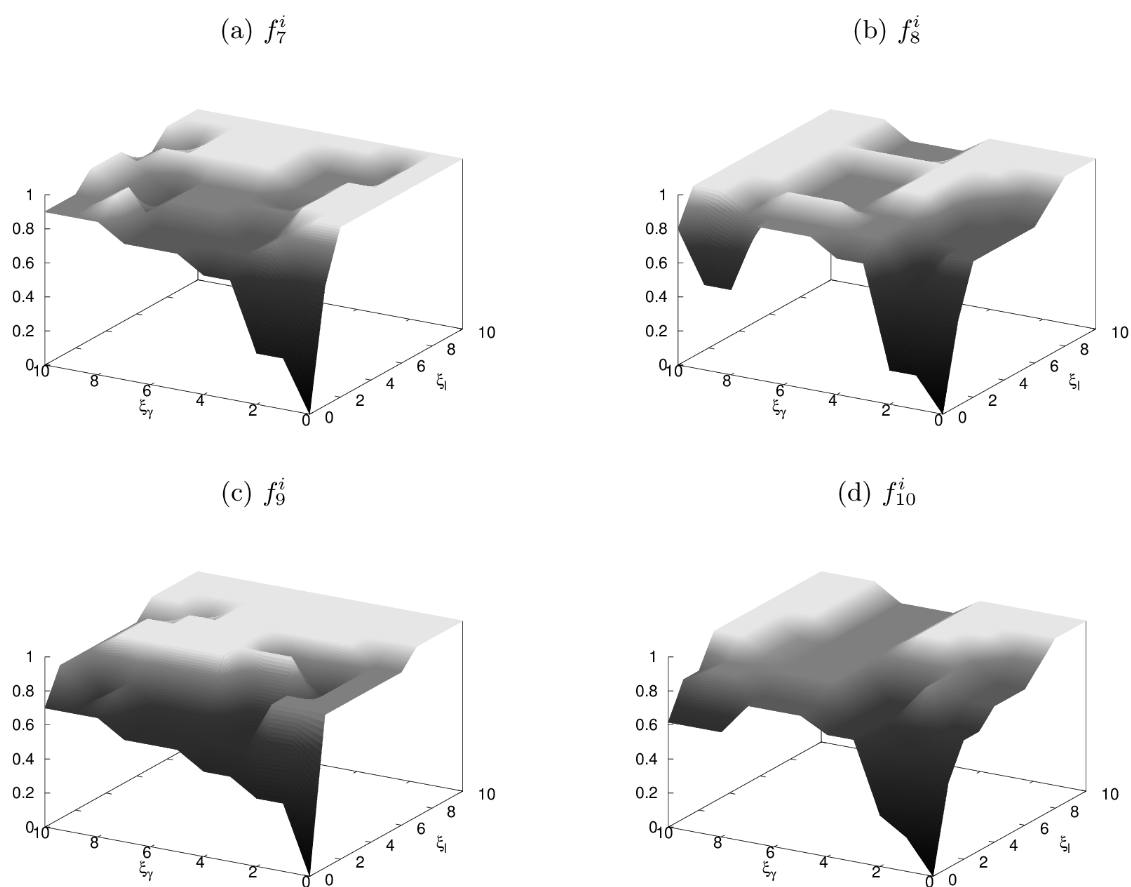


Figure A15 – Proportion des optima changés, pour différents stades d'évolution des fonctions d'évaluation (de f_7^i à f_{10}^i)



(a) f_7^i (b) f_8^i (c) f_9^i (d) f_{10}^i

FIGURE A16 – Évolution des optima locaux de la fontion générée sur l'instance $NK_{128,1}$ en fonction du nombre d'itération de la génération, de la fitness des optima et de leur proportion d'apparition sur un nombre réduit de génération

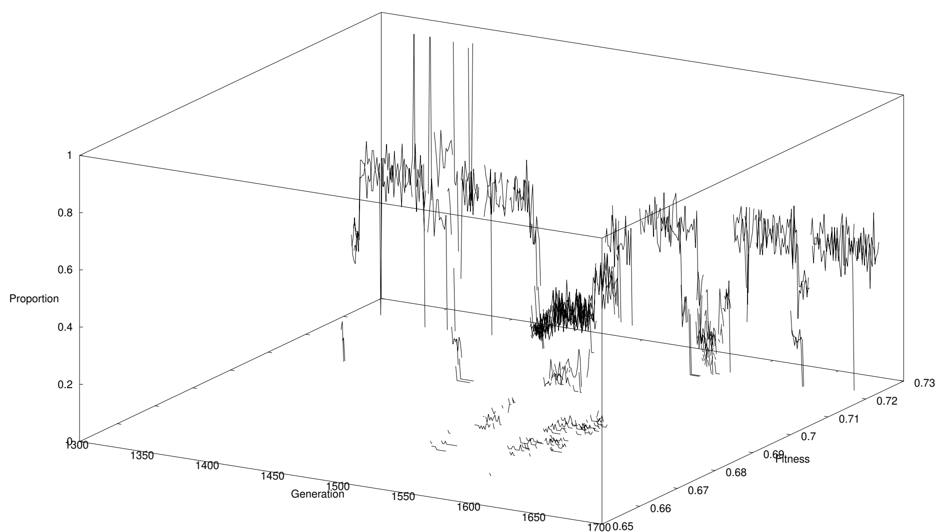


FIGURE A17 – Visualisation de hill climbers guidés par la fonction objectif (grise) et par une fonction aléatoire (noire), $f_{\text{obj}} = NK_{128,1}$

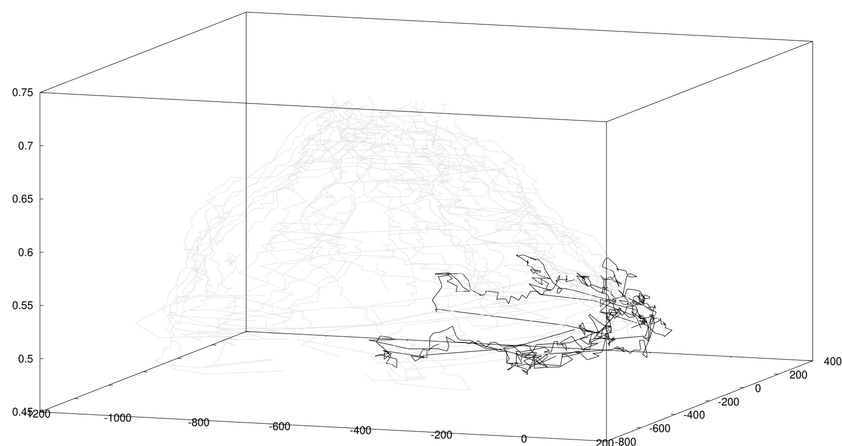


Figure A18 – Visualisation de hill climbers guidés par la fonction objectif (grise) et par une fonction générée peu avancée (noire), $f_{\text{obj}} = \text{NK}_{128,1}$

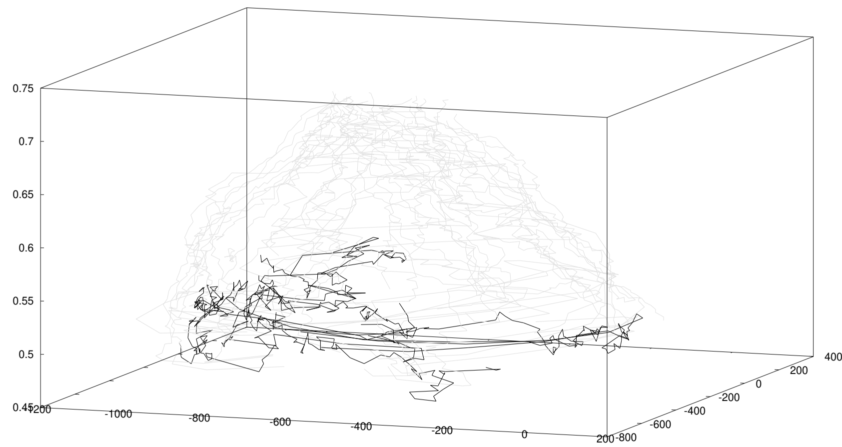


Figure A19 – Visualisation de hill climbers guidés par la fonction objectif (grise) et par une fonction générée intermédiaire (noire), $f_{\text{obj}} = \text{NK}_{128,1}$

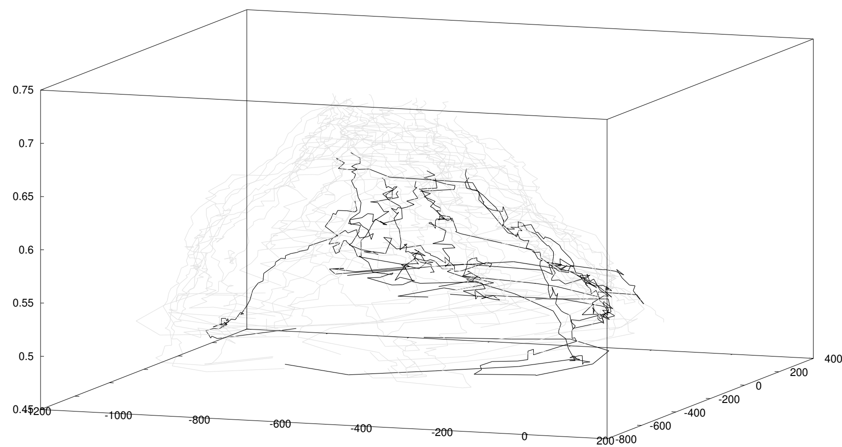
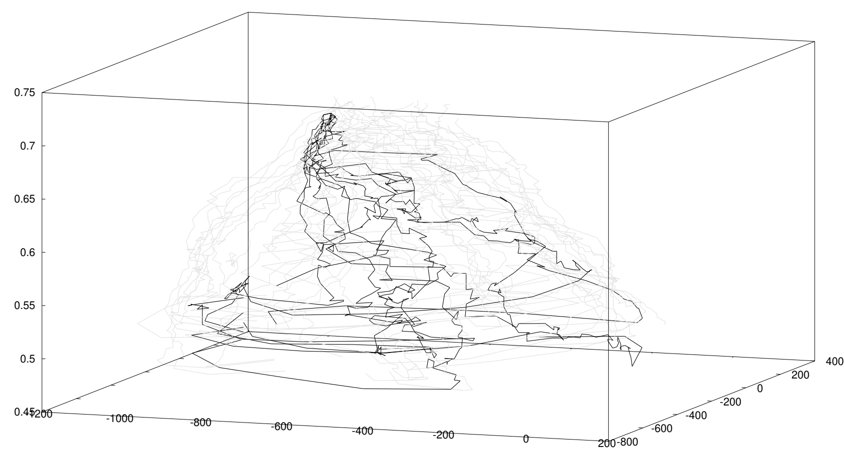


Figure A20 – Visualisation de hill climbers guidés par la fonction objectif (grise) et par une fonction générée avancée (noire), $f_{\text{obj}} = \text{NK}_{128,1}$



BIBLIOGRAPHIE

[1] S. Wright, "The roles of mutation, inbreeding, crossbreeding, and selection in evolution," 1932.

[2] R. B. Heckendorn, S. Rana, and D. Whitley, "Test function generators as embedded landscapes," *Foundations of Genetic Algorithms*, vol. 5, pp. 183–198, 1999.

[3] D. Whitley, "Mk landscapes, nk landscapes, max-ksat : A proof that the only challenging problems are deceptive," in *Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation*, 2015, pp. 927–934.

[4] G. Ochoa, F. Chicano, and M. Tomassini, "Global landscape structure and the random max-sat phase transition," in *International Conference on Parallel Problem Solving from Nature*. Springer, 2020, pp. 125–138.

[5] S. A. Kauffman and E. D. Weinberger, "The nk model of rugged fitness landscapes and its application to maturation of the immune response," *Journal of theoretical biology*, vol. 141, no. 2, pp. 211–245, 1989.

[6] M. R. Garey and D. S. Johnson, "Computers and intractability."

[7] B. Selman, H. A. Kautz, B. Cohen *et al.*, "Local search strategies for satisfiability testing." *Cliques, coloring, and satisfiability*, vol. 26, pp. 521–532, 1993.

[8] S. A. Cook, "The complexity of theorem-proving procedures," in *Proceedings of the third annual ACM symposium on Theory of computing*, 1971, pp. 151–158.

[9] G. H. I. de Azevedo, A. A. Pessoa, and A. Subramanian, "A satisfiability and workload-based exact method for the resource constrained project scheduling problem with generalized precedence constraints," *European Journal of Operational Research*, vol. 289, no. 3, pp. 809–824, 2021.

[10] C. Audet and W. Hare, "Derivative-free and blackbox optimization," 2017.

[11] E. L. Lawler and D. E. Wood, "Branch-and-bound methods : A survey," *Oper. Res.*, vol. 14, pp. 699–719, 1966.

[12] R. E. Gomory, "An algorithm for integer solutions to linear programs," *Recent advances in mathematical programming*, vol. 64, no. 260-302, p. 14, 1963.

[13] R. Gomory, "An algorithm for the mixed integer problem," RAND CORP SANTA MONICA CA, Tech. Rep., 1960.

[14] J. E. Mitchell, "Branch-and-cut algorithms for combinatorial optimization problems," *Handbook of applied optimization*, vol. 1, pp. 65–77, 2002.

[15] M. Tawarmalani and N. V. Sahinidis, "A polyhedral branch-and-cut approach to global optimization," *Mathematical programming*, vol. 103, no. 2, pp. 225–249, 2005.

[16] G. J. Woeginger, "Exact algorithms for np-hard problems : A survey," in *Combinatorial optimization—eureka, you shrink !* Springer, 2003, pp. 185–207.

[17] D. H. Wolpert and W. G. Macready, "No free lunch theorems for optimization," *IEEE transactions on evolutionary computation*, vol. 1, no. 1, pp. 67–82, 1997.

[18] K. Sorensen, M. Sevaux, and F. Glover, "A history of metaheuristics," *arXiv preprint arXiv :1704.00853*, 2017.

[19] M. Abdel-Basset, L. Abdel-Fatah, and A. K. Sangaiah, "Metaheuristic algorithms : A comprehensive review," *Computational intelligence for multimedia big data on the cloud with engineering applications*, pp. 185–231, 2018.

[20] K. Hussain, M. N. M. Salleh, S. Cheng, and Y. Shi, "Metaheuristic research : a comprehensive survey," *Artificial Intelligence Review*, vol. 52, no. 4, pp. 2191–2233, 2019.

[21] M. A. Lones, "Mitigating metaphors : A comprehensible guide to recent nature-inspired algorithms," *SN Computer Science*, vol. 1, no. 1, pp. 1–12, 2020.

[22] K. Sörensen, "Metaheuristics—the metaphor exposed," *International Transactions in Operational Research*, vol. 22, no. 1, pp. 3–18, 2015.

[23] J. Del Ser, E. Osaba, D. Molina, X.-S. Yang, S. Salcedo-Sanz, D. Camacho, S. Das, P. N. Suganthan, C. A. C. Coello, and F. Herrera, "Bio-inspired computation : Where we stand and what's next," *Swarm and Evolutionary Computation*, vol. 48, pp. 220–250, 2019.

[24] C. L. Camacho-Villalón, M. Dorigo, and T. Stützle, "The intelligent water drops algorithm : why it cannot be considered a novel algorithm," *Swarm Intelligence*, vol. 13, no. 3, pp. 173–192, 2019.

[25] C. L. C. Villalón, T. Stützle, and M. Dorigo, “Grey wolf, firefly and bat algorithms : Three widespread algorithms that do not contain any novelty,” in *International Conference on Swarm Intelligence*. Springer, 2020, pp. 121–133.

[26] C. C. Villalón, T. Stützle, and M. Dorigo, “Cuckoo search≡(μ+ λ)–evolution strategy,” 2021.

[27] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, “Optimization by simulated annealing,” *science*, vol. 220, no. 4598, pp. 671–680, 1983.

[28] V. Čern`y, “Thermodynamical approach to the traveling salesman problem : An efficient simulation algorithm,” *Journal of optimization theory and applications*, vol. 45, no. 1, pp. 41–51, 1985.

[29] V. Granville, M. Krivánek, and J.-P. Rasson, “Simulated annealing : A proof of convergence,” *IEEE transactions on pattern analysis and machine intelligence*, vol. 16, no. 6, pp. 652–656, 1994.

[30] F. Glover, “Future paths for integer programming and links to artificial intelligence,” *Computers & operations research*, vol. 13, no. 5, pp. 533–549, 1986.

[31] M. Dorigo, M. Birattari, and T. Stutzle, “Ant colony optimization,” *IEEE computational intelligence magazine*, vol. 1, no. 4, pp. 28–39, 2006.

[32] M. Yannakakis, “The analysis of local search problems and their heuristics,” in *STACS 90*, C. Choffrut and T. Lengauer, Eds. Berlin, Heidelberg : Springer Berlin Heidelberg, 1990, pp. 298–311.

[33] P. Hansen and N. Mladenović, “First vs. best improvement : An empirical study,” *Discrete Applied Mathematics*, vol. 154, no. 5, pp. 802–817, 2006, iV ALIO/EURO Workshop on Applied Combinatorial Optimization. [Online]. Available : https://www.sciencedirect.com/science/article/pii/S0166218X05003070

[34] D. Whitley, A. Howe, and D. Hains, “Greedy or not ? best improving versus first improving stochastic local search for maxsat,” in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 27, no. 1, 2013.

[35] M. Basseur and A. Goëffon, “Hill-climbing strategies on various landscapes : an empirical comparison,” in *Proceedings of the 15th annual conference on Genetic and evolutionary computation*, 2013, pp. 479–486.

[36] G. Ochoa, S. Verel, and M. Tomassini, “First-improvement vs. best-improvement local optima networks of nk landscapes,” in *Parallel Problem Solving from Nature, PPSN XI*,

R. Schaefer, C. Cotta, J. Kołodziej, and G. Rudolph, Eds. Berlin, Heidelberg : Springer Berlin Heidelberg, 2010, pp. 104–113.

[37] M. Basseur and A. Goëffon, "On the efficiency of worst improvement for climbing nk-landscapes," in *Proceedings of the 2014 Annual Conference on Genetic and Evolutionary Computation*, 2014, pp. 413–420.

[38] S. Tari, M. Basseur, and A. Goëffon, "Worst improvement based iterated local search," in *European Conference on Evolutionary Computation in Combinatorial Optimization*. Springer, 2018, pp. 50–66.

[39] H. R. Lourenço, O. C. Martin, and T. Stützle, "Iterated local search," in *Handbook of metaheuristics*. Springer, 2003, pp. 320–353.

[40] J. Baxter, "Local optima avoidance in depot location," *Journal of the Operational Research Society*, vol. 32, no. 9, pp. 815–819, 1981.

[41] E. Baum, "Iterated descent : A better algorithm for local search in combinatorial optimization problems," *Manuscript*, 1986.

[42] E. B. Baum, "Towards practical 'neural'computation for combinatorial optimization problems," in *AIP Conference Proceedings*, vol. 151, no. 1. American Institute of Physics, 1986, pp. 53–58.

[43] O. Martin, S. W. Otto, and E. W. Felten, *Large-step Markov chains for the traveling salesman problem*. Citeseer, 1991.

[44] D. S. Johnson, "Local optimization and the traveling salesman problem," in *International colloquium on automata, languages, and programming*. Springer, 1990, pp. 446–461.

[45] O. C. Martin and S. W. Otto, "Combining simulated annealing with local search heuristics," *Annals of operations research*, vol. 63, no. 1, pp. 57–75, 1996.

[46] G. Ochoa and N. Veerapen, "Deconstructing the big valley search space hypothesis," in *Evolutionary Computation in Combinatorial Optimization*. Springer, 2016, pp. 58–73.

[47] C. Voudouris, E. P. Tsang, and A. Alsheddy, "Guided local search," in *Handbook of metaheuristics*. Springer, 2010, pp. 321–361.

[48] A. Davenport, E. Tsang, C. J. Wang, and K. Zhu, "Genet : A connectionist architecture for solving constraint satisfaction problems by iterative improvement," in *AAAI*, 1994, pp. 325–330.

[49] N. Mladenovic, "A variable neighborhood algorithm-a new metaheuristic for combinatorial optimization," in *papers presented at Optimization Days*, vol. 12, 1995.

[50] N. Mladenović and P. Hansen, "Variable neighborhood search," *Computers & operations research*, vol. 24, no. 11, pp. 1097–1100, 1997.

[51] J. H. Holland *et al.*, *Adaptation in natural and artificial systems : an introductory analysis with applications to biology, control, and artificial intelligence.* MIT press, 1992.

[52] J. Grefenstette, R. Gopal, B. Rosmaita, and D. Van Gucht, "Genetic algorithms for the traveling salesman problem," in *Proceedings of the first International Conference on Genetic Algorithms and their Applications*, vol. 160, no. 168. Lawrence Erlbaum, 1985, pp. 160–168.

[53] H. J. Bremermann *et al.*, "Optimization through evolution and recombination," *Self-organizing systems*, vol. 93, p. 106, 1962.

[54] I. Rechenberg, "Evolutionsstrategie : Optimierung technischer systeme nach prinzipien der biologischen evolution, frommann–holzboog," 1973.

[55] H.-P. Schwefel, *Numerische Optimierung von Computer-Modellen mittels der Evolutionsstrategie : mit einer vergleichenden Einführung in die Hill-Climbing-und Zufallsstrategie.* Springer, 1977, vol. 1.

[56] N. Hansen, S. D. Müller, and P. Koumoutsakos, "Reducing the time complexity of the derandomized evolution strategy with covariance matrix adaptation (cma-es)," *Evolutionary computation*, vol. 11, no. 1, pp. 1–18, 2003.

[57] T. Bäck, F. Hoffmeister, and H.-P. Schwefel, "A survey of evolution strategies," in *Proceedings of the fourth international conference on genetic algorithms.* Citeseer, 1991.

[58] S. Tari, M. Basseur, and A. Goëffon, "Sampled walk and binary fitness landscapes exploration," in *International Conference on Artificial Evolution (Evolution Artificielle).* Springer, 2017, pp. 47–57.

[59] W. Banzhaf, P. Nordin, R. E. Keller, and F. D. Francone, *Genetic programming : an introduction.* Morgan Kaufmann Publishers San Francisco, 1998, vol. 1.

[60] H. A. Kamal and M. H. Eassa, "Solving curve fitting problems using genetic programming," in *11th IEEE Mediterranean Electrotechnical Conference (IEEE Cat. No. 02CH37379).* IEEE, 2002, pp. 316–321.

[61] J. B. J. baron de Fourier, *Théorie analytique de la chaleur.* Firmin Didot, 1822.

[62] S. Verel, B. Derbel, A. Liefooghe, H. Aguirre, and K. Tanaka, “A surrogate model based on walsh decomposition for pseudo-boolean functions,” in *International Conference on Parallel Problem Solving from Nature*. Springer, 2018, pp. 181–193.

[63] S. S. Garud, I. Karimi, and M. Kraft, “Smart sampling algorithm for surrogate model development,” *Computers & Chemical Engineering*, vol. 96, pp. 103–114, 2017. [Online]. Available : https://www.sciencedirect.com/science/article/pii/S0098135416303210

[64] N. V. Queipo, R. T. Haftka, W. Shyy, T. Goel, R. Vaidyanathan, and P. K. Tucker, “Surrogate-based analysis and optimization,” *Progress in aerospace sciences*, vol. 41, no. 1, pp. 1–28, 2005.

[65] A. I. Forrester and A. J. Keane, “Recent advances in surrogate-based optimization,” *Progress in aerospace sciences*, vol. 45, no. 1-3, pp. 50–79, 2009.

[66] Y. Mack, T. Goel, W. Shyy, and R. Haftka, “Surrogate model-based optimization framework : a case study in aerospace design,” in *Evolutionary computation in dynamic and uncertain environments*. Springer, 2007, pp. 323–342.

[67] J. Leguy, T. Cauchy, M. Glavatskikh, B. Duval, and B. Da Mota, “Evomol : a flexible and interpretable evolutionary algorithm for unbiased de novo molecular generation,” *Journal of cheminformatics*, vol. 12, no. 1, pp. 1–19, 2020.

[68] L. P. Kaelbling, *Learning in embedded systems*. MIT press, 1993.

[69] J. Boyan and A. W. Moore, “Learning evaluation functions to improve optimization by local search,” *Journal of Machine Learning Research*, vol. 1, no. Nov, pp. 77–112, 2000.

[70] J. A. Boyan and A. W. Moore, “Learning evaluation functions for global optimization and boolean satisfiability,” in *AAAI/IAAI*, 1998, pp. 3–10.

[71] D. Whitley, S. Rana, J. Dzubera, and K. E. Mathias, “Evaluating evolutionary algorithms,” *Artificial intelligence*, vol. 85, no. 1-2, pp. 245–276, 1996.

[72] A. Sokolov and D. Whitley, “Unbiased tournament selection,” in *Proceedings of the 7th annual conference on Genetic and evolutionary computation*, 2005, pp. 1131–1138.

[73] J. R. Rice, “The algorithm selection problem,” in *Advances in computers*. Elsevier, 1976, vol. 15, pp. 65–118.

[74] K. Leyton-Brown, E. Nudelman, G. Andrew, J. McFadden, and Y. Shoham, “A portfolio approach to algorithm selection,” in *IJCAI*, vol. 3, 2003, pp. 1542–1543.

[75] F. Hutter, H. H. Hoos, K. Leyton-Brown, and T. Stützle, “Paramils : an automatic algorithm configuration framework,” *Journal of Artificial Intelligence Research*, vol. 36, pp. 267–306, 2009.

[76] H. H. Hoos, “Automated algorithm configuration and parameter tuning,” in *Autonomous search*. Springer, 2011, pp. 37–71.

[77] M. López-Ibánez, J. Dubois-Lacoste, L. P. Cáceres, M. Birattari, and T. Stützle, “The irace package : Iterated racing for automatic algorithm configuration,” *Operations Research Perspectives*, vol. 3, pp. 43–58, 2016.

[78] Y. Hamadi, E. Monfroy, and F. Saubion, “What is autonomous search ?” in *Hybrid optimization*. Springer, 2011, pp. 357–391.

[79] E. K. Burke, M. R. Hyde, G. Kendall, G. Ochoa, E. Özcan, and J. R. Woodward, “A classification of hyper-heuristic approaches : revisited,” in *Handbook of metaheuristics*. Springer, 2019, pp. 453–477.

[80] N. Belkhir, J. Dréo, P. Savéant, and M. Schoenauer, “Per instance algorithm configuration of cma-es with limited budget,” in *Proceedings of the Genetic and Evolutionary Computation Conference*, 2017, pp. 681–688.

[81] K. M. Malan and A. P. Engelbrecht, “A survey of techniques for characterising fitness landscapes and some possible ways forward,” *Information Sciences*, vol. 241, pp. 148–163, 2013.

[82] F. J. Poelwijk, S. Tănase-Nicola, D. J. Kiviet, and S. J. Tans, “Reciprocal sign epistasis is a necessary condition for multi-peaked fitness landscapes,” *Journal of theoretical biology*, vol. 272, no. 1, pp. 141–144, 2011.

[83] J. Garnier and L. Kallel, “Efficiency of local search with multiple local optima,” *SIAM Journal on Discrete Mathematics*, vol. 15, no. 1, pp. 122–141, 2001.

[84] M. Basseur, A. Goëffon, F. Lardeux, F. Saubion, and V. Vigneron, “On the attainability of nk landscapes global optima,” in *Seventh Annual Symposium on Combinatorial Search*. Citeseer, 2014.

[85] G. Ochoa, M. Tomassini, S. Vérel, and C. Darabos, “A study of nk landscapes’ basins and local optima networks,” in *Proceedings of the 10th annual conference on Genetic and evolutionary computation*, 2008, pp. 555–562.

[86] G. Ochoa and N. Veerapen, “Mapping the global structure of tsp fitness landscapes,” *Journal of Heuristics*, vol. 24, no. 3, pp. 265–294, 2018.

[87] W. B. Langdon, N. Veerapen, and G. Ochoa, “Visualising the search landscape of the triangle program,” in *European Conference on Genetic Programming.* Springer, 2017, pp. 96–113.

[88] G. Ochoa and N. Veerapen, “Additional dimensions to the study of funnels in combinatorial landscapes,” in *Proceedings of the Genetic and Evolutionary Computation Conference 2016*, 2016, pp. 373–380.

[89] H. H. Hoos and T. Stützle, *Stochastic local search : Foundations and applications.* Elsevier, 2004.

[90] F. Wilcoxon, “Individual comparisons by ranking methods,” in *Breakthroughs in statistics.* Springer, 1992, pp. 196–202.

[91] P. Auer, “Using confidence bounds for exploitation-exploration trade-offs,” *Journal of Machine Learning Research*, vol. 3, no. Nov, pp. 397–422, 2002.

[92] L. DaCosta, A. Fialho, M. Schoenauer, and M. Sebag, “Adaptive operator selection with dynamic multi-armed bandits,” in *Proceedings of the 10th annual conference on Genetic and evolutionary computation*, 2008, pp. 913–920.

[93] R. A. Gonçalves, C. P. Almeida, and A. Pozo, “Upper confidence bound (ucb) algorithms for adaptive operator selection in moea/d,” in *International Conference on Evolutionary Multi-Criterion Optimization.* Springer, 2015, pp. 411–425.

[94] S. Herrmann, G. Ochoa, and F. Rothlauf, “Communities of local optima as funnels in fitness landscapes,” in *Proceedings of the Genetic and Evolutionary Computation Conference 2016*, 2016, pp. 325–331.

[95] V. Hénaux, A. Goëffon, and F. Saubion, “Evolving fitness landscapes with complementary fitness functions,” in *International Conference on Artificial Evolution (Evolution Artificielle).* Springer, 2019, pp. 110–120.

[96] ——, “Evolving stochastic hill climbers,” in *International Workshop on Stochastic Local Search Algorithms*, 2019, pp. 21–22.

[97] ——, “From fitness landscapes evolution to automatic local search algorithm generation,” *International Transactions in Operational Research*, 2020.

[98] ——, “Evolution of deterministic hill-climbers,” in *2020 IEEE 32nd International Conference on Tools with Artificial Intelligence (ICTAI).* IEEE, 2020, pp. 564–571.

[99] ——, “Evolving search trajectories,” in *Proceedings of the 2020 Genetic and Evolutionary Computation Conference Companion*, 2020, pp. 101–102.

[100] "Satlib - benchmark problems," https://www.cs.ubc.ca/~hoos/SATLIB/benchm.html.

DOCTORAT BRETAGNE LOIRE MATHSTIC

université angers

Titre : Génération d'algorithmes de recherche locale

Mot clés : Recherche locale, Paysages de fitness, Algorithme évolutionnaire, Optimisation combinatoire

Résumé : Résoudre un problème d'optimisation consiste à en trouver les meilleures solutions possibles. Pour y parvenir, une approche commune est d'utiliser des algorithmes spécifiques, en général conçus pour des classes de problèmes précises. Cette approche souffre néanmoins de deux désavantages. D'abord à chaque nouveau type de problème, un nouvel algorithme doit souvent être défini, ce qui est un processus long, nécessitant une connaissance des propriétés du problème en question. Ensuite, si ces algorithmes ne sont testés que sur certaines instances du problème, il est possible qu'ils s'avèrent trop spécifiques et donc finalement moins performants sur l'ensemble des instances de la classe. Dans ce travail de thèse, nous explorons la possibilité de générer automatiquement des algorithmes d'optimisation pour un problème donné. Le processus de génération reste suffisamment générique tandis que les algorithmes ainsi produits peuvent être très spécifiques afin d'être les plus efficaces possibles.

Plus précisément, nous faisons évoluer de simples algorithmes de recherche par voisinage via les fonctions d'évaluation qu'ils utilisent pour explorer l'espace des solutions du problème. Le processus évolutionnaire permet implicitement d'adapter le paysage de recherche à la stratégie de résolution basique, tout en conservant une cohérence avec la fonction objectif initiale du problème à résoudre. Ce processus de génération est testé sur deux classes de problèmes dont les difficultés sont très différentes, et obtient des résultats encourageants. Cette expérimentation est complétée par une analyse du processus de génération et des algorithmes ainsi générés.

Title: Local search algorithms generation

Keywords: Local search, Fitness landscapes, Evolutionnary algorithm, Combinatorial optimization

Abstract: Solving an optimization problem is about finding the best possible solutions. To achieve this, a common approach is to use specific algorithms, usually designed for specific classes of problems. However, this approach suffers from two disadvantages. First with each new type of problem, a new algorithm often has to be defined, which is a lengthy process, requiring knowledge of the properties of the problem in question. Then, if these algorithms are only tested on certain instances of the problem, it is possible that they turn out to be too specific and therefore ultimately less efficient on all the instances of the class. In this thesis work, we explore the possibility of automatically generating optimization algorithms for a given problem. The generation process remains sufficiently generic while the algorithms thus produced can be very specific in order to be as efficient as possible.

More precisely, we develop simple neighborhood search algorithms via the evaluation functions that they use to explore the problem solution space. The evolutionary process implicitly makes it possible to adapt the search landscape to the basic solution strategy, while maintaining consistency with the initial objective function of the problem to be solved. This generation process is tested on two classes of problems whose difficulties are very different, and obtains encouraging results. This experiment is completed by an analysis of the generation process and of the algorithms thus generated.