RÉPUBLIQUE ALGÉRIENNE DÉMOCRATIQUE ET POPULAIRE
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

UNIVERSITE MOHAMED KHIDER
BISKRA

Faculté des Sciences Exactes et des Sciences de la Nature et de la Vie
Département d'informatique

N° Ordre :
N° Série :

Thèse

Présentée pour obtenir le diplôme de
Docteur en sciences
Spécialité : Informatique
par

Abdelkamel BEN ALI

Contributions à la résolution de problèmes d'optimisation combinatoires NP-difficiles

Soutenue le 19/04/2018

Membres du jury

Président	Mohamed Chaouki Babahenini	Professeur, Université de Biskra
Rapporteur	Kamal Eddine Melkemi	Professeur, Université de Biskra
Examinateurs	Foudil Cherif	Professeur, Université de Biskra
	Mohamed Khireddine Kholladi	Professeur, Université d'El-Oued
	Mohamed Chaouki Batouche	Professeur, Université de Constantine 2
	Souham Meshoul	Professeur, Université de Constantine 2

A ma chère mère qui croit toujours en moi

A ma femme qui a toujours était à mes cotés

A mes filles Rahaf et Raouane

A mon frère Lamine, ainsi que ses enfants Ziad et Razane

Résumé

Cette thèse porte sur des algorithmes efficaces pour la résolution de problèmes d'optimisation combinatoires NP-difficiles, avec deux contributions.

La première contribution consiste en la proposition d'un nouvel algorithme multiobjectif hybride combinant un algorithme génétique avec un opérateur de recherche basé sur l'optimisation par essaims de particules. L'objectif de cette hybridation est de surmonter les situations de convergence lente des algorithmes génétiques multiobjectifs lors de la résolution de problèmes difficiles à plus de deux objectifs. Dans le schéma hybride proposé, un algorithme génétique multiobjectif Pareto applique périodiquement un algorithme d'optimisation par essaim de particules pour optimiser une fonction d'adaptation scalaire sur une population archive. Deux variantes de cet algorithme hybride sont proposées et adaptées pour la résolution du problème du sac à dos multiobjectif. Les résultats expérimentaux prouvent que les algorithmes hybrides sont plus performants que les algorithmes standards.

La seconde contribution concerne l'amélioration d'un algorithme heuristique de recherche locale dit PALS (pour l'anglais *Problem Aware Local Search*) spécifique au problème d'assemblage de fragments d'ADN, un problème d'optimisation combinatoire NP-difficile en bio-informatique des séquences. Deux modifications à PALS sont proposées. La première modification permet d'éviter les phénomènes de convergence prématurée vers des optima locaux. La seconde modification conduit à une réduction significative des temps de calcul tout en conservant la précision des résultats. Après des expérimentations réalisées sur les jeux de données disponibles dans la littérature, nos nouvelles variantes de PALS se révèlent très compétitives par rapport aux variantes existantes et à d'autres algorithmes d'assemblage.

Mots clés : Algorithmes génétiques multiobjectifs Pareto, Optimisation par essaims de particules, Hybridation, Problème du sac à dos multiobjectif, Assemblage de fragments d'ADN, Recherche locale spécifique.

Abstract

The purpose of this thesis is to suggest efficient algorithms for solving NP-hard combinatorial optimization problems. We proposed two major contributions.

The first contribution is the development of a new hybrid multiobjective algorithm by combining a genetic algorithm (GA) with particle swarm optimization (PSO). The goal of this hybridization is to avoid the problem of the slow convergence of multiobjective GAs when solving optimization problems with more than two objectives. In the proposed synergetic hybrid scheme, a multiobjective GA does the multiobjective search, and activates periodically a PSO algorithm to optimize a scalar-valued fitness on its current archive population. Two combinatorial variants of this hybrid algorithm are developed and adapted to the multiobjective binary knapsack problem. Computational experiments manifest that our hybrid algorithms are more effective and efficient than the standard algorithms.

The second part of this thesis deals with the DNA fragment assembly problem, a NP-hard combinatorial optimization problem in bio-informatics. In this part we thought to fill the gaps of the most efficient heuristic algorithm for this problem in the literature, called the Problem Aware Local Search (PALS). Two modifications to the PALS heuristic are proposed in order to ameliorate its performance. The first modification enables the algorithm to improve the tentative solutions in a more appropriate and beneficial way. The second modification permits a significant reduction in the computational demands of the algorithm without significant accuracy loss. Computational experiments performed on available benchmark datasets show very competitive results when compared to other existing PALS variants and other assemblers.

Keywords: Pareto Multiobjective Genetic Algorithm, Particle Swarm Optimization, Hybridization, Multiobjective 0/1 Knapsack problem, DNA fragment assembly, Problem Aware Local Search.

ملخص

الهدف من أعمال مذكرة الدكتوراء هذه يتمثل في إقتراح طرق حل فعالة لمسائل التحسين الصعبة. مشاركتان علميتان أساسيتان تم إقتراحهما:

المشاركة الأولى تتمثل في إقتراح خوارزمية مهجنة لحل مسائل التحسين متعددة الأهداف، تضم خوارزمية وراثية متعددة الأهداف مع خورازمية التحسين بإستعمال سرب من الجسيمات. الهدف من وراء هذا المزج بين خوارزميتين مستوحاتيين من الطبيعة هو الحد من ظواهر التقارب البطيء للخورازميات الوارثية أثناء عملية حل المسائل الصعبة التي تضم أكثر من هدفين. في مخطط التحسين المهجن المقترح، خوارزمية وراثية متعددة الأهداف تقوم بشكل دوري بإستدعاء لخوارزمية التحسين بإستعمال سرب من الجسيمات لغرض إستغلال فئة من الحلول المنتخبة وفقا لدالة هدف واحدة تجمع بين جميع أهداف المسألة. لغرض قياس كفاءة وفعالية الخوارزمية المقترحة تم تطوير شكلين لها وتكيفهما لغرض حل مسألة حقيبة الظهر المتعددة الأهداف، النتائج التجريبية أثبتت تفوق كبير للخوارزميات المهجنة على الخوارزميات القاعدية.

المشاركة الثانية تتمثل في تحسين كفاءة وفعالية خوارزمية خاصة بمسألة تجميع قطع الحمض النووي (اي دي ان)، مسألة تحسين صعبة في مجال المعلوماتية-البيولوجية. هذه الخوارزمية الخاصة تعاني من مشاكل في التقارب أثاء التعامل مع مسائل صعبة ذات حجم كبير، فهي تعطي حلول مثلى محليا. قمنا بإقتراح تعديلين على هذه الخوارزمية لغرض تحسين أداءها. التعديل الأول يمس طريقة التعامل مع الحركات التي تطبق على الحل لغرض تحسينه، أما التعديل الثاني يسمح بتقليص زمن الحساب بدون المساس بدرجة دقة النتائج. النتائج التجريبية المتحصل عليها أثبتت نجاعة كبيرة للتعديلات المقترحة مقارنة بنتائج خوارزميات معدلة موجودة وخورزميات أخرى.

كلمات مفتاحية: الخوارزميات الوراثية متعددة الأهداف، خوارزمية التحسيين بإستعمال سرب من الجسيمات، التهجين بين الخوارزميات، مسألة حقيبة الظهر المتعددة الأهداف، مسألة تجميع قطع الحمض النووي، خوارزمية بحث محلي خاصة.

Remerciements

Merci tout d'abord à Kamal Eddine Melkemi, pour m'avoir soutenu et encadré au cours de ces années de thèse. Je le remercie également pour la confiance qu'il m'a témoignée et pour la liberté qu'il m'a accordée en recherche. Elles ont été pour moi une grande source de motivation.

Je remercie tout particulièrement monsieur Enrique Alba, professeur à l'Université Universidad de Málaga à l'Espagne, de m'avoir chaleureusement accueilli au sein de son équipe. Je remercie également monsieur Gabriel Luque, docteur à la même université, pour sa bonne humeur, ses conseils avisés et sa disponibilité. J'ai pris grand plaisir à travailler avec eux et j'espère que notre collaboration ne s'arrêtera pas là.

J'adresse mes plus sincères remerciements à Mohamed Chaouki Babahenini qui a accepté la fonction de Président du Jury. Je voudrais aussi exprimer ma gratitude à Mohamed Batocuhe, Souham Meshoul, Mohamed Khireddine Kholladi et Foudil Cherif, qui m'ont fait l'honneur de participer à mon jury et d'accepter de juger ce travail.

Merci aux personnes qui ont influencées, de près ou de loin, les travaux présentés dans cette thèse.

Table des matières

Liste des figures

Liste des algorithmes

Liste des tableaux

Liste des sigles

ACO	*Ant Colony Optimization*
ADN	Acide désoxyribonucléique
ERX	*Edge Recombination Crossover*
FAP	*Fragment Assembly Problem*
GA	*Genetic Algorithm*
GRASP	*Greedy Randomized Adaptive Search Procedure*
ILS	*Iterated Local Search*
KS	*Knapsack Problem*
MOGA	*Multiobjective Genetic Algorithm*
MOP	*Multiobjective Optimization Problem*
NSGA	*Nondominated Sorting Genetic Algorithm*
OLC	Overlap-Layout-Consensus
OX	*Order Crossover*
PALS	*Problem Aware Local Search*
PSO	*Particle Swarm Optimization*
SA	*Simulated Annealing*
SPEA	*Strength Pareto Evolutionary Algorithm*
TS	*Tabu Search*
TSP	*Travelling Salesman Problem*
VNS	*Variable Neighborhood Search*
VRP	*Vehicle Routing Problem*

Introduction générale

Les travaux de cette thèse s'intéressent à la résolution des problèmes d'optimisation mono-objectifs et multiobjectifs difficiles.

Contexte de recherche

L'optimisation combinatoire est un domaine de recherche très actif et très important. Elle couvre un large éventail des techniques et s'appuie particulièrement sur la recherche opérationnelle, les mathématiques et l'informatique. Elle s'applique à des domaines extrêmement variés de la vie courante : ingénierie, télécommunications, bioinformatique, logistique, transport, environnement, surveillance, data mining, énergie, commerce, finance, etc. Elle consiste, pour un problème donné, à chercher une meilleure configuration de l'ensemble de variables parmi un ensemble fini de configurations admissibles. En effet, une grande partie des problèmes du monde réel sont caractérisés par des espaces de recherche complexes et très larges. Ces sont les problèmes d'optimisation dits *difficiles*. Les méthodes de résolution exactes, qui font des recherches exhaustives de l'ensemble de solutions possibles, deviennent rapidement inutilisables avec la complexité du problème, au regard des temps de calcul qu'elles nécessitent.

Les algorithmes (méta)heuristiques sont largement adoptés pour la résolution approchée de problèmes d'optimisation difficiles. Néanmoins, les approches standards sont peu satisfaisantes sur de nombreux problèmes, soit parce qu'elles posent des difficultés de convergence lente sur des problèmes de grandes dimensions, soit parce qu'elles souffrent des phénomènes de convergence prématurée vers des solutions sous-optimales sur des problèmes caractérisés par des espaces de recherche complexes. Les

travaux de recherche récents se concentrent sur la conception des algorithmes hybrides combinant des éléments et concepts issus d'algorithmes d'optimisation différents, et sur l'exploitation des connaissances des problèmes traités pour concevoir des heuristiques de recherche. Les travaux réalisés au cours de cette thèse de doctorat s'inscrivent dans ces lignes de recherche. Les principales contributions sont présentées brièvement ci-dessous.

Contributions de la thèse

Les travaux présentés dans la première partie de cette thèse (les deuxième et troisième chapitres) se concentrent sur la proposition d'un nouvel algorithme métaheuristique hybride pour la résolution de problèmes d'optimisation multiobjectifs. Il s'agit d'un algorithme combinant un algorithme génétique multiobjectif avec un algorithme d'optimisation par essaims de particules. L'objectif de cette hybridation entre deux algorithmes métheuristiques est de s'affranchir des situations de convergence lente des algorithmes génétiques multiobjectifs lors de la résolution de problèmes complexes à plus de deux objectifs contradictoires. Nous avons conçu un algorithme génétique multiobjectif appliquant un opérateur de recherche basé sur l'optimisation par essaims de particules pour exploiter l'information d'une population d'élites. La recherche effectuée par l'algorithme génétique se base sur une approche de Pareto, tandis que l'opérateur de recherche par essaims de particules utilise une approche d'agrégation des objectifs pour transformer le problème multiobjectif en un problème mono-objectif. Le choix de l'algorithme d'optimisation par essaims de particule est justifié principalement par le fait qu'il permet d'exploiter rapidement l'information d'un ensemble de solutions pour trouver de nouvelles solutions améliorées. Deux variantes de cet algorithme hybride sont proposées et adaptées pour la résolution d'un problème d'optimisation combinatoire bien connu, le problème du sac à dos multiobjectif. Les résultats expérimentaux prouvent que les algorithmes hybrides testés sont beaucoup plus performants que les algorithmes de base. Les résultats préliminaires de cette forme d'hybridation ont fait l'objet d'une communication lors de la conférence CTAACS'13, à Skikda (Ben Ali et Melkemi, 2013).

Nous nous sommes intéressés dans la deuxième partie de cette thèse (les quatrième et cinquième chapitres) à l'utilisation des algorithmes (méta)heuristiques pour la résolution du problème d'assemblage de fragments d'ADN. C'est un problème d'optimisation combinatoire NP-difficile en bio-informatique des séquences. Il consiste à déterminer la séquence complète d'un génome à partir d'une collection d'un nombre très grand de petites séquences (appelées fragments). L'objectif est de minimiser le nombre de contigs (sous-séquences séparées) dans la séquence construite. Dans notre travail, nous nous sommes intéressés à l'amélioration des performances d'un algorithme heuristique de référence de la littérature, appelé PALS (de l'anglais, *Problem Aware Local Search*). Cet algorithme de recherche locale exploite les longueurs de chevauchement entre paires de fragments pour construire une séquence minimisant le nombre de contigs. En fait, l'algorithme PALS souffre des situations de convergence vers des optima locaux sur les problèmes de grande taille. Dans cette thèse, deux modifications à cet algorithme sont proposées pour remédier à ce problème de convergence. La première modification concerne la façon dont cet algorithme améliore les solutions au cours d'itérations successives. Plus précisément, les critères de sélection de mouvements à appliquer aux solutions candidates sont reformulés. La seconde modification concerne l'exploitation des invariants pour améliorer le temps de calcul. En comparaison avec l'algorithme de base et les variantes existantes, les variantes proposées se sont révélées être de loin le plus performantes. Cette étude a fait l'objet d'une publication dans une revue internationale (Ben Ali et al., 2017).

Organisation de la thèse

Ce manuscrit est organisé de la manière suivante :

Le premier chapitre est dédié à la présentation des notions et outils nécessaires à la bonne compréhension de cette thèse. Nous présentons tout d'abord les principales définitions et notions liées à la théorie de la complexité et à l'optimisation combinatoire. Nous présentons ensuite un état de l'art des algorithmes métaheuristiques de manière générale. Nous ferons en particulier une description détaillée de deux métaheuristiques à base de population principales : les algorithmes génétiques et l'optimisation par es-

saims de particules. Enfin, un état de l'art des algorithmes métaheuristiques hybrides est décrit.

Le deuxième chapitre présente, dans un premier temps, les concepts de base liés à l'optimisation multiobjectif, ainsi que les principales définitions. Dans un deuxième temps, nous présentons une généralisation des approches de résolution de problèmes l'optimisation multiobjectifs. Un troisième volet sera consacré à l'étude des algorithmes génétiques comme des techniques performantes qui possèdent des caractéristiques intéressantes pour résoudre les problèmes multiobjectifs difficiles. Nous en présenterons deux algorithmes génétiques multiobjectifs de référence de la littérature : NSGA-II et SPEA2. Ces algorithmes sont choisis pour servir de base à notre algorithme multiobjectif hybride présenté dans le troisième chapitre. Nous terminons en présentant deux métriques permettant d'évaluer le niveau de performance des ensembles de solutions.

Le troisième chapitre présente un nouvel algorithme hybride incorporant un opérateur de recherche basé sur l'optimisation par essaims de particules dans un algorithme génétique multiobjectif basé sur une approche Pareto. Après avoir motivé notre travail et nos choix, les différents éléments et étapes de cet algorithme multiobjectif hybride sont décrits. Deux variantes de cet algorithme hybride et une phase d'expérimentation sont proposées pour résoudre le problème du sac à dos multiobjectif. Les résultats obtenus sont comparés aux résultats d'algorithmes standards.

Le quatrième chapitre est consacré à une étude bibliographique du problème d'assemblage de fragments d'ADN. Ce chapitre fournit en particulier les éléments permettant d'appréhender le travail présenté dans le cinquième chapitre. Nous présenterons d'abord un aperçu général de ce qu'est la bio-informatique des séquences. Nous nous focaliserons par la suite sur le problème d'assemblage de fragments d'ADN, en présentant l'approche habituelle de sa résolution et en décrivant ses différents modèles formels. Trois types de graphes et leur lien à l'assemblage de génomes sont également décrits. Nous ferons ensuite un état de l'art des différents algorithmes heuristiques et métaheuristiques proposés pour résoudre le problème d'assemblage de fragments d'ADN. Cet état de l'art permet de décrire le contexte dans lequel se situent nos algorithmes d'assemblage développés au cours du cinquième chapitre. En dernier lieu,

nous présenterons les jeux de données les plus utilisés dans la littérature pour tester les algorithmes d'assemblage.

Le cinquième et dernier chapitre est consacré à la présentation de nos nouvelles variantes de l'algorithme PALS. Nous formulons d'abord les motivations et les objectifs de cette étude. Ensuite, après avoir décrit les variantes proposées, nous étudions leurs performances et comportements de façon comparative. Nous comparerons les résultats obtenus avec les résultats des variantes existantes et les résultats d'autres algorithmes d'assemblage de la littérature, sur les mêmes jeux de données.

Nous terminons cette thèse par une conclusion générale dans laquelle nous récapitulerons nos contributions, puis nous proposerons des perspectives pour la poursuite de ces travaux de recherche.

Chapitre 1

Optimisation combinatoire et algorithmes métaheuristiques

Nous présentons dans ce premier chapitre les notions et les outils nécessaires à la bonne compréhension de cette thèse. Nous introduisons d'abord les concepts de base en théorie de la complexité des algorithmes et des problèmes. Puis, nous donnons les principales définitions et notions liées à l'optimisation combinatoire. Nous décrivons ensuite les métaheuristiques les plus importantes pour la résolution de problèmes d'optimisation difficiles. On distingue entre deux grandes classes de métaheuristiques : les méthodes de recherche locale qui font évoluer itérativement une seule solution (dont les exemples typiques sont le recuit simulé, la recherche tabou, la recherche à voisinage variable, la recherche locale itérée et la méthode GRASP) et les méthodes qui manipulent une population de solutions (dont les algorithmes génétiques, l'optimisation par colonie de fourmis et l'optimisation par essaims de particules sont les représentant les plus célèbres). A la fin du chapitre, nous présentons un état de l'art des algorithmes métaheuristiques hybrides, qui ont été développées pour s'affranchir des limitations des métaheuristiques standards, en les combinant avec d'autres techniques d'optimisation ou en les optimisant par d'autres concepts.

1.1 Théorie de la complexité

La théorie de la complexité a été introduite afin de répondre à des questions centrales de l'informatique théorique : Quelles sont les limites des ordinateurs ? Quelles sont les limites fondamentales qui sont indépendantes de la technologie ? Quels sont les problèmes de calcul qui sont hors de portée ? La théorie de la complexité permet de déterminer et qualifier la difficulté intrinsèque des problèmes et fournit les outils mathématiques nécessaires à l'analyse des performances des algorithmes.

Les définitions données dans cette section sont inspirées du livre (Talbi, 2009).

1.1.1 Complexité des algorithmes

Un algorithme conçu pour résoudre un problème donné nécessite deux ressources importantes lorsqu'il s'exécute sur une machine : le temps CPU et l'espace mémoire. L'étude de la complexité des algorithmes (Papadimitriou, 2003) correspond à l'étude de l'efficacité comparée des algorithmes en termes de consommation de ces ressources de calcul. On parle ainsi de complexité temporelle (plus importante) et de complexité spatiale. L'évaluation de la complexité algorithmique peut se faire de façon expérimentale ou formelle. La complexité temporelle expérimentale consiste à mesurer le temps requis par un algorithme (implémenté sur une machine) lors de son exécution. Bien entendu, le temps d'exécution dépond des facteurs matériels et du langage de programmation utilisé. La démarche formelle est indépendante de ces caractéristiques techniques. Il s'agit d'estimer le coût d'un algorithme en fonction du nombre d'instructions nécessaires pour que l'algorithme retourne la solution du problème. La mesure de coût est donnée en termes de la taille des données, notée n, du problème en question. En d'autres termes, la fonction de complexité, notée $f(n)$, d'un algorithme fait correspondre pour des données de taille n le nombre d'opérations fondamentales qui lui est nécessaire pour résoudre une instance quelconque de cette taille. La complexité est généralement évaluée dans le pire cas.

L'objectif lors de la détermination de la complexité en temps d'un algorithme n'est pas d'obtenir une quantité exacte, mais une borne asymptotique sur le nombre d'opérations. En effet, évaluer la complexité d'un algorithme consiste à donner l'ordre

de grandeur du nombre d'opérations qu'il effectue lorsque la taille du problème qu'il résout augmente. Plusieurs notations asymptotiques sont utilisées pour donner des bornes sur l'ordre de grandeur et comparer les taux d'accroissement de différentes fonctions de coût. Le symbole asymptotique "O" (lit 'est grand oh de') est la plus populaire.

Définition 1.1 (Notation O) *Un algorithme a une complexité $f(n) \in O(g(n))$ s'il existe deux constantes positives c et n_0, tels que $\forall n > n_0, f(n) \leq c \cdot g(n)$.*

On dit que la fonction $f(n)$ est asymptotiquement majorée (ou dominée) par la fonction $g(n)$. La notation "O" est utilisée pour exprimer une borne supérieure sur la complexité des algorithmes.

La distinction entre *algorithmes polynomiaux* et *algorithmes exponentiels* est essentielle pour qualifier la difficulté intrinsèque des problèmes.

Définition 1.2 (Algorithme polynomial) *Un algorithme est dit en temps polynomial si sa complexité est en $O(p(n))$, où $p(n)$ est une fonction polynomiale par rapport à la taille des données n.*

Une fonction polynomiale de degré $k = 1, 2, \ldots$ peut être définie comme suit :

$$p(n) = a_k \cdot n^k + a_{k-1} \cdot n^{k-1} \ldots a_1 \cdot n + a_0$$

où $a_k > 0$. L'algorithme correspondant a une complexité $O(n^k)$.

Définition 1.3 (Algorithme exponentiel) *Un algorithme est dit en temps exponentiel si sa complexité est en $O(c^n)$, où c est une constante strictement supérieure à 1.*

Les algorithmes de complexité polynomiale sont *réalisables*, tandis que les algorithmes de complexité exponentielle sont généralement *irréalisables*.

1.1.2 Complexité des problèmes

La complexité d'un problème est équivalente à la complexité du meilleur algorithme (en termes de rapidité d'exécution) connu pouvant résoudre ce problème. On distingue ainsi des problèmes *traitables* (ou relativement faciles à résoudre) pour lesquels il existe des algorithmes de complexité polynomiale les résolvant et des problèmes *intraitables* (ou difficiles à résoudre) pour lesquels il n'existe aucun algorithme de complexité polynomiale[1].

La théorie de la complexité des problèmes concerne les problèmes de décision[2]. Certaines études l'ont étendu aux problèmes d'optimisation[3] (citons par exemple, (Cook, 1971, Garey et Johnson, 1979, Karp, 1972)). L'aspect important de la théorie de la complexité est de catégoriser les problèmes en des classes de complexité selon leur complexité en temps. Il y a deux grandes classes de problèmes de décision : la classe $\mathcal{P}$ (*Polynomial*) et la classe $\mathcal{NP}$ (*Non-déterministe Polynomial*).

- La classe $\mathcal{P}$ représente tous les problèmes de décision qui peuvent être résolus par une machine[4] déterministe en un temps polynomial. Un algorithme (déterministe) est polynomial si sa complexité en cas pires est bornée par une fonction polynomiale par rapport à la taille de l'instance du problème traité. Ainsi, la classe $\mathcal{P}$ correspond à l'ensemble des problèmes pour lesquels des algorithmes polynomiaux existent pour les résoudre.

- La classe $\mathcal{NP}$ contient tous les problèmes de décision qui peuvent être résolus par une machine non déterministe[5] en un temps polynomial (i.e., on peut tester la

[1] Il faut distinguer entre les problèmes intraitables pour des données de grande taille en raison de la croissance exponentielle de leur complexité en temps, des problèmes insolubles algorithmiquement. Ces derniers, dits *indécidables*, ne pourraient jamais avoir aucun algorithme pour les résoudre, même avec des ressources de calcul illimitées (Sudkamp, 2006). L'exemple le plus célèbre de problèmes indécidables est le problème d'arrêt ou "Halting problem" (Turing, 1937).

[2] Un problème de décision est un problème dont la réponse est "oui" ou "non".

[3] Un problème d'optimisation, c'est-à-dire dont la réponse est de type numérique, peut toujours être réduit à un problème de décision.

[4] En théorie de la complexité, les machines de Turing et les *Random Access Machines* servent d'étalon à la mesure des complexités en temps et en espace.

[5] Une machine non déterministe, contrairement à une machine déterministe, a plusieurs transitions possibles à partir desquelles plusieurs continuations différentes sont possibles sans aucune spécification de la transition qui sera prise.

validité d'une solution proposée (certificat) en un temps polynomial par rapport à la taille du problème).

La question de savoir si "$\mathcal{P} = \mathcal{NP}$" est centrale et ouverte en complexité. Elle prend de l'importance dès 1971 avec l'article (Cook, 1971). La réponse à cette question revient à savoir si trouver une solution est aussi efficace que de vérifier sa validité. Autrement dit, la réponse positive par "oui" revient à monter que tous les problèmes de décision de la classe $\mathcal{NP}$ sont aussi dans la classe $\mathcal{P}$. Bien évidemment, la classe $\mathcal{NP}$ englobe la classe $\mathcal{P}$, c'est-à-dire pour chaque problème de décision dans $\mathcal{P}$ il existe un algorithme non déterministe le résolvant. Alors, $\mathcal{P} \subseteq \mathcal{NP}$. Cependant, la conjecture $\mathcal{NP} \subset \mathcal{P}$ reste une question ouverte, qui résiste depuis plusieurs décennies.

Une autre question fondamentale en complexité consiste à savoir si un problème de $\mathcal{NP}$ est plus difficile que tous les autres. De tels problèmes définissent la classe *$\mathcal{NP}$-complet*.

Définition 1.4 ($\mathcal{NP}$-complet) *Un problème de décision* Π *est dit $\mathcal{NP}$-complet s'il appartient à la classe $\mathcal{NP}$, et tous les problèmes de la classe $\mathcal{NP}$ peuvent être réduits à* Π *en un temps polynomial.*

Un problème de décision Π *se réduit* à un problème de décision Π' s'il existe un algorithme polynomial qui transforme toute entrée u de Π en une entrée u' de Π', telle que u correspond à une instance positive (dont la réponse est "oui") de Π si et seulement si u' est une instance positive de Π'.

La classe $\mathcal{NP}$-complet fait référence aux problèmes les plus difficiles de $\mathcal{NP}$, pour lesquels on ne trouve jamais d'algorithmes polynomiaux pour les résoudre à l'optimalité avec une machine déterministe. Les problèmes $\mathcal{NP}$-complets sont équivalents par transformation polynomiale, ce qui implique la propriété intéressante suivante : si l'on trouvait un algorithme polynomial pour résoudre un problème $\mathcal{NP}$-complet quelconque, on pourrait en déduire des algorithmes polynomiaux pour tous les autres problèmes de cette classe, et on pourrait alors conclure que $\mathcal{P} = \mathcal{NP}$ (Solnon, 2005).

Définition 1.5 ($\mathcal{NP}$-difficile) *Les problèmes $\mathcal{NP}$-difficiles sont les problèmes d'optimisation dont les problèmes de décision associés sont $\mathcal{NP}$-complets.*

En fait, la plupart de problèmes d'optimisation réels sont $\mathcal{NP}$-difficiles et n'admettent pas des solutions algorithmiques efficaces valables pour résoudre des instances de grande taille (Talbi, 2009). La recherche de solutions exactes de ces problèmes requière des temps exponentiels. Les algorithmes métaheuristiques constituent une alternative importante pour la résolution approchée des problèmes $\mathcal{NP}$-difficiles.

1.2 Optimisation combinatoire

Dans cette section, nous donnons les notions de base en optimisation combinatoire, puis nous présentons une généralisation des différentes approches de résolution des problèmes d'optimisation difficiles.

1.2.1 Problème d'optimisation combinatoire

De nombreux problèmes d'optimisation pratiques et théoriques, consistent dans un ensemble de solutions admissibles, à trouver une meilleure solution (ou un ensemble de meilleures solutions). Une solution correspond à une configuration faisable de l'ensemble de variables du problème. L'ensemble de toutes les solutions possibles correspond à l'***espace de recherche***. L'optimalité de solutions est définie en terme de minimisation (ou maximisation) d'une fonction (équation mathématique) quantifiant le coût (ou la qualité) d'une solution. Cette fonction est appelée ***fonction de coût***, ***fonction objectif*** ou ***fitness***. Les problèmes d'optimisation se répartissent naturellement en deux grandes catégories : les problèmes à *variables réelles* et les problèmes à *variables discrètes*. Parmi les problèmes d'optimisation discrète, il y a une sous-catégorie de problèmes appelée ***optimisation combinatoire***. Les problèmes combinatoires sont caractérisés par des espaces *énumérables* de solutions possibles (Papadimitriou et Steiglitz, 1982). La solution de ce type de problèmes pourra être aussi bien un nombre entier, une chaîne de bits, un vecteur d'entiers, une structure de graphe, qu'une combinaison de ces représentations dans des structures complexes.

De manière très formelle, un problème d'optimisation combinatoire (mono-objectif) peut être défini de la manière suivante (inspirée de (Blum et Roli, 2003)) :

Définition 1.6 (Problème d'optimisation combinatoire) *Un problème d'optimisation combinatoire $(\mathcal{S}, f)$ est défini par :*

- *n variables de décision $X = (x_1, x_2, \ldots, x_n) \in \mathbb{Z}^n$;*
- *le domaine de variation (bornes explicites) de chaque variable de décision : $x_i \in D_i = [x_{i\_min}, x_{i\_max}]$ $(i = 1, 2, \ldots, n)$;*
- *contraintes à satisfaire par les variables de décision ;*
- *une fonction objectif f à minimiser[6], où $f : D_1 \times \cdots \times D_n \to \mathbb{R}$;*

L'espace de recherche (appelé aussi l'espace des paramètres, l'espace des configurations ou l'environnement) correspond à l'ensemble des valeurs qui peuvent être prises par les variables de décision :

$$\Omega = \{s = ((x_1, v_1), \ldots, (x_n, v_n)) \mid v_i \in D_i\}.$$

L'ensemble de toutes les configurations réalisables (appelé aussi l'espace réalisable ou l'espace de décision ou l'espace des solutions) est :

$$\mathcal{S} = \{s \in \Omega \mid s \text{ respecte toutes les contraintes}\}.$$

La résolution du problème $(\mathcal{S}, f)$ consiste à trouver une solution $s^* \in \mathcal{S}$ tel que $f(s^*)$ soit minimal.

$$s^* = \arg\min\{f(s) \mid s \in \mathcal{S}\}$$

La fonction objectif f permet de définir une relation d'ordre total entre les solutions de $\mathcal{S}$. La meilleure solution possible s^* est appelé l'***optimum***[7] ***global*** ou la ***solution optimale*** du problème.

[6]Notons que la recherche des solutions qui minimisent la fonction objectif est équivalente à la recherche des solutions qui maximisent l'inverse de la fonction objectif : $\min\{f(x)\} = \max\{-f(x)\}$. Pour cette raison, nous parlerons par la suite, sans perte de généralité, que de problèmes de minimisation.

[7]On dit aussi *minimum* (*maximum*) si le problème est posé en terme de minimisation (maximisation) d'une fonction objectif.

Définition 1.7 (Optimum global) *Une solution $s^* \in \mathcal{S}$ est globalement optimale ssi $\forall s \in \mathcal{S} : f(s^*) \leq f(s)$. On parle d'un optimum global strict si $f(s^*) < f(s)\ \forall s \in \mathcal{S}$.*

La solution qui est meilleure localement dans une région restreinte de l'espace de recherche est appelée un ***optimum local*** ou une ***solution sous-optimale***. La figure 1.1 illustre les notions d'optimum global et d'optimum local.

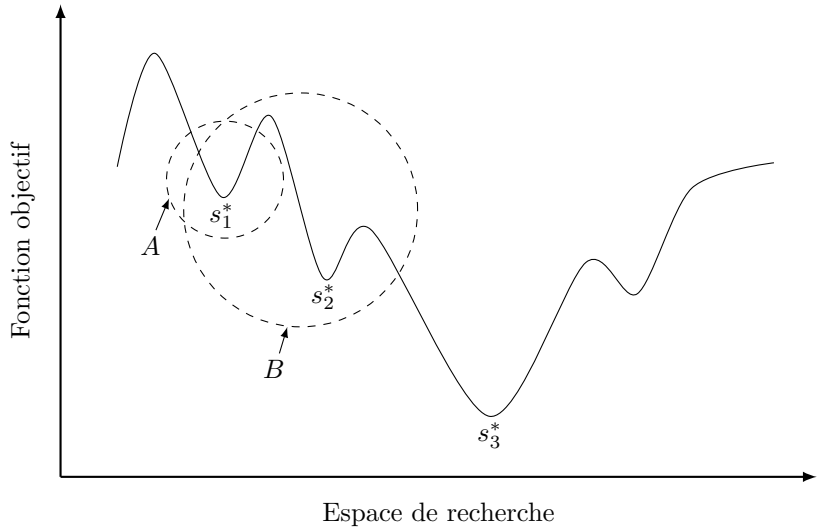


Figure 1.1 – Illustration de la différence entre un optimum global et un optima local. s_1^* est un optimum local de la région de solutions A ; s_2^* est un optimum local de la région de solutions B ; s_3^* est un optimum global.

En fait, la définition de la notion d'optima locaux nécessite que l'espace de solutions $\mathcal{S}$ soit topologique, i.e., sur lequel on peut définir des relations de ***voisinage***. Le voisinage d'une solution englobe les solutions voisines qui peuvent être générées en procédant de petits changements sur cette solution. Plus formellement, la notion de voisinage se définit de la manière suivante (en inspirant de (Blum et Roli, 2003)).

Définition 1.8 (Structure de voisinage) *Une structure (ou une relation) de voisinage est une fonction $\mathcal{N} : \mathcal{S} \rightarrow 2^{|\mathcal{S}|}$ qui assigne à chaque solution $s \in \mathcal{S}$ un ensemble de voisins $\mathcal{N}(s) \subseteq \mathcal{S}$. $\mathcal{N}(s)$ est dit le voisinage de s.*

Comme déjà mentionné, le plus souvent, les structures de voisinage sont implicitement définies par la spécification des changements qui doivent être appliqués à

une solution s pour générer toutes ses voisines. L'application d'un tel opérateur pour produire une solution voisine $s' \in \mathcal{N}(s)$ d'une solution s est communément appelée ***mouvement***.

La notion d'optimum local (ou de solution sous-optimale) relativement à un voisinage peut être définie formellement de la manière suivante.

Définition 1.9 (Optimum local) *Une solution $\hat{s} \in \mathcal{S}$ est localement optimale relativement à une relation de voisinage $\mathcal{N}$ ssi $\forall s \in \mathcal{N}(\hat{s}) \subset \mathcal{S} : f(\hat{s}) \leq f(s)$. On parle d'un optimum local strict si $f(\hat{s}) < f(s)\ \forall s \in \mathcal{N}(\hat{s})$.*

Notons que tout optimum global est évidement également un optimum local relativement à toute relation de voisinage.

On distingue généralement deux familles de problèmes d'optimisation combinatoire : les ***problèmes de permutation*** dont les solutions peuvent être modélisées par des permutations, et les ***problèmes binaires*** dont les solutions peuvent être modélisées à l'aide des structures binaires. On peut citer comme exemple de ces deux familles le problème du voyageur de commerce (*Travelling Salesman Problem* : TSP)[8], dans le premier cas, et, dans le second cas, le problème du sac à dos binaire (*Knapsack Problem* : KS)[9]. Les solutions du TSP peuvent être codées par des permutations (circulaires) de $\{1, 2, \ldots\ n\}$, où n est le nombre de villes à parcourir. Chaque élément dans la permutation identifie une ville, tandis que l'ordre des éléments de la permutation donne directement l'ordre dans lequel les villes sont parcourues. Pour un problème du TSP symétrique, $|\mathcal{S}| = (n-1)!/2$ correspond au nombre de tous les trajets possibles. Les solutions du KS peuvent être codées sous la forme des chaînes de bits de longueur de n, où n est le nombre d'objets. Le bit i dans la chaîne indique l'état de l'objet i ; 0 signifie que cet objet est sélectionné, 1 qu'il est écarté. La taille de l'espace de recherche du KS est $|\mathcal{S}| = 2^n$.

[8]Le problème du voyageur de commerce est un problème caractéristique d'optimisation combinatoire qui consiste à trouver le trajet de longueur minimale passant par toutes les villes et revenant au point de départ (distance euclidienne).

[9]Le problème du sac à dos binaire est un problème académique d'optimisation combinatoire qui consiste à remplir un sac d'une capacité limitée, par une partie d'un ensemble donné d'objets ayant chacun un poids et une valeur (prix ou profit). L'objectif est de maximiser la valeur totale des objets mis dans le sac, sans dépasser la capacité.

Une ***instance*** d'un problème d'optimisation est obtenue en donnent des valeurs numériques à tous les paramètres. En d'autres termes, la notion de problème réfère à une tâche abstraite générale à traiter, tandis que l'instance correspond à une instanciation concrète possible de cette tâche avec des données entièrement spécifiées. Par exemple, une instance du TSP est une liste de villes (une liste de points du plan). Une instance du KS correspond à un ensemble d'objets (poids et valeurs) avec une capacité du sac à dos.

1.2.2 Résolution des problèmes combinatoires difficiles

Les problèmes d'optimisation combinatoire sont le plus souvent très difficiles à résoudre de manière exacte en termes de temps de calcul. Pour de nombreux problèmes pratiques, la difficulté majeure est liée à l'explosion combinatoire des solutions possibles à explorer. En termes théoriques, comme nous avons vu dans la section 1.1.2, ces problèmes sont $\mathcal{NP}$-difficiles, pour lesquels on ne connaît pas d'algorithmes exacts en temps polynomial (Garey et Johnson, 1979). La célèbre conjecture $\mathcal{P} \neq \mathcal{NP}$ nous indique qu'il est peu probable que de tels algorithmes existent. La complexité en taille de l'espace de recherche peut aussi être due à de très grosses instances d'un problème facile (appartenant à la classe $\mathcal{P}$). En outre, la difficulté de résolution d'un problème à traiter pourrait aussi être liée à une structure complexe de l'espace de recherche et à une évaluation coûteuse de la fonction objectif. Ces difficultés excluent l'énumération exhaustive comme méthode de recherche de la solution. Pour cela, une grande variété d'approches mathématiques et d'algorithmes ont été proposés pour résoudre les problèmes d'optimisation combinatoires difficiles efficacement.

La littérature classe les méthodes d'optimisation en deux grandes familles : les méthodes *exactes* et les méthodes *approchées*. Les méthodes exactes garantissent de trouver des solutions optimales pour des instances de taille finie dans un temps limité, mais elles deviennent rapidement impraticables pour des instances de grande taille. Elles se basent sur la programmation dynamique ou utilisent des techniques d'énumération implicite comme les techniques de séparation et évaluation, aussi dites méthodes de *Branch & Bound*. Elles requièrent un temps exponentiel dans le pire des

cas. Les méthodes approchées constituent une alternative très intéressante lorsqu'on est confronté à des problèmes complexes de taille importante. Contrairement aux méthodes exactes, les méthodes approchées permettent d'obtenir des solutions de "bonne qualité" (dites approchées) en un temps de calcul *raisonnable*.

Les méthodes approchées sont classées dans la littérature selon deux grandes catégories : les *heuristiques* et les *métaheuristiques*. Les heuristiques sont des méthodes dédiées à un type de problème. Les métaheuristiques[10] sont des méthodes approximatives génériques, pouvant être adaptées et appliquées à différents problèmes d'optimisation. Il existe deux types de métaheuristiques : les métaheuristiques à base de solution unique, aussi appelées les méthodes de recherche locale, et les métaheuristiques à base de population de solutions. Dans la suite de ce chapitre, nous donnons une description simplifiée des métaheuristiques les plus importantes de chaque classe. Pour une présentation détaillée de ces métaheuristiques (et d'autres encore), voir par exemple (Blum et Roli, 2003, Boussaïd et al., 2013, Gendreau et Potvin, 2010a, Talbi, 2009).

Deux concepts fondamentaux dans le développement des métaheuristiques sont les notions de *diversification* (ou *exploration*) et d'*intensification* (ou *exploitation*) (Blum et Roli, 2003). La diversification consiste à échantillonner convenablement les différentes régions de l'espace de solutions, alors que l'intensification consiste à concentrer la recherche dans une région précise afin de trouver un optimum local. Ainsi, les algorithmes d'optimisation doivent assurer un bon équilibre entre l'intensification et la diversification de la recherche.

1.3 Métaheuristiques à solution unique

Les métaheuristiques à base de solution unique manipulent une seule solution durant le processus de recherche. Ces métaheuristiques sont aussi appelées les *méthodes de recherche locale* ou les *méthodes de trajectoire*. L'idée générale d'une recherche locale consiste à lancer un processus de recherche avec une seule solution initiale, puis à

[10]Le terme métaheuristique est dérivé de la composition de deux mots grecs. Le mot *heuristique* est dérivé du mot grec ancien *heuriskein* qui signifie "découvrir", tandis que le suffixe *méta*, qui vient aussi d'un mot grec ancien, signifie "au-delà" ou "à un plus haut niveau".

améliorer sa qualité au cours d'itérations successives, en se déplaçant dans un voisinage prédéfini de la solution courante.

Étant donnée une relation de voisinage $\mathcal{N}$, une méthode de recherche locale simple est définie de la manière suivante (voir l'algorithme 1.1). Dans un premier temps, une solution initiale est générée soit de manière aléatoire ou en utilisant une heuristique gloutonne ; voir fonction `GenererSolutionInitiale()`. Puis, à chaque itération de l'algorithme, une solution $s' \in \mathcal{N}(s)$ meilleure que s est choisie ; voir fonction `SelectionnerVoisinAmeliore(`$\mathcal{N}(s)$`)`. Cette nouvelle solution améliorée remplace la solution courante et on commence une nouvelle itération. La recherche continue jusqu'à ce qu'aucune amélioration ne soit plus possible. Ce procédé correspond à la méthode de recherche locale la plus simple, appelée dans la littérature la *méthode de descente*. Elle est également appelée *hill climbing* dans le contexte de problèmes posés en terme de maximisation.

Algorithme 1.1 : Algorithme de recherche locale

```
1 s ← GenererSolutionInitiale()
2 tant que Un optimum local relativement à N n'est pas atteint faire
3 |   s ← SelectionnerVoisinAmeliore(N(s))
4 fin
5 retourner s
```

Dans la littérature, il y a principalement deux façons d'implémenter la fonction `SelectionnerVoisinAmeliore(`$\mathcal{N}(s)$`)`. La première façon, appelée par la suite "*best-improvement*", consiste à choisir le meilleur voisin :

$$s' = \arg\min\{f(s'') \mid s'' \in \mathcal{N}(s)\}$$

Ce qui nécessite la génération et l'évaluation de fitness de tous les voisins possibles.

La seconde façon, appelée par la suite "*first-improvement*", consiste à générer les voisins, un par un, et à retourner le premier voisin qui améliore la qualité de la solution. Cette politique de recherche est utilisée lorsque la taille de voisinage est très importante et/ou l'évaluation de tous les voisins est très coûteuse en termes de temps de calcul.

En général, la performance d'une méthode de recherche locale dépend fortement de la structure de voisinage $\mathcal{N}$ utilisée. En fait, un algorithme de recherche locale partitionne l'espace de recherche en plusieurs *bassins d'attractions* d'optima locaux relativement au voisinage utilisé. Un bassin d'attraction $B(\widehat{s})$ d'un optimum local $\widehat{s} \in \mathcal{S}$ est un sous-ensemble de l'espace de recherche, i.e., $B(\widehat{s}) \subset \mathcal{S}$, tel que $\forall s \in B(\widehat{s}), f(s) \geq f(\widehat{s})$. Lorsqu'on lance une méthode de recherche locale à partir d'une solution $s \in B(\widehat{s})$, cette méthode retournera $\widehat{s}$ à la fin de recherche (si cette méthode n'est pas équipée d'un mécanisme d'échappement d'optima locaux).

Un autre concept fondamental utilisé dans l'étude et l'analyse du comportement des méthodes de recherche locale est la notion de *paysage de fitness* (ou, en anglais, *fitness landscape*) (Watson, 2010). Cette notion réfère à la structure topologique sur laquelle la recherche est exécutée. Etant donné une structure de paysage spécifique, définie à partir d'un espace de recherche, d'une fonction objectif, et d'un opérateur de voisinage, une méthode de recherche locale peut être vue comme une stratégie pour parcourir cette structure afin de trouver des solutions optimales ou sous-optimales.

Dans ce qui suit, nous présentons les métaheuristiques de recherche locale les plus importantes : le recuit simulé, la recherche locale itérée, la recherche à voisinage variable, la recherche tabou, et la méthode GRASP. Chacune de ces métaheuristiques possède son propre mécanisme pour échapper aux optima locaux (i.e., un mécanisme permettant de diversifier l'exploration de l'espace de recherche et d'augmenter la chance de rencontrer un optimum global).

1.3.1 Recuit simulé

Le recuit simulé (*Simulated Annealing* : SA) (Nikolaev et Jacobson, 2010) est une métaheuristique classique de recherche locale. Il trouve ses origines dans le formalisme de mécanique statistique (le critère d'acceptation de Metropolis (Metropolis et al., 1953)). Il a été présenté comme une méthode pour la résolution des problèmes d'optimisation combinatoires pour la première fois dans (Černỳ, 1985, Kirkpatrick et al., 1983). Le principe de base de SA est inspiré du processus de recuit physique utilisé en métallurgie pour améliorer la qualité d'un solide. Lors du passage d'un métal de l'état

liquide à l'état solide, le métal perd de l'énergie et prend finalement une structure cristalline. La perfection ou l'optimalité de cette structure cristalline dépond de la vitesse de refroidissement. Une diminution progressive de la température permet d'obtenir un minimum absolu d'énergie et donc un cristal bien structuré. En optimisation, la fonction objectif du problème traité est assimilée à l'énergie du matériau.

SA -est la première métaheuristique qui- applique une stratégie explicite afin d'éviter les optima locaux. Il s'agit d'une stratégie stochastique permettant le parcours d'un voisinage de qualité inférieure lorsqu'un optimum local est atteint. L'idée est d'effectuer un mouvement (i.e., déplacement dans le voisinage de la solution courante) selon une distribution de probabilité qui dépend de la qualité des différents voisins : les meilleurs voisins ont des probabilités plus élevées, tandis que les moins bons ont des probabilité plus faibles. L'algorithme de SA utilise un paramètre, appelé la *température* (par analogie avec l'inspiration naturelle), qui joue un rôle important. A haute température, toutes les solutions voisines ont à peu près la même probabilité d'être acceptées. A basse température, les mouvements qui détériorent la solution ont des faibles probabilités d'être appliqués. Alors que pour une température nulle, aucune dégradation de la qualité de solution n'est acceptée. Ainsi, la température est élevée au début de la recherche, puis diminue progressivement pour tendre vers 0 à la fin. De cette façon, la probabilité de dégradation de la solution diminue avec le temps.

Le pseudo-code de SA est donné dans l'algorithme 1.2. L'algorithme commence par générer une solution de départ et par initialiser la température (lignes 1 à 3). A chaque itération k de la boucle (lignes 4 à 13), une solution s' est générée dans le voisinage $\mathcal{N}(s)$ de la solution courante s (ligne 5) soit de façon aléatoire ou en utilisant une règle prédéfinie. Si s' est de meilleure qualité que s, alors s' devient la nouvelle solution courante (ligne 7). Sinon, si s' est moins bonne que s, s' sera acceptée avec une probabilité qui dépend de la température courante T_k, et de la différence entre les valeurs de fitness de s et s' ($f(s') - f(s)$; c'est le degré de dégradation de la solution) (ligne 9). Cette probabilité est usuellement calculée conformément à une distribution de Boltzmann $\exp(-\frac{f(s')-f(s)}{T_k})$. La température diminue après chaque itération de l'algorithme (ligne 11). Elle est contrôlée au cours de la recherche par une fonction décroissante qui définit un schéma de refroidissement.

Algorithme 1.2 : Algorithme de recuit simulé

```
1  s ← GenererSolutionInitiale()
2  k ← 0
3  T_k ← TemperatureInitiale()
4  tant que La condition d'arrêt n'est pas vérifiée faire
5  |  s' ← GenererSolutionVoisine(N(s))
6  |  si (f(s') < f(s)) alors
7  |  |  s ← s'
8  |  sinon
9  |  |  Accepter s' comme une nouvelle solution avec une probabilité :
   |  |  exp(-(f(s')-f(s))/T_k)
10 |  fin
11 |  T_{k+1} ← Refroidissement(T_k, k)
12 |  k ← k + 1
13 fin
14 retourner s
```

Le processus de recherche de l'algorithme de SA (i.e., la séquence des solutions générées) peut être modélisé par une chaîne de Markov (non homogènes) (Romeo et Sangiovanni-Vincentelli, 1991). Cela est dû au fait que la solution suivante est choisie en fonction uniquement de la solution courante. Ce qui signifie que l'algorithme de SA de base est dépourvu de mémoire.

1.3.2 Recherche locale itérée

La recherche locale itérée (*Iterated Local Search* : ILS) (Lourenço et al., 2010) est une métaheuristique de recherche locale. Le principe de cette algorithme itératif est simple : au lieu d'appliquer de façon répétitive une procédure de recherche locale à des solutions générées indépendamment les unes des autres, ILS génère les solutions de départ pour une procédure de recherche locale en perturbant aléatoirement les solutions sous-optimales trouvées précédemment. Cela veut dire que cet algorithme est plus performant qu'une recherche locale multi-départs utilisant la même heuristique ou procédure de recherche locale. L'étape de perturbation joue un rôle important, car elle permet d'explorer différentes régions de l'espace de solutions. Elle consiste à fournir une nouvelle solution initiale pour la recherche locale, afin de s'échapper des

bassins d'attraction d'optima locaux. Le mécanisme de perturbation doit respecter les exigences suivantes. La solution perturbée doit être située dans un bassin d'attraction différent de celui de l'optimum local actuel. En d'autres termes, le mécanisme de perturbation doit assurer que la procédure de recherche locale utilisée ne revient pas à l'optimum local actuel à partir de la solution perturbée. Cependant, dans le même temps, la solution perturbée doit être plus proche de la solution sous-optimale précédente que d'une solution générée de manière aléatoire. Ceci est pour éviter une recherche locale multi-départs à partir de solutions initiales générées aléatoirement.

Algorithme 1.3 : Algorithme de recherche locale itérée

```
1 s ← GenererSolutionInitiale()
2 s ← RechercheLocale(s)
3 tant que La condition d'arrêt n'est pas satisfaite faire
4 |   s' ← Perturbation(s, histoire)
5 |   s' ← RechercheLocale(s')
6 |   s ← CritereAcceptation(s, s', histoire)
7 fin
8 retourner s
```

Le pseudo-code de la recherche ILS est décrit par l'algorithme 1.3. L'algorithme commence par créer une solution initiale s ; voir fonction `GenererSolutionInitiale()`. Cette solution est ensuite améliorée via la procédure de recherche locale utilisée ; voir fonction `RechercheLocale(`s`)`. Après cette initialisation, l'optimisation se fait par itération de trois étapes. La première étape consiste à appliquer une procédure de perturbation sur l'optimum local actuel s pour obtenir une solution perturbée s' ; voir fonction `Perturbation(`$s, histoire$`)`. Le paramètre *histoire* réfère à l'influence possible de l'historique de recherche dans le processus. Comme déjà mentionné plus haut, le mécanisme de perturbation doit empêcher les cycles dans la recherche, c'est-à-dire, le retour à des optima locaux déjà visités. De plus, le *degré* de perturbation (i.e., la quantité de changement) doit être bien choisi. Ceci est dû au fait que, d'une part, une faible perturbation pourrait ne pas permettre à l'algorithme de s'échapper du bassin d'attraction actuel, et d'autre part, une forte perturbation pourrait rendre l'algorithme semblable à une recherche locale multi-départs à partir des solutions aléatoires. La deuxième étape consiste à appliquer la procédure de recherche locale sur la

solution perturbée pour obtenir un nouvel optimum local, stocké dans s'. La dernière étape consiste à choisir entre l'optimum local courant s et le nouvel optimum trouvé s' ; voir fonction `Acceptation`$(s, s', histoire)$. Plus usuellement, l'algorithme ILS choisit la meilleure solution entre s et s'. Néanmoins, d'autres critères d'acceptation qui exploitent l'historique de recherche peuvent être utilisés.

1.3.3 Recherche à voisinage variable

La recherche à voisinage variable (*Variable Neighborhood Search* : VNS) (Hansen et al., 2010) est une métaheuristique proposée par Mladenović et Hansen en 1995 dans (Mladenovic, 1995, Mladenović et Hansen, 1997). VNS est fondé sur l'idée de changement systématique de structure de voisinage. Le changement de voisinage intervient dans deux phases, dans une phase de recherche locale -qui choisit le meilleur voisin améliorant la solution courante (*best improvement*)- pour trouver des optima locaux, et dans une phase de perturbation pour s'échapper des vallées qui les contiennent.

Le pseudo-code de l'algorithme de VNS de base est décrit dans l'algorithme 1.4. L'algorithme travaille sur un ensemble préfini de structures de voisinage, souvent rangées par ordre croissant de leur cardinalité : $|\mathcal{N}_1| < |\mathcal{N}_2| < \cdots < |\mathcal{N}_{k_{max}}|$. On désigne par k l'indice du voisinage courant $\mathcal{N}_k$. L'algorithme part par une solution initiale s en ligne 1. Le voisinage est initialisé à $\mathcal{N}_1$ en ligne 3. Chaque itération de la boucle interne (des lignes 4 à 13) consiste en l'application d'une perturbation à s dans le $k^{\text{ième}}$ voisinage (ligne 5), suivie d'une procédure de recherche locale (*best-improvement*) appliquée à la solution perturbée s' (ligne 6). La recherche locale peut utiliser n'importe quelle structure de voisinage et n'est pas limitée à l'ensemble de voisinages $\mathcal{N}_k$ $(k = 1, 2, \ldots k_{max})$. Si la nouvelle solution obtenue s'' est de meilleure qualité que s, s'' remplace s (ligne 8) et le voisinage est réinitialisé à $\mathcal{N}_1$ (ligne 9). Sinon, l'algorithme considère le voisinage suivant $\mathcal{N}_{k+1}$ (ligne 11).

Le processus de changement de structure de voisinage en cas d'absence d'amélioration correspond à un mécanisme de diversification de la recherche. Comme décrit dans (Blum et Roli, 2003), l'efficacité de cette stratégie de changement systématique de voisinage peut s'expliquer par le fait qu'une “mauvaise” place sur le paysage de

Algorithme 1.4 : Algorithme de recherche à voisinage variable

Soit $\mathcal{N}_k$ $(k = 1, 2, \ldots, k_{max})$ un ensemble préfini de structures de voisinage
$s \leftarrow$ `GenererSolutionInitiale()`
tant que *La condition d'arrêt n'est pas vérifiée* **faire**
  $k \leftarrow 1$
  tant que $(k < k_{max})$ **faire**
    $s' \leftarrow$ `SelectionAleatoire(`$s, \mathcal{N}_k$`)`
    $s'' \leftarrow$ `RechercheLocale(`s'`)`
    si $(f(s'') < f(s))$ **alors**
      $s \leftarrow s''$
      $k \leftarrow 1$
    sinon
      $k \leftarrow k + 1$
    fin
  fin
fin
retourner s

recherche donné par un voisinage pourrait une "bonne"' place sur le paysage de recherche donné par un autre voisinage. En d'autres termes, des voisinages différents ne donneront pas forcément les mêmes paysages de recherche, et en conséquence une stratégie de recherche locale se comporte différemment sur chacun d'eux. Ainsi, le choix des structures de voisinage est un point critique de l'algorithme de VNS. Les structures de voisinage doivent exploiter les différentes propriétés et caractéristiques de l'espace de recherche, c'est-à-dire, elles doivent fournir des abstractions différentes de l'espace de recherche.

1.3.4 Recherche tabou

La méthode de recherche tabou (*Tabu Search* : TS) fait partie des algorithmes métaheuristiques les plus utilisés en optimisation combinatoire. Le premier article dans la littérature introduisant l'idée de base de TS a été publié en 1986 (Glover, 1986). Une présentation détaillée de cette méthode et de ses concepts fondamentaux peut être trouvée, par exemple, dans (Gendreau et Potvin, 2010b, Glover et Laguna, 1997, 2013). L'algorithme de TS exploite explicitement l'historique de la recherche, à la fois pour éviter les optima locaux, et pour mettre en œuvre une stratégie d'exploration de

l'espace de recherche. Comme l'algorithme de recuit simulé, TS accepte des solutions de qualité inférieure lorsqu'un optimum local est obtenu. Dans ce qui suit, nous nous inspirons surtout de (Blum et Roli, 2003) pour décrire cette méthode.

L'algorithme de TS simple applique une recherche locale de type *best-improvement* pour améliorer la solution courante, et utilise une mémoire -à court-terme- pour s'échapper des optima locaux et empêcher les cycles dans la recherche. Cette mémoire est mise en œuvre par une liste appelée *liste tabou* qui conserve les solutions les plus récemment visitées et interdit le retour à ces dernières. Les voisines de la solution courante sont donc limitées aux solutions qui n'appartiennent pas à la liste tabou, c'est-à-dire, les solutions qui n'ont pas encore été visitées. Dans ce qui suit, on utilise l'expression *ensemble autorisé* pour se référer à l'ensemble de solutions non visitées. Plus formellement, le fonctionnement canonique de la méthode de TS est décrit par l'algorithme 1.5. L'algorithme commence par générer une solution de départ s, initialiser la meilleure solution obtenue s_{opt} par s, et insérer s dans la liste tabou $ListeTabou$ (lignes 1-3). À chaque itération de l'algorithme, la meilleure solution voisine dans l'ensemble autorisé ($\mathcal{N}(s) \setminus ListeTabou$) est choisie, même si elle est moins bonne, comme la nouvelle solution courante s (voir fonction `MeilleurVoisin(`$\mathcal{N}(s) \setminus ListeTabou$`)`). Ensuite, cette solution est ajoutée à la liste tabou (voir fonction `MettreAJour(`$ListeTabou, s$`)`). Si la lite tabou est pleine, elle remplace un de ses éléments (généralement dans l'ordre FIFO, c'est-à-dire, la liste tabou est implémentée en utilisant une file).

Algorithme 1.5 : Algorithme de recherche tabou

```
1  s ← GenererSolutionInitiale()
2  s_opt ← s
3  ListeTabou ← {s}
4  tant que La condition d'arrêt n'est pas satisfaite faire
5  |  s ← MeilleurVoisin(N(s) \ ListeTabou)
6  |  MettreAJour(ListeTabou, s)
7  |  si (f(s) < f(s_opt)) alors
8  |  |  s_opt ← s
9  |  fin
10 fin
11 retourner s_opt
```

Comme déjà mentionné plus haut, l'utilisation d'une liste tabou interdit les déplacements vers des solutions récemment visitées ; donc elle empêche les cycles et force l'algorithme à accepter les déplacements non améliorants. La taille de la liste tabou (ou, en anglais, *the tabu tenure*) peut influencer fortement la recherche : d'un part, une taille trop petite forcera l'algorithme à explorer des régions plus vastes de l'espace de recherche ; d'autre part, une taille trop grande peut amener l'algorithme à exploiter une région locale, puisqu'elle interdit les déplacements vers un très grand nombre de solutions. La taille de la liste tabou peut évoluer au cours du temps de manière dynamique (Battiti et Tecchiolli, 1994). Cette stratégie diminue la taille de la liste tabu lorsque des cycles ont été détectés (pour apporter plus de diversification), et la augmente en cas d'absence d'amélioration (pour favoriser l'intensification).

Pour une implémentation plus pratique et efficace, les attributs des solutions visitées sont conservés au lieu des solutions complètes. Les attributs sont généralement des composants de solution, des mouvements, des différences entre solutions, etc. Puisque plus d'un attribut peut être considéré, une liste tabou est introduite pour chacun d'eux. L'ensemble autorisé est donc généré selon les conditions tabou définies par l'ensemble des attributs et les listes tabou correspondantes. Cette stratégie est plus efficace, mais entraine une perte d'information, car interdire un attribut implique probablement l'exclusion de plusieurs solutions. Ainsi, il est possible que des solutions non visitées de bonne qualité soient exclues de l'ensemble autorisé. Pour s'affranchir de ce problème, des critères dits d'*aspiration* sont définis pour autoriser l'acceptation d'une solution même si elle est interdite par les conditions tabou. Le critère d'aspiration le plus communément utilisé accepte les solutions qui sont meilleures que la solution optimale courante.

1.3.5 Méthode GRASP

La méthode GRASP (*Greedy Randomized Adaptive Search Procedure* : GRASP) (Resende et Ribeiro, 2010) est une technique de recherche locale conceptuellement simple, mais souvent efficace. Elle a été proposée dans (Feo et Resende, 1989, 1995). Cette métaheuristique itérative ou à départs multiples combine des heuristiques de

construction avec une étape de recherche locale. Chaque itération consiste en la construction d'une solution suivie d'une recherche locale appliquée à cette solution. La meilleure solution obtenue est retournée à la fin du processus de recherche. Un schéma général de la méthode GRASP est présenté dans l'algorithme 1.6.

Algorithme 1.6 : Algorithme GRASP

```
1 tant que La condition d'arrêt n'est pas satisfaite faire
2 |   s ← ConstructionGloutonneAleatoire()
3 |   s ← RechercheLocale(s)
4 |   si (f(s) < f(s_opt)) alors
5 |   |   s_opt ← s
6 |   fin
7 fin
8 retourner s_opt
```

La procédure principale de l'algorithme GRASP est la construction probabiliste de solutions. C'est une construction itérative, i.e., en partant d'une solution vide, et en ajoutant des composants de solution jusqu'à obtenir une solution complète. Cette construction qui n'utilise pas de mémoire[11] s'appuie sur une stratégie gloutonne aléatoire. A chaque étape de construction, les composants de solution à ajouter à la solution en cours de construction (i.e., une solution partielle) sont choisis à partir d'une liste appelée la liste de candidats restreints (de l'anglais *Restricted Candidate List* : RCL). Cette liste contient à chaque étape de construction un sous-ensemble de composants de solution qui peuvent être utilisés pour étendre la solution partielle. Les éléments de la liste RCL sont sélectionnés en fonction d'un critère heuristique qui estime localement la qualité des composants de solution. Après la détermination de la liste RCL, un composant de solution est choisi uniformément au hasard. Un paramètre important de l'algorithme GRASP est la taille de la liste RCL, notée α. Si $\alpha = 1$, la construction de solutions est déterministe et les solutions construites sont gloutonnes. Dans l'autre cas extrême, -c'est-à-dire, si la liste RCL contient toujours tous les composants de solution qui peuvent être ajoutés aux solutions partielles,- une

[11]En revanche, les algorithmes de colonies de fourmis (*Ant Colony optimization* : ACO), voir la section 1.4.3, construisent également des solutions de façon incrémentale, mais ils mémorisent et utilisent l'historique de la recherche.

solution aléatoire est générée sans aucune influence de la règle heuristique. De ce fait, α est un paramètre critique qui nécessite un réglage fin.

La seconde phase de la méthode GRASP consiste en l'application d'une procédure de recherche locale sur la solutions construite. Ceci peut être une recherche locale de base comme la descente, ou plus avancée comme un algorithme de recuit simulé ou une recherche tabou. D'après (Blum et Roli, 2003), l'algorithme GRASP peut être plus efficace si deux conditions sont satisfaites : (1) la stratégie de construction de solutions échantillonne des régions prometteuses de l'espace de recherche, et (2) les solutions construites sont de bons points de départ pour la recherche locale utilisée, i.e., les solutions construites sont situées dans des bassins d'attraction d'optima locaux de qualité élevée.

1.4 Métaheuristiques à population de solutions

Les métaheuristiques à base de population travaillent sur un ensemble (dit population) de solutions (on dit aussi individus). Comme elles manipulent une population de solutions, elles fournissent une capacité naturelle et intrinsèque pour l'exploration de l'espace de recherche. Leurs performances dépondent fortement de la façon dont la population est manipulée. Elles sont divisées principalement en deux grandes catégories : les algorithmes évolutionnaires et les algorithmes d'intelligence en essaim.

Les algorithmes évolutionnaires (*Evolutionary algorithms* : EAs) Ces algorithmes sont inspirés de la théorie darwinienne évolutionniste (*Néodarwinisme* ou *théorie synthétique de l'évolution*), i.e., la capacité d'une population biologique à s'adapter à son environnement complexe et généralement dynamique. Divers EAs ont été proposés dans la littérature. Cette grande catégorie comprend principalement la programmation évolutionnaire (*Evolutionary Programming* : EP) (Fogel et al., 1966), les stratégies d'évolution (*Evolutionary Strategies* : ES) (Rechenberg, 1973) et les algorithmes génétiques (*Genetic Algorithms* : GAs) (Goldberg, 1989, Holland, 1975). La EP a été originalement proposée pour faire évoluer des représentations discrètes d'automates à état fini. Les ES ont été développées pour la résolution des problèmes

d'optimisation en variables continues. Les GAs, qui seront détaillés dans la section 1.4.1, ont été appliqués principalement pour la résolution de problèmes d'optimisation combinatoires. D'autres techniques évolutionnaires ont été développés, comme la programmation génétique (*Genetic programming* : GP) (Koza, 1992) et la recherche dispersée (*Scatter Search* : SS) (Glover et al., 2000). Malgré la diversité des EAs, ils sont tous fondés sur les principes de l'évolution naturelle des êtres vivants, via les processus de sélection, recombinaison et mutation pour générer de nouveaux individus de bonne qualité.

Les algorithmes d'intelligence en essaim Ce type d'algorithmes s'inspire des comportements collectifs observés chez les insectes sociaux et certains animaux, qui émergent des comportements simples des individus. Les algorithmes d'intelligence en essaim mettent en jeu une population d'agents simples (petites entités homogènes capables de faire certaines opérations simples) en interaction locale l'un avec l'autre et avec leur environnement. Ces entités avec leurs capacités individuelles très limitées peuvent accomplir, conjointement, des tâches complexes nécessaires à leur survie. Bien qu'il n'existe aucune entité de contrôle centralisée qui distribue le travail entre les agents du système et surveille le processus de recherche, les interactions locales entre les agents conduisent souvent à l'*émergence* d'un comportement global organisé et adapté. Plusieurs métaheuristiques basées sur l'intelligence en essaim existent. Les deux principaux types sont l'optimisation par colonies de fourmis et l'optimisation par essaims de particules, qui seront détaillées dans les sections 1.4.2 et 1.4.3, respectivement. D'autres algorithmes ont été développés, comme l'algorithme de colonies d'abeilles, l'algorithme inspiré du système immunitaire et l'algorithme de l'optimisation par coucou. Pour une présentation des différentes métaheuristiques relavant de l'intelligence en essaim, nous référons le lecteur intéressé à (Blum et Li, 2008, Boussaïd et al., 2013, Engelbrecht, 2006).

1.4.1 Algorithmes génétiques (GAs)

1.4.1.1 Description

Les GAs sont les représentants les plus connus des méthodes évolutionnaires. Ils ont été popularisés par John Holland et ses collègues de l'université du Michigan (Goldberg, 1989, Holland, 1975) comme des méthodes fondées sur une analogie entre la résolution d'un problème d'optimisation et l'évolution d'une population biologique. Les individus d'une population biologique n'ont pas été "programmés" pour résoudre un problème spécifique, mais ils changent continûment en s'adaptant à leur environnement. La théorie de néodarwinisme a montré que cette caractéristique (d'adaptation à l'environnement ou d'évolution naturelle) des êtres vivants résulte de la combinaison d'une génération de diversité et du mécanisme de sélection naturelle. La diversité (i.e., variation ou différence entre individus) provient des mutations du caractère génétique et du mécanisme de la reproduction sexuée. Le mécanisme de sélection naturelle avantage les individus qui sont "le plus adaptés" dans le mécanisme de reproduction. Les GAs ont été construits sur la base de ce modèle biologique.

Maintenant, nous expliquons le schéma général de fonctionnement d'un GA. Après création d'une population initiale d'individus codant des solutions possibles pour le problème traité, un GA fait ensuite évoluer les individus par opérateurs d'***évolution*** et ***sélection*** selon une ***fonction d'adaptation*** (fitness), jusqu'à produire une population d'individus codant de bonnes solutions. Le pseudo-code d'un GA est illustré dans l'algorithme 1.7. La population initiale constitue le point de départ de l'algorithme (ligne 1). Elle est usuellement générée aléatoirement, bien que des connaissances du domaine du problème traité puissent mener au développement de GAs efficaces (par exemple, en injectant des solutions gloutonnes dans la population initiale). Après génération de la population initiale, les valeurs d'adaptation de tous les individus sont évaluées (ligne 2). L'étape d'évaluation consiste à calculer la valeur d'adaptation de chaque individu de la population en fonction de son chromosome[12]. Le cycle de reproduction (boucle des lignes 3 à 9) est lié à la génération d'une nou-

[12]Pour les problèmes mono-objectifs, les valeurs d'adaptation sont le plus souvent données par la fonction objectif du problème traité.

velle population. Cette phase comprend la sélection des parents géniteurs (ligne 4), leur combinaison (ligne 5), la mutation des enfants obtenus (ligne 6) et, ensuite, l'évaluation de ces derniers (ligne 7). L'opérateur de sélection et les opérateurs d'évolution (***croisement*** et ***mutation***) sont typiques dans les GAs. La nouvelle population générée (*enfants*) dans ce cycle de reproduction est utilisée en combinaison avec la population courante (P) pour déterminer la population de la génération suivante (ligne 8). L'algorithme retourne à la fin de recherche la meilleure solution trouvée.

Algorithme 1.7 : Pseudo-code d'un algorithme génétique

```
1  P ← GenererPopulationInitiale(N)
2  Evaluation(P)
3  tant que La condition d'arrêt n'est pas vérifiée faire
4  |  parents ← SelectionParents(P)
5  |  enfants ← Croisement(parents, Pc)
6  |  enfants ← Mutation(enfants, Pm, Pm_g)
7  |  Evaluation(enfants)
8  |  Remplacement(P, enfants)
9  fin
10 retourner la meilleure solution trouvée
```

Dans les paragraphes suivants, nous reviendrons plus en détail sur les éléments essentiels d'un GA ainsi que ses principaux paramètres de réglage.

1.4.1.2 Codage de la solution

La représentation des solutions (ou le codage) est la première étape dans la résolution d'un problème d'optimisation en utilisant les GAs. Chaque solution possible au problème traité doit être codée (de manière *univoque*, si c'est possible) en une chaîne de longueur finie, appelée ***chromosome***, ***génotype*** ou ***individu***. Le chromosome pourra être aussi bien une liste d'entiers, un vecteur de nombres réels ou une chaîne de bits, qu'une combinaison de ces représentations dans des structures complexes. Le choix du type de codage à utiliser dépend des caractéristiques du problème traité. Les éléments du chromosome (i.e., les variables de décision du problème traité) sont appelés les ***gènes***. Les valeurs possibles des gènes sont appelées les ***allèles***.

Pour les problèmes d'optimisation en variables discrètes, on distingue deux types principaux de représentation :

- ***la représentation binaire*** : pour de nombreux problèmes d'optimisation combinatoires la solution peut être représentée par une chaîne de bits, où chaque gène prend seulement les valeurs 0 ou 1. Par exemple, les solutions du problème du sac à dos binaire peuvent être modélisées par une chaîne de bits. Chaque bit dans la chaîne indique l'état d'un objet particulier de l'ensemble d'objets. 1 signifie que cet objet est sélectionné, 0 qu'il est écarté. La figure 1.2 illustre une solution possible de ce problème avec un ensemble de 10 objets. Cette solution prend les objets 1, 4, 6, 7 et 9 et écarte les objets 2, 3, 5, 8, 10.

1	0	0	1	0	1	1	0	1	0

Figure 1.2 – Codage binaire.

- ***la représentation par permutation*** : ici, les gènes sont des nombres ou des lettres, mais la valeur de chaque gène dans le chromosome doit être unique. L'ordre dans la séquence de gènes est signicatif. Ce type de codage est beaucoup plus efficace pour certain type de problèmes discrets, comme les problèmes d'ordonnancement (ou *scheduling problems*), le problème du voyageur de commerce (ou *Travelling Salesman Problem* : TSP) et le problème de tournées de véhicules (ou *Vehicle Routing Problem* : VRP). La figure 1.3 illustre une représentation chromosomique par une liste d'entiers caractérisant l'ordre de parcours des différentes villes pour TSP avec 12 villes.

10	4	6	11	9	1	7	2	8	3	12	5

Figure 1.3 – Codage de permutation.

1.4.1.3 Sélection

L'opérateur de sélection consiste à choisir, en fonction de leur valeur d'adaptation respective, les individus qui survivent à la génération suivante. Autrement dit, les

individus inadaptés à l'environnement ne survivent pas à la génération suivante. La sélection s'intervient aussi au début de la génération pour sélectionner les meilleurs individus, appelés ***parents géniteurs*** (ou *Pool mating*), en vue d'un croisement puis d'une mutation.

La sélection a pour but d'orienter la recherche vers les solutions les mieux adaptées. Nous citons ci-dessous quelques stratégies de sélection plus connues (Mais et al., 2010).

1. Sélection de la roulette (*Roulette Wheel Selection*) La population est représentée comme une roue de roulette, où chaque individu occupe un secteur dont l'angle est proportionnel à sa valeur de fitness. La largeur du secteur pour un individu sera d'autant plus importante que la valeur de fitness le sera ; plus le secteur est important plus l'individu aura de chance d'être sélectionné. La sélection d'un individu se fait en tournant la roue en face d'un pointeur fixe. La sélection est répétée jusqu'à l'obtention du nombre d'individus requis.

Concrètement, la probabilité de sélection d'un individu $i = 1, 2, \ldots, N$ (où N est la taille de la population) est calculée comme suit (au cas de problème de minimisation[13]) :

$$p_i = \frac{f(i)^{-1}}{\sum_{j=1}^{N} f(j)^{-1}} \tag{1.1}$$

Un individu $i = 1, 2, \ldots, N$ est sélectioné si un nombre aléatoire tiré de l'intervalle $[0, 1]$ appartient à l'intervalle $\left[\sum_{j=1}^{i-1} p_j, \sum_{j=1}^{i} p_j\right]$.

Lorsque la fonction objectif a une amplitude (i.e., distance entre ses valeurs extrêmes) très élevée, cet opérateur de sélection risque d'écarter les individus de mauvaise qualité.

2. Sélection par rang (*Ranking selection*) Tous les individus sont d'abord rangés par ordre croissent (au cas de problème de minimisation) selon leurs valeurs de fitness. Les rangs premiers (à partir de 1) seront attribués aux individus de meilleure qualité. Le rang N sera alors attribué au mauvais individu. Les probabilités de sélection des individus sont données par leur rang dans le classement. Concrètement, la

[13]Au cas de problème de maximisation, on utilise la même formule mais sans inverser la fonction de fitness.

probabilité de sélection d'un individu $i = 1, 2, \ldots, N$ peut être donnée par (Alba et Dorronsoro, 2009a) :

$$p_i = \frac{2 \cdot (N - j)}{N \cdot (N - 1)} \tag{1.2}$$

où $j = 1, 2, \ldots, N$ est le rang dans le classement. Par exemple, pour $N = 5$, les probabilités de sélection sont 0.4, 0.3, 0.2, 0.1 et 0.0 pour les individus de rang 1, 2, 3, 4, et 5, respectivement. Cette méthode de sélection augmente un peu la chance de sélectionner les individus de mauvaise qualité.

3. Sélection par élitisme Seuls les meilleurs individus seront sélectionnés (i.e., la partie supérieure de la population). Cette méthode pose comme inconvénient le risque de convergence rapide.

4. Sélection par tournoi (*Tournament selection*) Cette méthode consiste à choisir, le meilleur individu parmi deux ou plusieurs individus sélectionnés aléatoirement de la population. Cette sélection est répétée jusqu'à l'obtention du nombre d'individus requis.

1.4.1.4 Croisement

L'opérateur de croisement (*crossover* ou recombinaison) permet de produire, à partir de deux individus "parents" sélectionnés par un opérateur de sélection, un ou deux nouveaux individus "enfants" par échange de sous-chaînes de gènes. En d'autres termes, cet opérateur permet aux enfants d'hériter une partie de chromosome du premier parent et l'autre partie du deuxième parent. Cet opérateur d'évolution, dont le rôle est de favoriser l'émergence de nouveaux individus de bonne qualité, est généralement appliqué avec une probabilité élevée (donnée par le paramètre Pc dans l'algorithme 1.7). Plusieurs méthodes de croisement ont été définies dans la littérature, qui dépendent essentiellement du type du codage utilisé et de la nature du problème traité. Nous illustrons ci-dessous quelques types de croisement plus utilisés pour la représentation binaire et la représentation de permutation (Mais et al., 2010).

1. *Ordre Crossover (OX)* L'opérateur OX est utilisé pour le codage de permutation. Il choisit deux points aléatoires de croisement. Le premier enfant hérite les éléments situés entre les deux locus de croisement de la permutation du premier parent. Ces éléments occupent les mêmes positions dans l'enfant et donc apparaissent dans le même ordre que dans le parent. Les éléments restants sont hérités du second parent dans l'ordre dans lequel ils apparaissent dans ce dernier, en commençant par la première position suivant le deuxième point de croisement et en omettant tous les éléments déjà présents dans l'enfant. Le deuxième enfant est construit de manière symétrique en inversant le rôle des parents. La figure 1.4 illustre un exemple de ce type de croisement.

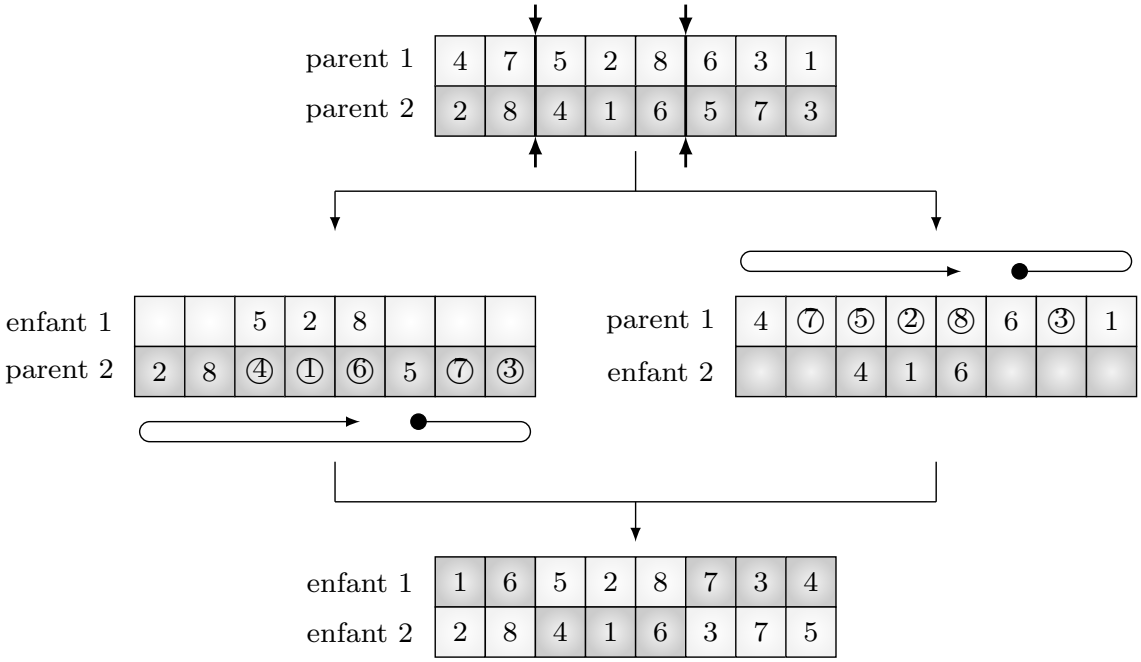


Figure 1.4 – Codage de permutation — Croisement OX.

2. *One point Crossover (1X)* Le croisement en un point représente l'opérateur de croisement le plus simple. Cet opérateur a été appliqué initialement pour le codage binaire. Il choisit dans un premier temps un seul point de croisement aléatoirement pour couper le chromosome de chaque parent en deux parties. Puis, le premier enfant est construit en combinant la première partie du premier parent avec la seconde

partie du second parent. A l'inverse, le deuxième enfant est la concaténation de la première partie du second parent et de la seconde partie du premier parent. Comme le montre figure 1.5, cette opération est simple pour le codage binaire. Pour produire des solutions valides pour le codage de permutation, cet opérateur est adapté comme suit. Pour construire le premier enfant, la première partie du premier parent est recopiée, puis ensuite les éléments de la seconde partie de ce parent sont réordonnés selon leur ordre d'apparition dans le second parent. Le second enfant est généré de maniéré symétrique, en échangeant le rôle des parents, comme le montre le figure 1.6.

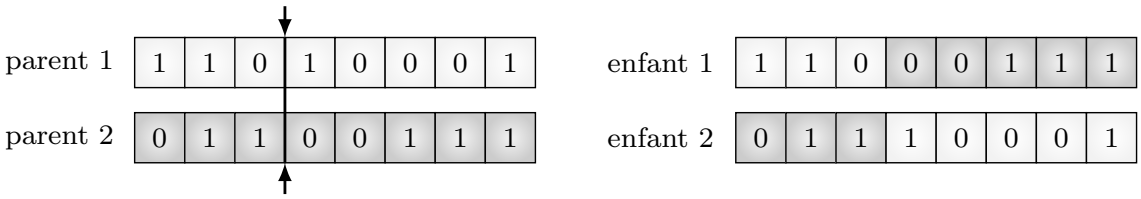


Figure 1.5 – Codage binaire — Croisement 1X.

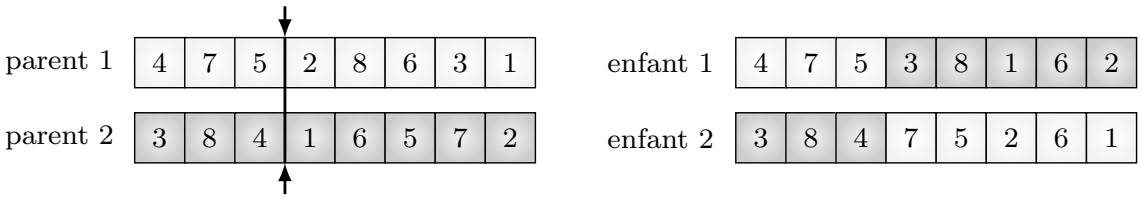


Figure 1.6 – Codage de permutation — Croisement 1X.

3. *N point Crossover (nX)* Cet opérateur applique la même règle du croisement 1X mais pour n points de croisement. Ces derniers sont choisis aléatoirement pour couper chaque parent en $n+1$ parties. Pour le codage binaire, le premier enfant hérite les éléments des parties impaires du premier parent et les éléments des parties paires du second parent. Pour construire le second enfant, les rôles des parents sont inversés. La figure 1.7 illustre un exemple d'un croisement à deux points binaire, qui consiste à échanger simplement les sous-chaînes des chromosomes parents situées entre les deux locus de croisement. Dans le cas d'un codage de permutation, pour construire le premier enfant, les éléments des parties impaires du premier parent sont recopiés, puis ensuite les éléments restants de chaque partie paire sont réordonnés dans le même

ordre que dans le second parent. De même, pour le second enfant, les rôles des parents sont inversés, comme le montre la figure 1.8.

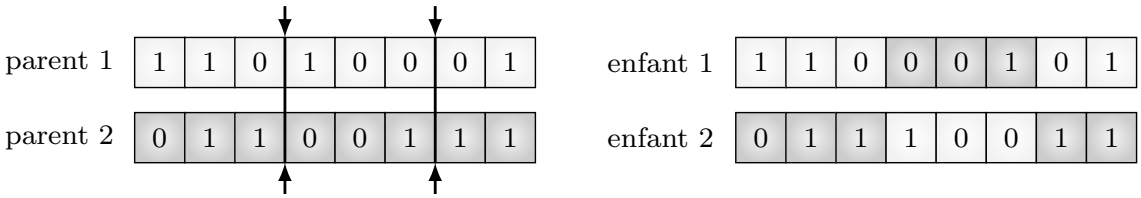


Figure 1.7 – Codage binaire — Croisement 2X.

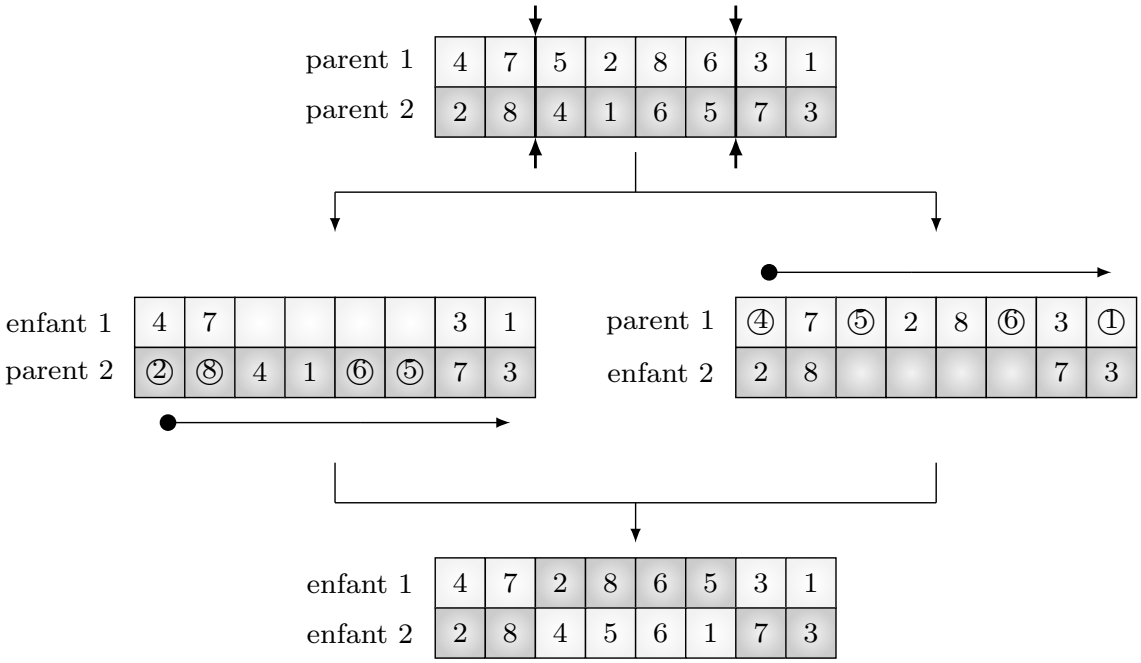


Figure 1.8 – Codage de permutation — Croisement 2X.

4. *Edge Recombination Crossover (ERX)* L'opérateur de croisement ERX a été établi pour la représentation par permutation (Whitley et al., 1989). Il utilise les listes d'adjacences pour construire un enfant qui hérite la meilleure information possible des structures des parents. Ces listes stockent les relations d'adjacences entre les éléments des deux parents. Pour le problème du TSP (symétrique), ces relations représentent les arcs entrants ou sortants dans une ville. Puisque la distance entre deux villes est indépendante du sens de parcours, chaque ville aura deux relations

d'adjacences au minimum et quatre au maximum (deux par parent). L'exemple donné ci-après a été fourni dans (Whitley et al., 1991) pour illustrer la construction des listes d'adjacences et l'opération de recombinaison, dans le cadre de la résolution du problème TSP. Les deux trajets parents considérés sont : [A B C D E F] et [B D C A E F]. Les listes d'adjacences pour les éléments de ces trajets sont :

A a la liste : B F C E	D a la liste : C E B
B a la liste : A C D F	E a la liste : D F A
C a la liste : B D A	F a la liste : E A B

L'algorithme d'ERX est le suivant (Whitley et al., 1991) :

1. Choisir une ville parmi les deux premières villes des parents (Elle peut être sélectionnée aléatoirement ou selon des critères décrits dans l'étape 4). Cette ville est la ville courante.

2. Supprimer toutes les occurrences de la ville courante de toutes les listes d'adjacence.

3. Si la ville courante a des relations d'adjacence dans sa liste, passer à l'étape 4 ; sinon aller l'étape 5.

4. Parmi les villes dans la liste d'adjacences de la ville courante, déterminer la ville qui a le moins de relations d'adjacence dans sa propre liste. Cette ville devient la ville courante. Dans le cas où il y a deux ou plusieurs villes ont le même nombre minimum de relations, sélectionner une ville au hasard parmi les candidates. Retourner à l'étape 2.

5. Si il y n'a aucune ville non visitée, arrêter. Sinon, sélectionner une ville non-visitée aléatoirement et retourner à l'étape 2.

Sur l'exemple proposé, l'opérateur ERO procède comme suit :

1. Puisque les deux premières villes A et B des trajets parents ont chacune quatre relations d'adjacence, la première ville de l'enfant doit être choisie aléatoirement parmi ces deux villes. On suppose que la ville B est choisie et supprimée de toutes les listes.

2. A partir de la liste d'adjacences de la ville B, les villes candidates pour la ville suivante sont A, C, D et F. Chacune de trois villes C, D, et F a deux relations dans sa liste. La ville A est écartée puisqu'elle a trois relations dans sa liste. On suppose que la ville C est choisie aléatoirement.

3. La ville C a dans sa liste les deux villes A et D. La ville D est ajoutée à l'enfant puisqu'elle a le moins de relations.

4. Puisque la ville D a une seule relation d'adjacence avec E, cette dernière est choisie comme l'élément suivant de l'enfant.

5. La ville E a deux villes voisines dans sa liste, qui sont A et F. Ces deux villes ont chacune une seule ville voisine. On suppose que la ville A est choisie au hasard.

6. La seule ville qui reste, F, est ajoutée au trajet enfant.

Le nouveau trajet obtenu est $[B\ C\ D\ E\ A\ F]$ qui se compose complétement de relations issues des deux trajets parents.

1.4.1.5 Mutation

L'opérateur de mutation est appliqué, lors de la phase de reproduction, aux enfants produits par l'opérateur de croisement avec une certaine probabilité (donnée par le paramètre Pm dans l'algorithme 1.7). La mutation a pour but de faire émerger de nouvelles informations dans la population. En d'autres termes, elle a pour but de générer des enfants dont certains gènes ne sont pas hérités de la population initiale, ce qui n'est pas atteint par le croisement. Cet opérateur consiste donc à modifier légèrement un ou plusieurs gènes du chromosome, choisis aléatoirement. La probabilité de changement de chaque gène est donnée par un paramètre de contrôle de l'algorithme (paramètre Pm_g dans l'algorithme 1.7). L'opérateur de mutation dépond du type de codage utilisé. Nous illustrons par la suite quelques opérateurs de mutation plus communément utilisés pour le codage binaire et le codage de permutation.

La figure 1.9 donne un exemple de mutation ***Bit-flip*** sur un codage binaire. Cet opérateur consiste à inverser certains bits dans la chaîne binaire.

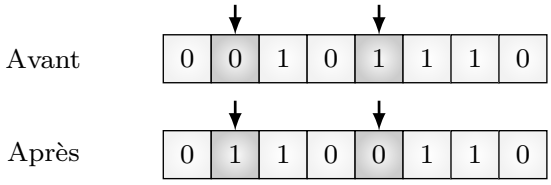


Figure 1.9 – Opérateur de mutation binaire *Bit-flip*.

La figure 1.10 donne des exemples de trois types de mutation pour le codage de permutation : ***échange*** (***2-opt*** ou ***swap***), ***insertion*** et ***inversion***. La mutation échange consiste à sélectionner aléatoirement deux éléments de la permutation et à les permuter. L'opérateur d'insertion choisit un élément au hasard et l'insère dans une autre position de la permutation. L'inversion consiste à inverse l'ordre des éléments entre deux positions choisies aléatoirement. Alba et Dorronsoro ont combiné ces trois opérateurs dans un seul opérateur de mutation dans le cadre de la résolution d'un problème de VRP (Alba et Dorronsoro, 2006).

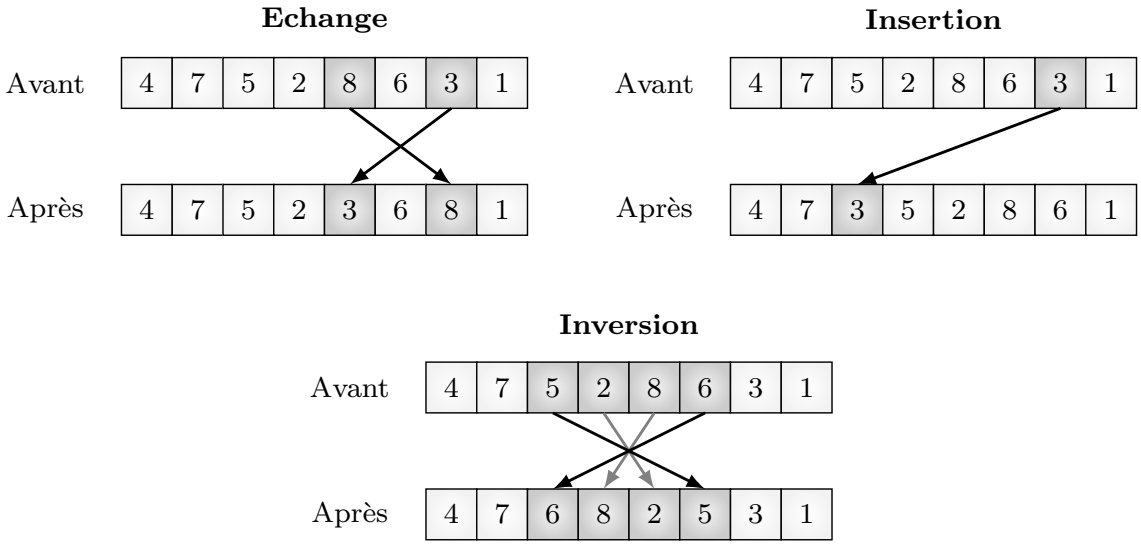


Figure 1.10 – Codage de permutation — Opérateurs d'échange, insertion et inversion.

1.4.1.6 Remplacement

La phase de reproduction (par l'application des opérateurs de croisement et de mutation) crée une grande population composée des anciens individus (les parents) et des nouveaux individus (les enfants). La phase de remplacement consiste alors à

sélectionner les survivants de la génération suivante (i.e., la nouvelle population). On distingue deux types d'algorithmes génétiques :

1. Algorithme génétique ***générationnel*** (*Generational Genetic Algorithm* : gGA) : les enfants remplaçant purement et simplement leurs parents pour créer la population suivante. Les enfants sont placés dans une population axillaire qui remplacera la population courante après avoir complété la reproduction, i.e., quand le nombre de nouveaux individus est égal à la taille de la population.

2. Algorithme génétique ***quasi-stationnaire*** ou ***élitiste*** (*Steady-State Genetic Algorithm* : ssGA) : une seule partie de la population est renouvelée à chaque génération. Un cas particulier est le cas où un seul individu est renouvelé à chaque génération. Dans ce cas, à chaque génération, deux parents sont choisis en utilisant un opérateur de sélection. Ensuite, ces deux individus subissent des opérateurs de croissement et de mutation. Puis, un des deux enfants est inséré dans la population ; typiquement, le nouvel individu remplace le mauvais individu de la population (s'il est de meilleure qualité).

La plupart des algorithmes génétiques récents sont quasi-stationnaires.

1.4.1.7 Paramètres de contrôle

Un GA nécessite l'ajustement de certains paramètres de contrôle (voir l'algorithme 1.7) :

- la ***taille de la population*** (N) : définit le nombre d'individus de la population initiale.

- le ***critère d'arrêt*** : un GA est un algorithme itératif, il faut donc définir un critère d'arrêt tel que par exemple un nombre maximal de générations, des valeurs seuils, l'absence d'amélioration et d'évolution des meilleurs individus ou l'absence de diversité dans la population.

- le ***taux de croisement*** (Pc) : intervient au niveau de la population, il représente le pourcentage de chance de croisement de deux parents.

- le ***taux de mutation*** (Pm) : intervient au niveau de la population, il représente le pourcentage de chance de mutation d'un enfant.
- le ***taux de mutation des gènes*** (Pm_g) : intervient au niveau de la représentation chromosomique, il représente le pourcentage de chance de changement d'un gène dans le chromosome d'un individu.
- ***Clonage*** : cette option permet d'accepter ou non la présence de plusieurs individus représentant la même solution dans la population.

1.4.2 Optimisation par essaim de particules (PSO)

1.4.2.1 Origines et principes

L'optimisation par essaim de particules (*Particle Swarm Optimization* : PSO) est une technique d'intelligence en essaim, originalement introduite par Kennedy et Eberhart en 1995 (Eberhart et Kennedy, 1995, Kennedy et Eberhart, 1995). Cette métaheuristique s'inspire des phénomènes de rassemblement et de nuée observées chez les animaux qui se déplacent et se groupent en essaim, tels que les oiseaux migrateurs et les poissons. De tels animaux adoptent ces comportements intelligents pour surmonter leurs capacités individuelles très limitées et pour avoir une capacité collective leur permettant de faire face à leurs prédateurs ou rechercher la nourriture.

L'algorithme PSO correspond à une stratégie d'optimisation générique et relativement simple à implémenter. Il a été proposé premièrement pour la résolution des problèmes d'optimisation en variables continues. Pour l'optimisation en variables discrètes, une première version binaire de l'algorithme a été proposée dans (Kennedy et Eberhart, 1997). Par ailleurs, une adaptation de l'algorithme pour l'optimisation multiobjectif a été aussi proposée (Hu et Eberhart, 2002). En fait, depuis l'apparition de cet algorithme, plusieurs travaux ont été effectués afin d'améliorer sa performance (Poli et al., 2007). Il a été appliqué avec succès pour la résolution d'une large variété de problèmes d'optimisation (Davoud et Ellips, 2009).

Dans cet algorithme à base de population de solutions, la population est appelée *essaim* et les individus, qui représentent chacun une solution potentielle au problème

traité, sont appelés *particules*. Le processus de recherche d'une solution de bonne qualité consiste à mettre en jeu des groupes de particules sous forme de vecteurs qui se déplacent dans l'espace des solutions. Chaque particule est caractérisée par un vecteur dit de *position* (i.e., la représentation effective de la solution) et un vecteur de changement de position appelé *vélocité*. Ces deux vecteurs déterminent la trajectoire de la particule dans l'espace de recherche. L'optimisation par essaim particulière repose sur la règle suivante :

- Chaque particule se souvient de la meilleure position par laquelle elle est déjà passée au cours de ses déplacements dans l'espace de recherche et tend à y retourner.
- Chaque particule est avertie de la meilleure position découverte par son voisinage (particules voisines) et tend à s'y rendre.

1.4.2.2 Modèle de base et formules

Une topologie de voisinage $\mathcal{N}$ est définie sur l'essaim. Les particules à l'intérieur d'un voisinage communiquent entre-elles. Un voisinage est considéré en fonction des identificateurs des particules et non des informations topologiques comme les distances euclidiennes entre des points dans l'espace de recherche.

Soit M la taille de l'essaim, et soit n la dimension de l'espace de recherche (i.e., nombre de variables de décision). Chaque particule $i = 1, 2, \ldots, M$ a deux variables d'état :

- la position courante : $X_i = (X_{i,1}, X_{i,2}, \ldots, X_{i,n})$;
- la vitesse courante : $V_i = (V_{i,1}, V_{i,2}, \ldots, V_{i,n})$.

et une variable auxiliaire lui permettant de mémoriser la meilleure position par laquelle elle est déjà passée, notée $P_i = (P_{i,1}, P_{i,2}, \ldots, P_{i,n})$, dont la valeur de fitness $(f(P_i))$ est notée *pbest*. La particule est aussi informée de la meilleure position connue au sein de son voisinage, notée $P_g = (P_{g,1}, P_{g,2}, \ldots, P_{g,n})$, dont la valeur de fitness $(f(P_g))$ est notée *gbest*. L'adaptation des particules (i.e., la qualité de ses positions de recherche) est donnée par la fonction objectif du problème traité.

La première étape de l'algorithme de PSO consiste à initialiser l'essaim. La position et la vitesse de chaque particule de l'essaim initial sont générées aléatoirement et/ou de façon heuristique dans les intervalles de variation permises. Pour chaque particule $i = 1, 2, \ldots, M$, la position initiale est placée à P_i. La meilleure position P_i de toutes les particules est ensuite placée à P_g. Après l'initialisation, l'algorithme accomplit une procédure d'évolution itérative. A chaque nouvelle itération $(t + 1)$ de l'algorithme, on modifie l'état (position et vitesse de recherche) de chaque particule dans l'essaim. Cette modification est réalisée en fonction de l'état courant ($X_i(t)$ et $V_i(t)$) de la particule, de sa meilleure position courante ($P_i(t)$) et de la meilleure position globale courante ($P_g(t)$), suivant les formules -de base- suivantes, pour chaque dimension de l'espace de recherche $j = 1, 2, \ldots, n$:

$$V_{i,j}(t+1) = V_{i,j}(t) + r_1 c_1 (P_{i,j}(t) - X_{i,j}(t)) + r_2 c_2 (P_{g,j}(t) - X_{i,j}(t)) \tag{1.3}$$

$$X_{i,j}(t+1) = X_{i,j}(t) + V_{i,j}(t+1) \tag{1.4}$$

où c_1 et c_2 sont deux paramètres déterminant l'influence de P_i et P_g dans la formule de mise à jour de vitesse, et r_1 et r_2 sont des nombres aléatoires choisis uniformément dans l'intervalle $[0, 1]$. L'algorithme PSO de base utilise un autre paramètre : $V_{max} > 0$, pour limiter chaque coordonnée de V_i à l'intervalle $[-V_{max}, V_{max}]$. Cette condition est ajoutée pour assurer que les particules ne se déplacent pas en dehors de l'espace de recherche.

Une illustration graphique de la modification de la position de recherche d'une particule est donnée dans la figure 1.11.

Après le déplacement des particules à leurs nouvelles positions, leurs nouvelles valeurs de fitness sont évaluées. Par la suite, la variable P_i pour chaque particule et la variable globale P_g sont mises à jour en fonction de la qualité des nouvelles positions des particules. Si la nouvelle position d'une particule est meilleure que son P_i courant, P_i prend cette nouvelle position et *pbest* prend la valeur de fitness correspondante. De même, si le meilleur P_i de toutes les particules est de meilleure qualité que P_g courant, P_g est remplacée par ce P_i et *gbest* prend la valeur de fitness correspondante. Le processus de déplacement et évaluation se répète jusqu'à satisfaction d'un certain

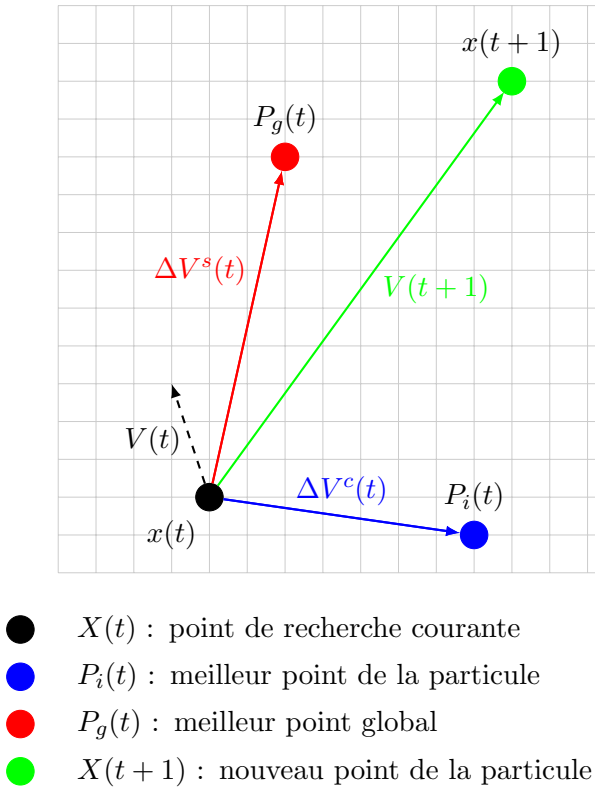


Figure 1.11 – Modification de la position de recherche d'une particule.

critère d'arrêt. La figure 1.12 illustre l'organigramme de fonctionnement global de l'algorithme de PSO.

C'est le vecteur de vitesse qui dirige le processus de recherche vers des régions prometteuses dans l'espace des solutions, et reflété la "sociabilité" des particules. La formule de changement de la vitesse comporte trois composants : le premier, dit d'*inertie*, correspond à la valeur courante de vitesse, le deuxième, dit de l'*influence cognitive*, donne la contribution des expériences réalisées par la particule individuellement et, le troisième, dit de l'*influence sociale*, donne la contribution des expériences réalisées collectivement par l'essaim. Ces trois composants sont illustrés dans la figure 1.11 par les vecteurs $V(t)$, $\Delta V^c(t)$ et $\Delta V^s(t)$ respectivement.

1.4.2.3 Variantes de l'algorithme PSO

L'algorithme PSO présente l'avantage d'être simple à implémenter, raison pour laquelle il est rapidement devenu très populaire et largement appliqué pour la résolu-

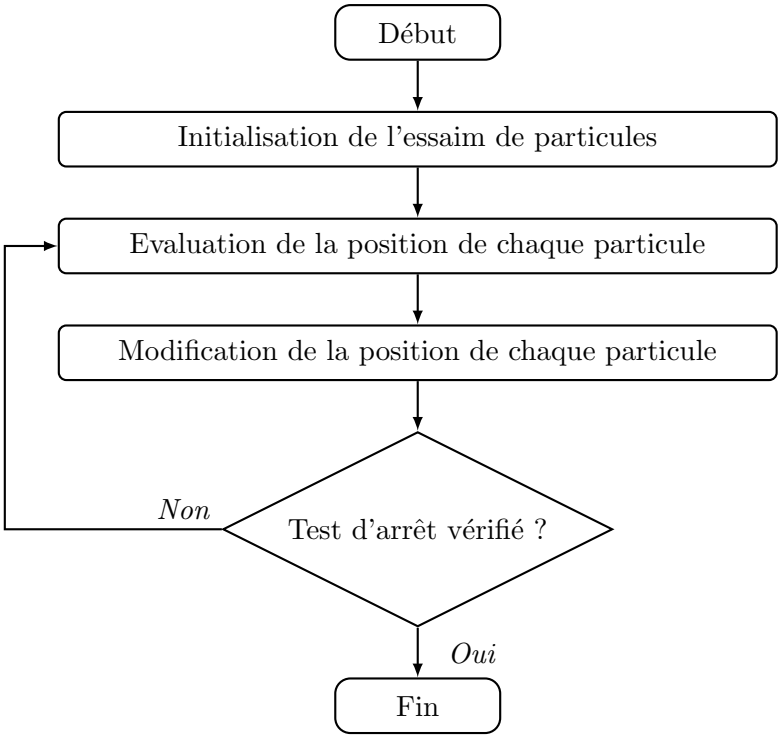


Figure 1.12 – Organigramme global de l'algorithme PSO.

tion d'une grande variété de problèmes pratiques. Un autre avantage principal est qu'il nécessite peu de paramètres de réglage. Malgré ses avantages, il souffre du phénomène de la convergence prématurée vers des solutions sous-optimales. Plusieurs études ont été menées afin de remédier ce problème et améliorer la performance de l'algorithme de base (Poli et al., 2007). Par la suite, nous présentons brièvement les deux premiers changements proposés à l'algorithme PSO de base.

Shi et Eberhart ont proposé dans (Shi et Eberhart, 1998) le modèle du *poids d'inertie*. Cette modification consiste, pour chaque particule $i = 1, 2, \ldots, M$ et chaque dimension $j = 1, 2, \ldots, n$, à multiplier la vitesse du pas de temps t avec un facteur (poids d'inertie) w :

$$V_{i,j}(t+1) = w \times V_{i,j}(t) + r_1 c_1 (P_{i,j}(t) - X_{i,j}(t)) + r_2 c_2 (P_{g,j}(t) - X_{i,j}(t)) \qquad (1.5)$$

Le rôle du paramètre w est de balancer entre exploration et exploitation. Il est initialisé à une grande valeur (proche de 1) afin de favoriser l'exploration globale

de l'espace de recherche. Une diminution graduelle (dans l'intervalle [0.2, 0.5]) de w permet d'obtenir des solutions plus fines (Eberhart et Shi, 2000, Shi et Eberhart, 1998).

Clerc a présenté dans (Clerc, 1999, Clerc et Kennedy, 2002) le modèle de *facteur de constriction* afin de réaliser un contrôle entre la diversification et l'intensification. Cette extension consiste, pour chaque particule $i = 1, 2, \ldots, M$ et chaque dimension $j = 1, 2, \ldots, n$, à multiplier la vitesse par un facteur de constriction χ avant de mettre à jour la position de la particule :

$$V_{i,j}(t+1) = \chi \left(V_{i,j}(t) + r_1 c_1 (P_{i,j}(t) - X_{i,j}(t)) + r_2 c_2 (P_{g,j}(t) - X_{i,j}(t)) \right) \tag{1.6}$$

Des petites valeurs de χ accélèrent la convergence et donnent peu d'exploration, tandis que des valeurs élevées ralentissent la convergence et donnent beaucoup d'exploration.

Une autre variante modifiant de même la formule de changement de la vitesse dans le but de contrôler la convergence a été proposée dans (He et al., 2004).

1.4.2.4 L'algorithme binaire de PSO

Différentes variantes de l'algorithme PSO ont été proposées pour l'optimisation à variables discrètes. La première version binaire a été proposée par Kennedy et Eberhart en 1997 dans (Kennedy et Eberhart, 1997). Dans cette version, notée BPSO (de l'anglais, *Binary Particule Swarm Optimisation*), pour chaque particule $i = 1, 2, \ldots, M$ de l'essaim et chaque dimension de l'espace de recherche $j = 1, 2, \ldots, n$, la valeur de position $X_{i,j}$ prend la valeur 0 ou 1, et la valeur de vélocité $V_{i,j}$ est interprétée comme la probabilité de l'élément $X_{i,j}$ de prendre la valeur 0 ou 1. Il y a plusieurs fonctions qui peuvent vérifier cette condition, c'est-à-dire des fonctions peuvent transformer la valeur de vélocité $V_{i,j}$ en une probabilité dans l'intervalle $[0, 1]$. La fonction Sigmoid est communément utilisée pour faire cette transformation. Elle est définie comme suit :

$$sig(V_{i,j}) = \frac{1}{1 + \exp(-V_{i,j})} \tag{1.7}$$

L'algoritme BPSO utilise cette fonction pour modifier la position de recherche d'une particule $i = 1, 2, \ldots, M$ comme suit :

$$X_{i,j} = \begin{cases} 1 & \text{si } \rho < sig(V_{i,j}) \\ 0 & \text{sinon} \end{cases} \quad j = 1, 2, \ldots, n \tag{1.8}$$

où ρ est un nombre aléatoire choisi uniformément entre 0.0 et 1.0.

La formule de mise à jour de la vitesse (de l'équation 1.3) reste inchangée, sauf que $X_{i,j}$ est maintenant une valeur binaire 0 ou 1 et, par conséquent, $(P_{i,j} - X_{i,j})$ et $(P_{g,j} - X_{i,j})$ prendront -1, 0 ou $+1$.

L'algorithme PSO pour l'optimisation à variables continues utilise le paramètre $V_{max} > 0$ pour limiter $V_{i,j}$ à l'intervalle $[-V_{max}, V_{max}]$. Ce paramètre est également utilisé dans l'algorithme BPSO, cependant, comme nous pouvons le voir, c'est simplement pour limiter la probabilité extrême que $X_{i,j}$ prend la valeur 0 ou 1. Par exemple, pour $V_{max} = 6.0$ (voir figure 1.13), les probabilités seront comprises entre 0.9975 et 0.0025. De ce fait, de nouvelles chaînes binaires seront encore testées, même après que chaque bit a pris sa meilleure valeur. Pour de grandes valeurs de V_{max}, par exemple 10.0, il est peu probable que de nouvelles chaînes binaires apparaissent. Ainsi le rôle de V_{max} dans l'algorithme BPSO est de fixer une limite pour l'exploration supplémentaire après la convergence de l'algorithme ; on pourrait dire, dans un sens, que ce paramètre contrôle le taux de mutation final du vecteur binaire. Il est à noter aussi que, bien qu'une grande valeur de V_{max} pour l'optimisation continue élargit la région explorée par une particule, le cas contraire se produit dans la version BPSO, une petite valeur pour V_{max} permet un taux de mutation plus élevé.

Plusieurs autres versions discrètes de l'algorithme PSO ont été présentées dans la littérature ; voir par exemple (Duran et al., 2008, Xu et al., 2008, Yu et al., 2008, Zhan et Zhang, 2009, Zhong et al., 2007, Zhou et al., 2007).

1.4.2.5 L'algorithme PSO pour les problèmes de permutation

Dans l'algorithme PSO de base, les particules sont représentées comme des points dans l'espace de recherche. Toutes les dimensions de l'espace de recherche sont indé-

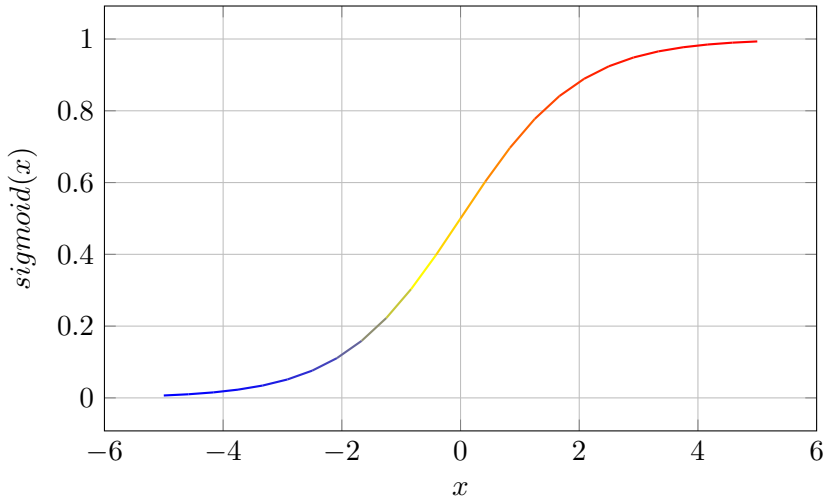


Figure 1.13 – La fonction Sigmoid.

pendantes l'une de l'autre et, par conséquent, les modifications de la position et de la vélocité sont réalisées de façon indépendante pour chaque dimension. Cependant, cette formule n'est pas applicable aux problèmes dont les solutions sont modélisées sous forme de permutations parce que les dimensions ne sont pas indépendantes. Il est possible de produire de cette façon des solutions invalides, par duplication et/ou omission de certains éléments de la permutation. Pour résoudre ces conflits, des stratégies de mise à jour de la particule ont été proposées dans la littérature pour le codage de permutation.

Dans la stratégie proposée dans (Hu et al., 2003), comme dans l'algorithme PSO de base, la vélocité représente la possibilité que la particule se déplace vers une nouvelle position dans l'espace de recherche ; il est très probable que si la vélocité est grande, la particule se change vers une nouvelle permutation. La formule de changement de la vélocité reste inchangée. Cependant, la vélocité est limitée à des valeurs absolues car elle représente seulement la différence entre les particules. La mise à jour d'une particule se réalise de la manière suivante : le vecteur de vélocité est d'abord normalisé en la divisant par le plus grand élément. Les éléments de vélocité sont alors compris entre 0.0 et 1.0 et, par conséquent seront interprétés comme des probabilités. Puis, pour chaque dimension de l'espace de recherche, on détermine de façon aléatoire s'il y a un échange de positions (*swap*) avec une probabilité donnée par la vélocité. Si un

échange est nécessaire, la position prendra la valeur de la même position de la meilleure particule dans l'essaim, P_g, en permutant les valeurs. Ce processus est illustré dans la figure 1.14 (tirée de (Hu et al., 2003)).

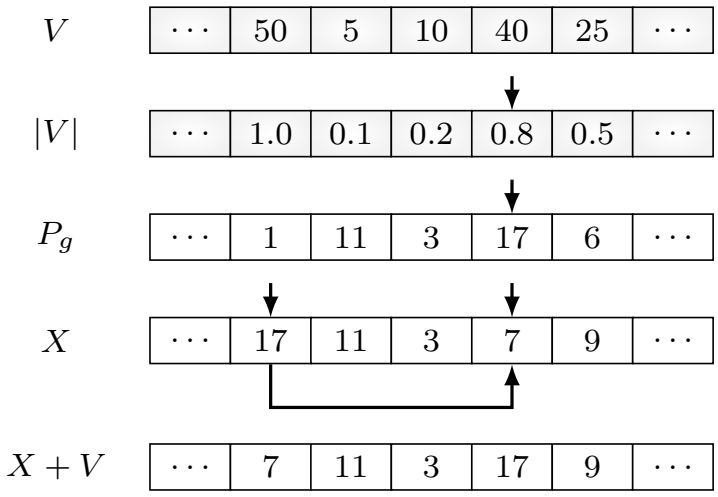


Figure 1.14 – Changement de position pour la représentation de permutation.

Comme nous pouvons le constater, avec cette stratégie de modification, la particule poursuit toujours la meilleure position dans l'essaim, P_g ; elle pourrait rester à sa position courante pour toujours si elle est identique à P_g. Afin de pallier ce problème, les auteurs ont proposé l'échange aléatoire (mutation) d'une paire de positions dans la permutation si la particule est identique à P_g.

Tasgetiren et al. dans (Tasgetiren et al., 2004a, b) ont proposé une règle heuristique appelée *plus petite valeur de position* (de l'anglais, *Smallest Position Value : SPV*) pour transformer la représentation réelle du vecteur de position en une permutation. Cette règle est basée sur la représentation par clés aléatoires (Bean, 1994). Afin de simplifier la description de cette transformation, nous illustrons dans la figure 1.15, pour une particule i, la représentation réelle de position X_i et la permutation correspondante π_i (une illustration tirée de (Tasgetiren et al., 2004b)). Dans cet exemple, selon la règle de SPV, la plus petite valeur de position est $X_{i,5} = -1.20$, alors la dimension $j = 5$ est affectée au premier élément $\pi_{i,1}$ de la permutation ; la deuxième plus petite valeur de position est $X_{i,2} = -0.99$, alors la dimension $j = 2$ est affectée au deuxième élément $\pi_{i,2}$ de la permutation, et ainsi de suite. En d'autres termes, les

dimensions sont triées selon la règle de SPV, i.e., selon les valeurs de position pour construire la permutation.

Dimension, j	1	**2**	3	4	**5**	6
Valeur de position, $X_{i,j}$	1.8	**−0.99**	3.01	−0.72	**−1.20**	2.15
Element de permutation, $\pi_{i,j}$	**5**	**2**	4	1	6	3

Figure 1.15 – Transformation de la représentation réelle en une permutation selon la règle de SPV.

1.4.3 Optimisation par colonies de fourmis (ACO)

L'optimisation par colonies de fourmis (*Ant Colony Optimization* : ACO) est une métaheuristique pour les problèmes d'optimisation à varibles discrètes, qui a été premièrement introduite par Marco Dorigo et ses collègues au début des années 1990 (Dorigo, 1992, Dorigo et al., 1996, 1991). Cette métaheuristique à base de population s'inspire du comportement collectif des fourmis réelles à trouver les chemins les plus courts entre leur nid et des sources de nourriture. Lors de la recherche de leur nourriture, les fourmis explorent d'abord leur environnement au hasard. Quand une fourmi trouve une source de nourriture, elle évalue sa qualité et sa quantité et porte une partie des aliments trouvés au nid. Le long de son chemin de retour au nid, elle dépose sur le sol une substance chimique appelée *phéromone*. La quantité de phéromone déposée, qui peut dépendre de la qualité et de la quantité de la nourriture, attirera les autres fourmis et les guidera vers cette source de nourriture. Cette communication indirecte entre les fourmis via les pistes de phéromone leur permet de trouver le chemin le plus court entre leur nid et la source de nourriture, i.e., le chemin qui présente la plus forte concentration de phéromone. Cette stratégie des colonies de fourmis réelles est exploitée dans les colonies de fourmis artificielles pour résoudre plusieurs problèmes d'optimisation combinatoires difficiles (Dorigo et Blum, 2005).

Les algorithmes de ACO sont des extensions des heuristiques constructives. Une solution valide pour le problème traité est assemblée comme un sous-ensemble de l'en-

semble $\mathcal{C}$ de tous les composants de solution[14]. La construction de solutions se base sur deux types d'informations numériques (López-Ibánez et al., 2015) : l'information heuristique, qui est dérivée de l'instance à traiter[15], et les pistes de phéromone artificielles (i.e., l'équivalent numérique de pistes de phéromone volatiles), qui sont mis à jour en fonction de la performance de la recherche pour guider la construction de solution vers des solutions de bonne qualité. Dans le cas standard, pour chaque composant de solution $c \in \mathcal{C}$, l'algorithme considère une valeur de phéromone $\tau_c \in \mathcal{T}$ et une valeur heuristique $\eta_c \in \mathcal{H}$, où $\mathcal{T}$ (resp. $\mathcal{H}$) est l'ensemble de toutes les valeurs de phéromone (resp. les valeurs heuristiques). La structure $\mathcal{T}$ est communément appelée le modèle de phéromone, qui représente l'élément central de tout algorithme de ACO.

Les étapes principales des algorithmes de ACO sont données dans l'algorithme 1.8 (López-Ibánez et al., 2015). Après l'initialisation des structures de données et des paramètres de contrôle, un algorithme de ACO génèrent des solutions candidates par l'application répétée d'une procédure de construction probabiliste de solution (fonction `ConstruireSolutions()`) suivie d'une procédure de mise à jour des pistes de phéromone (fonction `MettreAJourPheromones()`). De plus, une procédure de recherche locale peut être appliquée pour améliorer une ou plusieurs solutions construites dans l'itération courante.

Algorithme 1.8 : Optimisation par colonie de fourmis

```
1 InitialiserPheromones()
2 tant que La condition d'arrêt n'est pas satisfaite faire
3 |   ConstruireSolutions()
4 |   RechercheLocale() {Optionnel}
5 |   MettreAJourPheromones()
6 fin
```

Construction de solutions A chaque itération de l'algorithme, des fourmis artificielles construisent chacune une solution. La construction d'une solution part d'une

[14]La définition des composants de solution dépend du problème traité. Dans le cas du problème du TSP, par exemple, $\mathcal{C}$ correspond à l'ensemble de tous les arcs du graphe en entrée, i.e., les relations d'adjacence entre les villes.

[15]Pour le problème du TSP, l'information heuristique correspond aux distances entre paires de villes.

solution vide $S^p = \langle\rangle$. Puis, à chaque étape de construction, le composant de solution suivant c' à ajouter à la solution partielle courante $S^p \subseteq \mathcal{C}$ est choisi parmi l'ensemble de composants candidats $\mathcal{N}(S^p) \subseteq \mathcal{C}$ en fonction d'une probabilité $Pr(c'|S^p)$. Cette probabilité dépend de la quantité de phéromone $\tau_{c'}$, et de la valeur heuristique $\eta_{c'}$, associées aux éléments de $\mathcal{N}(S^p)$. Plusieurs algorithmes de ACO utilisent la règle de probabilité qui a été initialement proposée pour *Ant System* (AS) (Dorigo et al., 1996) pour choisir un composant parmi l'ensemble $\mathcal{N}(S^p)$, qui est donnée par l'équation :

$$Pr(c'|S^p) = \frac{{\tau_{c'}}^{\alpha} \cdot \eta_{c'}^{\beta}}{\sum_{c'' \in \mathcal{N}(S^p)} {\tau_{c''}}^{\alpha} \cdot \eta_{c''}^{\beta}} \qquad \forall c' \in \mathcal{N}(S^p). \tag{1.9}$$

où α et β sont des paramètres qui déterminent respectivement les influences des valeurs de phéromone et des valeurs heuristiques sur les probabilités de choix. Si α est proche de zéro, la procédure de construction de solutions correspond à un algorithme glouton à départs multiples ; si β est proche de zéro, la construction de solutions est guidée uniquement par les pistes de phéromone, et l'information heuristique est négligée.

Mise à jour des pistes de phéromone Les valeurs de phéromones sont mise à jour durant l'exécution d'un algorithme de ACO pour orienter la recherche vers de bonne solutions. Cela se fait en deux étapes complémentaires : *évaporation* de phéromone et *dépôt* de phéromone. L'évaporation de phéromone consiste, dans la plupart des algorithmes de ACO, à réduire toutes les valeurs de phéromone par un certain facteur. Elle peut être définie par l'équation suivante (López-Ibánez et al., 2015) :

$$\tau_c = (1 - \rho) \cdot \tau_c \qquad \forall c \in \mathcal{C}. \tag{1.10}$$

où $\rho \in [0, 1]$ est le taux d'évaporation.

Le dépôt de phéromone consiste à augmenter les valeurs de phéromone des composants de solution qui apparaissent dans un ensemble de bonnes solutions construites dans cette itération ou dans les itérations précédentes. La forme générale de cette

opération s'exprime par l'équation suivante (López-Ibánez et al., 2015) :

$$\tau_c = \tau_c + \sum_{s_k \in S^{upd} | c \in s_k} w_k \cdot F(s_k), \tag{1.11}$$

où S^{upd} est l'ensemble de bonne solutions choisies pour déposer le phéromone, w_k représente le poids affecté à la solution $s_k \in S^{upd}$, et $F(s_k)$ est une fonction proportionnelle à la quantité de s_k, c'est-à-dire, si $f(s) < f(s')$ au cas de problème de minimisation, alors cette fonction assure que $F(s) \geq F(s')$. La quantité $w_k \cdot F(s_k)$ correspond alors à la quantité de phéromone qui est déposée par la solution s_k.

Plusieurs améliorations ont été apportées à l'algorithme de ACO initial (*Ant System* (AS)), donnant naissance à différentes variantes : *Elitist Ant System* (EAS) (Dorigo et al., 1996), *Ant Colony System* (ACS) (Dorigo et Gambardella, 1997), *Max-Min Ant System* (MMAS) (Stützle et Hoos, 2000), *Rank-based Ant System* (RAS) (Bullnheimer et al., 1999), *Best-Worst Ant System* (BWAS) (Cordon et al., 2000). Pour plus d'informations, nous référons le lecteur intéressé à (Dorigo et Stützle, 2010, López-Ibánez et al., 2015).

1.5 Métaheuristiques hybrides

Quand les métaheuristiques de base ont atteint leurs limites, depuis deux décennies, la recherche a été orientée vers la combinaison de différents algorithmes d'optimisation. Un nombre assez impressionnant d'algorithmes récents ne suivent pas le paradigme d'une seule métaheuristique traditionnelle. Ils combinent plusieurs composants/concepts algorithmiques, souvent issus d'algorithmes de domaines d'optimisation différents. Ces approches sont communément appelées *métaheuristiques hybrides*. Le but est d'exploiter le caractère complémentaire de différentes stratégies d'optimisation, c'est-à-dire que les approches hybrides sont censées bénéficier d'une synergie (Blum et al., 2011). En général, les métaheuristiques hybrides sont développées pour fournir un bon équilibre entre l'intensification et la diversification de la recherche. En fait, une combinaison adéquate d'éléments/concepts algorithmiques complémentaires peut produire un algorithme plus performant pour la résolution de plusieurs problèmes

d'optimisation difficiles. Néanmoins, le développement des métaheuristiques hybrides efficaces est en général une tâche difficile qui demande une bonne maîtrise de différents domaines d'optimisation. En outre, la littérature a montré qu'un algorithme hybride pourrait s'opérer bien pour des problèmes spécifiques mais moins bien pour d'autres (Blum et al., 2011). Cependant, il existe des types d'hybridation qui se sont révélés efficaces pour plusieurs applications.

Une première taxonomie de l'hybridation des métaheuristiques a été proposée dans (Talbi, 2002). Le premier livre dans la littérature spécifiquement dédié aux métaheuristiques hybrides a été publié en 2008 (Blum et al., 2008). Dans (Blum et al., 2011), Blum et al. ont présenté un état de l'art des métaheuristiques hybrides conçues pour les problèmes d'optimisation combinatoires (mono-objectifs) ; ils ont illustré surtout des exemples marquants. D'autres revues de littérature et méta-analyses peuvent être trouvées dans (Blum et Raidl, 2016, Raidl, 2006, Raidl et al., 2010, Talbi, 2013).

Les métaheuristiques hybrides peuvent se distinguer en deux grands types d'hybridation : hybridation des métaheuristiques avec des métaheuristiques ou des heuristiques et hybridation des métaheuristiques avec des méthodes complètes. Une grande partie de l'état de l'art de ces hybridations présenté par la suite est notamment inspirée de la revue de la littérature (Blum et al., 2011).

1.5.1 Hybridation métaheuristiques/(méta)heuristiques

Algorithmes mémétiques Quand les chercheurs se sont d'abord intéressés à l'hybridation de leurs métaheuristiques préférées avec d'autres techniques d'optimisation, la plupart ont commencé à envisager des combinaisons possibles avec des heuristiques ou d'autres métaheuristiques. En fait, la forme d'hybridation la plus répondue est l'incorporation de techniques de recherche locale dans des méthodes à base de population. En fait, les EAs et les algorithmes de ACO appliquent souvent une procédure de recherche locale pour améliorer les solutions trouvées au cours de la recherche. Nous pouvons citer, par exemple, les algorithmes hybrides présentés dans (Alba et Luque, 2008, Elbenani et al., 2012) qui remplacent l'opérateur de mutation d'un GA par une procédure de recherche locale, et les algorithmes hybrides proposés dans (Azimi, 2005,

Gambardella et al., 2012, McKendall et Shang, 2006) qui incorporent une méthode de recherche locale dans un algorithme de ACO. Ces tendances peuvent être attribuables au fait que les métaheuristiques à base de population permettent une meilleure diversification de la recherche et une meilleure identification des régions de solutions prometteuses. Lors de l'initialisation, ces métaheuristiques essayent généralement de percevoir d'information globale sur l'espace des solutions. Puis, elles concentrent au cours d'itérations successives la recherche dans les régions prometteuses. Cependant, elles ne sont pas généralement efficaces en ce qui concerne l'exploitation de l'expérience de recherche, c'est-à-dire la localisation des meilleures solutions dans les régions de bonne qualité. D'autre part, les méthodes de recherche locale permettent une bonne exploitation du voisinage d'une solution initiale. En résumé, les métaheuristiques à population sont efficaces en ce qui concerne l'identification des régions de solutions prometteuses, dans lesquelles les méthodes de recherche locale peuvent rapidement localiser de bonnes solutions. Ce type d'hybridation est, habituellement, très efficace. Les EAs qui appliquent des méthodes de recherche locale sont connus dans la littérature sous le nom de "algorithmes mémétiques" (Krasnogor et Smith, 2005, Moscato, 1999). La structure d'un algorithme mémétique est schématisée dans la figure 1.16, qui illustre la relation entre la partie évolutionnaire et la partie de recherche locale.

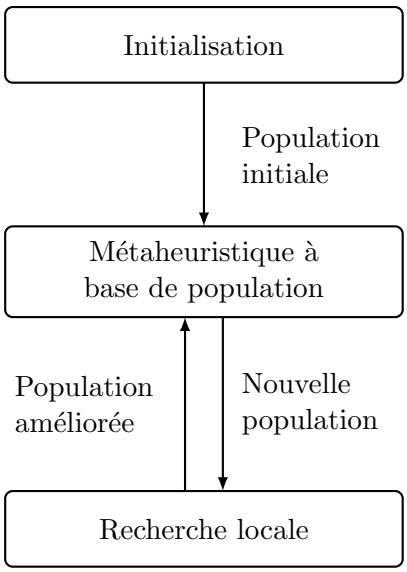


Figure 1.16 – La forme générique d'un algorithme mémétique.

Recherche locale à base de population Contrairement à l'application des méthodes de recherche locale comme des opérateurs dans les méthodes à base de population, ces dernières années ont vu l'apparition des méthodes de recherche locale optimisées par des concepts issus des approches à base de population. Un exemple de ce type d'hybridation est l'application de la recherche locale itérée sur une population de solutions −au lieu d'une seule solution−, présentée dans (Stützle, 2006, Thierens, 2004). Cet algorithme hybride est plus efficace pour la résolution du problème d'affectation quadratique (ou *Quadratic Assignment Problem* : QAP). Un autre exemple similaire a été donné dans (Lozano et García-Martínez, 2010), où les auteurs ont utilisé un algorithme évolutionnaire comme une procédure de perturbation pour la recherche locale itérée. Dans (Resende et Ribeiro, 2010), les authors ont conçu plusieurs algorithmes hybrides combinant la méthode GRASP avec des techniques de *Path-relinking* (techniques de recherche locale) pour le problème de la diversité maximale/minimale (ou *max-min devrsity problem* en anglais). Un exemple parmi ces algorithmes hybrides est un algorithme évolutionnaire de *Path-relinking* qui fait évoluer une population de solutions pour être à la fois diversifiée et de bonne qualité.

Techniques d'optimisation multiniveaux Les techniques d'optimisation multiniveaux (Walshaw, 2004, 2008) sont une autre forme d'hybridation des métaheuristiques avec des concepts heuristiques, qui ont été introduites en particulier pour traiter les problèmes de grande taille. Ces approches heuristiques sont basées sur une idée simple : partitionner progressivement la structure de l'instance originale de problème en sous-instances de plus en plus petites, jusqu'à ce qu'une condition d'arrêt soit vérifiée. Cette décomposition fournit une hiérarchie d'instances de problème. La sous-instance d'un certain niveau est toujours plus petit que (ou de même taille que) la sous-instance du niveau supérieur suivant. Après décomposition, une technique d'optimisation (comme, par exemple, un algorithme métaheuristique) est appliquée pour construire une solution à la plus petite sous-instance. Cette solution est successivement transformée en une solution à la sous-instance du niveau suivant jusqu'à obtenir une solution pour l'instance complète. La solution obtenue à chaque niveau peut être améliorée en utilisant, souvent, un algorithme métaheuristique.

Hyper-heuristiques Les hyper-heuristiques (Burke et al., 2003, 2010) sont des approches de haut niveau qui ont pour objectif commun l'automatisation de la conception et de l'adaptation des méthodes heuristiques pour la résolution de problèmes d'optimisation difficiles. La motivation derrière ces approches est d'augmenter le niveau de généralité auquel les méthodologies d'optimisation peuvent fonctionner. Dans un contexte d'adaptation, une hyper-heuristique est une approche de haut niveau qui, pour une instance particulière d'un problème et à partir d'un ensemble d'heuristiques spécifiques de bas niveau, peut choisir et appliquer les heuristiques appropriés à chaque point de décision (Burke et al., 2003, Pisinger et Ropke, 2007). Une autre tendance de recherche consiste à générer automatiquement des heuristiques adéquates pour un problème spécifique ou une classe de problèmes. Ceci est typiquement fait en combinant, grâce à l'utilisation de la programmation génétique, par exemple, des composants d'heuristiques (Burke et al., 2007).

Optimisation des métaheuristiques par des techniques additionnelles Une autre approche d'hybridation consiste à optimiser les métaheuristiques par des techniques additionnelles pour améliorer le temps de calcul et/ou les résultats. Dans (Montemanni et Smith, 2010), les auteurs ont proposé une approche basée sur la recherche tabou pour la résolution du problème d'affectation de fréquences (ou *frequency assignment problem* en anglais). Dans cette approche, la recherche tabou est combinée avec un principe heuristique fondé sur l'addition des contraintes au problème traité afin de réduire l'espace de recherche. Un autre exemple est la proposition donnée dans (Chaves et al., 2007), qui consiste à détecter des régions de recherche prometteuses en se fondant sur la recherche par clustering. Cette approche consiste à répéter de manière itérative les trois étapes suivantes : d'abord un algorithme métaheuristique est utilisé pour générer de nouvelles solutions. Ces dernières sont ensuite regroupées en clusters. Enfin, une recherche locale est appliquée pour améliorer les solutions dans le cluster jugé prometteur.

Une autre approche heuristique pour améliorer la performance des métaheuristiques est le principe d'optimalité proche (de l'anglais *proximate optimality principale*) qui a été introduit par Glover et Laguna dans le cadre de la recherche tabou

(Glover et Laguna, 1993, Glover et Taillard, 1993). Cette approche se fonde sur l'intuition que de bonnes solutions sont susceptibles d'avoir des éléments en commun et peuvent donc être trouvées proches les unes des autres dans l'espace de recherche. Dans (Fleurent et Glover, 1999), ce principe a été utilisé dans le cadre des solutions partielles générées par la méthode GRASP. L'idée était que les mauvaises décisions prises durant le processus de construction peuvent être éliminées par une procédure de recherche locale durant (et non pas seulement à la fin de) la phase de construction.

EAs structurés et EAs quantiques Pour améliorer les capacités d'exploration des EAs, les EAs structurés ont été développés. Ce type d'algorithme introduit une structure de voisinage sur la population de solutions. Parmi les EAs structurés, les EAs *distribués* et les EAs *cellulaires* sont les plus connus. Dans le cas des EAs distibués (Alba et Tomassini, 2002), la population est divisée en sous-populations dans chacune un EA isolé est exécuté. Des individus migrent entre les sous-populations pour offrir une certaine diversité dans ces dernières et ainsi éviter les optima locaux. Les EAs cellulaires (Alba et Dorronsoro, 2009a) utilisent le concept de voisinage de la même manière que les automates cellulaires, de sorte que les individus ne peuvent interagir qu'avec leurs voisins les plus proches dans la population.

Une limitation majeure des EAs réside dans le fait que leurs espaces de recherche ne représentent pas tous les solutions possibles. Cela a conduit à la conception des AEs exploitant des concepts *quantiques* (en particulier pour les problèmes d'optimisation en variables binaires). Les EAs quantiques exploitent le principe de l'informatique quantique de la superposition d'états : les bits usuels sont remplacés par des bits quantiques. Un bit quantique peut avoir les deux valeurs binaires 0 et 1 en même temps. Il est représenté par deux valeurs réelles α et β chacune variant dans l'intervalle $[0, 1]$; il vaut 0 avec une probabilité α^2 et 1 avec β^2 (ce qui implique que $\alpha^2 + \beta^2 = 1$). Avec le codage quantique, qui offre plus de diversité, les EAs peuvent travailler sur un nombre réduit des individus sans perdre la performance. Pour de plus amples détails sur les EAs quantiques, nous renvoyons le lecteur à (Han et Kim, 2002, Zhang, 2011). D'ailleurs, la métaheuristique PSO a été aussi optimisée par des concepts quantiques ; voir par exemple (Sun et al., 2004, Yang et al., 2004).

Hybridations inusuelles Pour conclure cette section, il convient de mentionner certaines hybridations inusuelles entre différents algorithmes métaheuristiques.

Il faut tout d'abord mentionner le travail récent de Villagra et al. qui ont proposé une méthodologie générale d'hybridation des métaheuristiques avec les composants actifs (ou essentiels) d'autres métaheuristiques. Dans (Villagra et al., 2015), ces auteurs ont optimisé les algorithmes génétiques cellulaires par les composants actifs du recuit simulé et de l'algorithme de PSO. Dans (Villagra et al., 2016), ils ont optimisé les mêmes algorithmes par les composants actifs de la recherche dispersée.

Un exemple de l'hybridation des algorithmes métaheuristiques est l'algorithme décrit dans (Lin et al., 2009) qui hybride le recuit simulé avec la recherche tabou pour résoudre le problème de VRP. De manière similaire, dans (Minetti et al., 2014) les auteurs ont combiné le recuit simulé avec un algorithme de recherche locale pour résoudre le problème d'assemblage de fragments d'ADN. Pour la résolution de ce même problème, un algorithme de PSO en combinaison avec la recherche tabou, le recuit simulé et la recherche à voisinage variable a été proposé dans (Huang et al., 2015). Un autre exemple est le travail proposé dans (Nemati et al., 2009), où les auteurs ont proposé une hybridation co-évolutionnaire entre un GA et un algorithme de ACO pour le problème de sélection de caractéristiques à partir d'une base de données de protéines. On peut citer également l'algorithme hybride proposé dans (Shi et al., 2005) qui combine un GA avec algorithme de PSO. Un dernier exemple est l'algorithme proposé dans (Greistorfer, 2003) qui optimise la recherche tabou par les principes de la recherche dispersée. Cependant, il convient de noter que ces algorithmes hybrides sont juste quelques exemples représentatifs de nombreuses différentes formes de combinaison de différentes métaheuristiques.

1.5.2 Hybridation métaheuristiques/méthodes complètes

L'hybridation englobe aussi la combinaison des méthodes exactes avec les algorithmes métaheuristiques. Au cours de ces dernières années, de nombreux travaux ont montré l'utilité de cette combinaison. Comme les méthodes exactes ne sont pas capables de résoudre les problèmes de grandes tailles, elles ont été combinées avec des

métaheuristiques pour résoudre des sous-problèmes de petites tailles ou pour exploiter les expériences réalisées par les algorithmes métaheuristiques comme une information heuristique pour faire une recherche non exhaustive.

Les algorithmes métaheuristiques ont été hybridés notamment avec quatre catégories importantes de méthodes exactes (Blum et al., 2011) : (i) la programmation par contraintes, (ii) les méthodes de recherche arborescente, (iii) les techniques de relaxation, et (iv) la programmation dynamique. Nous décrivons brièvement par la suite chaque catégorie et mentionnons quelques exemples. Nous reportons le lecteur à (Blum et al., 2011) pour une revue de la littérature sur ces catégories d'hybridation.

1.5.2.1 Hybridation des métaheuristiques avec la programmation par contraintes

En programmation par contraintes (ou CP pour *Constraint Programming* en anglais), chaque contrainte du problème traité est utilisée par un algorithme de filtrage (ou propagation) pour supprimer les valeurs de variables qui ne peuvent pas prendre part à des solutions réalisables. Le processus de recherche de solution est caractérisé par la succession d'une phase de propagation de contraintes, pour réduire l'espace des solutions, et d'une phase de recherche systématique durant laquelle des affectations possibles de chacune des variables sont effectuées. Lorsque le filtrage détecte qu'une affectation partielle de variables viole une contrainte, un mécanisme de retour sur trace (ou *backtracking* en anglais) est utilisé afin de remettre en cause la dernière affectation élémentaire (d'une seule variable) effectuée. Dans le cas de résolution d'un problème d'optimisation, une contrainte est ajoutée à la fonction objectif chaque fois qu'une nouvelle solution améliorée est trouvée. Les métaheuristiques explorent usuellement un espace de recherche dans laquelle les configurations sont définies par des affectations complètes de variables, et sont généralement guidées par une information locale sur la fonction objectif. Elles ne sont pas usuellement assez puissantes pour traiter les problèmes de satisfaction de contraintes. En revanche, l'importance des techniques de CP réside dans leurs capacités d'explorer un espace d'affectations partielles et de trouver une solution respectant les contraintes du problème, mais elles ne sont pas capables de résoudre les problèmes d'optimisation de grande taille. Ces deux méthodes

très différentes sont complémentaires et il est donc tout à fait naturel d'essayer de les combiner afin d'exploiter des synergies possibles (Blum et al., 2011).

Un premier exemple de ce type d'hybridation concerne la recherche à voisinage large basée sur CP (*CP-based large neighborhood search* : LNS) qui utilise une technique de CP pour explorer un voisinage typiquement très large. En d'autres termes, une technique de CP est appliquée pour trouver une affectation optimale d'un sous-ensemble de variables. Cette hybridation combine l'avantage de LNS, qui améliore souvent la capacité d'exploration de la recherche locale, avec une exploration exhaustive par CP qui est plus rapide qu'une énumération. LNS a été proposée pour la première fois dans (Shaw, 1998).

Un deuxième exemple est la combinaison de ACO avec CP (Solnon, 2013), qui sont des techniques constructives complémentaires : ACO est caractérisée par sa capacité d'apprentissage par renforcement, tandis que CP est efficace pour traiter les contraintes. Dans (Meyer, 2008) une combinaison ACO-CP a été proposé pour un problème d'ordonnancement. Dans cette combinaison, CP est utilisée par les fourmis artificielles durant la phase de construction de solutions pour viser des solutions réalisables. En d'autres termes, CP joue le rôle de filtrage, tandis que ACO par son mécanisme de construction probabiliste joue le rôle de sélection de valeurs/variables. Une autre algorithme ACO-CP a été présenté dans (Khichane et al., 2010). Il comporte en deux phases. Dans la première phase, une recherche par ACO typique est effectuée et la matrice de phéromones est sauvegardée. Dans la seconde phase, un algorithme de CP effectue une recherche complète en exploitant la structure de phéromones comme une information heuristique pour l'affectation de valeurs aux variables.

1.5.2.2 Hybridation des métaheuristiques avec les techniques de recherche arborescente

L'hybridation des métaheuristiques avec les techniques de recherche arborescente est parmi les formes les plus populaires de combinaison de différents algorithmes d'optimisation. C'est parce que plusieurs métaheuristiques ainsi que certains des algorithmes complets les plus importants effectuent des déplacements (dans l'espace des solutions) de type recherche arborescente. Ces techniques considèrent l'espace de re-

cherche d'un problème d'optimisation sous forme d'un arbre. Un tel arbre est défini (parfois seulement implicitement) par un mécanisme d'extension de solutions partielles. Chaque chemin allant de la racine jusqu'à une des feuilles correspond à la construction d'une solution candidate. Les nœuds internes de l'arbre correspondent à des solutions partielles. Le déplacement d'un nœud interne à un de ses descendants est une extension de la solution partielle correspondante.

Comme déjà mentionné, la classe de techniques de recherche arborescente englobe des métaheuristiques et des algorithmes complets. La métaheuristique ACO (voir section 1.4.3) et la méthode GRASP (voir section 1.3.5) sont des exemples spécifiques des métaheuristiques qui appliquent une recherche arborescente. Ces algorithmes constructifs génèrent des solutions candidates par l'application répétée d'une procédure de construction probabiliste. Un exemple important d'algorithmes complets de la classe de techniques de recherche arborescente est la méthode de *Branch & Bound*, qui peut être également appliquée dans plusieurs variantes heuristiques comme, par exemple, sous forme de recherche par faisceau (de l'anglais *beam search*). Alors que la méthode de *Branch & Bound* considère (implicitement) tous les nœuds d'un certain niveau de l'arbre de recherche, la recherche par faisceau explore un nombre limité de nœuds.

Les métaheuristiques constructives, comme les algorithmes de ACO et la méthode GRASP, présentent deux inconvénients. D'une part, même elles prennent des décisions probabilistes pour construire des solutions, elles n'évitent pas des actions gloutonnes, ce qui augmente le risque de converger vers des résultats moins précis. D'autre part, en comparaison avec les algorithmes complets, ces deux métaheuristiques n'appliquent pas des mécanismes pour réduire l'espace de recherche. Afin de remédier à ces inconvénients, plusieurs travaux récents ont considéré l'incorporation de propriétés de la méthode *Brand & Bound* dans des métaheuristiques constructives. Un exemple représentatif est les algorithmes proposés dans (Blum, 2010, 2005, 2008, Blum et al., 2013, López-Ibáñez et Blum, 2010) qui utilisent des concepts de la recherche par faisceau pour optimiser la construction de solutions dans un algorithme de ACO. L'idée principale de cet algorithme hybride, nommé Beam-ACO, est que, à chaque itération, plusieurs solutions sont construites en parallèle de manière non indépendante, comme c'est le cas dans la recherche par faisceau (En revanche, à chaque itération

d'un algorithme de ACO, des fourmis artificielles construisent des solutions de manière indépendante les unes des autres). Cependant, contrairement à la recherche par faisceau qui étend les solutions partielles de manière déterministe en utilisant des informations heuristiques, dans Beam-ACO le choix d'extensions réalisables est effectué de manière probabiliste de la même manière qu'un algorithme de ACO. En plus des traces de phéromones et des informations heuristiques qui sont utilisées pour guider la recherche, Beam-ACO calcule aussi des bornes pour évaluer les solutions partielles et, ainsi, pour couper des branches de l'arbre de recherche.

Les solveurs MIP (pour *Mixed Integer Programming*) sont en général basés sur des techniques de recherche arborescente et appliquent la programmation linéaire sur des relaxations continues du problème traité afin d'obtenir des bornes inférieurs et supérieurs de la fonction objectif. De tels solveurs peuvent être utiles pour explorer des voisinages très larges dans le cadre de résolution de problèmes d'optimisation difficiles par algorithmes métaheuristiques, étant donné que le problème en question peut être exprimé par un modèle MIP. Un exemple de telles applications est le voisinage large décrit dans (Prandtstetter et Raidl, 2008) dans le cadre de la résolution du problème de séquençage de voitures (ou, en anglais, *car sequencing problem*). L'objectif est de déterminer une bonne séquence (i.e., une permutation) d'un ensemble de voitures à assembler, c'est-à-dire une séquence permettant d'égaliser la charge de travail des postes de la ligne d'assemblage. Dans cette application, un solveur MIP est appliqué pour repositionner à l'optimalité un certain nombre de voitures.

1.5.2.3 Hybridation des métaheuristiques avec les tehniques de relaxation

L'amélioration des performances des algorithmes métaheuristiques par des informations provenant de relaxations du problème initial représente un type d'hybridation très populaire au cours des dernières années. Une version relaxée d'un problème donné est obtenue en simplifiant et/ou en supprimant des contraintes. Lorsqu'on supprime des contraintes, elles peuvent soit être abandonnées, soit être intégrées dans la fonction objectif. Dans le cas où le problème relaxé peut être résolu efficacement, l'espoir est que la structure de la solution du problème relaxé en combinaison avec la valeur de la fonction objectif peuvent faciliter d'une façon ou d'une autre la résolution du

problème initial. La relaxation est également fortement utilisée, par exemple, dans les techniques exactes comme la méthode de *Branch & Bound*. C'est parce que la valeur de fitness de la solution optimale du problème relaxé peut être considérée comme une borne pour la valeur de fitness de la solution optimale du problème initial et, par conséquent, peut être utilisée pour élaguer l'arbre de recherche. Un type très important de relaxation dans l'optimisation combinatoire concerne la suppression des contraintes d'intégralité sur les variables entières d'un modèle MIP. La version relaxée obtenue, qui est un programme linéaire (ou *linear program*), peut être résolue à l'optimalité par une méthode efficace comme l'algorithme du simplexe.

Un premier exemple de ce type d'hybridation est une approche métaheuristique basée sur la relaxation lagrangienne (Boschetti et Maniezzo, 2009, Boschetti et al., 2010). Une relaxation lagrangienne est obtenue en intégrant des contraintes dans la fonction objectif. A chaque itération du processus d'optimisation, la solution optimale de la relaxation courante est utilisée pour la génération des solutions réalisables au problème initial. Ces solutions sont utilisées pour définir une nouvelle relaxation améliorée. Cela peut se faire au moyen des heuristiques simples ou au moyen des concepts métaheuristiques. Le processus de recherche peut s'arrêter lorsque les bornes inférieures et supérieures de la fonction objectif coïncident.

Dans (Wilbaut et Hanafi, 2009), plusieurs heuristiques basées sur des relaxations itératives ont été présentées pour la résolution de problèmes de programmation mixte en nombres binaires (ou, en anglais, 0-1 *mixed integer programming problems*). Les auteurs ont combiné deux techniques de relaxation dans un ensemble d'algorithmes heuristiques : une relaxation linéaire (ou continue) qui consiste à supprimer les contraintes d'intégralité sur les variables entières, et une relaxation dite MIP qui impose des contraintes d'intégralité sur un sous-ensemble de variables binaires. Les solutions obtenues par les deux relaxations sont exploitées itérativement pour réduire l'espace de recherche du problème initial.

1.5.2.4 Hybridation des métaheuristiques avec la programmation dynamique

La programmation dynamique (ou DP pour *Dynamic Programming* en anglais) (Bertsekas, 2017) est une méthode algorithmique d'optimisation qui résout un problème combinatoire comme suit. Tout d'abord, le problème à traiter est divisé en sous-problèmes. Ensuite, la (ou une) solution du problème est obtenue en combinant les solutions de sous-problèmes résolus précédemment en des solutions à des problèmes plus grands jusqu'à ce que le problème initial soit résolu. Un point clé de DP réside dans le fait qu'elle résout chaque sous-problème une seule fois et mémorise le résultat, évitant ainsi le recalcul (lorsque les sous-problèmes comportent des parties communes). En fait, un problème d'optimisation doit présenter deux propriétés pour être résolu par DP (Blum et al., 2011) :

1. Les solutions optimales du problème doivent contenir des solutions optimales de sous-problèmes. Cette caractéristique garantit qu'il existe une formule permettant de déduire la (ou une) solution optimale du problème en combinant les solutions optimales d'une série de sous-problèmes.

2. L'espace de sous-problèmes doit être relativement petit. Typiquement, le nombre total de sous-problèmes distincts est polynomial en la taille de l'entrée.

Dans (Congram et al., 2002), DP est utilisée comme une stratégie d'exploration de voisinage dans une procédure de recherche locale itérée. Dans cette approche proposée pour la résolution d'un problème d'ordonnancement, un algorithme d'énumération récursif basé sur DP explore un voisinage de taille exponentielle en un temps polynomial.

Une métaheuristique hybride inspirée de DP, appelée en anglais "*Corridor Mehtod*", a été présentée dans (Sniedovich et Viß, 2006). Cette méthode a été conçue dans le but d'appliquer DP d'une manière heuristique pour la résolution de problèmes de grande taille. Elle applique itérativement DP sur des sous-espaces de recherche construits à partir de la solution courante.

Une autre application a été présentée dans (Juang et Su, 2008) pour l'alignement multiple de séquences en bio-informatique. Le processus d'alignement multiple de

séquences par DP fonctionne comme suit. Tout d'abord, deux séquences sont alignées à l'optimalité. Puis, la séquence obtenue est alignée avec une troisième séquence. Ce processus est répété jusqu'à ce que toutes les séquences soient considérées. A la fin de chaque étape, un algorithme de PSO est appliqué pour optimiser le résultat.

1.6 Conclusion

Ce chapitre dresse un état de l'art de l'optimisation combinatoire par algorithmes métaheuristiques. Nous avons rappelé dans un premier temps quelques notions de base de l'optimisation combinatoire et de la théorie de complexité. Puis, dans un deuxième temps, nous avons décrit les principales métaheuristiques. On distingue les métaheuristiques à base de solution unique et les métaheuristiques à base de population de solutions. Les métaheuristiques à base de solution unique se basent sur la notion de voisinage et sur la recherche locale pour trouver des solutions sous-optimales du problème traité. Elles permettent en général l'exploitation des régions de l'espace de recherche. Par contre, les métaheuristiques à base de population se fondent sur une recherche globale et, par conséquent, permettent une meilleure exploration de l'espace de recherche. Un intérêt particulier a été porté aux GAs et à PSO, puisque nous avons conçu, durant cette thèse, un algorithme hybride combinant les composants de ces métaheuristiques pour la résolution efficace de problèmes combinatoires multiobjectifs, que nous allons présenter dans le chapitre 3.

A la fin du chapitre, nous avons décrit un état de l'art des métaheuristiques hybrides. L'hybridation consiste à combiner des métaheuristiques avec d'autres techniques d'optimisation pour tirer profit des avantages cumulés de différentes méthodes. Durant ces dernières années, les métaheuristiques hybrides sont devenues très populaires en raison de leurs bons résultats pour un grand nombre de problèmes d'optimisation combinatoires.

Le chapitre suivant est consacré à la présentation de l'optimisation combinatoire multiobjectif par algorithmes génétiques utilisant une approche de Pareto.

Chapitre 2

Optimisation multiobjectif par algorithmes génétiques

2.1 Introduction

L'optimisation multiobjectif consiste à optimiser plusieurs objectifs souvent contradictoires en même temps. Elle fait depuis plusieurs décennies l'objet de recherches intensives et s'applique à de nombreux domaines (finance, ingénierie, télécommunications, environnement, énergie, bio-informatique, surveillance, transport, logistique). Par exemple, les problèmes de tournés multiobjectifs comme TSP et VRP sont largement étudiés en raison de leurs nombreuses applications réelles dans le domaine du transport et de la logistique (Jozefowiez et al., 2008). Pour ces problèmes, les objectifs les plus communs comprennent la minimisation de la distance totale traversée, du temps total nécessaire, du coût total de la tournée, et/ou du nombre de véhicules de manutention utilisés, et la maximisation de la qualité de service et/ou du profit collecté. Par la suite, sans perte de généralité, nous considérons la minimisation de toutes les fonctions objectifs.

Contrairement à l'optimisation mono-objectif, la solution optimale d'un problème d'optimisation multiobjectif (MOP par la suite, de l'anglais *Multiobjective Optimization Problem*) n'est pas une solution unique, mais une variété de solutions, appelé l'ensemble des solutions *Pareto optimales* ou le *front Pareto*. L'optimalité des solutions n'est pas définie en termes de minimisation simultanée des fonctions objectifs, mais

en termes de *compromis* vis-à-vis des objectifs du problème. Toute solution de compromis est optimale dans le sens qu'aucune amélioration ne peut être réalisée sur une fonction objectif sans dégradation d'au moins une autre fonction objectif. Cet aspect provient du fait de la présence de conflits entre les divers objectifs du problème.

Organisation. Ce chapitre est consacré à l'optimisation multiobjectif par les algorithmes génétiques (GAs) utilisant une approche de Pareto. Dans la première section, nous introduisons les principales définitions et notions de base liées à l'optimisation multiobjectif. Puis, nous présentons dans la deuxième section une généralisation des approches de résolution de MOPs. La troisième section présente le principe de fonctionnement et les mécanismes de recherche notables des GAs multiobjectifs (*Multiobjective Genetic Algorithms* : MOGAs[1]) basés sur une approche Pareto. Dans la quatrième section, nous portons notre attention sur deux MOGAs basés sur une approche Pareto de référence de la littérature (à savoir le NSGA-II et le SPEA2). Enfin la dernière section est consacrée à la description de deux métriques complémentaires utilisées dans la littérature pour évaluer la qualité d'une approximation du front Pareto pour un MOP.

2.2 Concepts de base et définitions

L'objectif de cette section est de présenter les principales définitions et les concepts de base rencontrés dans le domaine de l'optimisation multiobjectif, comme la dominance, l'optimalité au sens de Pareto, l'ensemble Pareto optimal et le front Pareto.

Tout comme pour l'optimisation mono-objectif, les MOPs se divisent principalement en deux grandes catégories : ceux dont les solutions peuvent être codées par des variables *réelles*, appelés aussi les problèmes *continus*, et ceux dont les solutions peuvent être codées à l'aide des variables *discrètes* comme les problèmes *combinatoires*. Dans le reste ce chapitre et de cette thèse, nous nous intéressons aux problèmes combinatoires multiobjectifs.

Définition 2.1 (Problème d'optimisation combinatoire multiobjectif) *Un*

[1]Le MOGA est un GA multiobjectif spécifique dans la littérature (Fonseca et al., 1993). Nous utilisons l'abréviation "MOGA" dans cette thèse pour désigner un GA multiobjectif général.

problème d'optimisation combinatoire multiobjectif $(\mathcal{S}, F)$ *peut être formulé par :*

$$\min \{F(X) = (f_1(s), f_2(s), \ldots, f_m(s)), s \in \mathcal{S}\} \tag{2.1}$$

où $m \geq 2$ *est le nombre de fonctions objectifs,* $s \in \mathbb{Z}^n$ *est une instanciation des variables de décision* $(x_1, x_2, \ldots, x_n)$*,* $\mathcal{S}$ *est l'ensemble des points réalisables (associé à des contraintes d'égalité, d'inégalité et des bornes explicites), et* $F : \mathbb{Z}^n \rightarrow \mathbb{R}^m$ *est le vecteur des objectifs à optimiser.*

L'espace de recherche ou l'espace des paramètres, noté Ω, correspond à l'ensemble des valeurs pouvant être pris par les variables. L'ensemble $\mathcal{S} \subset \Omega$ représente l'*espace de décision* ou l'espace des solutions du problème. On note par $\Lambda = F(\mathcal{S}) := \{F(s), s \in \mathcal{S}\} \subseteq \mathbb{R}^m$ l'ensemble des points réalisables dans l'*espace des objectifs*. La figure 2.1 illustre les différents espaces pour un problème bi-objectifs à deux paramètres.

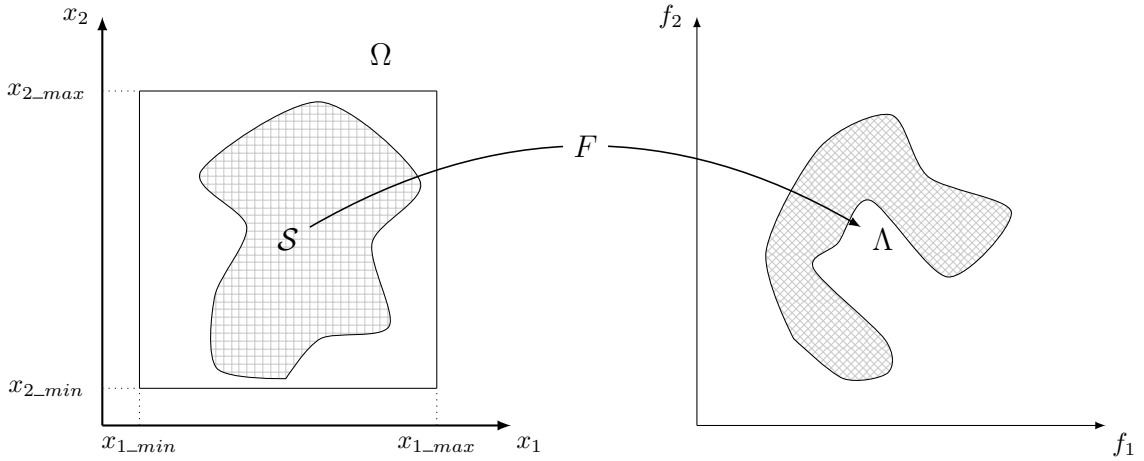


Figure 2.1 – Espace de recherche, espace réalisable et espace des objectifs.

Les problèmes d'optimisation combinatoires multiobjectifs sont caractérisés par un ensemble de solutions réalisables et un ensemble de solutions optimales finis.

L'optimisation combinatoire multiobjectif constitue depuis les années 1990 un domaine de recherche d'un grand intérêt, comme montré dans (Ehrgott et Gandibleux, 2000, 2003). De nombreux problèmes réels et académiques populaires ont été formulés comme des problèmes d'optimisation combinatoires multiobjectifs (par exemple : le

problème du TSP, le problème de VRP, le problème du sac à dos, le problème du QAP, des problèmes d'ordonnancement). La plupart des modèles multiobjectifs concernent des problèmes d'optimisation $\mathcal{NP}$-difficiles.

Pour un problème mono-objectif, la solution optimale est clairement définie : celle qui minimise la fonction objectif. Dans le cas de l'optimisation multiobjectif, il n'est pas usuel d'avoir une solution $s^* \in \mathcal{S}$ qui est optimale pour tous les objectifs, c'est-à-dire :

$$\forall s \in \mathcal{S}, f_i(s^*) \leq f_i(s), \quad i = 1, \ldots, m \tag{2.2}$$

Comme cette situation arrive rarement dans les problèmes réels où les objectifs sont en conflit, l'identification des meilleurs compromis entre tous les objectifs nécessite la définition d'une relation d'ordre partiel entre les points réalisables dans l'espace des objectifs, appelée ***relation de dominance***. La plus utilisée est la relation de dominance au sens de Pareto, formulée par l'économiste Vilfredo Pareto à la fin du 19$^{\text{ième}}$ siècle (Pareto, 1896). De manière à définir formellement cette notion, les relations de comparaison $=, >, <$ usuelles sont étendues aux vecteurs.

Définition 2.2 (Dominance au sens de Pareto) *Un vecteur des objectifs* $u = (u_1, \ldots, u_m)$ *domine un autre vecteur des objectifs* $v = (v_1, \ldots, v_m)$ *(on note* $u \prec v$*) ssi* u *est partiellement inférieure à* v*, c'est-à-dire,*

$$\forall i \in \{1, \ldots, m\}, u_i \leq v_i \wedge \exists i \in \{1, \ldots, m\} : u_i < v_i.$$

Les vecteurs des objectifs qui ne se dominent pas entre eux sont dits ***équivalents*** ou ***incomparables***. La figure 2.2 illustre, sur un exemple en deux dimensions, la relation de dominance au sens de Pareto. Pour un point y dans l'espace des objectifs, on distingue trois zones :

- la zone de préférence qui contient les points dominés par le point y ;
- la zone de dominance qui contient les points dominant le point y ;
- la zone d'équivalence qui contient les points incomparables avec le point y.

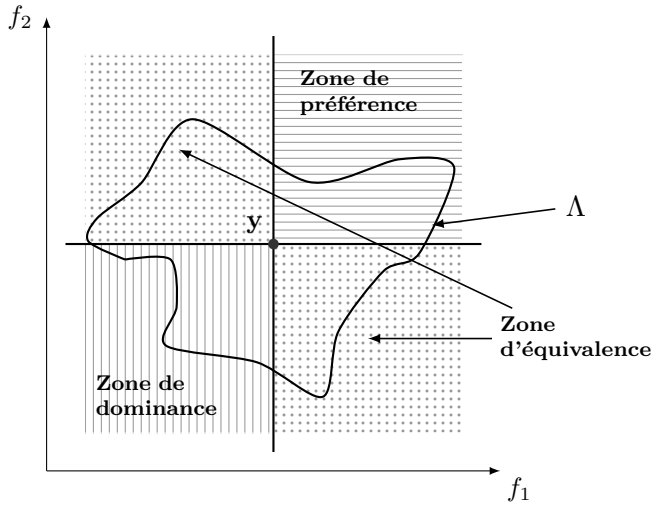


Figure 2.2 – Dominance au sens de Pareto.

Le concept généralement utilisé est la notion d'*optimalité Pareto*. La définition de solution Pareto optimale découle directement de la notion de dominance.

Définition 2.3 (Optimalité globale au sens Pareto) *Une solution* $s^* \in \mathcal{S}$ *est globalement Pareto optimale ssi* $\forall s' \in \mathcal{S}, F(s') \not\prec F(s^*)$.

Définition 2.4 (Optimalité locale au sens Pareto) *Une solution* $\hat{s} \in \mathcal{S}$ *est localement Pareto optimale ssi* $\forall s' \in (\mathcal{N}(\hat{s}) \subseteq \mathcal{S}), F(s') \not\prec F(\hat{s})$, *où* $\mathcal{N}(\hat{s})$ *est un ensemble de solutions voisines de* $\hat{s}$.

Les notions d'optimalité au sens de Pareto sont illustrées graphiquement dans la figure 2.3, en deux dimensions.

Nous rappelons que la solution d'un MOP n'est pas une solution unique mais un collection de solutions appelé l'ensemble de *meilleurs compromis*. La définition de cet ensemble découle directement du concept de dominance au sens de Pareto. Nous définissons d'abord formellement cet ensemble dans l'espace de décision.

Définition 2.5 (Ensemble de Pareto optimal) *Pour un MOP donné* $(\mathcal{S}, F)$, *l'ensemble de Pareto optimal correspond à l'ensemble des solutions non dominées de* $\mathcal{S}$: $\mathcal{P}^* = \{s \in \mathcal{S} | \neg\exists s' \in \mathcal{S}, F(s') \prec F(s)\}$.

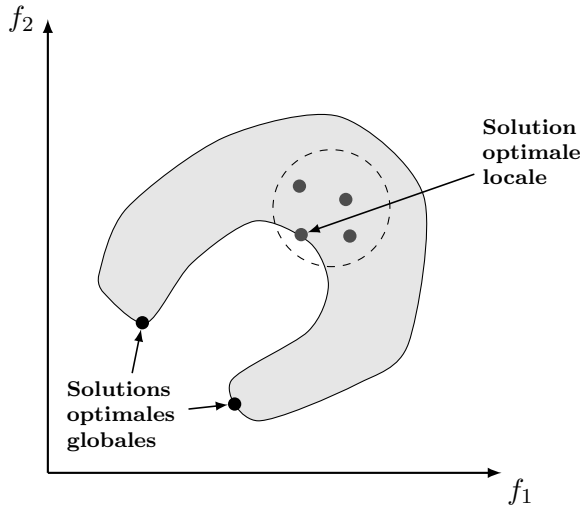


Figure 2.3 – Optimalité globale et optimalité locale au sens de Pareto.

L'ensemble de Pareto optimal regroupe tous les points de l'espace des solutions qui dominent les autres mais ne se dominent pas entre eux. Le front de Pareto de l'espace des objectifs est défini de manière analogue.

Définition 2.6 (Front de Pareto global) *Pour un MOP donné $(\mathcal{S}, F)$, le front Pareto correspond à l'image dans l'espace des objectifs de son ensemble Pareto optimal $\mathcal{P}^*$: $\mathcal{FP}^* = \{F(s), s \in \mathcal{P}^*\}$.*

Le front de Pareto global regroupe tous le meilleurs compromis possibles au MOP. Cet ensemble est aussi appelé le ***front optimal de Pareto***, ou plus généralement la ***surface de compromis*** pour des problèmes de dimension plus élevée. En pratique, puisque le front de Pareto optimal peut contenir un nombre très grand de points, les méthodes de résolution doivent localiser un assortiment de solutions proches du front Pareto global.

2.3 Approches de résolution

Le but de résolution des MOPs est d'aider le décideur (une personne ou un groupe de personnes) dans le choix d'une solution parmi l'ensemble Pareto optimal. Une des questions fondamentales en optimisation multiobjectif est liée à l'interaction entre la

méthode de résolution et le décideur. En fait, les solutions Pareto optimales ne peuvent être rangées globalement. Le rôle de décideur est de spécifier ses préférences afin d'induire un ordre total entre les solutions Pareto, puis, sélectionner sa solution préférée. Selon le moment où les préférences du décideur sont incorporées dans le processus de résolution, trois grandes catégories d'approches utilisées pour la résolution d'un MOP sont distinguées (Miettinen, 1999) :

- **Méthodes *a priori*** (*Préférences* $\rightarrow$ *Recherche*) : Ces approches nécessitent pour le décideur de définir le compromis qu'il désire atteindre avant de lancer l'optimisation. Différentes formes peuvent remplir ce rôle. Dans de nombreux cas, les approches suggérées consistent à transformer le problème multiobjectif en un problème à un objectif en combinant les différentes fonctions objectifs en une seule fonction objectif (méthodes d'agrégation des objectifs). Le problème réduit à résoudre sera :

  $$\min v(F(s)), \quad s \in \mathcal{S}$$

  où v est la fonction d'utilité qui exprime les préférences du décideur. Elle agrège le vecteur des objectifs en une seule fonction scalaire : $v : \mathbb{R}^m \rightarrow \mathbb{R}$.

  La fonction d'utilité définira un ordre total dans l'ensemble Pareto optimal. Par exemple, si la fonction d'utilité est linéaire, c'est-à-dire

  $$v(F(s)) = \sum_{i=1}^{m} \lambda_i f_i(s),$$

  le décideur doit évaluer *a priori* le poids $\lambda_i \geq 0$ de chaque fonction objectif $i = 1, \ldots, m$, avec $\sum_{i=1}^{m} \lambda_i = 1$. Cependant, dans de nombreux cas, il est très difficile de définir la fonction d'utilité avant le processus d'optimisation. Ces approches nécessitent alors pour le décideur d'avoir une bonne connaissance de son problème.

- **Méthodes progressives ou interactives** (*Recherche* $\leftrightarrow$ *Préférences*) : Dans ce cas, le décideur affine progressivement son choix des compromis au cours de l'optimisation. A partir des connaissances extraites durant la résolution du

problème, le décideur peut définir ses préférences de manière claire et compréhensible. Ces préférences sont prises en compte par la méthode de résolution durant les étapes de recherche suivantes. Ce processus est répété plusieurs fois jusqu'à la satisfaction du décideur.

- **Méthodes *a posteriori*** (*Recherche* → *Préférences*) : Ces méthodes ne nécessitent ni la spécification des préférences, ni l'interaction avec le décideur. Elles fournissent à la fin de recherche au décideur un ensemble de solutions de compromis plutôt qu'un unique compromis lui laissant un choix à faire *a posteriori*. Cette approche est pratique si le nombre d'objectifs est petit et la cardinalité de l'ensemble Pareto optimal est réduite. En termes de résolution de problème, cette approche est plus complexe dans le sens qu'une variété de solutions Pareto optimales doit être fournie.

En effet, chaque approche de résolution a ses avantages et ses inconvénients. Le choix d'une méthode dépond des propriétés du problème à traiter et des compétences du décideur.

Les méthodes *a priori* ne sont efficaces que lorsque l'espace des objectifs présente des propriétés de convexité[2].

Les méthodes progressives sont limitées par le fait qu'elles monopolisent l'attention du décideur tout au long du processus d'optimisation. Comme signalé dans (Regnier, 2003), cet aspect peut être pénalisant si l'évaluation des fonctions objectifs nécessite un temps de calcul important et que les interventions du décideur sont fréquentes.

L'avantage majeur des méthodes *a posteriori* est la reproduction d'un ensemble de solutions optimales. Néanmoins, cela impose le choix d'une solution à partir de cet ensemble selon des préférences choisies par le décideur. Ce choix peut être guidé par la prise en compte de critères supplémentaires (Regnier, 2003). La plupart de ces méthodes utilisent des algorithmes métaheuristiques capables d'explorer plusieurs solutions en parallèle et permettant l'implémentation d'approches Pareto. En particulier, les algorithmes évolutionnaires multiobjectifs de type GAs rencontrent ces deux

[2]Un front Pareto est dit *convexe* si, étant donné deux points distincts quelconques de ce front, le segment de droite qui relie ces deux points est toujours contenu dans l'ensemble des points réalisables de l'espace des objectifs.

dernières décennies un succès grandissant en raison de leur bonne adéquation avec les MOPs (Coello et al., 2007, Deb, 2001). Ces algorithmes à base de population de solutions sont très recommandés pour les problèmes d'optimisation réels dont les espaces des objectifs présentent des propriétés de non-convexité, multimodalité, discontinuité, etc.

Dans le reste ce chapitre et de cette thèse, nous nous intéresserons aux MOGAs utilisant des approches Pareto. L'aspect d'aide à la décision pour le choix d'une solution de compromis parmi les solutions Pareto fournies, n'est pas abordé (Miettinen, 1999).

2.4 Principaux composants de recherche d'un logarithme génétique multiobjectif

L'objectif qui doit être pris en compte dans la résolution Pareto d'un MOP par un GA est : converger vers des solutions diversifiées dans le front Pareto. Pour répondre à cet objectif, en plus de concepts communs des GAs mono-objectifs (voir chapitre 1), les MOGAs basés sur des approches Pareto contiennent trois composants de recherche essentiels (Talbi, 2009) :

- **Affectation de fitness (ou *ranking*)** : Le rôle de cette procédure est de guider la recherche vers des solutions Pareto optimales. Elle permet de quantifier le compromis entre toutes les objectifs. Elle calcule pour chaque nouvelle solution (un vecteur des objectifs) une valeur scalaire pour former son adaptation.

- **Maintien de la diversité (ou *nichage*)** : Le but ici est de générer un ensemble de solutions Pareto optimales diversifiées dans l'espace des paramètres et/ou dans l'espace des objectifs.

- **Elitisme** : L'idée est de réserver les meilleures solutions (Pareto optimales) pour accélérer la convergence et améliorer les performances de l'algorithme.

Par ailleurs, si les MOGAs sont combinés avec d'autres techniques d'optimisation -pour améliorer leurs performances-, on parle des MOGAs **hybrides**.

La figure 2.4 illustre la structure d'un MOGA intégrant ces mécanismes. Les différences entre les MOGAs récents résident principalement dans ces mécanismes et la manière de sélection de parents géniteurs. Nous décrivons par la suite comment ces mécanismes peuvent être définis indépendamment pour concevoir un MOGA. Les techniques utilisées par les algorithmes NSGA-II (*Nondominated Sorting Genetic Algorithm*) et SPEA2 (*Strength Pareto Evolutionary Algorithm*), présentés dans les sections 2.5.1 et 2.5.2 respectivement, donnent des exemples à ces mécanismes.

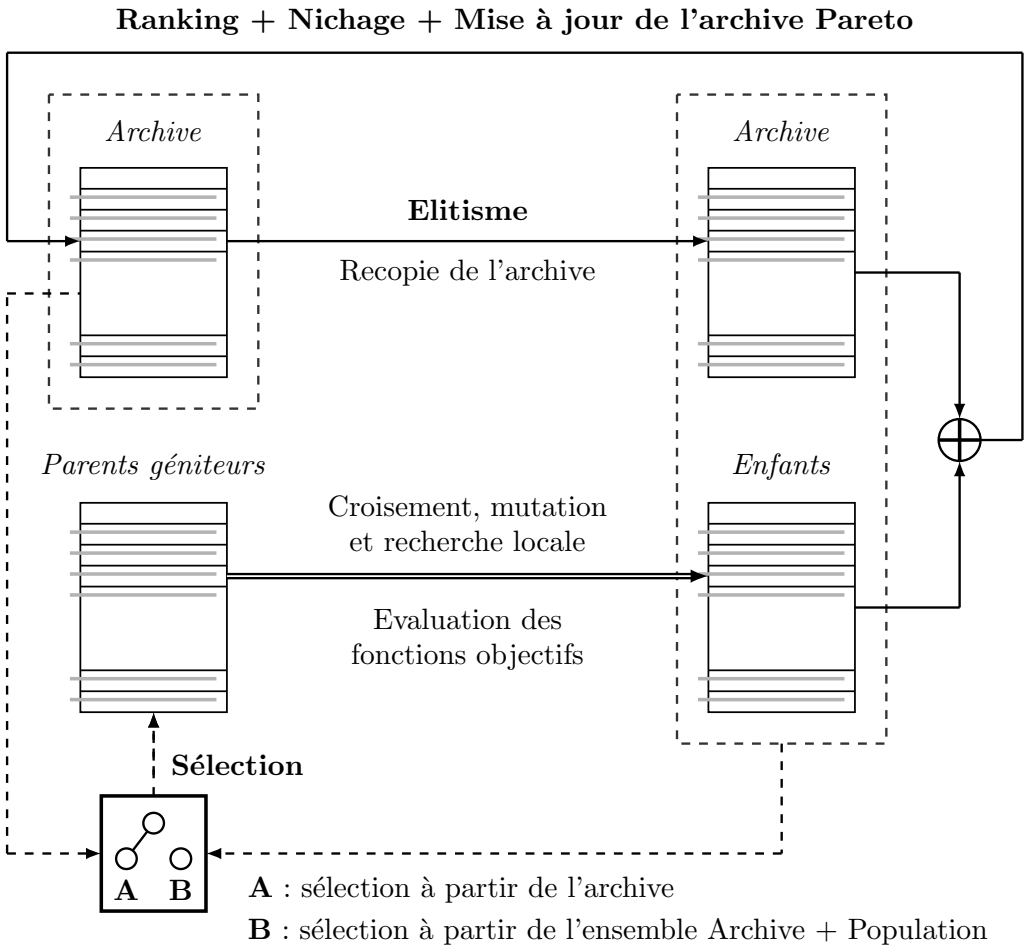


Figure 2.4 – Structure d'un MOGA élitiste avec nichage (inspirée de (Regnier, 2003)).

2.4.1 Procédures de ranking

Une procédure de sélection Pareto utilise directement la notion d'optimalité Pareto pour ranger les individus de la population, contrairement aux autres approches (non Pareto) qui transforment le MOP en un problème mono-objectif ou qui traitent séparément les différents objectifs. Cette idée a été initialement proposée par Goldberg dans (Goldberg, 1989). En effet, l'avantage majeur des approches Pareto est qu'elles sont capables de générer des solutions Pareto optimales dans les portions concaves du front Pareto. Plusieurs méthodes de *ranking* ont été utilisées dans la littérature (Zitzler et al., 2004). La plus communément utilisée est la technique NSGA qui décompose une population de solutions en plusieurs fronts de Pareto (voir paragraphe 2.4.1.1).

Puisqu'une valeur unique de fitness (rang ou score) est affectée à chaque solution de la population, tout composant de recherche d'une métaheuristique mono-objectif peut être utilisé pour la résolution de MOPs. Par exemple, le mécanisme de sélection de parents géniteurs dans les MOGAs peut être dérivé des mécanismes de sélection utilisés dans l'optimisation mono-objectif. En comparaison avec les approches qui utilisent des fonctions d'utilité, l'intérêt des approches Pareto est qu'elles évaluent la qualité d'une solution par rapport à la population entière ; pas de valeurs absolues affectées aux solutions.

2.4.1.1 Procédure de ranking NSGA

La procédure de ranking NSGA a été initialement proposée par Goldberg dans (Goldberg, 1989) et implémentée par Srinivas et Deb dans (Srinivas et Deb, 1994). Elle est appliquée pour établir un ordre entre les solutions d'une population. Initialement, tous les individus non dominés de la population reçoivent le rang 1 et forment le premier front $\mathcal{F}_1$. Puis, les individus qui ne sont dominés que par les individus de $\mathcal{F}_1$ reçoivent le rang 2 ; ils forment le deuxième front $\mathcal{F}_2$. D'une manière générale, un individu reçoit un rang k si et seulement si il n'est dominé que par les individus appartenant à l'union $\mathcal{F}_1 \cup \mathcal{F}_2 \cdots \cup \mathcal{F}_{k-1}$. Le processus est réitéré jusqu'à ce que tous les individus de la population aient un rang. La figure 2.5 illustre graphiquement la notion de rang de dominance dans le cas de problèmes bi-objectifs. La figure 2.6 illustre le

déroulement de cette procédure de ranking sur un ensemble de solutions.

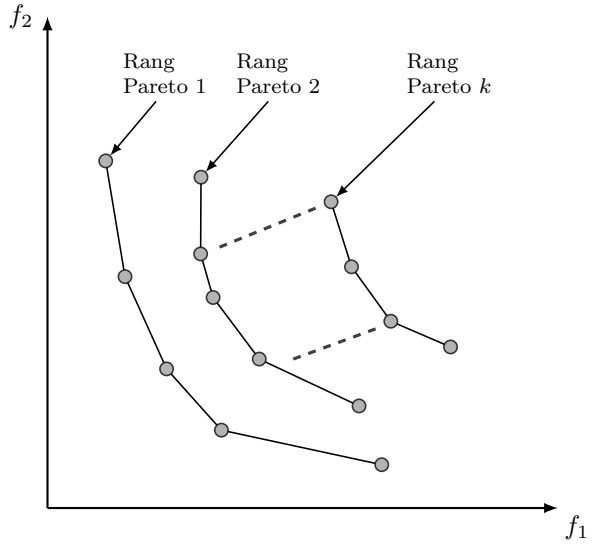


Figure 2.5 – Ranking de Goldberg (NSGA), pour un MOP bi-objectifs.

L'algorithme 2.1 donne le pseudo-code de l'algorithme d'affectation du rang de Pareto. Dans cet algorithme, la variable P désigne l'ensemble de points (p_i) dont on désire identifier les différents rangs. Cet algorithme intuitif a une complexité moyenne en $O(mN^3)$, où m correspond au nombre de fonctions objectifs et N à la taille de la population. Les auteurs de NSGA-II (Deb et al., 2002) ont amélioré cet algorithme et réduit sa complexité à $O(mN^2)$.

2.4.2 Nichage

Nous rappelons que l'objectif qui doit être pris en compte dans la résolution Pareto d'un MOP est d'obtenir des solutions diversifiées sur le sous-espace Pareto optimal. Les méthodes de ranking, comme la méthode NSGA présentée dans le paragraphe précédent, assurent la convergence vers des solutions Pareto optimales. Cependant, la diversification des solutions n'est pas prise en compte, ce qui rend impossible la découverte de l'intégralité du front Pareto. Pour obtenir une population diversifiée, dans l'espace des paramètres et/ou dans l'espace des objectifs, il faut combiner les méthodes de ranking avec les techniques de formation et de maintien de niches. En général, les techniques de diversification pénalisent les solutions à forte densité de

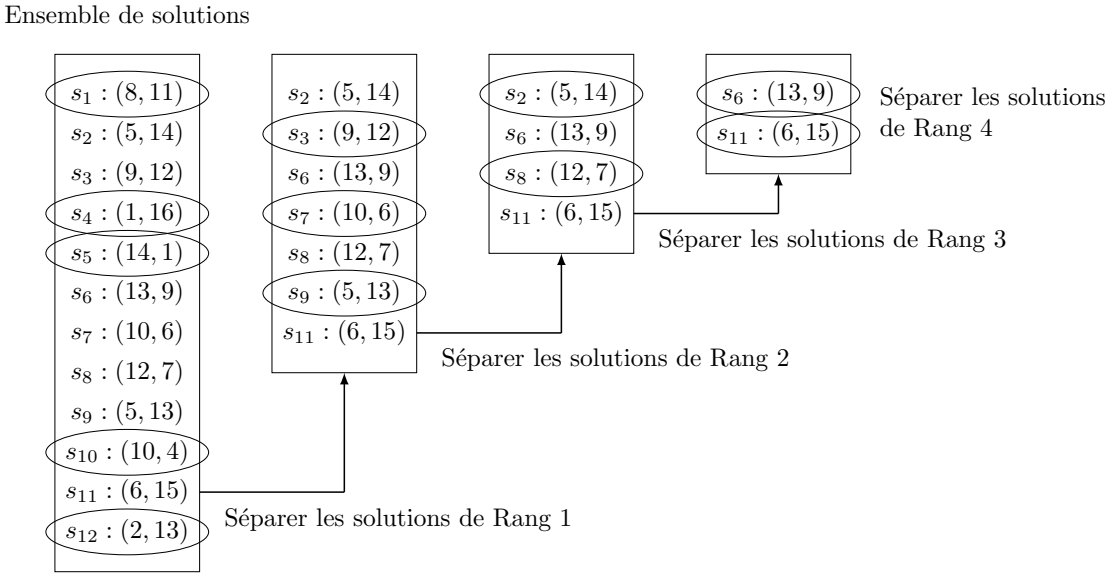


Figure 2.6 – Illustration de la procédure de ranking NSGA pour un MOP bi-objectifs (inspirée de (Talbi, 2009)).

Algorithme 2.1 : Affectation du rang de Pareto (Ranking NSGA)

```
1  RangCourant ← 1
2  tant que (|P| ≠ 0) faire
3  |  pour chaque p_i ∈ P faire
4  |  |  si (p_i est non dominé) alors
5  |  |  |  Rang(p_i) ← RangCourant
6  |  |  fin
7  |  fin
8  |  pour chaque p_i ∈ P faire
9  |  |  si (Rang(p_i) = RangCourant) alors
10 |  |  |  Enlever temporairement p_i de P
11 |  |  fin
12 |  fin
13 |  RangCourant ← RangCourant + 1
14 fin
```

solutions autour d'elles. Elles peuvent être classées en trois catégories (Talbi, 2009, Zitzler et al., 2004) :

- **Méthodes de voisinage** : Comme nous allons le voir dans la suite de ce chapitre, les algorithmes SPEA2 et NSGA-II sont caractérisés par des estimations de la densité basées sur le principe de voisinage. SPEA2 prend en compte la

distance entre une solution s_i et son $k^{\text{ième}}$ voisin le plus proche pour estimer la densité. NSGA-II est caractérisé par une estimation de la densité liée à l'environnement de chaque solution, appelée distance de *crowding*.

- **Méthodes de partage de performance** : Ces méthodes définissent le voisinage d'une solution selon une fonction kernel K, appelée fonction de partage, qui a comme paramètre d'entrée la distance entre deux solutions. Pour chaque solution s_i, les distances $d(s_i, s_j)$ entre s_i et toutes les autres solutions s_j de la population *pop* sont calculées. Ensuite, la densité estimée de la solution s_i, appelée *compte de niche*, est donnée par la somme des valeurs de la fonction de partage K appliquée à toutes les distances :

$$m(s_i) = \sum_{s_j \in pop} K(d(s_i, s_j))$$

Les valeurs de la fonction m doivent être élevées pour les solutions qui sont proches les unes des autres et faibles pour les solutions isolées.

Ainsi, la performance partagée f' d'une solution s_i est égale à la valeur de fitness originale $f(s_i)$ (i.e., la valeur scalaire retournée par une méthode de ranking) divisée par la densité de la solution $m(s_i)$:

$$f'(s_i) = \frac{f(s_i)}{m(s_i)}$$

De cette manière, la performance d'une solution va être dégradée en fonction de la densité de solutions se trouvant à proximité.

Une des fonctions de partage les plus utilisées est celle proposée par Goldberg et Richardson dans (Goldberg et al., 1987) :

$$sh(d(s_i, s_j)) = \begin{cases} 1 - \frac{d(s_i, s_j)}{\sigma} & \text{si } d(s_i, s_j) < \sigma \\ 0 & \text{sinon} \end{cases}$$

Cette fonction retourne une valeur dans l'intervalle $]0, 1]$ si les deux solutions

sont proches, 0 si la distance entre les deux dépasse un certain seuil σ. Une telle méthode de partage a été utilisée dans l'algorithme NSGA (Srinivas et Deb, 1994), le précurseur de NSGA-II.

- **Histogrammes** : Cette approche consiste à partitionner l'espace de recherche en plusieurs hypergrilles définissant les voisinages. La densité de la population autour d'une solution est estimée par le nombre de solutions occupant le même hypercube.

Il est à noter que l'une des questions les plus importantes dans les approches de maintien de la diversité concerne la mesure de distance. Plusieurs mesures peuvent être utilisées, comme la distance de Hamming ou les distances Euclidiennes. D'ailleurs, la distance peut être calculée dans l'espace des paramètres et/ou dans l'espace des objectifs. En général, la formation et le maintien des niches se font dans l'espace des objectifs en raison de leur simplicité.

2.4.3 Elitisme

Dans le cas mono-objectif, l'élitisme consiste à conserver le meilleur individu pour les générations futures afin d'éviter sa perte éventuelle au cours de l'application des opérateurs génétiques de croisement et de mutation. Réaliser un GA élitiste dans le cadre de résolution de MOPs est plus difficile que pour le cas mono-objectif. Cette difficulté réside dans le fait que la meilleure solution n'est pas une solution unique, mais un ensemble de solutions. L'idée consiste donc à conserver un nombre limité de ces solutions optimales dans une population secondaire, appelée *archive*, de taille constante associée à la population courante (Zitzler et al., 2000).

Comme décrit dans (Talbi, 2009), si l'élitisme est utilisé uniquement pour éviter la perte éventuelle des solutions Pareto optimales obtenues, on parle d'une stratégie d'élitisme passif. Dans ce cas, l'archive est considérée comme une population secondaire séparée qui n'a aucun impact sur le processus de recherche. L'élitisme passif ne garantira qu'une performance non dégradante de l'algorithme en termes d'approximation du front Pareto. En revanche, si l'élitisme est utilisé dans le processus de recherche, c'est-à-dire les solutions élites sont utilisées pour générer de nouvelles solutions, on

parle d'une stratégie d'élitisme actif. Cet élitisme permet une convergence rapide et robuste vers une bonne approximation du front Pareto (Zitzler et al., 2000). La plupart des MOGAs récents appliquent un élitisme actif. Comme le montre la figure 2.4, les parents géniteurs peuvent être sélectionnés à partir de l'archive (comme c'est le cas pour NSGA-II et SPEA2) ou à partir de la population formée par les membres de l'archive et les enfants de la génération précédente. Il est à noter que les tailles des populations archive, de géniteurs et d'enfants, peuvent être identiques ou distinctes.

A chaque génération, l'archive est mise à jour en fonction des nouvelles solutions explorées. En général, les membres de l'archive sont choisis à l'aide d'une méthode de nichage qui préserve les solutions les plus diversifiées sur l'ensemble du front Pareto. L'archive conserve donc les solutions Pareto optimales les plus représentatives obtenues tout au long de la recherche.

2.4.4 Hybridation

Tout comme pour le cas mono-objectif, un autre mécanisme important afin améliorer les performances d'une métaheuristique multiobjectif consiste à l'hybrider avec d'autres méthodes d'optimisation. Les métaheuristiques multiobjectifs hybrides sont reconnues depuis plus d'une décennie comme des approches compétitives pour traiter les MOPs de grande taille (Ehrgott et Gandibleux, 2008, Talbi, 2015). Talbi dans (Talbi, 2015) a classé les métaheuristiques hybrides dans le domaine de la résolution des MOPs en trois grandes catégories :

- **Hybridation des métaheuristiques avec des (méta)heuristiques** : C'est la forme d'hybridation la plus répondue. Elle consiste à combiner des concepts et composants de recherche de (méta)heuristiques différentes pour produire un algorithme performant.

- **Hybridation des métaheuristiques avec les méthodes exactes** : Dans ce cas, des métaheuristiques pures sont combinées avec des méthodes de programmation mathématiques, comme la méthode *branch and bound* et la méthode *branch and cut*, pour explorer des voisinages très larges ou pour compléter des solutions partielles.

- **Hybridation des métaheuristiques avec les techniques de *data mining* et d'apprentissage** : Cette catégorie englobe les approches qui choisissent et adaptent les paramètres de contrôle d'un algorithme sans intervention de l'utilisateur, les approches qui appliquent les opérateurs de recherche (par exemple, les opérateurs de recombinaison des métaheuristiques à population, les structures de voisinage des métaheuristiques à solution unique) de manière adaptative, et les approches qui transforment le MOP initial à un MOP réduit en le décomposant en plusieurs sous-problèmes ou en ajoutant de nouvelles contraintes.

Pour les MOGAs, l'hybridation consiste souvent à substituer l'opérateur de mutation par une procédure de recherche locale (i.e., algorithmes mémétiques). Celle-ci est appliquée pour générer un (ou le) ensemble de voisins de l'individu qui va subir une mutation. Parmi plusieurs algorithmes multiobjectifs mémétiques proposés dans la littérature, nous pouvons citer : l'algorithme présenté dans (Barichard et Hao, 2003) qui combine un MOGA et une recherche taboue (appliqué au problème du sac à dos multiobjectif), et l'algorithme proposé dans (Meunier et al., 2000) qui applique une procédure de recherche locale pour améliorer la population d'un MOGA (appliqué au problème du design de réseaux).

2.5 Deux algorithmes génétiques multiobjectifs performants

Les deux MOGAs basés Pareto choisis pour servir de base à notre approche hybride présenté dans le chapitre 3 sont NSGA-II et SPEA2. Ces deux algorithmes (classiques) de référence de la littérature servent de guide pour le développement de nouveaux algorithmes multiobjectifs depuis leur apparition.

Notons que NSGA-II et SPEA2 serviront non seulement à construire la partie génétique de notre approche hybride, mais également pour fournir des éléments de comparaison afin d'évaluer les performances de l'hybridation proposée.

2.5.1 Le NSGA-II

L'algorithme NSGA-II (Deb et al., 2002) est une version élitiste de NSGA (Srinivas et Deb, 1994) qui est une implémentation directe de la technique de ranking de Goldberg (Goldberg, 1989). Pour maintenir la diversité et forcer le nichage, NSGA-II utilise un opérateur de sélection (d'individus élites), basé sur un calcul de la distance de "crowding" dans l'espace des objectifs, différent de celui de son précurseur.

2.5.1.1 Principe général de fonctionnement

Comme le montre la figure 2.7, NSGA-II manipule deux populations de même taille N : La population P_t (archive) qui contient les meilleurs individus rencontrés jusqu'à la génération t, et la population Q_t (d'enfants) qui contient les individus générés à partir des individus de P_t. La première étape d'une génération t consiste à fusionner ces deux populations $R_t = P_t \cup Q_t$ et à classer tous les individus selon le front Pareto auquel ils appartiennent (i.e., l'application de la procédure de ranking NSGA ; voir paragraphe 2.4.1).

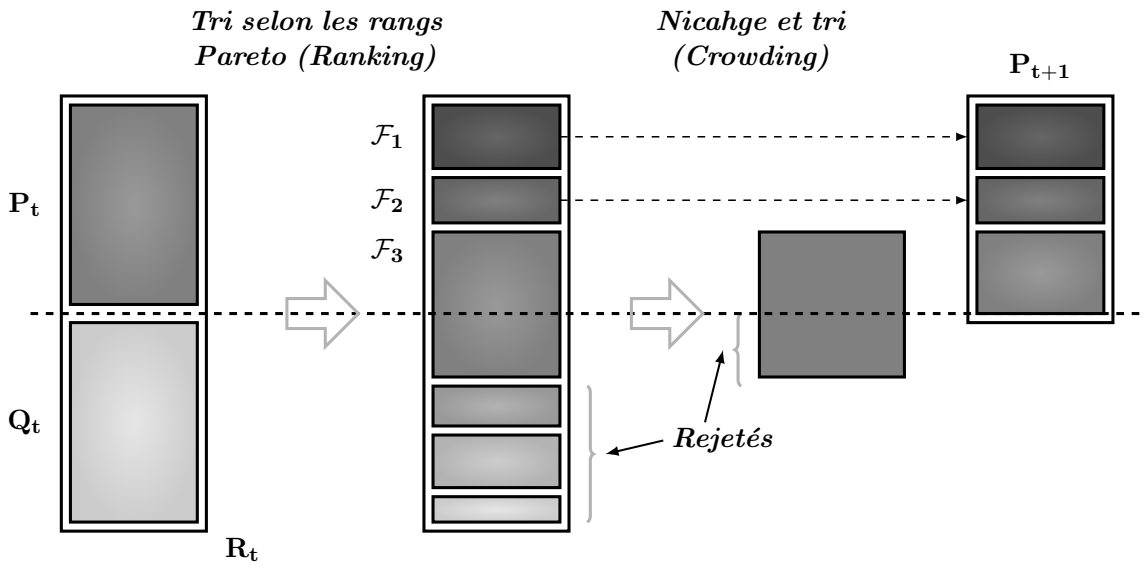


Figure 2.7 – Fonctionnement général de l'algorithme NSGA-II (d'après (Deb et al., 2002)).

La deuxième phase consiste à construire l'archive diversifiée P_{t+1} qui contient les N meilleurs individus de la population R_t. Il faut pour cela insérer dans P_{t+1} les individus

de fronts successifs de R_t tant que le nombre d'éléments insérés reste inférieur à la taille limitée N. Lorsqu'il n'est pas possible d'insérer tous les individus d'un même front dans l'archive, le choix se fait par troncation à partir de la distance de *crowding* (décrite ci-dessous).

La dernière phase consiste à créer une nouvelle population d'enfants Q_{t+1} en appliquant les opérateurs de sélection, de croisement et de mutation sur les individus de P_{t+1}. L'algorithme 2.2 regroupe ces étapes.

Algorithme 2.2 : Pseudo-code de l'algorithme NSGA-II (Deb et al., 2002)

```
– Initialiser l'archive P_0 de taille N
– Créer la population Q_0 de taille N à partir de P_0 par application des
  opérateurs de sélection, de croisement et de mutation
– Initialiser le compteur du temps t ← 0
tant que (La condition d'arrêt n'est pas vérifiée) faire
    – Création de R_t = P_t ∪ Q_t
    – Calculer les fronts F = {F_1, F_2, ...} de R_t par un algorithme de
      ranking (voir algorithme 2.1)
    – Vider P_{t+1} et initialiser i = 0
    tant que (|P_{t+1}| + |F_i| ≤ N) faire
        P_{t+1} = P_{t+1} ∪ F_i
        i = i + 1
    fin
    – Calculer la distance de crowding pour chaque individu dans F_i
    – (voir algorithme 2.3)
    – Trier les individus du front F_i selon la distance de crowding
      par ordre décroissant
    – Choisir les N − |P_{t+1}| premiers éléments de F_i :
      P_{t+1} = P_{t+1} ∪ F_i[1 : N − |P_{t+1}|]
    – Sélection des géniteurs à partir de P_{t+1} et création d'une nouvelle
      population Q_{t+1} par application des opérateurs de croisement et de
      mutation
    – Incrémenter le compteur du temps t ← t + 1
fin
```

2.5.1.2 Nichage : calcul de la distance de "crowding"

L'algorithme NSGA-II est enrichi par une méthode permettant de partager l'adaptation entre les individus d'un même front, appelée la distance de *crowding* (Deb et al.,

2002). L'algorithme 2.3 présente le pseudo-code de l'algorithme d'affectation de cette distance. Dans cet algorithme, $\mathcal{I}(i).k$ désigne la valeur de la $k^{\text{ième}}$ fonction objectif de l'individu i appartenant au front $\mathcal{I}$. La complexité de cette procédure dépend de l'algorithme de tri. Dans le pire cas (tous les individus de la population forment un seul front), le tri fonctionne en $O(mN \log N)$ étapes.

Algorithme 2.3 : Pseudo-code de l'algorithme d'affectation de la distance de *crowding* (Deb et al., 2002)

```
pour chaque individu i appartenant au front I faire
    I_distance(i) = 0
    l = |I|
    pour chaque fonction objectif k = 1, ..., m faire
        – Trier les l individus du front I selon la valeur de la fonction
          objectif k par ordre croissant
        – Imposer I_distance(1) = I_distance(l) = ∞ (de sorte que les points
          extrêmes du front soient toujours sélectionnés)
        pour i ← 2 à (l − 1) faire
            I_distance(i) = I_distance(i) + (I(i + 1).k − I(i − 1).k)
        fin
    fin
fin
```

La distance de *crowding* associée au point i est égale à la somme sur toutes les fonctions objectifs de l'écart entre les deux points qui délimitent ce point. Cette mesure privilégie d'abord les points extrêmes, puis les plus isolés. La figure 2.8 illustre, pour un MOP bi-objectifs, le calcul associé au point i.

La distance de *crowding* n'intervient pas uniquement dans la sélection de membres de l'archive, mais sert aussi à la sélection de parents géniteurs. Elle est utilisée comme un critère de choix lorsque les individus en compétition font partie d'un même front. Reposant sur le principe de formation de l'adaptation de SPEA2 (voir section 2.5.2), nous pouvons former l'adaptation (fitness scalare) de l'algorithme NSGA-II comme suit :

$$Fitness_{NSGA-II}(i) = \mathcal{I}(i) + \frac{1.0}{\mathcal{I}_{distance}(i) + 2.0} \tag{2.3}$$

où $\mathcal{I}(i)$ correspond au front de l'individu i et $\mathcal{I}_{distance}(i)$ à sa distance de *crowding*. Dans le dénominateur, la valeur 2.0 est ajoutée pour s'assurer que ses valeurs soient

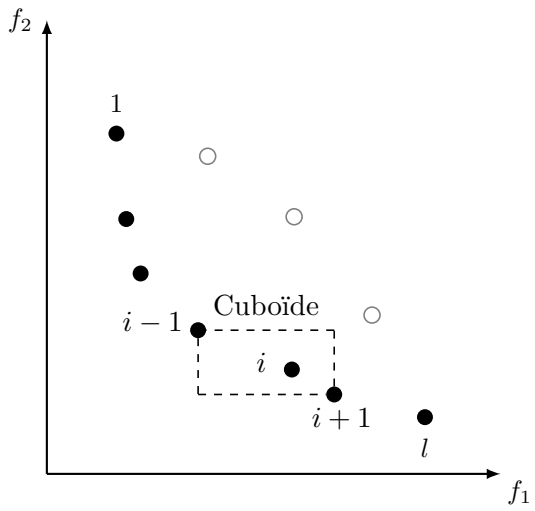


Figure 2.8 – Illustration du calcul de la distance de *crowding* (d'après (Deb et al., 2002)). Les points marqués par des cercles noirs correspondent à des solutions du même front.

supérieures à 0 et que les valeurs du second terme de la somme soient inférieures à 1. De cette manière, les valeurs d'adaptation des individus du $k^{\text{ième}}$ front sont dans l'intervalle $]k, k+0.5]$.

2.5.2 Le SPEA2

Zitzler et al. ont aussi apporté des modifications à leur algorithme SPEA (Zitzler et Thiele, 1999) pour améliorer ses performances (Zitzler et al., 2001).

2.5.2.1 Principe général de fonctionnement

La première étape consiste à créer une population initiale $\mathbf{P}_0$ de taille N et à initialiser l'archive externe $\overline{\mathbf{P}}_0$ de taille $\overline{N}$ à l'ensemble vide. L'algorithme SEPA2 va mettre à jour régulièrement cette archive en fonction des nouveaux individus non dominés.

A chaque génération, l'archive est diversifiée et remplie en gardant les $\overline{N}$ individus non dominés de l'archive et de la population précédentes. Lorsqu'il n'est pas possible d'insérer tous les individus non dominés, le choix se fait par troncation basée sur une relation de dominance prenant en compte les distances (dans l'espace des objectifs)

entre les individus. Par contre, si le nombre des individus non dominés est inférieur à $\overline{N}$, l'archive est complétée avec les individus les moins dominés parmi l'archive et la population, relativement à l'adaptation décrite ci-après.

La dernière phase d'une génération consiste à créer la population $\mathbf{P}_{t+1}$. Il suffit pour cela d'appliquer les opérateurs de croisement et de mutation sur des individus sélectionnés à partir de $\overline{\mathbf{P}}_{t+1}$, et d'insérer les enfants dans $\mathbf{P}_{t+1}$.

La figure 2.9 et l'algorithme 2.4 illustrent le schéma du fonctionnement global de SPEA2.

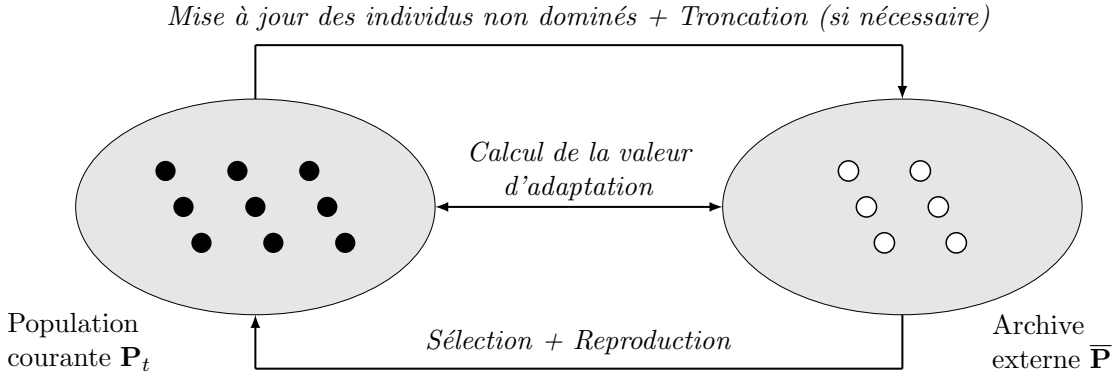


Figure 2.9 – Illustration du fonctionnement global de l'algorithme SPEA2.

2.5.2.2 Affectation de la valeur d'adaptation

Le calcul de la valeur d'adaptation de SPEA2 s'effectue en trois étapes. Initialement, chaque individu i de l'ensemble $\mathbf{P} \cup \overline{\mathbf{P}}$ reçoit une force (*strength*) $S(i)$ égale au nombre total d'individus qu'il domine :

$$S(i) = \left|\left\{j | j \in \mathbf{P}_t \cup \overline{\mathbf{P}}_t \wedge i \prec j\right\}\right| \tag{2.4}$$

où $|.|$ correspond à la cardinalité d'un ensemble et le symbole $\prec$ à la relation de dominance au sens de Pareto. Ensuite, chaque individu reçoit un score (*raw fitness*) égal à la somme des forces des individus qui le dominent parmi l'archive et la population :

$$R(i) = \sum_{j \in \mathbf{P}_t \cup \overline{\mathbf{P}}_t \wedge j \prec i} S(j) \tag{2.5}$$

Algorithme 2.4 : Pseudo-code de l'algorithme SPEA2 (Zitzler et al., 2001)

- Générer la population initiale $\mathbf{P}_0$ de taille N
- Créer l'archive externe vide $\overline{\mathbf{P}}_0$ de taille $\overline{N}$

tant que (*La condition d'arrêt n'est pas vérifiée*) **faire**

- Calculer la valeur d'adaptation pour tous les individus de $\mathbf{P}_t \cup \overline{\mathbf{P}}_t$ (voir paragraphe 2.5.2.2)
- Insérer tous les individus non dominés de $\mathbf{P}_t \cup \overline{\mathbf{P}}_t$ à $\overline{\mathbf{P}}_{t+1}$.
  - Si la taille de $\overline{\mathbf{P}}_{t+1}$ excède $\overline{N}$, réduire $\overline{\mathbf{P}}_{t+1}$ par l'opérateur de troncation (voir paragraphe 2.5.2.3)
  - Sinon, si la taille de $\overline{\mathbf{P}}_{t+1}$ est inférieure à $\overline{N}$, l'archive $\overline{\mathbf{P}}_{t+1}$ est remplie en insérant les $\overline{N} - \left|\overline{\mathbf{P}}_{t+1}\right|$ meilleurs individus dominés de $\mathbf{P}_t \cup \overline{\mathbf{P}}_t$
- Sélection des parents géniteurs à partir de $\overline{\mathbf{P}}_{t+1}$ et création de la population $\mathbf{P}_{t+1}$ par application des opérateurs de croisement et de mutation
- Incrémenter le compteur du temps $t \leftarrow t + 1$

fin

La figure 2.10 illustre, sur un exemple en dimension 2, la procédure d'affectation des scores.

Enfin, pour distinguer les individus ayant un même score, une information $D(i)$ estimant la densité de solutions autour d'un individu, est ajoutée au score pour former l'adaptation :

$$Fitness_{SPEA2}(i) = R(i) + D(i) \tag{2.6}$$

où la fonction $D(i)$ est définie comme suit :

$$D(i) = \frac{1}{\sigma_i^k + 2.0} \tag{2.7}$$

où σ_i^k correspond à la distance (dans l'espace des objectifs) entre l'individu i et son $k^{\text{ième}}$ voisin le plus proche. Plus précisément, pour chaque individu i, les distances à tous les individus j de l'ensemble $\mathbf{P} \cup \overline{\mathbf{P}}$ sont calculées et stockées dans une liste. Après avoir trié cette liste par ordre croissant, le $k^{\text{ième}}$ élément donne la distance désirée. Typiquement, $k = \sqrt{N + \overline{N}}$.

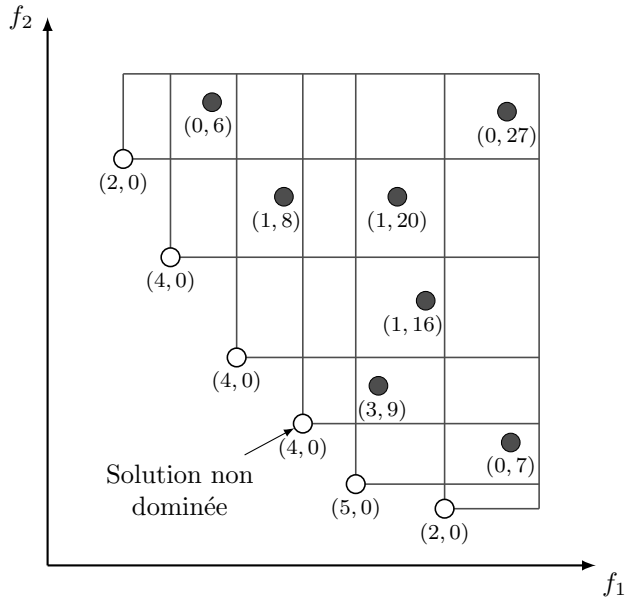


Figure 2.10 – Procédure d'affectation des scores (*strength*, *raw fitness*) de SPEA2.

2.5.2.3 Procédure de troncation de l'archive

Lorsque l'ensemble non dominé courant excède la taille limitée de l'archive ($\overline{N}$), une procédure de troncation est invoquée pour supprimer $\left|\overline{\mathbf{P}}_{t+1}\right| - \overline{N}$ individus. A chaque itération, l'individu i pour lequel $i \leq_d j$ pour tous les individus j de $\overline{P}_{t+1}$ est supprimé, avec :

$$i \leq_d j \qquad \forall\, 0 < k < \left|\overline{\mathbf{P}}_{t+1}\right| : \sigma_i^k = \sigma_j^k \;\; \vee \tag{2.8}$$

$$\exists\, 0 < k < \left|\overline{\mathbf{P}}_{t+1}\right| : \left[\left(\forall\, 0 < l < k : \sigma_i^l = \sigma_j^l\right) \;\wedge\; \sigma_i^k < \sigma_j^k\right] \tag{2.9}$$

où σ_i^k correspond à la distance entre l'individu i et son $k^{\text{ième}}$ voisin le plus proche dans $\overline{\mathbf{P}}_{t+1}$. En d'autres termes, l'individu qui a une distance minimale à un autre individu est choisi à chaque étape ; s'il y a plusieurs individus avec la même distance minimale le choix se fait à partir de la deuxième distance et ainsi de suite. La figure 2.11 illustre la technique de troncation de l'archive. Il est à noter enfin que cette procédure de troncation a une complexité moyenne en $O(M^2 \log M)$, où $M = N + \overline{N}$ (Zitzler et al., 2001).

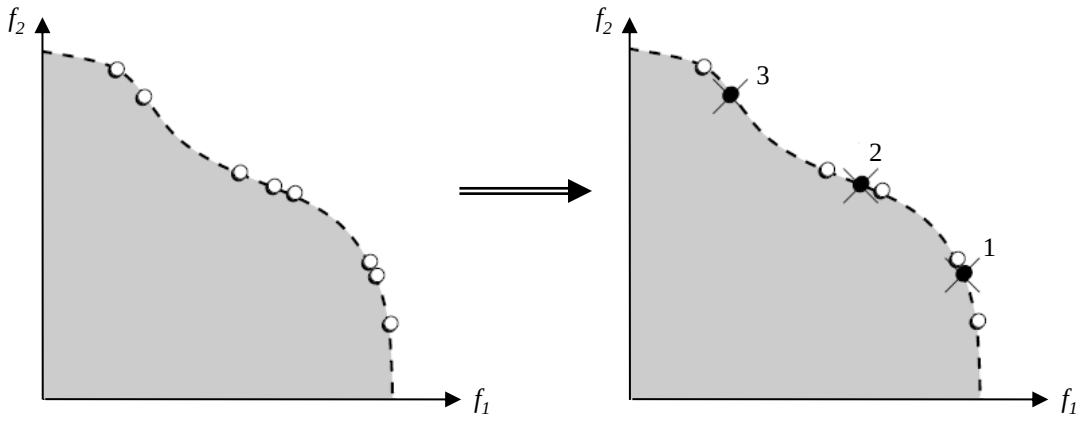


Figure 2.11 – Illustration de la procédure de troncation de l'archive utilisée dans l'algorithme SPEA2 (d'après (Zitzler et al., 2001)). La partie gauche de la figure donne le front Pareto. La partie droite illustre l'ordre dans lequel l'opérateur de troncation élimine des solutions. Notons que le front est défini ici en terme de maximisation et que la taille de l'archive est $\overline{N} = 5$.

2.6 Evaluation d'un ensemble de solutions

Les algorithmes multiobjectifs doivent retourner à la fin de recherche un ensemble de solutions (usuellement réalisables) pour le problème traité. D'après (Ehrgott et Gandibleux, 2007), cet ensemble de solutions définit des bornes supérieures sur le front Pareto du problème. Mais comment peut-on évaluer la qualité d'une approximation de l'ensemble Pareto optimal pour un MOP ? Plusieurs métriques pour la comparaison entre des ensembles de solutions ont été proposées dans la littérature (Zitzler et al., 2000). Elles évaluent la qualité et/ou la distribution d'un ensemble de solutions. Cependant, d'après (Ehrgott et Gandibleux, 2008), aucune métrique n'été adoptée universellement dans la littérature d'optimisation multiobjectif. Actuellement, il n'existe aucune métrique qui garantit la supériorité d'un algorithme multiobjectif par rapport un autre pour des problèmes particuliers. De ce fait, il est essentiel d'utiliser deux ou plusieurs métriques se focalisant sur divers aspects de la qualité des solutions pour comparer les algorithmes multiobjectifs. Nous présentons par la suite les deux métriques que nous avons utilisées dans les travaux réalisés durant cette thèse (présentés dans le chapitre 3). Ces métriques (complémentaires, d'après (Zitzler et Thiele, 1999)) considèrent que l'ensemble Pareto optimal (théorique) est inconnu.

2.6.1 Métrique d'hypervolume

La métrique d'hypervolume introduite dans (Zitzler et Thiele, 1999) prend en compte la qualité et la diversité des solutions. Soient $\mathcal{S}' = \{x_1, x_2, \ldots, x_k\} \subseteq \mathcal{S}$ un ensemble de k solutions et $Y = \{y_1, y_2, \ldots, y_k\} \subseteq \Lambda$ son image dans l'espace des objectifs, i.e., $y_i = F(x_i) = (f_1(x_i), f_2(x_i), \ldots, f_m(x_i)), i = 1, 2, \ldots, k$. La fonction $\mathcal{HV}(\mathcal{S}')$ calcule le volume de l'union des polytopes $p_1, p_2, \ldots, p_k$,

$$\mathcal{HV}(\mathcal{S}') = volume\left(\bigcup_{y_i \in Y} p_i\right) \tag{2.10}$$

où chaque p_i est formé par les intersections des hyperplans résultants de chaque point $y_i \in Y$ et d'un point de référence $\hat{y}$ ($\hat{y}$ peut être constitué, par exemple, par la plus mauvaise valeur pour chaque fonction objectif ; avec ce choix le meilleur ensemble de solutions est celui de plus grand score, i.e., la mesure d'hypervolume est définie en terme de maximisation) : pour chaque axe dans l'espace des objectifs, il y a un hyperplan perpendiculaire à l'axe et passe par le point y_i. Dans le cas d'un problème bi-objectifs, chaque p_i représente un rectangle défini par les deux points y_i et $\hat{y}$ comme les sommets de la diagonale. La figure 2.12 illustre ce cas.

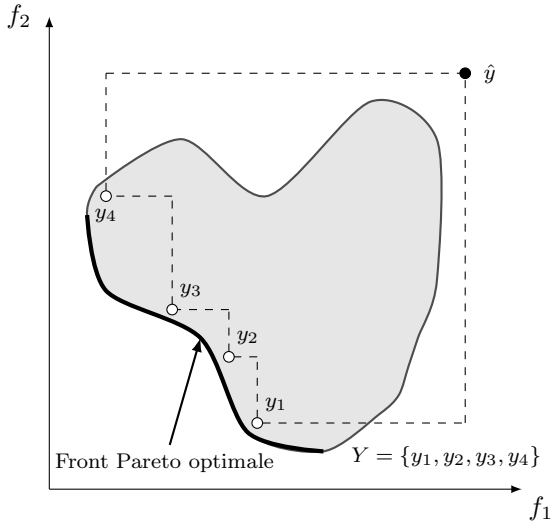


Figure 2.12 – L'hypervolume dans le cas d'un problème bi-objectifs de minimisation. La zone entourée par les lignes discontinues correspond à l'union des rectangles définis par les points $Y = \{y_1, y_2, y_3, y_4\}$ et le point de référence $\hat{y}$.

Cette mesure est très utilisée en raison de sa bonne propriété de monotonie par rapport à la qualité de chaque solution trouvée. Elle a aussi l'avantage que chaque algorithme peut être évalué indépendamment d'autres algorithmes. Cependant, son inconvénient réside dans le fait que les régions convexes peuvent être préférées aux régions concaves, avec le risque de rater certaines solutions (Zitzler et al., 2003).

2.6.2 Métrique de couverture

Cette mesure a été aussi introduite dans (Zitzler et Thiele, 1999). Pour deux ensembles de solutions $\mathcal{S}', \mathcal{S}'' \subseteq \mathcal{S}$, la fonction $\mathcal{C}$ donne à la paire ordonnée $(\mathcal{S}', \mathcal{S}'')$ une valeur dans l'intervalle $[0, 1]$:

$$\mathcal{C}(\mathcal{S}', \mathcal{S}'') = \frac{|\{s'' \in \mathcal{S}''; \exists s' \in \mathcal{S}' : s' \prec s''\}|}{|\mathcal{S}''|} \tag{2.11}$$

Le cas $\mathcal{C}(\mathcal{S}', \mathcal{S}'') = 1$ signifie que tous les points de $\mathcal{S}''$ sont dominés par des points dans $\mathcal{S}'$. Dans le cas contraire, $\mathcal{C}(\mathcal{S}', \mathcal{S}'') = 0$, aucun point de $\mathcal{S}''$ n'est dominé par des points de $\mathcal{S}'$. Cette mesure donne le pourcentage de points de $\mathcal{S}''$ qui sont dominés par au moins un point de $\mathcal{S}'$. Les deux valeurs $\mathcal{C}(\mathcal{S}', \mathcal{S}'')$ et $\mathcal{C}(\mathcal{S}'', \mathcal{S}')$ doivent être considérés, puisque $\mathcal{C}(\mathcal{S}', \mathcal{S}'')$ n'est pas nécessairement égal à $\mathcal{C}(\mathcal{S}'', \mathcal{S}')$, i.e., si $\mathcal{S}'$ domine $\mathcal{S}''$ alors $\mathcal{C}(\mathcal{S}', \mathcal{S}'') = 1$ et $\mathcal{C}(\mathcal{S}'', \mathcal{S}') = 0$.

Cette mesure permet de s'affranchir de l'inconvénient de la métrique d'hypervolume. Elle peut être utilisée pour démontrer que les résultats d'un algorithme dominent les résultats d'un autre algorithme. Cependant, elle peut retourner des scores peu concluants si les deux ensembles comparés se croisent.

2.7 Conclusion

Nous avons présenté dans ce chapitre les principales définitions et notions de base rencontrées dans le domaine de la résolution d'un MOP. Le concept de dominance et la notion d'optimalité, au sens de Pareto, ont été développés. Après un bref aperçu des approches de résolution pour traiter un MOP, un intérêt particulier est apporté aux MOGAs utilisant une approche Pareto. Lorsqu'ils sont associés à une stratégie

d'élitisme et couplés avec une technique de maintien de la diversité, les MOGAs utlisant une approche Pareto fournissent souvent des performances satisfaisantes. Nous avons vu aussi que la combinaison d'un MOGA avec une autre technique d'optimisation pourrait améliorer son efficacité sur les MOPs de grande taille. Parmi plusieurs MOGAs utilisant des approches Pareto dans la littérature, les algorithmes NSGA-II et SPEA2, que nous avons présentés, tiennent une place très importante. A la fin du chapitre, nous avons présentés deux métriques d'évaluation de la qualité d'une approximation du front Pareto : le métrique d'hypervolume et la mesure de couverture.

Nous avons conçu, durant cette thèse, un algorithme métaheuristique multiobjectif combinant un algorithme MOGA basé sur une approche Pareto élitiste avec un composant de recherche basé sur l'algorithme de PSO, que nous allons présenter dans le chapitre 3 suivant. NSGA-II et SPEA2 servirons à définir des variantes de cet algorithme hybride.

Chapitre 3

Un algorithme métaheuristique hybride pour l'optimisation multiobjectif

Dans ce chapitre nous présentons et évaluons un nouvel algorithme hybride proposé pour résoudre les problèmes d'optimisation multiobjectifs difficiles. Il combine deux algorithmes métaheuristiques à base de populations de solutions, un algorithme génétique et un algorithme d'optimisation par essaim de particules.

Les résultats préliminaires de l'algorithme hybride proposé ont fait l'objet d'une communication en anglais lors de la conférence CTAACS'13, à Skikda (Ben Ali et Melkemi, 2013).

Dans un premier temps, nous fournissons la motivation principale de notre travail. Ensuite, nous détaillons les éléments et le schéma de fonctionnement global de l'algorithme hybride proposé. Dans la troisième section, nous étudions deux variantes de cet algorithme. Dans cette section sont présentés les algorithmes implémentés, les paramètres de contrôle associés, le problème de test et les résultats expérimentaux obtenus. Nous terminons le chapitre par une conclusion.

3.1 Constat et Objectifs

Pour les MOPs qui sont caractérisés par des espaces de recherche larges et complexes, la présence de plus de deux objectifs contradictoires a un grand impact sur leur résolution par les MOGAs. Ces derniers rencontrent des situations de convergence lente tout au long du processus de résolution. Après l'exploitation d'une population initiale de solutions, les MOGAs restent coincés longtemps dans les sous-espaces des solutions qui présentent des compromis partiels par rapport aux objectifs du problème traité. Cette constatation oriente, depuis plus de quinze années, la recherche principalement vers le développement d'algorithmes métaheuristiques hybrides. Ce type d'algorithmes incorpore des techniques/composants de recherche ou des mécanismes de guidance dans des algorithmes métaheuristiques. Les algorithmes métaheuristiques hybrides sont reconnus comme des approches compétitives pour traiter les problèmes d'optimisation difficiles, et c'est dans les deux cas mono-objectif (voir section 1.5) et multiobjectif (voir section 2.4.4). Des revues de littérature et méta-analyses liées aux algorithmes métaheuristiques multiobjectifs hybrides peuvent être trouvés dans (Ehrgott et Gandibleux, 2008, Talbi, 2015).

Deux concepts fondamentaux dans le développement des algorithmes d'optimisation sont les notions de *diversification* (ou *exploration*) et d'*intensification* (ou *exploitation*) de la recherche (Blum et Roli, 2003). La diversification réfère à la capacité de l'algorithme à échantillonner convenablement différentes régions de l'espace des solutions, alors que l'intensification réfère à sa capacité à concentrer la recherche dans une région précise. Un algorithme d'optimisation efficace devrait assurer un bon équilibre entre ces deux capacités de recherche complémentaires. La forme d'hybridation d'algorithmes la plus répondue dans la littérature pour atteindre cet objectif consiste à incorporer des techniques de recherche locale dans des algorithmes à base de population de solutions. Dans ces algorithmes hybrides, connus dans la littérature sous le nom de "algorithmes mémétiques", les algorithmes à base de population accomplissent l'exploration de l'espace de recherche alors que les opérateurs de recherche locale sont responsables de l'amélioration des solutions découvertes. Pour les problèmes d'optimisation à (un ou) deux objectifs, ce type d'algorithmes hybrides a été appliqué avec

succès à des problèmes variés et a montré son efficacité (Knowles et Corne, 1999, Lacomme et al., 2006, Liefooghe et al., 2014, Tan et al., 2007, 2006a, b, T'kindt et al., 2002, Yapicioglu et al., 2007). Cependant, pour les problèmes à trois objectifs ou plus, il est difficile d'avoir avec cette forme d'hybridation un bon équilibre entre la diversification et l'intensification de la recherche.

Dans la vaste littérature des algorithmes évolutionnaires multiobjectifs (Coello et al., 2007), les MOGAs sont connus par leur capacité d'exploration globale de l'espace des solutions, mais souffrent des situations de convergence lente sur les problèmes à plus de deux objectifs de grande taille. Dans la littérature de l'algorithme de PSO (Davoud et Ellips, 2009, Poli et al., 2007), la version standard (mono-objectif) est connue par sa capacité d'exploiter rapidement une population initiale de solutions. Inspiré par l'idée d'hybridation d'algorithmes, nous proposons un algorithme combinant un algorithme MOGA avec un algorithme de PSO afin d'exploiter leurs capacités de recherche complémentaires et s'affranchir des situations de convergence lente des MOGAs sur les problèmes complexes à plus de deux objectifs. Pour résumer en une phrase, l'algorithme hybride proposé correspond à un MOGA qui applique périodiquement un opérateur de recherche basé sur l'algorithme de PSO pour améliorer une population archive sur la base d'une fonction d'utilité combinant tous les objectifs du problème traité. Cet algorithme hybride général, appelé MOGA-PSO, est détaillé dans la section suivante.

3.2 MOGA-PSO : un algorithme métaheuristique multiobjectif hybride

La pseudo-code de l'algorithme MOGA-PSO est décrit par l'algorithme 3.1.

Comme nous avons déjà mentionné, l'algorithme MOGA-PSO combine deux algorithmes métaheuristiques à base de population : un algorithme MOGA et un algorithme de PSO. L'algorithme MOGA utilise une approche de Pareto pour classer et comparer les solutions, tandis que l'algorithme de PSO utilise une méthode d'agrégation des objectifs pour transformer le problème multiobjectif en un problème à

objectif unique. En d'autres termes, notre algorithme hybride combine à la fois deux algorithmes métaheuristiques (MOGA et PSO) et deux paradigmes de résolution d'un MOP (une approche Pareto et une méthode d'agrégation). L'idée consiste à appliquer périodiquement un algorithme de PSO pour optimiser une fonction d'utilité sur la population archive d'un MOGA.

Algorithme 3.1 : Pseudo-code de l'algorithme hybride MOGA-PSO

```
1  – Initialiser et évaluer l'archive P_0 de taille N
2  – Créer la population Q_0 de taille N à partir de P_0 par application des
3    opérateurs de sélection, de croisement et de mutation
4  – Initialiser le compteur du temps t = 0
5  tant que t < T faire
6  |  si t mod I_g = 0 alors
7  |  |  – Initialiser l'essaim E_t de taille N_E à partir de P_t
8  |  |  – Appliquer un algorithme de PSO pour optimiser une fonction
9  |  |    d'utilité
10 |  |
   |  |                  f_PSO(s) = Σ_{k=1}^{m} λ_k · f_k(s), s ∈ Ω
   |  |
11 |  |    sur E_t en effectuant un nombre maximal d'itérations nI_PSO
12 |  sinon
13 |  |  – Initialiser l'essaim E_t à l'ensemble vide
14 |  fin
15 |  – Création de R_t = P_t ∪ Q_t ∪ E_t
16 |  – Appliquer une procédure de ranking basée sur une relation de
17 |    dominance au sens de Pareto pour ranger les individus de R_t, puis
18 |    former les niches
19 |  – Invoquer un opérateur de troncation utilisant les rangs et les
20 |    densités d'individus pour réduire la taille de R_t à N
21 |  – Former l'adaptation des individus de R_t
22 |  – Copier R_t dans P_{t+1}
23 |  – Sélection des parents à partir de P_{t+1} et création d'une nouvelle
24 |    population Q_{t+1} par application des opérateurs de croisement et de
25 |    mutation
26 fin
```

L'algorithme MOGA-PSO fonctionne de la façon suivante : dans un premier temps, une population de parents P_0 de taille N est généré aléatoirement (ou en utilisant une heuristique). Cette population est triée par une procédure de ranking Pareto. Puis,

une population d'enfants Q_0 est générée à partir de P_0 par application des opérateurs de sélection, de croisement et de mutation. La taille commune des populations P et Q est notée N. Comme l'élitisme est procédé par la comparaison des nouveaux individus avec les individus élites, la procédure est différente après la génération initiale. Après cette phase d'initialisation, l'algorithme entre dans une boucle d'optimisation pour un nombre maximal de générations T. Nous décrivons ci-après une génération t de l'algorithme MOGA-PSO.

Aux générations $t = 0, I_g, 2I_g, 3I_g, \ldots$ (i.e., des intervalles régulières de générations), un algorithme de PSO est appliqué sur un essaim de départ E_t de taille N_E initialisé à partir de l'archive courante P_t. La fitness de cet algorithme correspond à une fonction d'utilité combinant (linéairement) toutes les objectifs du problème traité. Cette opération est pour un nombre maximal d'itérations nI_{PSO}. Pour toutes les autres générations, E_t est vide. Une population combinée est ensuite créée $R_t = P_t \cup Q_t \cup E_t$. La taille de cette population est $N_R = 2N$ si $E_t = \phi$ et $N_R = 2N + N_E$ sinon. La deuxième phase consiste à classer tous les individus de R_t (ranking Pareto) et à estimer la densité des solutions autour de chaque individu (nichage). Une procédure de troncation est ensuite invoquée pour réduire la taille de R_t à N en se basant sur les rangs des individus et sur leurs densités. Après troncation, R_t constitue l'archive P_{t+1}. A partir de cette nouvelle archive, une nouvelle population d'enfants Q_{t+1} est produite par application des opérateurs de sélection, de croisement et de mutation. La procédure de MOGA-PSO est illustrée aussi dans la figure 3.1.

3.3 Résultats expérimentaux

Dans cette section, nous évaluons la performance de l'algorithme MOGA-PSO. Nous commençons par présenter, dans la section 3.3.1, les variantes que nous avons implémentées. Puis, dans la section 3.3.2, nous présentons le problème de test utilisé avec les détails d'implémentation associés. Nous fixons ensuite dans la section 3.3.3 les paramètres de contrôle, qui sont communs à tous les algorithmes testés. Dans la section 3.3.4 nous analysons les comportements des algorithmes hybrides testés et comparons les résultats obtenues avec les résultats des algorithmes standards.

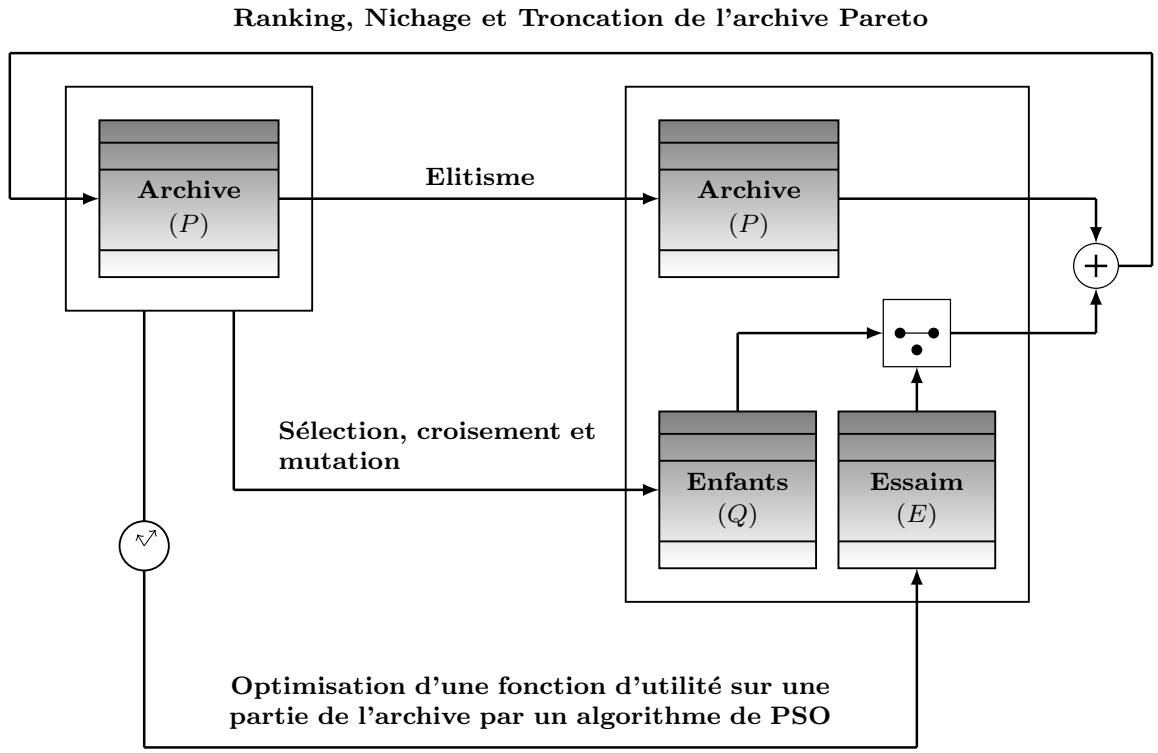


Figure 3.1 – Différentes populations manipulées par l'algorithme hybride proposé (MOGA-PSO).

Tous les algorithmes sont codés en C++ et tous les tests ont été effectués sur une machine standard dotée d'un processeur Intel Core i5 (2.5 GHz, 4 GB), sous Windows 10.

Afin d'obtenir des résultats significatifs, nous avons exécuté chaque algorithme 30 fois par problème de test. Une autre population initiale générée aléatoirement est prise à chaque exécution, et pour chaque problème de test tous les algorithmes commencent par les mêmes 30 populations initiales. En outre, nous avons procédé à des tests statistiques pour comparer les résultats obtenues. Pour tous les tests statistiques effectués, le niveau de signification de 0.05 (i.e., p-value < 0.05) a été considéré. Dans un premier temps, pour vérifier la normalité de la distribution des données (Gaussienne ou non), nous avons utilisé des tests statistiques de Kolmogorov-Smirnov. Tous les tests de Kolmogorov-Smirnov effectués dans cette étude ont rejeté l'hypothèse nulle (i.e., les résultats n'étaient pas normalement distribués), et par conséquent, des tests de Kruskal-Wallis ont été appliqués pour comparer les résultats des algorithmes.

3.3.1 Variantes implémentées

Deux variantes de l'algorithme MOGA-PSO ont été implémentées. La première variante utilise l'algorithme NSGA-II (voir section 2.5.1) pour définir l'opérateur génétique, tandis que la seconde utilise l'algorithme SPEA2 (voir section 2.5.2). Ces deux variantes sont notées respectivement NSGA-PSO et SPEA-PSO. Pour le composant de recherche basé sur l'algorithme de PSO, nous avons utilisé l'algorithme binaire de PSO, BPSO (présentée dans la section 1.4.2.4), puisque nous avons testé les algorithmes sur un problème à variables binaires (le problème du sac-à-dos binaire). Les détails d'implémentation liés à ce problème sont présentés dans le paragraphe 3.3.2.

Nous avons utilisés deux mesures complémentaires pour évaluer la qualité des approximations du front Pareto fournies par les algorithmes : la métrique d'hypervolume et la mesure de couverture, présentées respectivement dans les sections 2.6.1 et 2.6.2. La première mesure quantifie le volume de l'espace couvert par l'ensemble de solutions Pareto retourné par un algorithme, tandis que la seconde donne le pourcentage de solutions d'un ensemble qui sont dominées par au moins une solution d'un autre ensemble. Nous avons choisi de normaliser les valeurs de la première mesure, pour être variées dans l'intervalle $[0, 1]$, en divisant les éléments des vecteurs des objectifs par les valeurs d'un point idéal théorique (les valeurs de la seconde mesure sont par définition dans l'intervalle $[0, 1]$). Nous avons utilisé ces critères d'évaluation pour comparer uniquement NSGA-PSO avec NSGA et SPEA-PSO avec SPEA, car notre objectif est de démonter la supériorité de l'algorithme hybride (NSGA-PSO respectivement SPEA-PSO) sur l'algorithme de base (NSGA respectivement SPEA).

3.3.2 Problèmes de test

Nous avons choisi le problème du sac-à-dos multiobjectif en 0/1 (*Multiobjective* 0/1 *Knapsack Problem*) comme un problème de test. Ce problème d'optimisation combinatoire $\mathcal{NP}$-difficile a été fréquemment utilisé pour évaluer les performances de nouveaux algorithmes multiobjectifs (Sato et al., 2013, 2010, Zitzler et Thiele, 1998, 1999). Il est facile à comprendre et à formuler. Il permet aussi des expériences immuablement répétables et vérifiables.

3.3.2.1 Formulation

Le problème du sac-à-dos binaire consiste à remplir un sac d'une capacité limitée par une partie d'un ensemble donné d'objets ayant chacun un poids et une valeur (profit). L'objectif est de trouver un sous-ensemble d'objets qui maximise le total de profits dans le sous-ensemble sans dépasser la capacité maximale du sac (Silvano et Paolo, 1990).

Le problème mono-objectif peut être directement étendu au cas multiobjectif en permettant un nombre arbitraire de sacs. Formellement, le problème du sac-à-dos multiobjectif considéré dans cette étude peut être défini de la manière suivante : étant donné un ensemble de n objets et un ensemble de m sacs, avec

$$\begin{aligned} w_{j,i} &= \text{poids de l'objet } i \text{ dans le sac } j \\ p_{j,i} &= \text{profit de l'objet } i \text{ dans le sac } j \\ c_j &= \text{capacité du sac } j \end{aligned}$$

trouver un vecteur binaire $x = \{x_1, x_2, \ldots, x_n\} \in \{0,1\}^n$, avec

$$\forall j \in \{1, 2, \ldots, m\} : \sum_{i=1}^{n} w_{j,i} \cdot x_i \leq c_j \tag{3.1}$$

qui maximise $F(x) = \{f_1(x), f_2(x), \ldots, f_m(x)\}$, où

$$f_j(x) = \sum_{i=1}^{n} p_{j,i} \cdot x_i \tag{3.2}$$

où $x_i = 1$ signifie que l'objet i est sélectionné ($x_i = 0$ qu'il est écarté).

L'importance du problème du sac-à-dos multiobjectif réside dans le fait que plusieurs problèmes pratiques peuvent être formulés de la même façon. Par exemple, les problèmes de budgets d'investissement et les problèmes d'allocation de ressources dans un système informatique distribué (Chu et Beasley, 1998).

3.3.2.2 Jeux de tests

Afin d'obtenir des résultats plus robustes, nous avons utilisé 16 problèmes de test différents où le nombre d'objets (n) et le nombre de sacs (m) sont variés. Deux, trois, quatre et cinq objectifs ont été considérés en combinaison avec 250, 500, 750 et 1000 objets. Pour dénoter les problèmes, on utilise les valeurs de n et de m séparées par un tiret (n-m).

Comme utilisé dans (Zitzler et Thiele, 1999) et comme suggéré dans (Michalewicz et Arabas, 1994, Silvano et Paolo, 1990), des poids et des profits non corrélés ont été choisis, où $w_{j,i}$ et $p_{j,i}$ sont des nombres entiers choisis aléatoirement dans l'intervalle $[10, 100]$. La capacité de chaque sac $j = 1, 2, \ldots, m$ a été fixée à la moitié du total des poids dans le sac :

$$c_j = 0.5 \sum_{i=1}^{n} w_{j,i} \tag{3.3}$$

Avec ce type de sac, la solution optimale (du problème mono-objectif) tiendra la moitié des objets (Silvano et Paolo, 1990).

3.3.2.3 Implémentation

Une chaîne de nombres binaires s de longueur correspondant au nombre d'objets (n) est utilisée pour coder la solution $x \in \{0, 1\}^n$. Puisque plusieurs configurations conduisent à des solutions non réalisables violant une ou plusieurs contraintes de capacité, des mécanismes ont été utilisés pour gérer celles-ci.

Dans tous les algorithmes d'étude, dans le cas d'une comparaison multiobjectif de solutions (principalement au niveau de la procédure de ranking), la gestion des contraintes est intégrée dans la procédure de test de dominance Pareto : un individu a domine (meilleur que) un autre individu b s'il viole moins de contraintes ou si les deux individus violent le même nombre de contraintes et le vecteur des objectifs $F(a)$ est partiellement supérieur au vecteur des objectifs $F(b)$.

Dans le cas mono-objectif, au niveau de l'algorithme de PSO qui agrège le vecteur des objectifs en une seule fonction d'utilité, une technique basée sur le principe de pénalisation est utilisée pour évaluer les solutions non réalisables. Il s'agit de transformer

le problème sous contraintes en un problème sans contrainte en associant à chaque fonction objectif une pénalité dès que la contrainte de capacité correspondante est violée[1]. Chaque fonction objectif f_j $(j = 1, 2, \ldots m)$ est alors remplacée par la fonction suivante :

$$f'_j(x) = f_j(x) - Pen_j(x) \tag{3.4}$$

où Pen_j est la fonction de pénalité associée au sac j, donnée par :

$$Pen_j(x) = \rho_j \times (\sum_{i=1}^{n} w_{j,i} - c_j) \tag{3.5}$$

où le paramètre ρ_j correspond au taux profit/poids maximal associé au sac j, donné par :

$$\rho_j = \max_{i=1,2,\ldots,n} \left\{ \frac{p_{j,i}}{w_{j,i}} \right\} \tag{3.6}$$

La valeur de la fonction Pen_j sera nulle pour une solution respectant la contrainte de capacité du sac j.

La fitness (i.e., la fonction d'utilité à optimiser) de l'algorithme de PSO est définie par (où tous les poids sont égaux) :

$$f_{PSO}(x) = \sum_{j=1}^{m} f'_j(x) \tag{3.7}$$

3.3.3 Paramètres de contrôle

Dans tous nos algorithmes, la sélection de parents géniteurs est basée sur un tournoi binaire combiné avec les valeurs d'adaptation (NSGA et SPEA forment l'adaptation des individus selon les formules 2.6 et 2.6 respectivement). Un opérateur de croisement à deux points a été utilisé avec une probabilité $Pc = 1.0$. Cet opérateur garde la plus grande partie du meilleur parent (comparaison sur la base des valeurs d'adaptation). Une mutation binaire (*Bit-Flip*) a été appliquée avec une portabilité $Pm_g = 1/n$. Le nombre de générations (T) a été fixé à 250 pour les deux algorithmes standards NSGA et SPEA. Pour que la comparaison soit équitable, nous avons attribué le même

[1]Il s'agit d'une extension directe de l'approche mono-objectif suggérée dans (Michalewicz et Arabas, 1994)

nombre d'évaluations complètes de solution (i.e., évaluations de toutes les fonctions objectifs) que pour NSGA et SPEA aux deux algorithmes hybrides NSGA-PSO et SPEA-PSO. Ainsi $250 \times N$ évaluations complètes de solution sont effectuées dans chaque exécution de chaque algorithme, où N est la taille commune de la population et de l'archive. Suivant les directives fournies dans (Zitzler et Thiele, 1999), cette taille (N) et la taille de l'essaim (N_E) de l'algorithme PSO ont été choisies en fonction de la complexité du problème à traiter, comme illustré dans le tableau 3.I : plus le nombre d'objets et le nombre de sacs augmentent, plus ces tailles sont grandes. De même, le nombre d'itérations (nI_{PSO}) effectué par l'algorithme de PSO a été fixé aussi en fonction de la complexité du problème (voir tableau 3.I). Les deux paramètres c_1 et c_2 de l'algorithme PSO, qui sont utilisés dans la formule de mise à jour de vitesse 1.3, ont été fixés les deux à 1.49618, comme suggéré dans (Eberhart et Shi, 2000). Pour ce même algorithme, aucune limitation n'a été imposée sur les cordonnées de vitesse, i.e., $V_{max} = \infty$, ce qui permet un taux de mutation plus faible. Enfin, le paramètre I_g, qui détermine l'intervalle de générations pour faire intervenir l'opérateur de recherche basé sur l'algorithme PSO, a été fixé dans toutes les expérimentations réalisées à 100.

Tableau 3.I – Paramètres de contrôle qui ont été choisis en fonction de la complexité du problème : la taille commune de la population et de l'archive (N), la taille de l'essaim (N_E), et le nombre maximal d'itérations de l'algorithme PSO (nI_{PSO}).

Nombre de sacs	Paramètres	Nombre d'objets			
		250	500	750	1000
2	N	100	150	200	250
	N_E	20	30	40	50
3	N	150	200	250	300
	N_E	30	40	50	60
4	N	200	250	300	350
	N_E	40	50	60	70
5	N	250	300	350	400
	N_E	50	60	70	80
2, 3, 4, et 5	nI_{PSO}	50	100	150	200

3.3.4 Résultats et comparaison de performances

Les tableaux 3.II et 3.III présentent les résultats expérimentaux moyens et les résultats de la comparaison des algorithmes : NSGA opposé à NSGA-PSO et SPEA opposé à SPEA-PSO, respectivement. Ils contiennent les valeurs moyennes de $\mathcal{VH}$ et de $\mathcal{C}$. Dans ces tableaux, pour chaque problème de test, les meilleurs résultats sont indiqués en gras. Les différences statistiquement significatives entre les résultats des algorithmes comparés sont notées par le signe "+", tandis que la non-significativité est notée par le signe "•".

Pour chaque algorithme (respectivement paire ordonnée d'algorithmes), il y a un échantillon de 30 valeurs de $\mathcal{VH}$ (respectivement de $\mathcal{C}$) par problème de test conformément aux 30 exécutions effectuées. Des boites à moustaches[2] sont utilisées pour visualiser les distributions des échantillons de $\mathcal{VH}$ (voir figure 3.2) et les distributions des échantillons de $\mathcal{C}$ (voir figures 3.3 et 3.4).

Les résultats expérimentaux présentés dans les tableaux 3.II et 3.III montrent que pour les problèmes de test de 500, 750 et 1000 objets les performances des algorithmes hybrides dépassent celles des algorithmes standards en termes des valeurs moyennes de $\mathcal{VH}$ et de $\mathcal{C}$. Pour ces problèmes, les différences entre les valeurs de $\mathcal{VH}$ et entre les valeurs de $\mathcal{C}$ pour chaque paire d'algorithmes comparés sont statistiquement significatives dans tous les cas, sauf sur les problèmes 500-2, 500-3 et 500-4 en termes de $\mathcal{VH}$ et le problème 500-2 en termes de $\mathcal{C}$. Pour les problèmes de 250 objets la supériorité des algorithmes hybrides n'est pas claire, sauf dans quelques cas en termes de la mesure $\mathcal{C}$: NSGA-PSO est plus performant que NSGA sur les problèmes 250-4 et 250-5, et SPEA-PSO est plus performant que SPEA sur le problème 250-4.

En ce qui concerne la distribution de $\mathcal{VH}$, comme nous pouvons le voir dans la figure 3.2, les médianes des valeurs des algorithmes hybrides sont supérieures aux médianes des valeurs liées aux algorithmes standards par plus de 50 quartiles sur la plupart des problèmes de 750 et 1000 objets. Pour la plupart des problèmes de 250 et 500 objets, les distributions de $\mathcal{VH}$ sont proches ; cependant, comme nous verrons

[2]Une boite à moustache résume 50% de données. Leur bordures supérieure et inférieure (ou gauche et droite, si l'orientation de la boité est horizontale) sont les $25^{ième}$ et $75^{\text{ième}}$ quartiles, tandis que la ligne (rouge) à l'intérieur de la boite représente la médiane. Les segments discontinus informent sur l'étendue et la forme de la distribution, et les signes "+" représentent les valeurs extrêmes.

Tableau 3.II – Comparaison entre les deux algorithmes NSGA et NSGA-PSO (Convergence). Valeurs moyennes de la mesure $\mathcal{VH}$ pour chaque algorithme et valeurs moyennes de la mesure $\mathcal{C}$ pour les deux paires ordonnées (NSGA, NSGA-PSO) et (NSGA-PSO, NSGA).

Nombre d'objets (n)	Nombre de sacs (m)	$\mathcal{VH}$(nsga)	$\mathcal{VH}$(nsga-pso)	Test $\mathcal{VH}$	$\mathcal{C}$(nsga, nsga-pso)	$\mathcal{C}$(nsga-pso, nsga)	Test $\mathcal{C}$
250	2	**0.6583**	0.6548	•	**0.54**	0.27	+
	3	**0.4843**	0.4804	•	0.28	**0.41**	•
	4	**0.3920**	0.3885	•	0.21	**0.33**	+
	5	0.2395	**0.2396**	•	0.06	**0.34**	+
500	2	0.6392	**0.6400**	•	0.27	**0.43**	•
	3	**0.4753**	0.4744	•	0.25	**0.52**	+
	4	0.3129	**0.3158**	•	0.07	**0.67**	+
	5	0.2154	**0.2245**	+	0.	**0.81**	+
750	2	0.6704	**0.6801**	+	0.04	**0.80**	+
	3	0.4236	**0.4354**	+	0.01	**0.79**	+
	4	0.2932	**0.3086**	+	0.	**0.91**	+
	5	0.1804	**0.1948**	+	0.	**0.95**	+
1000	2	0.6232	**0.6377**	+	0.01	**0.91**	+
	3	0.4343	**0.4530**	+	0.	**0.94**	+
	4	0.2757	**0.2960**	+	0.01	**0.94**	+
	5	0.1845	**0.2021**	+	0.	**0.96**	+
moy.		0.4064	**0.4141**		0.11	**0.69**	

Tableau 3.III – Comparaison entre les deux algorithmes SPEA et SPEA-PSO (Convergence). Valeurs moyennes de la mesure $\mathcal{VH}$ pour chaque algorithme et valeurs moyennes de la mesure $\mathcal{C}$ pour les deux paires ordonnées (SPEA, SPEA-PSO) et (SPEA-PSO, SPEA).

Nombre d'objets (n)	Nombre de sacs (m)	$\mathcal{VH}$(spea)	$\mathcal{VH}$(spea-pso)	Test $\mathcal{VH}$	$\mathcal{C}$(spea, spea-pso)	$\mathcal{C}$(spea-pso, spea)	Test $\mathcal{C}$
250	2	0.6461	**0.6470**	•	**0.31**	0.30	•
	3	**0.4640**	0.4610	•	**0.27**	0.22	•
	4	**0.3757**	0.3714	•	0.20	**0.23**	+
	5	**0.2322**	0.2303	•	0.13	**0.21**	•
500	2	0.6343	**0.6358**	•	0.30	**0.51**	•
	3	**0.4664**	0.4658	•	0.23	**0.46**	+
	4	0.3070	**0.3084**	•	0.05	**0.56**	+
	5	0.2145	**0.2203**	+	0.01	**0.66**	+
750	2	0.6652	**0.6770**	+	0.05	**0.79**	+
	3	0.4188	**0.4300**	+	0.	**0.81**	+
	4	0.2880	**0.3018**	+	0.	**0.88**	+
	5	0.1795	**0.1918**	+	0.	**0.90**	+
1000	2	0.6184	**0.6358**	+	0.	**0.98**	+
	3	0.4300	**0.4498**	+	0.	**0.89**	+
	4	0.2732	**0.2936**	+	0.	**0.96**	+
	5	0.1830	**0.2011**	+	0.	**0.95**	+
moy.		0.3998	**0.4076**		0.10	**0.64**	

par la suite, la mesure $\mathcal{C}$ indique une supériorité claire des algorithmes hybrides sur les algorithmes standards dans la plupart de cas.

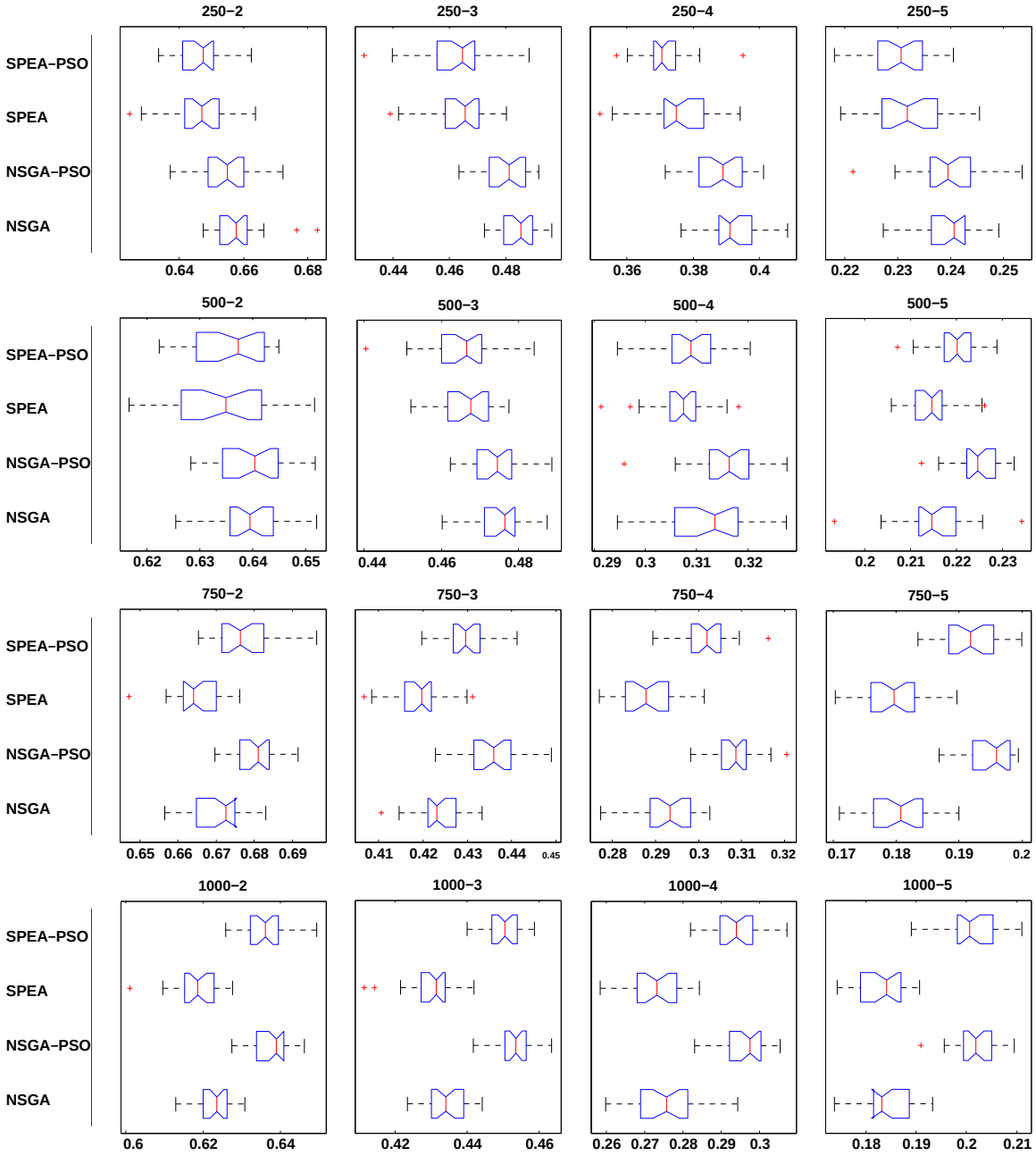


Figure 3.2 – Distribution des valeurs de $\mathcal{VH}$ de tous les algorithmes sur toutes les instances.

Quant à la distribution des valeurs de la mesure $\mathcal{C}$, comme nous pouvons le constater dans les figures 3.3 et 3.4, les algorithmes hybrides couvrent en moyenne plus de

90% des fronts Pareto calculés par les algorithmes standards sur les problèmes de 1000 objets, plus de 80% sur les problèmes de 750 objets, et plus de 50% sur les problèmes de 500 objets. Par contre, les algorithmes standards couvrent en moyenne moins de 20% des fronts Pareto retournés par les algorithmes hybrides sur les problèmes de 500 objets. Pour les instances de 750 et 1000 objets, les algorithmes standards ne couvrent aucune solution des fronts Pareto des algorithmes hybrides dans plus de 99% d'exécutions. Pour les instances de 250 objets, les fronts Pareto calculés par chaque paire d'algorithmes comparés se croisent. Un aspect de convergence important réside dans le fait que plus la taille du problème est grande (en termes de nombre d'objets et de nombre de sacs), plus les valeurs de quartiles de $\mathcal{C}(\text{nsga-pso}, \text{nsga})$ et de $\mathcal{C}(\text{spea-pso}, \text{spea})$ sont grandes (vers 1) et proches. Par contre plus la complexité du problème augmente, plus les valeurs de quartiles de $\mathcal{C}(\text{nsga}, \text{nsga-pso})$ et de $\mathcal{C}(\text{spea}, \text{spea-pso})$ sont petites (vers 0) et proches. Ceci montre la domination et la robustesse des algorithmes hybrides. La figure 3.5 illustre aussi la supériorité des algorithmes hybrides sur les algorithmes standards, dans laquelle les fronts Pareto calculés pour les problèmes à deux objectifs sont tracés.

Afin de comprendre l'impact de l'intervention de l'opérateur de recherche basé sur l'algorithme de PSO sur la convergence des algorithmes hybrides, nous avons tracé les courbes de convergence des valeurs de $\mathcal{VH}$ de tous les algorithmes pendant des exécutions typiques de $500 \times N$ évaluations complètes de solutions. Les courbes de convergence liées aux problèmes de 500, 750 et 1000 sont présentées dans les figures 3.6, 3.7 et 3.8 respectivement (Les courbes de convergence liées aux problèmes de 250 objets ne sont pas considérées dans ces figures car tous les algorithmes présentent des comportements similaires). Comme nous pouvons le voir dans ces figures, les algorithmes hybrides commencent par des archives Pareto ayant des valeurs de $\mathcal{VH}$ plus grandes que les archives de départ des algorithmes standards. Ceci montre que l'application de l'algorithme de PSO au début de résolution est très pratique. Les autres interventions de l'algorithme de PSO sont après 100, 200 et 300 générations. Ces interventions accélèrent généralement la convergence des algorithmes hybrides ; voir les petits sauts dans la plupart des courbes des algorithmes hybrides notamment après 100 et 200 générations.

Du point de vue du coût en temps d'exécution, le tableau 3.IV reporte les temps d'exécution moyens en secondes de tous les algorithmes sur chaque problème de test. Comme nous pouvons l'observer, bien que tous les algorithmes fassent le même nombre d'évaluations de solutions, dans tous les cas les algorithmes hybrides prennent en moyenne moins de temps que les algorithmes standards. Les différences entre les temps d'exécution des algorithmes comparés sont statistiquement significatives dans tous les cas (pour cette raison les résultats des tests statistiques ne sont pas inclus dans le tableau). Pour comprendre les raisons derrière cette réduction du temps d'exécution, nous avons mesuré le temps pris par la procédure de gestion de l'archive (ranking, nichage et troncation) de chaque algorithme. Les pourcentages de temps moyens des 30 exécutions pris par cette procédure dans chaque cas sont donnés aussi dans le tableau 3.IV. Comme nous pouvons le voir, la procédure de gestion de l'archive pour les algorithmes hybrides prend toujours moins de temps que pour les algorithmes standards. Cela provient principalement à la qualité et à la densité des solutions de l'archive tout au long du processus de recherche. Ainsi, les solutions manipulées par les algorithmes hybrides réduisent la complexité de la procédure de gestion de l'archive. Une question ouverte est de savoir les caractéristiques précises de ces solutions en termes de rang Pareto et de densité de la population autour d'elles en comparaison avec les solutions manipulées par les algorithmes standards. Cette question fera l'objet d'une prochaine étape de recherche.

En conclusion, les résultats expérimentaux obtenus montrent que nos algorithmes hybrides sont plus performants que les algorithmes standards. En fait, les fronts Pareto calculés par les algorithmes hybrides sont significativement meilleurs en termes des métriques $\mathcal{VH}$ et $\mathcal{C}$ que ceux obtenus par les algorithmes standards. En outre, les algorithmes hybrides consomment moins de temps de calcul.

3.4 Conclusion

Dans ce chapitre nous avons proposé un nouvel algorithme multiobjectif hybride, MOGA-PSO. L'algorithme MOGA-PSO est une hybridation d'un algorithme génétique multiobjectif utilisant une approche Pareto et élitiste (MOGA, pour se référer

Tableau 3.IV – Comparaisons entre les paires d'algorithmes NSGA vs. NSGA-PSO et SPEA vs. SPEA-PSO (Temps de calcul). La colonne intitulée "t (s)" contient les temps d'exécution moyens (en secondes), et la colonne intitulée "rtr (%)" donne les pourcentages de temps moyens pris par la procédure de ranking-nichage-troncation de l'archive Pareto.

Nb.	Nb. de	NSGA		NSGA-PSO		SPEA		SPEA-PSO	
d'objets	sacs	t (s)	rtr (%)	t (s)	rtr (%)	t (s)	rtr (%)	t (s)	rtr (%)
250	2	3.37	58	**3.14**	55	4.69	70	**4.31**	67
	3	5.34	61	**4.98**	56	9.57	78	**8.92**	75
	4	7.46	61	**7.00**	58	18.93	85	**17.08**	83
	5	10.44	63	**9.59**	59	41.69	91	**36.98**	90
500	2	9.80	61	**9.08**	53	10.77	64	**9.92**	58
	3	13.32	59	**12.29**	52	17.74	69	**16.68**	63
	4	16.71	59	**16.68**	50	35.52	80	**28.19**	72
	5	20.97	59	**19.44**	51	69.27	88	**52.50**	82
750	2	20.28	62	**17.77**	54	20.21	62	**17.65**	55
	3	25.33	59	**22.75**	51	32.02	68	**28.40**	61
	4	30.93	58	**27.98**	50	65.73	79	**45.61**	70
	5	37.63	58	**33.91**	49	145.88	89	**86.15**	80
1000	2	33.88	63	**28.43**	52	33.17	62	**27.13**	51
	3	40.39	59	**34.91**	48	49.39	66	**40.16**	56
	4	44.08	57	**37.73**	47	76.75	75	**47.31**	58
	5	55.77	56	**48.38**	45	218.58	89	**110.64**	76
moy.		23.48	60	**20.88**	52	53.12	76	**36.10**	69

un algorithme général) avec un algorithme d'optimisation par essaims de particule (PSO). Alors que les MOGAs sont directement prévus de fonctionner sur les problèmes multiobjectifs, l'algorithme PSO standard est adapté aux problèmes mono-objectifs. Pour cause, l'algorithme de PSO est capable d'exploiter rapidement un ensemble de solutions dans un contexte mono-objectif. Pour incorporer cet algorithme à base de population dans un MOGA comme un opérateur de recherche supplémentaire, nous avons choisi de procéder à une agrégation de fonctions objectifs pour combiner tous les objectifs en une seule fonction scalaire. Cet opérateur basé sur l'algorithme de PSO est appliqué à des intervalles de temps réguliers pour exploiter l'information de la population archive d'un MOGA, et ainsi accélérer la convergence.

Pour évaluer les performances de l'algorithme hybride proposé, nous avons défini deux variantes en choisissant NSGA-II et SPEA2, deux algorithmes plus répondus dans la littérature, pour définir sa partie génétique, et l'algorithme binaire de PSO. Nous avons choisi le problème du sac à dos multiobjectif comme un problème de test. Pour la comparaison des algorithmes testés, nous avons utilisé l'indicateur hypervolume $\mathcal{VH}$ et la mesure de couverture $\mathcal{C}$, qui permettent d'évaluer la qualité des fronts Pareto obtenus et quantifier le gain apporté par un algorithme sur un autre. Sur la base des valeurs de ces métriques, les algorithmes hybrides sont significativement plus performants que les algorithmes standards. Ils se sont révélés être de loin les plus performants. Plus la taille du problème traité augmente, en termes de nombre de variables et/ou en termes de nombre d'objectifs, plus les algorithmes hybrides sont performants.

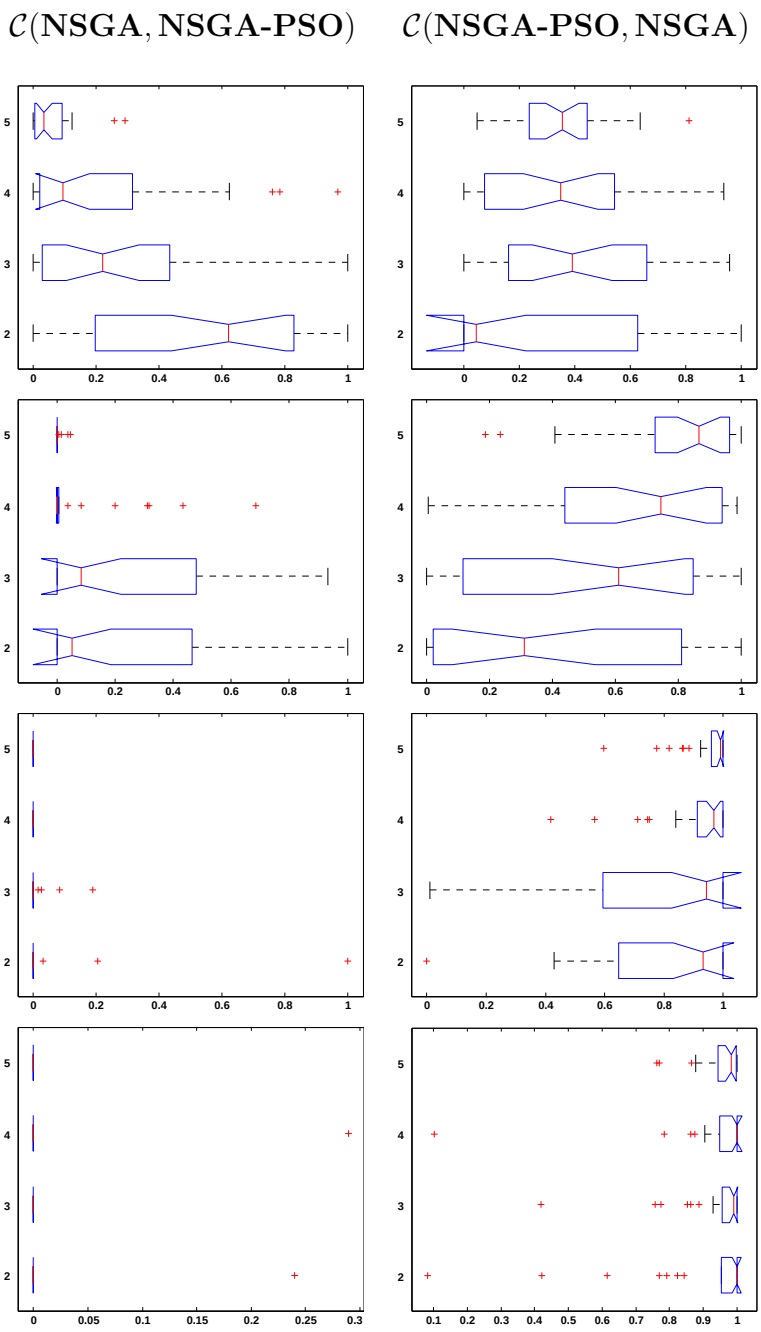


Figure 3.3 – Boîtes à moustaches des valeurs de $\mathcal{C}$. Chaque rectangle contient quatre boîtes à moustaches représentant la distribution des valeurs de $\mathcal{C}$ pour une paire ordonnée d'algorithmes ; les rectangles de gauche sont liés à (NSGA, NSGA-PSO) tandis que les rectangles de droite sont liés à (NSGA-PSO, NSGA) ; les boîtes à moustaches de chaque rectangle sont, de plus bas au plus haut, pour les nombres de sacs 2, 3, 4 et 5 respectivement ; les quatre lignes de rectangles, sont respectivement, du plus haut au plus bas, pour les instances de 250, 500, 750 et 1000 objets.

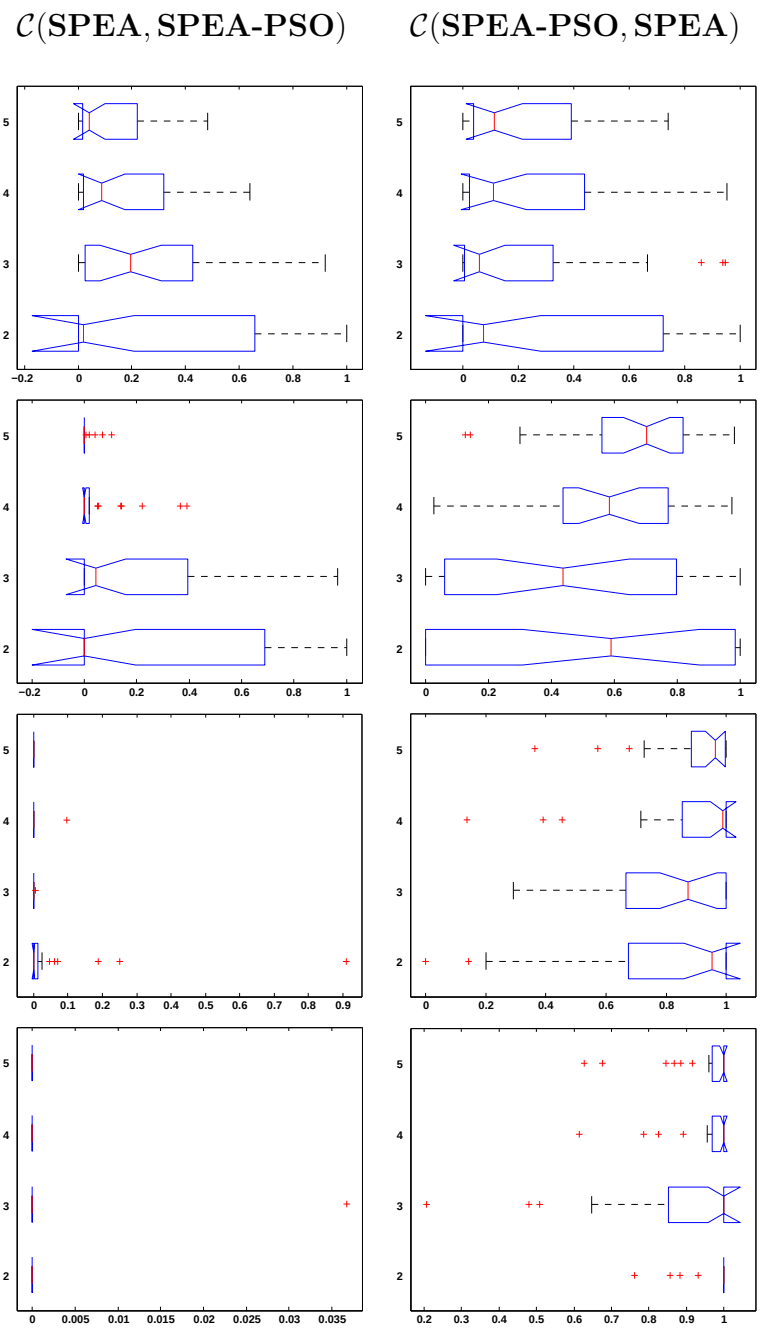


Figure 3.4 – Boîtes à moustaches des valeurs de $\mathcal{C}$. Chaque rectangle contient quatre boîtes à moustaches représentant la distribution des valeurs de $\mathcal{C}$ pour une paire ordonnée d'algorithmes ; les rectangles de gauche sont liés à (SPEA, SPEA-PSO) tandis que les rectangles de droite sont liés à (SPEA-PSO, SPEA) ; les boîtes à moustaches de chaque rectangle sont, de plus bas au plus haut, pour les nombres de sacs 2, 3, 4 et 5 respectivement ; les quatre lignes de rectangles, sont respectivement, du plus haut au plus bas, pour les instances de 250, 500, 750 et 1000 objets.

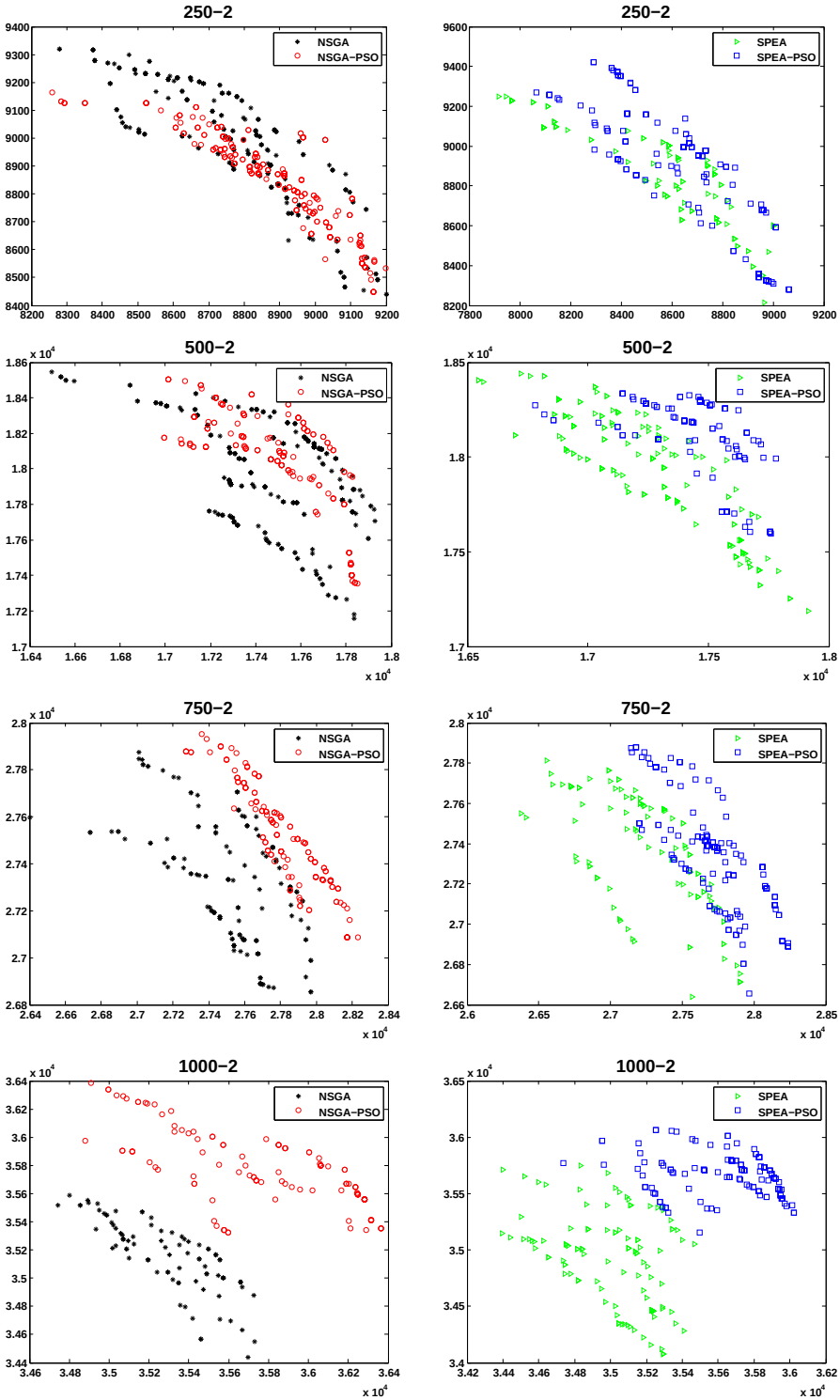


Figure 3.5 – Fronts Pareto pour deux sacs : ici, toutes les points des fronts Pareto retournés de cinq premières exécutions sont tracés.

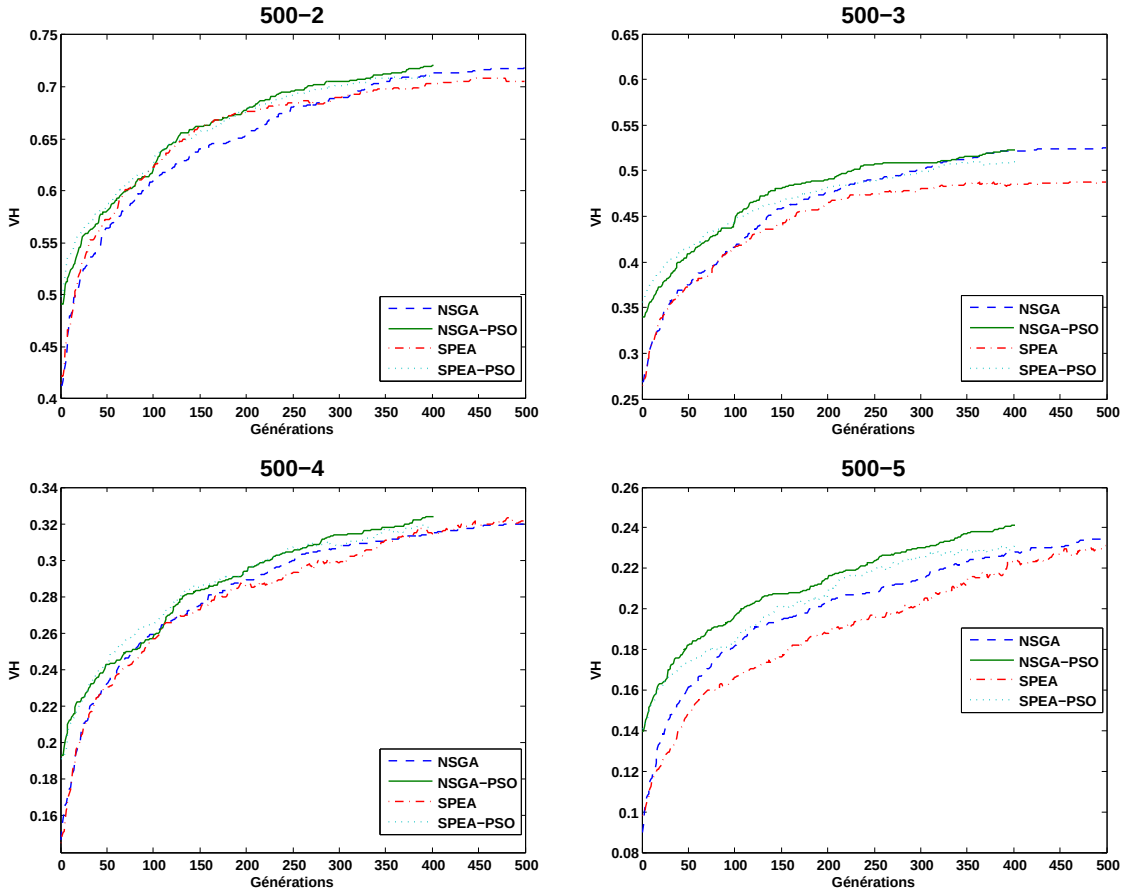


Figure 3.6 – Courbes de convergence des valeurs de $\mathcal{V}H$ en fonction de nombre de générations pour une exécutuoon typique sur les instances de 500 objets.

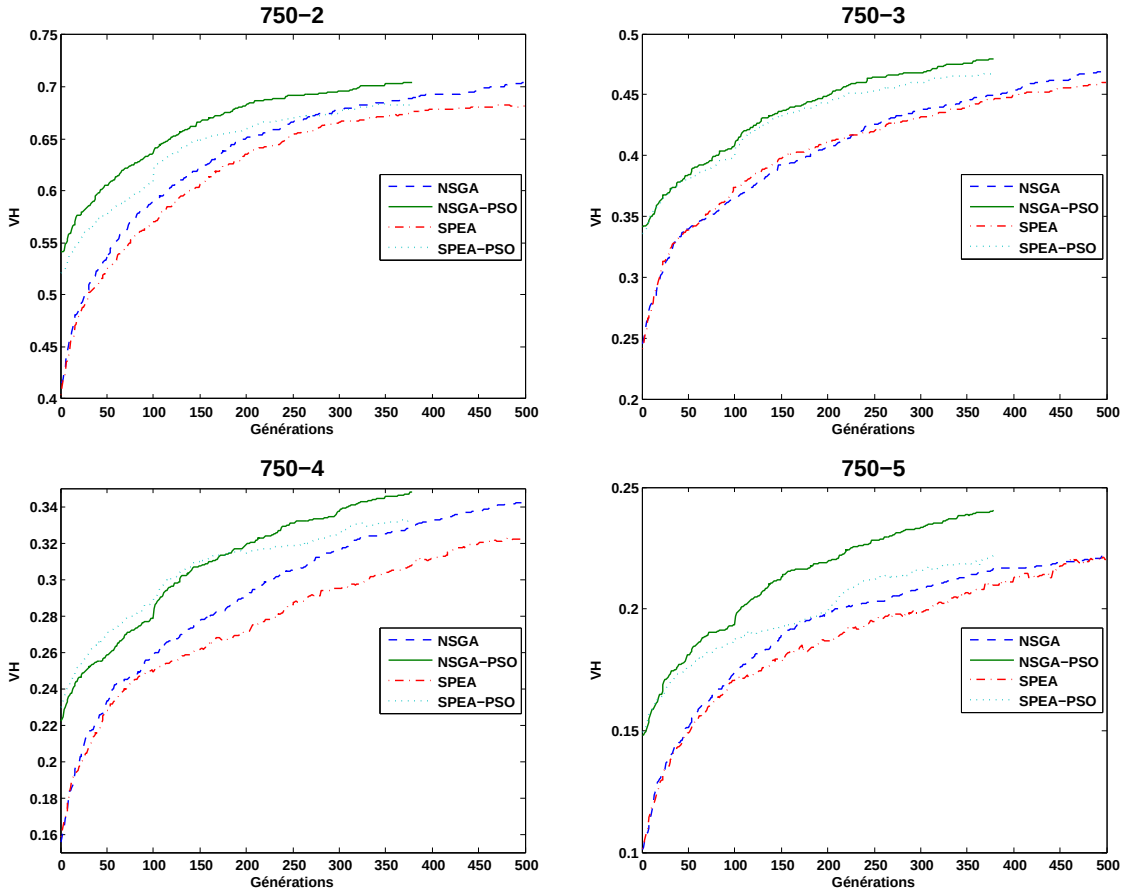


Figure 3.7 – Courbes de convergence des valeurs de $\mathcal{V}H$ en fonction de nombre de générations pour une exécutuoon typique sur les instances de 750 objets.

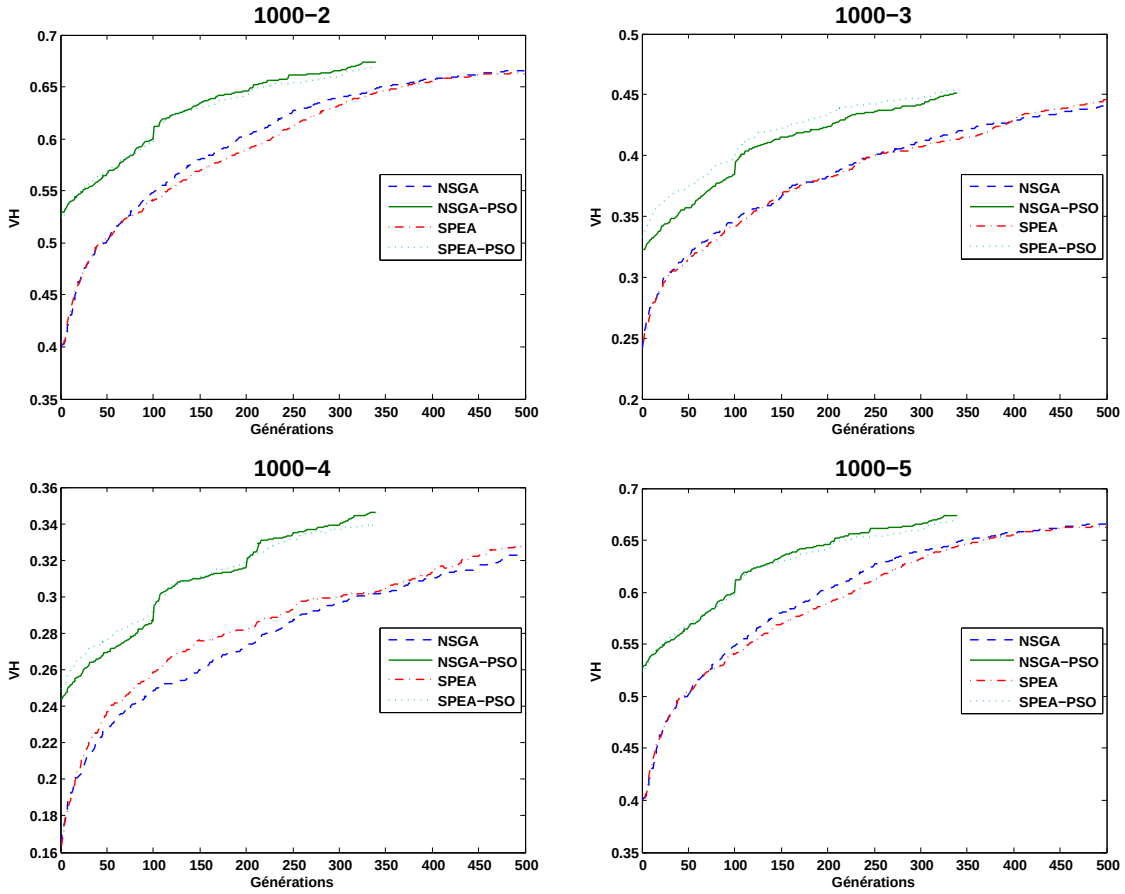


Figure 3.8 – Courbes de convergence des valeurs de $\mathcal{V}H$ en fonction de nombre de générations pour une exécutuoon typique sur les instances de 1000 objets.

Chapitre 4

Problème d'assemblage de fragments d'ADN : état de l'art

Dans le présent chapitre, nous abordons le problème d'assemblage de fragments d'ADN (ou *DNA Fragment Assembly Problem* : DNA FAP), qui se pose au niveau du séquençage des génomes. Nous commençons, dans la première section, par présenter brièvement la bio-informatique des séquences. Puis, dans la deuxième partie, nous nous focalisons sur le problème de DNA FAB. Nous portons ensuite, dans la troisième section, notre attention sur trois types de graphes et leur lien à l'assemblage de génomes. Dans la quatrième section, nous passons en revue les travaux portant sur les algorithmes heuristiques et métaheuristiques pour résoudre le problème de DNA FAB. Enfin, dans la cinquième section, nous présentons les benchmarks (jeux de données) les plus utilisés dans la littérature pour tester les algorithmes d'assemblage.

4.1 Bio-informatique

La bio-informatique est un domaine de recherche très actif depuis plusieurs dizaines d'années (Hesper et Hogeweg, 1970, Hogeweg, 2011, Moody, 2004). Ce domaine interdisciplinaire s'appuie essentiellement sur la biologie, l'informatique, les statistiques et les mathématiques. Elle été mis en place dans le but de résoudre les problèmes scientifiques posés par la biologie. Il s'agit en fait de rassembler, comparer, analyser ou modéliser les données biologiques caractérisées par leurs très grandes tailles.

Le rôle de l'informatique est de développer les méthodes et les outils logiciels nécessaires au stockage, à l'analyse, à l'interprétation et à la modélisation de l'information biologique.

4.1.1 Structure de L'ADN

Le matériel génétique permet le développement, le fonctionnement et la reproduction des êtres vivants. Cette substance, appelée *génome*, est stockée au niveau de toutes les cellules vivantes sous la forme d'une molécule, appelée l'ADN (Acide désoxyribonucléique). L'ADN peut être codé par des chaînes de caractères, ce qui permet ainsi le traitement automatique de l'information biologique.

L'ADN est un acide nucléique formé de deux brins polymères enroulés l'un autour de l'autre pour former une double hélice (voir figure 4.1). Les deux brins sont appelés polynucléotides et les éléments simples qui les constituent sont appelés nucléotides. Chaque nucléotide est composé d'une base nucléique (ou base azotée) et d'un sucre, le désoxyribose, attaché à un groupe phosphate. Les quatre bases nucléiques (par la suite *bases*) sont l'Adénine (A), la Cytosine (C), la Guanine (G), et la Thymine (T). Chaque brin d'ADN est une séquence de nucléotides qui sont reliés entre eux par des liaisons covalentes entre le sucre d'un nucléotide et le groupe phosphate du nucléotide suivant. Cette succession de nucléotides forme une chaîne alternant des sucres et des phosphates, avec des bases liées chacune à un sucre. De plus, les bases d'un brin sont attachées aux bases de l'autre brin à travers des liaisons hydrogènes. Les deux brins sont complémentaires due à l'appariement des bases A avec T et G avec C (voir figure 4.1). Ces paires de bases complémentaires sont appelés *paires de bases*, notés *pb*.

Puisque l'appariement des bases est bien défini, seule la séquence de bases consécutives d'un brin est utilisée (la séquence de l'autre brin est facilement déductible). Ainsi, une molécule d'ADN est codée par une suite de lettres sur l'alphabet $\Sigma = \{A, C, G, T\}$, qui représente un des deux brins complémentaires.

De façon similaire, la plupart de protéines peut être représentée par une suite sur un alphabet de 20 lettres, représentant les 20 acides aminés (succession des bases) qui constituent la plupart de protéines issues des gènes (Attwood et al., 2011).

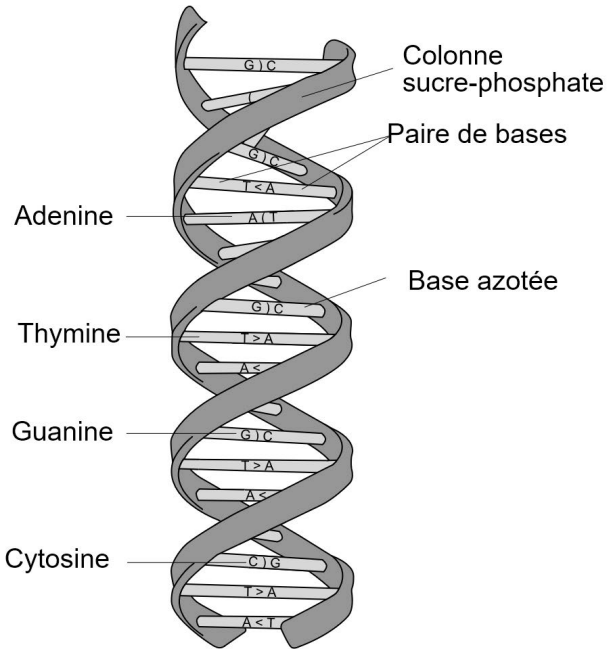


Figure 4.1 – Structure de l'ADN (image tirée de Wikipédia)

4.1.2 Problèmes combinatoires en bio-informatique des séquences

La bio-informatique des séquences traite de l'analyse de données biologiques issues du génome et représentées par des séquences d'ADN ou par des séquences de protéines. Elle s'intéresse en particulier à la recherche des ressemblances entre les séquences (Gusfield, 1997, Rajasekaran et al., 2001a, b), à la recherche de motifs ou structures consensus pour caractériser les séquences (Meneses et al., 2005, Mousavi et al., 2012, Pappalardo et al., 2013), à l'assemblage de fragments d'ADN pour déterminer la séquence complète du génome (Kececioglu et Myers, 1995, Myers Jr, 2016), à l'identification des gènes ou de régions pertinentes dans les séquences (Burge et Karlin, 1998, Claverie, 1992, Mathé et al., 2002), en se basant sur la succession de bases nucléotides ou d'acides animés.

Plusieurs problèmes biologiques de séquences peuvent être modélisés sous la forme d'un problème d'optimisation combinatoire. Pour les modèles ($\mathcal{NP}$)-difficiles de ces problèmes, les méthodes exacts, et donc de complexité exponentielle, basés sur la programmation dynamique ou utilisant des techniques de recherche arborescence (comme

la technique de *Branch & Bound*), deviennent rapidement inutilisables pour les instances de grande taille. Les algorithmes (méta)heuristiques sont donc inévitables pour calculer des solutions approchées en des temps raisonnables.

Plus récemment, Blum et Festa ont publié un livre sur les problèmes d'optimisation combinatoires ($\mathcal{NP}$-)difficiles en bio-informatique des séquences (Blum et Festa, 2016). Ce livre se focalise sur les récents travaux portant sur les algorithmes heuristiques et métaheuristiques pour la résolution d'une collection de ces problèmes.

4.2 Assemblage de fragments d'ADN

La première étape dans l'étude du génome d'un organisme spécifique est la détermination de la séquence complète d'ADN : c'est le processus de ***séquençage d'ADN***. Les informations fournies par cette étape servent à identifier les fonctions des gènes dans le génome (i.e., la prédiction des gènes).

4.2.1 Définitions et notations

Nous donnons ici quelques définitions et notations relatives aux séquences, qui sont nécessaires à la compréhension de la suite de cette thèse.

Une séquence d'ADN est une suite finie non vide de lettres sur l'alphabet $\Sigma = \{A, C, G, T\}$. La longueur d'une séquence s est noté $|s|$ et $s[i]$, $i = 1, 2, \ldots, |s|$, désigne sa i-ème lettre.

Définition 4.1 (Complément inverse) *Etant donnée une séquence d'ADN* $s = s[1]s[2]\ldots s[|s|-1]s[|s|]$*, son complément inverse ("reverse complement" en anglais), noté* $\overleftarrow{s}$ *est la séquence* $\overleftarrow{s[|s|]}\,\overleftarrow{s[|s|-1]}\ldots\overleftarrow{s[2]}\,\overleftarrow{s[1]}$*, où* $\overleftarrow{A} = T$*,* $\overleftarrow{T} = A$*,* $\overleftarrow{C} = G$*, et* $\overleftarrow{G} = C$*.*

Soit l un entier positif. Le préfixe (resp. suffixe) de s de longueur l est noté $pref_l(s)$ (resp. $suff_l(s)$).

Etant donné deux séquences d'ADN s et t tel que $suff_l(s) = pref_l(t)$, nous notons par $s|_l t$ la séquence obtenue par la concaténation de s avec les $|t| - l$ dernières lettres de t : $s|_l t = s[1]\ldots s[|s|]t[l+1]\ldots t[|t|]$. Par exemple, si $s = ATGCCTGG$ et

$t = CTGGATT$ alors $s|_4 t = ATGCCTGGATT$. La notation $s_1|_{l_1} s_2|_{l_2} \ldots |_{l_{m-1}} s_m$ est utilisée pour dénoter la séquence s'_{m-1} obtenue de la manière suivante. Dans un premier temps, nous obtenons la séquence $s'_1 = s_1|_{l_1} s_2$, puis, pour chaque $i = 2, 3, \ldots m-1$ nous posons $s'_i = s'_{i-1}|_{l_i} s_{i+1}$. Donc, pour les deux séquences s et t précédentes, et une troisième séquence $x = ATTGACT$, la notation $s|_4 t|_3 x$ se donne pour la séquence $ATGCCTGGATTGACT$.

Soient k un entier positif et $\mathcal{L}$ une collection de séquences d'ADN.

Définition 4.2 (Contig) *(inspirée de (Ferreira et al., 2002)) Si une suite de séquences $s = \langle s_1, s_2, \ldots s_m \rangle$ dans $\mathcal{L}$ vérifie pour toutes deux séquences successives, s_j et s_{j+1}, il existe un entier positif $l_j \geq k$ tel que $pref_{l_j}(s_j) = suff_{l_j}(s_{j+1})$, et ni s_j est une sous-séquence de s_{j+1} ni s_{j+1} est une sous-séquence de s_j, alors la séquence $z = s_1|_{l_1} s_2|_{l_2} \ldots |_{l_{m-1}} s_m$ est appelée un k-contig dans $\mathcal{L}$.*

4.2.2 Séquençage shotgun

Toutes les machines de séquençage imposent une longueur de lecture maximale et produisent des erreurs (Myers Jr, 2016). C'est-à-dire, un séquenceur d'ADN peut déterminer la séquence des bases A, C, T et G, de gauche à droite le long d'un brin d'une double hélice d'ADN jusqu'à une longueur moyenne typique, et a un taux d'erreur typique. La courte séquence fournie par le séquenceur est appelée une ***lecture*** (issue du terme anglais ***read***).

A cause de cette limitation technologique, un génome qui comporte généralement plusieurs très longs chromosomes est typiquement déterminé par la ***méthode shotgun*** introduite en 1982 par Sanger (Sanger et al., 1982). Comme il est illustré dans la figure 4.2, tout d'abord, plusieurs copies purifiées de la séquence ***cible*** (i.e., le génome ou son segment clonal) sont créées. Puis, les copies sont découpées (au moyen de substances chimiques) en un nombre très important de petits ***fragments*** en des positions aléatoires. Un séquenceur d'ADN est ensuite utilisé pour séquencer chaque fragment[1]. Ces trois premières étapes se déroulent en laboratoire. Afin de s'assurer que

[1]Un séquenceur d'ADN donne des fichiers électroniques contenant des séquences numériques d'ADN.

suffisamment de fragments se chevauchent, la lecture de fragments se poursuit jusqu'à ce qu'une ***couverture*** soit suffisante. Les chevauchements entre les fragments sont utilisés pour former des ***contigs*** (voir définition 4.2) qui représentent des portions du génome. Une couverture suffisante permet également de réduire davantage les erreurs dues au séquençage.

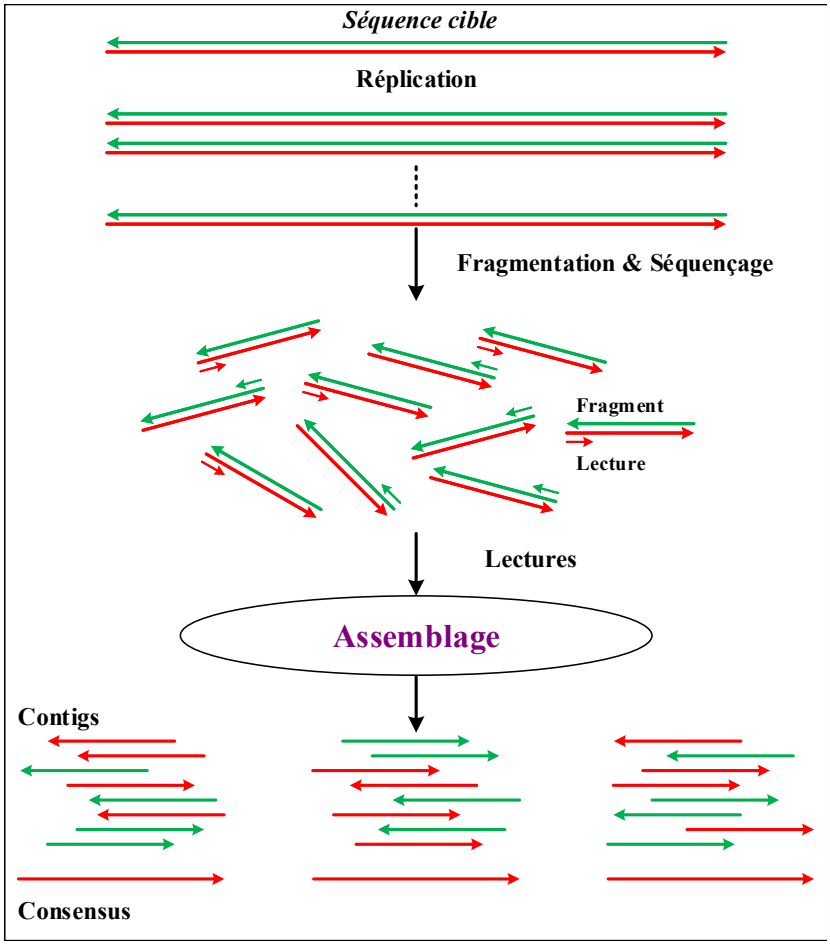


Figure 4.2 – Illustration graphique de la séquençage shotgun d'ADN (Modification de l'illustration donnée dans (Myers Jr, 2016))

La couverture à une position particulière le long de la séquence cible indique le nombre de fragments dans cette position. Elle mesure la redondance d'information des fragments, et indique le nombre de fragments, en moyenne, dans lequel une molécule donnée dans la séquence cible est prévue d'apparaitre. Cette mesure est calculée comme le nombre de bases lu à partir des fragments divisé par la longueur de la

séquence originale (Setubal et Meidanis, 1997) :

$$c = \frac{\sum_{f \in \mathcal{L}} |f|}{G} \tag{4.1}$$

où $\mathcal{L}$ est la collection de fragments d'ADN, $|f|$ est la longueur du fragment $f \in \mathcal{L}$, et G est la longueur de la séquence cible. La distribution de la couverture sert à mesurer la qualité de la ***séquence consensus***. Plus la couverture est grande, moins il y a de gaps, et meilleur est le résultat.

Après l'obtention d'une collection de fragments d'ADN ($\mathcal{L} = \{f_1, \ldots, f_n\}$), la séquence complète du génome peut être reconstruite par assemblage de fragments. Le ***problème d'assemblage*** est généralement traité selon une approche traditionnelle désignée par le signe "***OLC***" pour ***Overlap-Layout-Consensus***. De grands génomes eucaryotes (ceux des animaux, végétaux et protozoaires) ont été assemblés avec succès par cette approche (Adams et al., 2000, Chinwalla et al., 2002, Venter et al., 2001). Nous décrivons l'approche OLC dans le paragraphe suivant.

La ***méthode paired-end shotgun***, proposée premièrement dans (Weber et Myers, 1997) en 1997, utilise un type de séquençage dit paired-end. Le séquençage paired-end génère à partir d'un fragment (suffisamment long) une paire de lectures éloignés d'une certaine longueur (de centaines *bp*) au maximum. Autrement dit, les deux extrémités d'un même fragment sont séquencées par un séquenceur. Ces lectures en paired-end servent à connecter les contigs, obtenus à partir de courtes lectures, entre eux. Si on repère un certain nombre de paires de lectures présents dans deux contigs (chaque lecture est dans un contig), on peut considérer que ces paires de lectures forment la jointure entre les deux contigs. Une collection de contigs ordonnés et connectés par paires de lectures est appelée ***scaffold*** (Paszkiewicz et Studholme, 2010).

4.2.3 L'approche OLC et difficultés d'assemblage

Selon l'approche OLC, le processus d'assemblage se déroule en trois phases (Myers Jr, 2016) : (1) recherche des chevauchements approximatifs, (2) détermination de l'alignement, et (3) construction de la séquence consensus. Dans ce qui suit, nous décrivons chaque phase.

Phase 1 : Recherche des chevauchements (*Overlap*) Cette phase consiste à chercher le bon ou le plus long alignement entre le suffixe d'une lecture et le préfixe d'une autre lecture. Dans cette étape, on considère toutes les paires de lectures et leurs compléments inverses (voir définition 4.1) pour déterminer leur similarité. Habituellement, un algorithme d'alignement semi-global par programmation dynamique est utilisé dans cette étape. L'intuition derrière la recherche des chevauchements entre paires de lectures est que deux lectures suffisamment chevauchantes se retrouvent très probablement l'une à côté de l'autre dans la séquence cible.

Phase 2 : Alignement (*Layout*) Cette phase consiste à chercher un ordre plausible de fragments en se basant sur les chevauchements calculés. En d'autres termes, le chevauchement est utilisé pour organiser les fragments l'un à côté de l'autre pour reconstituer la séquence originale dont ils sont issus. C'est la plus difficile étape parce que la décision d'assembler deux fragments se base sur leur chevauchement qui peut être approximatif du fait des erreurs de séquençage. Les difficultés auxquelles on doit porter attention sont (Alba et Luque, 2007, Myers Jr, 2016) :

1. Orientation inconnue. Après le découpage de la séquence originale en un très grand nombre de fragments, l'orientation est oubliée (i.e., on ne sait pas lequel des deux brins d'une double hélice d'ADN on séquence). Si une lecture ne se chevauche pas avec toute autre lecture, il est encore possible que son complément inverse ait un tel chevauchement. Pour n fragments, il y a 2^n combinaisons possibles en termes d'orientation (pour un seul arrangement de fragments).

2. Erreurs de séquençage. Les erreurs de séquençage peuvent survenir sous plusieurs formes : substitutions, insertions et délétions. Elles sont dues à des erreurs expérimentales dans la procédure électrophorèse (une technique de laboratoire utilisée pour la lecture des séquences d'ADN). Elles peuvent biaiser la détection des chevauchements entre paires de lectures. De ce fait, la détermination de la séquence consensus nécessite des alignements multiples dans les régions fortement couvertes.

3. Manque de couverture. Cette difficulté est due à la couverture insuffisante ou à

l'échantillonnage non aléatoire de fragments. Elle apparait quand l'algorithme d'assemblage ne peut assembler la collection de fragments dans un seul contig. Un contig est une séquence continue et ordonnée dans laquelle le chevauchement entre les fragments successifs est supérieur un seuil prédéfini appelé *cutoff* (le paramètre k dans la définition 4.2).

4. Régions répétées. Les répétitions sont des séquences qui apparaissent plusieurs fois dans la séquence cible. Elles représentent l'une des sources les plus importantes de difficultés de tout projet de séquençage. On ne sait pas si une séquence est répétée, combien de fois elle est répétée, ou si une répétition est son complément inverse. En effet, aucun assembler existant ne traite de manière parfaite les répétitions.

5. Séquences chimériques et Contamination. Deux ou plusieurs fragments séparés sont chimériques s'ils se fusionnent pour former un seul fragment. On parle de contamination lorsque la purification des fragments, après clonage, de l'ADN du vecteur est incomplète. Les fragments contaminés doivent être éliminés avant l'assemblage.

Après avoir déterminé l'ordre d'assemblage de fragments d'ADN, un algorithme d'alignement progressif est appliqué pour combiner tous les alignements par paire calculés dans la première phase.

Phase 3 : Génération d'une séquence consensus (*Consensus*) Cette phase consiste à produire une séquence d'ADN à partir du résultat de la phase d'alignement. La technique la plus utilisée dans cette phase consiste à appliquer une règle de majorité pour construire la ***séquence consensus***.

Une simple illustration des trois phases de l'approche OLC est donnée dans la figure 4.3. A partir d'un ensemble de cinq fragments, dans un premier temps les chevauchements maximaux (exacts) entre les paires de fragments sont calculés. Puis, un ordre d'assemblage de fragments est déterminé. Enfin, une séquence consensus est construite en assemblant les cinq fragments. Le problème réel est certainement plus

compliqué que cette illustration, puisque il comporte, comme nous avons déjà discuté, des compléments inverses, des erreurs de séquençage, des répétitions, et d'autres difficultés.

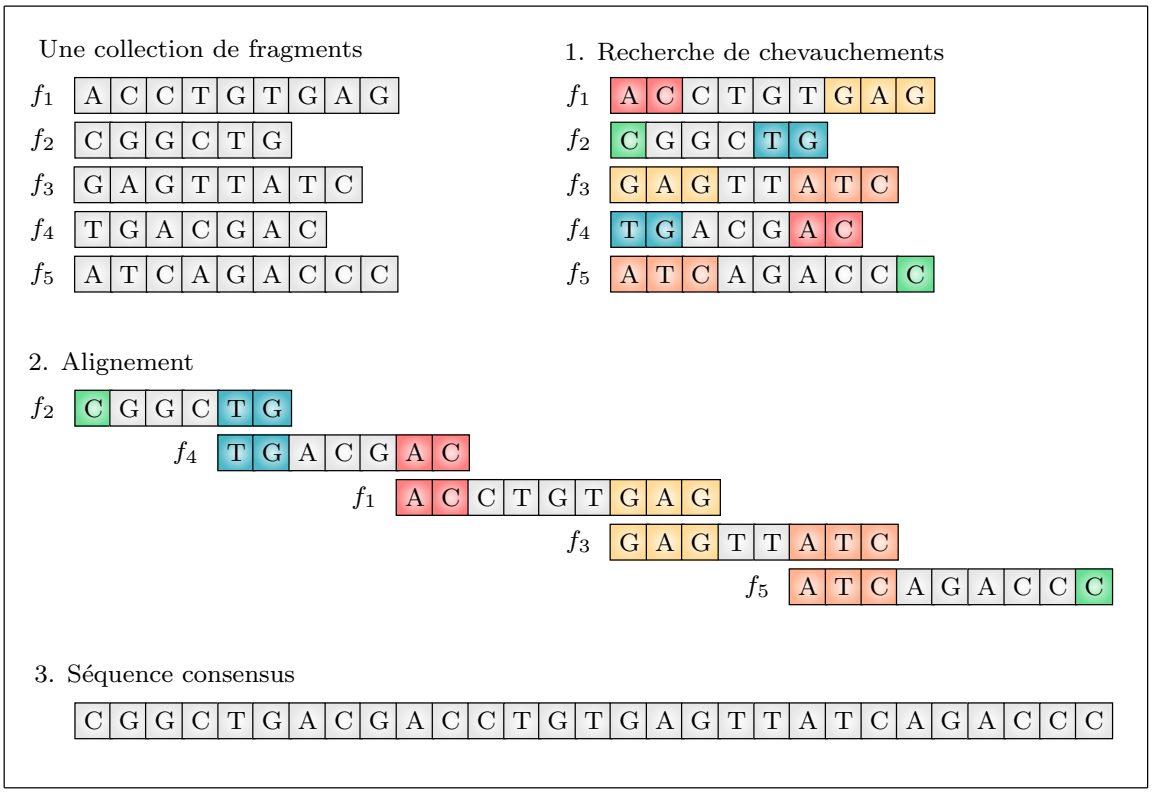


Figure 4.3 – Illustration des trois phases de l'approche OLC.

Par la suite de ce chapitre et de cette thèse, nous adressons le problème d'alignement de la deuxième phase, qui est connu dans la littérature sous le nom, en anglais, "DNA fragment assembly problem" - littéralement, "problème d'assemblage de fragments d'ADN". Ce problème a été montré $\mathcal{NP}$-difficile par Pevzner (Pevzner, 2000), par réduction au problème de chemin hamiltonien, ce qui signifie que des algorithmes (méta)heuristiques sont nécessaires pour calculer des solution approchées.

Aux difficultés mentionnées plus haut Kim et Mohan (Kim et Mohan, 2003) ont ajouté une autre difficulté majeure : l'absence d'un modèle formel pleinement satisfaisant pour le problème d'assemblage de fragments d'ADN. Dans la section suivante, nous présentons les modèles formels existants.

4.2.4 Modèles formels existants

Quatre modèles formels ont été proposés pour le problème d'assemblage de fragments d'ADN (Kim et Mohan, 2003) :

- **Plus courte "super-chaîne" commune** (en anglais *Shortest Common Superstring* : SCS) : Dans ce modèle, le problème d'assemblage de fragments d'ADN consiste à chercher la plus courte chaîne qui contient chaque fragment comme une sous-chaîne. Ce problème est connu comme étant $\mathcal{NP}$-difficile (Kececioglu et Myers, 1995, Kececioglu, 1993, Räihä et Ukkonen, 1981). Il a été simplifié et considéré équivalent à chercher le chemin hamiltenien de poids minimum dans le graphe de chevauchement (voir section 4.3.1 pour une représentation de ce type de graphe) (Parsons et al., 1993). Ce qui ne répond pas toujours aux besoins des biologistes dans le cas de répétitions.

- **Reconstruction** : Dans ce modèle, le problème est de chercher la plus courte chaîne dont la distance (d'édition) à tout fragment ou à son complément inverse est inférieure à un seuil donné. Ce modèle traite la difficulté liée à l'orientation inconnue des fragments et les erreurs de séquençage, mais pas le cas de répétitions et le cas de faible couverture.

- **Multicontig** : L'objectif est de chercher une partition de la collection de fragments en un nombre minimal de sous-ensembles tel que chaque sous-ensemble admet un "t-contig" pour un "ϵ-consensus", où :

  - Un "t-contig" est un alignement d'un sous-ensemble de fragments avec le lien la plus faible (i.e., le plus court chevauchement entre deux fragment successifs) est supérieur ou égal un seuil t (voir définition 4.2) ;
  - Un "ϵ-consensus" est une séquence consensus qui vérifie la condition suivante : la distance entre chaque fragment f et sa projection dans cette séquence est au plus $\epsilon\,|f|$, où $|f|$ est la longueur de f.

  Le problème a été démontré $\mathcal{NP}$-difficile dans (Ferreira et al., 2002). L'inconvénient majeur de ce modèle réside dans le fait que le résultat d'assemblage est

une collection de contigs avec plusieurs gaps entre eux due à la faible couverture ou au taux élevé d'erreurs.

- **Approche du chemin Eulérien** : Ce modèle transforme le problème à un problème de chemin Eulérien après la construction d'un graphe à chaînes ou d'un graphe de de Bruijn (voir sections 4.3.2 et 4.3.3 pour une description de ces types de graphes). Il a été développé afin de résoudre les répétitions. Son inconvénient est qu'il ne traite pas les autres difficultés. Parmi les premiers travaux qui ont utilisé ce modèle on peut citer par exemple (Pevzner et al., 2001).

Dans la section suivante, nous présentons le lien entre l'assemblage de génomes et trois principaux graphes, qui sont : le graphe de chevauchement, le graphe à chaînes et le graphe de de Bruijn.

4.3 Assemblage de génomes

Les assembleurs existants (dont certains ont été utilisés par les biologistes dans des projets d'assemblage de génome) utilisent différentes heuristiques et structures de données. Les trois principaux graphes sont : le graphe de chevauchements, le graphe à chaînes, et le graphe de de Bruijn. La description donnée ci-après de ces structures s'inspire essentiellement de (Myers Jr, 2016).

4.3.1 Graphe de chevauchements

Le graphe de chevauchements (en anglais *overlap graph*) est un graphe orienté pondéré où les sommets représentent les fragments d'ADN et les arêtes représentent les chevauchements entre ces fragments. Ce type de graphe a été décrit pour la première fois dans la littérature en 1983 par Gallant (Gallant, 1983). La première formulation mathématique a été proposée dans (Myers, 1995). Par souci de simplicité, toutes les arêtes dont l'un de ses fragments est inclus dans l'autre sont supprimées. Soient $seq(f)$ la séquence du fragment f, et $seq(f \rightarrow g)$ la suffixe du fragment g qui n'est pas

en chevauchement avec le fragment f. Alors, le chemin $f_1 \rightarrow f_2 \rightarrow f_3 \rightarrow \ldots f_n$ représente l'ordre d'assemblage de n fragments dans un contig qui a la séquence $seq(f_1)seq(f_1 \rightarrow f_2)seq(f_2 \rightarrow f_3)\ldots seq(f_{n-1} \rightarrow f_n)$. Intuitivement, un ensemble de chemins parcourant le graphe donne une collection de contigs qui peut constituer un assemblage. En effet, un chemin Hamiltonien représente un assemblage de tous les fragments dans un seul contig.

Les premiers travaux ont considéré le problème d'assemblage de fragments d'ADN comme une généralisation du problème de plus courte "super-chaîne" commune (SCS), comme suit : étant donné n fragments $f_1, f_2, \ldots, f_n$, chercher la plus courte chaîne S telle que chaque fragment ou son complément inverse est inclus complètement dans S. Ce problème a été considéré équivalent à chercher le chemin Hamiltonien dans le graphe de chevauchements, i.e., plus les fragments se chevauchent, plus courte est la chaîne dérivée par le chemin. C'est un problème $\mathcal{NP}$-difficile (Hoffman et al., 1986), pour lequel des heuristiques gloutonnes ont été utilisées. Ce type d'algorithmes ajoute progressivement les arêtes de chevauchements maximaux au chemin solution en construction (tout en évitant la création de cycles). Les solutions fournies par ce type d'algorithmes ne sont pas toujours optimales. Les tous premiers travaux ont utilisé des variations de cette approche gloutonne qui considère le chevauchement pour l'incorporation des arêtes dans le chemin partiel selon un certain ordre, où la différence entre les variantes réside principalement dans l'ordre utilisé.

Ce modèle formel (de SCS) échoue dans le cas de répétitions dans les longues séquences. L'objectif était éventuellement de construire des solutions dans lesquelles les répétitions sont surcomprimées en pliant les copies de chaque répétition. Par ailleurs, le graphe de chevauchements consomme trop d'espace mémoire et un temps de calcul très important lorsque le nombre de fragments d'ADN est assez grand, comme c'est le cas avec la technologie de séquençage à haut débit (*Next Generation Sequencing*, NSG)[2] (Church, 2006, Hall, 2007). Plusieurs études ont été menées afin de réduire ces coûts. Des méthodes pour éliminer les mauvais chevauchements ont été proposées (Margulies et al., 2005, Rasmussen et al., 2006). Des techniques pour réduire le temps

[2]Les séquenceurs de nouvelle génération sont des appareils massivement parallèles et produisent beaucoup de courtes lectures.

de calcul des chevauchements ont été aussi proposés (Mallén-Fullerton et al., 2013, Myers, 2014).

4.3.2 Graphe à chaînes

Le premier article dans la littérature présentant l'idée de graphe à chaînes (*string graph*) a été publié en 1995 par Eugene Myers (Myers, 1995). Ce type de graphe a été introduit afin de faciliter l'assemblage des répétitions du génome.

En fait, le graphe à chaînes est obtenu en compressant le graphe de chevauchements, dans l'optique de diminuer l'espace mémoire occupé par la structure. La compression se fait comme suit : on effectue dans un premier temps les réductions transitives (une arête $a \rightarrow b$ est transitivement réductible, et par conséquent peut être supprimée, s'ils existent une lecture c et deux arêtes $a \rightarrow c$ et $c \rightarrow b$ telle que le chevauchement entre les lectures a et b est impliqué par les chevauchements avec c), puis dans un deuxième temps on compresse les chaînes de sommets de degré entrant et sortant 1 en une seule arête composée. Avec cette construction, les séquences uniques dans l'échantillon biologique se réduisent en des arêtes composées uniques. De même, les copies de chaque répétition forment aussi une arête composée unique, mais si celle-ci est suffisamment longue (en termes des sommets qu'elle regroupe), le taux d'arrivée sert à distinguer clairement les arêtes composées représentant des séquences uniques de celles représentant des répétitions.

Le graphe à chaînes a été utilisé premièrement pour l'assemblage paired-end du génome complet dans l'assembleur Celera (Myers et al., 2000). Les algorithmes développés pour cet assemblage fonctionnent en trois étapes : (1) construction d'un graphe à chaînes à partir de courtes lectures, puis détermination de toutes les arêtes composées (contigs) qui représentent des séquences uniques ; (2) jonction de contigs en utilisant des lectures paired-end ; (3) fermeture de gaps entre les contigs en assemblant uniquement les longes séquences paired-end utilisées pour connecter les contigs.

Comme dans le cas du graphe à chevauchements, le problème d'assemblage de fragments d'ADN peut être considéré comme le problème de chercher un chemin parcourant le graphe à chaînes, où celui-ci a l'avantage d'être plus simple et n'élimine

aucune solution potentielle. En effet, un assemblage correct correspond à un chemin Eulerien généralisé respectant le nombre de copies de chaque arête composée. En d'autres termes, dans ce chemin, chaque arête composée représentant une séquence unique doit être traversée une seule fois, tandis que chaque arête composée représentant une répétition doit être traversée autant de fois que le nombre de copies de la répétition dans la séquence cible. Une analyse de flot sert à inférer le nombre de copies des répétitions (Myers, 2005).

4.3.3 Graphe de de Bruijn

Le graphe de *de Bruijn* (Idury et Waterman, 1995) et le graphe à chaînes ont été présentés au même colloque (DIM, 1994). Toutefois le graphe de de Bruijn reste inutilisé jusqu'à l'arrivée de la technologie de séquençage de nouvelle génération (depuis 2002). Les approches gloutonnes et les stratégies utilisant ce type de graphe sont devenues populaires pour cette technologie (Miller et al., 2010). Cela inclut par exemple les applications d'assemblage Velvet (Zerbino et Birney, 2008), Edena (Hernandez et al., 2008), ABySS (Simpson et al., 2009), et SOAPdenovo (Li et al., 2010).

Pour un entier $k > 0$, le graphe de de Bruijn pour une collection de lectures d'ADN est construit à partir de toutes les sous-séquences de longueur k de chaque lecture. Ces sous-séquences sont appelées k-mers. Dans ce graphe, les sommets sont les k-mers possibles et les arêtes (orientées) se présentent entre chaque paire de k-mers ayant un chevauchement de $k-1$ nucléotides. Il est clair que le graphe de de Bruijn est transitivement réduit par construction (i.e., les chevauchements de moins de $k-1$ nucléotides ne sont pas considérés pour les arêtes, puisqu'ils sont impliqués par les chevauchements de $k-1$ nucléotides). D'autre part, par une analogie avec la construction du graphe à chaînes, les chaînes de sommets de degré entrant et sortant 1 dans le graphe de de Bruijn peuvent être regroupées pour former des arêtes composées. En effet, il y a une similarité déjà remarquable entre le graphe de de Bruijn et le graphe à chaînes ; dans un cas les sommets sont des k-mers et les arrêtes représentent des chevauchements exacts de $k-1$ nucléotides, et dans l'autre cas les sommets sont des lectures et les arêtes correspondent à des chevauchements approximatifs. Cependant,

il y a des différences algorithmiques significatives en raison de la différence entre ce qu'un sommet représente (i.e., une lecture versus un k-mer), comme suit.

La chaîne de k-mers d'une des lectures correspond à un chemin dans le graphe. Ce chemin est appelé l'***image*** de la lecture. Soit $cnt(a \rightarrow b)$ le nombre de lectures dont l'image passe par l'arête $(a \rightarrow b)$. Si la majorité des k-mers dans le graphe sont corrects (i.e., ils ne contiennent pas des erreurs de séquençage) et la couverture de la séquence ciblée est élevée, alors les arêtes ayant de très faibles valeurs de cnt sont très probablement dues à des erreurs de séquençage. En effet, la majorité des assembleurs de de Bruijn éliminent ces arêtes. Supposons que les arêtes correctes sont conservées. Alors le graphe pourrait contenir uniquement les arêtes dans l'image de la séquence cible inconnue et, par conséquent, un certain chemin Eurlien pourrait la produit. Ainsi, le problème d'assemblage avec un graphe de de Bruijn est de chercher un chemin qui est cohérant avec les images de toutes les lectures, en ignorant les arêtes écartées et en traversant les arêtes correctes autant de fois que leur nombre de répétitions dans la séquence cible.

Pour finir, il est à noter que le choix optimal de la longueur de sommets (k) repose sur deux critères contradictoires : k doit être assez grand de sorte que chaque k-mer ne se produit qu'une seule fois dans la séquence cible (sauf les répétitions), mais en contrepartie, il doit être suffisamment petit pour qu'une majorité significative de k-mers d'une des lectures soit correcte.

4.4 Etat de l'art en assemblage de fragments d'ADN par (méta)heuristiques

Comme la tâche d'assemblage de fragments d'ADN (plus précisément la phase d'alignement de fragments) se ramène à un problème d'optimisation combinatoire $\mathcal{NP}$-difficile, depuis trois décennies, de nombreux chercheurs ont appliqué différents algorithmes heuristiques et métaheuristiques pour sa résolution en des temps raisonnables (Indumathy et Maheswari, 2012, Luque et Alba, 2005). Les algorithmes exacts, basés sur la programmation linéaire ou utilisant par exemple la technique

par séparation et évaluation "Branch & Bound" (Ferreira et al., 2002, Jing et Khuri, 2003), deviennent rapidement inutilisables avec la complexité du problème. Le choix d'algorithmes (méta)heuristiques inclut une variété d'algorithmes : algorithmes de recherche locale, algorithmes évolutionnaires et algorithmes bio-inspirés. Cette voie de recherche reste encore expérimentale et peu investie en raison du fait que ces types d'algorithmes n'ont pas encore reçu d'application réelle (les applications d'assemblage populaires utilisent des approches gloutonnes).

Dans cette partie, nous présentons une revue de la littérature portant sur l'application des algorithmes heuristiques et métaheuristiques pour la résolution approchée du problème d'assemblage de fragments d'ADN. Toutes les techniques exposées considèrent le problème selon l'approche OLC. Nous commençons par présenter les critères utilisés pour quantifier la qualité d'un résultat d'assemblage de fragments d'ADN. Puis, nous présentons brièvement un assembleur heuristique très connu dans la littérature et mentionnons ses variantes existantes. Nous passons ensuite en revue les différents travaux portant sur l'application des algorithmes génétiques (GAs). Nous abordons par la suite les différentes techniques basées sur l'intelligence en essaim. Enfin, nous présentons brièvement les travaux portant sur l'utilisation d'autres algorithmes métaheuristiques.

4.4.1 Evaluation d'un résultat d'assemblage de fragments d'ADN

L'objectif est de chercher un ordre total des fragments donnés qui aboutit à une séquence consensus reflétant avec précision la séquence parent. Si la séquence parent est préalablement connue, on peut juger la qualité d'un résultat d'assemblage par une mesure de ressemblance entre la séquence consensus finale et la séquence parent connue. Cependant, en pratique, l'ensemble de fragments d'ADN est généré pour trouver la séquence parent. Ainsi, d'autres critères doivent être définis pour quantifier la qualité d'un résultat d'assemblage de fragments d'ADN.

Les algorithmes présentés par la suite considèrent le problème d'assemblage de fragments d'ADN comme un problème combinatoire de permutation. Ils travaillent sur une matrice de score w, où chaque élément $w_{i,j}$ contient le chevauchement entre

les deux fragments i et j, $1 \leq i, j \leq n$ (i.e., un graphe à chevauchements). Nous présentons ci-dessous trois fonctions objectifs qui ont été utilisés par la plupart de ces algorithmes.

4.4.1.1 Maximiser les chevauchements entre les fragments successifs

La fonction objectif la plus utilisée dans la littérature est formulée en analogie avec la fonction objectif du problème TSP (Parsons et al., 1995). Elle est calculée comme la somme des chevauchements entre les fragments successifs :

$$F_1(s) = \sum_{i=1}^{n-1} w_{s[i],s[i+1]} \tag{4.2}$$

où $s[i] = j$ indique que le fragment j est apparu dans la position i dans la permutation -de taille n- s. Cette fonction objectif a pour but d'orienter le processus de recherche vers les configurations qui présentent de forts chevauchements entre les fragments successifs (ce qui donne un problème de maximisation). L'évaluation de cette fonction demande un temps en $O(n)$.

4.4.1.2 Minimiser les chevauchements entre les fragments distants

Une autre fonction objectif considère les chevauchements entre toutes les paires de fragments non successifs (Parsons et al., 1995). Elle permet de pénaliser les configurations qui présentent de forts chevauchements entre les fragments non successifs. Cette mesure, notée F_2,

$$F_2(s) = \sum_{i=1}^{n} \sum_{j=1}^{n} |i - j| \times w_{s[i],s[j]} \tag{4.3}$$

utilise la valeur absolue de la distance entre deux fragments dans la permutation comme un poids associé à leur chevauchement. Le problème consiste donc à minimiser cette fonction. La complexité en temps de cette fonction est en $O(n^2)$, parce que toutes les paires de fragments doivent être considérés.

4.4.1.3 Minimiser le nombre de contigs

Alba et Luque ont déclaré dans (Alba et Luque, 2007) que l'objectif primordial de l'assemblage de fragments d'ADN consiste à minimiser le nombre de contigs, avec le but d'atteindre un seul contig. Dans (Parsons et al., 1995), les auteurs ont écarté ce choix pour la raison suivante : plusieurs permutations extrêmement différentes peuvent présenter le même nombre de contigs, ce qui empêche la distinction entre eux dans un processus de recherche d'une bonne solution. Pour régler ce problème, Alba et Luque ont utilisé dans leur algorithme de recherche locale, nommé PALS (voir section 5.2 pour une présentation détaillée), la fonction objectif F_1 comme un critère secondaire pour parfaire la comparaison dans le cas où il y a plusieurs solutions candidates avec le même nombre de contigs. On note par *cutoff* le chevauchement minimal pour considérer que deux fragments successifs sont dans le même contig. Le nombre de contigs peut être donné par :

$$nContigs(s) = \sum_{i=1}^{n-1} \sigma_{s[i],s[i+1]} + 1 \tag{4.4}$$

où $\sigma_{s[i],s[i+1]}$ vaut 1 si le chevauchement entre les fragments des positions successives i et $i+1$ est inférieur *cutoff*, et 0 sinon. L'évaluation du nombre de contigs demande un temps linéaire avec le nombre de fragments ($O(n)$).

4.4.2 L'algorithme PALS et ses variantes

L'algorithme PALS (pour *Problem Aware Local Search*) a été proposé par Alba et Luque en 2007 (Alba et Luque, 2007). C'est une méthode de recherche locale qui fait évoluer une seule solution en utilisant la notion de voisinage et des critères permettant de définir le sous-ensemble de solutions acceptables parmi toutes les solutions voisines d'une solution courante. La solution (un ordre de fragments) est codée en représentation par permutation. L'algorithme PALS modifie itérativement la solution courante par l'application de mouvements d'une manière bien structurée. A chaque itération de l'algorithme, le voisinage complet de la solution courante, défini par le mouvement, est généré. Le meilleur mouvement améliorant est ensuite appliqué à la solution courante.

Ce processus est répété jusqu'aucune amélioration n'est pas possible. Le point clé de l'algorithme PALS réside dans la manière avec laquelle il évalue les mouvements et la qualité des solutions. Il utilise le nombre de contigs comme un premier critère pour ordonner les mouvements et pour juger la qualité des solutions. Dans le cas d'égalité, il considère le niveau de chevauchement maximal entre les fragments successifs pour parfaire l'évaluation. Nous reviendrons plus en détail au chapitre 5 sur cet algorithme heuristique.

La limitation majeure de l'algorithme PALS est la forte dépendance de ses résultats de la configuration initiale et sa convergence rapide vers des optimums locaux. En effet, cet algorithme souffre de l'instabilité de ses résultats, notamment sur les instances de grande taille. Pour résoudre ces problèmes de convergence, des approches combinant l'algorithme PALS avec des algorithmes métaheuristiques ont été développées. Nous présentons ci-dessous ces approches hybrides. Il est à noter que dans tous ces algorithmes hybrides la solution est codée en représentation par permutation.

L'algorithme PALS a été appliqué dans (Alba et Luque, 2008) comme un opérateur de recherche additionnel dans un GA. Dans cette approche hybride, les nouveaux chromosomes, nés par l'opérateur de croissement et l'opérateur de mutation, sont exploités pour fournir des configurations initiales pour PALS. L'idée derrière ce type d'hybridation (un algorithme mémétique) est que, après que l'algorithme GA localise une bonne région de solutions dans l'espace de recherche (exploration), PALS permet l'amélioration (exploitation) de cette région via la recherche locale. Un opérateur de croisement OX et un opérateur de mutation échange (1-Opt ou *Swap mutation*) ont été utilisés dans cet algorithme hybride.

Dans deux autres études, (Alba et Dorronsoro, 2009b, Dorronsoro et al., 2008), l'algorithme PALS a été appliqué aussi comme un opérateur de recherche additionnel dans un algorithme génétique cellulaire (*Cellular Genetic Algorithm* : cGA). Dans les deux algorithmes cGAs, la population est structurée en une grille bidimensionnelle, de sorte que les individus ne peuvent interagir qu'avec leurs voisins les plus proches. La différence principale entre ces deux approches réside dans le fait que la première utilise un cGA canonique, tandis que la seconde possède un mécanisme d'auto-adaptation des paramètres de contôrle. Ce mécanisme d'auto-adaptation a pour but de maintenir

un bon équilibre entre l'exploration et l'exploitation de l'espace des solutions, en donnant à la population une forme plus rectangulaire pour favoriser l'exploration et une forme plus carrée pour favoriser l'exploitation. Les deux algorithmes cGAs utilisent un opérateur de croisement OX et un opérateur de mutation échange.

Minetti et al. dans (Minetti et al., 2014) ont proposé récemment une combinaison de l'algorithme PALS avec un algorithme de recuit simulé (voir section 1.3.1 pour une description de cette métaheuristique) pour pouvoir traiter les données bruitées. Dans cette approche hybride, l'algorithme de recuit simulé est appliqué pour diversifier la recherche et ainsi éviter la convergence prématurée. Plus précisément, il est appliqué lorsque l'algorithme PALS n'accepte aucun nouveau mouvement (pour améliorer la solution courante) pendant un certain nombre d'itérations. Ceci permet de produire de nouveaux points de départ pour l'algorithme PALS, vu que l'algorithme de recuit simulé accepte des solutions de qualité inférieure avec une grande probabilité pendant la première phase de recherche (dans laquelle la température est faible). Le test de cette approche hybride a été établi sur des instances bruitées et non bruitées. Les instances de test bruitées ont été générées selon trois sources de bruit : bases moléculaires, fonction objectif et chevauchements.

Comme il a été constaté par leurs développeurs, ces approches hybrides améliorent la qualité de résultats, mais augmentent considérablement le temps de calcul par rapport l'algorithme PALS.

4.4.3 Approches basées sur les GAs

Dans cette section, nous passons en revue plusieurs travaux utilisant les GAs (voir section 1.4.1 pour une présentation de GAs et ses opérateurs) pour résoudre le problème d'assemblage de fragments d'ADN.

Les études présentées dans (Parsons et Johnson, 1995, Parsons et al., 1993, 1995) étaient parmi les premiers travaux qui ont appliqué un GA pour la résolution du problème d'assemblage de fragments d'ADN. Dans (Parsons et al., 1995), les auteurs ont comparé deux codages par permutation et les opérateurs génétiques associés. Les deux fonctions d'évaluation F_1 et F_2 ont été considérées dans cette étude. Les expériences

réalisées ont montré que le GA donne des résultats plus performants qu'une technique de recherche gloutonne.

Li et Khuri dans leur étude comparative (Li et Khuri, 2004) ont implémenté un GA et ont considéré aussi ces deux mêmes fonctions objectifs. Deux opérateurs de croisement OX et ERX, et un opérateur de mutation échange ont été appliqués dans le GA étudié.

Dans (Kim et Mohan, 2003) une approche d'assemblage hiérarchique utilisant un GA distribué adaptatif a été présentée. Dans cette approche, le chromosome des individus est représenté par une liste doublement chaînée de séquences, où chaque élément représente la séquence d'un fragment unique ou la séquence résultant de l'assemblage de deux ou plusieurs fragments. Le modèle hiérarchique consiste à assembler un certain nombre de fragments pour former un seul bloc (ceci est basé sur la qualité de leur alignement) pour les itérations futures de l'AG. L'algorithme s'opère donc selon une série de niveaux, réduisant l'espace de recherche, et travaillant avec les blocs de fragments déjà identifiés. Le modèle adaptif consiste à générer les paramètres de contrôle (le nombre d'individus et le nombre de sous-populations) en fonction des données en entrée. La distribution réside dans le fait que plusieurs sous-populations évoluent selon un modèle insulaire avec échanges de solutions. Une procédure de recherche locale de type "Hill-climbing" a été appliquée pour améliorer la qualité de la meilleure solution obtenue. Cette solution améliorée est utilisée ensuite pour changer le niveau de représentation du chromosome, ce qui implique la réinitialisation de toutes les sous-populations. La fonction objectif utilisée pour évaluer la qualité des solutions correspond à une combinaison linéaire de plusieurs critères : le nombre de contigs, la longueur de la super-chaîne obtenue, le nombre de bases ambiguës, la longueur de chevauchement de la super-chaîne obtenue avec des motifs de gènes, et la correspondance d'orientation entre fragments consécutifs. Cette approche d'assemblage hiérarchique a montré sa bonne performance en comparaison avec l'assembler CAP3 (Huang et Madan, 1999).

Dans (Minetti et al., 2008b) les auteurs ont présenté un GA appliquant une recherche à voisinage variable (voir section 1.3.3 pour une description de cette méthode de recherche locale) pour améliorer les individus de la population. L'objectif était de

maximiser la fonction objectif F_1. Les résultats obtenus par cet algorithme hybride sont comparables aux résultats obtenus par l'algorithme heuristique PALS.

Dans le but de concevoir un GA plus adapté au problème, Minetti et al. dans (Minetti et al., 2008a) ont étudié deux éléments algorithmiques essentiels de GA : la procédure d'initialisation de la population et l'opérateur de croissement. Les auteurs ont considéré le problème de maximiser la fonction objectif F_1. Ils ont comparé plusieurs variantes de chaque élément d'étude. Cette étude a montré que la combinaison d'un tirage aléatoire et d'une heuristique pour générer la population de départ avec l'utilisation d'un opérateur de croissement ERX conduisent à un GA plus efficace pour approcher la solution optimale du problème.

Nebro et al. ont présenté dans (Nebro et al., 2008) un GA distribué. L'objectif est de minimiser le nombre de contigs dans la séquence de fragments. Pour cela, ils ont défini une fonction objectif prenant en compte une approximation du nombre de contigs. Le but principal des auteurs était de réduire le temps de calcul face à des instances de grande taille en distribuant un GA. Cette distribution est réalisée selon un modèle maitre-esclave et réside dans le fait que plusieurs individus de la population sont évalués en parallèle d'une manière asynchrone. Un opérateur de croissement OX et une mutation échange ont été appliqués dans ce GA. Les expériences ont montré que cette approche distribuée améliore considérablement la performance de GA pour le problème d'assemblage de fragments d'ADN.

Kikuchi et Chakraborty (Kikuchi et Chakraborty, 2006, 2012) ont proposé un GA pour approximer le problème de maximiser la fonction objectif F_1. Deux points clés de cette approche sont : (1) réduction de l'espace de recherche et, en conséquence, du temps de calcul ; (2) application d'une heuristique gloutonne pour améliorer la qualité des solutions obtenues. La réduction de l'espace de recherche se fait en réduisent la taille du chromosome (permutation d'entiers) périodiquement. Pour cela, on identifie les fragments successifs formant un contig dans le meilleur chromosome, puis on supprime cet ensemble de fragments successifs de tous les chromosomes de la population (une réduction similaire à celle appliquée dans (Kim et Mohan, 2003), décrit plus haut). Après avoir ajouté ce contig séparé à un ensemble de contigs, on vérifie si on peut l'assembler avec les contigs existants. Si c'est le cas, on réduit encore une fois

le chromosome des individus en supprimant les fragments affectés par la combinaison. L'étape d'amélioration est déclenchée périodiquement et s'opère seulement sur les meilleurs individus de la population. Elle consiste en des opérations d'échange de gènes successifs. Un opérateur de croisement OX et un opérateur de mutation échange ont été utilisé dans ce GA.

Plus récemment, Hughes et al. dans (Hughes et al., 2016) ont combiné un GA avec plusieurs stratégies pour maximiser la fonction objectif F_1.

- Tout d'abord, ce GA utilise une stratégie des variations clonales (en anglais, "ring species" - littéralement, "espèces en anneau"). Cette stratégie permet de restreindre l'évolution entre les individus. Elle consiste à arranger les individus de la population dans un anneau (arrangement linéaire) en fonction des distances entre eux dans l'espace des paramètres, de telle façon que les individus les plus isolés seront contigus. Ultérieurement, selon cet arrangement, les individus voisins uniquement peuvent se reproduire entre eux. De ce fait, cette stratégie a été adoptée pour faire apparaitre de nouvelles espèces dans la population.

- Deuxièmement, ce GA évolue selon un modèle insulaire. Il manipule plusieurs sous-populations séparées concurremment dans le but de maintenir la diversité dans la population.

- Troisièmement, ce GA nommé "Recentering-Restarting Genetic Algorithm" redémarre périodiquement à partir d'une nouvelle population centrée autour d'une solution dite de centre. La population de départ de chaque redémarrage est générée à partir de la solution de centre courante en la modifiant pour la génération de chaque individu par une suite de mouvements différente. La solution de centre du premier démarrage est obtenue de façon aléatoire et/ou heuristique. A chaque redémarrage (après avoir évolué la population un certain nombre de générations), la meilleure solution de la population est comparée avec la solution de centre courante. Si cette meilleure solution est meilleure que la solution de centre courante, elle devient la nouvelle solution de centre et le processus complet est répété mais avec des suites de mouvements différentes. Sinon, la solution de centre courante reste inchangée et le processus est répété avec des

suites de mouvements modifiées. Cette stratégie de redémarrage-recentrage a été appliquée pour éviter la convergence prématurée vers des optimums locaux.

De plus, deux codages du chromosome ont été testés dans cette étude. Le premier codage est direct et consiste en une représentation par permutation (codage usuel). Le second codage est indirect et consiste en une suite de mouvements de taille variable à appliquer à la solution de centre courante pour en obtenir une permutation de fragments modifiée. Les expériences présentées ont montré la capacité de cette combinaison de plusieurs stratégies à éviter la convergence prématurée vers des optima locaux.

D'autres travaux basés sur les GAs ont été publiés dans (Alba et Luque, 2006, Alba et al., 2005, Fang et al., 2005, Luque et Alba, 2011, Rathee et Kumar, 2014).

4.4.4 Algorithmes basés sur l'intelligence en essaim

Dans ce paragraphe, nous passons en revue les travaux portant sur les algorithmes qui s'inspirent de l'intelligence en essaim (Blum et Li, 2008). Ce type d'algorithmes s'inspire des comportements collectifs de certaines espèces qui se déplacent ou évoluent en groupes, comme : l'optimisation par colonies de fourmis, par la suite ACO (voir section 1.4.3), l'optimisation par essaims de particules, par la suite PSO (voir section 1.4.2), l'optimisation par colonie d'abeilles, la recherche coucou, etc.

Un algorithme de ACO pour résoudre le problème d'assemblage de fragments d'ADN a été présenté dans (Meksangsouy et Chaiyaratana, 2003). L'objectif était de maximiser la fonction objectif F_1. Les auteurs ont justifié le choix de cet algorithme métaheuristique par la similarité de ce problème avec le problème du TSP, le problème ordinaire d'illustration des algorithmes de ACO (López-Ibánez et al., 2015) ; c'est par analogie que, le chevauchement entre deux fragments peut être vu comme l'inverse de la distance entre deux sites. Ils ont proposé une représentation asymétrique pour coder les solutions, en considérant l'orientation de chaque fragment dans la permutation (sens de lecture, gauche-à-droite ou droite-à-gauche) et le brin (ordinaire ou son complémentaire) d'où ce fragment vient. Ce codage a été justifié principalement par le fait qu'il représente univoquement toute solution possible (En revanche, avec la

représentation usuelle par permutation simple, des permutations différentes peuvent produire la même solution). Les auteurs ont comparé leur algorithme avec l'heuristique constructive "voisin le plus proche". Les résultats expérimentaux ont montré que l'algorithme de ACO a donné des résultats plus performants que les résultats de l'algorithme heuristique pour les instances multicontigs.

Dans (Verma et al., 2012) un algorithme de PSO a été proposé pour maximiser le chevauchement entre les fragments successifs (i.e., fonction objectif F_1). Comme l'algorithme de PSO manipule les solutions sous forme de vecteurs réels (vecteurs de positions des particules), il est nécessaire de transformer la représentation réelle en une permutation de fragments. Pour cela, les auteurs ont utilisé la règle de SPV (voir section 1.4.2.5 pour une présentation de cette règle heuristique). Tous les travaux portant sur des algorithmes de PSO décrits ci-après ont utilisé cette même règle pour transformer les vecteurs de position en permutations de fragments.

Mallén-Fullerton et Fernández-Anaya ont présenté un algorithme distribué combinant un algorithme de PSO et l'évolution différentielle (ED) pour optimiser F_1 (Mallén-Fullerton et Fernández-Anaya, 2013). La distribution réside dans le fait que la population de particules est divisée en petites sous-populations évoluant séparément sur différents processeurs. A chaque itération de l'algorithme, pour chaque particule de chaque sous-population, un vecteur dit de perturbation est calculé à partir des vecteurs de positions de trois particules sélectionnées de façon aléatoire de la même sous-population. Ensuite, ce vecteur de perturbation est croisé avec le vecteur de position de la particule courante pour substituer certains éléments. En outre, deux opérateurs d'échange ont été appliqués : un échange au niveau de fragments pour améliorer le vecteur de perturbation, et un échange au niveau de contigs (successions de fragments) pour améliorer le vecteur de position de la particule courante après modification par l'opérateur de croisement.

Les auteurs de (Huang et al., 2015) ont proposé un algorithme de PSO en combinaison avec des algorithmes de recherche locale pour résoudre ce même problème. La recherche locale a été appliquée à deux niveaux : pour générer une population initiale et pour améliorer la qualité de la meilleure solution obtenue après l'arrêt de l'algorithme. Un algorithme de recuit simulé et une recherche tabu ont été appliqués

pour l'initialisation, et une recherche à voisinage variable a été utilisée pour l'amélioration (voir sections 1.3.1, 1.3.4 et 1.3.3 pour des descriptions générales de ces trois méthodes de recherche locale, respectivement). Après le test de cette approche hybride, les auteurs ont constaté qu'elle est lente et nécessite un temps de calcul très important.

Dans (Rajagopal et Maheswari Sankareswaran, 2015), les auteurs ont étudié trois variantes de l'algorithme de PSO pour résoudre le problème de maximiser le chevauchement. L'objectif était d'étudier l'impact du poids d'inertie et des composants cognitif et social dans la formule de changement de vitesse (voir équation 1.3). Deux variantes du modèle de poids d'inertie, dans lequel la vitesse d'un pas de temps est multipliée avec un facteur (poids d'inertie), ont été considérées : dans la première variante le poids d'inertie est constant pendant toute la durée de l'algorithme, tandis que dans la deuxième variante il diminue graduellement. Dans une troisième variante dite adaptative, les deux coefficients (c_1 et c_2) qui déterminent les influences cognitive et sociale dans la formule de mise à jour de vitesse sont modifiés graduellement chacun dans un intervalle spécifique (ces coefficients sont fixes dans l'algorithme de base). Au début de la recherche, pour une bonne exploration de l'espace des solutions, l'influence cognitive est élevée tandis que l'influence sociale est faible. En contrepartie, à la fin de recherche, pour intensifier la recherche autour de la meilleure particule dans l'essaim, l'influence cognitive est faible tandis que l'influence sociale est élevée. Cette étude a montré la bonne performance de la variante adaptive.

Plus récemment, Huang et al. ont proposé un algorithme hybride combinant un nouvel algorithme métaheuristique basé sur l'intelligence en essaim et quelques algorithmes de recherche locale (Huang et al., 2016). L'objectif était de maximiser le chevauchement entre les fragments successifs. L'algorithme d'intelligence en essaim utilisé s'inspire des lois de la gravitation de Newton, appelé en anglais "Gravitation Search Algorithm (GSA)". En fait, ce travail est similaire au travail précédent de ces mêmes auteurs (Huang et al., 2015), que nous avons décrit plus haut. La différence entre les deux études réside principalement dans l'algorithme d'intelligence en essaim utilisé (PSO vs GSA). A la différence de l'algorithme de PSO, dans l'algorithme de GSA les interactions entre les particules sont commandées par les forces gravitation-

nelles et les lois de mouvement. Les algorithmes de recherche locale ont été appliqués dans les deux études de la même façon. De même, les auteurs ont conclu que leur approche hybride nécessite un temps de calcul élevé en comparaison avec quelques algorithmes d'assemblage existants.

Dans (Firoz et al., 2012) les auteurs ont proposé de résoudre ce même problème par des algorithmes de colonie d'abeilles artificielles. Ils ont considéré la présence de bruits perturbant les données. Pour générer des données bruitées, ils ont introduit des erreurs dans le mécanisme de fragmentation et de clonage en considèrent plusieurs configurations de paramètres. Après le test, les auteurs ont constaté que leurs algorithmes garantissent une certaine robustesse vis-à-vis du bruit en comparaison avec quelques algorithmes standards.

Un autre algorithme récent d'intelligence en essaim a été aussi adapté avec succés pour maximiser la fonction objectif F_1 (Ezzeddine et al., 2014). Cet algorithme (nommé en anglais, *firefly algorithm*) s'inspire du comportement des lucioles lorsqu'elles produisent des lumières pour devenir attractives. Ces insectes coléoptères lumineux sont attirés par les lumières aux fins de la reproduction entre mâles et femelles. Cette approche repose sur deux règles principales : (1) l'attractivité est proportionnelle à l'intensité de la lumière produite par la luciole ; les lucioles moins lumineuses se meuvent vers les lucioles plus lumineuses ; (2) l'intensité de la lumière décroit proportionnellement avec la distance entre deux lucioles, à cause de l'absorption de lumière. Dans cet algorithme, chaque luciole représente une solution (permutation de fragments), et l'intensité de la lumière est quantifiée par la mesure de qualité de la solution. La distance entre les solutions est calculée dans l'espace de décision (distance entre permutations). Deux mouvements dans l'espace de recherche entre les couples de solutions qui s'attirant deux à deux (i.e., reproduction de nouvelles solutions) ont été appliqués : un mouvement purement aléatoire et un mouvement qui crée de nouvelles solutions en combinant des parties des deux solutions. Les auteurs ont comparé les résultats obtenus avec les résultats de plusieurs algorithmes existants.

Indumathy et al. ont adapté dans (Indumathy et al., 2015) un algorithme très récent inspiré des phénomènes naturels dit l'algorithme de la recherche coucou (en anglais, *Cuckoo Search Algorithm*) pour maximiser la fonction objectif F_1. Cet algo-

rithme à base de population s'inspire du comportement de reproduction intelligent des oiseaux parasites appelés "Coucous" et du comportement du vol de Lévy (marche aléatoire). La population de l'algorithme est formée par un ensemble de solutions possibles appelées "nids" ou "coucous". Au cours du processus de la recherche, de nouveaux nids sont créés en fonction des nids courants et du vol Lévy. L'évaluation de la qualité de nouveaux nids sert à sélectionner certains d'entre eux et, en contrepartie, à abandonner certains nids courants. Les auteurs ont comparés leur algorithme métaheuristique avec quelques variantes de l'algorithme PSO.

Très récemment, Gheraibia et al. ont adapté un algorithme métaheuristique très récent pour maximiser le chevauchement entre les fragments successifs dans l'ordre d'assemblage (Gheraibia et al., 2016). Cet algorithme qui appelé PeSOA (de l'anglais, *Penguins Search Optimisation Algorithm*) s'inspire du comportement de chasse collective observé chez les pingouins. Il commence par la génération d'une population aléatoire de solutions. Ensuite, cette population initiale est subdivisée de façon aléatoire en plusieurs groupes de même taille. A chaque itération d'algorithme, les solutions de chaque groupe sont améliorées séparément en autorisant la modification des positions d'un sous-ensemble de fragments uniquement, appelés fragments actifs. La division de l'ensemble de fragments en plusieurs petits sous-ensembles est réalisée de façon aléatoire. Chaque solution (pingouin) dans chaque groupe est améliorée par une procédure de recherche locale en considèrent ses propres fragments actifs uniquement. A la fin chaque itération, pour chaque groupe, les nouvelles solutions obtenues sont combinées pour construire une seule solution locale améliorée. Dans cette étape, chaque pingouin associé à chaque solution dans les groupes est mis à jour, en fonction de l'amélioration réalisée et d'un paramètre dit réserve d'oxygène. Celui-ci aide le pingouin à décider de quitter ou bien rester dans son groupe, et à déterminer le nombre de transitions à effecteur par itération durant la recherche locale. La solution globale est construite en combinant les meilleures parties des solutions locales des groupes.

Un autre algorithme fondé sur l'intelligence en essaim a été proposé dans (al Rifaie et al Rifaie, 2015) pour maximiser les chevauchements entre fragments successifs. Cet algorithme (dit, en anglais, *Stochastic Diffusion Search*) s'inspire du comportement de recherche de nourriture d'une espèce spécifique de fourmis.

4.4.5 Autres techniques

Dans (Kubalik et al., 2010) une approche itérative a été appliquée pour maximiser la fonction objectif F_1. Cette approche utilise un algorithme évolutionnaire pour chercher, à chaque itération d'optimisation, une bonne modification de la solution courante. Une modification représente une suite de mouvements primitifs (à appliquer à la solution courante) définis spécialement pour le problème d'assemblage de fragments d'ADN. Une modification est évaluée en considérant le degré d'amélioration/dégradation qu'elle réalise sur la solution courante. Après l'arrêt de l'algorithme évolutionnaire, si la modification retournée est améliorante ou (au moins) non-dégradante, la solution modifiée par cette modification est considérée comme la nouvelle solution courante de l'itération suivante. Pour améliorer la performance de l'algorithme évolutionnaire, le nichage de la population dans l'espace des paramètres est forcé à deux niveaux : au niveau de la sélection de parents géniteurs et au niveau de la sélection de survivants d'une génération (remplacement). Les niches sont formées en fonction du nombre de mouvements élémentaires dans les modifications. Les stratégies de sélection/remplacement utilisées servent à orienter la recherche vers des modifications avec des nombres de mouvements maximaux. Les expériences réalisées ont montré la bonne performance de cette approche en comparaison avec d'autres assembleurs.

Dans (Minetti et al., 2012) une technique combinant un algorithme de recuit simulé avec des opérateurs génétiques de croisement a été proposée. L'objectif était de maximiser la fonction objectif F_1 et en même temps minimiser le nombre de contigs. Dans cette approche, les solutions voisines à la solution courante de l'algorithme de recuit simulé sont générées par l'application d'opérateurs génétiques. A chaque itération d'algorithme, deux solutions modifiées sont obtenues par l'application d'un opérateur d'inversion à la solution courante, puis une troisième solution est générée par l'application d'un opérateur de croissement OX sur ces deux solutions modifiées. A la fin de l'itération, cette troisième solution remplace la solution courante si elle est meilleure ou elle est acceptée selon la règle de l'algorithme de recuit simulé (règle basée sur la distribution de Boltzman).

Plus récemment, un nouvel algorithme métaheuristique dit l'algorithme de recherche par harmonies (en anglais, *Harmony search algorithm*) a été adapté dans (Ülker, 2016) pour résoudre le problème d'assemblage de fragments d'ADN. Cet algorithme s'inspire du comportement des musiciens de jazz pour générer la meilleure harmonie musicale. Il manipule une population d'harmonies représentant des solutions possibles sous forme de vecteurs réels. Afin de transformer la représentation réelle en une permutation de fragments, comme pour les approches appliquant l'algorithme de PSO décrites dans la section précédente, la réglé de SPV est utilisée dans cette adaptation. La première étape de l'algorithme consiste à générer une population initiale d'harmonies. Ensuite, l'algorithme génère de nouvelles harmonies par l'application de ses opérateurs. Ces opérateurs considèrent la sélection d'harmonies courantes à partir de la population, des harmonies générées aléatoirement, et une règle de modification de variables d'harmonies. L'étape suivante consiste à remplacer la mauvaise harmonie dans la population par la meilleure harmonie obtenue. Les étapes de modification et replacement sont répétées jusqu'à satisfaire un certain critère d'arrêt. Cet algorithme a été testé sur un petit jeu de données réels.

4.5 Benchmarks

Dans cette section, nous présentons les détails liés aux benchmarks (jeux de tests) les plus connus du problème d'assemblage de fragments d'ADN.

Une collection de quinze (15) instances simulées a été utilisée avec succès pour tester de nombreux algorithmes d'assemblage dans la littérature. Ces problèmes benchmarks ont été générés à partir de séquences obtenues du site web NCBI : National Center for Biotechnology Information[3]. Cette collection et d'autres sont présentées dans (Mallén-Fullerton et al., 2013), et disponibles en ligne à l'adresse suivante : `http://chac.sis.uia.mx/fragbench/`.

Le premier ensemble d'instances (instances GenFrag) a été généré par une application UNIX/C dite GenFrag (Engle et Burks, 1993). Cette application permet de générer de façon aléatoire un ensemble de fragments chevauchants en fonction d'un

[3] `http://www.ncbi.nlm.nih.gov/`

certain nombre de paramètres (la langueur moyenne de fragments et la couverture de la séquence ciblée). Quatre séquences cibles ont été utilisées pour générer cet ensemble d'instances :

1. Une répétition de type II de la fibronectine appartenant au complexe majeur d'histocompatibilité humain de classe II (HUMMHCFIB), avec le numéro d'accession X60189, qui a une longueur de 3,835 bases.

2. Une apolipoprotéine humaine HUMAPOBF, avec le numéro d'accession M15421, qui a une longueur de 10,089 bases.

3. Le génome complet du bactériophage lambda, avec le numéro d'accession J02459, qui a une longueur de 20k bases.

4. Neurospora crassa (une espèce de champignon), avec le numéro d'accession BX842596, qui a une longueur de 77,292 bases.

Le second ensemble d'instances (instances DNAgen) a été généré par une autre application dite DNAgen (Minetti et al., 2012). Les séquences ciblées de cette collection, qui ont été aussi obtenues du site web NBCI, correspondent à une bactérie humaine ATCC 49176, avec des numéros d'accession de ACIN02000001 à ACIN02000026.

Le tableau 4.I contient les caractéristiques des ensembles de fragments générés pour chaque séquence cible.

Il est à noter que les instances avec les numéros d'accession BX842596 et ACIN sont très difficiles à résoudre, puisqu'elles sont générées à partir de séquences très longues en utilisant de petites/moyennes valeurs de couverture et une coupure (paramètre *cutoff*) très restrictive.

Calcul de chevauchements entre fragments Après la génération d'un ensemble de fragments, l'étape suivante consiste à calculer les chevauchements approximatifs entre les fragments. La recherche des chevauchements se fait par un algorithme basé sur la programmation dynamique pour l'alignement semi-global de séquences. L'algorithme de Smith-Waterman (Smith et Waterman, 1981) est usuellement utilisé pour faire cet alignement. La configuration la plus utilisée de cet algorithme est comme

Tableau 4.I – Caractéristiques de 15 benchmarks. Les numéros d'accession sont utilisés pour nommer les instances.

Instances	Paramètres			
	Couverture	Longueur moyenne de fragments	Nombre moyen de fragments	Longueur de la séquence d'origine
Instances GenFrag				
X60189(4)	4	395	39	3,835
X60189(5)	5	386	48	
X60189(6)	6	343	66	
X60189(7)	7	387	68	
M15421(5)	5	398	127	10,089
M15421(7)	7	383	177	
J02459(7)	7	405	352	20,000
BX842596(4)	4	708	442	77,292
BX842596(7)	7	703	773	
Instances DNAgen				
ACIN1	26	182	307	2,170
ACIN2	3	1002	451	147,200
ACIN3	3	1001	601	200,741
ACIN5	2	1003	751	329,958
ACIN7	2	1003	901	426,840
ACIN9	7	1003	1049	156,305

suit : 2 pour un match, −3 pour un mismatch, et −2 pour un gap. Un exemple de l'alignement semi-global avec cette configuration est donné dans la figure 4.4. Dans cet exemple, le score final d'alignement des deux fragments d'ADN S et T est déterminé, et la séquence T est changée à "CCT_T" où le symbole '_' représente un gap. L'alignement doit être appliqué sur toutes les paires de fragments et leurs complémentaires, dû au fait que l'orientation d'un fragment n'est pas préalablement connue. Les longueurs de plus longs chevauchements sont collectées dans une matrice de sorte que l'on peut ensuite chercher le chevauchement entre deux fragments quelconques. La procédure entière est facile à implémenter, mais nécessite un temps de calcul relativement important pour les instances de grande taille. Beaucoup de recherche a été fait pour améliorer le temps de calcul de chevauchements (Myers, 2014).

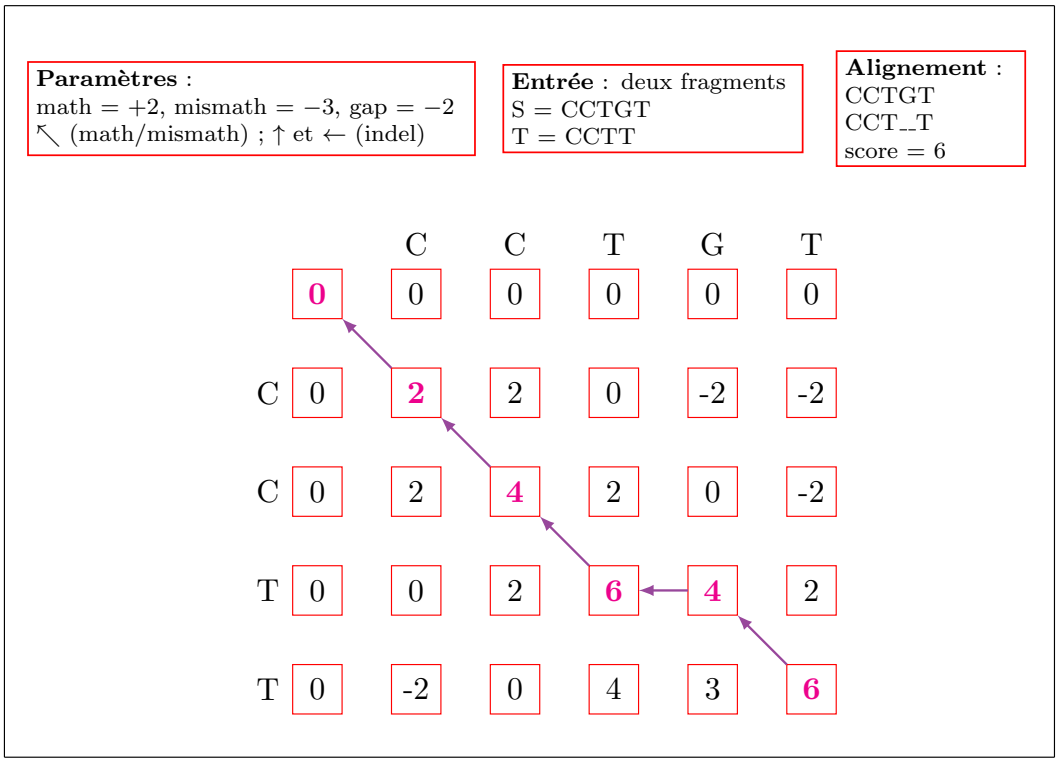


Figure 4.4 – Illustration du calcul d'un score d'alignement par un algorithme de l'alignement semi-global.

4.6 Conclusion

Nous avons présenté dans ce chapitre un état de l'art sur le problème d'assemblage de fragments d'ADN et les méthodes de résolution de ce problème. Nous avons introduit dans un premier temps les principales définitions et notions bio-informatique nécessaires à la compréhension de l'assemblage des séquences. Puis, dans un deuxième temps, nous avons abordés brièvement le lien entre l'assemblage de génomes et certains types de graphes. Ensuite, nous avons passé en revue les travaux portant sur les algorithmes (méta)heuristiques pour la résolution approchée de ce problème d'optimisation combinatoire $\mathcal{NP}$-*difficile*. Ces algorithmes d'assemblage ont été regroupés en quatre catégories : l'algorithme heuristique PALS et ses variantes, les approches basées sur les algorithmes génétiques, les approches basées sur l'intelligence en essaim, et d'autres techniques. Enfin, nous avons présenté le jeu de données artificiel principalement utilisé dans la littérature pour tester les nouveaux algorithmes d'assemblage.

Nous avons développé, durant cette thèse, de nouveaux algorithmes d'assemblage qui correspondent à des variantes de l'algorithme PALS, que nous allons présenter dans le chapitre 5 suivant. Ces variantes ont été conçues pour améliorer les performances de cet algorithme heuristique. Elles ont été testées en utilisant le jeu de données présenté dans la section précédente, et ont été comparées entre eux et avec la littérature.

Chapitre 5

Algorithmes performants pour l'assemblage de fragments d'ADN

Dans ce chapitre nous proposons et évaluons deux modifications à l'algorithme heuristique PALS, une méthode de recherche locale spécifique au problème d'assemblage de fragments d'ADN. Les travaux présentés dans ce chapitre ont fait l'objet d'une publication dans le journal *Soft Computing* (Ben Ali et al., 2017).

La structure de ce chapitre est la suivante. Tout d'abord, nous formulons les motivations et les objectifs de ce travail dans la section 5.1. Ensuite, dans la section 5.2, nous présentons en détail l'algorithme PALS original. Nos propositions en vue d'améliorer la performance de cet algorithme sont présentées dans la section 5.3. Dans la section 5.4, nous analysons et comparons les résultats expérimentaux obtenus. Finalement, dans la section 5.5, nous concluons en résumant les avantages de nos propositions.

5.1 Problématique et Objectifs

Comme nous l'avons vu dans le chapitre précédent, la détermination de la séquence d'ADN d'un organisme (ou le processus de séquençage d'ADN) représente une étape cruciale dans tout projet de génomique. En effet, cette étape conditionne fortement les succès des tâches postérieures (i.e., l'identification des régions codantes et la prédiction des gènes). La séquençage *shotgun* est la stratégie la plus utilisée dans les projets de séquençage de génomes (Pop, 2004). Cette stratégie crée d'abord plusieurs copies de l'ADN initial. Puis, elle coupe de façon aléatoire chaque copie en plusieurs fragments, suffisamment courts pour être séquencés par un séquenceur. Ensuite, tous les fragments séquencés (ou *lectures*) sont assemblés l'un avec l'autre pour construire la séquence parent initiale. Cependant, après le découpage de la séquence initiale du génome en un nombre très grand de fragments, la position et la copie de génome de chaque fragment sont oubliées. De plus, du fait qu'un brin d'ADN peut être lu dans les deux sens, on ne connait pas aussi l'orientation du fragment. La détermination de l'ordre et de l'orientation de chaque fragment conduit au problème connu dans la littérature sous le nom de problème d'assemblage de fragments d'ADN. En fait, il s'agit donc d'un problème d'optimisation combinatoire, connu comme étant $\mathcal{NP}$-difficile (Pevzner, 2000) : pour n fragments, il y a $2^n n!$ configurations possibles, où 2^n est le nombre de toutes les combinaisons possibles en termes d'orientation de fragments, et $n!$ est le nombre de toutes les permutations possibles des fragments.

Le problème d'assemblage de fragments d'ADN a été traité par plusieurs approches. Les approches les plus communes sont basées sur des heuristiques gloutonnes et utilisent des structures de données de type graphe. Les algorithmes gloutons ont été utilisés dans la plupart des packages les plus populaires, comme PHRAP (Green, 1994), Celera assembler (Myers, 1995), TIGR assembler (Sutton et al., 1995), STROLL (Chen et Skiena, 1997), CAP3 (Huang et Madan, 1999), et EULER (Pevzner, 2000). D'autres approches utilisent des algorithmes heuristiques et métaheuristiques pour pouvoir traiter des instances plus difficiles et de grande taille. L'algorithme heuristique la plus connu dans la littérature est l'algorithme PALS (Alba et Luque, 2007), qui est très performant pour trouver des solutions précises et est plus rapide que les

techniques précédemment existantes. Dans la section 4.4, nous avons passé en revue plusieurs travaux dans la littérature portent sur les algorithmes heuristiques et métaheuristiques. Cette revue de la littérature présente plusieurs types d'algorithmes : des algorithmes génétiques (Hughes et al., 2016, Kikuchi et Chakraborty, 2006, 2012, Li et Khuri, 2004, Minetti et al., 2008a, b, Nebro et al., 2008, Parsons et al., 1995), des algorithmes basés sur l'intelligence par essaim (Ezzeddine et al., 2014, Firoz et al., 2012, Gheraibia et al., 2016, Huang et al., 2015, 2016, Indumathy et al., 2015, Mallén-Fullerton et Fernández-Anaya, 2013, Meksangsouy et Chaiyaratana, 2003, Rajagopal et Maheswari Sankareswaran, 2015, Verma et al., 2012), des métaheuristiques hybrides (Alba et Dorronsoro, 2009b, Alba et Luque, 2008, Dorronsoro et al., 2008, Minetti et al., 2014), et d'autres techniques (Kubalik et al., 2010, Minetti et al., 2012, Ülker, 2016).

Comme nous l'avons déjà mentionné, l'algorithme heuristique de recherche locale PALS est bien connu dans la littérature. Bien que cet algorithme soit plus performant que plusieurs autres assembleurs, sa performance peut être améliorée, surtout sur des instances de grande taille. En fait, la stratégie de recherche de l'algorithme PALS conduit à une convergence rapide vers des optimums locaux. Effectivement, l'algorithme doit être capable de diversifier la recherche pour trouver des solutions beaucoup plus précises. Pour offrir cette capacité d'exploration, des variantes ont été proposées pour cet algorithme. Alba et Luque ont proposé dans (Alba et Luque, 2008) d'appliquer l'algorithme PALS comme opérateur de mutation dans un GA. Dans ce schéma hybride, la population de GA est exploitée pour fournir à l'algorithme PALS de multiples configurations de départ. Deux autres approches similaires ont été présentées dans (Alba et Dorronsoro, 2009b, Dorronsoro et al., 2008). Pour le même but, Minetti et al. ont récemment proposé dans (Minetti et al., 2014) une méthode distribuée combinant l'algorithme PALS avec un algorithme de recuit simulé. L'objectif était de pouvoir traiter les données bruitées. Dans cette approche hybride, l'algorithme de recuit simulé est utilisé pour fournir des configurations de départ pour l'algorithme PALS, tandis que le modèle distribué est adopté afin de promouvoir la diversification de recherche. Ces approches hybrides qui combinent l'algorithme PALS avec des algorithmes métaheuristiques ont été décrites dans la section 4.4.2. Elles améliorent

la qualité des résultats, mais en contrepartie elles augmentent fortement le temps de calcul par rapport à l'algorithme PALS.

Dans ce travail, nous proposons deux modifications à l'algorithme PALS original afin d'améliorer sa performance. Le but visé par la première modification est d'éviter les phénomènes de convergence prématurée vers des optima locaux. La stratégie de sélection de mouvements à appliquer à la solution courante est modifiée de manière à ce qu'une grande amélioration soit subséquemment achevée. L'objectif visé par la seconde modification est de réduire le temps de calcul. Cette modification consiste à appliquer à chaque itération de l'algorithme plusieurs mouvements (au lieu d'un seul mouvement) pour améliorer la solution courante. Nous proposons aussi quelques alternatives possibles à cette approche et les comparons afin de tirer des conclusions.

5.2 Présentation détaillée de l'algorithme PALS

L'algorithme PALS (de l'anglais *Problem Aware Local Search*) est une méthode de recherche locale plus rapide et plus compétitif, développé pour résoudre le problème d'assemblage de fragments d'ADN par Alba et Luque (Alba et Luque, 2007). L'algorithme 5.1 représente le pseudo-code de cette méthode heuristique.

L'algorithme PALS améliore de façon itérative une seule solution. La solution est codée par une permutation d'entiers représentant les numéros des fragments, où tous deux fragments successifs se chevauchent. Cette représentation nécessite une liste de fragments assignés chacun à un identificateur (nombre entier) unique. A chaque itération de l'algorithme, la solution courante s est remplacée par une bonne solution dans son voisinage $\mathcal{N}(s)$. La génération de solutions voisines se fait en considérant l'application de mouvements à la solution courante. Un mouvement consiste à inverser l'ordre des fragments situés entre deux positions distinctes i et j dans la permutation s (i.e., inversion d'une sous-permutation). Ainsi, l'ensemble complet des solutions voisines est défini par la considération de tous les mouvements possibles (pour tout $1 \leq i < j \leq n$, où n est le nombre de fragments).

Le point clé de l'algorithme PALS est l'utilisation du nombre de contigs comme un critère principal pour juger la qualité d'un résultat d'assemblage de fragments d'ADN.

Algorithme 5.1 : PALS

```
1  s ← generateInitialSolution() {créer une solution de départ}
2  répéter
3  |  L ← φ
4  |  pour i ← 1 à n − 1 faire
5  |  |  pour j ← i + 1 à n faire
6  |  |  |  ⟨Δ_f, Δ_c⟩ ← calculateDelta(s, i, j) {voir algorithme 5.2}
7  |  |  |  si (Δ_c < 0) ou (Δ_c = 0 et Δ_f > 0) alors
8  |  |  |  |  L ← L ∪ ⟨i, j, Δ_f, Δ_c⟩ {Mémoriser tous les mouvements non
   |  |  |  |   détériorants}
9  |  |  |  fin
10 |  |  fin
11 |  fin
12 |  si L ≠ φ alors
13 |  |  ⟨i, j, Δ_f, Δ_c⟩ ← selectMovement(L) {Sélectionner un mouvement}
14 |  |  applyMovement(s, ⟨i, j, Δ_f, Δ_c⟩) {Améliorer la solution}
15 |  fin
16 jusqu'à pas de changement
17 retourner s
```

En revanche, les assembleurs classiques utilisent le niveau de chevauchement entre les fragments successifs comme mesures de qualité, en utilisant des fonctions comme celle donnée dans l'équation 4.2. L'utilisation de telles fonctions pourrait conduire à des situations indésirables dans lesquelles une solution obtenue peut être meilleure qu'une autre solution, avec un grand nombre de contigs (une mauvaise solution). Face à des situations où plusieurs solutions différentes présentant un même nombre de contigs, les auteurs de PALS ont utilisé le niveau de chevauchement comme un critère secondaire pour évaluer la qualité des solutions. Une solution s est considéré meilleure qu'une autre solution s' si et seulement si les conditions suivantes sont vérifiées :

$$nContigs(s) < nContigs(s') \ \textbf{ou} \ (nContigs(s) = nContigs(s') \ \textbf{et} \ F_1(s) > F_1(s')). \tag{5.1}$$

où la fonction $nContigs$, donnée dans l'équation 4.4, calcule le nombre de contigs, et la fonction F_1, donnée dans l'équation 4.2, calcule le niveau de chevauchement entre les fragments successifs.

Comme l'opération de calcul du nombre de contigs de chaque solution modifiée est coûteuse en temps de calcul, l'algorithme PALS fait appel à une évaluation incrémentale mesurant le nombre de contigs qui ont été créés ou supprimés pendant la manipulation des solutions tentatives. Pour chaque mouvement possible, la variation de la longueur totale de chevauchement, notée Δ_f, et la variation du nombre de contigs, notée Δ_c, entre la solution courante et la solution modifiée après l'application du mouvement sont calculées (voir algorithme 5.2). Le calcul de ces variations -exploite les invariants et- nécessite l'analyse des fragments affectés par le mouvement uniquement (i.e., fragments i, j, $i-1$ et $j+1$). La valeur de Δ_f est calculée par la soustraction de la longueur de chevauchement des fragments affectés de la solution courante de celle de la solution modifiée (instructions des lignes 3-4 de l'algorithme 5.2). La valeur de Δ_c est calculée en testant, après l'application d'un mouvement, si un contig courant est coupé ou non (les deux premières conditionnelles de l'algorithme 5.2) ou si deux contigs ont été assemblés ou non (les deux dernières conditionnelles de l'algorithme 5.2). Ce calcul de Δ_c est basé sur la valeur d'un paramètre dit *cutoff*, qui représente la longueur de chevauchement minimale pour considérer que deux fragments adjacents étant dans le même contig.

A chaque itération de l'algorithme, on calcule les deux critères Δ_f et Δ_c pour tous les mouvements possibles et stocke les mouvements acceptés dans une liste $\mathcal{L}$. On considère uniquement les mouvements qui améliorent la qualité de la solution : ceux qui réduisent le nombre de contigs et ceux qui maintiennent le nombre de contigs et ne diminuent pas le niveau de chevauchement entre les fragments adjacents (la condition de la ligne 7 de l'algorithme 5.1). Après avoir stocké les mouvements candidates dans la liste $\mathcal{L}$, on sélectionne un mouvement spécifique et l'applique à la solution courante pour produire une nouvelle solution améliorée. Ce processus est répété jusqu'à aucune amélioration sera possible (i.e., la liste $\mathcal{L}$ est vide).

Deux choses restent à déterminer pour compléter la description de l'algorithme PALS : comment générer la solution de départ (la procédure `generateInitialSolution`), et comment sélectionner un mouvement de la liste $\mathcal{L}$ à chaque itération (la procédure `selectMovement`). Les auteurs de l'algorithme PALS, ont conclu, après avoir comparé plusieurs choix possibles, que la bonne configuration est de générer la

Algorithme 5.2 : `calculateDelta`(s, i, j) function

```
1  Δc ← 0
2  Δf ← 0
   {Calculer la variation du score de chevauchement :}
   {Ajouter le score de chevauchement des fragments affectés de la
    solution modifiée}
3  Δf ← w_{s[i−1]s[j]} + w_{s[i]s[j+1]}
   {Supprimer le score de chevauchement des fragments affectés de la
    solution courante}
4  Δf ← Δf − w_{s[i−1]s[i]} − w_{s[j]s[j+1]}
   {Calculer la variation du nombre de contigs :}
   {Incrémenter le nombre de contigs si un contig est coupé}
5  si w_{s[i−1]s[i]} > cutoff alors
6  |  Δc = Δc + 1
7  fin
8  si w_{s[j]s[j+1]} > cutoff alors
9  |  Δc = Δc + 1
10 fin
   {Decrémenter le nombre de contigs si deux contigs sont fusionnés}
11 si w_{s[i−1]s[j]} > cutoff alors
12 |  Δc = Δc − 1
13 fin
14 si w_{s[i]s[j+1]} > cutoff alors
15 |  Δc = Δc − 1
16 fin
17 retourner (Δf, Δc)
```

solution initiale de façon aléatoire et d'appliquer le mouvement ayant la plus petite valeur de Δ_c. Dans le cas où il y a plusieurs mouvements avec la même valeur minimale de Δ_c, on parfait le choix avec le mouvement ayant la plus grande valeur de Δ_f (ce qui donne une recherche *agressive*). Plus formellement, un mouvement m donné par $\langle i, j, \Delta_c, \Delta_f \rangle$ est considèré meilleur (en termes de degré d'amélioration à réaliser sur la solution courante) qu'un autre mouvement m' donné par $\langle i', j', \Delta'_c, \Delta'_f \rangle$ si et seulement si les conditions suivantes sont vérifiées :

$$\Delta_c < \Delta'_c \ \textbf{ ou } \ (\Delta_c = \Delta'_c \ \textbf{ et } \ \Delta_f > \Delta'_f). \tag{5.2}$$

5.3 Deux modifications à l'algorithme PALS

Les inconvénients majeurs de l'algorithme PALS sont la dépendance de ses résultats à la configuration de départ et sa convergence prématurée vers des optimums locaux. Dans cette section, nous présentons nos propositions pour réduire ces phénomènes et ainsi améliorer la performance de l'algorithme.

5.3.1 Première modification : PALS2

L'idée principale de l'algorithme PALS est basée sur une double observation. D'une part, l'optimisation de la fonction objectif F_1 n'est pas équivalente à l'optimisation du nombre de contigs (l'objectif réel du problème). Les deux objectifs sont complémentaires dans un certain sens, mais maximiser la fonction objectif F_1 n'est pas toujours corrélé avec minimiser le nombre de contigs. D'autre part, l'objectif de minimiser le nombre de contigs définit pour toute instance de problème un paysage de recherche (*search landscape*) caractérisé par plusieurs solutions avec le même nombre de contigs, parce qu'il y a $2^n n!$ points dans l'espace de recherche avec seulement n nombres de contigs différents. Par conséquent, la recherche locale est susceptible de visiter de nombreux optima locaux. Pour ces raisons, Alba et Luque ont combiné dans leur algorithme PALS le nombre de contigs avec la fonction objectif F_1 pour évaluer les solutions voisines durant la recherche locale. Afin d'orienter la recherche vers des solutions minimisant le nombre de contigs, l'algorithme PALS choisit à chaque itération le mouvement avec la plus petite valeur de Δ_c, i.e., le mouvement qui réduit (Δ_c est égal -2 ou -1) ou maintient (Δ_c est égal 0) le nombre de contigs. Dans le cas où il y a plusieurs mouvements avec la même valeur minimale de Δ_c, l'algorithme choisit le mouvement avec la valeur maximale de Δ_f (i.e., celui qui augmente au maximum le chevauchement entre les fragments successifs). Cependant, cette sélection stricte de mouvements conduit à une convergence rapide vers une solution sous-optimale fermée pour laquelle aucune amélioration supplémentaire n'est possible.

Pour éviter ce genre de situation, nous proposons de modifier la stratégie de sélection d'un mouvement améliorant à appliquer à la solution courante à chaque itération de l'algorithme. Dans le nouveau l'algorithme, noté PALS2, on choisit toujours le

meilleur mouvement en termes d'amélioration du nombre de contigs, mais contrairement à l'algorithme PALS, on préfère le mouvement avec la plus petite valeur de Δ_f dans le cas où il y a plusieurs mouvements avec la même valeur minimale de Δ_c. De manière formelle, un mouvement m donné par $\langle i, j, \Delta_c, \Delta_f \rangle$ est considéré meilleur qu'un autre mouvement m' donné par $\langle i', j', \Delta'_c, \Delta'_f \rangle$ si et seulement si les conditions suivantes sont satisfaites :

$$\Delta_c < \Delta'_c \ \textbf{ou} \ (\Delta_c = \Delta'_c \ \textbf{et} \ \Delta_f < \Delta'_f). \tag{5.3}$$

En d'autres termes, dans le cas où le critère d'évaluation principal n'est pas discriminant, on discrimine les mouvements en termes de degré de susceptibilité de la solution après la modification à des améliorations supplémentaires en termes de nombre de contigs au cours d'itérations futures. Cette susceptibilité sera élevée par la sélection du mouvement réalisant la plus faible amélioration en termes de chevauchement. La stratégie avec laquelle l'algorithme PALS original applique les mouvements améliorants oriente rapidement la recherche vers une région prometteuse dans l'espace de solutions et, par la suite permet l'exploitation cette région. Tandis que la sélection moins stricte proposée lance dans un premier temps une exploitation de la région courante, oriente ensuite la recherche vers la région prometteuse suivante, et répète l'évolution.

Avec cette nouvelle stratégie de sélection de mouvements l'algorithme va examiner plus de solutions potentielles, et par conséquent, sa convergence sera plus lente qu'avec la stratégie originale. Nous avons besoin alors d'un mécanisme additionnel pour améliorer le temps de calcul, ce qui constitue notre deuxième objectif.

5.3.2 Seconde modification : PALS2-many

L'algorithme PALS applique à chaque itération un seul mouvement pour améliorer la solution courante. En effet, de cette manière, une grande quantité de calculs est refait à chaque itération. En d'autres termes, on réévalue les mêmes mouvements pour un certain nombre d'itérations successives. La plupart de ces mouvements candidats ont de grandes chances d'être appliqués dans les prochaines itérations. Pour éviter

la réévaluation de mêmes mouvements, nous proposons d'appliquer plusieurs mouvements en même temps à chaque itération de l'algorithme. Nous avons développé cette variante pour améliorer le temps de calcul de notre nouvel algorithme PALS2. A chaque itération, on sélectionne plusieurs mouvements non conflictuels (i.e., leurs fragments affectés sont différents) à partir de la liste $\mathcal{L}$ de tous les mouvements candidats, puis on les applique tous à la solution courante. Nous notons l'algorithme général par PALS2-many. Nous proposons quelques stratégies pour sélectionner un sous-ensemble de mouvements non conflictuels, qui sont :

- ***Sélection déterministe***. Les mouvements dans la liste $\mathcal{L}$ sont d'abord triés par ordre croissant selon la valeur de Δ_c, puis par ordre croissant selon la valeur de Δ_f. Suivant cet ordre, chaque mouvement est appliqué s'il n'est pas en conflit avec les mouvements précédemment appliqués. Cette variante de l'algorithme est notée PALS2-many*.

- ***Sélection probabiliste.*** Selon l'ordre précèdent, chaque mouvement est appliqué s'il n'est pas en conflit avec les mouvements précédemment appliqués et un nombre aléatoire généré dans l'intervalle $[0, 1]$ est supérieur à 0.5. Cette variante est notée PALS2-many(prob).

- ***Sélection N-probabiliste***. Comme la sélection précédente est non déterministe, plusieurs sous-ensembles de mouvements non conflictuels peuvent être générés, puis on sélectionne le meilleur sous-ensemble selon les deux critères de sélection Δ_c et Δ_f en considérant leurs sommes sur tous les mouvements dans le sous-ensemble. Cette variante est noté PALS2-many(N-prob).

- ***Sélection exacte***. En utilisant une méthode exacte pour résoudre le problème de sélection d'un sous-ensemble (en anglais, *subset selection problem* : SSP), le meilleur sous-ensemble est appliqué. Le problème de SSP est connu comme $\mathcal{NP}$-difficile, et par conséquent la recherche du meilleur sous-ensemble est assez couteuse en temps de calcul. La taille de l'espace de recherche à examiner est $2^{|\mathcal{L}|}-1$, où $|\mathcal{L}|$ est la taille de l'ensemble $\mathcal{L}$ à une certaine itération de l'algorithme. Cependant, les résultats de cette variante de PALS2-many peuvent être utilisés

pour analyser la qualité des résultats obtenus par les autres variantes. Cette variante est notée PALS2-many(exact).

Chacune de ces variantes de l'algorithme PALS2-many continue jusqu'à ce qu'aucune amélioration ne peut être réalisé sur la solution. Dans la section suivante, nous comparons leurs performances.

5.4 Résultats expérimentaux

Dans cette section, nous analysons l'impact de nos propositions présentées dans la section précédente. Nous commençons par présenter les jeux de données d'entrainement. Puis, nous étudions l'influence des deux modifications proposées sur la performance de l'algorithme PALS. Nous comparons ensuite les résultats obtenus avec les résultats des variantes de PALS existantes ainsi qu'avec les résultats d'autres assembleurs.

L'algorithme original et les algorithmes modifiés sont programmés en C++ et toutes les expériences ont été conduites sur un ordinateur standard équipé d'un processeur Intel Core i5 (2.5 GHz, 4 GB) avec Windows 8.1 comme système d'exploitation.

Du fait de la nature stochastique de nos algorithmes, nous avons exécuté chaque algorithme 30 fois par instance de test pour obtenir des résultats significatifs. Une autre solution de départ générée aléatoirement est prise à chaque exécution, et pour chaque instance de test tous les algorithmes commencent par les mêmes 30 solutions de départ. Pour comparer les résultats de différents algorithmes, nous avons procédé à des tests statistiques. Pour tous les tests statistiques effectués, le niveau de signification de 0.05 (p-value < 0.05) a été considéré. Dans un premier temps, nous avons utilisés des tests statistiques de Kolmogorov-Smirnov pour vérifier la normalité de la distribution des données (Gaussienne ou non). Pour tous les tests de Kolmogorov-Smirnov effectués dans notre étude, l'hypothèse nulle a été rejetée (i.e., les résultats n'étaient pas normalement distribués), et par conséquent, des tests de Kruskal-Wallis ont été appliqués pour comparer les résultats de différents algorithmes.

5.4.1 Instances de test

Pour tester nos algorithmes d'assemblage, nous choisissions une collection de 15 instances artificielles connues dans la littérature, que nous avons présentée dans la section 4.5. Il est à noter que les instances avec les numéros d'accession BX842596 et ACIN sont très difficiles à résoudre, puisqu'elles sont générées à partir de séquences très longues en utilisant de petites/moyennes valeurs de couverture et une coupure (paramètre *cutoff*) très restrictive. La combinaison de tels paramètres produit des instances très difficiles. En revanche, des séquences très longues ont été utilisées dans la littérature (Huang et Madan, 1999), mais elles ont été séquencées avec des couvertures élevées. La couverture mesure la redondance de l'information ; plus elle est grande, plus le problème d'assemblage est facile.

Dans toutes les expériences, nous fixons le paramètre de coupure à 30 (une valeur très élevée), ce qui fournit un filtre pour les chevauchements controuvés dues à des erreurs expérimentales.

Nous décrivons par la suite comment nous avons calculé la matrice de score qui associe des scores d'alignement (niveaux de chevauchement) à chaque parie de fragments. Notre méthode de calcul a donné les mêmes jeux de données sur lesquels l'algorithme PALS original et ses variantes existantes (présentées dans la section 4.4.2) ont été testés.

5.4.1.1 Alignement par paire

L'alignement par paire se base sur un algorithme utilisant la programmation dynamique. Pour chaque paire de fragments (r, s), cet algorithme retourne le meilleur chevauchement entre un suffixe de r et un préfixe de s. Rappelons que l'orientation de fragments est inconnue (i.e., les fragments d'ADN peuvent provenir de n'importe quelle des brins). De ce fait, si un fragment ne se chevauche pas avec tout autre fragment, il est encore possible que son complément inverse ait un tel chevauchement. D'ailleurs, nous supposons qu'un brin est lu dans un seul sens (les deux sens de lecture sont appelés *forward* et *reverse*). Donc, quatre scores d'alignement sont calculés pour chaque paire de fragments (r, s) :

- $Overlap(r, s)$,
- $Overlap(r, \overleftarrow{s})$,
- $Overlap(\overleftarrow{r}, s)$,
- $Overlap(\overleftarrow{r}, \overleftarrow{s})$.

où $\overleftarrow{r}$ désigne le complément inverse du fragment r. D'où les égalités suivantes :

- $Overlap(r, s) = Overlap(\overleftarrow{s}, \overleftarrow{r})$,
- $Overlap(r, \overleftarrow{s}) = Overlap(s, \overleftarrow{r})$.

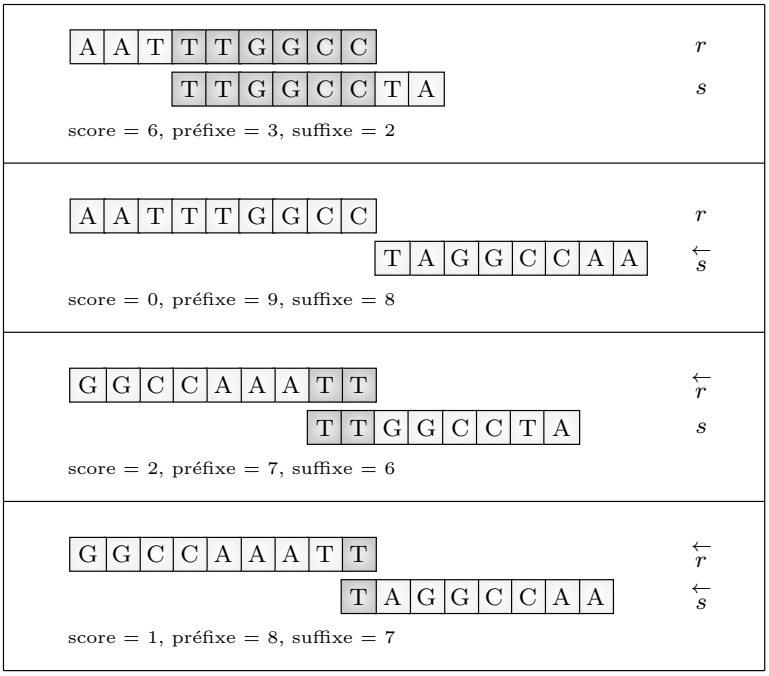
En profitant de ces relations d'appariement de fragments, comme illustré sur la figure 5.1, nous avons calculé les chevauchements entre les fragments d'une manière plus efficace. Pour n fragments, l'algorithme d'alignement est appliqué sur $4 \times n(n-1)/2$ paires, qui sont pour les quatre cas d'alignement par paire d'une matrice triangulaire supérieure. Il s'agit d'une matrice de score symétrique. Donc, après avoir complété une matrice triangulaire supérieure, sa transposée (avec l'échange du premier score avec le quatrième score) donne une matrice triangulaire inférieure, d'où on déduit une matrice de score complète.

Le chevauchement peut être évalué par plusieurs critères (Pop, 2004). Dans notre étude, nous avons utilisé la longueur de la région de chevauchement (i.e., chevauchement exact, voir figure 5.1).

5.4.2 Analyse des résultats expérimentaux

Dans ce paragraphe, nous analysons l'impact de nos propositions sur la performance de l'algorithme PALS en termes de précision et d'efficacité. Pour ce faire, nous étudions dans un premier temps séparément la première modification (i.e., l'algorithme PALS2). Puis, dans un deuxième temps, nous comparons les quatre variantes de l'algorithme général PALS2-many, présentées dans la section 5.3.2. Enfin, nous évaluons la performance de la seconde modification.

Quatre scores d'alignement pour la paire de fragments (r, s)



Quatre scores d'alignement pour la paire de fragments (s, r)

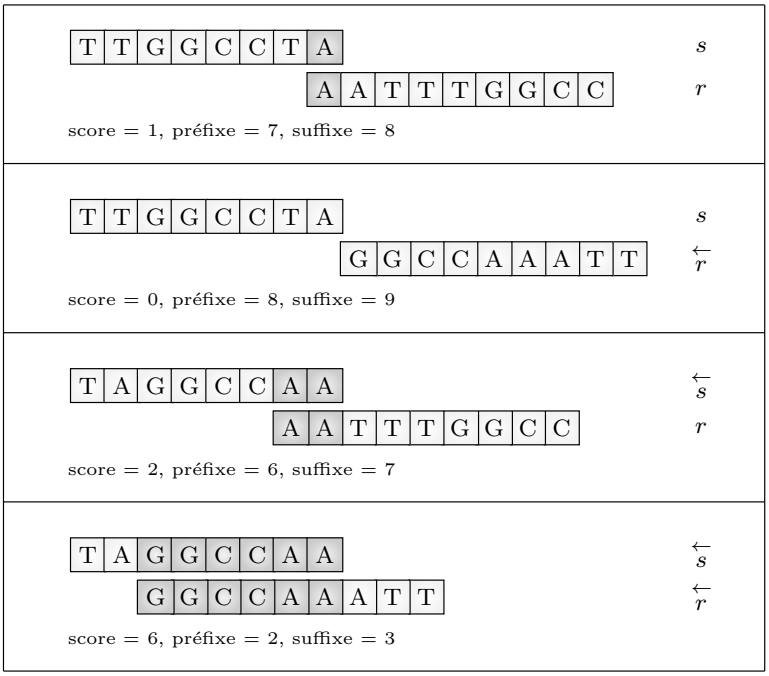


Figure 5.1 – Différentes cas de l'alignement de deux fragments d'ADN ; les scores correspondent aux longueurs de chevauchement.

Nous avons appliqué tous les algorithmes aux 15 instances de test listées dans le tableau 4.I. Nous avons comparé les algorithmes en termes de précision moyenne (i.e., solution moyenne : valeur moyenne de fitness et nombre moyen de contigs) et d'efficacité moyenne (i.e., temps de calcul moyen). Les tableaux 5.I, 5.II, 5.III et 5.IV regroupent les résultats expérimentaux moyens et les résultats de comparaison. Dans ces tableaux, pour chaque instance de test, les résultats qui sont statistiquement meilleurs (meilleurs que les autres et similaires entre eux) sont marqués en gras. Les différences statistiquement significatives entre les résultats dans tous ces tableaux sont notées par le signe "+", tandis que la non-significativité est notée par le signe "•".

Le tableau 5.I contient les résultats moyens en termes de précision de l'algorithme PALS original, de l'algorithme PALS2 et de la meilleure variante de l'algorithme PALS2-many (la variante notée PALS2-many*), tandis que le tableau 5.II présente leurs temps de calcul moyens. Les colonnes intitulées "Test" dans ces deux tableaux donnent le résultat de la comparaison par paires d'algorithmes, où "P", "P2" et "P2m*" sont des abréviations respectivement pour PALS, PALS2 et PALS2-many*. Les solutions moyennes des quatre variantes de l'algorithme PALS2-many sont affichées dans le tableau 5.III, tandis que leurs temps de calcul moyens sont donnés dans le tableau 5.IV. Les tableaux 5.II et 5.IV indiquent également, entre crochets, le nombre total moyen de mouvements appliqués par chaque algorithme sur chaque instance de test.

Pour une illustration visuelle des comportements des algorithmes PALS, PALS2 et PALS2-many*, nous avons tracé les courbes de convergence du nombre de contigs et de la valeur de fitness en fonction du nombre d'itérations. Les figures 5.2 et 5.3 présentent ces courbes pour les instances les plus difficiles. Ces courbes sont des exemples représentatifs pour le reste d'instances.

5.4.2.1 Impact de la première proposition

D'abord, nous analysons l'impact de la nouvelle stratégie de sélection d'un mouvement améliorant en comparant l'algorithme PALS original avec l'algorithme PALS2. Comme nous pouvons le voir dans le tableau 5.I, l'algorithme PALS2 donne les résultats les plus précis en termes de nombre de contigs par rapport à l'algorithme PALS original dans tous les cas. Ceci est un résultat très intéressant qui indique que

Tableau 5.I – Solutions moyennes (*valeur moyenne de fitness/nombre moyen de contigs*) obtenues par les trois algorithmes PALS, PALS2, PALS2-many* sur toutes les instances de test.

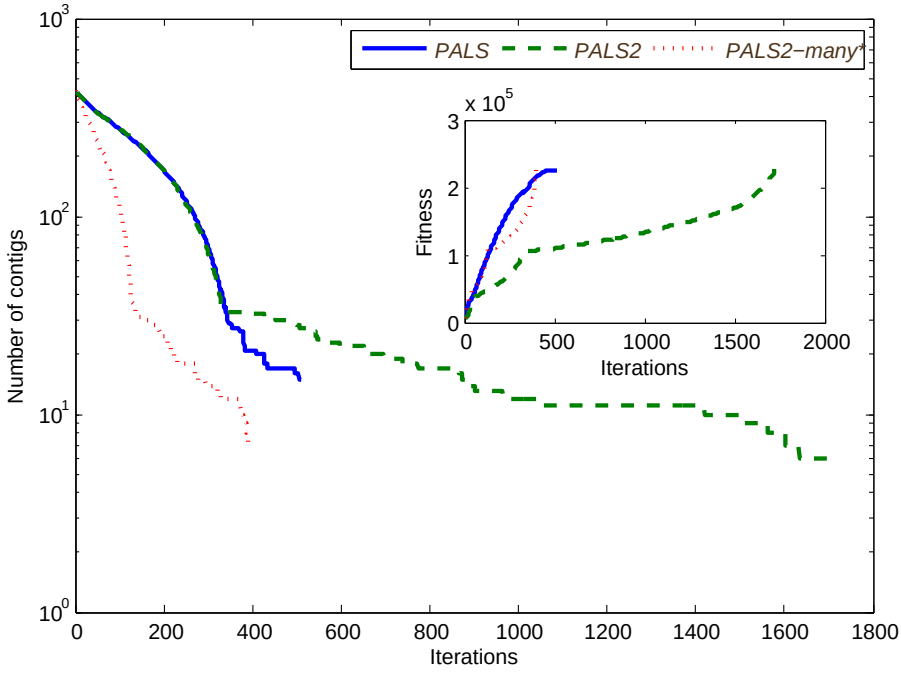
Instances	PALS	PALS2	PALS2-many*	Test fitness / contigs		
				(P, P2)	(P, P2m*)	(P2, P2m*)
X60189(4)	**11478** / **1.0**	**11478** / **1.0**	**11478** / **1.0**	• / •	• / •	• / •
X60189(5)	**13985** / 1.4	**13992** / **1.0**	**14007** / **1.0**	• / +	• / +	• / •
X60189(6)	**18229** / **1.2**	**18137** / **1.0**	**18132** / **1.0**	• / •	• / •	• / •
X60189(7)	**21136** / 1.3	20830 / **1.0**	20674 / **1.0**	+ / +	+ / +	• / •
M15421(5)	38480 / 4.2	**38553** / **1.3**	**38567** / **1.3**	+ / +	+ / +	• / •
M15421(7)	**54875** / 3.0	54608 / **2.0**	54476 / **2.0**	+ / +	+ / +	• / •
J02459(7)	**115563** / 3.7	**115670** / **1.1**	**115712** / **1.3**	• / +	• / +	• / •
BX842596(4)	225930 / 15.6	**227034** / **6.8**	**226895** / **6.9**	+ / +	+ / +	• / •
BX842596(7)	**441883** / 10.1	**441049** / **3.4**	**440506** / **3.2**	• / +	• / +	• / •
ACIN1	**46818** / 9.3	**46715** / **4.3**	**46650** / **4.2**	• / +	• / +	• / •
ACIN2	**144089** / 59.6	93394 / **31.6**	102535 / **31.8**	+ / +	+ / +	+ / •
ACIN3	**160775** / 72.3	76604 / **36.7**	98415 / **33.7**	+ / +	+ / +	+ / •
ACIN5	**156733** / 83.5	77048 / **29.2**	87093 / **28.8**	+ / +	+ / +	+ / •
ACIN7	**173093** / 78.2	69159 / **27.2**	83156 / **24.7**	+ / +	+ / +	+ / •
ACIN9	**319125** / 58.6	148640 / **13.8**	158016 / **13.7**	+ / +	+ / +	+ / •

le nouveau schéma d'exploration/exploitation de l'espace des solutions, promu par la nouvelle stratégie de sélection, est adéquat pour le problème d'assemblage de fragments d'ADN. D'après les tests statistiques réalisés sur les nombres de contigs des deux algorithmes, des différences significatives en faveur de l'algorithme PALS2 sont notées dans tous les cas (sauf pour les deux instances faciles, X60189(4) et X60189(6), pour lesquelles les deux algorithmes donnent le meilleur résultat).

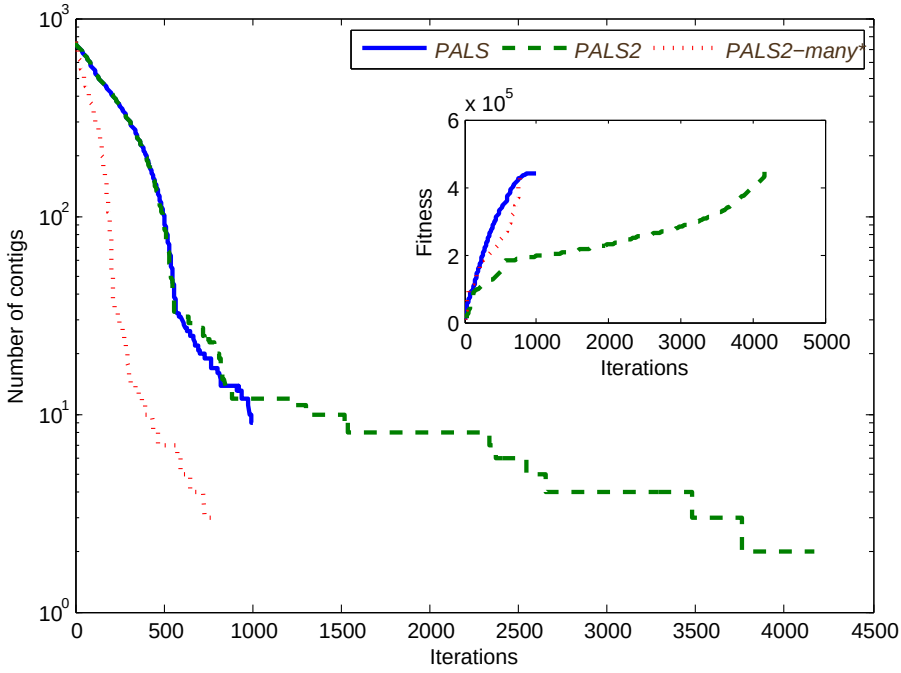
Quant aux valeurs de fitness de solutions, nous pouvons voir dans le tableau 5.I que pour la collection d'instances GenFrag, l'algorithme PALS est légèrement plus performant que l'algorithme PALS2 pour les instances X60189(7) et M15421(7), mais l'algorithme PALS2 donne les meilleurs résultats pour les instances M15421(5) et BX842596(4). Pour le reste d'instances GenFrag, les deux algorithmes retournent des résultats similaires. Sur la collection d'instances ACIN, les valeurs de fitness retournées par l'algorithme PALS sont significativement plus grandes que celles obtenues par l'algorithme PALS2 dans tous les cas, sauf pour l'instance ACIN1.

Les résultats des valeurs de fitness et les résultats des nombres de contigs confirment que dans certains cas il y a une corrélation négative entre les deux fonctions d'évaluation de qualité : fitness (chevauchement) versus contigs. A ce stade, nous notons que l'objectif final est la minimisation du nombre de contigs, tandis que l'utilisation du niveau de chevauchement est un moyen indirect pour atteindre cet objectif, qui, d'après ces résultats, n'est pas approprié pour tous les cas.

Du point de vue de la convergence des solutions, d'après les courbes des figures 5.2 et 5.3, nous pouvons observer que les deux algorithmes PALS et PALS2 ont la même vitesse de convergence du nombre de contigs dans la première phase de recherche, mais après cette phase l'algorithme PALS s'arrête tandis que l'algorithme PALS2 continue l'amélioration lentement. Quant à la convergence de la valeur de fitness, nous pouvons constater que l'algorithme PALS converge plus rapidement que l'algorithme PALS2. En résumé, les courbes de convergence de l'algorithme PALS2 montrent clairement le comportement attendu de la méthode proposée. Dans la première phase de recherche, pendant que l'algorithme cherche des solutions avec des nombres minimaux de contigs, les valeurs de fitness se croissent rapidement. Mais, dans la seconde phase, quand l'algorithme ne peut plus être guidé par le nombre de contigs (les nouvelles solutions

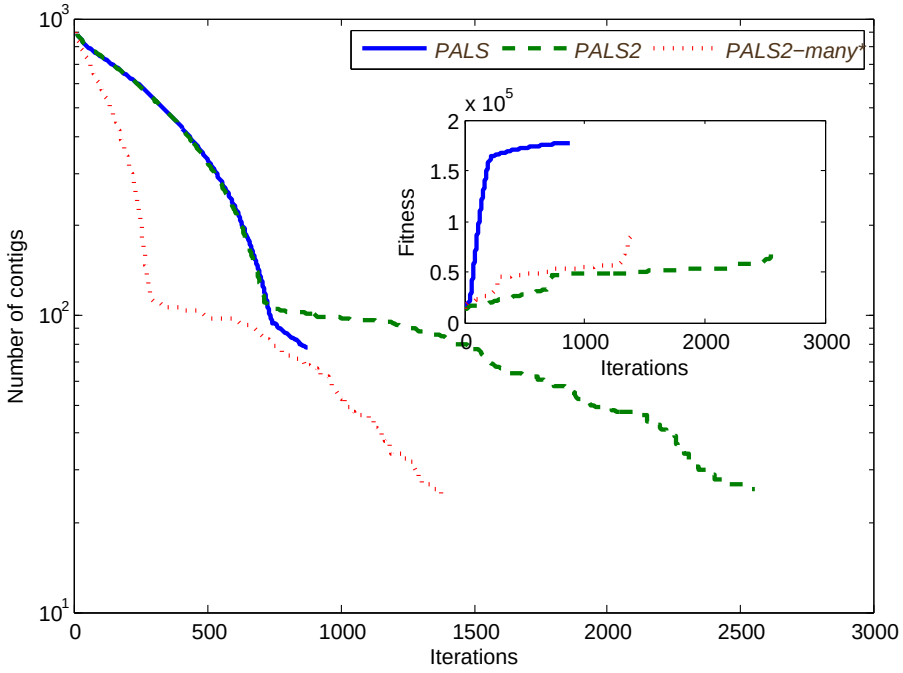


Instance BX842596(4)

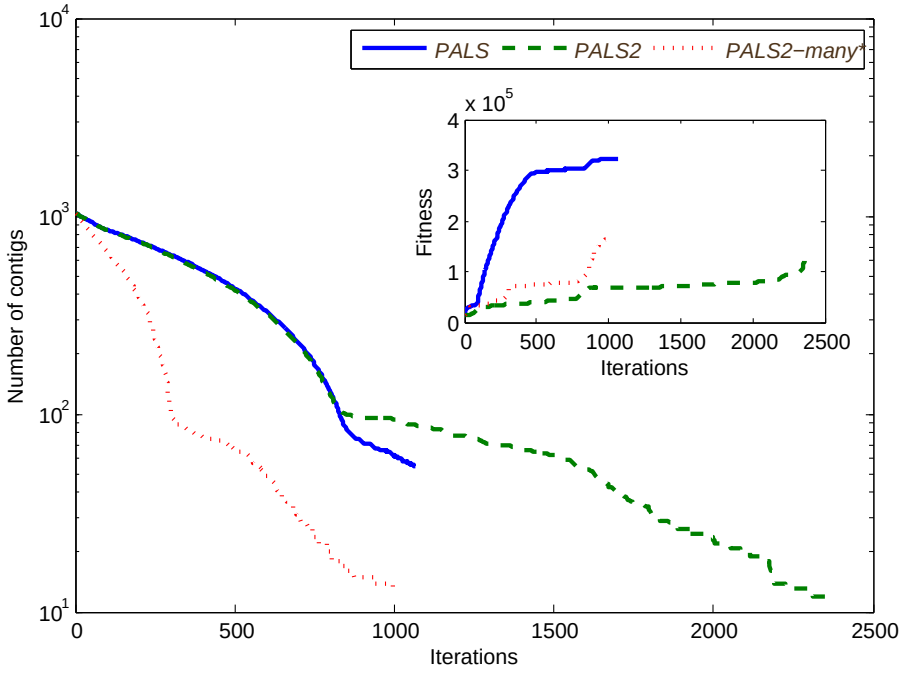


Instance BX842596(7)

Figure 5.2 – Courbes de convergence du nombre de contigs et de la valeur de fitness (à l'intérieur) pendant une exécution typique sur les instances BX842596(4) et BX842596(7).



Instance ACIN7



Instance ACIN9

Figure 5.3 – Courbes de convergence du nombre de contigs et de la valeur de fitness (à l'intérieur) pendant une exécution typique sur les instances ACIN7 et ACIN9.

ont les mêmes nombres de contigs que les solutions courantes), l'algorithme effectue une recherche profonde dans le voisinage (il sélectionne des solutions ayant des valeurs de fitness similaires aux valeurs de fitness des solutions courantes), ce qui permet une bonne (mais plus lente) exploration de cette région.

Le contenu du tableau 5.II confirme numériquement le fait que notre première proposition (la sélection moins stricte de mouvements) conduit à une convergence plus lente. Nous pouvons voir dans ce tableau que le nombre total moyen de mouvements appliqués par l'algorithme PALS2 est beaucoup plus grand que celui appliqué par l'algorithme PALS dans tous les cas. En conséquence, comme le confirment les résultats des tests statistiques réalisés sur les temps d'exécution moyens, l'algorithme PALS2 est plus lent que l'algorithme PALS, notamment sur les instances de grande taille.

5.4.2.2 Impact de la seconde proposition

Pour améliorer le temps de calcul de l'algorithme PALS2, comme présenté dans la section 5.3.2, nous proposons d'appliquer plusieurs mouvements améliorants en même temps à chaque itération d'algorithme (ce qui donne l'algorithme général PALS2-many). Avant de confirmer l'efficacité de cette idée, nous comparons d'abord les quatre variantes proposées. Le tableau 5.III contient les résultats moyens de ces variantes. Il contient aussi les résultats des tests statistiques effectués pour comparer les résultats (fitness/contigs) de chaque variante avec les meilleurs résultats obtenus.

Vu que les quatre variantes de l'algorithme général PALS2-many s'opèrent suivant la nouvelle stratégie de sélection de mouvements, d'après le tableau 5.III, il n'y a pas des différentes statistiquement significatives entre leurs résultats, sauf dans le cas de la variante PALS2-many(prob) pour les instances J02459(7), BX842596(4), BX842596(7), ACIN2, ACIN5, et ACIN7 en termes de nombre de contigs. Cette variante n'a pas obtenu les meilleurs résultats pour ces instances parce qu'avec sa sélection non-déterministe d'un sous-ensemble de mouvements non-conflictuels la probabilité de sélectionner un sous-ensemble de qualité inférieure est grande. Pour surmonter cet inconvénient, la variante PALS2-many(N-prob), qui applique aussi une sélection non-déterministe, génère plusieurs sous-ensemble de mouvements non-conflictuels, puis elle sélectionne le meilleur sous-ensemble entre eux.

Tableau 5.II – Temps de calcul moyens (en secondes) de trois algorithmes PALS, PALS2 et PALS2-many* [*Nombres moyens de mouvements appliqués* sont fournis].

Instances	PALS	PALS2	PALS2-many*	Test			Gain en temps de calcul (%)
				(P, P2)	(P, P2m*)	(P2, P2m*)	P2m*/P2
X60189(4)	**0.0005** [38]	**0.0021** [75]	**0.0015** [71]	•	•	•	–
X60189(5)	**0.0021** [50]	**0.0037** [136]	**0.0031** [127]	•	•	•	–
X60189(6)	**0.0021** [69]	0.0131 [219]	**0.0051** [197]	+	•	+	61.10%
X60189(7)	**0.0046** [73]	0.0173 [288]	0.0092 [264]	+	+	+	46.80%
M15421(5)	**0.0273** [142]	0.0691 [360]	**0.0282** [350]	+	•	+	59.20%
M15421(7)	**0.0834** [207]	0.2911 [769]	0.0959 [713]	+	+	+	67.06%
J02459(7)	0.6850 [434]	2.2385 [1510]	**0.5568** [1503]	+	+	+	75.13%
BX842596(4)	1.3005 [505]	4.0213 [1675]	**1.1885** [1624]	+	+	+	70.45%
BX842596(7)	7.8167 [961]	30.7683 [4093]	**6.2885** [3839]	+	+	+	79.56%
ACIN1	**0.4979** [371]	2.5943 [2188]	**0.4828** [1939]	+	•	+	81.39%
ACIN2	**1.5068** [443]	5.5187 [2074]	2.6224 [1768]	+	+	+	52.48%
ACIN3	**3.6422** [577]	15.3682 [3153]	5.7723 [1950]	+	+	+	62.44%
ACIN5	**8.0995** [729]	26.2182 [2961]	12.2349 [2420]	+	+	+	53.33%
ACIN7	**13.2868** [893]	33.8287 [2809]	16.0401 [2449]	+	+	+	52.58%
ACIN9	23.8007 [1047]	44.6704 [2420]	**19.9800** [2212]	+	+	+	55.27%

Tableau 5.III – Solutions moyennes (*valeur moyenne de fitness/nombre moyen de contigs*) obtnues par les quatre variantes de l'algorithme PALS2-many.

Instances	PALS2-many*		PALS2-many(prob)		PALS2-many(N-prob)		PALS2-many(exact)	
	Average sol.	Test	Average sol.	Test	Average sol.	Test	Average sol.	Test
X60189(4)	**11478 / 1.0**	(•/•)	**11478 / 1.0**	(•/•)	**11478 / 1.0**	(•/•)	**11478 / 1.0**	(•/•)
X60189(5)	**14007 / 1.0**	(•/•)	**14027 / 1.0**	(•/•)	**14013 / 1.0**	(•/•)	14001 / **1.0**	(+/•)
X60189(6)	**18132 / 1.0**	(•/•)	**18053 / 1.0**	(•/•)	**18134 / 1.0**	(•/•)	**18121 / 1.0**	(•/•)
X60189(7)	**20674 / 1.0**	(•/•)	**20495 / 1.0**	(•/•)	**20566 / 1.0**	(•/•)	**20555 / 1.0**	(•/•)
M15421(5)	**38567 / 1.3**	(•/•)	**38579 / 1.4**	(•/•)	**38529 / 1.3**	(•/•)	**38445 / 1.4**	(•/•)
M15421(7)	**54458 / 2.0**	(•/•)	**54668 / 2.0**	(•/•)	**54520 / 2.0**	(•/•)	**54448 / 2.1**	(•/•)
J02459(7)	**115716 / 1.3**	(•/•)	**115229** / 1.7	(•/+)	**115312 / 1.2**	(•/•)	**115648 / 1.2**	(•/•)
BX842596(4)	**226895 / 6.9**	(•/•)	**226654** / 7.5	(•/+)	**226971 / 6.6**	(•/•)	–	–
BX842596(7)	**440506 / 3.2**	(•/•)	**440525** / 3.7	(•/+)	**439885 / 3.6**	(•/•)	–	–
ACIN1	**46650 / 4.2**	(•/•)	**46785 / 4.4**	(•/•)	**46711 / 4.3**	(•/•)	–	–
ACIN2	**102535 / 31.8**	(•/•)	**104402** / 33.1	(•/+)	**102103 / 32.0**	(•/•)	–	–
ACIN3	**98415 / 33.7**	(•/•)	**98605 / 33.7**	(•/•)	**96823 / 34.5**	(•/•)	–	–
ACIN5	**87093 / 28.8**	(•/•)	**88787** / 31.0	(•/+)	83146 / **29.7**	(+/•)	–	–
ACIN7	**83156 / 24.7**	(•/•)	**86656** / 25.9	(•/+)	**83246 / 23.3**	(•/•)	–	–
ACIN9	**158016 / 13.7**	(•/•)	**159367 / 14.5**	(•/•)	**154911 / 14.3**	(•/•)	–	–

En termes de temps de calcul (voir tableau 5.IV), nous pouvons observer que pour les instances de plus petite taille (x60189), les résultats sont statistiquement similaires sauf dans le cas de la variante PALS2-many(exact) qui consomme plus de temps. Pour les instances de moyenne et grande tailles, nous pouvons voir que la variante PALS2-many* nécessite moins de temps de calcul que les autres variantes, avec des différences statistiquement significatives dans la plupart de cas.

Comme attendu, la variante PALS2-many(exact), qui applique une technique d'énumération pour trouver le meilleur sous-ensemble de mouvements non-conflictuels, demande beaucoup de temps de calcul. Rappelons que nous ne considérons cette variante que pour évaluer les performances des autres variantes, pas comme une technique appropriée pour le problème à cause de sa très grande exigence en temps de calcul. En dépit de cela, afin d'obtenir quelques résultats dans des temps raisonnables, nous avons réduit la complexité du problème de sélection d'un sous-ensemble en limitant la taille de l'ensemble $\mathcal{L}$ de tous les mouvements non détériorants à $\sqrt{n}$ (ce qui réduit le nombre de sous-ensembles envisageables). Lorsque l'ensemble $\mathcal{L}$ courant excède la taille limitée, le nouveau mouvement candidat remplace le plus mauvais mouvement dans $\mathcal{L}$ s'il est le meilleur en termes de nombre de contigs et de chevauchement. Avec cette réduction, une seule exécution sur l'instance BX842596(4) atteint 1371.44 secondes. Pour les instances les plus difficiles BX842596(7) et la collection ACIN, même pour réaliser une seule exécution, c'est un défi énorme (ce qui nécessitera plusieurs heures de calcul). Comme nous avons déjà indiqué, sur les instances de petite et moyenne tailles, les résultats de la variante PALS2-many(exact) et les résultats de la variante la plus rapide PALS2-many* sont statistiquement similaires en termes de précision.

Puisque la variante PALS2-many* (par la suite PALS2-many) est plus rapide que les autres variantes à plusieurs mouvements, nous comparons ses résultats avec les résultats de l'algorithme PALS2. Afin de faciliter la comparaison, nous avons présenté les résultats de comparaison dans les tableaux 5.I et 5.II. Nous avons affiché aussi les courbes de convergence de cette variante dans les figures 5.2 et 5.3.

Considérons d'abord l'influence de l'application de plusieurs mouvements sur la qualité des solutions. D'après la colonnes intitulée "Test/(P2, P2m*)" du tableau 5.I,

Tableau 5.IV – Temps de calcul moyens (en secondes) de quatre variantes de l'algorithme PALS2-many [*Nombres moyens de mouvements appliqués* sont fournis].

Instances	PALS2-many*		PALS2-many(prob)		PALS2-many(N-prob)		PALS2-many(exact)	
	Time (s)	Test	Time (s)	Test	Time (s)	Test	Time (s)	Test
X60189(4)	**0.0015** [71]	•	**0.0010** [69]	•	**0.0027** [70]	•	**0.0059** [75]	+
X60189(5)	**0.0031** [127]	•	**0.0041** [120]	•	**0.0031** [130]	•	**0.0062** [132]	+
X60189(6)	**0.0051** [197]	•	**0.0078** [193]	•	0.0095 [202]	+	0.0255 [223]	+
X60189(7)	**0.0092** [264]	•	**0.0101** [235]	•	**0.0126** [273]	•	0.033 [281]	+
M15421(5)	**0.0282** [350]	•	**0.0310** [330]	•	**0.0285** [351]	•	0.2488 [355]	+
M15421(7)	**0.0959** [713]	•	0.1063 [671]	•	0.1115 [728]	+	1.9458 [776]	+
J02459(7)	**0.5578** [1503]	•	0.6300 [1387]	+	0.6068 [1510]	+	152.5724 [1573]	+
BX842596(4)	**1.1885** [1624]	•	1.3192 [1539]	+	**1.2359** [1546]	•	–	–
BX842596(7)	**6.2885** [3839]	•	7.0015 [3621]	+	6.9364 [3859]	+	–	–
ACIN1	**0.4828** [1939]	•	**0.4995** [1831]	•	0.7948 [1894]	+	–	–
ACIN2	**2.6224** [1768]	•	2.9744 [1771]	+	3.7943 [1814]	+	–	–
ACIN3	**5.5610** [1950]	•	5.9942 [1891]	+	7.0953 [1873]	+	–	–
ACIN5	**12.2349** [2430]	•	13.8277 [2427]	+	16.5849 [2486]	+	–	–
ACIN7	**16.0401** [2449]	•	18.8467 [2577]	+	22.7809 [2705]	+	–	–
ACIN9	**19.9800** [2112]	•	22.2391 [2130]	+	23.2869 [2130]	+	–	–

nous pouvons observer qu'il n'y a pas des différences statistiquement significatives entre les résultats de l'algorithme PALS2-many et les résultats de l'algorithme PALS2 en termes de précision (fitness/contigs) dans tous les cas. De plus, d'après les courbes de convergence des deux algorithmes présentés dans les figures 5.2 et 5.3, nous pouvons constater que, pour les deux types d'instances de test, l'algorithme PALS2-many montre une convergence similaire à celle de l'algorithme PALS2, mais d'une manière compacte. De ce fait, nous pouvons conclure que les deux algorithmes ont des comportements similaires et obtiennent des résultats similaires en termes de qualité des solutions.

En termes d'efficacité, d'après les résultats présentés dans le tableau 5.II, nous pouvons remarquer que les nombres totaux de mouvements appliqués par les deux algorithmes PALS2 et PALS2-many sont similaires dans la plupart de cas. Cependant, l'algorithme PALS2-many est beaucoup plus rapide que l'algorithme PALS2 puisqu'il exploite, à chaque itération, le même calcul des variations des critères d'évaluation de qualité (Δ_f et Δ_c) pour appliquer plusieurs mouvements améliorants à la solution courante. La dernière colonne du tableau 5.II contient les facteurs de réduction des temps de calcul de l'algorithme PALS2-many par rapport l'algorithme PALS2. Comme nous pouvons le voir, la réduction des coûts de calcul est spectaculaire, avec un facteur de réduction supérieur à 0.5 sur les instances de grande taille. Les différences entre les temps d'exécution des deux algorithmes sont statistiquement significatives dans tous les cas, sauf pour les instances de plus petite taille pour lesquelles les temps d'exécution sont négligeables. En outre, en comparant l'algorithme PALS et l'algorithme PALS2-many, étonnamment, l'algorithme PALS2-many présente des temps de calcul similaires à ceux présentés par l'algorithme PALS, et même il est significativement plus rapide sur certaines instances de grande taille, bien qu'il applique des nombres de mouvements beaucoup plus grands (voir colonne intitulée "Test/(P, P2m*)" du tableau 5.II). En conclusion, l'idée d'appliquer plusieurs mouvements à chaque itération de l'algorithme conduit à une réduction significative des coûts de calcul tout en conservant la qualité des résultats.

5.4.2.3 Combinaison des deux propositions

En résumé, les résultats expérimentaux prouvent que les deux modifications apportées à l'algorithme PALS améliorent la qualité des solutions et le temps de calcul. D'une part, la nouvelle stratégie de sélection de mouvements permet d'éviter les optima locaux et d'envisager beaucoup plus de solutions possibles, mais ralentit le processus de recherche. D'autre part, l'application de plusieurs mouvements à chaque itération de l'algorithme réduit le temps de calcul tout en conservant la qualité élevée.

5.4.3 Comparaison de PALS2-many avec la littérature

Dans cette section, nous comparons le nouvel algorithme d'assemblage PALS2-many avec les variantes de l'algorithme PALS existantes et avec quelques assembleurs existants dans la littérature. Il est à noter que toutes les méthodes comparées ici utilisent la même valeur du paramètre *cutoff* (30).

La première comparaison est réalisée avec le quatre variantes de PALS existantes dans la la littérature (décrites dans la section 4.4.2), qui sont : GAPALS (Alba et Luque, 2008), SACMA (Dorronsoro et al., 2008), cGA-PALS (Alba et Dorronsoro, 2009a), et PHPALS (Minetti et al., 2014). Chacune des trois premières variantes combine un algorithme génétique avec l'algorithme PALS. La principale différence entre eux réside principalement dans le type d'algorithme génétique adopté. GAPALS utilise un algorithme génétique standard, SACMA utilise un algorithme génétique cellulaire, et cGA-PALS ajoute un mécanisme d'auto-adaptation à un algorithme génétique cellulaire. La variante PH-PALS combine l'algorithme PALS avec un algorithme de recuit simulé, qui est en plus évolue selon un modèle en îles. Cet algorithme hybride a été développé principalement pour pouvoir traiter les données bruitées. Nous considérons ici ses résultats pour des données non bruitées. La comparaison est exposée dans le tableau 5.V, où seules les plus grandes instances de test sont considérées, parce que, les trois premières variantes ont été testées uniquement sur les deux instances avec le numéro d'accession BX842596, et la quatrième variante a été testée en plus sur la collection d'instances ACIN. Dans ce tableau, la colonne intitulée "Avg Contigs" contient les nombres de contigs moyens, et la colonne intitulée "Avg t (s)" donne les temps de

Tableau 5.V – Comparaison de PALS2-many avec les variantes de PALS existantes dans la littérature. Le symbole "–" indique que l'information n'est pas fournie dans l'article correspondant.

Instances	PALS2-many		GA-PALS Alba et Luque (2008)		SACMA Alba et Luque (2008)		cGA-PALS Dorronsoro et al. (2008)		PH-PALS Minetti et al. (2014)	
	Avg Contigs	Avg t (s)	Avg Contigs	Avg t (s)	Avg Contigs	Avg t (s)	Avg Contigs	Avg t (s)	Avg Contigs	Avg t (s)
BX842596(4)	6.9	**1.19**	4.71	298.06	**1.00**	422.60	**1.06**	280.99	4.5	60.00
BX842596(7)	3.2	**6.29**	3.69	1531.31	**1.01**	1769.35	**1.00**	2334.30		
ACIN	**22.8**	**9.52**	–	–	–	–	–	–	159.5	60.00

calcul moyens. En ce qui concerne la variante PH-PALS, les résultats correspondent aux moyennes par groupe d'instances. Chacune des trois premières variantes a été codée en Java et exécutée sur un ordinateur équipé d'un processeur Pentium IV 2.8 GHz, tandis que la quatrième variante a été programmée en C++ et exécuté sur un cluster de 12 ordinateurs dotés chacun d'un processeur Intel Core 2 Duo 3 GHz.

Comme nous pouvons le voir dans le tableau 5.V, les deux algorithmes hybrides SACMA et PH-PALS ont été capables de construire un seul contig presque dans toutes les exécutions pour les instances BX842596. Pour la collection d'instances ACIN, nous pouvons observer que notre algorithme PALS2-many donne des solutions avec des nombres de contigs plus petits en comparaison avec l'algorithme PH-PALS. Du point de vue de temps de calcul, les variantes de l'algorithme PALS déjà existantes sont très couteuses, en raison des multiples exécutions de l'algorithme PALS dans le schéma hybride. Quoi qu'il en soit, les résultats de notre algorithme PALS2-many sont vraiment bons, car ils amélioreront considérablement le temps d'exécution de ces algorithmes hybrides si on utilise l'algorithme PALS2-many au lieu de l'algorithme PALS original (Ceci pourrait constituer une prochaine étape de recherche).

Maintenant, nous comparons notre algorithme PALS2-many avec quelques assembleurs de la littérature : une méthode d'optimisation itérative notée POEMS (Kubalik et al., 2010), un algorithme basé sur l'appariement de motifs (ou, en anglais, *pattern matching*) noté PMA (Li et Khuri, 2004), et deux packages commercialement disponibles : CAP3 (Huang et Madan, 1999) et PHRAP (Green, 1994). La comparaison est faite en termes de meilleur résultat obtenu (et de résultat moyen, si disponible). Les résultats de ces algorithmes d'assemblage sont disponibles uniquement sur la première collection d'instances (GenFrag). D'après le contenu du tableau 5.VI, pour les instances de petite taille, tous les algorithmes comparés construisent un seul contig. Pour les instances de moyenne et grande tailles, nous pouvons remarquer une très petite différence entre les nombres moyens de contigs de l'algorithme PALS2-many et ceux de l'algorithme POEMS. Dans tous les cas, l'algorithme PALS2-many obtient le même (meilleur) nombre de contigs que les autres algorithmes. Nous n'avons pas inclus les temps d'exécution dans le tableau car, en général, ils ne sont pas fournis par les auteurs. Alba et Luque ont commenté dans (Alba et Luque, 2007) que les temps

Tableau 5.VI – Meilleurs nombres de contigs obtenus par PALS2-many et d'autres assembleurs.

Instances	PALS2-many	PEOMS Kubalik et al. (2010)	PMA Li et Khuri (2004)	CAP3 Huang et Madan (1999)	Phrap Green (1994)
X60189	1(1.0)	1(1.0)	1	1	1
M15421(5)	1(1.3)	1(1.9)	1	2	1
M15421(7)	2(2.0)	2(2.0)	2	2	2
J02459(7)	1(1.3)	1(1.0)	1	1	1
BX842596(4)	6(6.9)	6(6.9)	7	6	6
BX842596(7)	2(3.2)	2(3.3)	2	2	2

d'exécution de ces assembleurs vont de dizaines de secondes pour les instances les plus faciles à plusieurs heures pour les instances les plus difficiles. Comme indiqué dans la section 5.4.2.2, l'algorithme PALS2-many prend bien moins d'une seule seconde pour les instances les plus faciles et quelques secondes seulement pour les instances les plus difficiles.

Pour finir, il est à noter que nous n'avons pas comparé notre algorithme avec l'algorithme proposé récemment dans (Huang et al., 2015). La principale raison est que les objectifs sont différents : notre objectif est de minimiser le nombre de contigs tandis que l'objectif de ce travail est de maximiser le chevauchement entre les fragments successifs. Comme nous l'avons déjà dit, ces deux objectifs ne sont pas équivalents, et dans la plupart de cas ne sont pas corrélés. D'autre part, le calcul de chevauchements dans cette étude est effectué d'une manière différente et, par conséquent, les chevauchements ne sont pas comparables.

5.5 Conclusion

L'assemblage de fragments d'ADN conduit à un problème d'optimisation combinatoire très important dans le domaine de la bio-informatique des séquences. Il représente un défi stimulant, non seulement des points de vue applicatif et technique mais aussi de part les difficultés algorithmiques qu'il pose. Comme ce problème est $\mathcal{NP}$-difficile, des assembleurs plus performants sont nécessaires pour traiter des instances de grande taille. Pour la résolution approchée de ce problème, l'algorithme heuristique PALS est connu dans la littérature en raison de sa précision et son efficacité.

Dans cette étude, nous avons proposé deux modifications à l'algorithme PALS afin d'améliorer ses performances sur les instances complexes. D'abord, les critères de sélection d'un mouvement améliorant à appliquer à chaque itération sont reformulés de manière à ce qu'une grande amélioration soit postérieurement obtenue. Il s'agit d'une sélection moins stricte de mouvements. Les résultats obtenus avec la nouvelle stratégie de sélection sont plus précis, mais en contrepartie la convergence de l'algorithme devient plus lente. La seconde proposition consiste à appliquer à chaque itération de l'algorithme, au lieu d'un seul mouvement, plusieurs mouvements non conflictuels.

Les résultats expérimentaux ont montré que cette approche réduit considérablement le temps de calcul tout en conservant la précision des résultats. En résumé, la combinaison des deux modifications produit un nouvel algorithme plus performant, appelé PALS2-many, améliorant à la fois la précision et le temps de calcul. En comparaison avec la littérature, l'algorithme PALS2-many se révèle très complétif par rapport aux variantes existantes et à d'autres algorithmes d'assemblage.

Conclusion générale et Perspectives

Cette thèse porte sur des algorithmes efficaces pour la résolution de problèmes d'optimisation combinatoires NP-difficiles, avec deux contributions essentielles. La première contribution consiste à proposer un algorithme métaheuristique hybride pour la résolution efficace de problèmes multiobjectifs de grandes dimensions. Une hybridation entre un algorithme génétique multiobjectif et un algorithme d'optimisation par essaims de particules a été proposée. La seconde contribution concerne la résolution efficace du problème d'assemblage de fragments d'ADN. De nouvelles variantes d'un algorithme heuristique spécifique à ce problème ont été développées.

Principales contributions

Une métaheuristique multiobjectif hybride combinant un MOGA avec un algorithme de PSO Dans la première partie de ce mémoire nous nous sommes intéressés à l'optimisation multiobjectif par algorithmes génétiques. Nous avons vu dans le chapitre 2 que les algorithmes génétiques multiobjectifs (MOGAs) utilisant des approches Pareto et élitistes fournissent souvent des performances satisfaisantes. En particulier, nous avons vu que la combinaison d'un MOGA avec une autre technique d'optimisation peut conduire à un algorithme performant capable d'éviter les situations de convergence lente lors de la résolution de problèmes complexes à plus de deux objectifs. L'algorithme MOGA-PSO proposé dans cette thèse (présenté dans le chapitre 3) fait partie de ces algorithmes hybrides. Il combine un MOGA basé sur une approche Pareto et élitiste avec un opérateur de recherche basé sur l'algorithme de PSO. Dans cet algorithme hybride, l'opérateur basé sur l'algorithme de PSO est appliqué périodiquement sur une partie de la population archive de l'algorithme MOGA

pour optimiser une fonction d'utilité agrégeant tous les objectifs du problème traité. Deux variantes de l'algorithme MOGA-PSO ont été proposées et adaptées pour la résolution du problème du sac à dos multiobjectif. Les résultats expérimentaux ont montré une nette supériorité des algorithmes hybrides testés sur les algorithmes standards.

Algorithmes performants pour la résolution du problème d'assemblage de fragment d'ADN Dans la deuxième partie de ce mémoire nous nous sommes intéressés au problème d'assemblage de fragments d'ADN. Afin de résoudre ce problème d'optimisation combinatoire NP-difficile, nous avons vu dans l'état de l'art présenté dans le chapitre 4 que les algorithmes heuristiques et métaheuristiques apportaient des solutions satisfaisantes à ce problème du monde réel. Dans cet état de l'art, quatre classes de méthodes ont été séparées : l'algorithme heuristique PALS et ses variantes, les approches basées sur les algorithmes génétiques, les algorithmes basés sur l'intelligence en essaim, et d'autres techniques. Des variantes améliorées de l'algorithme PALS ont été proposées dans cette thèse (présentées dans le chapitre 5). Elles reposent sur une sélection moins stricte de mouvements à appliquer à la solution courante. Nous avons proposé aussi des stratégies permettant la diminution du temps de calcul de l'algorithme. Des expérimentations ont été réalisées sur les jeux de données les plus utilisés dans la littérature. Les résultats numériques ont montré de très bonnes performances en comparaison avec l'algorithme de base, les variantes existantes et d'autres algorithmes d'assemblage.

Perspectives

Les travaux réalisés au cours de cette thèse nous permettent d'ouvrir plusieurs perspectives pour améliorer les performances obtenues ou pour solutionner les questions restant posées. Nous envisageons quelques aspects qui nous semblent particulièrement intéressants à poursuivre et à approfondir. Ces perspectives sont présentées autour de deux axes correspondant aux deux principales contributions présentées.

Validation et adaptation de l'algorithme MOGA-PSO De point de vue validation, il nous semble très intéressant de développer d'autres variantes de MOGA-PSO pour la résolution d'autres problèmes combinatoires académiques ou réels. Pour ce faire, un nombre très important de fonctions combinatoires multiobjectifs de test sont disponible dans la littérature. Il serait également intéressant d'adapter et tester l'algorithme MOGA-PSO sur des problèmes d'optimisation à variables continues ou réelles. Plusieurs fonctions ont été proposées dans la littérature pour tester les performances des algorithmes multiobjectifs (Deb, 1999, Deb et al., 2005, Fonseca et Fleming, 1995, Vlennet et al., 1996, Zitzler et al., 2000).

Etude de la structure de paysage de recherche donnée par l'algorithme MOGA-PSO Comme nous avons déjà constaté dans la section 3.3.4, la procédure de gestion de l'archive pour les algorithmes hybrides (NSGA-PSO et SPEA-PSO) demande toujours moins de temps que pour les algorithmes standards (NSGA et SPEA). Ainsi, les structures de paysage de recherche données par les algorithmes hybrides sont plus faciles que celles données par les algorithmes standards. En d'autres termes, cette diminution du temps de calcul provient principalement aux caractéristiques des solutions élites : leurs rangs Pareto et les densités de la population autour d'elles. Une question restant ouverte est de savoir ces caractéristiques précises des solutions élites des algorithmes hybrides au cours de générations successives en comparaison aux solutions élites des algorithmes standards.

Extension de MOGO-PSO sur les problèmes d'optimisation à objectifs multiples Il nous semble intéressant d'étendre la stratégie hybride MOGA-PSO pour résoudre les problèmes d'optimisation à objectifs multiples (en anglais, *Many-objective optimization problems*) (Brockhoff et Zitzler, 2009, Singh et al., 2011). Ces sont les problèmes à plus de trois objectifs. Bien que nous ayons montré la supériorité de nos algorithmes hybrides sur des problèmes à quatre et à cinq objectifs, nous n'avons pas étudié leur comportement face à des problèmes complexes à plus de cinq objectifs.

Test et adaptation de l'algorithme PALS2-many Il nous semble très intéressant de test l'algorithme PALS2-many sur des problèmes artificiels ou réels de grandes dimensions (Mallén-Fullerton et al., 2013). Il nous paraît aussi intéressant de l'adapter pour pouvoir traiter les données bruitées, comme c'est le cas dans (Firoz et al., 2012, Minetti et al., 2014).

Hybridation de l'algorithme PALS2-many avec des algorithmes métaheuristiques Il serait certainement possible de combiner l'algorithme PALS2-many avec un algorithme métaheuristique. L'objectif de cette hybridation est d'éviter les situations de dépendance de résultats aux configurations initiales, et ainsi de fournir des résultats beaucoup plus précis sur des instances de grande taille.

Multiobjectivation du problème d'assemlage de fragments d'ADN Les résultats expérimentaux de l'algorithme PALS2-many ont montré que la minimisation du nombre de contigs et la maximisation du niveau de chevauchements ne sont pas souvent corrélées : un nombre minimal de contigs n'implique pas forcément un niveau maximal de chevauchement, et inversement. De ce fait, il nous semble très intéressant de *multiobjectiver* la formulation du problème, en minimisant le nombre de contigs et maximisant le niveau de chevauchement en même temps. L'objectif de cette multiobjectivation (*multiobjectivization*, (Jensen, 2003, Neumann et Wegener, 2006)) serait double. D'une part, la résolution du nouveau problème bi-objectif fournissait des solutions qui présentent le nombre minimal de contigs et maximisent le plus possible le niveau de chevauchement (i.e., une amélioration en termes de précision de résultats). D'autre part, cette multiobjectivation transformerait le paysage de recherche en un autre paysage plat dans laquelle les comportements d'algorithmes métaheuristiques seraient plus performants (i.e., une réduction de la complexité du problème original).

Bibliographie

DIMACSWorkshop on Combinatorial Methods for DNA Mapping and Sequencing, October, 1994. URL `http://dimacs.rutgers.edu/Workshops/DNAMapping/program.html`.

Mark D Adams, Susan E Celniker, Robert A Holt, Cheryl A Evans, Jeannine D Gocayne, Peter G Amanatides, Steven E Scherer, Peter W Li, Roger A Hoskins, Richard F Galle et al. The genome sequence of Drosophila melanogaster. *Science*, 287(5461):2185–2195, 2000.

Fatimah Majid al Rifaie et Mohammad Majid al Rifaie. Investigating Stochastic Diffusion Search in DNA sequence assembly problem. Dans *SAI Intelligent Systems Conference (IntelliSys), 2015*, pages 625–631. IEEE, 2015.

Enrique Alba et Bernabé Dorronsoro. Computing nine new best-so-far solutions for capacitated VRP with a cellular genetic algorithm. *Information Processing Letters*, 98(6):225–230, 2006.

Enrique Alba et Bernabé Dorronsoro. *Cellular genetic algorithms*, volume 42. Springer, 2009a.

Enrique Alba et Bernabé Dorronsoro. *Cellular genetic algorithms*, volume 42, chaptitre 15 - Bioinformatics : The DNA Fragment Assembly Problem, pages 203–210. Springer, 2009b.

Enrique Alba et Gabriel Luque. Performance of distributed GAs on DNA fragment assembly. Dans *Parallel Evolutionary Computations*, pages 97–115. Springer, 2006.

Enrique Alba et Gabriel Luque. A new local search algorithm for the DNA fragment assembly problem. Dans Carlos Cotta et Jano van Hemert, éditeurs, *Evolutionary Computation in Combinatorial Optimization*, volume 4446 de *LNCS*, pages 1–12. Springer, EvoCOP 2007, Valencia, Spain, April 11–13 2007.

Enrique Alba et Gabriel Luque. A hybrid genetic algorithm for the dna fragment assembly problem. Dans Carlos Cotta et Jano van Hemert, éditeurs, *Recent Advances in Evolutionary Computation for Combinatorial Optimization*, volume 153 de *Studies in Computational Intelligence*, pages 101–112. Springer, 2008.

Enrique Alba, Gabriel Luque et Sami Khuri. Assembling DNA fragments with parallel algorithms. Dans *Evolutionary Computation, 2005. The 2005 IEEE Congress on*, volume 1, pages 57–64. IEEE, 2005.

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

TK Attwood, A Gisel, E Bongcam-Rudloff et NE Eriksson. *Concepts, historical milestones and the central place of bioinformatics in modern biology : a European perspective*, volume 13. INTECH Open Access Publisher, 2011.

Zahra Naji Azimi. Hybrid heuristics for examination timetabling problem. *Applied Mathematics and Computation*, 163(2):705–733, 2005.

Vincent Barichard et Jin-Kao Hao. Genetic tabu search for the multi-objective knapsack problem. *Tsinghua Science and Technology*, 8(1):8–13, 2003.

Roberto Battiti et Giampietro Tecchiolli. The reactive tabu search. *ORSA journal on computing*, 6(2):126–140, 1994.

James C Bean. Genetic algorithms and random keys for sequencing and optimization. *ORSA journal on computing*, 6(2):154–160, 1994.

Abdelkamel Ben Ali, Gabriel Luque, Enrique Alba et Kamal E. Melkemi. An improved problem aware local search algorithm for the DNA fragment assembly problem. *Soft Computing*, 21(7):1709–1720, 2017.

Abdelkamel Ben Ali et Kamal E. Melkemi. Multi-Objective Combinatorial Optimization using a Hybrid Genetic Algorithm & PSO. Dans *2nd Conference on Theoretical and Applicative Aspects of computer science (CTAACS'13) Skikda, Algeria*, November 2013.

Dimitri P Bertsekas. *Dynamic programming and optimal control, 4th edition*. Athena Scientific Nashua, NH, 2017.

C. Blum. Beam-ACO for the longest common subsequence problem. Dans *IEEE Congress on Evolutionary Computation*, pages 1–8, July 2010.

Christian Blum. Beam-ACO—Hybridizing ant colony optimization with beam search : An application to open shop scheduling. *Computers & Operations Research*, 32(6): 1565–1591, 2005.

Christian Blum. Beam-ACO for simple assembly line balancing. *INFORMS Journal on Computing*, 20(4):618–627, 2008.

Christian Blum, Maria José Blesa Aguilera, Andrea Roli et Michael Sampels, éditeurs. *Hybrid Metaheuristics : An Emerging Approach to Optimization*. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008.

Christian Blum, Maria J Blesa et Borja Calvo. Beam-ACO for the repetition-free longest common subsequence problem. Dans *International Conference on Artificial Evolution (Evolution Artificielle)*, pages 79–90. Springer, 2013.

Christian Blum et Paola Festa. *Metaheuristics for String Problems in Bio-informatics*. John Wiley & Sons, 2016.

Christian Blum et Xiaodong Li. Swarm intelligence in optimization. Dans *Swarm Intelligence*, pages 43–85. Springer, 2008.

Christian Blum, Jakob Puchinger, Günther R Raidl et Andrea Roli. Hybrid metaheuristics in combinatorial optimization : A survey. *Applied Soft Computing*, 11(6): 4135–4151, 2011.

Christian Blum et Günther R. Raidl. *Hybrid Metaheuristics : Powerful Tools for Optimization*. Springer International Publishing, Cham, 2016.

Christian Blum et Andrea Roli. Metaheuristics in combinatorial optimization : Overview and conceptual comparison. *ACM Computing Surveys (CSUR)*, 35(3):268–308, 2003.

Marco Boschetti et Vittorio Maniezzo. Benders decomposition, lagrangean relaxation and metaheuristic design. *Journal of Heuristics*, 15(3):283–312, 2009.

Marco Boschetti, Vittorio Maniezzo et Matteo Roffilli. Decomposition Techniques as Metaheuristic Frameworks. Dans Vittorio Maniezzo, Thomas Stützle et Stefan Voß, éditeurs, *Matheuristics : Hybridizing Metaheuristics and Mathematical Programming*, pages 135–158. Springer US, Boston, MA, 2010.

Ilhem Boussaïd, Julien Lepagnot et Patrick Siarry. A survey on optimization metaheuristics. *Information Sciences*, 237:82–117, 2013.

Dimo Brockhoff et Eckart Zitzler. Objective reduction in evolutionary multiobjective optimization : Theory and applications. *Evolutionary Computation*, 17(2):135–166, 2009.

Bernd Bullnheimer, Richard F Hartl et Christine Strauss. A new rank based version of the Ant System. A computational study. *Cent. Eur. J. Oper. Res. Econ.*, 7(1): 25–38, 1999.

Christopher B Burge et Samuel Karlin. Finding the genes in genomic DNA. *Current opinion in structural biology*, 8(3):346–354, 1998.

Edmund Burke, Graham Kendall, Jim Newall, Emma Hart, Peter Ross et Sonia Schulenburg. Hyper-heuristics : An emerging direction in modern search technology. Dans *Handbook of metaheuristics*, pages 457–474. Springer, 2003.

Edmund K Burke, Matthew Hyde, Graham Kendall, Gabriela Ochoa, Ender Özcan et John R Woodward. A classification of hyper-heuristic approaches. Dans *Handbook of metaheuristics*, pages 449–468. Springer, 2010.

Edmund K Burke, Matthew R Hyde, Graham Kendall et John Woodward. Automatic heuristic generation with genetic programming : evolving a jack-of-all-trades or a master of one. Dans *Proceedings of the 9th annual conference on Genetic and evolutionary computation*, pages 1559–1565. ACM, 2007.

Vladimír Černỳ. Thermodynamical approach to the traveling salesman problem : An efficient simulation algorithm. *Journal of optimization theory and applications*, 45 (1):41–51, 1985.

Antonio Augusto Chaves, Francisco de Assis Correa et Luiz Antonio N Lorena. Clustering search heuristic for the capacitated p-median problem. Dans *Innovations in Hybrid Intelligent Systems*, pages 136–143. Springer, 2007.

Ting Chen et Steven S Skiena. Trie-based data structures for sequence assembly. Dans *Combinatorial Pattern Matching*, pages 206–223. Springer, 1997.

Asif T Chinwalla, Lisa L Cook, Kimberly D Delehaunty, Ginger A Fewell, Lucinda A Fulton, Robert S Fulton, Tina A Graves, LaDeana W Hillier, Elaine R Mardis, John D McPherson et al. Initial sequencing and comparative analysis of the mouse genome. *Nature*, 420(6915):520–562, 2002.

Paul C Chu et John E Beasley. A genetic algorithm for the multidimensional knapsack problem. *Journal of heuristics*, 4(1):63–86, 1998.

George M Church. Genomes for all. *Scientific American*, 294(1):46–54, 2006.

Jean-Michel Claverie. Identifying coding exons by similarity search : alu-derived and other potentially misleading protein sequences. *Genomics*, 12(4):838–841, 1992.

Maurice Clerc. The swarm and the queen : towards a deterministic and adaptive particle swarm optimization. Dans *Evolutionary Computation, 1999. CEC 99. Proceedings of the 1999 Congress on*, volume 3, pages 1951–1957. IEEE, 1999.

Maurice Clerc et James Kennedy. The particle swarm-explosion, stability, and convergence in a multidimensional complex space. *IEEE transactions on Evolutionary Computation*, 6(1):58–73, 2002.

Carlos A Coello Coello, Gary B Lamont, David A Van Veldhuizen et al. *Evolutionary algorithms for solving multi-objective problems*, volume 5. Springer, 2007.

Richard K Congram, Chris N Potts et Steef L van de Velde. An iterated dynasearch algorithm for the single-machine total weighted tardiness scheduling problem. *INFORMS Journal on Computing*, 14(1):52–67, 2002.

Stephen A Cook. The complexity of theorem-proving procedures. Dans *Proceedings of the third annual ACM symposium on Theory of computing*, pages 151–158. ACM, 1971.

Oscar Cordon, Iñaki Fernández de Viana, Francisco Herrera et Llanos Moreno. A New ACO Model Integrating Evolutionary Computation Concepts : The Best-Worst Ant System. Dans *Dorigo, M., Middendorf, M., Stützle, T. (eds.) Abstract proceedings of ANTS 2000 - From Ant Colonies to Artificial Ants : Second International Workshop on Ant Algorithms*, pages 22–29. IRIDIA„ Universite Libre de Bruxelles, Brussels, Belgium, 2000.

S. Davoud et M. Ellips. Particle Swarm Optimization Methods, Taxonomy and Applications. *International Journal of Computer Theory and Engineering*, 1(5):1793–8201, 2009.

Kalyanmoy Deb. Multi-objective genetic algorithms : Problem difficulties and construction of test problems. *Evolutionary computation*, 7(3):205–230, 1999.

Kalyanmoy Deb. *Multi-objective optimization using evolutionary algorithms*, volume 16. John Wiley & Sons, 2001.

Kalyanmoy Deb, Amrit Pratap, Sameer Agarwal et TAMT Meyarivan. A fast and elitist multiobjective genetic algorithm : Nsga-ii. *IEEE transactions on evolutionary computation*, 6(2):182–197, 2002.

Kalyanmoy Deb, Lothar Thiele, Marco Laumanns et Eckart Zitzler. Scalable test problems for evolutionary multiobjective optimization. *Evolutionary Multiobjective Optimization. Theoretical Advances and Applications*, pages 105–145, 2005.

Marco Dorigo. Optimization, learning and natural algorithms. *Italian PhD dissertation Politecnico di MilanoMilan*, 1992.

Marco Dorigo et Christian Blum. Ant colony optimization theory : A survey. *Theoretical computer science*, 344(2-3):243–278, 2005.

Marco Dorigo et Luca Maria Gambardella. Ant colony system : a cooperative learning approach to the traveling salesman problem. *IEEE Transactions on evolutionary computation*, 1(1):53–66, 1997.

Marco Dorigo, Vittorio Maniezzo et Alberto Colorni. Ant system : optimization by a colony of cooperating agents. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 26(1):29–41, 1996.

Marco Dorigo, Vittorio Maniezzo, Alberto Colorni et Vittorio Maniezzo. Positive feedback as a search strategy. 1991.

Marco Dorigo et Thomas Stützle. Ant colony optimization : overview and recent advances. Dans *Handbook of metaheuristics*, pages 227–263. Springer, 2010.

Bernabé Dorronsoro, Enrique Alba, Gabriel Luque et Pascal Bouvry. A self-adaptive cellular memetic algorithm for the DNA fragment assembly problem. Dans *IEEE Congress on Evolutionary Computation*, pages 2651–2658. IEEE, 2008.

Orlando Duran, Nibaldo Rodriguez et Luiz Consalter. Hybridization of pso and a discrete position update scheme techniques for manufacturing cell design. *MICAI 2008 : Advances in Artificial Intelligence*, pages 503–512, 2008.

Russ C Eberhart et Yuhui Shi. Comparing inertia weights and constriction factors in particle swarm optimization. Dans *Evolutionary Computation, 2000. Proceedings of the 2000 Congress on*, volume 1, pages 84–88. IEEE, 2000.

Russell Eberhart et James Kennedy. A new optimizer using particle swarm theory. Dans *Micro Machine and Human Science, 1995. MHS'95., Proceedings of the Sixth International Symposium on*, pages 39–43. IEEE, 1995.

Matthias Ehrgott et Xavier Gandibleux. A survey and annotated bibliography of multiobjective combinatorial optimization. *Or Spectrum*, 22(4):425–460, 2000.

Matthias Ehrgott et Xavier Gandibleux. Multiobjective combinatorial optimization theory, methodology, and applications. Dans *Multiple criteria optimization : State of the art annotated bibliographic surveys*, pages 369–444. Springer, 2003.

Matthias Ehrgott et Xavier Gandibleux. Bound sets for biobjective combinatorial optimization problems. *Computers & Operations Research*, 34(9):2674–2694, 2007.

Matthias Ehrgott et Xavier Gandibleux. Hybrid metaheuristics for multi-objective combinatorial optimization. Dans *Hybrid metaheuristics*, pages 221–259. Springer, 2008.

Bouazza Elbenani, Jacques A Ferland et Jonathan Bellemare. Genetic algorithm and large neighbourhood search to solve the cell formation problem. *Expert Systems with Applications*, 39(3):2408–2414, 2012.

Andries P Engelbrecht. *Fundamentals of computational swarm intelligence*. John Wiley & Sons, 2006.

Michael L Engle et Christian Burks. Artificially generated data sets for testing DNA sequence assembly algorithms. *Genomics*, 16(1):286–288, 1993.

Anna Bou Ezzeddine, Stefan Kasala et Pavol Navrat. APPLYING THE FIREFLY APPROACH TO THE DNA FRAGMENTS ASSEMBLY PROBLEM. *Annales Univ. Sci. Budapest., Sect. Comp.*, 42:69–81, 2014.

Shu-Cherng Fang, Yong Wang et Jie Zhong. A genetic algorithm approach to solving DNA fragment assembly problem. *Journal of Computational and Theoretical Nanoscience*, 2(4):499–505, 2005.

Thomas A Feo et Mauricio GC Resende. A probabilistic heuristic for a computationally difficult set covering problem. *Operations research letters*, 8(2):67–71, 1989.

Thomas A Feo et Mauricio GC Resende. Greedy randomized adaptive search procedures. *Journal of global optimization*, 6(2):109–133, 1995.

Carlos Eduardo Ferreira, C Carvalho de Souza et Yoshiko Wakabayashi. Rearrangement of DNA fragments : a branch-and-cut algorithm. *Discrete Applied Mathematics*, 116(1):161–177, 2002.

Jesun Sahariar Firoz, M Sohel Rahman et Tanay Kumar Saha. Bee algorithms for solving DNA fragment assembly problem with noisy and noiseless data. Dans *Proceedings of the 14th annual conference on Genetic and evolutionary computation*, pages 201–208. ACM, 2012.

Charles Fleurent et Fred Glover. Improved constructive multistart strategies for the quadratic assignment problem using adaptive memory. *INFORMS Journal on Computing*, 11(2):198–204, 1999.

Lawrence J Fogel, Alvin J Owens et Michael J Walsh. Artificial intelligence through simulated evolution. 1966.

Carlos M Fonseca et Peter J Fleming. An overview of evolutionary algorithms in multiobjective optimization. *Evolutionary computation*, 3(1):1–16, 1995.

Carlos M Fonseca, Peter J Fleming et al. Genetic algorithms for multiobjective optimization : Formulationdiscussion and generalization. Dans *Icga*, volume 93, pages 416–423. Citeseer, 1993.

John K Gallant. The complexity of the overlap method for sequencing biopolymers. *Journal of Theoretical Biology*, 101(1):1–17, 1983.

Luca Maria Gambardella, Roberto Montemanni et Dennis Weyland. Coupling ant colony systems with strong local searches. *European Journal of Operational Research*, 220(3):831–843, 2012.

Michael R Garey et David S Johnson. Computers and intractability : a guide to the theory of NP-completeness. *San Francisco, LA : Freeman*, 58, 1979.

Michel Gendreau et Jean-Yves Potvin. *Handbook of metaheuristics*, volume 2. Springer, 2010a.

Michel Gendreau et Jean-Yves Potvin. Tabu Search. Dans *Handbook of Metaheuristics*, pages 41–59. Springer US, 2010b.

Youcef Gheraibia, Abdelouahab Moussaoui, Sohag Kabir et Smaine Mazouzi. PeDFA : Penguins Search Optimisation Algorithm for DNA Fragment Assembly. *International Journal of Applied Metaheuristic Computing (IJAMC)*, 7(2):58–70, 2016.

Fred Glover. Future paths for integer programming and links to artificial intelligence. *Computers & operations research*, 13(5):533–549, 1986.

Fred Glover et Manuel Laguna. Tabu search. Dans *Modern heuristic techniques for combinatorial problems*, pages 70–150. John Wiley & Sons, Inc., 1993.

Fred Glover et Manuel Laguna. *Tabu Search*. Kluwer Academic Publishers, Norwell, MA, USA, 1997.

Fred Glover et Manuel Laguna. Tabu Search*. Dans *Handbook of Combinatorial Optimization*, pages 3261–3362. Springer New York, 2013.

Fred Glover, Manuel Laguna et Rafael Martí. Fundamentals of scatter search and path relinking. *Control and cybernetics*, 29(3):653–684, 2000.

Fred Glover et Eric Taillard. A user's guide to tabu search. *Annals of operations research*, 41(1):1–28, 1993.

David E Goldberg. Genetic algorithms in search, optimization and machine learning 'addison-wesley, 1989. *Reading, MA*, 1989.

David E Goldberg, Jon Richardson et al. Genetic algorithms with sharing for multimodal function optimization. Dans *Genetic algorithms and their applications : Proceedings of the Second International Conference on Genetic Algorithms*, pages 41–49. Hillsdale, NJ : Lawrence Erlbaum, 1987.

P. Green. *Phrap, http ://www.phrap.org*, 1994.

Peter Greistorfer. A tabu scatter search metaheuristic for the arc routing problem. *Computers & Industrial Engineering*, 44(2):249–266, 2003.

Dan Gusfield. *Algorithms on strings, trees and sequences : computer science and computational biology*. Cambridge university press, 1997.

Neil Hall. Advanced sequencing technologies and their wider impact in microbiology. *Journal of Experimental Biology*, 210(9):1518–1525, 2007.

Kuk-Hyun Han et Jong-Hwan Kim. Quantum-inspired evolutionary algorithm for a class of combinatorial optimization. *IEEE transactions on evolutionary computation*, 6(6):580–593, 2002.

Pierre Hansen, Nenad Mladenović, Jack Brimberg et José A. Moreno Pérez. Variable Neighborhood Search. Dans *Handbook of Metaheuristics*, pages 61–86. Springer US, 2010.

S He, QH Wu, JY Wen, JR Saunders et RC Paton. A particle swarm optimizer with passive congregation. *Biosystems*, 78(1):135–147, 2004.

David Hernandez, Patrice François, Laurent Farinelli, Magne Østerås et Jacques Schrenzel. De novo bacterial genome sequencing : millions of very short reads assembled on a desktop computer. *Genome research*, 18(5):802–809, 2008.

B Hesper et P Hogeweg. Bioinformatica : een werkconcept. *Kameleon*, 1(6):28–29, 1970.

AJ Hoffman, J Wolfe, RS Garfinkel, DS Johnson, CH Papadimitriou, PC Gilmore, EL Lawler, DB Shmoys, RM Karp, JM Steele et al. *The traveling salesman problem : a guided tour of combinatorial optimization*. J. Wiley & Sons, 1986.

Paulien Hogeweg. The roots of bioinformatics in theoretical biology. *PLoS Comput Biol*, 7(3):e1002021, 2011.

John H Holland. Adaptation in natural and artificial systems. An introductory analysis with application to biology, control, and artificial intelligence. *Ann Arbor, MI : University of Michigan Press*, 1975.

Xiaohui Hu et Russell Eberhart. Multi-objective optimization with using dynamic neighborhood particle swarm optimization. Dans *Proceedings of the IEEE Congress on Evolutionary Computation*, pages 1677–1681, 2002.

Xiaohui Hu, Russell C Eberhart et Yuhui Shi. Swarm intelligence for permutation optimization : a case study of n-queens problem. Dans *Swarm intelligence symposium, 2003. SIS'03. Proceedings of the 2003 IEEE*, pages 243–246. IEEE, 2003.

Ko-Wei Huang, Jui-Le Chen, Chu-Sing Yang et Chun-Wei Tsai. A memetic particle swarm optimization algorithm for solving the DNA fragment assembly problem. *Neural Computing and Applications*, 26(3):495–506, 2015. ISSN 0941-0643.

Ko-Wei Huang, Jui-Le Chen, Chu-Sing Yang et Chun-Wei Tsai. A memetic gravitation search algorithm for solving DNA fragment assembly problems. *Journal of Intelligent & Fuzzy Systems*, 30(4):2245–2255, 2016.

Xiaoqiu Huang et Anup Madan. CAP3 : A DNA sequence assembly program. *Genome research*, 9(9):868–877, 1999.

James Alexander Hughes, Sheridan Houghten et Daniel Ashlock. Restarting and recentering genetic algorithm variations for DNA fragment assembly : The necessity of a multi-strategy approach. *Biosystems*, 150:35–45, 2016.

Ramana M Idury et Michael S Waterman. A new algorithm for DNA sequence assembly. *Journal of computational biology*, 2(2):291–306, 1995.

R Indumathy, S Uma Maheswari et G Subashini. Nature-inspired novel Cuckoo Search Algorithm for genome sequence assembly. *Sadhana*, 40(1):1–14, 2015.

R Indumathy et SU Maheswari. Nature inspired algorithms to solve DNA fragment assembly problem : A survey. *International Journal of Bioinformatics Research and Applications*, 2(2):45–50, 2012.

Mikkel Jensen. Guiding single-objective optimization using multi-objective methods. *Applications of Evolutionary Computing*, pages 91–98, 2003.

Yimin Jing et Sami Khuri. Exact and heuristic algorithms for the DNA fragment assembly problem. Dans *Proceedings of the IEEE Computer Society Conference on Bioinformatics*, page 581. IEEE Computer Society, 2003.

Nicolas Jozefowiez, Frédéric Semet et El-Ghazali Talbi. Multi-objective vehicle routing problems. *European journal of operational research*, 189(2):293–309, 2008.

Wang-Sheng Juang et Shun-Feng Su. Multiple sequence alignment using modified dynamic programming and particle swarm optimization. *Journal of the Chinese institute of engineers*, 31(4):659–673, 2008.

Richard M Karp. Reducibility among combinatorial problems. Dans *Complexity of computer computations*, pages 85–103. Springer, 1972.

John D Kececioglu et Eugene W Myers. Combinatorial algorithms for DNA sequence assembly. *Algorithmica*, 13(1-2):7, 1995.

John Dimitri Kececioglu. *Exact and approximation algorithms for DNA sequence reconstruction*. Thèse de doctorat, The University of Arizona, 1993.

J. Kennedy et R.C. Eberhart. Particle Swarm Optimization. Dans *Proc. IEEE Int. Conf. on N.N*, pages 1942–1948, 1995.

J. Kennedy et R.C. Eberhart. A discrete binary version of the particle swarm algorithm. Dans *Int. IEEE Conf. on Systems, Man, and Cyber*, volume 5, pages 4104–4108, 1997.

Madjid Khichane, Patrick Albert et Christine Solnon. Strong combination of ant colony optimization with constraint programming optimization. Dans *International Conference on Integration of Artificial Intelligence (AI) and Operations Research (OR) Techniques in Constraint Programming*, pages 232–245. Springer, 2010.

Satoko Kikuchi et Goutam Chakraborty. Heuristically tuned GA to solve genome fragment assembly problem. Dans *2006 IEEE International Conference on Evolutionary Computation*, pages 1491–1498. IEEE, 2006.

Satoko Kikuchi et Goutam Chakraborty. An efficient genome fragment assembling using GA with neighborhood aware fitness function. *Applied Computational Intelligence and Soft Computing*, 2012:7, 2012.

Kieun Kim et Chilukuri K Mohan. Parallel hierarchical adaptive genetic algorithm for fragment assembly. Dans *Evolutionary Computation, 2003. CEC'03. the 2003 Congress on*, volume 1, pages 600–607. IEEE, 2003.

Scott Kirkpatrick, C Daniel Gelatt, Mario P Vecchi et al. Optimization by simulated annealing. *science*, 220(4598):671–680, 1983.

Joshua Knowles et David Corne. The pareto archived evolution strategy : A new baseline algorithm for pareto multiobjective optimisation. Dans *Evolutionary Computation, 1999. CEC 99. Proceedings of the 1999 Congress on*, volume 1, pages 98–105. IEEE, 1999.

John R Koza. *Genetic programming : on the programming of computers by means of natural selection*, volume 1. MIT press, 1992.

Natalio Krasnogor et James Smith. A tutorial for competent memetic algorithms : model, taxonomy, and design issues. *IEEE Transactions on Evolutionary Computation*, 9(5):474–488, 2005.

Jiří Kubalik, Petr Buryan et Libor Wagner. Solving the DNA fragment assembly problem efficiently using iterative optimization with evolved hypermutations. Dans *Proceedings of the 12th annual conference on Genetic and evolutionary computation*, pages 213–214. ACM, 2010.

Philippe Lacomme, Christian Prins et Marc Sevaux. A genetic algorithm for a bi-objective capacitated arc routing problem. *Computers & Operations Research*, 33 (12):3473–3493, 2006.

Lishan Li et Sami Khuri. A comparison of DNA fragment assembly algorithms. Dans *METMBS*, volume 4, pages 329–335, 2004.

Ruiqiang Li, Hongmei Zhu, Jue Ruan, Wubin Qian, Xiaodong Fang, Zhongbin Shi, Yingrui Li, Shengting Li, Gao Shan, Karsten Kristiansen et al. De novo assembly of human genomes with massively parallel short read sequencing. *Genome research*, 20(2):265–272, 2010.

Arnaud Liefooghe, Sébastien Verel et Jin-Kao Hao. A hybrid metaheuristic for multiobjective unconstrained binary quadratic programming. *Applied Soft Computing*, 16:10–19, 2014.

Shih-Wei Lin, Zne-Jung Lee, Kuo-Ching Ying et Chou-Yuan Lee. Applying hybrid meta-heuristics for capacitated vehicle routing problem. *Expert Systems with Applications*, 36(2):1505–1512, 2009.

M López-Ibánez, T Stützle et M Dorigo. Ant colony optimization : A component-wise overview. *Techreport, IRIDIA, Universite Libre de Bruxelles*, 2015.

Manuel López-Ibáñez et Christian Blum. Beam-ACO for the travelling salesman problem with time windows. *Computers & Operations Research*, 37(9):1570–1583, 2010.

Helena R Lourenço, Olivier C Martin et Thomas Stützle. Iterated local search : Framework and applications. Dans *Handbook of metaheuristics*, pages 363–397. Springer, 2010.

Manuel Lozano et Carlos García-Martínez. Hybrid metaheuristics with evolutionary algorithms specializing in intensification and diversification : Overview and progress report. *Computers & Operations Research*, 37(3):481–497, 2010.

Gabriel Luque et Enrique Alba. Metaheuristics for the DNA fragment assembly problem. *International Journal of Computational Intelligence Research*, 1(2):98–108, 2005.

Gabriel Luque et Enrique Alba. Parallel gas in bioinformatics : assembling DNA fragments. Dans *Parallel Genetic Algorithms*, pages 135–147. Springer, 2011.

HAJ-RACHID Mais, Christelle Bloch, Wahiba Ramdane-Cherif et Pascal Chatonnay. Différentes opérateurs évolutionnaires de permutation : sélections, croisements et mutations. Rapport technique ERR 2010-07, Laboratoire d'informatique de l'université de Franche-Comté, 2010.

Guillermo M Mallén-Fullerton et Guillermo Fernández-Anaya. DNA fragment assembly using optimization. Dans *Evolutionary Computation (CEC), 2013 IEEE Congress on*, pages 1570–1577. IEEE, 2013.

Guillermo M Mallén-Fullerton, James Alexander Hughes, Sheridan Houghten et Guillermo Fernández-Anaya. Benchmark datasets for the DNA fragment assembly problem. *International Journal of Bio-Inspired Computation*, 5(6):384–394, 2013.

Marcel Margulies, Michael Egholm, William E Altman, Said Attiya, Joel S Bader, Lisa A Bemben, Jan Berka, Michael S Braverman, Yi-Ju Chen, Zhoutao Chen et al. Genome sequencing in microfabricated high-density picolitre reactors. *Nature*, 437 (7057):376–380, 2005.

Catherine Mathé, Marie-France Sagot, Thomas Schiex et Pierre Rouzé. Current methods of gene prediction, their strengths and weaknesses. *Nucleic acids research*, 30 (19):4103–4117, 2002.

Alan R McKendall et Jin Shang. Hybrid ant systems for the dynamic facility layout problem. *Computers & Operations Research*, 33(3):790–803, 2006.

Prakit Meksangsouy et N Chaiyaratana. DNA fragment assembly using an ant colony system algorithm. Dans *Evolutionary Computation, 2003. CEC'03. The 2003 Congress on*, volume 3, pages 1756–1763. IEEE, 2003.

Claudio N Meneses, Carlos AS Oliveira et Panos M Pardalos. Optimization techniques for string selection and comparison problems in genomics. *IEEE Engineering in Medicine and Biology Magazine*, 24(3):81–87, 2005.

Nicholas Metropolis, Arianna W Rosenbluth, Marshall N Rosenbluth, Augusta H Teller et Edward Teller. Equation of state calculations by fast computing machines. *The journal of chemical physics*, 21(6):1087–1092, 1953.

Herve Meunier, E-G Talbi et Philippe Reininger. A multiobjective genetic algorithm for radio network optimization. Dans *Evolutionary Computation, 2000. Proceedings of the 2000 Congress on*, volume 1, pages 317–324. IEEE, 2000.

Bernd Meyer. Hybrids of Constructive Metaheuristics and Constraint Programming : A Case Study with ACO. Dans Christian Blum, Maria José Blesa Aguilera, Andrea Roli et Michael Sampels, éditeurs, *Hybrid Metaheuristics : An Emerging Approach to Optimization*, pages 151–183. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008.

Zbigniew Michalewicz et Jarosław Arabas. Genetic algorithms for the 0/1 knapsack problem. Dans *International Symposium on Methodologies for Intelligent Systems*, pages 134–143. Springer, 1994.

Kaisa Miettinen. *Nonlinear Multiobjective Optimization*. Kluwer, 1999.

Jason R Miller, Sergey Koren et Granger Sutton. Assembly algorithms for next-generation sequencing data. *Genomics*, 95(6):315–327, 2010.

Gabriela Minetti, Enrique Alba et Gabriel Luque. Seeding strategies and recombination operators for solving the DNA fragment assembly problem. *Information Processing Letters*, 108(3):94–100, 2008a.

Gabriela Minetti, Guillermo Leguizamón et Enrique Alba. Sax : a new and efficient assembler for solving DNA fragment assembly problem. Dans *2012 Argentine Symposium on Artificial Intelligence*, 2012.

Gabriela Minetti, Guillermo Leguizamón et Enrique Alba. An improved trajectory-based hybrid metaheuristic applied to the noisy DNA Fragment Assembly Problem. *Information Sciences*, 277:273–283, 2014.

Gabriela Minetti, Gabriel Luque et Enrique Alba. Variable neighborhood search as genetic algorithm operator for DNA fragment assembling problem. Dans *Hybrid Intelligent Systems, 2008. HIS'08. Eighth International Conference on*, pages 714–719. IEEE, 2008b.

Nenad Mladenovic. A variable neighborhood algorithm-a new metaheuristic for combinatorial optimization. Dans *papers presented at Optimization Days*, volume 12, 1995.

Nenad Mladenović et Pierre Hansen. Variable neighborhood search. *Computers & operations research*, 24(11):1097–1100, 1997.

Roberto Montemanni et Derek H Smith. Heuristic manipulation, tabu search and frequency assignment. *Computers & Operations Research*, 37(3):543–551, 2010.

Glyn Moody. *Digital code of life : how bioinformatics is revolutionizing science, medicine, and business*. John Wiley & Sons, 2004.

Pablo Moscato. Memetic algorithms : A short introduction. Dans *New ideas in optimization*, pages 219–234. McGraw-Hill Ltd., UK, 1999.

Sayyed Rasoul Mousavi, Maryam Babaie et Manizheh Montazerian. An improved heuristic for the far from most strings problem. *Journal of Heuristics*, 18(2):239–262, 2012.

Eugene W Myers. Toward simplifying and accurately formulating fragment assembly. *Journal of Computational Biology*, 2(2):275–290, 1995.

Eugene W Myers. The fragment assembly string graph. *Bioinformatics*, 21(suppl 2): ii79–ii85, 2005.

Eugene W Myers, Granger G Sutton, Art L Delcher, Ian M Dew, Dan P Fasulo, Michael J Flanigan, Saul A Kravitz, Clark M Mobarry, Knut HJ Reinert, Karin A Remington et al. A whole-genome assembly of drosophila. *Science*, 287(5461): 2196–2204, 2000.

Gene Myers. Efficient local alignment discovery amongst noisy long reads. Dans *International Workshop on Algorithms in Bioinformatics*, pages 52–67. Springer, 2014.

Eugene W Myers Jr. A history of DNA sequence assembly. *It-Information Technology*, 58(3):126–132, 2016.

Antonio J Nebro, Gabriel Luque, Francisco Luna et Enrique Alba. DNA fragment assembly using a grid-based genetic algorithm. *Computers & Operations Research*, 35(9):2776–2790, 2008.

Shahla Nemati, Mohammad Ehsan Basiri, Nasser Ghasem-Aghaee et Mehdi Hosseinzadeh Aghdam. A novel aco–ga hybrid algorithm for feature selection in protein function prediction. *Expert systems with applications*, 36(10):12086–12094, 2009.

Frank Neumann et Ingo Wegener. Minimum spanning trees made easier via multi-objective optimization. *Natural Computing*, 5(3):305–319, 2006.

Alexander G Nikolaev et Sheldon H Jacobson. Simulated annealing. Dans *Handbook of metaheuristics*, pages 1–39. Springer, 2010.

Christos H Papadimitriou. *Computational complexity*. John Wiley and Sons Ltd., 2003.

Christos H Papadimitriou et Kenneth Steiglitz. *Combinatorial optimization : algorithms and complexity*. Dover Publications, Inc., New York, 1982.

Elisa Pappalardo, Panos M Pardalos, Giovanni Stracquadanio et al. *Optimization approaches for solving string selection problems*. Springer, 2013.

Vilfredo Pareto. *Cours d'Ãl'conomie politique*. Rouge, Lausanne, Switzerland, 1896.

Rebecca Parsons et Mark E Johnson. DNA Sequence Assembly and Genetic Algorithms-New Results and Puzzling Insights. Dans *ISMB*, volume 3, pages 277–84, 1995.

Rebecca J Parsons, Stephanie Forrest et Christian Burks. Genetic algorithms for DNA sequence assembly. Dans *ISMB*, pages 310–318, 1993.

Rebecca J Parsons, Stephanie Forrest et Christian Burks. Genetic algorithms, operators, and DNA fragment assembly. *Machine Learning*, 21(1-2):11–33, 1995.

Konrad Paszkiewicz et David J Studholme. De novo assembly of short sequence reads. *Briefings in bioinformatics*, page bbq020, 2010.

Pavel Pevzner. *Computational molecular biology : an algorithmic approach*. MIT press, 2000.

Pavel A Pevzner, Haixu Tang et Michael S Waterman. An Eulerian path approach to DNA fragment assembly. *Proceedings of the National Academy of Sciences*, 98 (17):9748–9753, 2001.

David Pisinger et Stefan Ropke. A general heuristic for vehicle routing problems. *Computers & operations research*, 34(8):2403–2435, 2007.

Riccardo Poli, James Kennedy et Tim Blackwell. Particle swarm optimization. *Swarm intelligence*, 1(1):33–57, 2007.

Mihai Pop. Shotgun sequence assembly. *Advances in Computers*, 60:193–248, 2004.

Matthias Prandtstetter et Günther R Raidl. An integer linear programming approach and a hybrid variable neighborhood search for the car sequencing problem. *European Journal of Operational Research*, 191(3):1004–1022, 2008.

Günther R Raidl. A unified view on hybrid metaheuristics. Dans *International Workshop on Hybrid Metaheuristics*, pages 1–12. Springer, 2006.

Günther R. Raidl, Jakob Puchinger et Christian Blum. Metaheuristic Hybrids. Dans Michel Gendreau et Jean-Yves Potvin, éditeurs, *Handbook of Metaheuristics*, pages 469–496. Springer US, Boston, MA, 2010.

Kari-Jouko Räihä et Esko Ukkonen. The shortest common supersequence problem over binary alphabet is NP-complete. *Theoretical Computer Science*, 16(2):187–198, 1981.

Indumathy Rajagopal et Uma Maheswari Sankareswaran. An adaptive particle swarm optimization algorithm for solving dna fragment assembly problem. *Current Bioinformatics*, 10(1):97–105, 2015.

Sanguthevar Rajasekaran, Y Hu, Jun Luo, H Nick, Panos M. Pardalos, Sartaj Sahni et G Shaw. Efficient algorithms for similarity search. *Journal of Combinatorial Optimization*, 5(1):125–132, 2001a.

Sanguthevar Rajasekaran, H Nick, Panos M. Pardalos, Sartaj Sahni et G Shaw. Efficient algorithms for local alignment search. *Journal of Combinatorial Optimization*, 5(1):117–124, 2001b.

Kim R Rasmussen, Jens Stoye et Eugene W Myers. Efficient q-gram filters for finding all ε-matches over a given length. *Journal of Computational Biology*, 13(2):296–308, 2006.

Manisha Rathee et TV Vijay Kumar. Dna fragment assembly using multi-objective genetic algorithms. *International Journal of Applied Evolutionary Computation (IJAEC)*, 5(3):84–108, 2014.

I. Rechenberg. *Evolutionsstrategie Optimierung technischer Systeme nach Prinzipien der biologishen Evolution*. Frommann-Holzboog, Stuttgart, 1973.

Jérémi Regnier. *Conception de systèmes hétérogènes en Génie Electrique par optimisation évolutionnaire multicritère*. Thèse de doctorat, Institut National Polytechnique de Toulouse, 2003.

Mauricio GC Resende et Celso C Ribeiro. Greedy randomized adaptive search procedures : Advances, hybridizations, and applications. Dans *Handbook of metaheuristics*, pages 283–319. Springer, 2010.

Fabio Romeo et Alberto Sangiovanni-Vincentelli. A theoretical framework for simulated annealing. *Algorithmica*, 6(1):302–345, 1991.

F Sanger, Ar R Coulson, GF Hong, DF Hill et GB Petersen. Nucleotide sequence of bacteriophage λ DNA. *Journal of molecular biology*, 162(4):729–773, 1982.

Hiroyuki Sato, Hernán Aguirre et Kiyoshi Tanaka. Variable space diversity, crossover and mutation in moea solving many-objective knapsack problems. *Annals of Mathematics and Artificial Intelligence*, 68(4):197–224, 2013.

Hiroyuki Sato, Hernán E Aguirre et Tanaka Kiyoshi. Effects of moea temporally switching pareto partial dominance on many-objective 0/1 knapsack problems. *Transactions of the Japanese Society for Artificial Intelligence*, 25:320–331, 2010.

Joao Carlos Setubal et Joao Meidanis. *Introduction to computational molecular biology*, chaptitre 4 - Fragment Assembly of DNA, pages 105–139. University of Campinas, Brazil, 1997.

Paul Shaw. Using Constraint Programming and Local Search Methods to Solve Vehicle Routing Problems. Dans Michael Maher et Jean-Francois Puget, éditeurs, *Principles and Practice of Constraint Programming — CP98 : 4th International Conference, CP98 Pisa, Italy, October 26–30, 1998 Proceedings*, pages 417–431. Springer Berlin Heidelberg, Berlin, Heidelberg, 1998.

XH Shi, YC Liang, HP Lee, C Lu et LM Wang. An improved ga and a novel pso-ga-based hybrid algorithm. *Information Processing Letters*, 93(5):255–261, 2005.

Yuhui Shi et Russell Eberhart. A modified particle swarm optimizer. Dans *Evolutionary Computation Proceedings, 1998. IEEE World Congress on Computational Intelligence., The 1998 IEEE International Conference on*, pages 69–73. IEEE, 1998.

Martello Silvano et Toth Paolo. *Knapsack problems : algorithms and computer implementations*. John Wiley & Sons, 1990.

Jared T Simpson, Kim Wong, Shaun D Jackman, Jacqueline E Schein, Steven JM Jones et Inanç Birol. ABySS : a parallel assembler for short read sequence data. *Genome research*, 19(6):1117–1123, 2009.

Hemant Kumar Singh, Amitay Isaacs et Tapabrata Ray. A pareto corner search evolutionary algorithm and dimensionality reduction in many-objective optimization problems. *IEEE Transactions on Evolutionary Computation*, 15(4):539–556, 2011.

Temple F Smith et Michael S Waterman. Identification of common molecular subsequences. *Journal of molecular biology*, 147(1):195–197, 1981.

Moshe Sniedovich et S Viß. The corridor method : a dynamic programming inspired metaheuristic. *Control and Cybernetics*, 35:551–578, 2006.

Christine Solnon. *Contributions à la résolution pratique de problèmes combinatoires –des fourmis et des graphes–*. Thèse de doctorat, Université Lyon 1, 2005.

Christine Solnon. *Ant colony optimization and constraint programming*. John Wiley & Sons, 2013.

Nidamarthi Srinivas et Kalyanmoy Deb. Muiltiobjective optimization using nondominated sorting in genetic algorithms. *Evolutionary computation*, 2(3):221–248, 1994.

Thomas Stützle. Iterated local search for the quadratic assignment problem. *European Journal of Operational Research*, 174(3):1519–1539, 2006.

Thomas Stützle et Holger H Hoos. MAX–MIN ant system. *Future generation computer systems*, 16(8):889–914, 2000.

TA Sudkamp. Languages and Machines : An Introduction to the Theory of Computer Science. 2006.

Jun Sun, Bin Feng et Wenbo Xu. Particle swarm optimization with particles having quantum behavior. Dans *Evolutionary Computation, 2004. CEC2004. Congress on*, volume 1, pages 325–331. IEEE, 2004.

Granger G Sutton, Owen White, Mark D Adams et Anthony R Kerlavage. TIGR assembler : A new tool for assembling large shotgun sequencing projects. *Genome Science and Technology*, 1(1):9–19, 1995.

E-G Talbi. A taxonomy of hybrid metaheuristics. *Journal of heuristics*, 8(5):541–564, 2002.

El-Ghazali Talbi. *Metaheuristics : from design to implementation*, volume 74. John Wiley & Sons, 2009.

El-Ghazali Talbi, éditeur. *Hybrid Metaheuristics*. Springer Berlin Heidelberg, Berlin, Heidelberg, 2013.

El-Ghazali Talbi. Hybrid metaheuristics for multi-objective optimization. *Journal of Algorithms & Computational Technology*, 9(1):41–63, 2015.

Kay Chen Tan, Chun Yew Cheong et Chi Keong Goh. Solving multiobjective vehicle routing problem with stochastic demand via evolutionary computation. *European Journal of operational research*, 177(2):813–839, 2007.

Kay Chen Tan, YH Chew et Loo Hay Lee. A hybrid multi-objective evolutionary algorithm for solving truck and trailer vehicle routing problems. *European Journal of Operational Research*, 172(3):855–885, 2006a.

Kay Chen Tan, Yoong Han Chew et LH Lee. A hybrid multiobjective evolutionary algorithm for solving vehicle routing problem with time windows. *Computational Optimization and Applications*, 34(1):115, 2006b.

M Fatih Tasgetiren, Mehmet Sevkli, Yun-Chia Liang et Gunes Gencyilmaz. Particle swarm optimization algorithm for permutation flowshop sequencing problem. Dans *International Workshop on Ant Colony Optimization and Swarm Intelligence*, pages 382–389. Springer, 2004a.

Mehmet Fatih Tasgetiren, Mehmet Sevkli, Yun-Chia Liang et Gunes Gencyilmaz. Particle swarm optimization algorithm for single machine total weighted tardiness problem. Dans *Evolutionary Computation, 2004. CEC2004. Congress on*, volume 2, pages 1412–1419. IEEE, 2004b.

Dirk Thierens. Population-based iterated local search : restricting neighborhood search by crossover. Dans *Genetic and Evolutionary Computation–GECCO 2004*, pages 234–245. Springer, 2004.

Vincent T'kindt, Nicolas Monmarché, Fabrice Tercinet et Daniel Laügt. An ant colony optimization algorithm to solve a 2-machine bicriteria flowshop scheduling problem. *European Journal of Operational Research*, 142(2):250–257, 2002.

Alan Mathison Turing. On computable numbers, with an application to the Entscheidungsproblem. *Proceedings of the London mathematical society*, 2(1):230–265, 1937.

Ezgi Deniz Ülker. Adaptation of harmony search algorithm for DNA fragment assembly problem. Dans *SAI Computing Conference (SAI), 2016*, pages 135–138. IEEE, 2016.

J Craig Venter, Mark D Adams, Eugene W Myers, Peter W Li, Richard J Mural, Granger G Sutton, Hamilton O Smith, Mark Yandell, Cheryl A Evans, Robert A Holt et al. The sequence of the human genome. *science*, 291(5507):1304–1351, 2001.

Ravi Shankar Verma et al. DSAPSO : DNA sequence assembly using continuous particle swarm optimization with smallest position value rule. Dans *Recent Advances in Information Technology (RAIT), 2012 1st International Conference on*, pages 410–415. IEEE, 2012.

Andrea Villagra, Enrique Alba et Guillermo Leguizamón. A methodology for the hybridization based in active components : The case of cga and scatter search. *Computational Intelligence and Neuroscience*, 2016, 2016.

Andrea Villagra, Guillermo Leguizamón et Enrique Alba. Active components of metaheuristics in cellular genetic algorithms. *Soft Computing*, 19(5):1295–1309, 2015.

R Vlennet, Christian Fonteix et Ivan Marc. Multicriteria optimization using a genetic algorithm for determining a pareto set. *International Journal of Systems Science*, 27(2):255–260, 1996.

Chris Walshaw. Multilevel refinement for combinatorial optimisation problems. *Annals of Operations Research*, 131(1):325–372, 2004.

Chris Walshaw. Multilevel refinement for combinatorial optimisation : Boosting metaheuristic performance. Dans *Hybrid Metaheuristics*, pages 261–289. Springer, 2008.

Jean-Paul Watson. An introduction to fitness landscape analysis and cost models for local search. Dans *Handbook of metaheuristics*, pages 599–623. Springer, 2010.

James L Weber et Eugene W Myers. Human whole-genome shotgun sequencing. *Genome Research*, 7(5):401–409, 1997.

Darrell Whitley, Timothy Starkweather et Dan Shaner. *The traveling salesman and sequence scheduling : Quality solutions using genetic edge recombination.* Colorado State University, Department of Computer Science, 1991.

L Darrell Whitley, Timothy Starkweather et D'Ann Fuquay. Scheduling problems and traveling salesmen : The genetic edge recombination operator. Dans *ICGA*, volume 89, pages 133–40, 1989.

Christophe Wilbaut et Said Hanafi. New convergent heuristics for 0–1 mixed integer programming. *European Journal of Operational Research*, 195(1):62–74, 2009.

Yiheng Xu, Qiangwei Wang et Jinglu Hu. An improved discrete particle swarm optimization based on cooperative swarms. Dans *Proceedings of the 2008 IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology-Volume 02*, pages 79–82. IEEE Computer Society, 2008.

Shuyuan Yang, Min Wang et al. A quantum particle swarm optimization. Dans *Evolutionary Computation, 2004. CEC2004. Congress on*, volume 1, pages 320–324. IEEE, 2004.

Haluk Yapicioglu, Alice E Smith et Gerry Dozier. Solving the semi-desirable facility location problem using bi-objective particle swarm. *European Journal of Operational Research*, 177(2):733–749, 2007.

Hualong Yu, Guochang Gu, Haibo Liu, Jing Shen et Changming Zhu. A novel discrete particle swarm optimization algorithm for microarray data-based tumor marker gene selection. Dans *Computer Science and Software Engineering, 2008 International Conference on*, volume 1, pages 1057–1060. IEEE, 2008.

Daniel R Zerbino et Ewan Birney. Velvet : algorithms for de novo short read assembly using de Bruijn graphs. *Genome research*, 18(5):821–829, 2008.

Zhi-Hui Zhan et Jun Zhang. Discrete particle swarm optimization for multiple destination routing problems. *Applications of Evolutionary Computing*, pages 117–122, 2009.

Gexiang Zhang. Quantum-inspired evolutionary algorithms : a survey and empirical study. *Journal of Heuristics*, 17(3):303–351, 2011.

Wen-hang Zhong, Jun Zhang et Wei-neng Chen. A novel discrete particle swarm optimization to solve traveling salesman problem. Dans *Evolutionary Computation, 2007. CEC 2007. IEEE Congress on*, pages 3283–3287. IEEE, 2007.

Yalan Zhou, Jiahai Wang et Jian Yin. A discrete estimation of distribution particle swarm optimization for combinatorial optimization problems. Dans *Natural Computation, 2007. ICNC 2007. Third International Conference on*, volume 4, pages 80–84. IEEE, 2007.

Eckart Zitzler, Kalyanmoy Deb et Lothar Thiele. Comparison of multiobjective evolutionary algorithms : Empirical results. *Evolutionary computation*, 8(2):173–195, 2000.

Eckart Zitzler, Marco Laumanns et Stefan Bleuler. A tutorial on evolutionary multiobjective optimization. *Metaheuristics for multiobjective optimisation*, pages 3–37, 2004.

Eckart Zitzler, Marco Laumanns, Lothar Thiele et al. Spea2 : Improving the strength pareto evolutionary algorithm. Rapport technique 103, Computer Engineering and Networks Laboratory (TIK), Swiss Federal Institute of Technology (ETH) Zurich, Gloriastrasse 35, CH-8092 Zurich, Switzerland, 2001.

Eckart Zitzler et Lothar Thiele. Multiobjective optimization using evolutionary algorithms–a comparative case study. Dans *international conference on parallel problem solving from nature*, pages 292–301. Springer, 1998.

Eckart Zitzler et Lothar Thiele. Multiobjective evolutionary algorithms : A comparative case study and the strength Pareto approach. *IEEE transactions on Evolutionary Computation*, 3(4):257–271, 1999.

Eckart Zitzler, Lothar Thiele, Marco Laumanns, Carlos M Fonseca et Viviane Grunert Da Fonseca. Performance assessment of multiobjective optimizers : An analysis and review. *IEEE Transactions on evolutionary computation*, 7(2):117–132, 2003.