RÉPUBLIQUE ALGERIÉNNE DÉMOCRATIQUE ET POPULAIRE
MINISTÉRE DE L'ENSEIGNEMENT SUPÉRIEUR ET DE LA RECHERCHE SCIENTIFIQUE

Université des sciences et de la technologie Houari Boumediene – USTHB
Faculté d'électronique & informatique
Département informatique

THÈSE

présentée pour obtenir

LE GRADE DE DOCTEUR D'ÉTAT EN SCIENCES DE L'USTHB

Spécialité : INFORMATIQUE

Par

Malek RAHOUAL

Sujet : **Contribution à l'optimisation combinatoire mono et multiobjectif par des méthodes coopératives : application aux problèmes d'ordonnancement.**

Soutenue le 17 décembre 2007

devant le jury composé de :

AHMED NACER Mohamed	Professeur à l'USTHB	*Co-directeur*
BADACHE Nadjib	Professeur à l'USTHB	*Président*
BERRACHEDI Abdelhafid	Professeur à l'USTHB	*Examinateur*
KONIG Jean-Claude	Professeur à l'université de Montpellier	*Directeur de thèse*
SAHNOUN Zaidi	Professeur à l'université de Constantine	*Examinateur*
TALBI El-Ghazali	Professeur à l'université de Lille	*Examinateur*

Résumé

L'optimisation combinatoire regroupe une large classe de problèmes ayant des applications dans de nombreux domaines de l'industrie. Ces problèmes ont souvent été abordés comme des problèmes monoobjectifs alors que les problèmes d'optimisation issus de problématiques réelles sont, la plupart du temps, de nature multiobjectif car plusieurs objectifs sont à considérer simultanément. Optimiser un tel problème relève donc de l'optimisation combinatoire multiobjectif.
Depuis plusieurs années, de nombreuses techniques ont été mises au point pour la résolution, exacte ou approchée, de problèmes d'optimisation NP-difficiles. C'est dans cette optique que cette thèse propose, au travers de plusieurs problèmes d'optimisation mono et multiobjectif, des approches de résolution à base de métaheuristiques séquentielles, parallèles et coopératives. Ces problèmes d'optimisation touchent aux domaines des systèmes parallèles, de logistique, ordonnancement d'ateliers et de bioinformatique. Plus précisément, cette thèse traite des approches de résolution à base de métaheuristiques hybrides, séquentielles et parallèles (algorithmes génétiques, recherche tabou et colonie de fourmis), et de la coopération entre une méthode exacte, la programmation par contraintes, et la méthode de colonie de fourmis. Ces différentes approches de résolution sont mise en œuvre au travers du problème d'ordonnancement de tâches sur une architecture parallèle, du problème de couverture d'ensembles, du problème du repliement de protéines, du problème du *flow-shop* bi-objectif et du problème bi-objectif de tournées de véhicules avec fenêtres de temps.

Mots clés : optimisation combinatoire, optimisation monoobjectif, optimisation multiobjectif, métaheuristique, programmation par contraintes, coopération et hybridation.

Abstract

Combinatorial optimization encompasses a large set of problems with numerous applications in various fields of Industry. Those have often been approached as mono-criterion problems while the consideration of several criteria would have been more relevant. Obviously, optimizing problems of that category falls within the theory of multi-objective combinatorial optimization.
Over the last few years, several techniques have been developed for solving or approximating NP-hard optimization problems. Along the same line of research, we undertake to present several metaheuristic-based approaches of our own making to solve mono- and multiobjective optimizing problems in this thesis. Our approaches are sequential, parallel or cooperative. The problems of interest to us range over a broad spectrum of applications pertaining to such diverse fields as parallel systems design, logistics, job shop scheduling and computational biology. More specifically, our thesis deals with hybrid metaheuristic-based approaches that combine sequential, parallel, as well as cooperative features (genetic algorithms, tabu search and ant colonies) along with a cooperation between an exact method, constraint programming, and the method of ants colonies. These methods have all been implemented through a scheduling problem on a parallel architecture together with the set cover problem, the bi-objective flow-shop problem, the protein folding problem and the vehicle routing problem with time windows.

Key words: combinatorial optimization, mono and multicriteria problems, metaheuristics, constraint programming, hybrid methods.

Remerciements

Je tiens à remercier vivement et particulièrement, *Jean-Claude KONIG et Mohamed AHMED NACER* pour l'intérêt qu'ils ont accordé à la direction de mon travail de recherche, pour leurs conseils avisés et leurs encouragements amicaux.
Jean-Claude, je te remercie de m'avoir accueilli, par le passé, dans ton équipe de recherche, pour ta confiance, ton soutien sans failles et tes remarques pertinentes et constructives.
Mohamed, je suis très reconnaissant de la grande gentillesse, de la disponibilité dont tu as toujours fait preuve, de tes multiples et incessantes relectures de mon manuscrit, de tes conseils avisés et de ton soutien.

Je tiens à exprimer ma gratitude à *Nadjib BADACHE, Abdelhafid BERRACHEDI, Ziadi SEHNOUN et El-Ggazali TALBI* pour avoir accepté de rapporter mon travail et pour leur participation très enthousiaste au jury. Leurs remarques et conseils m'ont permis, sans aucuns doutes, d'améliorer la rédaction du document final.

Je remercie mes collègues d'IBISC pour leur soutien et pour l'ambiance fort sympathique dans laquelle j'ai évolué durant trois années. En particuliers *Evripidis Bampis et Eric Angel* pour m'avoir aidé, encouragé et soutenu durant toute la période que j'ai passé au sein de l'équipe OPAL.

Mes derniers mercis, mais non les moindres, je les adresse à ma famille, ma femme et nos enfants. Merci de votre appui et de votre patience.

Prologue

Cette thèse s'inscrit dans le domaine de l'optimisation combinatoire monoobjectif et multiobjectif. Elle a été réalisée en partie à l'USTHB en Algérie, et depuis octobre 2002, au sein de l'équipe *Optimisation discrète et Algorithmique* (OPAL) du *Laboratoire Informatique, Biologie Intégrative et Systèmes Complexes* (IBISC) de l'université d'Evry-Val d'Essonne.

Ce travail dirigé par le professeur *Jean-Claude KONIG* (ancien chef d'équipe d'OPAL) s'inscrit dans la continuité de mes travaux de recherche dans le cadre d'un magister en informatique (soutenu en 1992 à l'USTHB). Portant principalement sur les problèmes d'ordonnancement appliqués aux systèmes informatiques. Mes activités de recherche se sont poursuivies en m'intéressant à des approches d'optimisation combinatoire de problèmes mono et multiobjectif touchant à la science de l'ingénieur, à la logistique, à la bioinformatique et à l'informatique parallèle.
Mes travaux s'intègrent parfaitement dans les thématiques de l'équipe OPAL et constituent un complément à d'autres travaux de l'équipe, en particulier la thèse de *Laurent Gourvès* [Gou 05] sur l'optimisation multicritère et l'approximation.

Certains aspects de mes travaux, en l'occurrence la coopération entre la programmation par contraintes et la métaheuristique de colonie de fourmis, ont été abordés dans le but de proposer des schémas de résolution réutilisables de problèmes d'optimisation complexes. Ces approches de résolutions génériques dont nous avons présenté une partie, en 2004, lors d'une rencontre des chercheurs du projet *Dimensionnement Réseaux, Programmation par Contraintes et Recherche Opérationnelle* (FADO) du *Réseau National des Technologies Logicielles* (RNTL), auraient pu servir ce projet si ce n'est le problème de temps. Bien que le projet FADO soit arrivé à terme, nous avons continué l'expérimentation de nos approches de coopération en les testant sur le problème du repliement de protéines. Ces travaux ont fait, entre autres, l'objet d'un stage de DEA [Bou 04].

Sommaire

Introduction générale

Cette thèse décrit des travaux de recherche dans le domaine de l'optimisation combinatoire [Pap 82][Coo 98][Pap 98]. Un problème combinatoire est d'habitude caractérisé par un ensemble de variables de décision discrètes et par une (respectivement plusieurs) fonction(s) objectif(s) qui associe(nt) à chaque solution (respectivement un ensemble de solutions) son coût (respectivement son vecteur coût), ou son intérêt. Il peut exister, par conséquent, un nombre fini, mais extrêmement grand, de solutions. Ainsi, dans un cadre monoobjectif, la résolution d'un problème d'optimisation combinatoire consiste à déterminer une solution qui optimise une fonction objectif donnée. Dans le cadre multiobjectif [Deb 01][Coe 02][Col 02][Ehr 02][Tki 02a], la résolution d'un problème consiste à trouver un ensemble de solutions optimisant simultanément les différents objectifs.

Dans ce contexte, la classe de problèmes NP-difficiles [Gar 79][Coo 98] rassemble des problèmes pour lesquels il n'existe pas d'algorithme qui fournit la (les) solution(s) optimale(s) en un temps polynomial en la taille de l'instance, sauf si P=NP. Face à cette situation, une question naturelle s'impose : quels plans d'attaque adopter pour résoudre un problème d'optimisation combinatoire, c'est-à-dire comment déterminer la (les) meilleure(s) solution(s) dans un espace de recherche vaste et ceci dans un temps *raisonnable* ?
Puisque le phénomène d'explosion combinatoire nous interdit l'exploration complète de l'espace de recherche dans un temps raisonnable, des méthodes ont été conçues pour restreindre la portion de l'espace explorée. Ces méthodes, dites *heuristiques*, s'appuient sur des propriétés spécifiques du problème à résoudre. Ces propriétés sont connues ou sont acquises par l'heuristique au cours de l'exploration de l'espace de recherche. Autrement dit, ce sont des recherches guidées par des "astuces" qui *dépendent* du problème traité. Ces heuristiques utilisent les particularités du problème traité afin d'engendrer, de manière partiellement stochastique ou non, de bonnes solutions. Ces approches sont appelées *heuristiques constructives*. Parmi ces approches, les algorithmes gloutons sont bien connus. Ils consistent à créer une (des) solution(s) pas à pas, en exploitant les particularités du problème. Pour le *problème du voyageur de commerce*, par exemple, un algorithme glouton basique serait de visiter, à chaque itération, la ville non visitée la plus proche.

Les *métaheuristiques*, telles que les *algorithmes génétiques*, le *recuit simulé*, les algorithmes de *colonies de fourmis*, la *méthode tabou* …, sont des méthodes qui ne sont pas spécifiques à un problème particulier ; elles n'ont aucune connaissance *a priori* du problème à résoudre [Bäc 00]. Le fonctionnement des métaheuristiques, centre d'intérêt de notre travail, est plutôt simple : d'une manière générale, ces méthodes s'appuient sur une ou plusieurs solutions, de qualités (coûts) quelconques, qu'elles tentent d'améliorer, pas à pas, au fil des itérations, en les modifiant partiellement.
Ces algorithmes intègrent des principes de résolution de problèmes combinatoires très généraux ; de ce fait, ils s'adaptent aisément à la résolution de nombreux problèmes. Cependant, la puissance d'une métaheuristique est d'abord liée à son aptitude à intégrer des connaissances spécifiques du problème afin d'améliorer ses performances. La connaissance du problème la plus fondamentale réside dans le codage du problème et dans le choix de la fonction de voisinage. Les métaheuristiques tentent également d'améliorer leur efficacité en incorporant des connaissances supplémentaires dans leurs opérateurs. Plus les opérateurs d'une méthode utilisent des connaissances spécifiques, plus la méthode dispose de moyens potentiels pour conduire efficacement la recherche. En contrepartie, l'intégration de ces

connaissances spécifiques (en supposant que ces connaissances soient disponibles) nécessite un effort pour spécialiser ou adapter la méthode.

Depuis quelques années, on constate que les métaheuristiques les plus performantes sont des méthodes hybrides, c'est-à-dire des méthodes combinant deux ou plusieurs métaheuristiques. A l'origine de l'hybridation des métaheuristiques se trouvent des constatations selon lesquelles certaines méthodes auraient tendance à trop *explorer* l'espace de recherche (*diversification*), alors que d'autres, au contraire, auraient plutôt tendance à faire plus d'*exploitation* des solutions déjà rencontrées (*intensification*). Ainsi, la conjugaison, dans une méthode hybride, de métaheuristiques aux comportements complémentaires, fournirait l'équilibre recherché entre diversification et intensification, donnant une métaheuristique performante. D'autres approches de *coopération/hybridation*, entre métaheuristiques, heuristiques spécifiques ou approches exactes, sont apparues dernièrement [Bac 98][Tal 02]. Une grande partie de ces travaux a comme objectif de proposer des coopérations non seulement performantes mais aussi réutilisables et permettant de dissocier, autant que possible, la représentation du problème de sa résolution. Dans cette perspective, nous nous sommes intéressés à une méthode de résolution exacte : *la programmation par contraintes* [Bar 98][Bar 99][Lus 01]. Cette dernière est une technique générale de résolution de problèmes d'optimisation combinatoire. Elle se situe au carrefour de l'intelligence artificielle et de la recherche opérationnelle. L'efficacité de la programmation par contraintes tient au fait qu'elle permet de dissocier la représentation du problème (définition des contraintes et des objectifs) de sa résolution, réalisée par le système. Notre but est d'exploiter sa coopération avec les métaheuristiques.

Une approche complémentaire, pour obtenir des temps de calcul raisonnables, est l'emploi de calculateurs puissants, c'est-à-dire intégrant plusieurs processeurs. Ainsi, pour gagner en efficacité, il convient d'intégrer à l'optimisation combinatoire des techniques de parallélisme pour concevoir des méthodes d'optimisation parallèles plus performantes. En effet, l'utilisation du parallélisme a permis d'obtenir des résultats satisfaisants, sur des instances de grandes tailles, pour des problèmes d'optimisation tels que le problème du voyageur de commerce [Cha 93a], le problème de tournées de véhicules [Gar 94], le problème de *job shop* [Fal 97], le problème de l'affectation quadratique [Cha 93b][Bac 96], ou le problème de partitionnement de graphes [Tal 91a][Tal 91b], ….

Dans cette thèse, nous nous sommes intéressés à la classe des métaheuristiques dans les cadres monoobjectif et multiobjectif ainsi qu'à leur parallélisation.
Dans le cas *monoobjectif*, un de nos objectifs est la mise en oeuvre de métaheuristiques et de techniques de parallélisation pour aboutir à la résolution de problèmes d'optimisation complexes. Nous étudions la résolution de deux problèmes classiques d'optimisation NP-difficiles. Le premier est le *problème d'ordonnancement de processus sur une architecture parallèle* [Lo 88][Kon 89][Fal 91][Tal 91b][Tal 92][Mun 93][Rah 95]. L'objet de ce travail est la conception d'un algorithme tabou parallèle, ainsi que d'autres heuristiques simples, parmi lesquelles un algorithme de liste.
Le second problème que nous étudions est *le problème de couverture d'ensembles*. Deux schémas d'hybridation, entre la métaheuristique des colonies de fourmis et la recherche locale, sont proposés, implémentés et testés sur des instances de grande taille de la littérature. Le premier schéma fait évoluer les deux algorithmes hybridés de manière séquentielle, alors que le second les fait évoluer simultanément. De plus, deux modèles génériques de parallélisation de l'algorithme hybride le plus performant sont implémentés.

Nous proposons aussi deux schémas de coopération génériques entre la métaheuristique de colonies de fourmis et la programmation par contraintes. Nous avons choisi le problème du repliement de protéines pour mettre en œuvre ce type de méthodes [Dil 95][Bac 01].
Plusieurs algorithmes sont proposés : deux modèles de programmation par contraintes (un algorithme de base et un algorithme plus élaboré), une recherche locale, un algorithme de colonies de fourmis et plusieurs combinaisons de ces différents algorithmes.

Dans le cas *multiobjectif*, nous proposons une étude expérimentale de différentes techniques d'optimisation au travers d'une métaheuristique à base de population. A cet effet, une approche Pareto basée sur les algorithmes génétiques est présentée, ainsi que l'introduction de plusieurs mécanismes : l'élitisme, la diversification, l'hybridation, le parallélisme, etc. Cet algorithme a été validé par son application à deux problèmes. Le premier est le problème du *flow-shop*, où les objectifs à optimiser sont la minimisation du temps de terminaison et celui des retards des tâches. Quant au second, il s'agit du *problème bi-objectif de tournées de véhicules avec des fenêtres horaires*, où les objectifs à optimiser sont la distance totale parcourue par les véhicules et le nombre de véhicules utilisés.

Cette thèse est articulée en deux parties.
La ***partie 1*** concerne l'utilisation de métaheuristiques pour la résolution de problèmes d'optimisation discrets monoobjectifs.

Le ***chapitre 1*** traite de l'optimisation monoobjectif avec des algorithmes hybrides, ainsi que leur parallélisation. Ces concepts sont appliqués au problème d'ordonnancement de processus sur une architecture parallèle. Les différentes approches de résolution de ce problème, dans la littérature, sont séquentielles. Notre but est de fournir un outil d'aide à sa résolution, en proposant une version parallèle de la métaheuristique tabou.
Dans le ***chapitre 2***, nous présentons des algorithmes de résolution basés sur la métaheuristique des colonies de fourmis. Deux schémas d'hybridation entre les colonies de fourmis et la recherche locale sont proposés, implémentés et testés sur des instances de grande taille du problème de couverture d'ensembles. Deux modèles parallèles sont également implémentés.
Quant au ***chapitre 3***, il propose des algorithmes génériques de coopération entre la programmation par contraintes et les colonies de fourmis. Ces algorithmes sont illustrés sur le problème du repliement de protéines qui est NP-difficile pour le modèle H-P. L'implémentation et l'expérimentation de ces différents algorithmes sont réalisées sous l'environnement de programmation *Mozart-Oz* [Moz 04].

La partie 2 traite de l'optimisation multiobjectif, en adoptant l'approche courbes de Pareto.

Le chapitre 4 propose une étude expérimentale de différentes techniques d'optimisation pour le problème du *flow-shop* bi-objectif. Une approche Pareto basée sur les algorithmes génétiques adaptés au cas multiobjectif est présentée, avec diverses stratégies de sélection, de maintien de la diversité de la recherche et d'hybridation. Leurs performances sont testées et comparées. De plus, nous proposons un modèle parallèle de l'algorithme génétique présenté. Parallèlement à ce travail d'optimisation, nous proposons un générateur d'instances du problème de *flow-shop* bi-objectif, palliant ainsi au manque de benchmarks dans ce cadre. Ce générateur se présente comme une extension des instances de Taillard qui ont été proposées dans le cadre monoobjectif [Tai 93].

Le chapitre 5 traite du problème de tournées de véhicules avec des fenêtres horaires dans sa version bi-objectif. Après avoir énuméré les principales caractéristiques de cette classe de problèmes, nous proposons des algorithmes génétiques utilisant l'approche Pareto ainsi que, entre autres, des mécanismes de sélection et de diversification.

Enfin, nous proposons en conclusion un bilan de nos travaux de recherche, ainsi qu'une analyse critique des différentes études menées. Quelques perspectives à ce travail sont également proposées.

Partie 1

Métaheuristiques hybrides pour l'optimisation combinatoire monoobjectif

Cette partie est consacrée aux méthodes de recherche itératives dans le cadre de l'optimisation monoobjectif. Des métaheuristiques séquentielles, parallèles et coopératives sont proposées et testées sur trois problèmes d'optimisation combinatoire : le problème d'ordonnancement de processus sur une architecture parallèle, le problème de couverture d'ensembles et le problème de repliement de protéines.

Chapitre 1

Parallélisation de la méthode tabou pour le problème d'ordonnancement de processus sur une architecture parallèle

Le problème d'ordonnancement sur une architecture parallèle est fondamental pour l'utilisation efficace des machines parallèles. Ce problème est NP-complet et a été largement étudié. L'objet de ce travail est d'étendre les études effectuées en appliquant des métaheuristiques au problème d'ordonnancement statique d'un ensemble de tâches (sans duplication et sans préemption) d'une application, sur une architecture multiprocesseur à mémoire distribuée, avec comme objectif la minimisation de la durée totale d'exécution du système de tâches ("makespan"). A cet effet, nous avons proposé des métaheuristiques séquentielles et parallèles, dont nous exposons les différents résultats expérimentaux, ainsi que l'analyse. Ces résultats sont comparés à ceux obtenus avec d'autres métaheuristiques [Rah 98]. Cette approche est, entre autres, justifiée par une étude de complexité de ce problème.

Les travaux décrits dans ce chapitre ont fait l'objet d'une publication dans le Journal of Combinatorial Mathematics and Combinatorial Computing JCMCC [Rah 04].

1.1. Introduction

Face à une demande sans cesse croissante de puissance de calcul, le parallélisme s'est imposé au cours des dernières décennies comme une solution réaliste, aux perspectives prometteuses, dans le domaine de la conception des ordinateurs.
En revanche, les gains de performance des ordinateurs séquentiels sont de moins en moins significatifs et tendent à buter sur des difficultés techniques liées aux limites physiques même du modèle. À moins donc d'une véritable percée technologique dans ce domaine, comme les recherches récentes sur les ordinateurs optiques ou les supraconducteurs permettent de l'espérer à long terme, on ne prévoit pas d'amélioration notable des performances du modèle séquentiel dans un avenir proche.
Remarquons que les deux solutions ne sont pas exclusives l'une de l'autre et que les solutions matérielles pourront intégrer les ordinateurs parallèles de demain, qui ne seront que plus puissants.
Différentes catégories de machines parallèles, constituées d'un ensemble de nœuds de travail, existent actuellement sur le marché. Au nombre de celles-ci, figurent les machines SIMD et MIMD, que nous allons brièvement décrire :

- SIMD (*Single Instruction Multiple Data*) : À chaque cycle de leur horloge globale, ces machines exécutent une même instruction sur chaque processeur ; cette instruction s'appliquant à des données différentes. Le *Maspar*, et la *CM2* de *TMC* font partie de cette catégorie.
- MIMD (*Multiple Instructions Multiple Data*) : Cette deuxième catégorie comprend les machines, où chaque nœud possède son propre programme et travaille sur ses propres données. La mémoire peut être soit partagée, soit distribuée selon le modèle. Ce dernier type de machine, MIMD à mémoire distribuée, est le type de machine qui nous intéressera tout au long de ce chapitre, en raison essentiellement de la prévalence de ce modèle sur le marché et du succès qu'il semble connaître auprès de nombreux constructeurs (*SP1 d'IBM*, *CS2* de *PCI*, *Paragon d'Intel*, *CM5 de TMC*, *T3D de Cray*). Notons également l'existence de nombreux modèles apparentés, tels que : le modèle réseau d'interconnexion multi-étages, qui nous servira de base pour le placement que nous proposons par la suite, et le modèle réseau point à point [Bok 81].

La plupart des outils d'aide à la programmation d'architectures parallèles [Cos 93][Gol 94] obligent l'utilisateur à définir son propre placement de tâches [Tal 91b], grâce à des primitives spécifiques. Le programme devra alors être réécrit si on l'exécute sur une autre configuration du réseau d'interconnexion, ce qui limite la portabilité des programmes et met en exergue, par ailleurs, l'intérêt que peut revêtir un ordonnancement automatique de processus [Tal 92][Gol 94].
On distingue deux types d'ordonnancement suivant l'instant où le placement est décidé. On dit qu'un ordonnancement est statique (respectivement dynamique) lorsque le placement des tâches se fait à la compilation (respectivement à l'exécution). Dans tous les cas, l'ordonnancement [Bla 87] consiste, pour chaque tâche, à affecter un processeur à son traitement et à déterminer sa date de début d'exécution, en vérifiant les contraintes de précédence. Cette allocation doit maximiser l'utilisation du parallélisme, en distribuant de manière équitable les tâches sur l'architecture multiprocesseur. En outre, elle doit minimiser les coûts de communication dans le système [Mun 93].

L'allocation statique [Tal 91b][Tal 93] (ordonnancement statique), objet de notre travail, a de nombreuses applications. D'une part, elle peut servir de point de départ à un

algorithme d'allocation dynamique. Elle est aussi utilisée dans les applications temps réel et graphiques [Lut 94]. D'autre part, les méthodes d'allocation statique sont aussi efficaces dans le calcul scientifique [Kon 89] et l'algèbre linéaire, comme, par exemple, lors du calcul d'un produit de deux matrices, ou la résolution d'équations différentielles, de systèmes linéaires, etc. Une instance d'un problème d'ordonnancement statique est définie par l'ensemble des traitements appelés tâches ou processus, la durée d'exécution de chacune de ces tâches, les contraintes de précédence auxquelles elles sont soumises, les durées de transfert des données, le protocole de communication et la topologie de l'architecture sur laquelle s'exécutera l'application.

L'ordonnancement de processus concurrents sur les architectures multiprocesseurs est NP-difficile dans le cas général [Bla 87]. La plupart des algorithmes proposés dans la littérature pour résoudre le problème d'ordonnancement sont séquentiels [Lo 88][Kon 89][Fal 91][Tal 91b][Mun 93][Rah 95]. Le but de notre travail est d'étendre certaines études effectuées en appliquant la méthode *tabou* au problème d'ordonnancement statique d'un ensemble de tâches (sans duplication et sans préemption) d'une application sur une architecture multiprocesseur à mémoire distribuée [Rah 97], pour minimiser la durée totale d'exécution du système de tâches. A cet effet, deux algorithmes *tabou*, l'un séquentiel et l'autre parallèle, ont été proposés. L'algorithme parallèle développé est général. En effet, il est indépendant de l'architecture sur laquelle il s'exécute et, d'autre part, aucune hypothèse n'est faite sur la structure de l'application. La mise en oeuvre et l'évaluation de performance de cet algorithme ont montré qu'il possède une accélération (*speedup)* presque linéaire et de bonnes performances globales en termes de qualité de la solution et de temps d'exécution de l'algorithme.

La section 1.2 de ce chapitre est consacrée à la présentation et à la formulation du problème étudié. La section 1.3 donne un aperçu sur l'état de l'art des approches de résolution de ce problème. La section 1.4 présente des généralités sur la recherche tabou. La section 1.5, quant à elle, présente deux algorithmes séquentiels de recherche tabou. Quant à la section 1.6, elle expose l'algorithme parallèle proposé. La section 1.7 présente les différents tests expérimentaux entrepris, leur analyse, ainsi qu'une étude comparative entre les différents algorithmes développés (algorithmes génétiques, recuit simulé et algorithmes hybrides [Rah 98]). La dernière section montre que le problème est NP-complet même pour des cas particuliers portant soit sur l'architecture, soit sur la relation de précédence. Elle montre également que le problème n'admet pas de schéma d'approximation polynomial à moins que P=NP.

1.2. Le problème d'ordonnancement

Un traitement informatique est une succession de transformations élémentaires d'un ensemble de données initiales. Ces transformations peuvent être regroupées pour former un ensemble de traitements partiels appelés tâches. La succession des tâches dans un ordre déterminé fournit le même résultat que le traitement global. Si l'un de ces traitements a pour résultat une donnée initiale d'un autre, il doit alors impérativement le précéder. Par ailleurs, si deux traitements sont indépendants, ils peuvent s'exécuter dans un ordre quelconque.

1.2.1. Modèle de tâches

L'outil de base que nous adoptons pour représenter un algorithme parallèle est le graphe de précédence des tâches, où une tâche élémentaire est un ensemble d'instructions entièrement définies par ses entrées, ses sorties et son temps d'exécution. Les transferts de données induisent des relations de précédence entre ces tâches. Une tâche reçoit des données de ses prédécesseurs immédiats, effectue des calculs sur ces données et envoie des résultats en sortie vers ses successeurs immédiats.
Un transfert de données d'une tâche T_i vers une tâche T_j prend un temps qui ne peut être négligé. Ce temps de communication [Mit 93][Mun 93] dépend du nombre de mots transférés et de la distance entre les processeurs affectés aux tâches T_i et T_j. Par contre, si le même processeur est affecté aux deux tâches, nous pouvons admettre que le temps de communication est nul.

Une application est décrite par un graphe orienté $G_T = (V_T, E_T)$. Ce graphe est connexe et sans circuit (DAG), où :
- L'ensemble des sommets V_T correspond à l'ensemble des tâches. Il existe un arc entre deux sommets T_i et T_j si et seulement si les deux tâches communiquent.
- L'arc de T_i vers T_j signifie que la tâche T_j ne peut commencer son exécution que lorsque la tâche T_i a terminé la sienne.
- La valeur associée à un sommet T_i indique la durée d'exécution d_i de la tâche T_i. La valeur associée à l'arc (T_i, T_j) représente le nombre Y_{ij} d'unités de données échangées entre les tâches T_i et T_j .

1.2.2. Modèle de l'architecture

Les architectures MIMD à mémoire distribuée concernent les calculateurs où chaque processeur dispose d'une mémoire indépendante de données et de programme. Dans ce modèle, dont le mode de fonctionnement est asynchrone, chaque processeur peut exécuter un processus différent. Les échanges inter-processeurs s'effectuent par transmission de message. Les processeurs sont connectés par des liens de topologies fixes (grille, arbre, hypercube,…) ou re-configurables (*Crossbar*, *réseau de Clos*, *réseau de Benes*, …) [Wu 84].

À l'actif de ce modèle, signalons également que les réseaux de stations, qui sont très répandus actuellement, fonctionnent aussi en mode MIMD, avec une communication par transmission de messages qui peut être gérée par PVM (*Parallel Virtual Machine*) sur un réseau local ou d'interconnexion.
En fait, un réseau de stations se présente souvent comme une architecture hybride, dans le sens où il peut y avoir une combinaison, sur un réseau local, de stations et de machines multiprocesseurs, gérant leur parallélisme interne par mémoire partagée, ou par bus ou réseau.

Le modèle qui nous intéresse est constitué de plusieurs processeurs identiques, dont chacun est composé d'une unité de contrôle, d'une unité de traitement et d'une mémoire locale à accès rapide. Ces processeurs communiquent par échange de messages au moyen d'un réseau d'interconnexion. La classe d'architectures utilisée est connue sous le nom d'architecture MIMD à mémoire distribuée, ou MIMD faiblement couplée [Dun 90][Cos 93][Hen 96].
La topologie du système est décrite par un graphe non orienté connexe $G_P=(V_P,E_P)$. L'ensemble des sommets V_P correspond à l'ensemble des processeurs. Il existe une arête entre

deux sommets si et seulement si les deux processeurs sont connectés directement. Une arête $(k,l) \in E_P$ est pondérée par C_{kl}, le temps de transfert d'une unité de donnée entre P_k et P_l (distance entre P_k et P_l comme poids dans G_p). On fait l'hypothèse que le temps de communication intra-processeur est nul, c'est-à-dire $C_{kk} = 0$ pour tout processeur k.

1.2.3. Formulation du problème d'ordonnancement

Pour formuler notre problème, nous avons besoin d'une définition supplémentaire. Soit (T_i, T_j) un arc de G_T, nous dirons qu'un processeur P_k *connaît* (T_i, T_j) au temps t si P_k a déjà exécuté T_i ou que les données nécessaires pour exécuter T_j après T_i ont déjà été transférées à P_k au temps t.
Maintenant, nous pouvons énoncer notre problème d'ordonnancement statique de tâches comme suit :

Soient un ensemble de tâches G_T et un réseau de processeurs G_P. L'ordonnancement de toutes ces tâches, avec comme objectif la minimisation du *makespan*, tient compte des contraintes suivantes :

(1) Les tâches sont non préemptives.

(2) A chaque instant, un processeur est dans l'un des états (en exclusion mutuelle) suivants : (a) exécution d'un processus, (b) inactif, (c) transfert de données vers un autre processeur.

(3) Un processeur P peut exécuter une tâche T_j seulement si P *connaît* tous les arcs (T_i, T_j).

(4) Un processeur ne peut pas communiquer avec plus d'un processeur simultanément.

(5) Si T_i est affectée à P_k et T_j à P_m, le coût de la communication pour le transfert de données de T_i à T_j est $(Y_{ij} * C_{km})$.

L'exemple suivant d'un ordonnancement réalisable, illustre comment la communication est prise en considération dans notre type de modèle.

L'exemple de la figure 1.1 montre que, par exemple, la tâche T_2 ne peut pas débuter son exécution à l'instant 1 (après la tâche T_1) à cause de la contrainte (2).
La tâche T_3 débute son exécution à l'instant 6 sur le processeur P_1, car c'est à cet instant que P_1 termine sa phase de communication avec P_3. De plus, bien que T_1 et T_3 soient liées par une contrainte de précédence, il n'y aura pas de communication, car elles sont exécutées sur le même processeur P_1.

1.3. Quelques méthodes de résolution

Les algorithmes gloutons et les *algorithmes de liste* sont très utilisés pour les problèmes d'ordonnancement. Un *algorithme de liste* consiste à établir tout d'abord une liste de priorités des tâches suivant certains critères (par exemple, coût de calcul, nombre de fils, etc.) et ensuite à placer les tâches en suivant l'ordre de cette liste sur les différents processeurs. Les listes doivent respecter les relations de précédence inter-tâches ; en fait, un ordre total est constitué à partir de l'ordre partiel des contraintes de précédence. La compacité, propriété de certains algorithmes de liste, consiste à ne pas laisser un processeur inactif alors qu'il existe une tâche prête à être exécutée.

La plupart des problèmes d'ordonnancement et de placement sont NP-difficiles [Gar 79] et la recherche d'une solution optimale (minimisant le temps d'exécution du programme parallèle) demande l'utilisation d'algorithmes de complexité exponentielle.

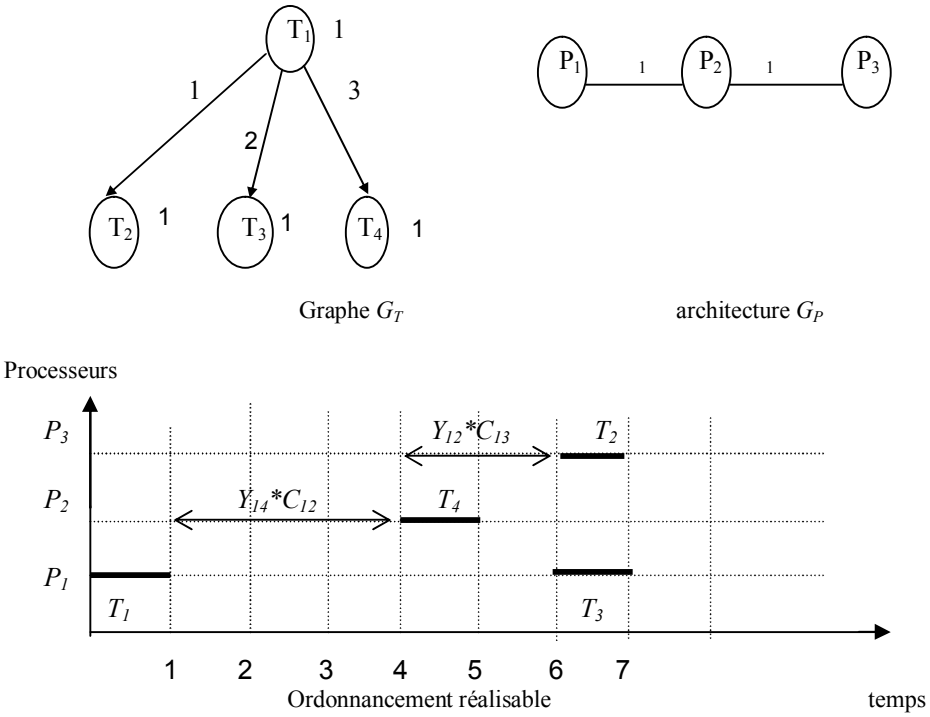


Figure 1.1. Exemple d'un ordonnancement réalisable pour une instance particulière.

On peut distinguer quelques uns des résultats existants suivant les coûts de calcul et de communication :

- Les graphes *UET* (*Unit Execution Time*) caractérisés par un coût d'exécution unitaire des différentes tâches et par l'absence de coût de communication. Coffman et Graham [Cof 72] ont trouvé un algorithme polynomial qui permet d'ordonnancer de manière optimale un DAG UET sur deux processeurs. Pour $m \geq 2$ et fixé (où m représente le nombre de machines), le problème reste ouvert. Avec m non borné, le problème devient simple à résoudre (une solution triviale serait de placer chaque tâche sur un processeur différent). Des résultats existent aussi pour des familles de graphes spécifiques telles que les arbres. Ainsi Hu [Hu 61] a obtenu un résultat d'optimalité en temps polynomial. Coffman-Graham ont en outre établi le rapport de performance (au pire) suivant pour leur algorithme : si ε représente la date de fin d'exécution pour l'ordonnancement fourni par cet algorithme et ε_{opt} l'optimum global, le rapport est : $\varepsilon/\varepsilon_{opt} \leq 2-2/m$ [Cof 72], où m est le nombre de processeurs. L'ordonnancement de tâches indépendantes sur un ordinateur parallèle non-préemptif (pas de pseudo-parallélisme) avec, pour objectif, la minimisation du temps total d'exécution, est un problème NP-complet [Gar 78].
- Les graphes *UECT* (*Unit Execution and Communication Time*) où les coûts de communication externe à un processeur sont constants et équivalents aux coûts de calcul des tâches. Une borne a été établie pour les algorithmes de liste fournissant des ordonnancements compacts (un processeur ne reste pas inactif si une tâche est prête à

être exécutée) par Rayward-Smith [Ray 87] : $\varepsilon \leq (3-2/m)\ \varepsilon_{opt} - (1-1/m)$. Un algorithme d'ordonnancement optimal pour les arbres a été trouvé par Varvarigou [Var 96].

- Les graphes ayant des coûts de communication quelconques et des coûts de calcul quelconque. Colin et Chrétienne [Col 91a][Col 91b] ont montré qu'en permettant la duplication des tâches, avec un nombre non borné de machines, et avec une hypothèse supplémentaire limitant la taille des communications par rapport aux calculs, on pouvait trouver un algorithme de type chemin critique qui était optimal.

Dans la classe des algorithmes exacts, on trouve de nombreux algorithmes de complexité exponentielle. Nous pouvons représenter sous forme d'arbre la construction de toutes les solutions possibles (soit tout l'ensemble des placements); en partant de la configuration où aucune tâche n'est placée (la racine) vers l'ensemble des tâches placées (les feuilles). Différentes explorations de cet arbre sont possibles :

- En profondeur d'abord, on construit le plus rapidement possible une solution [Pea 90].
- En développant le noeud ayant la plus petite valeur, étant donné la fonction coût et les tâches déjà placées (meilleur d'abord) [She 85].
- En développant le noeud ayant le plus de chance d'être à la base d'une solution optimale ; en tenant compte des tâches déjà placées et en utilisant une heuristique sous-estimant les tâches restant à placer (A*) [Har 68].

Il existe d'autres méthodes d'énumération de l'ensemble des solutions et de recherche de l'optimum. On peut, par exemple, découper le problème en sous-problèmes et envisager une recherche par *Séparation & Evaluation* [Pap 82][Lo 85].

Certaines méthodes permettent de trouver un placement grossier rapidement, c'est le cas des algorithmes gloutons. On trouve, dans cette catégorie, les algorithmes proposés par [Paz 89], dont le seul critère est l'équilibrage de charge. Benhamamouch et Plateau [Ben 89] décrivent un système basé également sur un équilibrage de charge, dans le cas où les communications sont négligeables et modifient le critère si elles deviennent importantes.
Parmi les algorithmes itératifs, citons, entre autres :

- les algorithmes du recuit simulé [Bol 88][Paz 89],
- l'algorithme de Bokhari [Bok 81] qui utilise une fonction coût (le cardinal) basé sur le nombre d'arêtes du graphe de tâches correctement placées sur le graphe des processeurs,
- les algorithmes génétiques [Tal 91a][Rah 95][Rah 97].

1.4. La recherche tabou

On doit la recherche tabou (*TS - Tabu Search*) à Glover [Glo 86][Glo 89][Glo 90] et Hansen [Han 86]. La recherche tabou est une métaheuristique performante qui s'applique à de nombreux problèmes d'optimisation combinatoire [Glo 95].
Comme le recuit simulé [Dré 03], la recherche tabou est une *méthode à solution unique* dont la mémoire contient la solution courante, la meilleure solution visitée depuis le début de la recherche et une valeur qui contrôle l'avancement de la recherche (voir figure 1.2). Le plus souvent, l'avancement de la recherche tabou est contrôlé par un critère d'arrêt basé sur le nombre d'itérations effectuées. Cependant, la mémoire de la recherche tabou est plus riche que celle du recuit simulé. En effet, l'originalité de la recherche tabou vient, entre autres, de

l'utilisation de la notion de *liste tabou*, qui consiste à retenir un historique de la recherche réalisée. Un candidat est déclaré tabou si la *transposition* qui permet de l'obtenir, à partir de la solution courante, a été employée récemment pour engendrer la solution courante. La liste tabou contient donc des *transpositions*.

Contrairement au recuit simulé, la recherche tabou est une méthode déterministe (voir la figure 1.2). Initialement, la recherche tabou se comporte comme une méthode de descente (descente stricte ou descente selon la plus forte pente) jusqu'à ce qu'un optimum local soit atteint. Quand elle se trouve bloquée sur un optimum local, la recherche tabou se poursuit en sélectionnant un candidat de moins bonne qualité que celle de la solution courante. Cette stratégie permet de sortir de l'optimum local, mais la recherche tabou, qui se comporte comme une descente en dehors des optima, pourrait retourner, dès l'itération suivante, à l'optimum qu'elle vient de quitter. Grâce à la liste tabou, ce parcours cyclique indésirable peut être évité. Pour empêcher les cycles, la recherche tabou, se référant à sa liste tabou, ne sélectionne pas les candidats qui sont proches d'une solution visitée récemment, autrement dit les candidats tabous. Cependant, si un candidat tabou est d'une grande qualité, il est tout de même sélectionné, selon un mécanisme appelé l'*aspiration*.

Dans la recherche tabou, le rôle original de la mémoire est de préserver des informations concernant les caractéristiques des solutions courantes successives qui permettent de déterminer le caractère tabou des candidats. L'ensemble de ces caractéristiques est conservé dans la *liste tabou*. Souvent cette liste, qui est mise à jour à chaque itération, est une file du type FIFO d'une taille l prédéterminée, qui est un paramètre de la recherche. Cette liste permet donc de stocker un historique concernant les l dernières solutions courantes. Cette longueur l est un paramètre sensible de la recherche tabou et doit être adaptée à la nature de l'instance. La taille de la liste tabou contrôle le pouvoir d'exploration de la recherche. En effet, quand la liste est courte, la recherche a un rôle d'exploitation. Comme l'historique est court, rien n'empêche la recherche de rester dans la région de l'espace de recherche où elle se trouve et de l'exploiter. Cependant, quand la liste est trop courte, la recherche tabou a tendance à cycler, et par conséquent il est inutile d'effectuer un grand nombre d'itérations et seule une portion très restreinte de l'espace de recherche est parcourue. Au contraire, quand la liste tabou est longue, la recherche sort rapidement de la région où elle se trouve et explore l'espace de recherche. Une recherche tabou dotée d'une grande liste tabou parvient à trouver des régions intéressantes de l'espace de recherche mais ne peut pas les exploiter car elle les traverse sans s'y attarder.

Comme toujours, l'idéal est de trouver le juste équilibre entre l'exploration et l'exploitation. C'est-à-dire trouver la bonne taille de la liste tabou. Cependant, en pratique, une taille de liste fixée pour toute la recherche n'est pas satisfaisante. En effet, comme pour le recuit simulé, il vaut mieux commencer avec un comportement explorateur (grande liste) pour trouver une région intéressante, puis l'exploiter (petite liste). Cependant le recuit simulé tient compte de la qualité des solutions, et plus une vallée est profonde, plus il a de chances d'y être piégé pour l'exploiter ensuite. Mais, à l'inverse, la recherche tabou ne prend pas en compte la qualité des solutions et ne rend pas compte d'une vallée plus profonde que les autres. Ainsi, la stratégie qui consisterait à diminuer la longueur de la liste tabou progressivement, comme pour la température du recuit simulé, n'a pas de sens et n'apporte rien d'intéressant pour la recherche tabou, car elle aboutirait à limiter la recherche dans une région quelconque de l'espace. Cependant, beaucoup d'applications de la recherche tabou utilisent une liste tabou de taille prédéfinie et fixe tout au long de la recherche.

Il existe de nombreuses variantes de la recherche tabou impliquant des techniques plus ou moins sophistiquées pour *intensifier* ou *diversifier* la recherche [Glo 95] (voir figure 1.3). D'une manière générale, une recherche tabou, même dans une version simple, s'avère efficace sur de nombreux problèmes de l'optimisation combinatoire. Pour le problème de *job-shop*, par exemple, la recherche tabou semble être d'une grande efficacité en comparaison à d'autres techniques d'optimisation [Tai 94].

```
Algorithme : tabou_générique

Début
Mémoire :  Solution courante S ; Meilleure solution M ; Liste
tabou ; Itération courante K.
Paramètres : Nombre d'itérations N ; Taille de la liste tabou.



      Choix solution initiale S0
      Solution courante S := S0
      Meilleure solution M := S
      K := 0
a.    - Si K ≥ N
          alors   aller en (c)  fin si
      - K := K + 1
      - Mise à jour de la liste tabou
      - Si φ(S)< φ(M)     /*φ : fonction coût */
            alors M := S fin si
      - Générer les candidats C /* tout ou partie des solutions
                                   voisines de la solution courante S*/
b.    - c := meilleur candidat de C
      - Si φ(c)< φ(S)
          Et Si c n'est pas tabou
          Ou Si c vérifie critère d'aspiration
           alors    S := c
           sinon Oter c de C ;  aller en (b) fin si.
      - aller en (a).
c.    - Retourner M
Fin.
```

Figure 1.2. Méthode de recherche tabou générique.

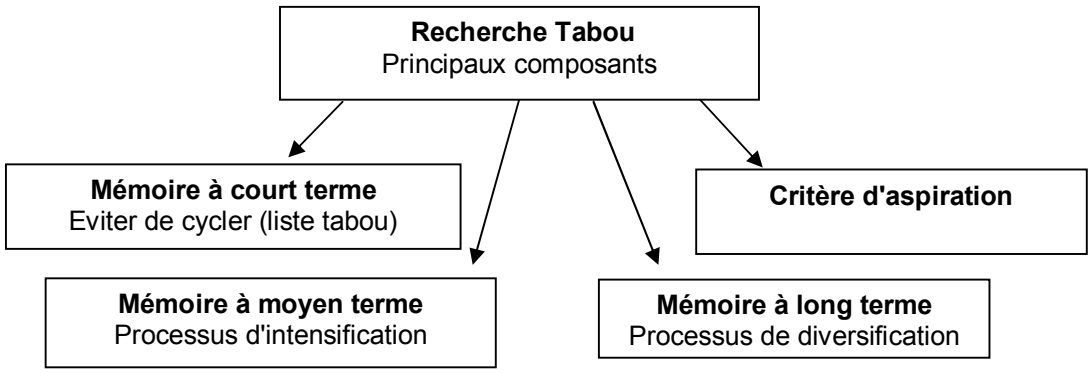


Figure 1.3. Principaux composants de la recherche tabou.

Les différents composants d'une recherche tabou, présentés ci-dessus, seront implémentés dans notre algorithme et détaillés dans les prochains paragraphes.

1.5. Adaptation de la recherche tabou pour le problème d'ordonnancement

Pour l'application de la méthode tabou au problème d'ordonnancement étudié, nous commençons par présenter les éléments de base de cette méthode, puis nous introduirons progressivement de nouveaux éléments, tels que l'aspiration, l'intensification et la diversification. Ainsi, plusieurs algorithmes seront proposés : un algorithme séquentiel simple, un algorithme séquentiel plus élaboré avec de l'intensification et de la diversification, et enfin un algorithme parallèle.

1.5.1. Les paramètres de la recherche tabou

1.5.1.1. Une solution initiale

L'algorithme de liste (ou les méthodes sérielles) est fondé sur un ordre de priorité entre les tâches, permettant de construire une solution approchée. Dans l'algorithme de liste, une heuristique gloutonne permet de trier les tâches selon un ordre de priorité.
A un instant "*t*" on affecte, parmi les tâches prêtes, c'est-à-dire, celles dont tous les prédécesseurs sont achevés, la tâche de plus forte priorité. Donc l'ordonnancement des tâches s'effectue en appliquant la règle : "dès qu'au moins une tâche est exécutable, traiter parmi celles-ci la plus prioritaire de la liste" sur le premier processeur disponible. Dans le cas général, rien ne permet d'affirmer que, pour un problème donné, il existe une liste dont l'ordonnancement associé est optimal.
En effet, les méthodes de liste imposent d'exécuter au moins une tâche prête s'il en existe, alors qu'il peut être plus intéressant de laisser momentanément des ressources inutilisées en vue d'exécuter une tâche plus urgente. Dans notre cas, on n'utilise l'algorithme de liste que pour obtenir une solution de départ que l'on va améliorer par l'application de la recherche tabou. L'ordre de priorité adopté est l'ordre croissant des dates de début au plus tard. Ce choix provient du fait que si plusieurs tâches sont disponibles à un instant donné, le choix se porte sur celle qui, si elle était retardée, rallongerait le plus le temps de l'application. Cette tâche, que nous disons appartenir au chemin critique, a justement la plus petite date de début au plus tard, d'où le choix de cet ordre de priorité. L'algorithme que nous utilisons est présenté dans la figure 1.4.

1.5.1.2. Voisinage d'une solution et liste tabou utilisée

Le voisinage *V(s)* proposé afin de définir les solutions voisines de la solution *s* utilise le mouvement *déplacement,* représenté par *depl* = (*proc_source*, *num_tâche*, *proc_dest*, *ind*). Un déplacement consiste à transférer une tâche *num_tâche* placée sur le processeur *proc_source*, vers le processeur *proc_dest* à la position *ind*.
Le voisinage *V(s)* (tous les déplacements réalisables à partir de *s*) est de grande taille et le seul moyen de déterminer la solution *s'* minimisant la fonction objectif sur *V(s)* est de passer en revue l'ensemble *V(s)* tout entier, ce qui est très coûteux en termes de temps et d'espace mémoire. On préfère alors utiliser la stratégie de la *liste candidate* pour restreindre le nombre

de solutions à examiner dans une itération donnée. Vu l'importance de cette stratégie, et afin d'évaluer efficacement les bons candidats, nous partitionnons le graphe de tâches G_T en niveaux (voir figure 1.5). Ainsi, pour chaque processeur *P*, le voisinage *V(s)* est constitué de tous les déplacements réalisables des tâches, placées sur ce dernier, de niveau minimal *min_niv* vers tous les autres processeurs. Les déplacements des tâches de niveau minimal d'un processeur sont les meilleurs mouvements du voisinage dans une itération donnée ; et représentent, donc, les éléments de la *liste candidate*. En effet, le déplacement de ces tâches n'engendre aucun traitement supplémentaire du fait qu'elles ont un niveau minimal parmi les tâches non encore traitées. C'est-à-dire qu'elles ont le moins de contraintes de précédence.

Algorithme : ***algorithme_liste***

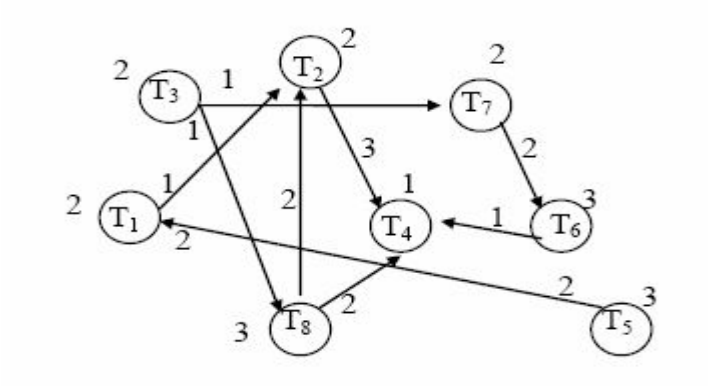
Début

```
Etape 1 :
     ens1 := ∅  /*ensemble des tâches candidates à l'exécution en
                   parallèle
     ens2 := {tâches de l'application} /*ensemble des tâches non
                                        encore affectées
     Initialisation et lecture des données du problème

Etape 2 :
     Déterminer le temps de début au plus tard (séquentiel) de
     chaque tâche et ainsi son ordre de priorité

Etape 3 :
   Tant que (ens2 ≠ ∅)
   faire
    Déterminer les tâches prêtes à s'exécuter et les ajouter à ens1
    Ordonner les tâches de ens1
    Tant que (ens1 ≠ ∅ et ∃ de processeurs disponibles)
       faire
          Choisir la tâche la plus prioritaire
          L'affecter au premier processeur libre
          Mettre à jour l'état des processeurs
          ens2 := ens2 - {Ti}
          ens1 := ens1 - {Ti}
       fait
   fait
FIN.
```

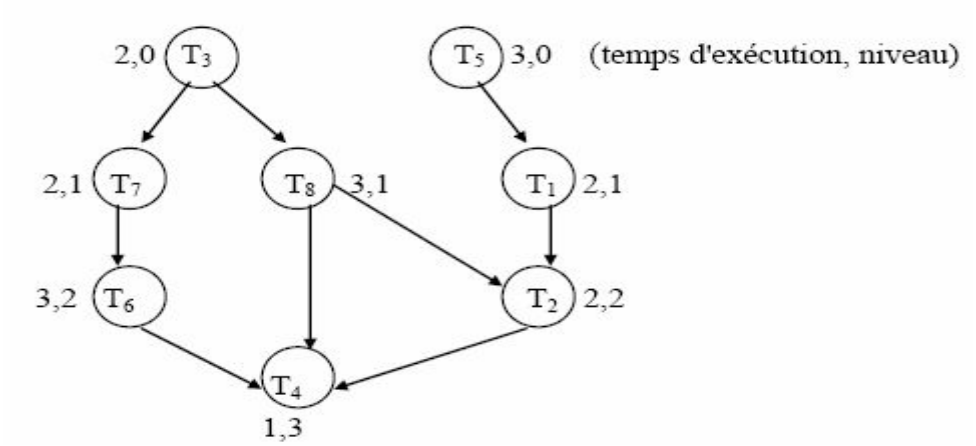
Figure 1.4. Algorithme de liste utilisé.

Chacun des éléments de la liste tabou que nous avons utilisée représente une série d'attributs caractérisant la solution (*déplacement*).
Pour le problème d'ordonnancement étudié, toute solution réalisable *s* est décrite par un ensemble d'attributs, dont la structure est un triplet *(processeur, tâche, position)*.
Ayant un mouvement *depl* = (P_l, T_i, P_k, *ind*), à appliquer à une solution *s*, pour éviter de boucler autour de cette solution, on interdit, pendant les *t* prochaines itérations (*longueur de la liste tabou*), de déplacer T_i vers le processeur P_l dans la position (par rapport au processeur P_l) d'où on vient juste de la déplacer. Autrement dit, à chaque fois qu'un mouvement *depl* = (P_l, T_i, P_k, *ind*) est appliqué à la solution courante *s*, l'attribut (P_l, T_i, $position_{il}$), tel que la $position_{il}$ est récupérée à partir de *s*, est inséré dans la liste *T*.

Un mouvement est dit tabou s'il crée une solution dont l'un de ses attributs appartient à la liste tabou *T*.



Graphe G_T



Graphe G_T décomposé en niveaux.

Figure 1.5. Exemple de partitionnement en niveaux d'un graphe.

Avec *Niveau*(T_i) est égal à 0 si T_i n'a pas de prédécesseur ou bien à 1+ MAX{*Niveau*(T_k)} si T_k appartient à l'ensemble des prédécesseurs de T_i.

L'algorithme pour le calcul du voisinage est le suivant (figure 1.6) :

```
Algorithme : voisinage
Début
Pour (tout processeur P1 de l'architecture GP)
  faire  Déterminer le niveau minimal min_niv des tâches affectées à
         P1(figure 1.5);
   Pour (toute tâche Ti placée sur P1 de niveau min_niv)
      faire pour (tout autre processeur P1 de GP)
            faire  Pour (tout indice ind de Pk)
                    faire
                     Générer le quadruplet depl = (P,T,P,ind)
                     Si     (réalisable     (depl))     /*tester     la
                           réalisabilité de la solution générée par
                           le mouvement depl */
                     alors insérer depl dans la liste candidate ;
                    fin si ;
                   fait ;
             fait ;
```

```
            fait ;
        fait ;
Fin.
```

Figure 1.6. Algorithme de détermination du voisinage : *voisinage*.

Le *critère d'aspiration* est un élément important de la flexibilité de la recherche tabou. L'état tabou d'une solution n'est pas absolu et peut être ignoré si certaines conditions sont satisfaites. Celles-ci sont exprimées sous forme de critères d'aspiration.
Si un mouvement *depl* produit une solution tabou (au moins l'un de ces attributs appartient à *T*) dont le coût est meilleur que toutes les solutions rencontrées jusqu'à présent, son état tabou est alors ignoré et la solution est admise (on parle alors d'*aspiration par objectif*). Le critère de *l'aspiration par défaut* est activé si tous les mouvements disponibles dans la liste candidate sont classés tabou et ne sont pas admissibles à l'aspiration par objectif. Ce critère provoque la sélection du premier mouvement qui perd son état *tabou*.

La recherche tabou s'arrête au bout d'un certain nombre *nbmax* d'itérations entre deux améliorations de la meilleure solutions s^* rencontrée.

L'algorithme tabou proposé est décrit comme suit (figure 1.7) :

```
Algorithme : Tabou 1
Début
Etape 1 : algorithme_liste(s)  /*génération de la solution initiale
          sélectionner une solution s de S ; s* :=s ; k := 0;
Etape 2 : k:=k+1;
          Établir un sous-ensemble S* de solutions de V(s) en
          utilisant l'algorithme voisinage;
Etape 3 : sélectionner la meilleure solution j de S* telle que
          f(j)≤ Asp(f(s)) ou j∉T;  s:=j;
          /*Asp est la fonction d'aspiration qui détermine le premier candidat le moins tabou */
Etape 4 : Si f(s)< f(s*) alors s*:=s;
Etape 5 : mise à jour des conditions tabou et d'aspiration;
Etape 6 : Si le nombre d'itérations maximal est atteint (critère
          d'arrêt vérifié)
          alors retourner s* et arrêt
          sinon aller à Etape 2
Fin.
```

Figure 1.7. Algorithme *Tabou 1*.

1.5.2. Intensification et diversification

La stratégie d'intensification est basée sur la modification des règles de sélection, afin de faire en sorte que les attributs des meilleures solutions rencontrées jusqu'à présent, au cours de la recherche, soient présents dans la solution courante. On cherche donc à encourager de tels attributs.
De Werra et *Hertz* [Dew 89] définissent le compromis à tenir entre intensification et diversification de la manière suivante : "*Une recherche intelligente ne doit pas seulement explorer entièrement les régions de bonnes solutions, mais elle doit aussi avoir une vue générale de l'espace de recherche et s'assurer qu'aucune région n'a été complètement ignorée*". La diversification désigne donc l'ensemble des moyens servant à explorer de nouvelles régions. Souvent ces mécanismes se basent sur une modification des règles de sélection, afin d'introduire des attributs rarement utilisés. Ces attributs peuvent être introduits

de manière partielle ou totale ou en pénalisant les solutions contenant des attributs fréquemment utilisés. Le moment d'introduire la diversification est souvent crucial, par exemple : après un certain nombre d'itérations, si aucune amélioration de *s** n'est rencontrée, ou, après un nombre déterminé d'itérations. La recherche tabou peut aussi se servir de la longueur de la liste des attributs tabou comme moyen de diversification. Quand la progression de la recherche devient lente, la *longueur de la liste* des attributs *tabou* est augmentée. Ce qui a pour effet de favoriser la diversification.

L'algorithme proposé comporte deux étapes :

- La première étape correspond au *processus d'intensification*, qui consiste à appliquer une recherche tabou avec une *longueur de la liste tabou* "normale" pendant au maximum *nb-local1* (nombre d'itérations dans la phase d'intensification) itérations. Un test est alors effectué, pour voir si la progression de la recherche dans la région courante s'est ralentie. Si c'est le cas, la seconde étape est alors exécutée.
- La seconde étape est un *processus de diversification*, dont le rôle est de permettre de s'écarter de la région courante.
  Si au bout de *nb_iter* itérations, aucune aspiration n'est activée et que tous les mouvements de la liste ne sont pas tabous (raison pour laquelle les grandes longueurs de la liste tabou courantes ne sont pas efficaces), alors la longueur de la liste est augmentée et le nombre d'itérations est diminué. Ce processus s'arrête lorsqu'une forte aspiration est activée, ou lorsque tous les mouvements du voisinage sont tabous.

L'algorithme tabou décrit ci-dessus avec la diversification et l'intensification est donné dans la figure 1.8.

1.6. Un algorithme parallèle de la méthode tabou

Parmi les différentes sources de parallélisme dans la recherche tabou, citons : le parallélisme dans la fonction d'évaluation, le parallélisme dans l'examen et l'évaluation du voisinage, le parallélisme dans la décomposition du graphe des niveaux et le parallélisme dans l'exploration de l'espace des solutions, en maintenant différents chemins de recherche dépendants ou indépendants.
Talbi, Hafidi et Geib [Tal 98] proposent une classification de ces approches selon deux catégories : dé *composition de domaines* et *recherche tabou multi-tâches* (figure 1.9).

L'algorithme parallèle que nous avons implémenté, *Parallel_Tabou*, lance une exploration parallèle de l'espace des solutions grâce à des processus de recherche qui calculent automatiquement et dynamiquement les paramètres de la recherche (appelés stratégie) : la taille de la liste tabou (*taille_liste_tabou),* et le nombre maximum d'itérations entre deux améliorations de la meilleure solution trouvée *s**, (*nb_iter)*.
Notre but est, d'une part, la réalisation d'une approche efficace qui équilibre les intérêts de l'intensification et de la diversification, ce qui permet d'améliorer la qualité de la solution trouvée par l'algorithme, et d'autre part, de réduire le temps d'exécution de l'algorithme *Tabou 1*.
Cette parallélisation consiste à créer différents processus parallèles indépendants appelés *chemins de recherche parallèle*. Un chemin est une exécution de l'algorithme *Tabou 1* à partir d'une solution initiale s_o et d'une stratégie *st* (*st* = (*taille_liste_tabou*, *nb_iter*)).

La solution proposée (parallélisation en mode maître/esclave), consiste à utiliser une architecture parallèle de multiples processeurs indépendants (MIMD à mémoire distribuée). Un processeur possède le statut de maître et contrôle les autres processus (esclaves) en modifiant dynamiquement les paramètres qui gouvernent la recherche. Ce type de parallélisation a l'avantage de minimiser la communication entre les différents processus, et il est bien adapté aux machines MIMD à mémoire distribuée.

```
Algorithme : Tabou 2
Début
Etape 1 : /* Initialisation
algorithme_liste(s)   /*génération de la solution initiale
s* := s ;
f* := f(s)
taille_liste_tabou := normale_taille_tabou /*Initialisation de la longueur de la
                                            liste tabou
Etape 2 : /* processus itératif d'intensification et de diversification

Pour (i:=1 à nbre_iter_limite) /* nbre_iter_limite : nombre d'itérations pour le
processus d'intensification et de diversification*/

  faire
    2.1. /*processus d'intensification
         Tabou_intensif (taille_liste_tabou, nb_local1);
    2.2. /* progression lente (pas d'amélioration de la solution courante depuis un certain nombre
         d'itérations) */
    2.3. /* initialisation de la longueur de la liste tabou à une grande valeur
          Taille_liste_tabou := grande_taille_tabou

          nb_iter:=nb_local2 ;/* nb_local2 compteur pour la diversification
         Tabou_diversif (taille_liste_tabou, nb_iter); /* processus de
         diversification où nb_iter diminue à chaque itération de 1 et la longueur de la liste
         augmente d'une valeur égale à pas_longueur*/
    2.4. /* remettre la liste tabou à la longueur normale
          Taille_liste_tabou := normale_taille_tabou ;
  fait
Retourner s*;
FIN.
```

Figure 1.8. L'algorithme *Tabou 2*.

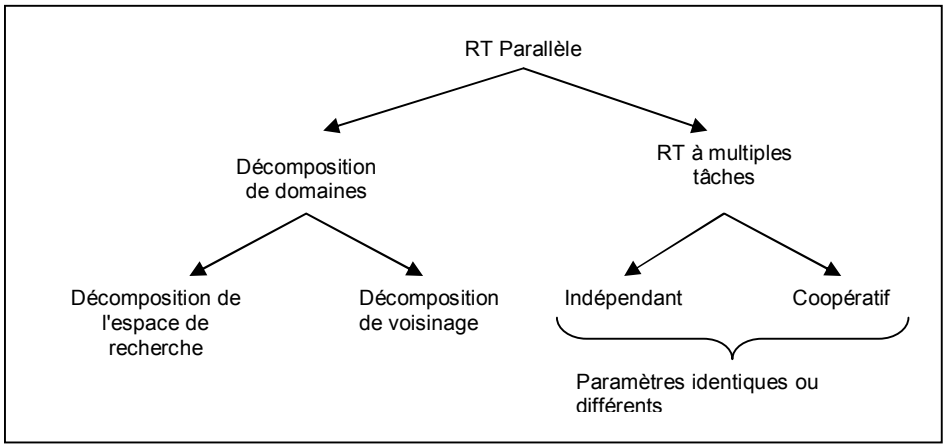


Figure 1.9. Classification des algorithmes de recherche tabou parallèles.

a) Processus maître

Le processus maître exécute un programme itératif. A chaque itération, *P* processus esclaves sont lancés. Au début de la recherche, le *maître* fournit à chaque *esclave* une solution initiale (obtenue par l'heuristique de liste proposée) et une stratégie *st* pour intensifier la recherche dans les différentes régions atteintes.

Quand la progression de la recherche, dans une région explorée par le processus esclave *i*, devient lente, le processus maître change la stratégie st_i pour diversifier la recherche. Ce changement s'opère sur les deux composantes de la stratégie st_i, à savoir la taille de la liste tabou ($taille\_liste\_tabou_i$) et le nombre maximum d'itérations entre deux améliorations de la meilleure solution trouvée (nb_iter_i).
Ce changement consiste à modifier ces deux paramètres en augmentant la taille de la liste tabou et en diminuant le nombre maximum d'itérations. Ces modifications continuent, c'est-à-dire la longueur de la liste tabou augmente ($taille\_liste\_tabou_i := taille\_liste\_tabou_i + pas\_longueur$) et le nombre d'itérations maximal diminue (nb_iter_i) jusqu'à ce qu'une solution meilleure s_i^*, que toutes celles rencontrées jusqu'à présent, soit trouvée.

A ce moment, la phase de diversification s'arrête, et la recherche s'intensifie dans la nouvelle région. La phase d'intensification est réalisée en affectant à la longueur de la liste tabou, une valeur "petite" ou "normale", et en affectant à nb_iter_i une valeur égale à *nb_local1*).

b) Processus esclave

Chaque processus esclave fait appel à l'algorithme *Tabou 1* (présenté dans la section 1.5.1.2) à partir de la solution initiale et la stratégie transmise par le processus maître. Donc l'algorithme *Tabou 1* est utilisé comme fonction de diversification pour certaines stratégies, et comme fonction d'intensification pour d'autres. La convergence de cette approche est assurée à partir de deux types de *longueur de la liste* : "petite" (st_1 = (*petite_longeur_tabou, nbre_iter*$_1$)) et "normale" (st_2 = (*normale_longueur_tabou, nbre_iter*$_2$)).

1.7. Partie expérimentale

Afin d'analyser les performances des algorithmes proposés, une série de tests ont été effectués sur des instances dont certaines sont empruntées à la littérature [Col 91a][Pic 92][Bou 94][Rah 98]. Les tests sont effectués sur un réseau de PC de type Pentium II cadencé à 700 MHz fonctionnant sous le système C/PVM. Nous avons réalisé 10 exécutions pour chaque instance et les résultats fournis dans les tableaux ci-dessous représentent les meilleures valeurs. Le critère d'arrêt de nos algorithmes tabou est donné par le nombre d'itérations maximum, qui est fixé expérimentalement à 30.

En raison du manque de résultats théoriques concernant notre problème, nous ne pouvons pas échapper à un réglage empirique des paramètres de nos algorithmes. Les valeurs retenues, lors des différentes expérimentations réalisées, sont celles pour lesquelles on a observé un compromis entre la qualité de la solution et le temps CPU.

1.7.1. Comparaison entre les heuristiques

Les instances utilisées lors de cette phase expérimentale sont empruntées à la littérature [Col 91a][Pic 92][Bou 94][Rah 98]. Les paramètres utilisés dans l'algorithme *Tabou 2* sont : *grande_longueur_tabou* = 7, *pas_longueur* = 5, *normal_longueur_tabou* = 5 (voir tableau 1.1).

instances			***nb_local1***	***nb_local2***	***nbre_iter _limite***
n°	G_T : | V_T |;| E_T|; d_i; Y_{ij}	G_P			
1	15;16; [1,2]; 1	H_3	2	2	3
2	15;14; 2; 5	T_{3x3}	2	5	4
3	16;15; [2,22]; [1,3]	K_4	2	7	5
4	16;15; [1,2]; 1	A_7	2	4	2
5	17;23; [1,4]; 8	G_{3x3}	2	4	4
6	31;30; 1;1	T_{4x3}	3	3	1
7	9;14; [1,4]; [1,2]	G_{3x3}	2	2	4
8	13;24; [1,10]; [1,4]	T_{4x3}	5	4	2
9	13;24; [1,10]; [1,4]	H_2	2	4	3
10	12;12; [1,2]; [2,3]	A_7	3	7	5
11	17;21; [1,4]; 8	T_{4x4}	5	8	5
12	9;8; [1,3]; [1,6]	K_6	3	8	3
13	11;10; [3,4]; [1,2]	K_6	3	10	1
14	10;9; [4,8]; [1;4]	K_6	2	5	2
15	9;14; [1,4]; [1,2]	K_4	2	6	5
16	12;12; [1,2]; [2,3]	K_4	2	6	2
17	8;7; 1; 7	K_4	2	5	4

Tableau 1.1. Les paramètres de l'algorithme *Tabou 2*.

Avec K : Complet ; H : Hypercube; T : Torre ; A : Anneau ; G : Grille, *nb_local1* : nombre d'itérations pour l'intensification ; *nb_local2* : nombre d'itérations pour la diversification ; *Nbre_iter _limite* : nombre d'itérations du processus d'intensification et de diversification ; Y_{ij} : le nombre d'unités de données échangées entre les tâches T_i et T_j et d_i : la durée d'exécution de la tâche T_i.

simulé [Rah 98]. Il est important de signaler que faute de benchmarks et de résultats de la littérature exploitables, nous avons constitué nos propres jeux de test [Rah 98]. Ces instances et les résultats obtenus dans [Rah 98] nous permettent d'effectuer une étude comparative des résultats obtenus avec les différentes métaheuristiques implémentées. Les résultats sont donnés dans le tableau 1.2.

Instances	*Tabou 2*		*Parallel_Tabou*		*Algorithme Génétique* [Rah 98]		*Recuit simulé* [Rah 98]	
	coût_sol	T_{cpu} *(s)*	*coût_sol*	T_{cpu} *(s)*	*coût_sol*	T_{cpu} *(s)*	*coût_sol*	T_{cpu} *(s)*
1	7	0,27	8	0,20	7	0,68	7	0,81
2	17	1,64	20	5,70	20	0,62	20	1,67
3	30	0,43	30	0,36	31	0,61	32	0,57
4	30	0,16	30	0,78	32	0,63	35	1,18
5	35	1,97	94	0,15	41	0,82	58	2,94
6	10	1,86	10	11,34	11	2,05	11	22,08
7	14	0,16	16	0,12	15	0,47	15	0,75
8	31	0,82	33	1,69	31	0,72	33	2,50
9	33	0,27	36	0,20	33	0,66	33	0,64
10	9	1,31	14	0,37	11	0,58	13	0,99
11	16	14,17	17	11,17	19	0,95	21	6,76

Tableau 1.2. Quelques résultats obtenus avec différentes métaheuristiques.

La comparaison entre les versions séquentielle et parallèle de tabou et les algorithmes génétiques et le recuit simulé, montre que les meilleurs résultats (en coût des solutions et temps d'exécution) sont ceux de l'algorithme *Tabou 2*. Pour la plupart des jeux d'essai :

- *Parallel_Tabou* procure de meilleurs coûts que ceux des algorithmes génétiques et du recuit simulé. Son temps d'exécution est meilleur que celui du recuit simulé et est voisin de celui des algorithmes génétiques.
- l'algorithme génétique donne de meilleurs résultats que le recuit simulé en ce qui concerne la qualité de la solution.

A partir de ces tests, on remarque l'influence de la topologie de l'architecture et des coûts de communication entre processeurs sur le temps d'achèvement du problème d'ordonnancement considéré.
D'autre part, la durée de communication entre les tâches a aussi un effet non négligeable sur la durée d'exécution de la métaheuristique. Par exemple, l'instance 11 exécutée sur un tore de 4x4 est le benchmark ayant donné les temps d'exécution (T_{cpu}, T_{cpu} *Parallel_Tabou*) les plus élevés par rapport aux autres benchmarks.
On peut conclure que les résultats obtenus dépendent de la nature du graphe de précédence (coûts de communication entre tâches, connexions dans le graphe). En effet, lorsque les coûts de communication sont faibles, nous constatons que les tâches sont éparpillées sur les différents processeurs de la topologie choisie. Dans le cas contraire, les tâches sont exécutées sur un ensemble restreint de processeurs.

Les résultats du tableau 1.2 permettent aussi de constater que nos stratégies de diversification et d'intensification sont convenables et expliquent largement les bonnes performances de nos algorithmes. Bien que ces observations ne devraient pas être considérées

comme preuve du bon comportement de nos algorithmes, elles fournissent un certain appui statistique à notre affirmation que les stratégies utilisées dans notre algorithme sont en effet convenables.

1.7.2. Comparaison avec des problèmes ayant des solutions optimales

Nous avons exécuté nos algorithmes sur quelques instances de petite taille (le nombre de tâches s'étend de 9 à 12) dont les solutions optimales ont été trouvées dans [Col 91a][Pic 92][Bou 94]. Nos résultats sont énumérés dans le tableau 1.3 ci-dessous. On peut remarquer que :

- la *Tabou 2* trouve l'optimum pour tous ces exemples
- le *Parallel_Tabou* trouve l'optimum pour certains de ces instances. Pour d'autres instances, on observe de petits écarts par rapport à l'optimum.

Instances	***Tabou 2***		***Parallel Tabou***		***Meilleure solution connue***
	coût_sol	T_{cpu} ***(s)***	***coût _sol***	T_{cpu} ***(s)***	
12	6	0,49	8	0,53	6
13	14	0,49	16	0,60	14
14	15	0,16	17	0,70	15
15	14	0,21	15	0,22	14
16	9	0,16	12	0,69	9
17	8	0,05	8	0,18	8

Tableau 1.3. Résultats obtenus pour des instances dont l'optimum est connu.

1.7.3. Tests sur des graphes aléatoires

Quelques tests ont été réalisés sur différentes instances dont les graphes de tâches ont été construits aléatoirement, avec des nombres de sommets multiples de dix. Ces tests ont pour effet d'évaluer l'effet de la topologie sur le coût de la solution et sur le temps de réponse de l'algorithme de recherche tabou *Tabou 2* implémenté (voir tableau 1.4).

Les résultats du tableau 1.4 montrent :

- Le temps d'exécution de la recherche tabou (l'algorithme *Tabou 2*) augmente en fonction de la taille du graphe de précédence (nbre de tâches), et du nombre de processeurs de l'architecture.
- L'augmentation du nombre de processeurs de l'architecture cible permet d'améliorer le coût de la solution trouvée, et conduit à l'augmentation du temps de réponse de l'algorithme tabou.
- La topologie choisie influe considérablement sur le coût de la solution trouvée et sur le temps de réponse de l'algorithme de recherche *Tabou 2* .

Instances		*coût _sol Tabou 2*	T_{cpu} *Tabou 2 (s)*
G_T avec | V_T |; | E_T| d_i; Y_{ij}	G_P		
10; 15; 1;1	H_3	7	0,49
20; 43; 1;1	A_5	10	1,59
30; 48; 1;1	T_{4x3}	13	3,68
10; 15; 1;1	K_6	7	0 ,38
20; 43; 1;1	K_6	8	0,87
30; 48; 1;1	K_2	16	0,71
30; 48; 1;1	K_4	11	1,37
30; 48; 1;1	K_6	10	1,70
30; 48; 1;1	K_8	10	2,04
30; 48; 1;1	H_3	11	17,41
30; 48; 1;1	A_5	11	1,42
30; 48; 1;1	T_{4x3}	13	3,68

Tableau 1.4. Qualité et temps d'exécution de l'algorithme *Tabou 2* sur des instances aléatoires.

1.8. Etude de complexité

Dans cette section, il sera surtout question de complexité. Notons $SC(G_T,G_p,t)$ le problème de décider s'il est possible d'ordonnancer un ensemble de tâches G_T muni d'une relation de précédence sur un réseau de processeurs G_P en un temps donné t. Le problème d'optimisation associé sera noté $SC(G_T,G_p)$ (ou simplement SC si l'instance (G_T,G_P) est sous-entendue). Dans un premier temps, afin de rendre compte de l'intraitabilité de ce problème, nous établirons la NP-complétude de $SC(G_T,G_p,t)$ restreint à des cas particuliers portant soit sur l'architecture, soit sur la relation de précédence. Ensuite, nous étudierons le cas où le graphe des contraintes de précédence a une profondeur constante, d'où l'on conclura que SC n'admet pas de rapport de schéma d'approximation polynomial, à moins que P=NP. Il convient de souligner ici que les versions restreintes de SC établies comme étant NP-complètes dans cette étude ont toutes des relations immédiates avec les types d'instances traitées expérimentalement dans ce chapitre.

1.8.1. Résultats de NP-complétude

SC est connu comme étant NP-dûr. Il est en fait NP-dûr pour toute architecture fixée à l'avance, pourvu que le nombre de processeurs soit suffisamment grand. Pour voir ceci, il suffit de considérer la réduction suivante du problème de la 3-partition : Soit $a_1,a_2,...,a_n$ une instance du problème de la 3-partition. À tout a_i , associons un chemin C_i de longueur a_i représentant une chaîne de a_i tâches, chacune ayant un temps d'exécution unitaire. La réunion (disjointe) de ces n chemins indépendants constitue un graphe sans circuit G_T. Considérons G_P comme étant un graphe quelconque d'ordre $n/3$. Alors il est facile de voir que $a_1, a_2,..., a_n$ admet une 3-partition si et seulement si l'instance de SC ainsi construite peut être

ordonnancée en temps $t = 3 * (\sum a_i / n)$. A partir de cette réduction, on peut aussi montrer que le sous-problème de *SC* dans lequel G_T est une arborescence est aussi NP-complet.

1.8.2. Difficulté de *SC* par rapport à l'approximation

Dans cette partie, nous nous proposons d'établir la NP-complétude de $SC(G_T,G_p,t)$ lorsque t est constant. Soit G un graphe d'ordre n, où n est multiple de 3. Notons H le graphe orienté sans circuit représenté dans la figure 1.10. Supposons que toutes les tâches (sommets) de H et toutes les communications sont unitaires. Alors H a la propriété suivante : H peut être ordonnancé en temps $t \leq 6$ sur une architecture donnée si et seulement si il existe trois processeurs induisant un chaîne de longueur 2 dans l'architecture correspondante.
La condition nécessaire est évidente. Montrons que la condition est suffisante. Soient P_1, P_2, P_3 trois processeurs tels que $P_1P_2P_3$ est une chaîne de longueur 2 dans G_p. H est formé de 3 chemins : un chemin supérieur, un chemin médian et un chemin inférieur, de longueurs respectives 5, 5 et 6, comme indiqué dans la figure 1.10. Considérons l'ordonnancement suivant : le processeur P_1 exécute le chemin supérieur de H en temps 6, en tenant compte d'un temps de communication avec P_2 qui prend fin à $t=2$. De même, P_2 exécute le chemin médian en temps 6, en tenant compte d'un temps de communication avec P_3 qui prend fin à $t=3$. Toutes ces communications se font en temps unité parce que les distances entre P_1 et P_2 et entre P_2 et P_3 sont unitaires dans G_P. Ainsi, P_3 va exécuter le chemin inférieur de H de longueur 6 en temps 6, puisqu'il n'a pas de communication à effectuer et que l'exécution de la cinquième tâche sur ce chemin peut commencer au temps $t=4$.

Construisons maintenant le graphe G_T réunion disjointe de $n/3$ copies indépendantes de H. Alors, à partir de la remarque précédente, il est facile de voir que
$SC(G_T,G)$ peut être ordonnancé en temps $t=6$ si et seulement si G_T admet une partition en chaînes de longueur 2, ce qui est NP-complet.

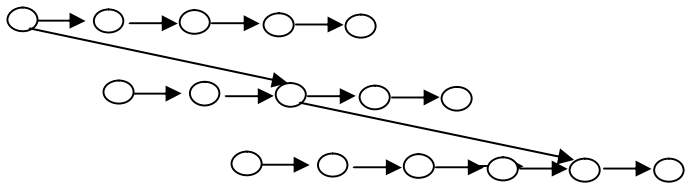


Figure 1.10. Le graphe orienté sans circuit : la composante *H*.

D'où le théorème suivant et son corollaire :

Théorème 1.1 : $SC(G_T,G_P,6)$ est NP-complet.

Corollaire 1.1 : SC ne peut admettre de rapport d'approximation inférieur à 7/6 à moins que P=NP.

Ainsi, SC est difficile pour l'approximation, dans le sens où il n'admet pas de schéma d'approximation polynomial à moins que P=NP. Dans le reste de cette partie, nous allons montrer que ce résultat négatif subsiste lorsque $G_P=K_{|T|}$, le graphe complet d'ordre $|T|$, où T est l'ensemble des tâches. Cette restriction est importante dans la mesure où cela suppose que le nombre de processeurs est illimité et que l'on a tous les liens physiques possibles entre

les processeurs. Il est intéressant de savoir que même dans ce cas, il ne peut y avoir de schéma d'approximation, et c'est ce que nous allons établir.

La réduction se fait à partir de *3-SAT'*, un sous-problème de *3-SAT* dans lequel chaque variable apparaît au plus trois fois. Nous admettons le lemme suivant dont la démonstration se trouve dans [Col 91a] [Col 91b].

Lemme1.1 : *3-SAT'* est NP-complet.

Nous pouvons maintenant formuler le résultat principal de cette section.

Théorème 1.2 : $SC(G_T, K_{|T|},21)$ est NP-complet.

Preuve : La réduction est de *3-SAT'*. Comme le nombre de processeurs est égal au nombre de tâches dans notre instance de $SC(G_T, K_{|T|},21)$ que nous allons construire, nous pouvons supposer sans perte de généralité que nous avons un nombre infini de processeurs. Soit f une formule logique instance de 3-SAT'. Notons ses littéraux $x_1, x_2,.., x_n$. Représentons chaque littéral x_i par une composante X_i qui est un graphe orienté construit de la façon suivante :

1. $V(X_i)=\{a\}\cup B\cup X'_i$; B est de cardinalité 3 et ses sommets sont étiquetés b_1,b_2,b_3. X'_i a trois sommets étiquetés x_i, $\overline{x_i}$ et q_i.
2. $(b_1,x_i)\in E(X_i)$; $(b_2,\overline{x_i})\in E(X_i)$; et $(b_3,q_i)\in E(X_i)$.
3. Pour tout $1\le j\le 3$, $(a,b_j)\in E(X_i)$,

Ce qui achève la description de X_i. Comme graphe orienté, X_i peut être considéré comme un ensemble de tâches devant respecter les contraintes de précédence sous-jacentes au graphe.

Les tâches de X'_i sont toutes de durée unitaires. La tâche a est de durée 2 et toutes les tâches de B sont de durée 3. La communication entre deux tâches quelconques de cette composante prend 3 unités de temps.
Remarquons que toutes les tâches de X_i peuvent être exécutées en temps $t=12$ (ce qui est optimal), mais alors une seule tâche de X'_i (c'est à dire x_i, $\overline{x_i}$ ou la tâche 'supplémentaire' q_i) sera exécutée avant $t=12$ et son temps d'exécution sera nécessairement $t=9$. Si la tâche étiquetée x_i est celle qui est exécutée au temps t=9 dans un ordonnancement optimal de X_i, on dira que x_i est 'en avance sur $\overline{x_i}$ ' et qu'il a "3 unités de temps d'avance sur $\overline{x_i}$ dans l'ordonnancement". Remarquons qu'en raison de la symétrie de la composante X_i, on peut aussi bien 'faire avancer' x_i par rapport à $\overline{x_i}$ qu'inversement (c'est à dire qu'il existe des ordonnancements optimaux dans lesquels l'une ou l'autre des variables est en avance). La figure 1.11 montre X_i avec un ordonnancement optimal. Dans cet ordonnancement, x_i est en avance. L'affectation des tâches aux processeurs n'est pas montrée, mais on peut la deviner à partir des temps d'exécution.

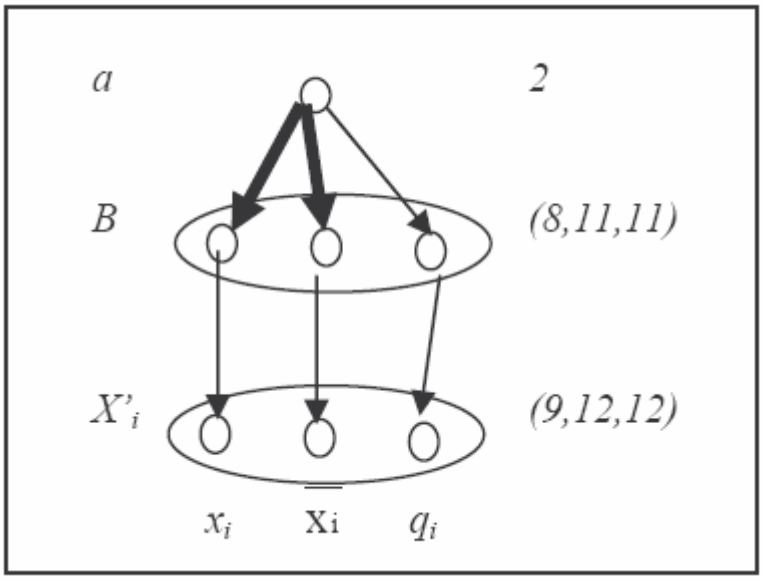


Figure 1.11. La composante X_i et un ordonnancement. Les arcs en gras indiquent une communication.

Représentons maintenant chaque clause C par la composante H_C de la figure 1.12. Plus exactement, H_C est le graphe orienté défini comme suit :

1. $V(H_C)=\{c\}\cup C_1\cup C\cup\{c'\}\cup C'\cup\{c''\}\cup C''\cup D_1\cup D$; chacun des C, C_1,C',C'',D_1,D a trois sommets étiquetés c_1,c_2,c_3 (respectivement c^1_1,c^1_2,c^1_3 $c'_1,c'_2,c'_3;c''_1,c''_2,c''_3;d^1_1,d^1_2,d^1_3$).
2. Pour tout $1\le j\le 3$, $(c,c^1_j)\in E(H_C)$, $(c',c'_j)\in E(H_C)$, $(c'',c''_j)\in E(H_C)$
3. Pour tout $1\le j\le 3$, $(c'_j,d^1_j)\in E(H_C)$; $(c''_j,d^1_j)\in E(H_C)$; $(c^1_j,c_j)\in E(H_C)$.
4. Pour tout $1\le j\le 3$, $(d^1_j,d_j)\in E(H_C)$ et $(c_j,d_j)\in E(H_C)$

Toutes les tâches de H_C -$\{c\}$ sont de durée 3, la tâche c est de durée 6 et le coût de communication entre une paire quelconque de tâches sur des processeurs adjacents est de 3 unités de temps.

De plus, pour chaque clause c de f, si la variable x_i est la première (respectivement la seconde, troisième) variable apparaissant dans c, on relie le sommet étiqueté x_i dans le composante X_i au premier (respectivement second, troisième) sommet du sous-ensemble C de H_C par un arc, et on définit le temps de communication entre la paire de sommets ainsi reliée comme unitaire.

Ceci achève la description de notre instance I de $SC(G_T,K_{|T|},21)$.

Avant d'aller plus loin dans la démonstration, remarquons les faits suivants concernant la composante H_C :

(i) Le temps d'exécution optimal pour H_C est de 21.

(ii) Pour tout ordonnancement optimal de H_C, toutes les trois tâches c_1, c_2, c_3 de C doivent être exécutées au temps $t=18$

(iii) Pour tout ordonnancement optimal de H_C, au moins une tâche de C doit être exécutée au temps t=15. On dira alors qu'une telle tâche est en avance de 3 unités de temps.

(iv) N'importe quelle tâche parmi c_1, c_2, c_3 peut être choisie comme étant en avance (c'est à dire que pour chaque variable choisie parmi c_1, c_2, c_3, il existe un ordonnancement optimal de notre composante dans lequel la variable choisie est en avance).

A des fins d'illustration, un ordonnancement optimal de H_C est exhibé par la figure ci-dessous, dans laquelle nous avons choisi c_2 comme tâche "en avance". Il est évident que, moyennant une permutation appropriée des tâches, on aurait pu "faire avancer" n'importe quelle tâche parmi c_1, c_2, c_3.

Nous affirmons maintenant que l'instance I de $SC(G_T,K_{|T|},21)$ ainsi construite admet un ordonnancement de temps ≤ 21 si et seulement si la formule logique f est satisfaisable.

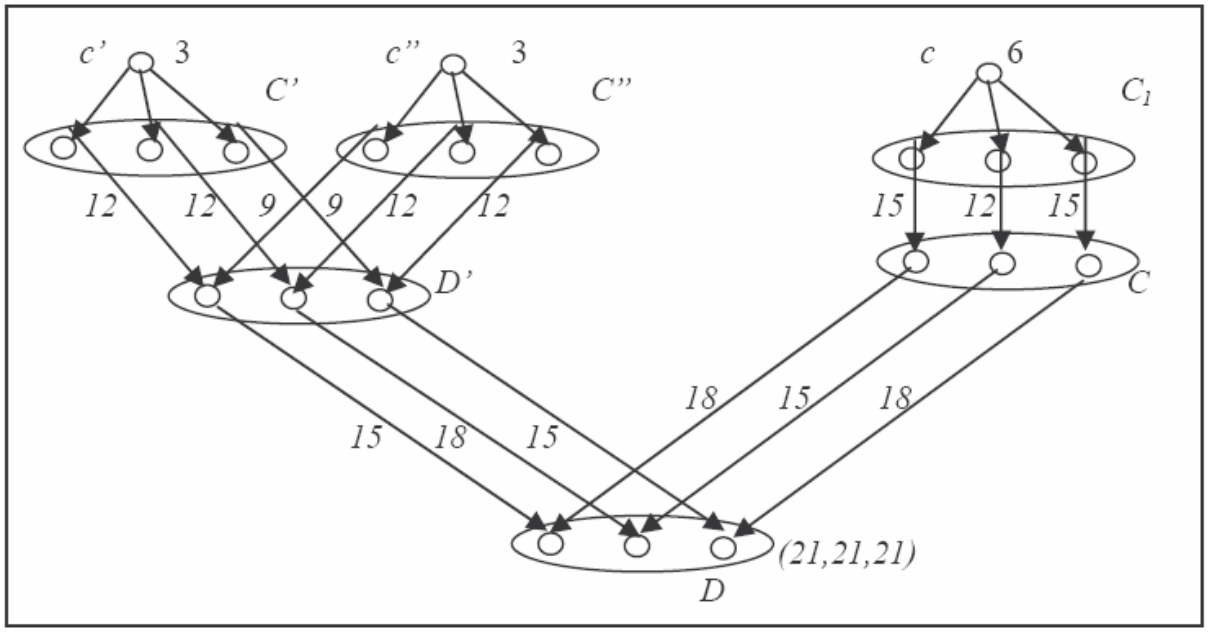


Figure 1.12. La composante H_C et un ordonnancement optimal.

Preuve de notre assertion : Supposons d'abord est que f est satisfaisable. Soient donc une affectation satisfaisant f, et φ une fonction qui associe à toute clause c, une variable $\varphi(c)$ satisfaisant c pour l'affectation considérée. Pour abréger nos notations, posons $\varphi(c)=x_c$ pour toute clause c et considérons la tâche c_t de $C \subset H_c$ à laquelle x_c est reliée par un arc dans la construction de I. Considérons maintenant l'ordonnancement suivant de I :

1- Pour tout littéral x_c, ordonnancer de manière optimale la composante X_c associée au littéral x_c de telle manière que la variable x_c soit 'en avance de 3 unités de temps', au sens qui a été défini précédemment.
2- Pour toute clause C, ordonnancer de manière optimale la composante H_C de telle manière que la variable c_t soit "en avance" dans l'ordonnancement de C, au sens qui a été déjà défini.

Remarquons que cet ordonnancement est réalisable parce qu'il faut au plus trois unités de temps à la variable x_c pour communiquer avec les autres tâches (puisque chaque variable apparaît au plus trois fois), ce qui établit la condition suffisante.

Montrons maintenant la condition nécessaire. Supposons que I peut être ordonnancé en temps t=21. Pour tout x_i, affectons la variable x_i à "vrai" si la tâche x_i de X_i est exécutée au temps t=9 (c'est à dire si elle est 'en avance dans l'ordonnancement induit de X_i'). Affectons toutes les autres variables à "faux".
Or, un ordonnancement de temps 21 induit un ordonnancement optimal pour chaque composante H_C. D'après les propriétés de H_C, chaque composante H_C doit avoir une tâche c_t de C telle que c_t est en avance. Il est évident que alors que la variable x_i reliée à c_t dans H_C est aussi en avance dans X_i, ce qui veut que x_i est vraie dans notre affectation et donc toutes les clauses sont satisfaites.

A partir de ce résultat, on conclut que *SC* ne peut admettre de rapport d'approximation polynomial inférieur à 22/21, à moins que P=NP; d'où le corollaire suivant :

Corollaire 1.2 : $SC(G_T,K_{|T|})$ ne peut admettre de schéma d'approximation polynomial, à moins que P=NP.

Vu les résultats précédents, nous pensons au fait que *SC* ne peut admettre de rapport d'approximation constant en temps polynomial, mais nous n'avons pas pu le démontrer.

1.9. Conclusion

Ce travail traite de l'ordonnancement des tâches sur une architecture MIMD à mémoire distribuée. Pour résoudre ce problème, nous avons proposé deux algorithmes séquentiels et une version parallèle de la recherche tabou.
Le premier algorithme (*Tabou 1*) représente la méthode *tabou* dans sa forme classique. Le second (*Tabou 2*) est une amélioration du premier, dans la mesure où il intègre des fonctions d'intensification et de diversification. Le troisième est un algorithme parallèle (*Parallel_Tabou*) et est d'une portée générale. En effet, d'une part, il est indépendant de la topologie de l'architecture sur laquelle il s'exécute, et, d'autre part, aucune hypothèse n'est admise sur la structure de l'application.
Parallel_Tabou est un algorithme synchrone. Il est obtenu en particularisant le rôle de l'un des processus (le maître), qui s'occupe de la distribution des solutions initiales et des stratégies entre les différents processus esclaves. Chacun de ces derniers applique l'algorithme *Tabou 2* sur la solution initiale qui lui est attribuée par le maître, selon la stratégie choisie par ce dernier. Les solutions (résultats) trouvées par les esclaves ensuite sont envoyées au processus maître qui sélectionnera la meilleure d'entre elles.

D'après les tests effectués sur les différents algorithmes développés, nous avons remarqué que les résultats obtenus dépendent de la nature du graphe de précédence (durées d'exécution des tâches, coûts de communication entre paires de tâches, densité du graphe). En effet lorsque les coûts de communication étaient faibles, les tâches se répartissaient assez bien sur les différents processeurs. Dans le cas contraire, elles avaient tendance à s'exécuter sur un nombre restreint de processeurs.
Le temps CPU des algorithmes développés varie en fonction des paramètres suivants :
- Nombre de tâches du graphe de précédence.
- Nombre de processeurs de l'architecture parallèle.
- Coûts de communication entre tâches et entre processeurs.

Dans le but d'évaluer les performances des algorithmes tabou développés, nous avons établi une étude comparative, portant sur un échantillon diversifié, entre nos résultats et ceux des deux autres heuristiques (recuit simulé, algorithmes génétiques) [Rah 98]. L'algorithme *Tabou 2* s'est avéré très efficace en qualité de solution par rapport aux *Tabou 1* et *Parallel_Tabou*. Compte tenu de la diversité des instances sur lesquelles nos heuristiques ont soutenu favorablement la comparaison, nos résultats semblent indiquer assez clairement que nos stratégies d'intensification et de diversification étaient efficaces.
Les différents algorithmes développés se sont avérés très efficaces en qualité de solution ainsi qu'en temps d'exécution par rapport aux algorithmes génétiques et le recuit simulé. D'un autre côté, des tests ont été réalisés pour le calcul de l'accélération de l'algorithme *Parallel_Tabou*. Ces expérimentations révèlent une accélération presque linéaire. Cette performance peut en effet se justifier par le fait que les coûts de communication inter-processeurs sont petits par rapport aux durées d'exécution des tâches.

Par ailleurs, nous avons montré que le problème est NP-complet même pour des cas particuliers qui nous intéresse. Nous avons également montré que le problème est dur au regard de l'approximation dans le sens qu'il n'admet pas de schéma d'approximation polynomial à moins que P=NP. Ainsi, nous avons adopté une approche plus pratique à ce problème.

Chapitre 2

Métaheuristique de colonie de fourmis parallèle pour le problème de couverture d'ensembles

Le problème de couverture d'ensembles (Set Covering Problem - SCP) est l'un des problèmes NP-difficiles les plus connus. Les algorithmes exacts, tels que la méthode par Séparation & Evaluation, ne trouvent des solutions optimales, dans des délais raisonnables, que pour des petites instances. Pour pallier à cette difficulté, les métaheuristiques itératives telles que les colonies de fourmis représentent une alternative qu'il est intéressant d'explorer. En effet, jusqu'à présent, les colonies de fourmis n'ont été que très peu appliquées à ce problème, alors que cette métaheuristique est réputée pour donner d'excellents résultats pour de nombreux problèmes de l'optimisation combinatoire.
Ce chapitre présente des algorithmes de résolution basés sur les colonies de fourmis. Deux schémas d'hybridation/coopération entre les colonies de fourmis et la recherche locale sont proposés. Par la suite, deux modèles génériques de parallélisation du meilleur algorithme de coopération proposé sont présentés.

Les travaux décrits dans ce chapitre ont fait l'objet de deux communications [Had 00] et [Rah 02].
Une description de notre travail est donnée par Dorigo et Stützle dans leur livre [Dor 04], de la page 182 à 184. Les auteurs de ce livre confirment les bons résultats obtenus par nos algorithmes ainsi que ceux de [Leg 99].

2.1. Introduction

Le problème de couverture d'ensembles, que nous notons par la suite SCP (de l'anglais *Set Covering Problem*), est l'un des problèmes les plus étudiés en optimisation combinatoire, [Sak 84][Teg 96][Cap 99]. Cela est justifié par le fait que plusieurs problèmes du monde réel peuvent être modélisés comme un problème de couverture d'ensembles [Bal 96][Bea 96]. Parmi ces problèmes, nous citons :

- L'ordonnancement des équipages de compagnies aériennes [Wed 95][And 97][Cap 99][Mar 00];
- Le déploiement de véhicules de service [Das 81][Vas 84] ;
- La simplification des expressions booléennes [Vas 87] ;
- Des problèmes d'ordonnancement [Bro 87];
- L'exploitation de la fabrication d'aciers [Vas 87] ;
- La planification du temps de travail de personnel [Jac 93].

On trouve dans la littérature plusieurs méthodes de résolution pour ce problème [Coo 98][Gar 79]. Il y a d'une part les algorithmes exacts, basés sur des procédures de recherche dans les structures d'arbres [Bal 80a][Bal 80b] utilisant des techniques de *Séparation & d'Evaluation* [Bal 96] ou des techniques dérivées de la théorie des graphes [Hey 83]. Ces algorithmes exacts ne peuvent trouver une solution optimale, dans des délais raisonnables, que pour des instances de petite taille [Bea 87][Fis 90][Bea 92]. On trouve d'autre part des algorithmes approchés pouvant s'appliquer à des instances de grande taille [Vas 02]. Parmi ces derniers on peut mentionner des algorithmes gloutons [Chv 79], et des métaheuristiques comme le recuit simulé, ou encore les algorithmes génétiques [Cap 00].

Dans ce chapitre, nous nous intéressons à la résolution du SCP à l'aide d'un algorithme hybride basé sur la métaheuristique des colonies de fourmis [Dor 92][Dor 96][Dor 97][Dor 04] et la recherche locale [Jac 95][Vae 98]. La recherche locale est utilisée comme un outil d'intensification, c'est-à-dire comme un moyen pour améliorer la qualité de la solution trouvée par les fourmis. Deux algorithmes hybrides faisant appel aux colonies de fourmis et à la recherche locale sont étudiés. Le premier schéma d'hybridation est en mode *relais de haut niveau (HRH- High-Level Relay Hybrid)* alors que le second est de *haut niveau en mode coévolution (HCH – High Co-evolutionary Hybrid)* [Tal 02]. Dans les hybridations de haut niveau en mode relais, les heuristiques hybridées conservent leur intégrité, c'est-à-dire que chaque métaheuristique intervenant dans ce processus d'hybridation reste inchangée et autonome. Les méthodes *HRH*-hybridées sont exécutées en séquence (par exemple, un algorithme génétique est exécuté pour localiser des régions intéressantes, puis une recherche tabou est lancée pour exploiter ces régions). Au contraire, dans les hybridations de haut niveau coévolutionnaires, les métaheuristiques hybridées, ont toujours une structure interne qui n'est pas modifiée, mais cette fois-ci elles sont exécutées simultanément et coopèrent pour résoudre le problème. Un exemple d'hybridation *HCH* est le modèle des algorithmes génétiques en îles. Dans ce modèle, la population est partagée en sous-populations où les sous-populations sont, par exemple, disposées dans un hypercube de dimension 4 [Tan 87]. Un algorithme génétique fait évoluer chaque sous-population et les individus peuvent migrer d'une sous-population à l'autre.

Pour la validation de notre approche et le réglage des paramètres, plusieurs tests ont été réalisés sur des benchmarks de la littérature, et plus précisément sur les instances de Beasley de la "OR Library" [Bea 90]. L'implémentation de cette approche a permis, dans certains cas, de trouver une solution optimale et ce dans des délais raisonnables. Afin d'améliorer la qualité des solutions trouvées et afin de réduire le temps de recherche, nous avons implémenté

deux modèles parallèles de l'algorithme hybride le plus performant que nous avons obtenu : la *parallélisation multidépart indépendante* et *la parallélisation des fourmis*. Dans le premier modèle, il s'agit de lancer simultanément plusieurs recherches basées sur un algorithme. Ce dernier est dupliqué sur les différents processeurs et est exécuté par chaque processeur d'une manière autonome. Quant au deuxième, une seule copie de l'algorithme est utilisée. Cet algorithme est parallélisé en considérant chaque fourmi comme un processus. La phase de construction des solutions par les fourmis est parallélisée en plaçant chaque fourmi sur un processeur indépendant. Les tests expérimentaux confirment l'amélioration des résultats de ces deux modèles dans plus de 65% des cas par rapport aux algorithmes séquentiels développés.

La suite de ce chapitre est organisée de la manière suivante. Dans la section 2.2, une présentation formelle du SCP est donnée. La section 2.3 présente la métaheuristique des colonies de fourmis. La section 2.4 donne les éléments de base d'un algorithme à base de colonies de fourmis. La section 2.5 expose le protocole expérimental utilisé pour la validation des différents algorithmes implémentés. La section 2.6 propose deux algorithmes hybrides de colonies de fourmis avec la recherche locale.
La section 2.7 présente les deux schémas de parallélisation de l'algorithme hybride en mode *HCH* (le plus performant), une analyse des résultats expérimentaux obtenus et compare ceux-ci aux meilleurs résultats de la littérature.

2.2. Le problème de couverture d'ensembles

2.2.1. Formulation du problème de couverture d'ensembles

Le problème de couverture d'ensembles (SCP) est défini de la manière suivante. Soit un ensemble de base $\mathcal{U} = \{e_1,...,e_m\}$ et un ensemble de sous-ensembles de $\mathcal{U}$, $\mathcal{S} = \{S_1,...,S_n\}$ où $S_j \subseteq \mathcal{U}$ pour $j \in \mathcal{J} = \{1,...,n\}$. On dit qu'un sous-ensemble $\mathcal{J}^* \subseteq \mathcal{J}$ représente une couverture de $\mathcal{U}$ si et seulement si $(\cup_{j \in J^*} S_j) = \mathcal{U}$. Un coût positif est associé à chaque sous-ensemble S_j, $j \in \mathcal{J}$. Le coût de la couverture $\mathcal{J}^*$ est égal à la somme des coûts des sous-ensembles S_j qui font partie de cette couverture, c'est-à-dire les sous-ensembles S_j tels que $j \in \mathcal{J}^*$. L'objectif du problème est de déterminer une couverture de $\mathcal{U}$ qui soit de coût minimum.

Dans la suite nous allons utiliser une formulation de ce problème sous forme matricielle (voir l'exemple de la figure 2.1).
Une instance du problème est représentée par une matrice booléenne $A = (a_{ij})$ de *m lignes,* qui correspondent aux éléments *de* $\mathcal{U}$, et de *n colonnes,* qui correspondent aux sous ensembles de $\mathcal{S}$, et par un vecteur d'entiers positifs *coût* = ($coût_1$, $coût_2$,.., $coût_n$) de dimension *n* correspondant au coût de chaque colonne. Le problème consiste à trouver un sous-ensemble de colonnes avec un coût minimal tel que chacune des lignes de la matrice soit recouverte par au moins une de ces colonnes. La ligne *i* est dite recouverte par la colonne *j* si a_{ij} est égal à 1.

Exemple :

Soient la matrice booléenne *A* de 3 lignes et 4 colonnes et un vecteur *coût* représentant une instance du problème du SCP.

$$A = \begin{pmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 \end{pmatrix} \qquad coût = \begin{pmatrix} 1 & 5 & 1 & 3 \end{pmatrix}$$

Figure 2.1. Exemple d'une instance du SCP.

La matrice *A* de l'exemple de la figure 2.1 permet de constater que la ligne 1 est recouverte par les colonnes 1 et 3. La ligne 2 est recouverte par les colonnes 2 et 3, quant à la ligne 3, elle est recouverte par les colonnes 1, 3 et 4.
En conformité avec la définition du SCP, on peut écrire :

$m = 3$ (nombres de lignes)
$n = 4$ (nombre de colonnes)
$\mathcal{U} = \{e_1, e_2, e_3\}$ avec e_1, e_2 et e_3 représentant respectivement les lignes 1, 2 et 3.
$\mathcal{J} = \{1, 2, 3, 4\}$ ensemble des colonnes de la matrice *A*
$S = \{S_1, S_2, S_3, S_4\}$ avec $S_1 = \{e_1, e_3\}$, $S_2 = \{e_2\}$, $S_3 = \{e_1, e_2, e_3\}$, $S_4 = \{e_3\}$

A partir de ces données, et parmi les couvertures réalisables possibles on peut, par exemple, en citer 3 :

- $J^* = \{3\}$ est une couverture réalisable de coût égal à 1 et avec $(\cup_{j \in J^*} S_j) = S_3 = \mathcal{U}$. *C'est la solution optimale.*
- $J^* = \{1, 3\}$ est une couverture réalisable de coût égal à 2 et avec $(\cup_{j \in J^*} S_j) = S_1 \cup S_3 = \{e_1, e_3\} \cup \{e_1, e_2, e_3\} = \mathcal{U}$
- $J^* = \{3, 4\}$ est une couverture réalisable de coût égal à 4 et avec $(\cup_{j \in J^*} S_j) = S_3 \cup S_4 = \{e_1, e_2, e_3\} \cup \{e_3\} = \mathcal{U}$

2.2.2. Méthodes de résolution du SCP

Un large éventail de méthodes ont été proposées pour résoudre le SCP. Les algorithmes exacts proposés trouvent la solution optimale pour des instances de tailles inférieures à 400 lignes (contraintes) ,et 4000 colonnes (variables) [Bal 80a][Bal 80b][Hey 83][Bea 87][Fis 90][Bea 92][Bal 96]. Le temps CPU moyen, pour toutes ces instances, sur une *DECstation 5000/240*, est de 2000 secondes [Cap 00]. Pour la résolution des instances de tailles supérieures à 400 lignes et 4000 colonnes, dans des temps de recherche acceptables, des heuristiques et des métaheuristiques ont été utilisées.

Chvàtal [Chv 79], proposa l'une des premières heuristiques utilisée pour la résolution du SCP. Cet algorithme de type glouton, construit séquentiellement une couverture en ajoutant, à chaque étape à la solution partielle, une colonne *j* ayant la plus grande valeur du rapport ($card_j/coût_j$). Dans ce rapport, $card_j$ représente le nombre de lignes recouvertes par la colonne *j* et non encore recouvertes dans la solution partielle, alors que $coût_j$ représente le coût de la colonne *j*. Cet algorithme a été utilisé dans [Jac 95] pour construire une solution réalisable, à partir de laquelle un algorithme de recuit simulé est appliqué pour améliorer la qualité de la solution. Dans [Mar 98] et [Mar 00], un algorithme utilisant l'heuristique gloutonne de Chvàtal et la recherche locale a été proposé, apportant des résultats compétitifs.

Plusieurs heuristiques performantes sont répertoriées dans [Cap 00]. Celles-ci s'appuient sur des techniques de relaxation linéaire [Wed 95], de relaxation lagrangienne [Bea 90][Fis 90][Bal 96][Cap 00], de relaxation agrégée (*surrogate relaxation*) [Lor 94] et de décomposition lagrangienne [Cap 99][Cap 00]. L'heuristique basée sur la relaxation lagrangienne [Cap 99][Cap 00] s'est avérée efficace pour des instances dont la taille est supérieure à 400 lignes et 4000 colonnes.
Le temps d'exécution moyen, sur une *DECstation 5000/240*, est de 50 secondes pour les instances de taille inférieure à 400 lignes et 4000 colonnes et de 280 secondes pour les instances de plus grande taille [Cap 00]. Cet algorithme a été testé avec succès sur des instances émanant d'une application d'ordonnancement d'équipages dans les chemins de fer italiens.

En ce qui concerne les métaheuristiques, des méthodes basées sur des algorithmes génétiques ont été proposées dans [Bea 96][Ere 99][Ere 00] et [Vas 05] ainsi que par Chu dans [Chu 98]. Dans [Vas 05], les auteurs proposèrent un algorithme génétique parallèle. Les colonies de fourmis ont été peu utilisées pour la résolution du SCP. En effet, hormis nos travaux [Had 00][Rah 02] et ceux de [Leg 99], le seul article (postérieur à notre étude) étudiant ce problème en utilisant les colonies de fourmis est celui de Lessing et al. [Les 04]. Leurs résultats confirment nos résultats c'est-à-dire les bonnes performances d'un algorithme hybride combinant les colonies de fourmis et la recherche locale [Dor 04]. Pour cette raison et malgré les bonnes performances de l'approche basée sur la décomposition lagrangienne de Caprara [Cap 99][Cap 00], qui reste par ailleurs difficile à mettre en œuvre de par sa spécificité, nous avons abordé cette problématique en utilisant les colonies de fourmis. Nous donnons dans la section suivante les notions de base de cette métaheuristique.

2.3. Les colonies de fourmis

La métaheuristique des colonies de fourmis fut proposée par Dorigo en 1991 dans le cadre de sa thèse de doctorat [Dor 92][Dor 96]. Cette métaheuristique est inspirée du comportement des fourmis réelles qui optimisent leur trajet entre le nid de la colonie et une source de nourriture. Dans la nature, les fourmis déposent le long de leur trajet des substances odorantes, appelées phéromones, qui marquent leur passage. Statistiquement, dans un temps donné, un trajet plus court reçoit plus de phéromones qu'un trajet plus long, car plus de fourmis l'empruntent. Ainsi, certains chemins deviennent plus marqués que d'autres. Or, comme les fourmis choisissent plus favorablement le trajet le plus marqué, le plus court chemin va être emprunté par de plus en plus de fourmis au cours du temps.

Depuis l'apparition des colonies de fourmis comme technique d'optimisation, de nombreuses variantes ont été proposées, plus ou moins proches du comportement des fourmis réelles. La variante que nous présentons ci-après (figure 2.2), bien qu'elle s'éloigne un peu du mécanisme naturel, apparaît plus performante que d'autres systèmes qui sont plus proches du comportement réel des fourmis.

Dans son principe de base, l'algorithme des colonies de fourmis utilise un ensemble de fourmis artificielles qui coopèrent pour résoudre une instance d'un problème décrit sous forme d'un graphe.
Initialement, chaque fourmi est affectée aléatoirement à un nœud du graphe ; ensuite, chacune essaye de construire une solution admissible en se déplaçant d'un nœud à un autre. Le choix

du prochain nœud à visiter est basé sur *une règle de transition aléatoire*, qui est fonction de deux paramètres :

- La distance de l'arc reliant le nœud courant à celui à visiter. Cette distance doit être la plus faible parmi tous les voisins du nœud courant.
- La quantité de phéromone déposée sur l'arc reliant le nœud courant à celui à visiter. Cette quantité, qui représente une mesure de désirabilité de chaque arc, doit être la plus importante parmi tous les voisins du nœud courant.

Dès que toutes les fourmis ont construit leur solution, *une mise à jour globale de la phéromone* est effectuée. Plusieurs schémas de mise à jour de la phéromone ont été proposés dans la littérature [Dor 04]. Ils consistent essentiellement en deux parties : l'évaporation et le renforcement (dépôt de phéromone). Pendant la phase d'évaporation, toutes les traces de phéromone sont diminuées suivant une valeur donnée (pourcentage, probabilité...). Cette évaporation permet d'éviter que la valeur de la phéromone ne déborde et que des arcs qui sont le moins visités soient défavorisés. Lors de la phase de renforcement, les meilleures traces de phéromone sont amplifiées. La quantité de la phéromone déposée sur chaque arc dépend de la qualité de la solution trouvée par chaque fourmi. Plus cette solution est meilleure, plus le taux de phéromone est élevé et vice-versa.

On retrouve pour les colonies de fourmis divers critères d'arrêt. En effet, la méthode peut être stoppée après qu'un certain nombre d'itérations ait été exécuté, ou bien, après qu'un certain nombre d'itérations n'ait produit aucune amélioration. Cependant, on considère aussi un autre critère d'arrêt spécifique particulièrement aux colonies de fourmis, appelé convergence. On dit qu'une méthode a convergé lorsque presque toutes les solutions de la population sont identiques.

Nous détaillons dans ce qui suit les étapes de l'algorithme à base de fourmis proposé pour la résolution du problème de couverture d'ensembles.

```
Algorithme : Ants_générique

Début
Etape 1 : /* Initialisation */
          Initialiser la phéromone.
Etape 2 : /* Itération */
         Pour chaque fourmi faire
         1. Construction de la solution en utilisant la règle de
            transition
         2. Mise à jour globale de la phéromone.
Jusqu'à atteindre un critère d'arrêt
Fin.
```

Figure 2.2. Schéma générique du fonctionnement de l'algorithme de colonie de fourmis.

2.4. Adaptation des colonies de fourmis au SCP

2.4.1. Principe de l'algorithme proposé *Ants*

Chaque colonne de la matrice A reçoit un taux de phéromone. Initialement ce taux est arbitraire, mais de même valeur pour chacune des colonnes. Ce taux de phéromone est utilisé à chaque étape de la phase de construction d'une solution par chacune des fourmis (le nombre maximum de fourmis utilisé est noté *ants_nb*). Plus une colonne a un taux de phéromone élevé et plus elle aura de chances de faire partie d'une solution construite par une fourmi.
Chaque fourmi conserve en mémoire un sous-ensemble de colonnes qui constituent une solution partielle. Initialement, cet ensemble est vide. A chaque étape, une fourmi sélectionne une colonne qu'elle ajoute au sous-ensemble qu'elle a en mémoire de manière à étendre sa solution partielle, pour obtenir finalement une solution réalisable (une couverture). Le choix de la colonne à ajouter est fonction, d'une part, du taux de phéromone de chacune des colonnes, mais également d'un *critère heuristique* chargé d'évaluer l'intérêt qu'il y a d'ajouter telle ou telle colonne à la solution partielle. Nous avons considéré deux critères heuristiques. Le premier est basé sur le critère $Val\_couv_j/coût_j$ où Val_couv_j est une valeur de couverture de la colonne j défini ci-après. Quant au second critère, il utilise le rapport $card_j/coût_j$, où $card_j$ représente le nombre de lignes recouvertes par la colonne j et non encore recouvertes dans la solution partielle.
La colonne choisie dans ces deux cas, par les fourmis, est celle qui a la plus grande valeur du rapport correspondant.

Le paramètre Val_couv_j appelé "valeur de couverture" d'une colonne j, est défini de la manière suivante. Notons s_k la solution partielle associée à la k-ième fourmi. On a alors

$$Val\_couv_j = \sum_{i \in couv(j,s_k)} \min\_coût(i)$$

où

- $couv(j,s_k)$ est l'ensemble des lignes recouvertes par la colonne j et qui ne sont pas encore recouvertes dans la solution partielle s_k.
- *min_coût(i)* est le coût minimum parmi les coûts des colonnes qui couvrent la ligne i.

Pour plus de précision, illustrons cette notion de "valeur de couverture" sur l'exemple de la figure 2.1.

Exemple :
Pour calculer, par exemple, la *valeur de couverture*, des colonnes 1 puis 3, on procède de la manière suivante :

- J=1 :

  Soit la solution partielle $s_1 = \varnothing$
  $couv(1, s_1) = \{e_1, e_3\}$
  $min\_coût\ (e_1) = \min\{1, 1\} = 1$
  La ligne 1 est recouverte par la colonne 1 avec un coût de 1 et par la colonne 3 avec un coût de 1 donc $min\_coût\ (e_1) = \min\{1,1\}$
  D'où $Val\_couv_1 = min\_coût(e_1) + min\_coût(e_3) = 1+1=2$

- J=3 :

  Soit $s_3=\varnothing$
  $couv(3, s_3) = \{ e_1, e_2, e_3\}$
  $min\_coût\ (e_1) = \min\{1, 1\} = 1$
  $min\_coût\ (e_2) = \min\{5, 1\} = 1$

$min\_coût\ (e_3) = \min\{1, 1, 3\} = 1$

$Val\_couv_3 = min\_coût(e_1) + min\_coût(e_2) + min\_coût(e_3) = 1+1+1=3$

Intuitivement cet exemple permet de conclure qu'il est plus intéressant d'ajouter la colonne 3 à une solution partielle $s_3=\varnothing$, que d'ajouter la colonne 1. En effet, $Val\_couv_3/coût_3 = 3/1 = 3$ et $Val\_couv_1/coût_1=2/1=2$.

Plus une colonne j a une valeur élevée du rapport $Val\_couv_j/coût_j$, plus elle est intéressante à ajouter à la solution partielle de la fourmi (ici la k-ième fourmi). En effet, plusieurs colonnes peuvent couvrir les mêmes lignes. Il serait donc plus intéressant, en terme de coût, que ce soit la colonne j ayant la valeur maximum de ce rapport qui soit ajoutée à la solution partielle de la fourmi, plutôt que d'ajouter d'autres colonnes (j est la colonne qui couvre le plus de lignes).

Lorsqu'une fourmi artificielle finit de construire une solution, c'est-à-dire un ensemble de colonnes qui couvrent toutes les lignes de la matrice, il se peut que certaines colonnes ne servent à rien. On dit d'une colonne qu'elle est redondante, si le fait de l'enlever de la solution permet d'obtenir une solution qui est toujours réalisable, c'est-à-dire qui couvre toujours l'ensemble des lignes de la matrice. Nous avons donc défini une phase d'*élimination des colonnes redondantes*. Dans l'exemple de la figure 2.1 avec la couverture réalisable $S_1 = \{e_1, e_3\}$, on remarque que la colonne 1 est redondante, car toutes les lignes sont recouvertes par la colonne 3. La colonne 1 peut en effet être supprimée.

Lorsque chacune des fourmis a construit une couverture, une mise à jour de la meilleure solution est effectuée. Puis, une mise à jour globale de la phéromone est effectuée. Une partie de cette phéromone s'évapore, puis chaque fourmi dépose une quantité de phéromone sur les colonnes qui constituent sa couverture. La quantité déposée dépend de la couverture établie par la fourmi. Par conséquent, plus une colonne est choisie, plus son taux de phéromone est élevé. A la fin, si les fourmis artificielles atteignent un nombre d'itérations limite (*nbiter_limite*), fixé expérimentalement à 50 lors de toutes les exécutions, alors le processus de recherche est arrêté et la meilleure solution trouvée est retournée. Sinon, une nouvelle itération de recherche est lancée.

Le schéma général de l'algorithme que nous proposons, appelé *Ants* par la suite, est donné à la figure 2.3. Cet algorithme utilise l'équation (1) pour le calcul de la probabilité du choix d'une colonne et l'équation (2) pour la mise à jour de la phéromone. Ces deux équations sont respectivement détaillées dans les sections 2.4.3 et 2.4.4.

2.4.2. Représentation de la solution et de la fonction objectif

Une solution, pour une fourmi donnée, est représentée par un vecteur de n bits, où n est le nombre de colonnes de l'instance du problème à traiter. Soit s le vecteur représentant la solution d'une fourmi. Si la colonne j appartient à la solution de la fourmi, alors $s[j]$ prend la valeur 1, et 0 sinon.

Colonne j	1	2	3	…	n-1	n
$Coût_j$	C_1	C_2	C_3		C_{n-1}	C_n

s[j]	1	0	1	…	0	0

Le coût d'une solution est alors donné par :

$$f(s) = \sum_{j=1..n} coût_j * s[j]$$

```
Algorithme : Ants

Début
Initialisation de la phéromone pour chaque colonne
Etape 1 : Initialiser chaque solution Fourmi_k
Etape 2 : Choisir une colonne selon les probabilités de l'équation
(1)
Insérer cette colonne dans Fourmi_k
Si Solution réalisable
  alors Eliminer les colonnes redondantes de Fourmi_k
        Si K:=k+1 > ants_nb
         alors Mettre à jour Fourmi-best
               Mettre à jour la phéromone selon l'équation (2)
               Si nb-iter ≥ nbiter_limite
                 alors Fin
                 sinon aller Etape 1
               fin si
         sinon aller  étape 2
        fin si
  sinon  aller Etape 2
fin si
Fin.
```

Figure 2.3. L'algorithme *Ants*.

2.4.3. La règle de transition d'une fourmi (choix d'une colonne)

Une fourmi *k* ajoute, à sa solution partielle, la colonne *j* selon la règle de transition (1) [Dor 92]. Dans cette règle, pour une colonne *j*, on multiplie la valeur de la phéromone *Phero[j]* par la valeur du critère heuristique correspondant *H[j]*. De cette manière, on favorise le choix des colonnes les plus prometteuses et qui ont un plus grand taux de phéromone.

$$P_k(j) = \begin{cases} \dfrac{Phero[j]^{\alpha} * H[j]^{\beta}}{\sum_{i=1}^{n} \left(Phero[i]^{\alpha} * H[i]^{\beta} \right)} & \text{si } j \notin s_k \\ 0 & \text{sinon} \end{cases} \qquad (1)$$

où :

- s_k est la solution courante, c'est-à-dire la solution partielle de la fourmi *k*.

- *Phero[j]* est le taux de phéromone de la colonne *j*.
- *n* est le nombre de colonnes.
- *H[j]* est la valeur du critère heuristique retenu
- α et β sont des paramètres déterminant l'importance relative de la phéromone par rapport au critère heuristique.

Afin de déterminer le critère heuristique intervenant dans le calcul de la probabilité du choix d'une colonne *j,* nous avons considéré deux versions d'un algorithme glouton. La première version, *Grcoval* (similaire à l'algorithme de [Mar 00]), est basée sur le critère $Val\_couv_j/coût_j$. Quant à la seconde, *Grcard* (similaire à l'algorithme de [Chv 79] et à l'algorithme de [Had 00]), elle utilise le rapport $card_j/coût_j$, où $card_j$ représente le nombre de lignes recouvertes par la colonne *j* et non encore recouvertes dans la solution partielle.

2.4.4. Mise à jour de la phéromone

Après avoir construit une couverture réalisable, la phéromone est mise à jour. Cette mise à jour permet d'influencer, par la suite, le choix des fourmis. Elle est réalisée en deux phases :

- Une phase d'évaporation, où chaque colonne perd une partie de sa phéromone. Cette évaporation permet d'éviter que certaines colonnes ne reçoivent trop de phéromone aux dépens d'autres, qui ne seraient jamais sélectionnées par les fourmis.
- Une phase d'intensification, où chaque fourmi dépose une quantité de phéromone sur les colonnes appartenant à sa solution. La quantité de phéromone déposée sur chaque colonne dépend de la qualité de la solution trouvée par la fourmi. Plus cette solution est de bonne qualité, c'est-à-dire moins son coût est élevé, et plus la quantité de phéromone ajoutée est importante.

Ainsi la mise à jour de la phéromone est réalisée de la manière suivante [Dor 92] :

$$Phero[j] \leftarrow (1-\rho) * Phero[j] + \sum_{k=1}^{m} \Delta Phero_k[j] \qquad (2)$$

$$\Delta Phero_k[j] = \begin{cases} 1/f(s_k) & \text{si } j \in s_k \\ 0 & \text{sinon} \end{cases}$$

Dans la formule ci-dessus, ρ est le paramètre qui détermine le taux de décroissance de la phéromone ($0<\rho\leq 1$), *m* est le nombre de fourmis et *f* représente la fonction coût. Le deuxième terme de l'addition dans cette formule fait apparaître, dans certaines versions des algorithmes de colonies de fourmis [Dor 04], un facteur de pondération. Ce dernier permet de contrôler l'interaction entre la phase d'évaporation et la phase de dépôt de la phéromone. Certaines versions des colonies de fourmis utilisent la valeur suivante :

$$\Delta Phero_k[j] = \begin{cases} Q/f(s_k) & \text{si } j \in s_k \\ 0 & \text{sinon} \end{cases}$$

Colorni et al. [Col 92] préconisent que, bien que la valeur de Q (paramètre fixé) ait peu d'influence sur le résultat final, cette valeur doit être du même ordre de grandeur qu'une estimation de la meilleure solution trouvée.
Pour rester dans le cadre d'un algorithme de base qui soit simple, nous avons opté pour une valeur de Q égale à 1.

2.4.5. Elimination des colonnes redondantes

Les colonnes redondantes d'une couverture s sont celles dont toutes les lignes sont recouvertes par d'autres colonnes. Lorsqu'une fourmi construit une couverture, les colonnes redondantes doivent être supprimées.
Dans [Mar 00], les colonnes redondantes sont supprimées pendant la construction de la couverture. Dans notre algorithme, nous avons choisi d'éliminer les colonnes redondantes après avoir construit la couverture.
En effet, nous montrons plus loin, dans la section 2.5.3, que l'élimination des colonnes redondantes après la construction d'une couverture est meilleure que leur élimination pendant la construction de la couverture. A cet effet, nous proposons deux algorithmes gloutons basés sur le critère heuristique $card_j/coût_j$ et qui sont similaires à celui donné dans [Chv 79]. Le premier algorithme *Grcard*(2), élimine les colonnes redondantes après la construction de la couverture, alors que l'algorithme *Grcard*(1) les élimine au cours de la construction de la couverture.
Ces algorithmes commencent par énumérer toutes les colonnes redondantes dans s. Puis, parmi ces colonnes, on supprime celles ayant la plus petite valeur du rapport ($card_j/coût_j$) que l'on a appelé "mauvaise" colonne dans l'algorithme. L'algorithme continue jusqu'à ce que l'on n'ait plus de colonne redondante. On élimine ainsi les colonnes qui engendreraient des efforts de calcul les plus élevés.
L'algorithme d'élimination des colonnes redondantes est donné dans la figure 2.4 ci-dessous.

```
Algorithme : Elim_col_redond

Début
Soit S une couverture réalisable
Etape 1 : Enumérer toutes les colonnes redondantes dans S
Supprimer de S la "mauvaise" colonne redondante
Si Colonnes redondantes alors aller Etape 1
                      sinon fin
Fin.
```

Figure 2.4. L'algorithme d'élimination des colonnes redondantes.

Avec toutes ces caractéristiques et après avoir fixé les ingrédients nécessaires à un algorithme à base de colonies de fourmis, nous présentons, dans ce qui suit, les spécificités des différentes expérimentations des algorithmes proposés.

2.5. Partie expérimentale

2.5.1. Environnement expérimental du SCP

Un ensemble de benchmarks représentant quelques instances du problème de couverture d'ensembles a été mis en ligne par Beasley [Bea 90]. La spécification de ces problèmes est donnée dans le tableau 2.1. Chaque classe de problèmes représente une famille d'instances ayant le même nombre de lignes et le même nombre de colonnes. La densité de remplissage indique le rapport entre le nombre de cases égales à 1 et celles égales à 0 dans la matrice booléenne.

Classe	Lignes (m)	Colonnes (n)	Densité de remplissage (%)	Intervalle des coûts	Nombre de problèmes
4	200	1000	2	1-100	10
5	200	2000	2	1-100	10
6	200	1000	5	1-100	5
A	300	3000	2	1-100	5
B	300	3000	5	1-100	5
C	400	4000	2	1-100	5
D	400	4000	5	1-100	5
E	500	5000	10	1-100	5
F	500	5000	20	1-100	5
G	1000	10000	2	1-100	5
H	1000	10000	5	1-100	5

Tableau 2.1. Les classes des problèmes du SCP [Bea 90].

La solution optimale est connue pour les instances des classes 4 à 6 et A à D. Les instances des classes E à H sont des instances de grande taille et la valeur de leurs solutions optimales n'est pas connue (tableau 2.2).

Dans la suite, tous les algorithmes implémentés sont testés sur les instances décrites ci-dessus [Bea 90] (voir tableau 2.1). Pour chaque instance, nous avons exécuté 10 fois l'algorithme.
Dans chaque tableau représentant les résultats des expérimentations réalisées :

- la colonne *Op/Bsf* indique la solution optimale ou la meilleure solution trouvée dans la littérature lorsque cette dernière n'est pas connue.
- la colonne *Best* indique la meilleure solution trouvée par l'algorithme sur les 10 exécutions de l'algorithme.
- *Fbest* indique la fréquence d'obtention de la meilleure solution dans les 10 exécutions effectuées.
- *Ecart* indique la différence moyenne de coût entre la solution trouvée par l'algorithme et la meilleure solution connue
- *Tbest* donne le temps moyen CPU écoulé pour l'obtention de la meilleure solution *Best*.
- *Tsol* indique le temps moyen (en secondes) au bout duquel une solution est trouvée.

Les algorithmes séquentiels proposés ont été écrits en C sous le système d'exploitation LINUX et testés sur un PC de type Pentium II (700 MHz, 64 Mo de RAM). Quant aux algorithmes parallèles, ils ont été codés en C sous le système de parallélisation PVM (réseau en anneau de Pentium II-700 MHz).

Instances	Coût de la meilleure solution	Instances	Coût de la meilleure solution
41	429	A1	253
42	512	A2	252
43	516	A3	232
44	494	A4	234
45	512	A5	236
46	560	B1	69
47	430	B2	76
48	492	B3	80
49	641	B4	79
410	514	B5	72
51	253	C1	227
52	302	C2	219
53	226	C3	243
54	242	C4	219
55	211	C5	215
56	213	D1	60
57	293	D2	66
58	288	D3	72
59	279	D4	62
510	265	D5	61
61	138	64	131
62	146	65	161
63	145		
E1	29	G1	176
E2	30	G2	154
E3	27	G3	166
E4	28	G4	168
E5	28	G5	168
F1	14	H1	63
F2	15	H2	63
F3	14	H3	59
F4	14	H4	58
F5	13	H5	55

Tableau 2.2. Les meilleures solutions connues pour les instances du SCP [Cap 00][Les 04].

2.5.2. Choix du critère heuristique

Pour fixer le critère heuristique à utiliser dans notre algorithme de colonies de fourmis, nous avons procédé à une comparaison de l'algorithme *Grcoval* basé sur le critère $Val\_couv_j/coût_j$ par rapport à l'algorithme *Grcard* basé sur le critère $card_j/coût_j$. Nous avons utilisé ces deux algorithmes pour résoudre quelques instances de la littérature [Bea 90]. Le tableau 2.3 rassemble les meilleurs résultats.

Inst.	Coût de la meilleure solution obtenue par *Grcard*	Coût de la meilleure solution obtenue par *Grcov*	Inst.	Coût de la meilleure solution obtenue par *Grcard*	Coût de la meilleure solution obtenue par *Grcov*	Inst.	Coût de la meilleure solution obtenue par *Grcard*	Coût de la meilleure solution obtenue par *Grcov*
41	**435**	**435**	51	269	**268**	F5	15	15
42	**529**	535	52	330	**317**	61	**144**	149
43	**540**	541	53	232	**230**	62	157	**156**
44	506	**503**	54	250	**246**	63	157	**150**
45	518	**515**	55	218	**217**	64	140	**135**
46	594	**588**	56	226	**219**	65	186	**180**
47	**447**	449	57	310	**302**	A1	261	**260**
48	525	**493**	58	311	**301**	A2	270	**262**
49	**664**	687	59	292	**281**	A3	**245**	246
410	528	**522**	510	277	**275**	A4	245	**242**
B1	73	**72**	D3	80	**78**	A5	251	**243**
B2	**78**	79	D4	68	**62**	G1	189	**186**
B3	**82**	84	D5	66	**65**	G2	**162**	167
B4	**83**	84	E1	30	30	G3	181	**179**
B5	75	**72**	E2	**32**	33	G4	181	**178**
C1	**237**	238	E3	31	**29**	G5	181	**179**
C2	**225**	232	E4	32	32	H1	69	69
C3	257	**254**	E5	30	30	H2	73	**70**
C4	237	**229**	F1	16	16	H3	63	63
C5	**219**	222	F2	16	16	H4	67	**65**
D1	67	**64**	F3	17	17	H5	61	**60**
D2	71	**68**	F4	17	17			

Tableau 2.3. Les tests comparatifs du critère heuristique pour le choix d'une colonne. Le temps CPU est, globalement, de l'ordre de 100 secondes.

Ces résultats permettent de constater que :

- dans 17% des benchmarks testés, les deux algorithmes trouvent les mêmes résultats.
- dans 22% des benchmarks testés, l'algorithme basé sur le critère heuristique *Grcard* trouve de meilleurs résultats que celui basé sur le critère heuristique *Grcov*.
- finalement, dans 61% des benchmarks testés, le critère heuristique *Grcov* trouve de meilleurs résultats que *Grcard*.

2.5.3. Choix de l'algorithme d'élimination de colonnes

Cette phase expérimentale, permet de fixer les avantages et les inconvénients quant au choix de la stratégie d'élimination des colonnes redondantes. Il s'agit, d'apprécier les résultats obtenus dans le cas où l'élimination des colonnes redondantes intervient pendant la construction d'une solution (*Grcard(1)*) ou à la fin (*Grcard(2)*). Les résultats de ces tests sont obtenus avec l'algorithme glouton *Grcard* et les résultats sont consignés dans le tableau 2.4.

Insts.	Coût de la meilleure solution obtenue par *Grcard(1)*	Coût de la meilleure solution obtenue par *Grcard (2)*	Insts.	Coût de la meilleure solution obtenue par *Grcard(1)*	Coût de la meilleure solution obtenue par *Grcard (2)*	Insts.	Coût de la meilleure solution obtenue par *Grcard(1)*	Coût de la meilleure solution obtenue par *Grcard (2)*
41	436	**435**	B4	83	83	61	144	144
42	533	**529**	B5	75	75	62	157	157
43	540	540	C1	239	**237**	63	157	157
44	512	**506**	C2	225	225	64	140	140
45	520	**518**	C3	257	257	65	186	186
46	605	**594**	C4	237	237	A1	261	261
47	447	447	C5	219	219	A2	272	**270**
48	525	525	D1	67	67	A3	245	245
49	671	**664**	D2	71	71	A4	248	**245**
410	531	**528**	D3	80	80	A5	**250**	251
51	269	269	D4	**67**	68	G1	189	189
52	330	330	D5	66	66	G2	162	162
53	232	232	E1	30	30	G3	182	181
54	250	250	E2	32	32	G4	181	181
55	218	218	E3	31	31	G5	181	181
56	226	226	E4	32	32	H1	69	69
57	310	310	E5	30	30	H2	73	73
58	311	311	F1	16	16	H3	63	63
59	292	292	F2	16	16	H4	67	67
510	277	277	F3	17	17	H5	61	61
B1	73	73	F4	17	17			
B2	78	78	F5	15	15			
B3	82	82						

Tableau 2.4. Tests comparatifs pour l'algorithme d'élimination des colonnes redondantes. Le temps CPU est de l'ordre de 100 secondes.

On constate que :

- Dans 80% des benchmarks testés, les deux algorithmes trouvent les mêmes résultats.
- Dans 3% des benchmarks testés, l'algorithme basé sur l'élimination des colonnes redondantes pendant la construction de la couverture *Grcard*(1) trouve de meilleurs résultats que celui basé sur l'élimination des colonnes redondantes après la construction de la couverture *Grcard*(2).
- Dans 17% des benchmarks testés, l'algorithme basé sur l'élimination des colonnes redondantes *Grcard*(2) après la construction de la couverture trouve de meilleurs résultats que celui basé sur l'élimination des colonnes redondantes pendant la construction de la couverture *Grcard*(1).
- Les résultats du tableau 2.4 sont, en effet, moins bons que ceux du tableau 2.3. Cette déduction est conforme à l'analyse présentée quant au choix du critère heuristique. En effet, l'algorithme glouton *Grcard* donne de moins bons résultats que l'algorithme *Grcov*.

Ces résultats permettent de conclure que l'élimination des colonnes redondantes après avoir construit une couverture *Grcard* (2) donne de meilleurs résultats que lorsqu'elle est effectuée pendant la construction de la couverture *Grcard* (1).

2.5.4. Etude et calibrage des paramètres de l'algorithme proposé *Ants*

L'efficacité de ce type de métaheuristiques est fortement liée aux valeurs de nombreux paramètres qu'elles utilisent. En absence d'étude théorique, nous ne pouvons échapper à un réglage empirique des valeurs des paramètres de cette méthode.
En fait, ces valeurs ont une influence directe sur le compromis diversification/intensification. Comme les autres méthodes à population de solutions, le système de fourmis n'est pas très ''exploiteur'' (intensificateur) ; il ne parvient pas à améliorer finement les solutions qu'il produit. Par contre, grâce à la diversité de la population initiale, cette méthode est naturellement exploratrice, c'est-à-dire qu'elle favorise la diversification.
Dans ce qui suit, nous présentons, suite aux différentes expérimentations réalisées sur des benchmarks de la littérature (tableau 2.1), les meilleures valeurs trouvées des paramètres.

- **Nombre de fourmis :** Le paramètre *Ants_nb* représente le nombre de fourmis qui sont utilisées dans l'algorithme. La valeur de *Ants_nb* est fixée à 25 car les tests ont montré qu'au-delà de 25 fourmis, les résultats ne sont pas améliorés.

- **Paramètres α et β :** Ces deux paramètres déterminent la relative influence de la phéromone par rapport au critère heuristique. Des tests sont réalisés et permettent de constater que plus $\alpha >> \beta$ plus le système converge rapidement vers des valeurs proches de l'optimum mais moins la diversification est importante. Par contre, pour des valeurs de $\alpha << \beta$, la diversification est plus importante, l'espace des solutions est plus grand, ce qui fait que la convergence des fourmis est plus lente. Nous avons fixé tout au long des tests effectués α à 1 et β à 5.

- **Paramètre ρ :** Ce paramètre représente le taux d'évaporation de la phéromone (formule (2)). En le faisant varier, on détermine l'influence de cette phéromone lors du choix des fourmis (règle de transition des fourmis) puisque, avec une valeur proche de 1, la phéromone s'est presque entièrement évaporée à chaque nouvelle itération, alors qu'une petite valeur privilégie les colonnes qui viennent d'être choisies au tour précédent, ce qui accélère la convergence des fourmis (cette situation est assimilée à une accumulation de la phéromone). Dans nos expérimentations, nous avons donné à ce paramètre la valeur de 0.5.

2.5.5. Tests expérimentaux de l'algorithme *Ants*

Nous avons exécuté notre algorithme 10 fois sur chacune des instances. Le nombre de fourmis est fixé à 25, α à 1, β à 5 et ρ à 0.,5. Les meilleurs résultats sont exposés dans le tableau 2.5.

Nous remarquons que l'algorithme trouve rarement la solution optimale ou encore la meilleure solution connue dans la littérature. Statistiquement, l'algorithme a trouvé la meilleure solution pour seulement 2 instances parmi les 65 testées.

D'autre part, l'écart moyen est estimé à 5,66 par rapport aux meilleures solutions connues. Ce manque d'efficacité est dû à une insuffisance des colonies de fourmis, dans l'intensification de la recherche. Pour pallier à cela et afin d'améliorer la qualité des solutions, nous avons hybridé l'algorithme *Ants* avec une recherche locale.

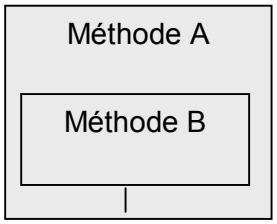
Insts.	*Op/Bsf*	*Ants*			Insts.	*Op/Bsf*	*Ants*		
		Best	*Fbest*	*Ecart %*			*Best*	*Fbest*	*Ecart %*
41	429	433	1	0,93	A1	253	256	0.2	1,19
42	512	535	0.4	4,49	A2	252	254	0.4	0,79
43	516	541	0.2	4,84	A3	232	234	0.4	0,86
44	494	503	0.6	1,82	A4	234	235	0.2	0,43
45	512	515	0.4	0,59	A5	236	236	0.6	0,00
46	560	588	0.4	5,00	B1	69	72	0.2	4,35
47	430	449	0.2	4,42	B2	76	79	1	3,95
48	492	493	1	0,20	B3	80	84	0.6	5,00
49	641	687	0.6	7,18	B4	79	84	0.6	6,33
410	514	522	0.6	1,56	B5	72	**72**	1	0,00
51	253	268	0.2	5,93	C1	227	238	0.4	4,85
52	302	317	0.4	4,97	C2	219	232	0.2	5,94
53	226	230	0.2	1,77	C3	243	254	0.2	4,53
54	242	246	0.2	1,65	C4	219	229	0.4	4,57
55	211	217	1	2,84	C5	215	222	0.2	3,26
56	213	219	1	2,82	D1	60	64	1	6,67
57	293	302	1	3,07	D2	66	68	1	3,03
58	288	301	0.4	4,51	D3	72	78	0.6	8,33
59	279	281	0.4	0,72	D4	62	**62**	1	0,00
510	265	275	0.2	3,77	D5	61	65	0.2	6,56
61	138	149	0.4	7,97	64	131	135	1	3,05
62	146	156	0.8	6,85	65	161	180	0.2	11,80
63	145	150	1	3,45					
E1	29	30	1	3,45	G1	176	180	0.4	2,27
E2	30	33	1	10,00	G2	154	167	0.2	8,44
E3	27	29	0.4	7,41	G3	166	179	0.4	7,83
E4	28	32	1	14,29	G4	168	178	0.2	5,95
E5	28	30	1	7,14	G5	168	179	0.2	6,55
F1	14	16	0.2	14,29	H1	63	69	1	9,52
F2	15	16	1	6,67	H2	63	70	0.6	11,11
F3	14	17	0.8	21,43	H3	59	63	0.2	6,78
F4	14	17	1	21,43	H4	58	65	0.2	12,07
F5	13	15	1	15,38	H5	55	60	0.8	9,09

Tableau 2.5. Résultats des tests de l'algorithme *Ants*. Le temps CPU varie, environ, entre 100 et 300 secondes pour les instances de tailles inférieures à 400 lignes et 4000 colonnes et est de l'ordre de 1000 secondes pour les instances plus grandes.

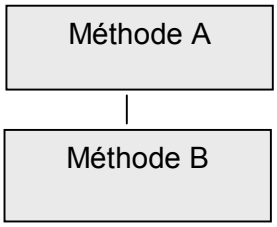
2.6. Hybridation de l'algorithme des colonies de fourmis avec une recherche locale

2.6.1. Quelques modèles d'hybridation

Cette section commence par donner un bref aperçu sur quelques types d'hybridations qui seront utilisées dans le cadre de ce travail.
Talbi dans [Tal 02] propose une taxinomie composée de deux modèles de classification : la *classification hiérarchique* et la *classification à plat*. Dans le cadre de cette thèse, nous ne nous intéressons qu'à la *classification hiérarchique*. Cette classification nécessite la définition de la notion de *niveau d'hybridation* (bas ou haut niveau) et de la notion de *mode d'hybridation (relais ou coévolution)*. L'*hybridation de haut niveau* conserve l'intégrité des méthodes qu'elle lie. Il n'y a pas de relation directe entre les mécanismes internes des métaheuristiques hybridées (figure 2.5).



Hybridation de *BAS niveau.*
La méthode A est modifiée et la méthode B devient un élément fonctionnel de A.



Hybridation de *HAUT niveau.*
Les méthodes A et B ne sont pas modifiées.

Figure 2.5. Les niveaux d'hybridation.

Le mode de l'hybridation distingue les hybrides en mode *relais* des hybrides en mode *coévolution.* En mode *relais*, un ensemble de métaheuristiques est appliqué l'une à la suite de l'autre dans un ordre prédéterminé, chacune utilisant les résultats de la précédente comme solution initiale. Par contre, l'hybridation en mode *coévolution* représente un modèle coopératif, dans lequel nous avons plusieurs agents coopérants parallèlement, chaque agent mettant en exécution une recherche dans un espace de solutions donné. Dans les hybrides de haut niveau en mode relais, les heuristiques hybridées conservent leur intégrité. Les méthodes HRH (*High-level Relay Hybrid*) hybridées sont exécutées en séquence. La figure ci-dessous montre des instances du schéma d'hybridation HRH.

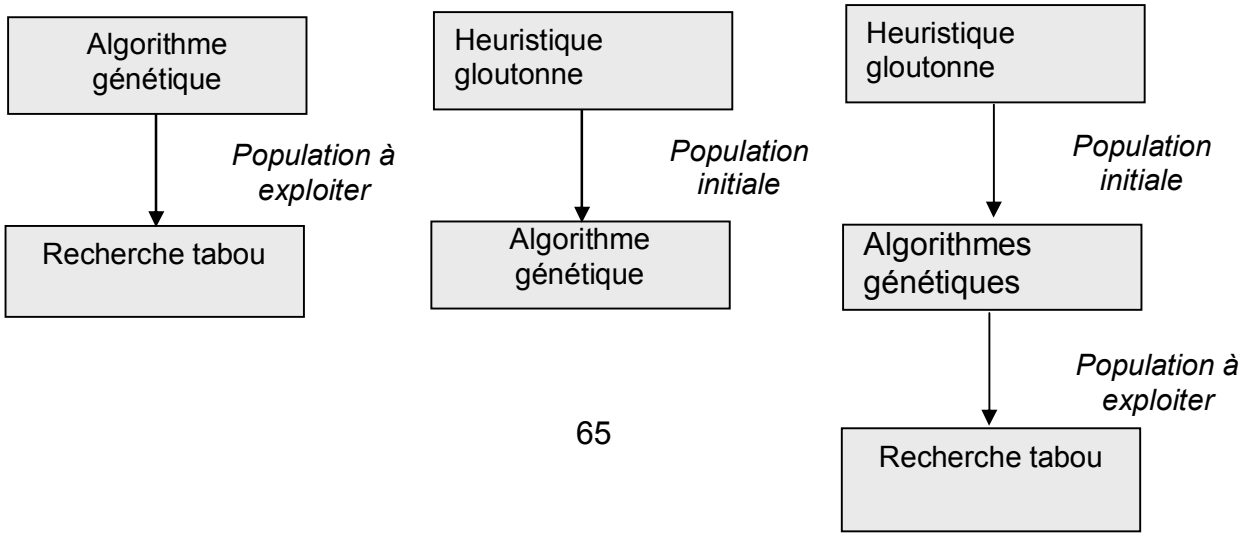


Figure 2.6. Exemples d'hybridation de haut niveau en mode relais : HRH (*High-level Relay Hybrid*).

Dans les hybridations de haut niveau coévolutionnaires HCH (*High-level Co-evolutionary Hybrid*), la structure interne des métaheuristiques hybridées n'est pas modifiée. Ces dernières sont exécutées simultanément et coopèrent pour résoudre le problème (voir exemple de la figure 2.6).

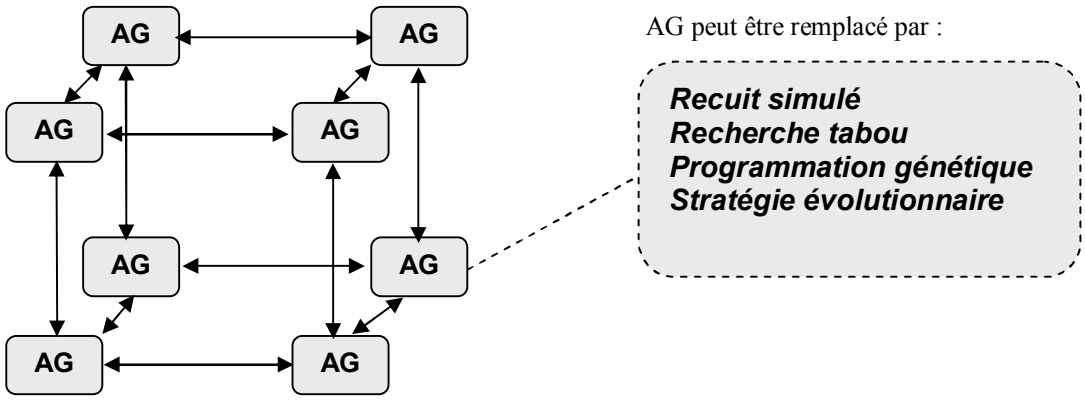


Figure 2.7. Hybridation de haut niveau coévolutionnaire : HCH (*High-level Co-evolutionary Hybrid*).

2.6.2. La recherche locale (*LS*)

Afin d'utiliser un algorithme de recherche locale il est nécessaire de définir le voisinage d'une solution. Pour le problème que nous considérons, la difficulté réside dans le fait que les solutions voisines doivent être réalisables, c'est-à-dire correspondre à des couvertures. Par conséquent la génération du voisinage d'une solution S se fait en 2 étapes : une étape de *perturbation* et une étape de *réparation*.

La première étape consiste à perturber la solution S, en choisissant aléatoirement un certain nombre de colonnes. Celles-ci constituent l'ensemble C. Ensuite, pour chacune des colonnes choisies, si la colonne choisie appartient à la solution, alors elle est supprimée de la solution S, sinon elle est ajoutée à S. Cette transformation revient à appliquer une différence symétrique entre l'ensemble des colonnes choisies C et l'ensemble des colonnes de S. Rappelons que la différence symétrique Δ entre deux ensembles (de colonnes ici) S et C est égale à $\{S\text{-}C \cup C\text{-}S\}$. Dans la deuxième étape, on répare la solution en ajoutant des colonnes j selon le critère heuristique ($Val\_couv_j / coût_j$) et tel que $coût_j$ soit inférieur ou égal à la plus grande valeur de coût des colonnes de la solution S, noté E.

L'algorithme de génération du voisinage de S, *Ant-voisinage(S)*, est explicité dans la figure 2.8. Les paramètres utilisés sont les suivants :

- $N(S)$: nombre de colonnes de S ;
- $Q(S)$: max ($coût_j \mid j \in S$), le coût maximum des éléments de S ;
- ρ_1 : le taux de suppression de colonnes d'une solution S ;
- ρ_2 : le taux de réduction du coût maximal ;

- $D=\rho_1 * N(S)$ et $E=\rho_2 * Q(S)$.

Exemple :

Pour plus de clarté, nous appliquons l'algorithme *Ant-voisinage(S_c)*, à l'exemple de la figure 2.1.

Initialisation :

$S_c=\{1,2\}$
$N(S_c)=2$
$Q(S_c)=5$
$E=\rho_2 * Q(S_c)=5,5$
$D=\rho_1 * N(S_c)=2$ avec D égal à $|C|$

Perturbation :

On choisit aléatoirement un ensemble C de colonnes dont le cardinal est égal à D
$C=\{2,4\}$
$|C|=2$
S_c est modifié comme suit $S_c \Delta C=\{1,4\}$ cette solution n'est pas réalisable d'où la nécessité de la réparer.

Réparation :

- Soit l'ensemble des colonnes j telles que $j \notin \{1,4\}$ la solution en cours et dont le coût de chaque colonne soit inférieur ou égal à E. Cet ensemble est égal à $\{2\}$
- Cette colonne est automatiquement insérée, selon le critère heuristique retenu, dans la solution à réparer.
- L'ensemble de colonnes obtenu est alors égal à $\{1,2,4\}$.
  Cet ensemble est bien une couverture.
  Donc la solution voisine S_v de S_c est égale à $\{1,2,4\}$.

Algorithme : *Ant-voisinage(S)*
Début
Soient les paramètres suivants :
S : la solution ;
ρ_1 : le taux de modification de colonnes d'une solution S ;
ρ_2 : le taux de réduction de coût de $Q(s)$;
N(S): le nombre de colonnes de S ;
Q(S): le maximum des coûts des colonnes de S ;
D = ρ1 *N(S) ;
E = ρ2 *Q(S) ;

Phase de perturbation

Choisir aléatoirement $|C|$ *colonnes* : $S := S \Delta C$ /*différence symétrique*/

Phase de réparation

- **Etape 1** : Enumérer toutes les colonnes j telles que $coût_j \leq E$ et $j \notin S$
- Choisir aléatoirement parmi les colonnes énumérées la colonne k qui possède le maximum des rapports cov_val_k / $coût_k$
- Insérer cette colonne dans S
- **Si** Solution réalisable **alors** fin

```
                    sinon aller étape 1
 fin si
Fin.
```

Figure 2.8. Génération du voisinage d'une solution *S* : *Ant-voisinage(S)*.

L'algorithme de recherche locale *LS* (voir la figure 2.9), utilise à chaque itération l'algorithme *Ant-voisinage(S_c)* pour tenter d'améliorer la solution courante S_c. En appliquant la phase de perturbation, puis la phase de réparation sur S_c, on obtient S_v. Si le coût de celle-ci est meilleur, alors elle devient la solution courante. Ce processus est itéré jusqu'à à ce que le nombre d'itérations, fixé expérimentalement, soit atteint.

```
Algorithme : LS

Début
  Soient les paramètres suivants :
  Nb_iteration : le nombre d'itérations à appliquer dans la
  recherche locale,
  k : la variable de contrôle de Nb_iteration,
  Sc : la solution courante,
  Sv : la solution voisine.

  - k:=1;
  - Etape 1 : Sv = Ant-voisinage(Sc)
  - Si f(Sv) < f(Sc) alors Sc := Sv
                          Etape 2 : Si k:=k+1 < Nb_iteration
                                    alors Fin
                                    sinon aller étape 1
                                   fin si
                    sinon aller étape 2
     fin si
Fin.
```

Figure 2.9. L'algorithme de recherche locale : *LS*.

Le paramètre *Nb_iteration* indique le nombre d'itérations totales effectuées par la recherche locale. Le temps de recherche dépend d'une façon linéaire de *Nb_iteration*. Nous avons choisi expérimentalement la valeur 50 lors de l'application de la recherche locale à la dernière itération des fourmis et la valeur 200 lors de son application à la meilleure solution trouvée.

2.6.3. Etude et calibrage des paramètres de la recherche locale

- **Paramètre ρ_l :** Ce paramètre détermine le taux de colonnes à modifier dans une solution pendant la recherche locale. Une grande valeur correspondrait à une nouvelle construction de la solution en utilisant le critère heuristique. Par contre, une petite valeur pourrait restreindre le voisinage d'une solution et ne pas être efficace. Pour le reste des expérimentations, nous avons choisi ρ_l pour qu'il soit uniformément distribué parmi les fourmis de telle manière que la valeur minimale soit égale à 0% et la valeur maximale soit égale à 15% [Jac 95].

- **Paramètre ρ_2 :** Ce paramètre détermine le taux du coût maximal des colonnes à choisir durant la recherche. Une petite valeur de ρ_2 pénalise les colonnes *j* telles que $coût_j > \rho_2 * Q(s)$, même si ces colonnes permettent d'avoir des solutions optimales, nous avons alors moins de diversification. Cependant, une grande valeur de ρ_2 assure une bonne diversification, mais pour un temps de recherche très important. C'est pourquoi, nous avons choisi de fixer la valeur de ρ_2 à 1,1 [Jac 95].

2.6.4. Tests expérimentaux de l'algorithme *LS*

Pour tester l'efficacité de l'algorithme basé sur la recherche locale (*LS*), chaque fourmi construit une solution selon le critère de l'heuristique gloutonne (voir la section 2.4.3) seulement. Ensuite, l'algorithme de recherche locale est lancé pour améliorer ces solutions. Les valeurs des paramètres de cet algorithme sont fixées comme suit : $0\% \leq \rho_1 \leq 15\%$, $\rho_2 = 1{,}1$ et *Nb_iteration*=50 lors de l'application de la recherche locale à la dernière itération des fourmis et *Nb_iteration*=200 lors de son application à la meilleure solution trouvée. Les résultats obtenus sont exposés dans le tableau 2.6.

Instances	*Op/Bsf*	*LS*			Instances	*Op/Bsf*	*LS*		
		Best	*Fbest*	*Ecart %*			*Best*	*Fbest*	*Ecart %*
41	429	433	1	0.93	A1	253	257	0.2	1.90
42	512	515	0.4	1.33	A2	252	258	0.4	2.62
43	516	519	0.2	1.36	A3	232	240	0.4	3.79
44	494	498	0.6	1.09	A4	234	236	0.2	1.20
45	512	514	0.4	0.51	A5	236	238	0.6	1.02
46	560	562	0.4	0.54	B1	69	**69**	0.2	1.45
47	430	433	0.2	0.88	B2	76	**76**	1	0.00
48	492	493	1	0.20	B3	80	**80**	0.6	1.00
49	641	668	0.6	4.43	B4	79	80	0.6	1.77
410	514	516	0.6	0.62	B5	72	**72**	1	0.00
51	253	**253**	0.2	2.45	C1	227	233	0.4	2.91
52	302	312	0.4	3.58	C2	219	224	0.2	2.83
53	226	228	0.2	1.24	C3	243	247	0.2	2.14
54	242	242	0.2	1.07	C4	219	223	0.4	2.10
55	211	214	1	1.42	C5	215	**215**	0.2	0.56
56	213	216	1	1.41	D1	60	62	1	3.33
57	293	297	1	1.37	D2	66	67	1	1.52
58	288	291	0.4	2.01	D3	72	76	0.6	6.11
59	279	**279**	0.4	0.22	D4	62	**62**	1	0.00
510	265	267	0.2	1.28	D5	61	62	0.2	2.95
61	138	142	0.4	3.33	64	131	134	1	2.29
62	146	149	0.8	2.33	65	161	171	0.2	7.20
63	145	148	1	2.07					
E1	29	**29**	1	0.00	G1	176	182	0.4	3.75
E2	30	32	1	6.67	G2	154	159	0.2	5.06
E3	27	28	0.4	5.93	G3	166	174	0.4	5.18
E4	28	29	1	3.57	G4	168	176	0.2	5.24
E5	28	**28**	1	0.00	G5	168	174	0.2	4.05
F1	14	**14**	0.2	5.71	H1	63	65	1	3.17
F2	15	**15**	1	0.00	H2	63	67	0.6	6.98

F3	14	15	0.8	8.57	H3	59	61	0.2	4.75
F4	14	15	1	7.14	H4	58	62	0.2	8.28
F5	13	15	1	15.38	H5	55	57	0.8	4.00

Tableau 2.6. Résultats des tests de l'algorithme *LS*. Le temps CPU ne dépasse pas, approximativement, 100 secondes.

Nous remarquons que l'algorithme de recherche locale trouve la solution optimale ou la meilleure solution connue dans la littérature pour 20% des benchmarks testés.
Ces résultats sont meilleurs que ceux trouvés par l'algorithme basé sur les colonies de fourmis *Ants*. Cette différence est due au manque de diversification qui peut être procurée par les colonies de fourmis.
Nous allons montrer que l'hybridation des deux méthodes de recherche va nettement améliorer la qualité des solutions trouvées.

2.6.5. Hybridation des colonies de fourmis et de la recherche locale en mode *HRH* : *Ants+LS*

L'algorithme global implémenté, qui hybride les colonies de fourmis et la recherche locale en mode *relais de haut niveau HRH,* est décrit comme suit (figure 2.10 cas (a)) : dans la première étape, tous les paramètres sont initialisés. Dans la deuxième étape, le processus de recherche des fourmis est lancé. A chaque itération, chaque fourmi construit une solution selon l'équation (1). Lorsque toutes les fourmis atteignent le nombre maximum d'itérations fixées préalablement, le processus de recherche des fourmis est arrêté. Ainsi, dans la dernière étape, l'algorithme de recherche locale est appliqué à la meilleure solution trouvée par les fourmis artificielles.

```
Algorithmes : (a) Ants+LS et (b) AntsLS

Début
 Les paramètres utilisés par ces deux algorithmes sont :
 Fourmi_k : solution de chaque fourmi ; Phéro_j : taux de phéromone de
 chaque colonne ; k : contrôle le nombre de fourmis ; Fourmi-best :
 Meilleure solution trouvée ; nb-iter : nombre d'itérations;
 ants_nb : nombre de fourmis ; nbiter_limite : nombre d'itérations
 limites ;

 Initialisation de la phéromone pour chaque colonne
 Etape 1 : Initialiser chaque solution Fourmi_k
 Etape 2 : Choisir une colonne selon l'équation (1)
 Insérer cette colonne dans Fourmi_k
 Si Solution réalisable
   alors  Eliminer les colonnes redondantes de Fourmi_k
          Cas (b): appliquer ici la recherche locale LS(Fourmi-best). On a alors AntsLS.
          Si k:=k+1 > ants_nb
           alors
             Mise à jour de Fourmi-best
             Mise à jour de la phéromone selon l'équation (2)
             Si nb-iter ≥ nbiter-limite
```

```
                    alors
                     Cas (a) : appliquer ici la recherche locale LS. On a alors Ants+LS.
                     Aller à fin
                    sinon aller étape 1
                  fin si
                 sinon aller étape 2
               fin si
          sinon aller étape 2
     fin si
Fin.
```

Figure 2.10. Deux algorithmes d'hybridation entre la recherche locale et les colonies de fourmis. La recherche locale *LS* est exécutée selon le cas : soit en (a) *Ants+LS* soit en (b) *AntsLS*.

2.6.6. Tests d'expérimentation de l'algorithme *Ants+LS*

Les résultats des tests de l'algorithme *Ants + LS* proposé sont basés sur 10 exécutions pour chaque instance. Le tableau 2.7 rapporte les résultats trouvés en appliquant les colonies de fourmis combinées avec la recherche locale en mode *HRH*.

Insts.	*Op/Bsf*	*Ants+LS*					Insts.	*Op/Bsf*	*Ants+LS*				
		Best	*Fbest*	*Ecart %*	*Tbest*	*Tsol*			*Best*	*Fbest*	*Ecart %*	*Tbest*	*Tsol*
41	429	430	0.3	0.56	70	32	A1	253	256	0.6	1.38	200	150
42	512	513	0.3	0.68	65	25	A2	252	254	0.3	1.15	230	151
43	516	517	0.3	0.60	67	35	A3	232	234	0.4	1.25	225	139
44	494	**494**	0.5	0.18	71	38	A4	234	235	0.3	0.81	212	148
45	512	**512**	0.3	0.27	50	28	A5	236	**236**	0.4	0.30	220	140
46	560	562	0.4	0.50	61	30	B1	69	**69**	1	0.00	40	38
47	430	431	0.7	0.44	66	31	B2	76	**76**	1	0.00	48	40
48	492	493	1	0.20	60	60	B3	80	81	1	1.25	50	48
49	641	648	0.3	1.33	59	29	B4	79	**79**	1	0.00	62	50
410	514	**514**	0.7	0.21	58	26	B5	72	**72**	1	0.00	60	56
51	253	**253**	0.1	0.87	75	42	C1	227	229	0.3	1.63	55	48
52	302	304	0.3	2.19	71	44	C2	219	221	0.6	1.19	59	49
53	226	228	0.3	1.24	73	49	C3	243	244	0.4	1.11	57	45
54	242	**242**	1	0.00	50	50	C4	219	**219**	0.6	0.55	56	46
55	211	**211**	1	0.00	54	51	C5	215	216	0.6	0.65	60	49
56	213	**213**	1	0.00	45	45	D1	60	**60**	0.6	0.67	65	55
57	293	294	0.6	0.48	96	47	D2	66	**66**	0.5	0.76	66	58
58	288	**288**	0.5	0.35	98	44	D3	72	74	0.6	3.33	68	59
59	279	**279**	0.5	0.18	97	48	D4	62	**62**	1	0.00	63	60
510	265	269	0.6	1.66	99	46	D5	61	**61**	1	0.00	62	60
61	138	141	0.5	2.54	102	50	64	131	**131**	0.7	0.31	91	58
62	146	**146**	0.4	0.68	98	56	65	161	162	0.2	2.98	98	59
63	145	**145**	0.5	1.03	99	61							
E1	29	**29**	1	0.00	443	120	G1	176	**176**	0.4	1.19	3660	1200
E2	30	**30**	1	0.00	445	125	G2	154	157	0.4	2.60	3637	1300
E3	27	28	1	3.70	443	110	G3	166	171	0.6	3.25	2010	986
E4	28	**28**	1	0.00	440	130	G4	168	172	0.6	2.86	3662	890
E5	28	**28**	1	0.00	440	128	G5	168	169	0.6	1.13	2009	1250

F1	14	**14**	1	0.00	438	30	H1	63	65	0.7	3.65	1821	1200
F2	15	**15**	1	0.00	438	25	H2	63	64	0.7	2.06	1822	1320
F3	14	**14**	0.7	2.14	438	23	H3	59	**59**	0.8	0.68	1821	1350
F4	14	**14**	1	0.00	437	29	H4	58	59	0.5	2.59	1820	750
F5	13	14	1	7.69	437	31	H5	55	**55**	0.6	0.73	1811	910

Tableau 2.7. Résultats de l'algorithme *Ants+LS*.

Les résultats fournis par cet algorithme hybride *Ants+LS* en mode *HRH* approchent les solutions optimales pour 49% des instances de taille inférieure à 400 lignes et 4000 colonnes et trouve la meilleure solution pour 55% des instances dont la taille est plus grande que 400 lignes et 4000 colonnes.
Ces résultats montrent clairement que les colonies de fourmis combinées avec la recherche locale donnent de meilleurs résultats que ceux trouvés lors de l'application de la recherche locale seulement ou des colonies de fourmis seulement. Nous montrons par la suite que ces résultats peuvent encore être améliorés en appliquant un autre type d'hybridation, le *HCH*.

2.6.7. Hybridation des colonies de fourmis et de la recherche locale en mode *HCH* : *AntsLS*

La figure 2.10, cas (b), présente une hybridation de *haut niveau en mode coévolution*. Dans la première étape, tous les paramètres sont initialisés. Dans la deuxième étape, le processus de recherche des fourmis est lancé. A chaque itération, chaque fourmi construit une solution selon l'équation (1). Après avoir construit une couverture réalisable (solution réalisable), chaque fourmi applique une recherche locale sur sa solution pour améliorer la qualité de la couverture obtenue.

2.6.8. Tests d'expérimentation de l'algorithme *AntsLS*

Le tableau 2.8 rassemble les résultats trouvés en appliquant l'algorithme hybride *AntsLS*. L'algorithme a été testé sur des instances de la littérature [Bea 90].
A partir de ces résultats, nous remarquons que l'algorithme *AntsLS* hybridant les colonies de fourmis et la recherche locale en mode *HCH* approche les solutions optimales pour 55% des instances dont la taille est plus petite que 400 lignes et 4000 colonnes et trouve la meilleure solution pour 60% des instances de plus grande taille.
D'autre part, nous remarquons que les temps de recherche ont été, à peu près, multipliés par 2. L'augmentation de ces temps de recherche est due à l'application de la recherche locale par chaque fourmi artificielle. Cependant, pour réduire le temps de recherche et essayer d'améliorer la qualité des solutions trouvées, nous allons présenter deux implémentations parallèles de l'algorithme *AntsLS*.

2.7. Implémentations parallèles de algorithme de recherche *AntsLS*

La parallélisation des méthodes à base de population de solutions est fortement inspirée des travaux de parallélisation des algorithmes génétiques [Kap 94][Sch 94][Tal 95][Sol 02]. Le principe de fonctionnement de ces algorithmes repose sur l'évolution d'une population [Fal 97][Bul 98]. Dans les phases de sélection, d'association et de remplacement,

l'algorithme doit gérer la globalité de la population. C'est un handicap pour la parallélisation, puisque la gestion d'une mémoire globale d'un grand volume (toute la population) requiert beaucoup d'accès sur une architecture à mémoire distribuée apportant un surcoût prohibitif. Ainsi, les différents modèles de parallélisation des méthodes à base de population de solutions reposent sur les différentes façons de distribuer la population en sous-populations sur les processeurs [Cat 01], chaque sous-population évoluant plus ou moins indépendamment des autres.

La parallélisation permet, entre autres, d'augmenter les chances d'atteindre les solutions optimales, d'aborder des instances de grande taille et de réduire le temps de recherche. Nous allons montrer, au travers des différentes implémentations parallèles proposées, et des différents tests effectués, que ces objectifs sont atteints.

Instances	*Op/Bsf*	*AntsLS*					Instances	*Op/Bsf*	*AntsLS*				
		Best	*Fbest*	*Ecart %*	*Tbest*	*Tsol*			*Best*	*Fbest*	*Ecart %*	*Tbest*	*Tsol*
41	429	430	0.3	0.56	140	64	A1	253	**253**	0.6	1.38	400	300
42	512	513	0.3	0.68	130	50	A2	252	**252**	0.3	1.15	460	302
43	516	517	0.3	0.60	134	70	A3	232	234	0.4	1.25	450	278
44	494	**494**	0.5	0.18	142	76	A4	234	235	0.3	0.81	424	296
45	512	**512**	0.3	0.27	100	56	A5	236	**236**	0.4	0.30	440	280
46	560	562	0.4	0.50	122	60	B1	69	**69**	1	0.00	80	76
47	430	431	0.7	0.44	132	62	B2	76	**76**	1	0.00	96	80
48	492	493	1	0.20	120	120	B3	80	81	1	1.25	100	96
49	641	648	0.3	1.33	118	58	B4	79	**79**	1	0.00	124	100
410	514	**514**	0.7	0.21	116	52	B5	72	**72**	1	0.00	120	112
51	253	**253**	0.1	0.87	150	84	C1	227	**227**	0.3	1.63	110	96
52	302	304	0.3	2.19	142	88	C2	219	221	0.6	1.19	118	98
53	226	228	0.3	1.24	146	98	C3	243	244	0.4	1.11	114	90
54	242	**242**	1	0.00	100	100	C4	219	**219**	0.6	0.55	112	92
55	211	**211**	1	0.00	108	102	C5	215	216	0.6	0.65	120	98
56	213	**213**	1	0.00	90	90	D1	60	**60**	0.6	0.67	130	110
57	293	294	0.6	0.48	192	94	D2	66	**66**	0.5	0.76	132	116
58	288	**288**	0.5	0.35	196	88	D3	72	74	0.6	3.33	136	118
59	279	**279**	0.5	0.18	194	96	D4	62	**62**	1	0.00	126	120
510	265	269	0.6	1.66	198	92	D5	61	**61**	1	0.00	124	120
61	138	141	0.5	2.54	204	100	64	131	**131**	0.7	0.31	182	116
62	146	**146**	0.4	0.68	196	112	65	161	162	0.2	2.98	196	118
63	145	**145**	0.5	1.03	198	122							
E1	29	**29**	1	0.00	891,6	240	G1	176	**176**	0.4	1.19	7329,6	2400
E2	30	**30**	1	0.00	897,2	250	G2	154	155	0.4	2.60	7274,8	2600
E3	27	28	1	3.70	891,6	220	G3	166	170	0.6	3.25	4062,4	1972
E4	28	**28**	1	0.00	893,6	260	G4	168	170	0.6	2.86	7350,4	1780
E5	28	**28**	1	0.00	893,2	256	G5	168	169	0.6	1.13	4028,0	2500
F1	14	**14**	1	0.00	876,4	60	H1	63	65	0.7	3.65	3623,6	2400
F2	15	**15**	1	0.00	876,8	50	H2	63	**63**	0.7	2.06	3644,0	2640
F3	14	**14**	0.7	2.14	876,8	46	H3	59	**59**	0.8	0.68	3624,8	2700
F4	14	**14**	1	0.00	877,2	58	H4	58	59	0.5	2.59	3619,2	1500
F5	13	14	1	7.69	877,2	62	H5	55	**55**	0.6	0.73	3617,2	1820

Tableau 2.8. Résultats de l'algorithme *AntsLS*.

2.7.1. Parallélisation multidépart indépendante

Il s'agit de lancer simultanément plusieurs recherches basées sur l'algorithme *AntsLS*. Cette parallélisation ne nécessitant pas de communication entre les processeurs, puisqu'il s'agit de recherches indépendantes, elle est particulièrement bien adaptée aux architectures dépourvues de communication à haut débit, telles qu'un réseau de stations de travail.
Le principe de cette implantation est très simple (voir la figure 2.11) : un processeur maître est chargé de créer plusieurs copies de l'algorithme de recherche *AntsLS*, puis, au fur et à mesure, le processus maître récupère le résultat de chaque recherche et garde la meilleure solution trouvée.

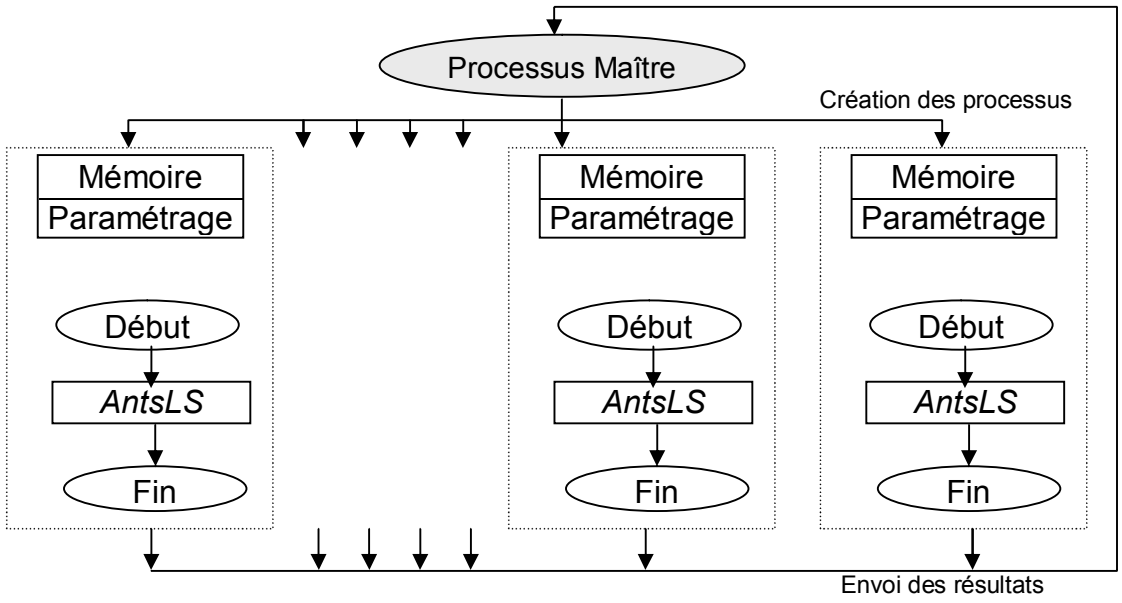


Figure 2.11. Parallélisation multidépart de l'algorithme *AntsLS*.

Pour mesurer l'efficacité de cet algorithme, nous l'avons testé sur les instances du SCP de tailles qui dépassent 500 lignes et 5000 colonnes [Bea 90].
L'algorithme a été exécuté sur un réseau de machines de type PENTIUM II-700MHz. Pour chaque test effectué, nous avons lancé 40 processus sur 40 machines différentes. Pour chaque processus lancé, nous avons pris 20 fourmis artificielles pour effectuer la recherche.
Le tableau 2.9 présente les résultats des tests effectués. La colonne *Tpar* (respectivement *Tseq*) donne le temps total de recherche en secondes de l'algorithme selon l'implémentation parallèle (respectivement séquentielle). Par ailleurs, la colonne *Accélération* donne le rapport entre *Tseq* et *Tpar,* et la colonne *Efficacité* indique le rapport entre l'accélération et le nombre de processus parallèles qui est égal à 40.

Nous constatons (voir tableau 2.9) que la recherche multidépart parallèle augmente sensiblement la qualité des solutions trouvées dans l'algorithme précédent *AntsLS*. En effet, nous avons pu améliorer la qualité des solutions trouvées pour les instances : E3, F5, G2, G3,

G5, et H1. D'autre part, nous avons obtenu la meilleure solution connue dans la littérature pour les instances : E3, F5, et G5. Cependant, l'écart moyen est de 0,74%.
Concernant le temps de recherche parallèle, nous remarquons que l'accélération moyenne, qui est égal à 37,17, est proche du nombre de processeurs utilisés, qui est égal à 40. L'efficacité moyenne de la recherche parallèle est ainsi égale à 92,93%.
Les résultats obtenus s'expliquent par le fait que la probabilité d'atteindre la meilleure solution a été multipliée par 40 (soit le nombre de processus parallèles) et une intensification au niveau de la recherche locale. Par ailleurs, l'efficacité de la recherche parallèle s'explique aussi par un taux de communication très faible entre le processus maître et les processus esclaves. En effet, la communication s'effectue lors de la création des processus esclaves et lors de l'envoi des résultats de recherche par les processus esclaves vers le processus maître.

Instances	*Op/Bsf*	*Ants+LS*	*Best*	*Ecart*	*Tpar (s)*	*Tseq (s)*	*Accélération*	*Efficacité*
E1	29	29	29	0,00%	24,5	891,6	36,36	90,91%
E2	30	30	30	0,00%	24,6	897,2	36,37	90,92%
E3	27	28	**27**	0,00%	24,5	891,6	36,32	90,79%
E4	28	28	28	0,00%	24,5	893,6	36,34	90,85%
E5	28	28	28	0,00%	24,6	893,2	36,29	90,74%
F1	14	14	14	0,00%	24,1	876,4	36,32	90,80%
F2	15	15	15	0,00%	24,1	876,8	36,32	90,80%
F3	14	14	14	0,00%	24,1	876,8	36,29	90,73%
F4	14	14	14	0,00%	24,1	877,2	36,29	90,73%
F5	13	14	**13**	0,00%	24,1	877,2	36,31	90,77%
G1	176	176	176	0,00%	184,1	7329,6	39,80	99,50%
G2	154	157	**155**	0,65%	182,8	7274,8	39,78	99,46%
G3	166	171	**169**	1,81%	101,7	4062,4	39,93	99,81%
G4	168	172	172	2,38%	185,3	7350,4	39,67	99,16%
G5	168	169	**168**	0,00%	101,0	4028,0	39,86	99,64%
H1	63	65	**64**	1,59%	100,3	3623,6	36,13	90,32%
H2	63	64	64	1,59%	100,2	3644,0	36,33	90,84%
H3	59	59	59	1,69%	99,7	3624,8	36,34	90,84%
H4	58	59	59	1,72%	100,0	3619,2	36,19	90,47%
H5	55	55	55	0,00%	99,9	3617,2	36,21	90,52%

Tableau 2.9. Résultats de la recherche parallèle multidépart.

2.7.2. Parallélisation des fourmis

Dans ce type de parallélisation, une seule copie de l'algorithme *AntsLS* est utilisée. En effet, l'algorithme en question est parallélisé en considérant chaque fourmi comme un processus de recherche à part. La phase de construction des solutions par les fourmis est parallélisé en plaçant chaque fourmi sur un processeur indépendant. Le processus maître envoie les informations nécessaires (phéromone) à chaque fourmi. Puis chaque fourmi, indépendamment des autres, construit sa solution.
Ainsi, le processus maître récupère les solutions au fur et à mesure de chaque fourmi artificielle. Lorsque le processus maître a reçu toutes les solutions, il met à jour le taux de phéromone, ainsi que la meilleure solution ; puis relance la recherche des fourmis en leur

renvoyant la nouvelle valeur de la phéromone. Le processus de recherche est arrêté lorsque le nombre d'itérations limite (fixé expérimentalement à 50) est atteint.
La partie entre l'étape 1 et l'étape 3, de l'algorithme de la figure 2.12, représente la boucle de recherche des fourmis artificielles objet de parallélisation. L'implantation parallèle de cet algorithme est représentée dans la figure 2.13 :

Pour chaque test effectué, nous avons considéré 20 fourmis et 20 processeurs. Le tableau 2.10 présente les résultats des tests effectués.

Dans ce type de parallélisation, nous nous sommes intéressés au gain obtenu dans le temps de recherche parallèle. Cependant, nous remarquons que l'accélération moyenne, qui est égal à 10,95 représente la moyenne du nombre de processeurs utilisés qui est égal à 20. Mais, nous remarquons également que la valeur de l'accélération n'est pas uniformément distribuée par rapport aux instances traitées.
Cette valeur augmente proportionnellement avec la taille de l'instance traitée (figure 2.14). Ceci s'explique par le fait que le temps de communication est très élevé par rapport au temps global de traitement dans les instances de petites tailles. Par contre, dans les instances de grandes tailles, le temps de communication est faible par rapport au temps global de recherche.

Instances	*Tpar (s)*	*Tseq (s)*	*Accélération*	*Efficacité*
41	2	15	7,95	39,75%
51	3	26	7,54	37,71%
61	2	12	4,21	21,03%
A1	4	55	11,58	57,92%
B1	4	48	10,26	51,28%
C1	6	100	15,20	75,98%
D1	5	83	14,19	70,93%
E1	7	125	16,69	83,47%
F1	9	122	13,51	67,53%

Tableau 2.10. Résultats de la recherche : fourmis parallèles.

```
Algorithme : AntsLS

Début
Initialisation de la phéromone pour chaque colonne
Etape 1 : Initialiser chaque solution Fourmi_k
Etape 2 : Choisir une colonne selon l'équation (1)
Insérer cette colonne dans Fourmi_k
Si Solution réalisable
   alors
                Eliminer les colonnes redondantes de Fourmi_k
                Appliquer la recherche locale LS (S_k)
                Etape 3 :
                Si K:=k+1 > ants_nb
                  alors
```

```
                    Mise à jour de Fourmi-best
                    Mise à jour de la phéromone selon l'équation (2)
                   Si nb-iter ≥ nbiter-limite
                     alors Fin
                     sinon aller étape 1
                    fin si
                  sinon aller étape 2
                fin si
    sinon aller étape 2
fin si
Fin.
```

Figure 2.12. La partie "fourmis" à paralléliser dans l'algorithme *AntsLS*.

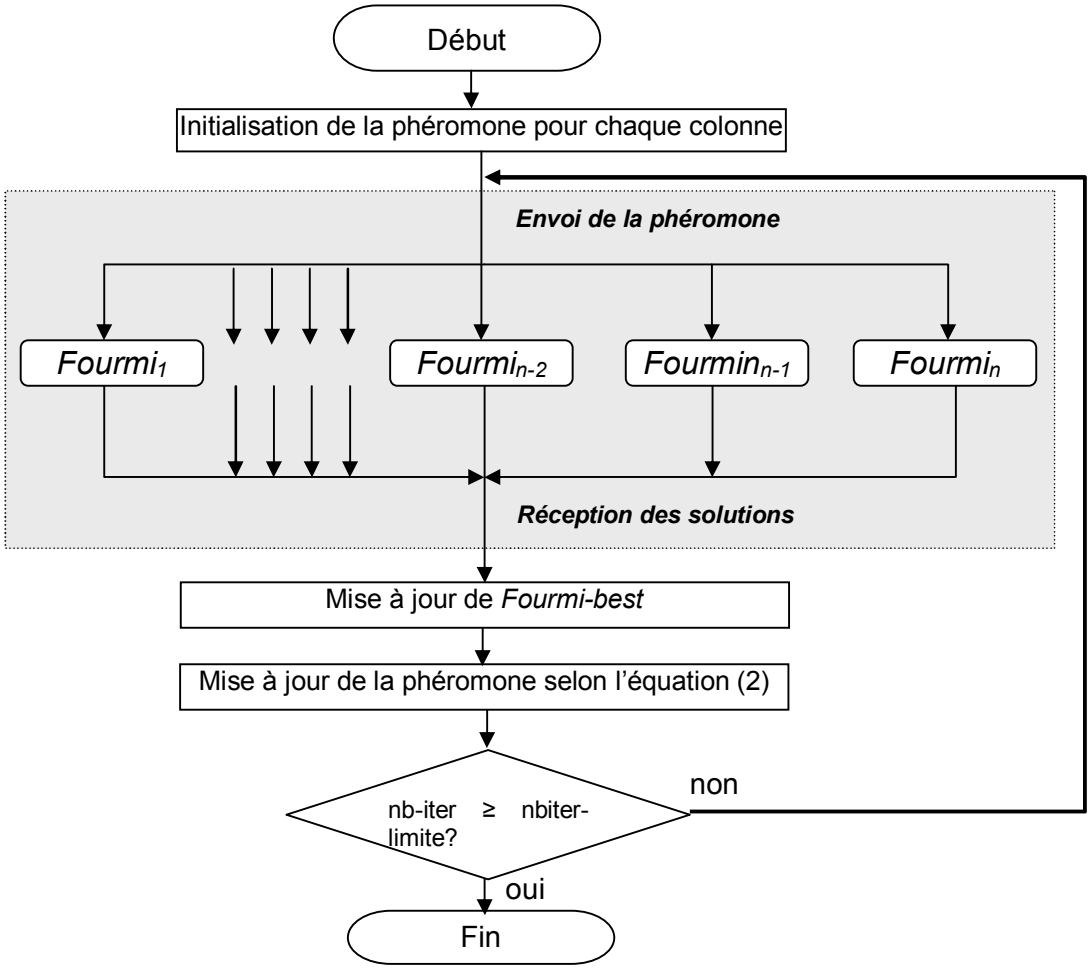


Figure 2.13. Parallélisation des "fourmis" dans l'algorithme *AntsLS*.

Figure 2.14. Accélération par rapport aux problèmes traités.

2.8. Conclusion

Dans ce chapitre, nous avons adapté la métaheuristique des colonies de fourmis pour le problème de couverture d'ensembles. Après la détermination des différents constituants de base de cette métaheuristique, nous avons proposé un algorithme glouton basé sur deux variantes du critère heuristique utilisé dans la règle de transition d'une fourmi. Des tests expérimentaux nous ont permis de déterminer la meilleure variante, à savoir celle basée sur le rapport de la couverture d'une colonne sur son coût. Un autre choix s'est imposé quant à la manière d'éliminer les colonnes redondantes. En effet, on peut soit les éliminer en cours de la construction d'une solution, soit à la fin. Les tests réalisés ont permis d'opter pour la deuxième solution.

L'algorithme que nous avons proposé *Ants* a ensuite été combiné avec une heuristique de recherche locale dans le but d'améliorer les résultats.
Deux types d'hybridations ont été mis en œuvre : l'hybridation *HRH*, ainsi que l'hybridation *HCH*. Les tests effectués ont montré que l'hybridation *HCH* est plus efficace que l'hybridation *HRH*.

La qualité des solutions obtenues a été améliorée en utilisant différentes implémentations parallèles de l'algorithme hybride le plus performant, *AntsLS*. Cette amélioration s'explique par le fait que la recherche locale a été appliquée sur chaque itération des fourmis (intense phase d'intensification).
Le modèle de parallélisation multidépart semble être plus performant que celui de la parallélisation des fourmis. L'efficacité du premier modèle s'explique par un taux de communication très faible entre le processus maître et les processus esclaves. En effet, la communication s'effectue lors de la création des processus esclaves et lors de l'envoi des résultats de recherche par les processus esclaves vers le processus maître.
Dans le second modèle, nous avons remarqué que l'efficacité de l'algorithme parallèle des fourmis dépend de la taille du problème traité. En effet, l'accélération augmente de manière proportionnelle avec la taille de l'instance traitée. Cette augmentation s'explique par le fait que le temps de communication est très élevé par rapport au temps global de traitement dans

les instances de petites tailles. Alors que, dans les instances de grande taille, le temps de communication est faible par rapport au temps global de recherche.

Chapitre 3

Programmation par contraintes et colonie de fourmis pour le problème du repliement de protéines

Le problème du repliement de protéines, qui est largement abordé dans la littérature, consiste, étant donnée une protéine, à trouver un repliement de moindre énergie. Ce problème est NP-difficile pour le modèle H-P que nous considérons par la suite. Notre approche de résolution pour ce problème consiste à proposer des modèles de coopération/hybridation entre des métaheuristiques et la programmation par contraintes. Ce chapitre présente, plusieurs algorithmes : deux modèles de programmation par contraintes, une recherche locale, un algorithme de colonies de fourmis et plusieurs combinaisons de coopération de ces différents algorithmes.

Les travaux décrits dans ce chapitre ont fait l'objet d'une présentation à la conférence MIC 2005 [Ang 05]. Dans un cadre plus général, ce travail a aussi été présenté dans un tutoriel [Rah 05].

3.1. Introduction

Il est généralement admis que les différentes méthodes d'optimisation possèdent chacune des avantages et des inconvénients, des forces et des faiblesses. Ainsi, de plus en plus d'études portent sur la coopération entres différents paradigmes d'optimisation [Tal 02]. En effet, pour rendre les algorithmes plus performants et fiables, de nombreux travaux proposent de combiner différentes heuristiques et/ou méthodes exactes. Ainsi, depuis quelques années, de nombreuses approches coopératives ont vu le jour dans la littérature. Nous nous intéressons, au travers de ce chapitre, à la collaboration/hybridation de métaheuristiques avec des méthodes exactes de type "programmation par contraintes". Parmi les méthodes exactes, la programmation par contraintes [Bar 98] s'est révélée être efficace pour résoudre de manière exacte de nombreux problèmes d'optimisation combinatoire, comme par exemple : des problèmes d'allocation de ressources, d'ordonnancement de tâches, de rotation de personnels, de gestion d'emplois du temps, des problèmes de transport, … [Bar 98][Bap 01]. Cette méthode exacte, ainsi que toutes les autres d'ailleurs, comme celles basées sur le concept d'énumération [Law 66][Nem 88][Pap 98] ou l'utilisation de la programmation linéaire en nombres entiers [Chv 83][Pap 98][Sch 98], est néanmoins limitée par la taille des problèmes et les temps de calcul, qui peuvent être prohibitifs. A l'opposé, les métaheuristiques fournissent en général rapidement des solutions approchées de bonne qualité. Plusieurs moyens peuvent être envisagés pour tirer parti du meilleur de ces deux mondes. Le critère auquel on s'intéresse est la performance : idéalement, on voudrait montrer que la collaboration/hybridation de méthodes permet d'améliorer significativement la qualité des solutions obtenues et de traiter des problèmes de grande taille. Nous avons abordé cette question pour un problème de bioinformatique bien connu, qui est celui du *repliement des protéines*.

Une molécule est, de façon générale, un agencement de divers éléments chimiques. Ces éléments interagissent entre eux ainsi qu'avec l'environnement, suivant plusieurs critères, pour donner une conformation spatiale à cette molécule. Une catégorie particulière de molécules, les protéines, possèdent des propriétés physiques et chimiques spécifiques à leur conformation spatiale.
Une protéine peut être vue comme un enchaînement d'acides aminés à caractère hydrophobe ou hydrophile. Une protéine a tendance à se replier sur elle-même, à l'image d'un ressort trop tendu. Néanmoins, une protéine ne peut pas se replier n'importe comment : elle doit atteindre un niveau d'énergie le plus bas possible. Déterminer la conformation spatiale de la protéine dans cet état de moindre énergie est connu sous l'appellation de *problème du repliement de protéines*. Celui-ci peut être modélisé comme un problème d'optimisation combinatoire. Son étude et sa résolution revêtent un intérêt dans différents domaines, comme la santé humaine et la biotechnologie. Dans le domaine de la santé humaine, savoir résoudre ce problème de manière exacte pourrait contribuer à la prédiction et au traitement des maladies d'Alzheimer, de Parkinson, de Creutzfeld-Jacob. En biotechnologie, il pourrait permettre la réalisation de procédés enzymatiques à grande échelle, le développement de médicaments protéiques (insuline, HGH) …

Le problème du repliement de protéines est NP-difficile pour le modèle H-P que nous considérons par la suite [Dil 95][Ber 98][Cre 98]. De ce fait, il n'existe pas d'algorithme en temps polynomial pouvant déterminer une conformation de plus basse énergie pour une protéine. L'objectif de ce travail est l'étude de plusieurs méthodes de résolution approchées de ce problème. Inspirés des travaux de [Bac 01][Les 03], des algorithmes de coopération entre

la programmation par contraintes, la recherche locale et les colonies de fourmis sont combinés. Nous avons ainsi utilisé :

- la programmation par contraintes, qui est une méthode de résolution exacte,
- les colonies de fourmis,
- la recherche locale en utilisant le voisinage "mouvement d'attraction" (*pull move*) [Les 03] qui consiste à rapprocher les acides aminés d'une manière particulière,
- et nous avons proposé des schémas de coopération entre ces différentes méthodes.

La section 3.2 de ce chapitre introduit le problème du repliement des protéines. La section 3.3 donne un aperçu sur les approches de résolution existantes pour ce problème. La section 3.4 explicite comment la programmation par contraintes aborde ce problème sous l'environnement de programmation Mozart-Oz [Moz 04]. A cet effet, deux modèles de programmation par contraintes sont proposés. Le premier transcrit directement et simplement les contraintes de base du problème (*PPC de base*). Tandis que le second, plus élaboré, ajoute des contraintes spécifiques au problème traité (*PPC étendue*). Dans la section 3.5, un algorithme de recherche locale est présenté. Quant à la section 3.6, elle propose un algorithme de colonies de fourmis. La section 3.7 expose plusieurs combinaisons de coopération/hybridation entre la programmation par contraintes, les colonies de fourmis et la recherche locale, selon deux motivations. La première motivation est de fournir un algorithme, qui se veut générique, entre la programmation par contraintes et un algorithme de colonies de fourmis simple. Quant à la seconde motivation, elle vise la performance en donnant des algorithmes dédiés au problème traité. Enfin, la dernière section est consacrée aux différents tests menés, sur des benchmarks de la littérature [Dil 95], et à l'analyse des résultats obtenus.

3.2. Le problème de repliement de protéines

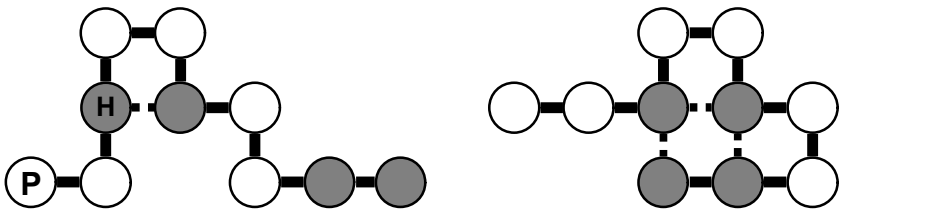
Habituellement, une protéine possède quelques centaines d'acides aminés [Dil 95][Cre 98]. Selon les conditions de son environnement, une protéine peut se trouver dans un état dénaturé (dépliée) ou natif (pliée).
Les interactions entre la protéine et l'eau jouent un rôle important dans ce phénomène de repliement. Dans l'état dénaturé, un grand nombre de monomères hydrophobes sont exposés au milieu aqueux, ce qui confère à la protéine un état énergétique élevé. Par contre, dans l'état natif, les régions hydrophobes ont tendance à se concentrer au cœur de la molécule, minimisant ainsi la surface exposée à l'eau et l'état énergétique de la protéine. La conformation la plus stable est celle qui requiert le moins d'énergie. Il est utile de trouver cette conformation native car c'est probablement la conformation naturelle de la protéine et la plus stable. Cette conformation donne des indications sur la fonction de celle-ci dans l'organisme.

Il existe deux méthodes pour déterminer la conformation native d'une protéine : la première (et la meilleure) serait de "photographier" aux rayons X la protéine dans une solution d'eau pure. L'inconvénient de cette méthode vient du coût de la technique.
La seconde technique, celle qui nous intéresse, est de modéliser ce problème comme un problème d'optimisation combinatoire et de le résoudre de manière exacte ou approchée en utilisant des algorithmes appropriés. Le problème étant complexe, de nombreux modèles simplifiés existent, comme le modèle HP proposé par Dill [Dil 95]. Ce modèle possède les caractéristiques suivantes :

1. Les acides aminés (monomères) ont tous la même taille.

2. Les acides aminés sont simplifiés. Il n'y en a plus que 2 types : les monomères hydrophobes, notés H par la suite, et les monomères hydrophiles ou polaires, notés P par la suite.
3. Les liens entre 2 monomères ont tous la même longueur.
4. Chaque monomère ne peut occuper une position que sur une grille en deux ou trois dimensions.
5. La fonction calculant l'énergie d'une protéine dans une conformation donnée, est décrite ainsi. Soient *a* et *b* deux monomères de type H non consécutifs de la protéine et *f* la fonction telle que *f* (*a*, *b*) = -1 si *a* et *b* sont voisins dans la conformation, et *f* (*a*, *b*) = 0 sinon. L'énergie de la protéine dans cette conformation est égale à la somme des *f(a,b)* sur tous les couples *a* et *b* possibles de monomères non consécutifs de la protéine. Le problème consiste donc à chercher une conformation de moindre énergie, ce qui revient à maximiser le nombre de contacts H-H.

La figure 3.1 montre deux conformations possibles pour la séquence PPHPPHPPHH. La conformation de moindre énergie est celle qui maximise le nombre de contacts H-H. A noter que, tout au long de ce chapitre, les monomères H seront représentés par un disque foncé et les monomères P par un disque blanc.



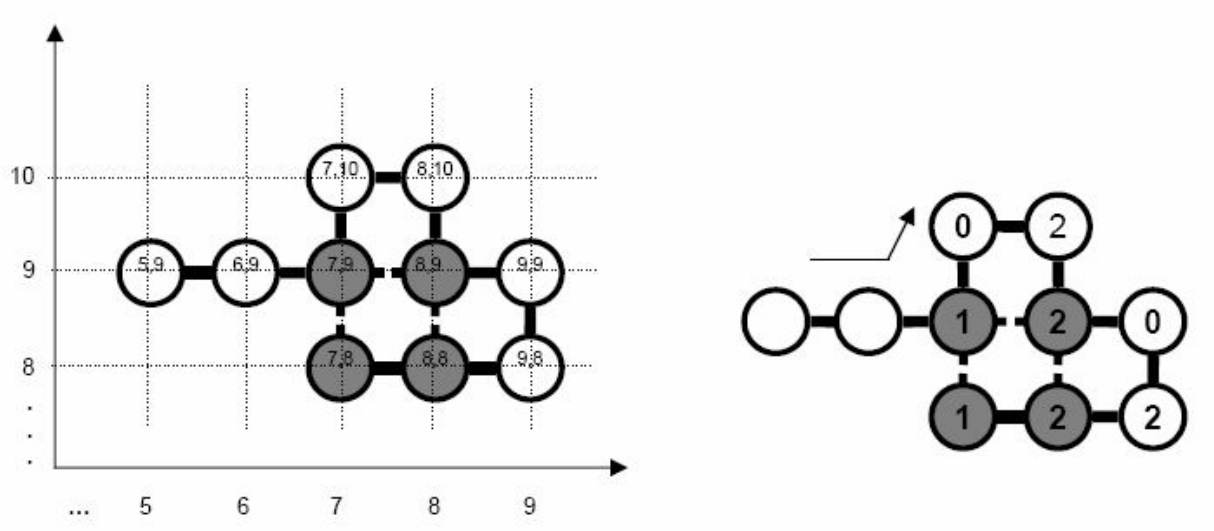
Conformation non optimale : 1 seul contact H-H conformation optimale : 3 contacts H-H

Figure 3.1. Deux exemples de repliement.

3.3. Approches de résolution du problème de repliement de protéines

De nombreuses méthodes de résolution qui abordent ce problème sont basées sur le modèle HP. Parmi celles-ci, on peut citer des méthodes approchées comme les algorithmes génétiques [Ung 93][Kra 99], la méthode de Monte-Carlo [Ung 93], Monte-Carlo evolutionnaire [Lia 01], le recuit simulé, les colonies de fourmis [Shm 02][Shm 03][Shm 05], la recherche tabou [Jia 03] et, entre autres, parmi les méthodes exactes, la programmation par contraintes [Bac 01]. Une approche par la programmation linéaire est présentée dans [Gre 04].

Il y a deux manières principales de représenter la conformation d'une protéine. La première consiste à utiliser des coordonnées absolues, autrement dit les abscisses et les ordonnées de chaque monomère de la séquence, dans le cas d'un repliement à deux dimensions (figure 3.2 (a)).
La deuxième utilise des coordonnées relatives, c'est-à-dire les positions des monomères par rapport aux monomères précédents dans la séquence. Dans ce cas, 3 possibilités d'orientation sont envisageables : "0" pour gauche, "1" pour tout droit et "2" pour droite (figure 3.2 (b)). A chaque fois qu'on décide de placer un monomère, on considère les deux qui le précèdent et on décide de son orientation (figure 3.2 (b)).



(a) Représentation en coordonnées absolues.

(b) Représentation en coordonnées relatives
0 : gauche, 1 : tout droit et 2 : droite.

Figure 3.2. Deux représentations d'une même conformation.

Les algorithmes de type génétique ou la métaheuristique des colonies de fourmis utilisent la deuxième représentation, qui leur est bien adaptée [Shm 02][Shm 03][Shm 05] mais les algorithmes de type exact [Cha 03][Xu 03] emploient des coordonnées absolues. Par la suite, nous serons amenés à utiliser simultanément les deux types de représentations dans les algorithmes hybrides proposés.

3.4. Résolution du problème du repliement des protéines par la programmation par contraintes

3.4.1. Généralités sur la programmation par contraintes

La programmation par contraintes est une technique de résolution de problèmes d'optimisation complexes [Bar 98][Lus 01]. Elle se situe au carrefour de nombreuses disciplines : intelligence artificielle, recherche opérationnelle,... L'efficacité de la programmation par contraintes tient au fait qu'elle permet de dissocier la représentation du problème (définition des contraintes et des objectifs) de sa résolution, réalisée par le système. Ce système, appelé "solveur", a la capacité d'énumérer l'ensemble des solutions de manière implicite et efficace, grâce à différentes techniques, comme par exemple la propagation de contraintes, qui a pour conséquence de réduire de manière importante l'espace de recherche.

La modélisation d'un problème pour la programmation par contraintes revient à introduire un ensemble de *variables* de décisions $x_1, x_2,..., x_n$ avec, pour chaque variable, le *domaine* correspondant D_i des valeurs qu'elle peut prendre. Des *contraintes* $(C_j)_{j=1..m}$ sont associées au problème et restreignent les valeurs que peuvent prendre ces variables.
Une contrainte C_j faisant intervenir les variables $x_1, x_2, \ldots, x_n$ peut ainsi être vue comme un sous-ensemble S de l'ensemble $D_1 x D_2 x \ldots x D_n$ indiquant les valeurs que peuvent prendre ces variables. La contrainte C_j est satisfaite si et seulement si $(x_1, x_2, \ldots, x_n) \in S$.

Une instance d'un tel problème de satisfaction de contraintes est représentée par un triplet (*V; D; C*). Une solution est une affectation complète des variables $V=\{x_1, x_2,..., x_n\}$ (c'est-à-dire l'attribution, à chaque variable, d'une valeur unique, dans son domaine D_i) telle que toutes les contraintes $C=\{C_1, C_2, \ldots \ldots\}$ soient satisfaites.

Les méthodes de résolution pour ce genre de problème s'appuient sur la *propagation* et la *distribution* des contraintes.

En pratique, des techniques de *propagation de contraintes* permettent à chaque contrainte de "jouer un rôle" dans la résolution du problème : chaque fois que le domaine (i.e., l'ensemble des valeurs possibles) d'une des variables impliquées dans la contrainte est réduit, cette réduction est propagée, de manière à supprimer des domaines des autres variables les valeurs qui ne peuvent plus apparaître dans une solution respectant la contrainte. Il s'agit en fait d'une réduction sophistiquée des domaines des variables pour éliminer les portions de l'espace de recherche ne pouvant faire partie d'une solution.

L'exemple de la figure 3.3 illustre cette technique. Soient deux variables entières $x \in D_x=[1..10]$ et $y \in D_y=[1..10]$. Soient deux propagateurs P1 et P2 qui imposent les contraintes P1: $y=2*x$ et P2: (x modulo 2)=1. L'application du propagateur P1 permet de réduire D_x à [1..5] et D_y à {2,4,6,8,10}, puis P2 réduit D_x à {1,3,5} et enfin P1 réduit D_y à {2,6,10}.

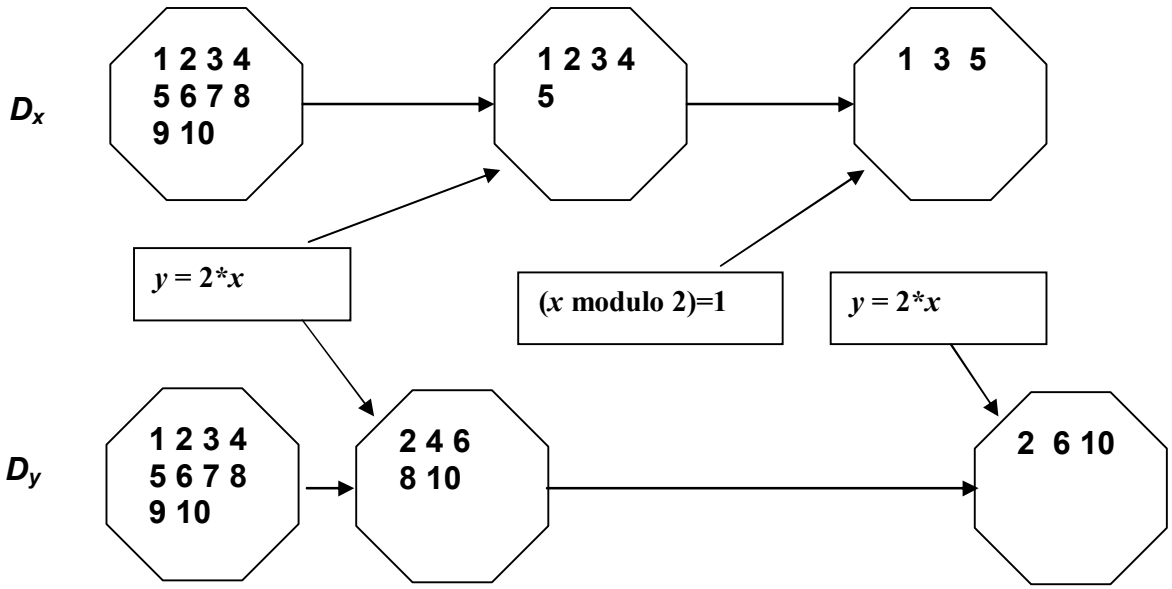


Figure 3.3. Exemple de propagation de contraintes.

Quant à la technique de *distribution*, elle consiste, lorsque la propagation des contraintes n'est pas suffisante pour arriver à une solution, à choisir une variable et à faire la distribution selon cette variable.

Le choix d'une variable peut être fait selon plusieurs stratégies. Par exemple, ce qui est souvent utilisé c'est de choisir la variable qui a le plus petit domaine.

L'exemple de la figure 3.4, explicite la notion de distribution.

Soient 3 variables entières telles que $x \in [1..8]$, $y \in [1..8]$ et $z \in [1..8]$ et qui doivent satisfaire les contraintes suivantes :

$$x \neq 7, z \neq 7 \text{ et } x-z = 3.y$$

- Après l'étape de propagation de contraintes, les domaines sont réduits à :

$$x \in \{4, 5, 6, 8\}, \quad y \in \{1, 2\} \text{ et } z \in \{1, 2, 3, 5\}.$$

- Si, par exemple nous choisissons de distribuer selon la variable x, avec la contrainte x=4, nous obtenons l'arbre de la figure 3.4 :

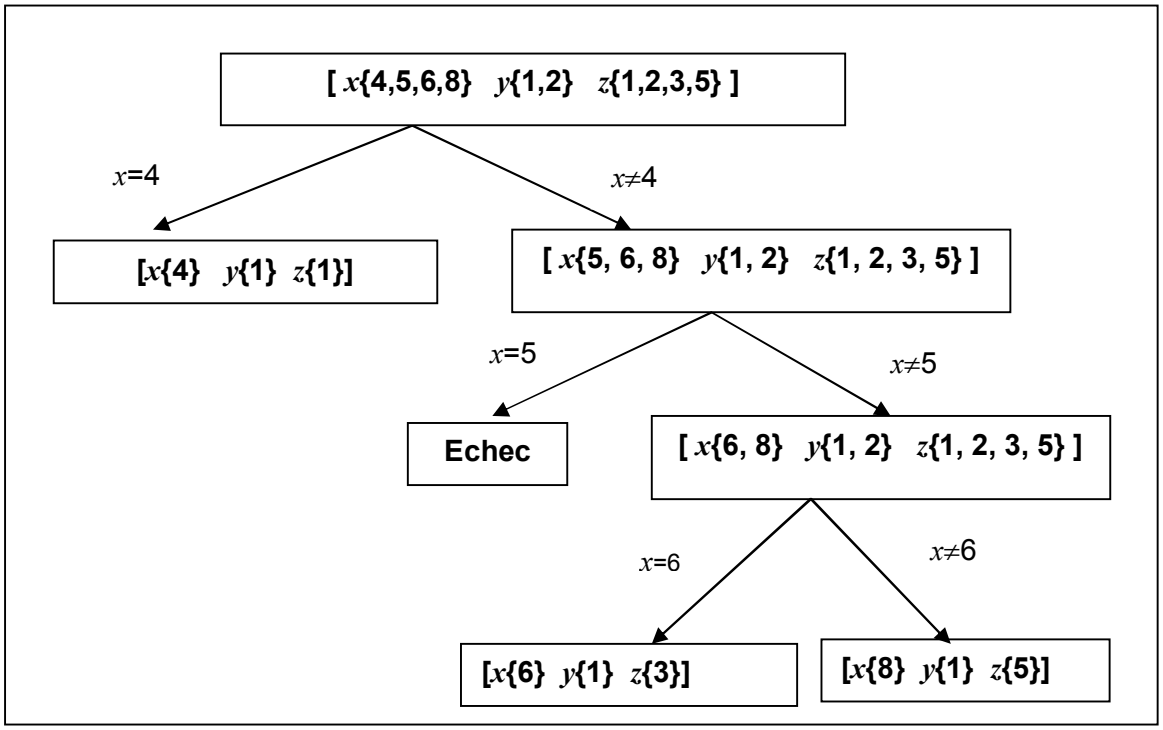


Figure 3.4. Arbre de recherche contenant 3 nœuds de décision, 1 nœud d'échec, et 3 noeuds de solution.

Le solveur réalise le contrôle de l'entrelacement des phases de propagation et de distribution (voir figure 3.5).
La méthodologie de résolution procède par la construction d'un arbre de recherche (voir la figure 3.6), dont chaque noeud correspond à un *espace* (variable et contraintes) obtenu après propagation des contraintes, et chaque branche correspond à une étape de distribution.

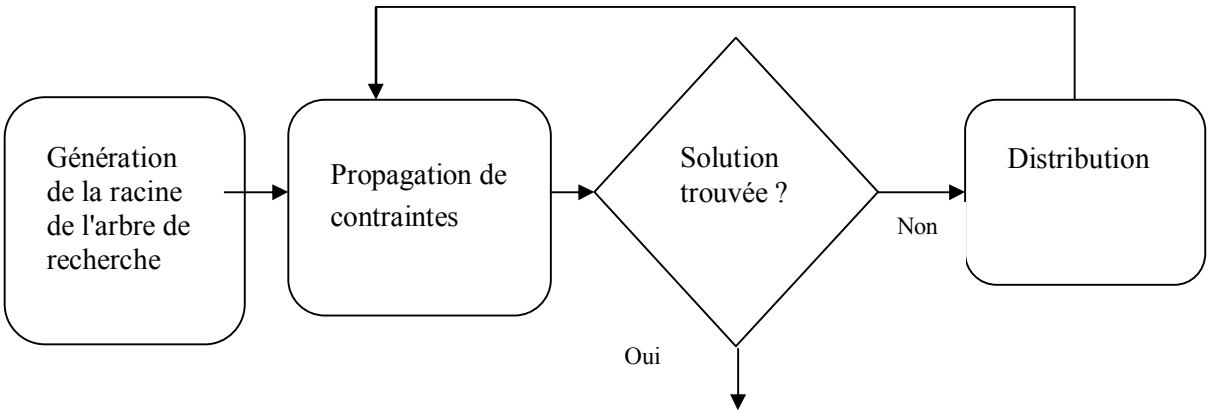


Figure 3.5. Procédure de propagation et de distribution.

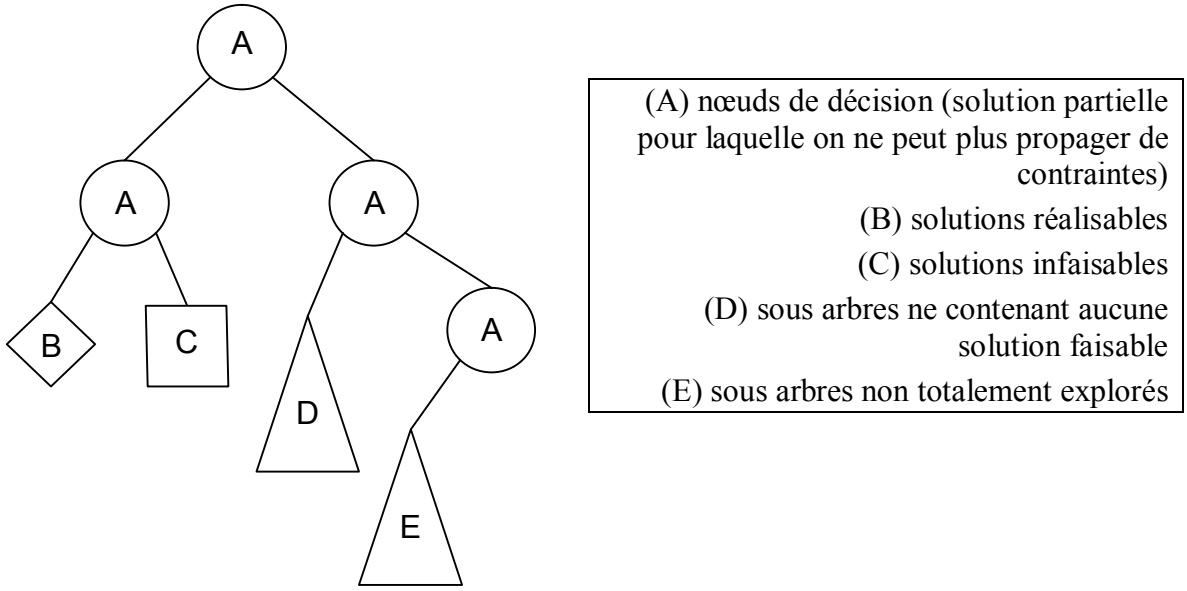


Figure 3.6. Arbre de distribution et de recherche.

Différentes stratégies d'exploration de l'arbre de recherche sont possibles : en profondeur, en largeur, hybride, ou autre. Ces stratégies ont chacune des avantages et des inconvénients. C'est l'exploration en profondeur qui est en général appliquée.

3.4.2. Programmation par contraintes pour les problèmes d'optimisation

Toutes les notions présentées sur la programmation par contraintes et que nous avons illustrées par des exemples (figures 3.3, 3.4, 3.5 et 3.6) ne prenaient pas en compte la qualité de la solution trouvée. Or, dans un problème d'optimisation combinatoire, il faut trouver une solution qui non seulement satisfait toutes les contraintes, mais optimise également une certaine fonction objectif. L'utilisation de la programmation par contraintes pour un problème d'optimisation est semblable à la méthode par *Séparation & Evaluation* utilisée dans la programmation linéaire en nombres entiers. Pour que l'exploration de l'arbre de recherche soit efficace, il est nécessaire d'élaguer des branches.
Lorsqu'une solution de coût *v* est trouvée (feuille de l'arbre de recherche), une contrainte est ajoutée au problème spécifiant que toute solution de valeur *v*, ou plus, n'est pas valide. Ce processus est répété jusqu'à ce qu'on ne puisse plus trouver de meilleure solution, et dans ce cas, la dernière solution est la solution optimale. Généralement, cette contrainte est prise automatiquement en charge par le système de programmation par contraintes (c'est le cas dans Mozart-Oz). Une autre manière d'élaguer des branches de l'arbre de recherche de solutions est la détermination de bornes.
Une autre technique pour élaguer des branches consiste à appliquer une recherche locale sur chaque feuille (solution) de l'arbre de recherche. On obtient ainsi une nouvelle solution de meilleure qualité, dont le coût peut servir de borne.

Ces notions de base sont, dans la suite, mises en œuvre au travers d'algorithmes de résolution du problème du repliement de protéines. Afin d'évaluer la performance de ces algorithmes, nous utilisons le système *Mozart-Oz* [Moz 04].

Mozart est une plateforme de développement avancée pour des applications intelligentes et réparties. Ce système est le résultat d'une décennie de recherche dans les domaines des langages de programmation, de l'inférence à base de contraintes, de l'informatique répartie, et des interfaces homme-machine. Le système Mozart utilise le langage de programmation *OZ 4*. Ce dernier est un langage multi-paradigme (possibilité d'employer différents modèles de programmation dans le même programme). Il fournit, entre autres, les dispositifs nécessaires à la programmation orientée objet, au calcul distribué et à la *programmation par contraintes*, notre centre d'intérêt.
Dans le domaine de la programmation par contraintes, plusieurs systèmes de développement existent, parmi lesquels *Ilog Solver* qui a été utilisé, avec succès, pour la résolution de plusieurs problèmes industriels (gestion des aéroports de Paris, planning de personnels, ordonnancement, et autres). Ce système, certes performant, reste inaccessible de par son prix alors que le système *Mozart* est téléchargeable gratuitement pour des plateformes Unix ou Windows. C'est pour cette raison que notre choix s'est porté sur ce système, en plus du fait qu'il est très bien documenté.

3.4.3. Programmation par contraintes : version de base

Soit $s \in \{H,P\}^n$ une séquence d'acides aminés de longueur n représentant une protéine dans le modèle HP. On note s_i le i-ième monomère de la séquence s, et on note $\mathcal{H}=\{1\leq i\leq n / s_i = H\}$ l'ensemble des indices des monomères H.
Soit c une conformation de la protéine s, on note $c_i \in ZxZ$ la position du monomère s_i dans cette conformation. Plus précisément, on se place dans le modèle en 2 dimensions et $c_i=(c_{i1},c_{i2})$ indique donc l'abscisse (respectivement l'ordonnée) c_{i1} (respectivement c_{i2}) du i-ième monomère s_i dans la conformation de la protéine s.
Une conformation n'est valide que si elle respecte les deux familles de contraintes suivantes :

- (C1) deux acides aminés consécutifs dans une séquence doivent être voisins dans la grille,
- (C2) deux monomères ne peuvent se chevaucher, c'est-à-dire occuper le même emplacement sur la grille.

A partir d'une conformation c d'une protéine s, on définit le nombre de contacts $Contact_s(c)$ de la manière suivante.
Pour chaque paire $\{i,j\}$ avec $i,j \in \mathcal{H}$, on dit qu'il y a "contact" entre les acides aminés s_i=H et s_j=H, si s_i et s_j sont voisins dans la conformation, mais pas dans la séquence. Le nombre de contacts H-H de la protéine s dans la conformation c, noté $Contact_s(c)$ est alors défini en faisant la somme du nombre de contacts entre les monomères s_i et s_j pour tous les i, j tel que i, $j \in \mathcal{H}$.

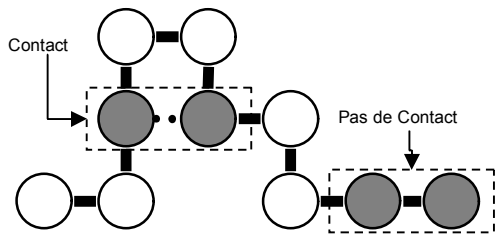


Figure 3.7. Contacts H-H.

Notons || . || la norme euclidienne dans ZxZ, c'est-à-dire $||(c_{i1},c_{i2})||= |c_{i1}|+|c_{i2}|$.

De manière formelle, la première série de contraintes (C1) peut s'écrire :

(C1) $\forall i,\ 1 \leq i < n,\ || c_{i+1} - c_i || = 1$.

La deuxième série de contraintes (C2) peut s'écrire :

(C2) $\forall i,j\ ,\quad 1 \leq i < j \leq n,\ c_i \neq c_j$.

Le nombre de contacts $Contact_s(c)$ est égal à :

$$\sum_{\substack{(i,j)\in \mathcal{H}^2 \\ j>i+2}} \delta_1 (|| c_i - c_j ||) \quad \text{avec} \quad \delta_i (j) = 1 \text{ si } i=j \text{ et 0 sinon.}$$

L'énergie d'une conformation c vaut $-Contact_s(c)$.

L'écriture de ces contraintes en OZ nécessite la définition des variables suivantes :

X_i , Y_i	Les cordonnées du *i-ième* monomère s_i de la protéine. $X_i \in [1\ldots n]$, $Y_i \in [1\ldots n]$,
$XDiff_{i,j}$, $YDiff_{i,j}$	La distance entre les monomères s_i et s_j, suivant l'axe des abscisses x (respectivement l'axe des ordonnées y). $XDiff_{i,j} = \| X_i - X_j \|$, $YDiff_{i,j} = \| Y_i - Y_j \|$ *avec* $i+1< j$ *et* $i \in [1...n]$, $j \in [2...n]$.
$Contact_{ij}$	S'il y a un contact entre les monomères s_i et s_j avec $i, j \in \mathcal{H}$ et $j>i+2$ cette valeur est égale à 1, et à 0 sinon. $Contact_{ij} \in \{0,1\}$
Energie	- (moins) la somme des contacts H-H.

Les contraintes ci-dessus peuvent alors s'écrire en OZ comme suit :

(C1) $\forall i,\ 1 \leq i < n$ $\qquad XDiff_{i,i+1} + YDiff_{i,i+1} = 1$

(C2) $\forall i,j\ /\ i \neq j$ et $1 \leq i, j \leq n$ $\qquad XDiff_{i,j} + YDiff_{i,j} \neq 0$

Pour des raisons d'efficacité dans la propagation des contraintes, les contraintes (C2) n'ont pas été implémentées à l'aide des variables *XDiff* et *YDiff*. Nous introduisons des variables P_i indiquant la position de chaque monomère s_i. Nous avons $P_i \in [1\ldots n^2]$ avec $P_i = n(Y_i - 1) + X_i$. L'ensemble des contraintes (C2) peut alors s'écrire sous la forme $\forall i,j$, $i \neq j$, $P_i \neq P_j$. Autrement dit, tous les P_i doivent être distincts deux à deux. Ces contraintes se formulent sous *Mozart-Oz* à l'aide de la contrainte : *Alldifferent*(P_1,P_2, ….,P_n). L'avantage de la contrainte *Alldifferent* est qu'il s'agit d'une contrainte "globale" faisant intervenir l'ensemble des variables P_1, P_2, …, P_n. L'étape de propagation des contraintes s'effectue de manière plus efficace avec ce genre de contraintes par rapport à des contraintes faisant intervenir des paires de variables.

La fonction objectif est : $Energie = - \sum_{i,j \in \mathcal{H}^2 \text{ et } i+1<j} Contact_{i,j}$, avec

$$Contact_{ij}=1 \Leftrightarrow XDiff_{i,j} + YDiff_{i,j}=1 \qquad avec \quad i,j \in \mathcal{H}$$

3.4.4. Programmation par contraintes : version étendue

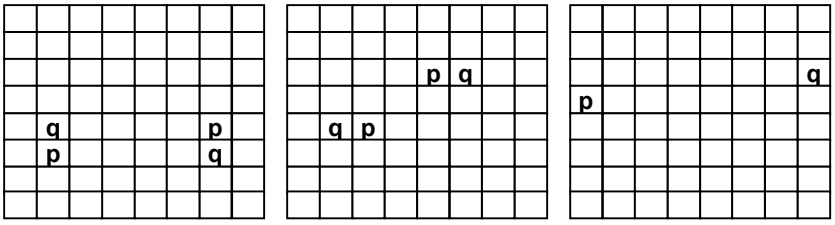
Pour avoir une méthode de résolution plus efficace, il faut ajouter des contraintes pour éliminer le plus de conformations inutiles. Pour cela, nous avons besoin de définir de nouveaux éléments, nécessaires pour l'écriture de contraintes additionnelles. Parmi ces éléments : la notion de *position voisine*, qui servira à définir la surface d'une conformation et ainsi faire le lien entre cette valeur et le nombre de contacts H-H (énergie), la définition des *P-singletons*, la notion de *cavité* dans une séquence donnée et la notion de "*cadre*" (*Frame*).

3.4.4.1. Définition de la notion de voisin

On définit désormais la grille G dans laquelle on représentera la conformation c ainsi : $G \in \{1,\ldots, n^2\}$. On rappelle qu'un monomère s_i occupe la position $P_i=n(Y_i-1)+X_i$. Soit (P_i, Q_i) deux positions de la grille G. Elles sont voisines si (voir figure 3.8) :

Cas 1 : $| P_i - Q_i | = n$ (voir figure 3.8a);

Cas 2 : $| P_i - Q_i | = 1$ sauf si $(P_i + Q_i)$ *modulo* $n = 1$ (voir figures 3.8b et 3.8c)



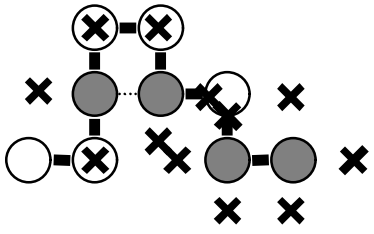
(a) oui (b) oui (c) non

Figure 3.8. Positions possibles entre monomères voisins.

3.4.4.2. Définition de la surface d'une conformation

On définit G_H l'ensemble des positions des monomères H sur la grille.
On définit la surface $Surf_s(c)$ comme le nombre de couples de positions *voisines* telles que la première position du couple est occupée par un monomère H, mais pas la seconde :

$$Surf_s(c) = |\{(p,q) / s_p=\text{H} , s_q\neq\text{H} \text{ et } s_p \text{ et } s_q \text{ sont voisins dans la grille}\}|$$



La surface est 12

Figure 3.9. Exemple de calcul de la surface d'une conformation.

Yue et Dill [Yue 93] ont observé qu'il y a une relation simple entre l'énergie et la surface. On note n_H le nombre de monomères H, et *HHBonds(s)* le nombre de liens physiques entre deux acides H. C'est-à-dire le nombre de fois où l'on a deux acides aminés H consécutifs dans la séquence. Ces valeurs sont des constantes pour une séquence donnée. On a alors la formule suivante :

$$4*n_H = 2*[Contact_s(c) + HHBonds(s)] + Surf_s(c)$$

D'après cette formule, on voit que maximiser le nombre de contacts, revient à minimiser la surface de la protéine.
Pour l'exemple de la figure 3.9, on a : n_H= 4, *HHBonds*=1, $Contact_s$=1 d'où $Surf_s$=16 – 4 = 12.

3.4.4.3. Définition du "cadre"

Si, dans la séquence de la protéine, se trouve un monomère P entouré de 2 monomères H, alors celui-ci est appelé *P-singleton*. On définit ainsi l'ensemble $P_{singleton}$ comme l'ensemble des indices de ces monomères.
Etant donnée une conformation *c*, une position $p \in G$ est appelée *cavité* (figure 3.10) si :

1. s_p =P
2. s_q =H et s_r =H où s_q *et* s_r sont 2 voisins de s_p sur l'un des axes de la grille.

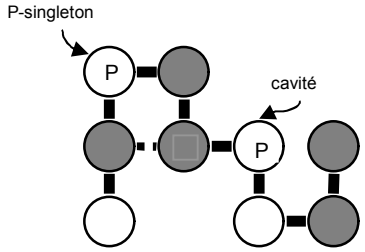


Figure 3.10. Exemple de P-singleton et de cavité.

Ce que l'on appelle "cadre" par la suite est une zone rectangulaire de la grille, dans laquelle doivent être présents tous les monomères H de la séquence. La présence d'un "cadre" restreint donc les conformations que peut prendre la séquence. Considérons par exemple la séquence HHPHHPHHPP. La figure 3.11 donne un exemple de "cadre" et de conformation pour cette séquence. De par la topologie de la grille en deux dimensions, il est facile de voir que, si tous les monomères H d'une séquence sont présents dans le "cadre", alors il en est de même de tous les P-singletons.

Si les dimensions du "cadre" sont correctement définies, alors, parmi toutes les conformations possibles en accord avec ce "cadre", il en existera une proche de l'optimum global. Un "cadre" permet donc de réduire l'ensemble des conformations à tester et accélère ainsi la recherche d'une bonne solution. Cette idée a, pour la première fois, été utilisée par Yue et Dill [Yue 93]. Ces auteurs proposèrent un algorithme exact qui utilise une méthode d'énumération. Cet algorithme énumère les différentes dimensions possibles du "cadre" et énumère ensuite les différentes conformations qui sont conformes avec le "cadre". Backofen [Bac 01] a repris cette idée, mais dans le contexte de la programmation par contrainte.

Notre approche est voisine de celle de Backofen [Bac 01], sauf que nous incorporons cette notion de "cadre" également dans l'algorithme de colonies de fourmis que nous avons proposé et dans l'approche de coopération de cette métaheuristique avec la programmation par contraintes. Dans notre algorithme, le dimensionnement du "cadre" est fait d'une manière empirique alors que Backofen utilise une méthode d'énumération. Dans notre approche, le réglage empirique se fait de manière progressive, en commençant par de petites valeurs de la hauteur et de la largeur du "cadre". Ces valeurs sont augmentées progressivement en fonction des résultats obtenus et en tenant compte du fait que le produit de la largeur du "cadre" par sa hauteur doit être inférieur ou égal à la taille de la séquence.
Une autre différence entre notre travail et celui de Backofen réside dans la manière dont nous avons implémenté les contraintes (C1) qui interdisent les chevauchements de deux monomères dans la grille. Backofen utilise des comparaisons deux à deux des coordonnées absolues de chaque monomère, alors que nous travaillons avec les positions P_i de chacun des monomères s_i. Les contraintes (C1) sont implémentées dans notre algorithme avec la contrainte *Alldifferent* appliquée à l'ensemble des positions $P_1, P_2, \ldots, P_n$.

On dimensionne le "cadre" de telle sorte qu'il soit le plus petit possible. Parfois, cette dimension ne permet pas de trouver la meilleure conformation de la protéine, on doit alors agrandir cette zone. On nomme les dimensions de ce "cadre" *Hauteur* et *Longueur*. En considérant que le coin supérieur gauche de ce "cadre" a pour coordonnées (S_x, S_y), on a pour tout $i \in (\mathcal{H} \cup P_{\text{singleton}})$ de la séquence les contraintes suivantes :

$$S_x \leq X_i \leq S_x + Largeur \qquad \text{et} \qquad S_y \leq Y_i \leq S_y + Hauteur.$$

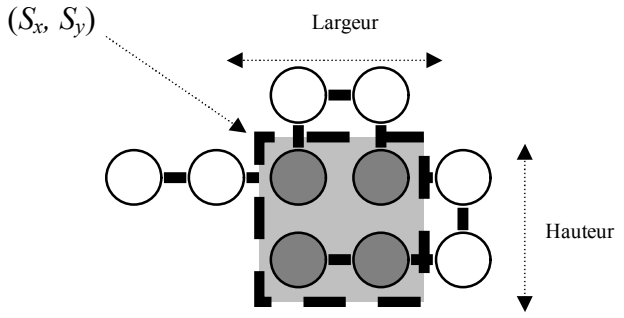


Figure 3.11. Exemple d'un "cadre".

Les variables utilisées dans la programmation par contraintes étendue, sachant que la distinction entre un monomère pair ou impair est relative à son indice dans la séquence, sont données dans le tableau 1.1.

X_i , Y_i	Les cordonnées du *i-ième* monomère s_i de la protéine. $X_i \in [1\ldots n]$, $Y_i \in [1\ldots n]$,
P_i	La position du *i-ième* monomère s_i dans la grille.
$XDiff_{i,j}$, $YDiff_{i,j}$	La distance entre les monomères s_i et s_j suivant l'axe des abscisses x (respectivement l'axe des ordonnées y). $XDiff_{i,j} = \mid X_i - X_j \mid$, $YDiff_{i,j} = \mid Y_i - Y_j \mid$ *avec* $i+1 < j$ *et* $i \in [1\ldots n]$, $j \in [2\ldots n]$.
$Contact_{ij}$	S'il y a un contact entre les monomères s_i et s_j avec $i, j \in \mathcal{H}$ et $j > i+2$ cette valeur est égale à 1, et à 0 sinon. $Contact_{ij} \in \{0,1\}$
Energie	"moins" la somme des contacts H-H
$Voisins_i$	Le nombre de monomères H voisins du monomère s_i = H.
Surf	La surface de la conformation.
XHE_i , YHE_i	Les coordonnées du *i-ième* monomère s_i=H pair relativement au "cadre" suivant l'axe des abscisses x (respectivement l'axe des ordonnées y).
XHO_i , YHO_i	Les coordonnées du *i-ième* monomère s_i=H impair relativement au "cadre" suivant l'axe des abscisses x (respectivement l'axe des ordonnées y).
XPE_i , YPE_i	Les coordonnées du *i-ième* monomère s_i qui est un P-singleton pair relativement au "cadre" suivant l'axe des abscisses x (respectivement l'axe des ordonnées y).
XPO_i , YPO_i	Les coordonnées du *i-ième* monomère s_i qui est un P-singleton impair relativement au "cadre" suivant l'axe des abscisses x (respectivement l'axe des ordonnées y).
$XSeh_i$, $YSeh_i$	Le nombre de monomères H pairs dans la *i-ième* ligne (respectivement colonne).
$XSoh_i$, $YSoh_i$	Le nombre de monomères H impairs dans la *i-ième* ligne (respectivement colonne).
Largeur	La largeur du "cadre".
Hauteur	La hauteur du "cadre".

Tableau 3.1. Les variables utilisées dans la programmation par contraintes étendue.

3.4.4.4. Conformations possibles de séquences partielles

Pour chacune des colonnes (respectivement lignes) d'une conformation, on peut imposer de nouvelles contraintes dues à la topologie de l'espace :

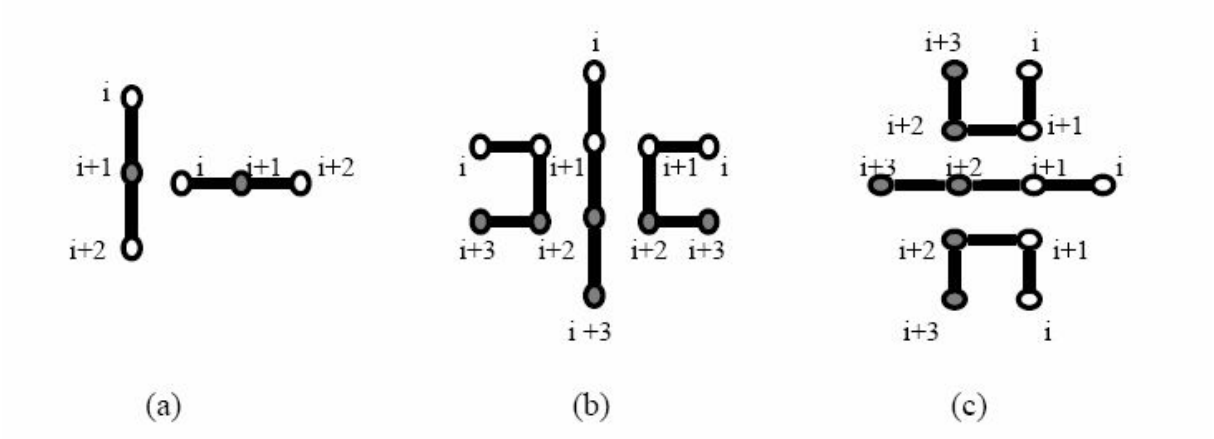


Figure 3.12. Conformations possibles de séquences partielles.

- Si les monomères *i* et *i*+2 sont dans la même colonne ou ligne, alors *i*+1 l'est aussi :

$$X_i = X_{i+2} \Rightarrow X_{i+1} = X_i \quad \text{(figure 3.12a gauche)}$$

$$Y_i = Y_{i+2} \Rightarrow Y_{i+1} = Y_i \quad \text{(figure 3.12a droite)}$$

- Si les monomères *i* et *i*+3 sont dans la même colonne ou ligne, alors *i*+1 et *i*+2 le sont aussi :

$$X_i = X_{i+3} \Rightarrow X_{i+1} = X_{i+2} \text{ (figure 3.12b)}$$

$$Y_i = Y_{i+3} \Rightarrow Y_{i+1} = Y_{i+2} \text{ (figure 3.12c)}$$

3.4.4.5. Calcul d'une borne inférieure de la surface d'une protéine

Rappelons que la fonction objectif retenue pour la résolution du problème de repliement de protéines est la maximisation du nombre de contacts H-H. Dans la section 3.4.4.2, nous avons montré que maximiser cette fonction objectif est équivalent à minimiser la surface de la protéine. De plus, comme nous l'avons vu à la section 3.4.2, pour que l'exploration de l'arbre de recherche soit efficace, il est indispensable de définir des bornes. Cette section explique comment on peut obtenir une borne inférieure sur la surface de la conformation d'une protéine.

Pour déterminer une borne inférieure de la surface d'une conformation nous distinguons les monomères H pairs et impairs. Cette distinction entre des monomères pairs et impairs est relative à leur indice dans la séquence. Par exemple, la première colonne de la figure 3.13 montre que nous avons zéro H pair et un H impair, un P-singleton pair et zéro P-singleton impair. Cette distinction est importante car les contacts H-H ne se font qu'entre les monomères H pairs et les monomères H impairs.
Intuitivement, il s'agit d'estimer la surface d'une conformation, en tenant compte uniquement des monomères qui se trouvent à l'intérieur du "cadre". Pour cela nous avons besoin de définir $XSeh_i$ (respectivement $YSeh_i$) comme le nombre de monomères H pairs de la *i-ième* ligne (respectivement *i-ième* colonne), $XSoh_i$ (respectivement $YSoh_i$) comme le nombre de monomères H impairs de la *i-ième* ligne (respectivement *i-ième* colonne).
On peut alors, à partir de ces variables, calculer une borne inférieure pour la surface en deux parties. La première partie est celle relative aux lignes du "cadre" (*Surfligne*) et la seconde partie est relative aux colonnes du "cadre" (*Surfcolonne*). Chacune de ces deux parties compte le nombre de H qui se trouve sur la bordure du "cadre" plus le nombre de couples de monomères voisins tels que le premier élément du couple est un monomère H, mais pas le

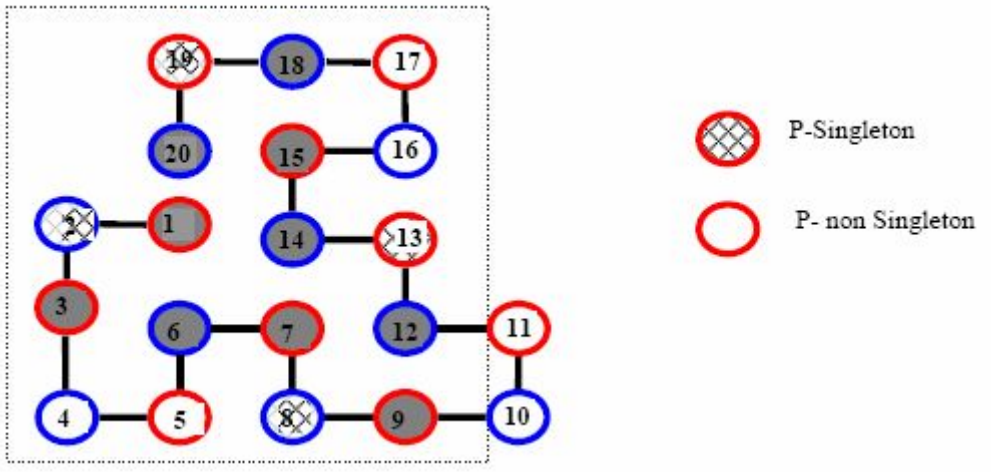
second. Le second composant de cette somme est obtenu par la différence entre le nombre de H pairs d'une colonne *i* (respectivement ligne *i*) moins les H impairs de la colonne *i+1* (respectivement ligne *i+1*) et les H impairs d'une colonne *i* (respectivement ligne *i*) moins les H pairs de la colonne *i+1* (respectivement ligne *i+1*). Nous appliquons cette différence sur toutes les colonnes (respectivement lignes) du cadre en supposant que, dans le pire des cas, tous les H pairs sont en contact avec les H impairs. Ainsi on peut écrire [Bac 01]:

$$Surfligne \geq XSoh_1+XSeh_1+XSoh_{Largeur}+XSeh_{Largeur}+(\Sigma_{1\leq i <Largeur} (|XSoh_i - XSeh_{i+1}|-| XSeh_i - XSoh_{i+1}|))$$

$$Surfcolonne \geq YSoh_1+YSeh_1+YSoh_{hauteur}+YSeh_{hauteur}+(\Sigma_{1\leq i <Hauteur} (|YSoh_i - YSeh_{i+1}|-| YSeh_i - YSoh_{i+1}|))$$

Donc $Surf \geq Surfligne + Surfcolonne$

Afin de mieux expliciter le calcul de cette borne inférieure, nous considérons la séquence suivante $H_1P_2H_3P_4P_5H_6H_7P_8H_9P_{10}P_{11}H_{12}P_{13}H_{14}H_{15}P_{16}P_{17}H_{18}P_{19}H_{20}$, où chaque indice de monomère indique sa position. Soit la conformation correspondante suivante (figure 3.13) :




```
YSeh (H pairs)  0     2       2     1
YSoh (H impairs)1     1       2     1
```

Figure 3.13. Exemple de calcul de la borne inférieure de la surface de cette conformation pour la séquence $H_1P_2H_3P_4P_5H_6H_7P_8H_9P_{10}P_{11}H_{12}P_{13}H_{14}H_{15}P_{16}P_{17}H_{18}P_{19}H_{20}$.

Le calcul de la borne inférieure de la surface de l'exemple de la figure 3.13 est donné par :

$$Surfcolonne \geq 1 + 0 + 1 + 1 +[|1-2|-|0-1|+ |1-2|-|2-2| + |2-1|-|2-1|]$$

Surfcolonne ≥ 4
De la même manière, on calcule la surface en ligne et on trouve : *Surfligne* ≥ 0
D'où Surf ≥ 4
D'après la relation entre la surface et le nombre de contacts de la section 3.4.4.2 : *Contacts* ≤16

3.5. Recherche locale

Dans les algorithmes que nous allons proposer par la suite nous incorporons une composante "recherche locale". Cet algorithme peut être non seulement utilisé par les colonies de fourmis, mais également par la programmation par contraintes, comme indiqué dans la section 3.4.2.
Dans cette section, nous décrivons le voisinage "mouvement d'attraction", introduit par Lesh dans [Les 03], et qui s'est révélé donner de bons résultats pour le problème du repliement de protéines.
Le voisinage "mouvement d'attraction" consiste à déplacer les acides aminés de la manière suivante.
Soit un acide aminé s_i et soit P_L une position libre dans l'espace L des positions adjacentes au monomère s_{i+1}, et P_C une position voisine de P_L et de s_i.
Pour pouvoir réaliser un mouvement "mouvement d'attraction", il y a deux cas possibles :

- P_C est libre (figure 3.14a)
- $P_C = s_{i-1}$ (figure 3.14b)

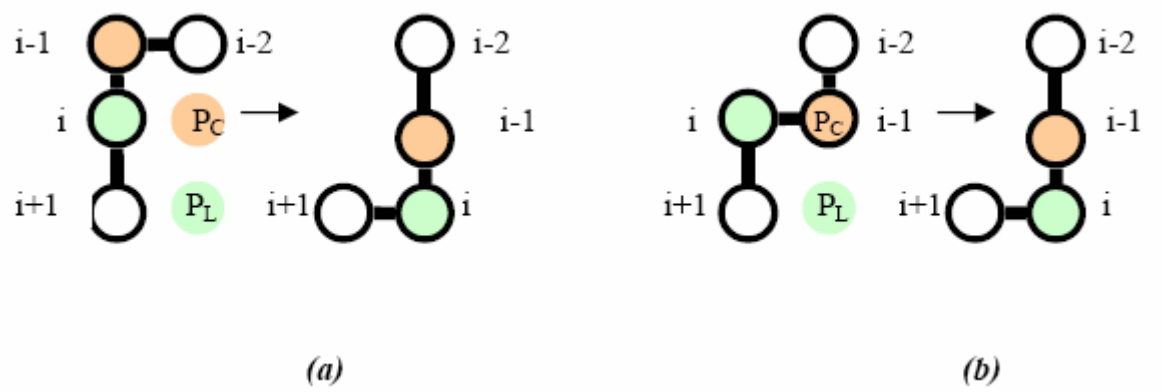


Figure 3.14. Le mouvement "mouvement d'attraction".

On place alors s_i à la position P_L et on déplace les acides aminés s_{i-1}, s_{i-2}… jusqu'à atteindre une conformation. On peut bien sûr effectuer le mouvement dans l'autre sens.

Il y a potentiellement 4 mouvements "mouvement d'attraction" pour chaque acide aminé :

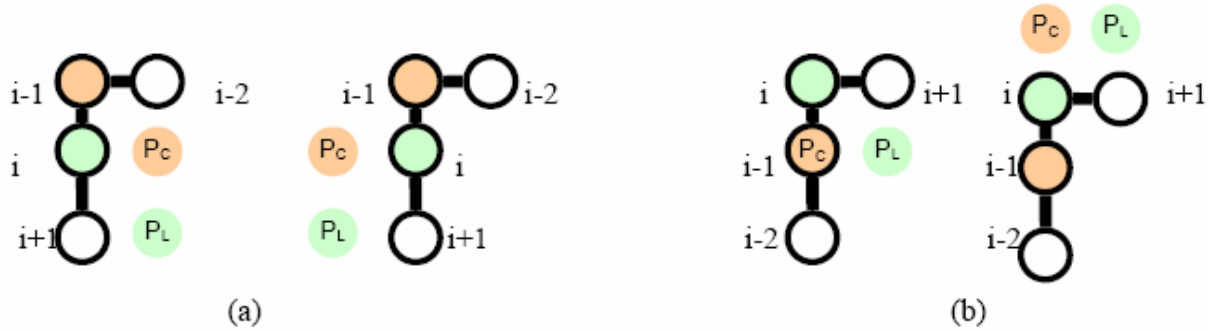


Figure 3.15. Ensemble des mouvements "mouvement d'attraction" possibles pour deux monomères donnés s_i et s_{i-1}

En effet, ces 4 mouvements sont possibles, car on peut parcourir la séquence dans les deux sens (voir les cas (a) et (b) de la figure 3.15), et de plus les positions P_L et P_C peuvent être placées de chaque côté du monomère.

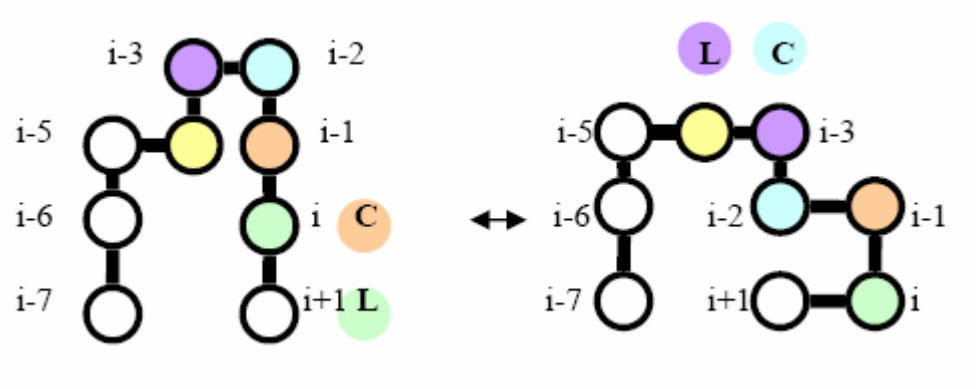


Figure 3.16. Un mouvement "mouvement d'attraction" (*Pull- Move*) sur un cas plus complexe.

Dans la figure 3.16, le mouvement "mouvement d'attraction" est plus complexe, car il implique un déplacement de 4 monomères de la séquence (le monomère jaune ne bouge pas). De plus, on peut remarquer que ce voisinage est réversible, comme le montre la figure 3.16. La conformation (a) est obtenue en appliquant un mouvement "mouvement d'attraction" sur les monomères s_{i-3} et s_{i-2} à partir de la conformation (b). Pour obtenir la conformation (b) à partir de (a), on applique le mouvement "mouvement d'attraction" sur les monomères s_i et s_{i-1}.

3.6. Les colonies de fourmis

Nous allons adapter la métaheuristique des colonies de fourmis au problème du repliement de protéines.

A chaque itération de l'algorithme, *k* fourmis construisent chacune une solution en prenant des décisions. Ces décisions sont prises à partir d'un critère heuristique, ainsi que sur les traces de phéromone. Ces traces sont mises à jour en fonction de la qualité des solutions obtenues. Elles sont renforcées pour les choix ayant donné de meilleures solutions, et diminuées pour les autres.

En pratique, on répète les trois phases suivantes, jusqu'à atteindre un nombre maximum d'itérations fixé initialement :

- *construction* de *k* solutions initiales,
- *application (ou non) d'une recherche locale* à chacune de ces solutions
- *mise à jour du taux de phéromone* en fonction des solutions trouvées

Shmygelska dans [Shm 02][Shm 05] propose un algorithme de colonies de fourmis pour la résolution du problème de repliement de protéines. La différence entre cet algorithme et le notre réside dans la phase de construction et dans l'algorithme de recherche locale.
Dans la phase de construction de l'algorithme de Shmygelska [Shm 02], chaque fourmi commence le placement des monomères d'une conformation donnée à partir d'un monomère choisi aléatoirement alors que, dans le notre, l'ordre d'apparition des monomères est respecté. Cette façon de procéder évite d'avoir à choisir le sens du parcours de la séquence initiale à partir du premier monomère placé. Autrement dit, on n'a pas à répondre à la question : faut-il placer d'abord les monomères qui se trouvent à gauche ou à droite du premier monomère choisi ?
Une autre différence dans cette phase consiste à introduire, dans notre algorithme, la notion de "cadre", afin de tenter de ramener tous les monomères H vers l'intérieur de ce cadre et ainsi tenter d'obtenir de meilleures conformations.
Dans la phase de recherche locale, notre algorithme utilise le voisinage "mouvement d'attraction" alors que l'algorithme de Shmygelska utilise le voisinage "long-range move". Ce voisinage consiste à déplacer un monomère choisi aléatoirement, vers une position prometteuse obtenue après avoir testé toutes les directions possibles. Les autres monomères voisins sont alors "poussés" vers différentes positions.

3.6.1. Principe de l'algorithme proposé

Les fourmis construisent des conformations possibles pour une protéine et actualisent les taux de phéromone à partir de la qualité des solutions obtenues.
Les conformations d'une protéine sont modélisées par une série de chiffres qui représentent la direction relative d'un monomère par rapport aux 2 monomères précédents le long de la séquence. Un 0 indique que le monomère est placé à gauche par rapport aux 2 monomères précédents (respectivement 1 pour la direction "avant" et 2 pour la direction "droite").
Comme les conformations ne changent pas si on effectue une rotation, la position des deux premiers monomères peut être fixée sans problème.
Par exemple (figure 3.17), une conformation possible de la séquence 1 [Dil 95] (HPHPPHHPHPPHPHHPPHPH) peut être représentée par la séquence (0100220100202200 10). On remarquera que, pour une protéine de longueur *n*, la conformation associée a une longueur de (*n*-2) motifs.

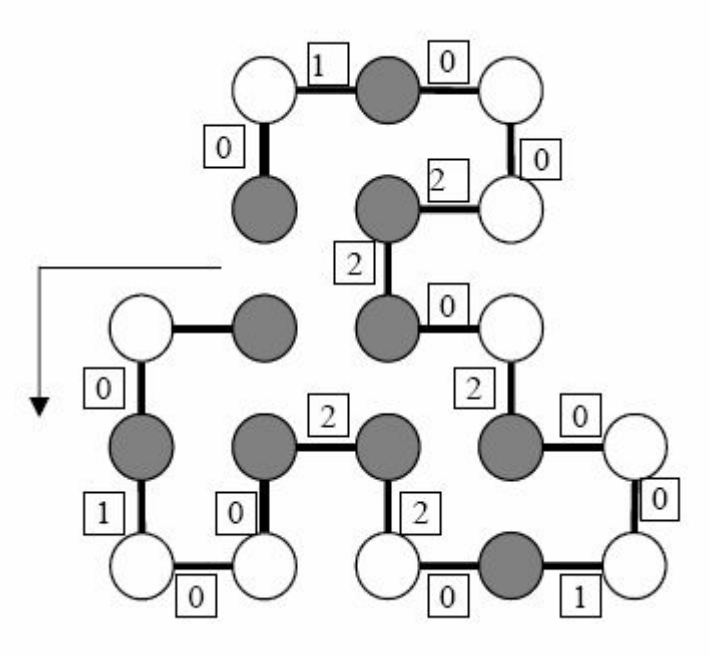


Figure 3.17. Représentation d'une conformation d'une protéine à l'aide des coordonnées relatives.

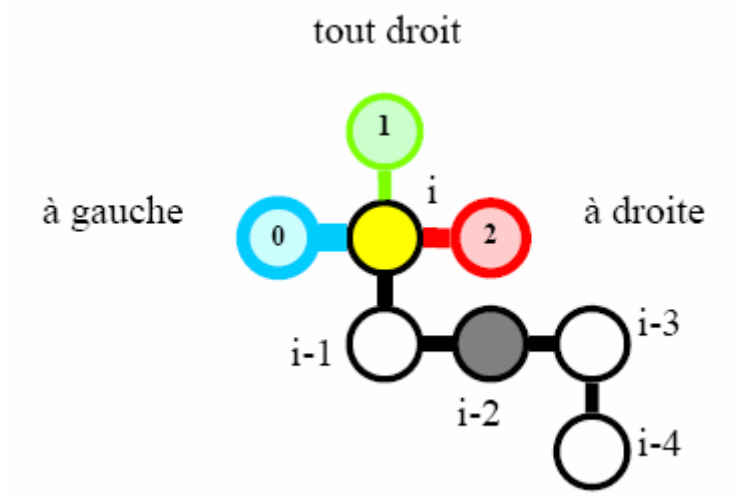


Figure 3.18. Choix du placement d'un monomère en fonction du taux de phéromone.

Pour chaque monomère s_i et pour chaque direction $d \in \{0,1,2\}$, on dispose d'un taux de phéromone $\tau_{i,d}$. Ce taux de phéromone intervient pour déterminer la probabilité qu'a une fourmi de placer le monomère s_{i+1} dans la direction d par rapport aux monomères s_i et s_{i-1} sachant que le monomère s_i est le dernier monomère placé. Ainsi, dans la figure 3.18, nous avons indiqué les choix possibles pour le prochain monomère s_{i+1} à placer. L'épaisseur des traits correspond au taux de phéromone. Plus la quantité est grande pour une direction, plus l'agent (ou fourmi) choisit cette direction. Ici, on a plus de chances de voir le prochain monomère placé à la gauche du dernier monomère s_i placé.

Afin de diminuer le nombre de possibilités de placement des monomères H, nous avons utilisé la notion de "cadre", présentée dans la section 3.4.4.3. Le but est qu'il puisse contenir, de préférence, tous les monomères H (voir figure 3.19).
La difficulté de cette procédure est de définir la taille adéquate de ce "cadre". Celle-ci est fixée expérimentalement en fonction de l'instance traitée.
Nous allons, par la suite, détailler le principe de fonctionnement de notre algorithme. Il en existe deux versions, selon que le "cadre" est pris en compte ou non. Ces deux versions présentent des aspects semblables que nous exposons, par la suite, sans distinction de cas.

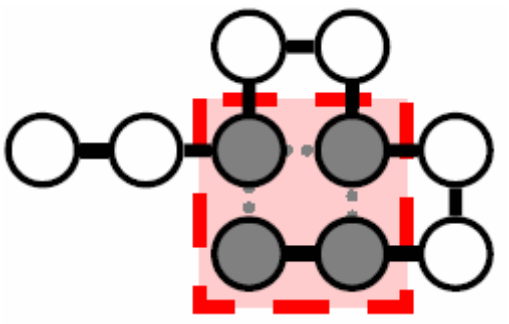


Figure 3.19. Protéine dont les monomères H sont à l'intérieur du "cadre".

3.6.2. Eléments de l'algorithme de colonie de fourmis

Lors de la phase de construction de notre algorithme, les deux premiers acides aminés sont fixés au départ sur la grille. Chaque fourmi commence donc à partir du $3^{ième}$ acide aminé de la séquence de la protéine donnée. A partir de cette position, la fourmi décide de suivre une direction relative à chaque étape de construction de l'algorithme, déterminée suivant une loi de probabilité basée sur des taux de phéromone et des valeurs heuristiques.

3.6.2.1. Positions initiales

Dans la version "sans cadre" de notre algorithme, le premier monomère est placé au centre de la grille. Dans la version avec "cadre", deux possibilités sont envisageables :
- Si un monomère P est en première position dans la séquence, on le place à l'extérieur du "cadre", limitant sa position à $\lceil$ *Hauteur* du "cadre"$/2\rceil$ positions (figure 3.20).

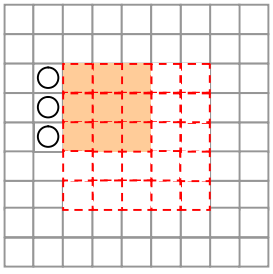


Figure 3.20. Positions possibles du premier monomère P pour une séquence commençant par "PH".

Dans notre exemple, le "cadre" est de taille 5*5 ; en retirant les symétries possibles, il ne reste que trois positions possibles pour placer le monomère P.
- Si la séquence de protéine commence par *l* monomères P, alors le premier monomère P de la séquence est situé à *l* cases du "cadre".
- Si un monomère H est en première position dans la séquence, on le place à l'intérieur du "cadre", dans l'une des cases marquées dans la figure 3.21 (c'est-à-dire 6 cases, en ayant retiré les cas de symétrie).

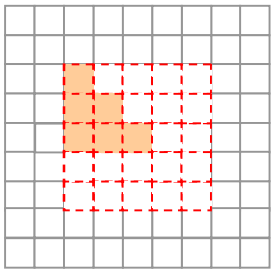


Figure 3.21. Positions possibles du premier monomère "H" pour une séquence commençant par "H".

3.6.2.2. Paramètres heuristiques employés

Pour chaque monomère s_i et une direction $d \in \{0,1,2\}$, une valeur de la phéromone $\tau_{i,d}$ est définie. Néanmoins, il n'est pas rare de trouver dans les algorithmes de colonies de fourmis d'autres paramètres servant à guider les fourmis dans leur phase de construction d'une solution. Ces autres paramètres sont communément désignés sous le nom de "paramètres heuristiques". Ainsi, nous utilisons une valeur heuristique $\mu_{i,d}$ dont le but est de guider la recherche vers de bonnes conformations. Cette valeur heuristique est basée sur $h_{i,d}$ le nombre de nouveaux contacts H-H obtenus par rapport aux monomères H déjà placés, en plaçant le monomère s_{i+1}, dans la direction d, relativement à s_i et s_{i-1}.
Nous définissons $\mu_{i,d} = h_{i,d} + 1$, pour éviter le problème de division par zéro. Ces paramètres sont utilisés dans les deux versions de notre algorithme.

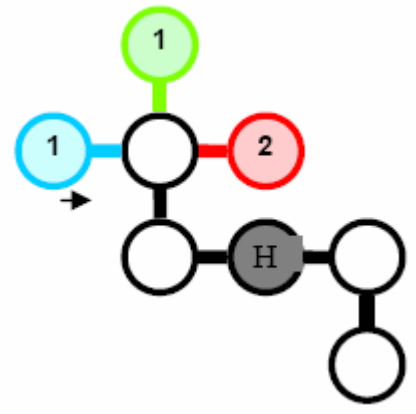


Figure 3.22. Choix de placement d'un monomère, en fonction du nombre de contacts possibles.

Dans la figure 3.22, la valeur de $h_{i,d}$ est inscrite à l'intérieur de chaque cercle et on suppose que le monomère à placer est un H. Si on place le monomère à droite, il y aura un contact, d'où la valeur de 2.

Les paramètres suivants sont utilisés uniquement dans l'algorithme des colonies de fourmis avec le "cadre" :

- $D_{i,d}$ la distance du monomère s_{i+1} par rapport au "cadre" si s_{i+1} est placé suivant la direction d par rapport à s_i et s_{i-1}. Si le monomère s_{i+1} est dans le "cadre", alors $D_{i,d}$ vaut 1 sinon, ce paramètre vaut sa distance réelle+1 (on ajoute 1 pour éviter le problème de la division par zéro).

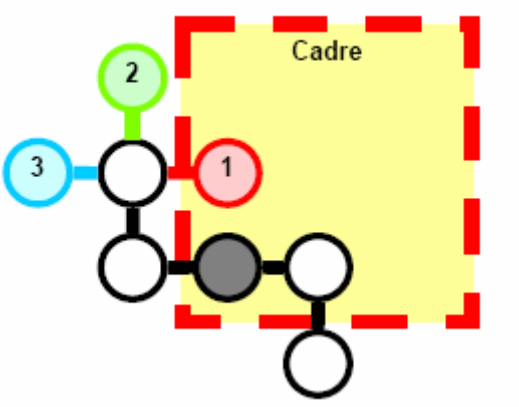


Figure 3.23. Choix de placement d'un monomère en fonction de la distance par rapport au "cadre".

Sur la figure 3.23, les valeurs de $D_{i,d}$ sont indiquées dans le cercle. Si on place le monomère à droite, la distance par rapport au "cadre" est nulle. Donc $D_{i,d}$ vaut 1.

- $C_{i,d}$ donne un score (un poids) au monomère s_i suivant sa position par rapport au "cadre". Plus le score $C_{i,d}$ est grand, plus on favorise le placement de s_i selon la direction d. Ce paramètre permet d'éviter, entre autres, la présence des cavités à l'intérieur du "cadre" et plus précisément en bordure du "cadre". Ces valeurs sont calculées pour chaque protéine de manière empirique et influent sur le calcul de la probabilité pour un monomère s_i d'être placé selon la direction d.
  Soient les monomères s_{i-1} et s_i déjà placés. On souhaite placer le monomère s_{i+1} par rapport à s_{i-1} et s_i. Ce placement peut s'effectuer soit à l'intérieur du "cadre", soit sur la bordure ou hors du cadre. On appellera bordure du "cadre" (figure 3.24) les cases du "cadre" se situant au bord de ce dernier.
  Des tests expérimentaux, ont permis de fixer la valeur de $C_{i,d}$ en fonction de la position de placement de s_{i+1} et de son type (voir l'exemple de la figure 3.25). Les valeurs retenues sont résumées dans le tableau 3.2.

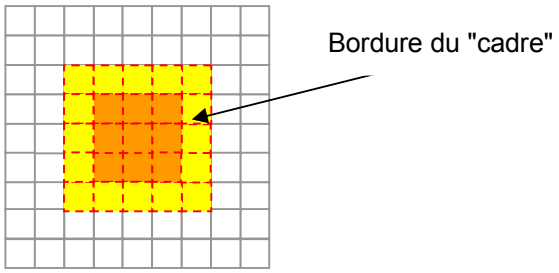


Figure 3.24. Représentation d'un "cadre" dans la grille et de sa bordure.

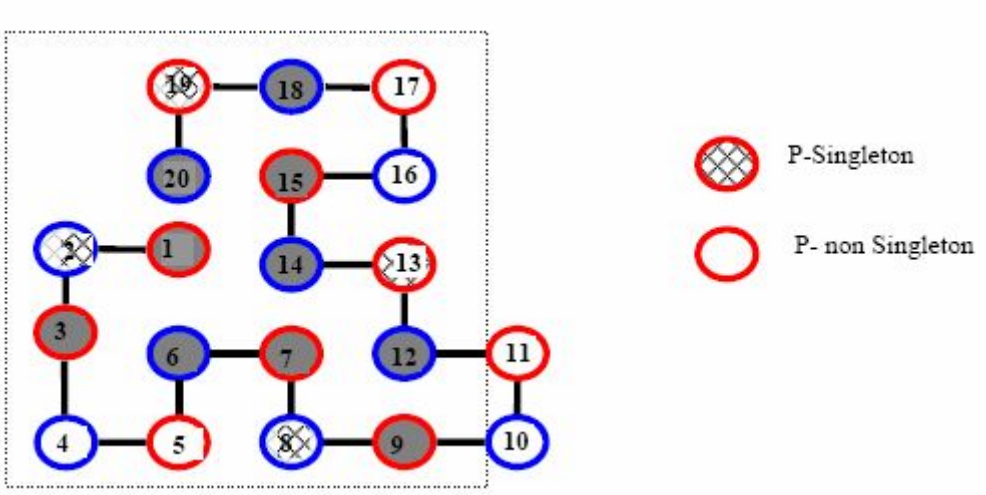


Figure 3.25. Une protéine dont les monomères sont classés par groupes.

La figure 3.25 montre une conformation de protéine dont les monomères P-singletons sont P_2, P_8, P_{13} et P_{19} et les monomères P non-singletons sont P_4, P_5, P_{10}, P_{11}, P_{16} et P_{17}. Les monomères H sont subdivisés en deux groupes ceux qui sont voisins de monomères P non-singletons qui sont H_3, H_6, H_9, H_{12}, H_{15} et H_{18} et les autres H_1, H_7, H_{14} et H_{20}.
Les scores retenus sont consignés dans le tableau 3.2. Pour un type de monomère s_i donné, si le mouvement, opéré selon la direction d, le conduit à l'intérieur du "cadre" ou sur la bordure du "cadre" ou hors du "cadre", les valeurs de $C_{i,d}$ sont données dans le tableau 3.2.

Monomère	Intérieur du "cadre"	Dans la bordure du "cadre"	Hors du "cadre"
P-singleton	1	2	0
P non-singleton	0	1	2
H voisins de P non-singletons	1	2	0
Autres H	2	2	0

Tableau 3.2. Les scores $C_{i,d}$ selon la direction d pour un monomère s_i.

On remarquera que, pour les séquences 3 à 5 provenant de benchmarks de la littérature [Dil 95], par exemple, la première ligne du tableau 3.2 n'est pas utilisée, car elles ne contiennent pas de P-singletons.

3.6.2.3. Calcul des probabilités

Pendant la phase de construction, pour étendre une conformation partielle $s_1 \ldots s_i$ à s_{i+1}, une fourmi choisit de suivre la direction relative $d \in \{0,1,2\}$ d'après les probabilités suivantes:

Cas 1 : algorithme des colonies de fourmis sans le "cadre"

La probabilité est donnée par :

$$p_{i,d} = \frac{\tau_{i,d}^{\alpha} * \mu_{i,d}^{\beta}}{\sum_{e \in \{0,1,2\}} \tau_{i,e}^{\alpha} * \mu_{i,e}^{\beta}}$$

avec α et β qui déterminent l'influence relative du taux de phéromone et du paramètre heuristique.

Cas 2 : algorithme des colonies de fourmis avec le "cadre"

La probabilité est donnée par :

$$p_{i,d} = \frac{\tau_{i,d}^{\alpha} * \mu_{i,d}^{\beta} * C_{i,d}^{\gamma} * D_{i,d}^{\delta}}{\sum_{e \in \{0,1;2\}} \tau_{i,e}^{\alpha} * \mu_{i,e}^{\beta} * C_{i,e}^{\gamma} * D_{i,e}^{\delta}}$$

avec
- γ qui détermine l'influence de la position du monomère par rapport au "cadre",
- δ qui détermine l'influence de la distance du monomère par rapport au "cadre",
- α et β qui déterminent l'influence relative du taux de phéromone et du paramètre heuristique.

3.6.2.4. Actualisation du taux de phéromone

Quand une solution est trouvée par une fourmi, cette solution, après application ou non d'une recherche locale, est évaluée. A la fin du processus de recherche de la meilleure solution par toutes les fourmis, on actualise le taux de phéromone à partir de la qualité de cette solution, de la manière suivante :

$$\forall i,d \qquad \tau_{i,d}^{+} = \tau_{i,d} * (1-\varepsilon) + \sum_c \Delta_{i,d,c}$$

avec $\Delta_{i,d,c} = E(c)/E^*$ si dans la conformation c, s_{i+1} est placé selon la direction d par rapport à s_i et s_{i-1} ,

0 sinon

L'évaporation permet de faire diminuer le taux de phéromone, dans le but d'effacer à terme les résultats obtenus, afin de permettre à d'autres solutions qui ne pouvaient pas être sélectionnées d'être testées.
ε est le taux d'évaporation, $E(c)$ est l'énergie de la conformation c calculée et E^* une estimation du nombre de contacts optimum de la solution. En pratique, la valeur de E^* est soit un optimum connu pour les benchmarks de la littérature [Dil 95], soit cette valeur est estimée par (|H| +1). En effet, un monomère H ne peut pas avoir plus de 2 contacts, sauf le premier et le dernier monomère. De plus les H impairs ne peuvent pas avoir de contacts entre eux ni les H pairs. Au maximum on peut donc avoir (|H| +1) contacts possibles.

Dans l'algorithme (figure 3.26), développé sous Mozart-Oz, les fourmis sont lancées en parallèle lors de chaque itération.

```
Algorithme : Colonie_Fourmis

Début
Initialiser Paramètres
Initialiser meilleure solution à 0
Pour chaque itération faire
      Pour chaque fourmi faire
            Pour chaque monomère, faire
                  Calculer les paramètres et déterminer la probabilité
                  pour chaque direction
                  Choisir une direction
            fait
            Si la conformation est valide
            alors      Effectuer une recherche locale (voisinage
                       "mouvement d'attraction")
                       Comparer à la meilleure solution
                       Si la solution trouvée est meilleure
                             alors elle devient la meilleure solution
                             sinon rien
                       fin si
            sinon rien
           fin si
      fait
      Réactualiser le taux de phéromone
fait
Retourner meilleure solution
FIN.
```

Figure 3.26. L'Algorithme de colonie de fourmis implémenté.

3.7. Coopération entre la programmation par contraintes et les colonies de fourmis

Nous avons cherché à obtenir des schémas de coopération génériques pouvant être utilisés pour de nombreux problèmes d'optimisation combinatoire.
Deux algorithmes de coopération ont ainsi été proposés. Ce type de coopération entre la métaheuristique des colonies de fourmis et la programmation par contraintes permet une meilleure recherche de la solution, en réduisant l'espace de recherche exploré par la programmation par contraintes de deux manières. En effet, d'une part il est possible d'exploiter une bonne solution déjà trouvée par les colonies de fourmis. Ces renseignements sont exprimés par des contraintes additionnelles : après qu'une solution ait été trouvée, la contrainte supplémentaire qu'une prochaine solution doit être meilleure est prise en considération. Avec cette contrainte supplémentaire, l'arbre de recherche peut devenir considérablement plus petit.
D'autre part, les valeurs de phéromone calculées par les fourmis sont utilisées pour explorer l'arbre de recherche. Celui-ci est exploré avec la stratégie en profondeur d'abord, mais, quand une distribution se produit, la branche sur laquelle la valeur de la phéromone est la plus importante est prise en premier. Il s'agit en fait de choisir l'ordre des directions à explorer pour chaque monomère et ainsi commencer par explorer les directions les plus prometteuses.

La réalisation de cette coopération nécessite l'introduction de nouvelles contraintes. Celles-ci traitent d'une représentation relative des monomères, qui sera utilisée dans notre algorithme à base de colonies de fourmis. La représentation relative des monomères nécessite la définition d'une nouvelle variable. Pour chaque monomère s_i, on introduit une variable $Succ_i \in \{0(\text{gauche}), 1(\text{avancer}), 2(\text{droite})\}$ qui indique la position de s_{i+1} par rapport à s_i et s_{i-1}. Cette variable étant définie, il faut établir une correspondance avec les positions absolues d'un monomère. Ainsi, les contraintes supplémentaires, liant ce concept de "successeur" à la notion de position absolue, sont définies comme suit :

$$
\begin{array}{lcll}
Succ_i = 0 & \Leftrightarrow & X_{i+1} = X_i - Y_i + Y_{i-1} & \wedge \quad Y_{i+1} = Y_i + X_i - X_{i-1} \\
Succ_i = 1 & \Leftrightarrow & X_{i+1} = 2X_i - X_{i-1} & \wedge \quad Y_{i+1} = 2Y_i - Y_{i-1} \\
Succ_i = 2 & \Leftrightarrow & X_{i+1} = X_i + Y_i - Y_{i-1} & \wedge \quad Y_{i+1} = Y_i - X_i + X_{i-1}
\end{array}
$$

L'exemple de la figure 3.27 montre comment la correspondance entre les positions absolues et relatives d'un monomère est réalisée. On désire placer le monomère s_{i+1} par rapport à s_i et s_{i-1} :

$$
\begin{array}{lcll}
Succ_i = 0 & \Leftrightarrow & X_{i+1} = 5 - 7 + 6 = 4 & \wedge \quad Y_{i+1} = 7 - 5 + 5 = 7 \\
Succ_i = 1 & \Leftrightarrow & X_{i+1} = 10 - 5 = 5 & \wedge \quad Y_{i+1} = 14 - 6 = 8 \\
Succ_i = 2 & \Leftrightarrow & X_{i+1} = 5 + 7 - 6 = 6 & \wedge \quad Y_{i+1} = 7 - 5 + 5 = 7
\end{array}
$$

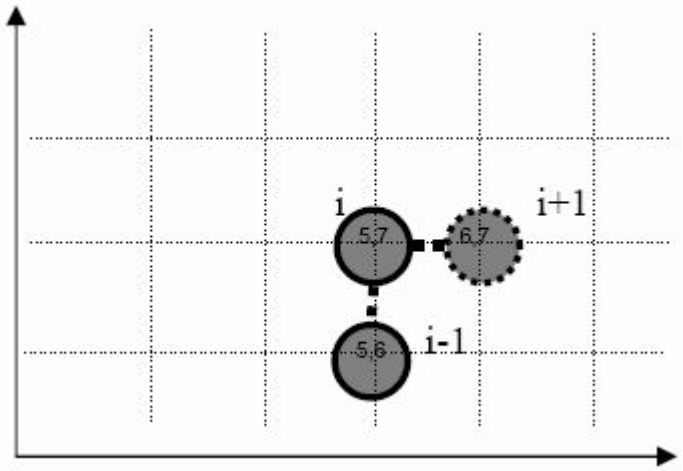


Figure 3.27. Exemple de correspondance entre les coordonnées absolues d'un monomère s_i et la notion de "successeur".

Le premier algorithme de coopération utilise un algorithme des *colonies de fourmis simple* (sans la notion de "cadre") et un algorithme de *programmation par contraintes de base*. Quant au second, plus spécifique au problème traité, il utilise l'algorithme de *programmation par contraintes étendue*. Ces deux algorithmes ont en commun la stratégie de distribution des variables, de la programmation par contraintes. Au lieu d'explorer des coordonnées absolues des monomères (X_i et Y_i), ce sont plutôt des positions relatives des monomères $Succ_i$ qui sont considérées, c'est-à-dire 0 (à gauche), 1 (tout droit) et 2 (à droite). On a donc deux types de variables qui coexistent : $Succ_i$ et X_i, Y_i. Le problème est de savoir par quel type de variables

la distribution se fera, afin que la programmation par contraintes soit la plus efficace possible. Ainsi, nous avons choisi les variables de plus petit domaine. Celles-ci correspondent bien à celles prévues dans notre conception, à savoir les variables $Succ_i$.
Lorsqu'un ensemble de solutions est trouvé par les colonies de fourmis, on peut l'utiliser de deux manières différentes.

- Le coût de la meilleure solution trouvée par les colonies de fourmis est utilisé comme une borne inférieure pour la programmation par contraintes.
- On peut également récupérer les taux de phéromone utilisés par les fourmis. Ces taux de phéromone permettront d'orienter l'arbre de recherche dans la programmation par contraintes. Classiquement, la programmation par contraintes ne privilégiera aucune direction (0(à gauche), 1(tout droit) et 2(à droite)) par rapport à une autre. Par contre, supposons que le taux de phéromone soit plus important pour la direction 1 puis pour la direction 2, puis la direction 0, alors la programmation par contraintes testera les directions dans l'ordre suivant : 1,2,0.

L'idée de la première coopération est d'exécuter, de manière *cyclique*, un algorithme de colonies de fourmis simple puis l'algorithme de programmation par contraintes de base. Chacun de ces deux algorithmes utilise, lors de son exécution, les connaissances recueillies par l'autre.
Au début, les colonies de fourmis établissent une solution qui servira de borne inférieure pour la programmation par contraintes et ainsi servira à élaguer des branches de l'arbre de recherche. De plus, grâce aux taux de phéromone calculés pour chaque monomère, la programmation par contraintes sera guidée, lors de l'exploration de l'arbre, en commençant par explorer les branches les plus prometteuses.
L'algorithme de programmation par contraintes peut être arrêté pour exécuter à nouveau l'algorithme des colonies de fourmis. Cet arrêt est réalisé sous Mozart-Oz par différentes commandes *(recherche de la meilleure solution, recherche de toutes les solutions, recherche de la première solution réalisable, arrêt de l'exécution, arrêt après le retour de la prochaine solution etc.)*. L'utilité de ce procédé est que l'algorithme des colonies de fourmis utilise les connaissances de la programmation par contraintes. En effet, au lieu de chercher la solution optimale par la programmation par contraintes, on ne cherche qu'une solution réalisable. Cette solution est ensuite utilisée comme si elle provenait d'une fourmi, ce qui permettra de calculer le taux de phéromone et de relancer à nouveau l'algorithme des colonies de fourmis. On itère ce processus cyclique un certain nombre de fois (expérimentalement 10 fois).

Le second algorithme de coopération utilise l'algorithme de programmation par contraintes étendu. L'algorithme des colonies de fourmis est exécuté en premier jusqu'à l'atteinte d'un nombre d'itérations fixé expérimentalement à 400 puis, grâce aux valeurs de la phéromone des monomères, la programmation par contraintes avec "cadre" est exécutée, jusqu'à l'obtention de la solution optimale (*Explorer.best*). Expérimentalement, pour certaines instances de grande taille (plus de 64 monomères), nous avons interrompu l'exécution après 15 heures de temps. Ce schéma de coopération est exécuté une fois.

On peut résumer les différentes approches proposées par la figure 3.28 :

(1) PPC de base seule,
(2) Colonies de fourmis sans "cadre" seule,
(3) PPC de base avec la recherche locale,
(4) PPC étendue avec la recherche locale,
(5) PPC étendue avec colonies de fourmis avec "cadre",
(6) PPC de base avec colonies de fourmis sans "cadre".

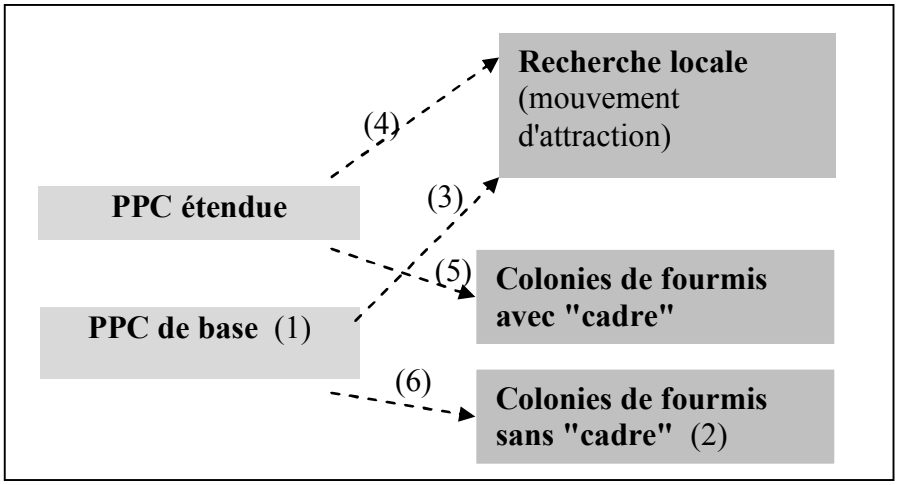


Figure 3.28. Approches de coopération réalisées.

3.8. Partie expérimentale

Pour analyser la performance des algorithmes proposés et développés en OZ sous l'environnement Mozart [Moz 04], une série de tests ont été effectués sur des benchmarks de la littérature [Dil 95]. Ces tests ont été réalisés sur un micro-ordinateur de type Pentium 4 (2.40GHz, 512 Mo de RAM) fonctionnant sous Windows.

3.8.1. Résultats de la littérature

Le tableau 3.3 présente les benchmarks de la littérature [Dil 95] qui nous ont servi lors des différentes expérimentations de nos algorithmes. Ce tableau présente les numéros de séquences (*N° Séq.)*, les protéines, la longueur correspondante (*Lg.*) ainsi que la meilleure solution connue (*Nb_contacts*), pour chaque séquence, en terme de nombre de contacts H-H.

N° Séq.	Protéine	Lg.	Nb_contacts
1	$(HP)_2\ PH\ (HP)_2\ (PH)_2\ HP\ (PH)_2$	20	9
2	$H\ (HP_2)_7\ H_2$	24	9
3	$(P_2H)_2\ H\ (P_4\ H_2)_3$	25	8
4	$P\ (P_2\ H2)_2\ P_5\ H_7\ P_2\ H_2\ P_4\ H\ (H\ P_2)_2$	36	14
5	$P_2\ H\ (P_2\ H_2)_2\ P_5\ H_{10}\ P_5\ (H_2\ P_2)_2\ H\ P_2\ H_5$	48	23
6	$H_2\ (P\ H)_4\ H_3\ P\ H\ (P_3\ H)_2\ P_4\ (H\ P_3)_2\ H\ P\ H_4\ (P\ H)_4\ H$	50	21
7	$P_2\ H_3\ P\ H_8\ P_3\ H_{10}\ P\ H\ P_3\ H_{12}\ P_4\ H_6\ P\ H_2\ P\ H\ P$	60	36
8	$H_{12}\ (P\ H)_2\ ((P_2\ H_2)_2\ P_2\ H\ )_3\ (P\ H)_2\ H_{11}$	64	42
9	$H_4\ P_4\ H_{12}\ P_6\ (H_{12}\ P_3)_3\ H\ P_2\ (H_2P_2)_2\ H\ P\ H$	85	53
10	$P_3\ H_2\ P_2\ H_4\ P_2\ H_3\ (P\ H_2)_3\ H_2\ P_8\ H_6\ P_2\ H_6\ P_9\ H\ P\ H_2\ P\ H_{11}\ P_2\ H_3\ P\ H_2\ P\ H\ P_2\ H\ P\ H_3\ P_6\ H_3$	100	50

11	P_6 H P H_2 P_5 H_3 P H_5 P H_2 $(P_2$ $H_2)_2$ P H_5 P H_{10} P H_2 P H_7 P_{11} H_7 P_2 H P H_3 P_6 H P H P_2	100	48

Tableau 3.3. Benchmarks de la littérature [Dil 95].

Le tableau 3.4 résume les différents résultats obtenus avec différentes approches de résolution. L'environnement matériel utilisé n'est pas connu pour la méthode de Monte-Carlo; mais toutes les autres ont été exécutées sur un Intel Pentium III cadencé à 1GHz, sauf pour la méthode *Gtabu*, qui a été utilisée sur un processeur Alpha à 1 GHz et ACO [Shm 05], qui a été exécuté sur un Pentium 4- 2.4 GHz, 256 Kb de cache et 1Mb de RAM.

On remarque que l'ensemble de ces méthodes marche très bien pour les protéines de taille allant jusqu'à 50 acides aminés (séquences de 1 à 6). Au-delà, il existe une divergence entre les méthodes. Seuls les articles les plus récents [Meh 02][Shm 03][Les 03][Shm 05] utilisent des séquences de taille plus grande (la puissance des machines aidant).

Les colonies de fourmis semblent donner de bons résultats, cependant le temps de calcul pour obtenir ces résultats est assez long, comme le montre le tableau 3.4 [Shm 03]. Le temps CPU de l'algorithme des colonies de fourmis étendu au cas 3D [Shm 05] améliore les temps de réponse. L'algorithme PERM (*PERM - Pruned enriched Rosenbluth method*) semble être le plus rapide. Il lui suffit, souvent, que d'une seconde pour trouver l'optimum dans plusieurs séquences (séquences 1, 2, 4, 6 et 7). Cependant, il ne trouve pas de solution optimale pour la séquence 8. Les colonies de fourmis ne trouvent un optimum que pour les séquences inférieures à 64 acides aminés.

N° Séq.	Lg.	Nb_contacts	MC [Ung 93]	EMC [Lia 01]	AG [Ung 93]	ACO [Shm 03]		ACO [Shm 05]		PERM [Hsu 03]		AG+Tabou [Jia 03]	GTabu [Les 03]	
			Sol.	Sol.	Sol.	Sol.	T_{CPU} (s)	Sol.	T_{CPU} (s)	Sol.	T_{CPU} (s)	Sol.	Sol.	T_{CPU} (s)
1	20	9	9	9	9	9	3.33	9	< 1	9	0.01	9	NT	
2	24	9	9	9	9	9	2.52	9	<1	9	0.02	9	NT	
3	25	8	8	8	8	8	10.62	8	<1	8	80.01	8	NT	
4	36	14	13	14	12	14	11.81	14	4	14	0.05	14	NT	
5	48	23	20	23	22	23	405.79	23	60	23	1762.69	23	NT	
6	50	21	21	21	21	21	4952.92	21	15	21	0.48	21	NT	
7	60	36	33	35	34	36	62471.24	36	1200	36	0.52	35	NT	
8	64	42	35	39	37	42	5844.93	42	5400	42	6.07	39	42	<1800
9	85	53	NT	NT	NT	51	21901.34	53	< 1 jour	53	31.95	NT	53	<9000
10	100	50	NT	NT	NT	47	29707.22	49	12 heures	50	19962	NT	50	<30600
11	100	48	NT	NT	NT	47	10835.51	47	10 heures	48	152.71	NT	48	<48600

Tableau 3.4. Quelques résultats de la littérature.

Avec Lg : longueur de la protéine; MC : Monte-Carlo; EMC : Monte-Carlo Evolutionnaire; AG : algorithme génétique; ACO : Colonies de fourmis; PERM : Pruned enriched Rosenbluth method ; Tabou : méthode Tabou et NT : Non Testé.

3.8.2. Expérimentation des modèles de programmation par contraintes proposés

Le tableau 3.5 présente les meilleurs résultats obtenus avec la PPC de base sans recherche locale et la PPC de base avec recherche locale. Quant au tableau 3.6, il expose les meilleurs résultas de la PPC étendue avec la recherche locale.
Il est important de rappeler que la recherche locale utilisée dans cette coopération est basée sur le voisinage "mouvement d'attraction" présenté dans la section 3.5. L'algorithme de recherche locale est appliqué aux feuilles de l'arbre de recherche de la programmation par contraintes. Ainsi, lors des itérations suivantes de la programmation par contraintes, des branches de l'arbre de recherche sont élaguées si la valeur trouvée est plus grande que celle retournée après la recherche locale.
Pour des raisons de temps de calcul, il a parfois été nécessaire d'interrompre la PPC avant que la totalité de l'arbre de recherche ne soit exploré. Dans ce cas, ce qui figure sur le tableau est la meilleure solution trouvée au moment de l'arrêt.

	Nb. Succès	Nœuds parcourus	Profondeur de l'arbre	Nb_contacts	T_{CPU} (s)
Séquence 1 - Longueur =20 - Optimum= 9					
Sans recherche locale	7	181833	39	9	152
Avec recherche locale	5	179047	39	9	172
Séquence 2 - Longueur =24 - Optimum= 9					
Sans recherche locale	5	79790	45	9	77
Avec recherche locale	5	79790	45	9	87
Séquence 3 - Longueur =25 - Optimum= 8					
Sans recherche locale	5	1435855	47	8	1319
Avec recherche locale	5	1435855	47	8	1509
Séquence 4 - Longueur =36 - Optimum= 14					
Sans recherche locale	13	51323106	66	14	38190
Avec recherche locale	5	18514580	63	14	46071
Séquence 5 - Longueur =48 - Optimum= 23					
Sans recherche locale	5	2384395	77	16	7264
Avec recherche locale	3	27953236	82	17	82027
Séquence 6 - Longueur =50 - Optimum= 21					
Sans recherche locale	6	1044654	92	15	27509
Avec recherche locale	5	1044662	92	15	28435
Séquence 8 - Longueur =64 - Optimum= 42					
Sans recherche locale	1	273002	106	34	15673
Avec recherche locale	1	273002	106	34	16288

Tableau 3.5. Résultats avec la programmation par contraintes de base.

Nb de succés	Noeuds parcourus	Profondeur de l'arbre	Nb_ Contacts	Optimum trouvé	Tcpu(s)
Séquence 1 - Longueur=20 - Optimum=9 – "cadre"=4x5					
4	17707	27	9	oui	27
Séquence 2 - Longueur=24 - Optimum=9 - "cadre"=4x4					
2	2837	27	9	oui	5
Séquence 3 - Longueur=25 - Optimum=8 – "cadre"=3x3					
1	24	21	8	oui	0.3
Séquence 4 - Longueur=36 - Optimum=14 – "cadre"=4x4					
1	1216	29	14	oui	6
Séquence 5 - Longueur=48 - Optimum=23 – "cadre"=5x5					
1	6423	47	23	oui	53
Séquence 6 - Longueur=50 - Optimum=21 – "cadre"=10x5					
6	6675	58	16	non	2172
Séquence 7 - Longueur=60 - Optimum=36 – Frame=8x7					
1	7759	60	33	non	3325
Séquence 8 - Longueur=64 - Optimum=42 – "cadre"=8x6					
7	7650	59	36	non	5446
Séquence 9 - Longueur=85 - Optimum= 53 – "cadre"=8x8					
1	50326	80	51	non	3377

Tableau 3.6. Résultats avec la programmation par contraintes étendue et la recherche locale.

Non seulement le nombre de nœuds parcourus a diminué par rapport à la version simple de la programmation par contraintes, mais le temps et le nombre de solutions trouvées avant l'optimum également. On remarquera la rapidité de l'algorithme pour les séquences qui ne possèdent pas de P-singletons. En effet, leur placement sur la grille rend les calculs plus longs, tandis que l'on sait où se placent les autres types de monomères. D'où ces bons résultats au niveau du temps pour les séquences 3, 4 et 5 (voir séquence 7, même si on n'atteint pas l'optimum pour cette séquence). Par contre, la séquence 6 possède plusieurs séquences de motifs "HP", ce qui provoque apparemment un plus grand temps de calcul afin de trouver la meilleure conformation.
On constate que le voisinage "mouvement d'attraction" n'apporte que très peu à la programmation par contraintes. La raison semble être due à la compacité de la solution initiale. En effet, seuls les monomères situés en bordure de la protéine peuvent se déplacer, ce qui limite l'effet de la recherche locale.

3.8.3. Coopération colonie de fourmis et programmation par contraintes

La première série de tests réalisés fixe les paramètres des algorithmes des colonies de fourmis développés. En effet, en l'absence de résultats théoriques exploitables, on ne peut échapper à un réglage empirique de ces paramètres.
Les paramètres à fixer sont donc :
- α, l'influence du taux de phéromone sur la solution.
- β, l'influence du nombre de contacts éventuels.

- γ, l'influence de la position du monomère par rapport au "cadre".
- δ, l'influence de la distance du monomère par rapport au "cadre".
- ε, le taux d'évaporation.
- le nombre de fourmis.
- le nombre d'itérations.

Pour procéder à ces réglages, on utilise 8 fourmis, un taux d'évaporation de 0,15 et 100 itérations. Ces tests ont porté sur chaque séquence d'acides aminés, comme suit :

- on fixe les valeurs γ et δ à 0, pour déterminer α et β. On effectue une dizaine de fois chaque test, avec α et β en valeurs entières comprises entre 0 et 5 inclus.
- Puis, en conservant les meilleurs valeurs de α et β (respectivement 1,0 et 5,0), on teste désormais γ, toujours en valeurs entières comprises entre 0 et 5 inclus.
- on fait de même avec δ.
- Finalement, on fait varier le nombre de fourmis et le nombre d'itérations.

En conclusion les valeurs des paramètres retenus sont celles qui retournent les meilleures solutions pour chaque séquence d'acides aminés :
α=1,0, β=5,0, γ=3,0, δ=5,0, ε=0,15, nombre de fourmis = 40 et nombre d'itérations = 400.

3.8.3.1. Coopération entre la programmation par contraintes de base et les colonies de fourmis

Les tests réalisés portent sur des séquences de petite et grande taille. Les résultats trouvés sont consignés dans le tableau 3.7.

Instances	Lg.	Nb_contacts	PPC seule		Colonies de fourmis seules sans "cadre"		PPC+ Colonies de fourmis sans "cadre"	
			Sol.	T_{CPU} (s)	Sol.	T_{CPU} (s)	Sol.	T_{CPU} (s)
1	20	9	**9**	152	**9**	14	**9**	17
2	24	9	**9**	77	**9**	20	**9**	15
3	25	8	**8**	1319	**8**	549	**8**	426
4	36	14	**14**	46071	14	50	14	47
5	48	23	16	7264	16	214	17	262
6	50	21	15	27509	14	128	15	240
7	60	36	30	36760	31	39	30	49
8	64	42	34*	> 15h	25	41	39	345
9	85	53	37*	> 15h	37	48	38	114
10	100	50	33*	> 15h	30	198	36	336
11	100	48	32*	> 15h	32	405	32	555

Tableau 3.7. Résultats de la coopération entre la programmation par contraintes de base et les colonies de fourmis sans "cadre". Le symbole "*" veut dire que l'exécution est interrompue après 15 heures.

On constate que les résultats trouvés sont, en général, pour les grandes instances, loin des optimums de la littérature. Ceci est en effet dû à la présence de groupe (sous-séquence) de

monomères P et au fait que l'utilisation de la programmation par contraintes "enferme", en quelque sorte, ces monomères P à l'intérieur de la protéine, ce qui crée des cavités, alors que cette conformation est à éviter.

Les résultats du tableau 3.7 révèlent que les résultats obtenus avec l'algorithme faisant coopérer la programmation par contraintes et les colonies de fourmis sont meilleurs que ceux obtenus par ces deux algorithmes exécutés séparément. De même, les temps CPU sont moins importants que ceux de l'algorithme de programmation par contraintes et légèrement plus importants que ceux obtenus par les colonies de fourmis, ce qui est normal. Ainsi ces résultats montrent bien l'apport de la stratégie qui permet de guider la programmation par contraintes par les colonies de fourmis et l'amélioration des résultats trouvés par ces deux algorithmes pris séparément.
Il est aussi important de remarquer que les résultats obtenus, pour les instances de grande taille (longueur supérieure à 50 monomères), sont moins bons que ceux de la littérature PERM et les deux versions d'ACO (tableau 3.4). La contre performance de l'algorithme des colonies de fourmis seul sur les grandes instances est probablement liée à l'exploitation du langage de programmation Oz et probablement due aussi à la manière dont la mise à jour de la phéromone est implémentée.

3.8.3.2. Coopération entre la programmation par contraintes étendue et les colonies de fourmis avec "cadre"

Les tests ont été effectués avec les paramètres suivants :
Pour les séquences 1 à 5 et 7 : α=1,0, β=5,0, γ=3,0, δ=5,0, ε=0,15, nombre de fourmis=40, nombre d'itérations=400. Le nombre d'itérations=1000 pour les autres séquences.
Les résultats des tests effectués sont résumés dans le tableau 3.8 :

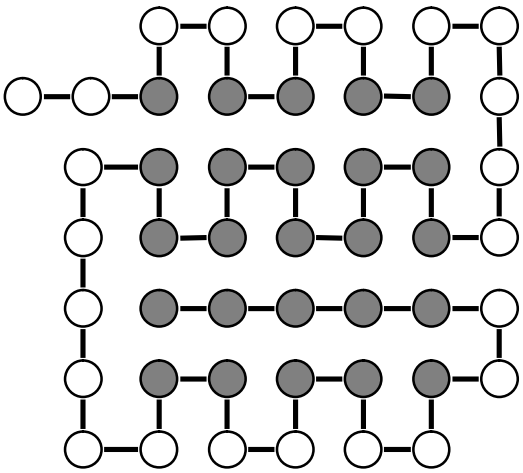
n° séq.	Lg.	Nb_contacts	Nb_contacts obtenu par C. Fourmis + PPC étendue sans la recherche locale	T_{cpu} (s)
1	20	9	9	25
2	24	9	9	27
3	25	8	8	27
4	36	14	14	30
5	48	23	23	31
6	50	21	21	>10h
7	60	36	35	>10h
8	64	42	42	≈10h
9	85	53	48	>10h
10	100	50	44	>10h

11	100	48	42	>10h

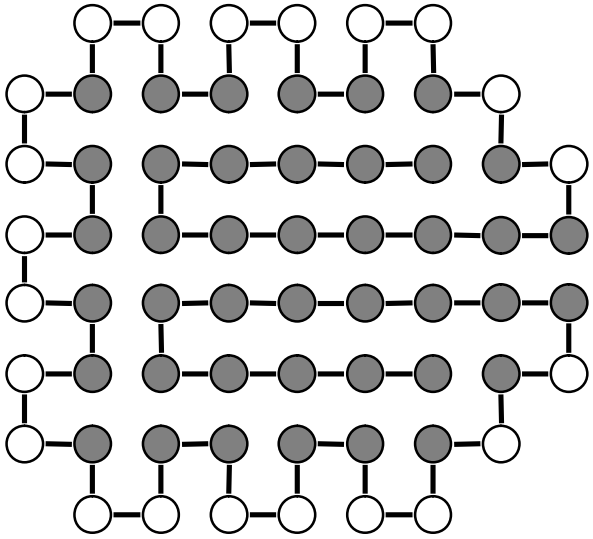
Tableau 3.8. Résultats de la coopération entre la programmation par contraintes étendue et les colonies de fourmis avec "cadre".

On remarque que, bien souvent, les colonies de fourmis trouvent une solution assez proche de la solution optimale. La programmation par contraintes permet donc de replacer les quelques monomères qui ne sont pas placés de manière légale (les H hors du "cadre" par exemple). Pour toutes les instances, cet algorithme arrive, après au moins 10h de temps de calcul, à trouver l'optimum, sauf pour les séquences, de grande taille (≥ 85), 9, 10 et 11 où la déviation est de l'ordre de 5 contacts. Par contre, pour la séquence 7, l'écart est de 1 contact. On constate aussi que le résultat de la séquence 9 avec l'algorithme utilisant la programmation par contraintes étendue et les colonies de fourmis, mais sans la recherche locale (tableau 3.8), est moins bon que celui obtenu en appliquant la programmation par contraintes étendue avec de la recherche locale (tableau 3.6). Ceci peut s'expliquer par le rôle intensificateur de la recherche locale.

La figure 3.29 donne un exemple de conformations obtenues pour les séquences 6 et 8 avec l'algorithme de coopération entre la programmation par contraintes et les colonies de fourmis :



Séquence 5



Séquence 8

Figure 3.29. Exemple de conformations obtenues par la coopération entre PPC étendue et colonies de fourmis avec "cadre".

3.9. Conclusion

Le problème du repliement de protéines est NP-complet pour le modèle H-P [Ber 98]. Il consiste, étant donnée une protéine, à chercher sa conformation de moindre énergie.
L'objectif de ce travail était de proposer une approche de résolution pour ce problème basée sur la programmation par contraintes.
Deux modèles de programmation par contraintes ont été proposés. Le premier, simple et de base, s'est révélé être insuffisant, même hybridé avec une recherche locale basée sur le voisinage "mouvement d'attraction". Nous avons alors utilisé un autre modèle, plus élaboré. Ce modèle introduit de nouvelles contraintes, basées sur la notion de "cadre". Il ne donne des résultats satisfaisants que pour les protéines de taille inférieure à 50. Pour pallier à ce manque, nous avons proposé un schéma de coopération entre la programmation par contraintes et les colonies de fourmis. Cette coopération permet une meilleure recherche de la solution en réduisant l'espace de recherche exploré par la programmation par contraintes. Les résultats de cette coopération entre une métaheuristique et la programmation par contraintes sont satisfaisants. Le temps de calcul de la programmation par contraintes est d'autant plus réduit que la recherche par colonies de fourmis est de qualité.

Partie 2

Problèmes d'optimisation multiobjectif

Cette partie est consacrée à la présentation des concepts de base de l'optimisation multiobjectif. Un accent est mis sur les algorithmes génétiques, utilisant des approches dites Pareto. Cette partie s'achève avec la présentation de deux travaux dans le domaine. En effet, au travers des problèmes de "flow-shop" et de routage de véhicules, nous présentons les différents algorithmes et mécanismes proposés pour leur résolution dans le cas bi-objectif.

Chapitre 4

Etude comparative de différentes techniques d'optimisation multiobjectif pour le problème du *flow-shop* bi-objectif

Les problèmes d'ateliers tels que le "flow-shop" ou le "job-shop" sont des problèmes cruciaux pour le domaine industriel. Les critères à optimiser sont généralement la minimisation du temps de terminaison ou celui des retards des tâches. Cependant, rares sont les approches de résolution qui prennent en compte différents critères à la fois.
Ce chapitre présente une étude comparative et progressive de différentes techniques d'optimisation multiobjectif. Cette étude utilise les algorithmes génétiques adaptés au cas multiobjectif et évalue diverses stratégies de sélection, de maintien de la diversité et d'hybridation avec la recherche locale. Nous proposons aussi un modèle parallèle. Celui-ci permet l'augmentation de la taille de la population et le nombre limite de générations et permet ainsi l'amélioration de certains résultats. Parallèlement à ce travail d'optimisation, nous proposons de nouveaux benchmarks bi-objectifs pour le problème de "flow-shop".

Les travaux décrits dans ce chapitre ont fait l'objet d'une présentation à la conférence MOSIM'01 [Mab 01], à la conférence EMO'01 [Tal 01] et d'une publication dans le Journal of Combinatorial Mathematics and Combinatorial Computing : JCMCC [Rah 05].

4.1. Introduction

Les problèmes d'ordonnancement de type *flow-shop* sont l'objet de nombreuses études [Gon 78][Raj 92][Raj 95][Ben 98][Bil 99]. Les méthodes de résolution adoptées varient entre les méthodes exactes, telles que la méthode par *Séparation & Evaluation* [Bra 91], les heuristiques [Sri 96][Bil 99] et les métaheuristiques [Sri 95][Ben 98][Mur 98][Now 99][Ade 04]. Cependant, la plupart de ces travaux concernent le problème dans sa forme monoobjectif, avec pour objectif principal la minimisation de la date de fin d'exécution des tâches (*makespan*) [Raj 92][Raj 95][Say 99].

Dans ce chapitre nous menons une étude expérimentale concernant la résolution d'un problème du *flow-shop* bi-objectif à l'aide d'un algorithme génétique.
Les algorithmes génétiques se sont illustrés par une grande efficacité face aux problèmes d'optimisation combinatoire [Hol 75][Haj 92][Hol 92][Kök 03]. L'extension de cette métaheuristique au cas multiobjectif a été élaborée de différentes façons [Fuj 98][Siv 98][Tal 99][Vel 99][Zit 99][Tki 00][Tki 02a].
La difficulté du cas multiobjectif réside dans l'absence d'une relation d'ordre totale qui lie l'ensemble des solutions du problème. Sur le plan des algorithmes génétiques, cela se traduit par la difficulté de concevoir un opérateur de sélection qui affecte à chaque individu une probabilité de sélection proportionnelle à la performance de cet individu. Un autre inconvénient est lié à la perte prématurée de la diversité. On parle de perte de la diversité ou de convergence prématurée, lorsque les individus de la population se ressemblent. Ainsi, les populations suivantes deviennent de plus en plus homogènes, l'évolution de la population est alors équivalente à l'évolution d'un individu et il y a enlisement de la recherche suite à la découverte du plus proche optimum local. D'où la nécessité de concevoir des techniques de maintien de la diversité au sein de la population.

La section 4.2 de ce chapitre donne un bref état de l'art sur l'optimisation multiobjectif. La section 4.3 est consacrée à la présentation du problème du *flow-shop* multiobjectif. Nous discutons des différents paramètres à optimiser ainsi que des contraintes à satisfaire. Dans la section 4.4, nous exposons les différentes possibilités d'extension des algorithmes génétiques au cas multiobjectif. Dans la section 4.5, différentes stratégies de sélection sont présentées et leurs performances comparées. Lors de la section 4.6, nous discutons des méthodes de maintien de la diversité ainsi que leurs apports sur la qualité des solutions. La section 4.7 est consacrée à l'hybridation de notre algorithme génétique multiobjectif avec la recherche locale. La section 4.8 compare notre algorithme à d'autres travaux de la littérature. Pour cela nous proposons une manière d'engendrer des benchmarks pour le problème du *flow-shop* bi-objectif. Dans la section 4.9, nous exploitons la possibilité de parallélisation des algorithmes génétiques. La dernière section est dédiée à l'introduction d'une nouvelle approche de génération de la population initiale basée sur l'heuristique *NEH* [Naw 83]. Avec cette heuristique modifiée, nous espérons améliorer davantage la qualité des solutions obtenues par les différents algorithmes génétiques considérés.

4.2. Généralités sur l'optimisation multiobjectif

De nombreux domaines de l'industrie, comme les télécommunications, la logistique ou les transports, sont confrontés à des problèmes d'optimisation complexes. La qualité d'une solution est le plus souvent exprimée par une fonction mathématique qui modélise

typiquement le mérite de la solution. Or, les problèmes sont rarement monoobjectifs mais multiobjectifs, c'est-à-dire qu'il est possible, pour un même problème, de définir plusieurs objectifs pertinents, mais souvent contradictoires. Un objectif est dit contradictoire avec un autre objectif si une amélioration du premier peut entraîner une dégradation du second.

En optimisation multiobjectif, on cherche à concevoir des méthodes efficaces pour la résolution de problèmes où tous les objectifs sont pris en compte. Cela signifie le plus souvent trouver un ensemble de solutions qui réalisent de bons compromis entre les différents objectifs, plutôt qu'une solution unique.

4.2.1. Définitions

Dans cette section, nous commençons par définir quelques notions de l'optimisation multiobjectif, puis nous présentons les différentes méthodes de résolution existantes dans ce domaine. On mettra l'accent, tout au long de ce chapitre, sur la notion d'optimisation Pareto avec les algorithmes génétiques pour des problèmes de minimisation.

4.2.1.1. Problème d'optimisation multiobjectif (PMO)

Définition 1. Dans un problème d'optimisation multiobjectif, on cherche à optimiser un vecteur de fonctions objectif :

$$(\text{POM}) = \begin{cases} \min F(x) = (f_1(x), f_2(x), ..., f_n(x)) \\ \text{t.q. } x \in \Omega, \end{cases}$$

où $n \geq 2$ est le nombre de fonctions objectif, $F(x) = (f_1(x), \ldots, f_n(x))$ est le vecteur des fonctions à optimiser et Ω représente l'ensemble des solutions réalisables que l'on appelle ensemble réalisable. Nous avons considéré ici, sans perte de généralité, que toutes les fonctions objectif devaient être minimisées. En général on peut chercher à minimiser ou à maximiser chacune des fonctions objectifs indépendamment les unes des autres.

4.2.1.2. Solution d'un problème d'optimisation multiobjectif

L'optimisation multiobjectif consiste à optimiser plusieurs composantes d'un vecteur fonction coût, chaque composante de ce vecteur correspondant à un objectif. Pour les solutions de problèmes multiobjectifs, la relation d'ordre n'est pas totale (une solution peut être meilleure qu'une autre sur certains objectifs et moins bonne sur les autres). Il convient alors de manipuler des populations de solutions dites Pareto-optimales et de définir la notion de Pareto-optimalité. Il convient aussi de définir d'autres notions s'y rapportant. En effet, il est possible de définir une relation d'ordre partiel, appelée relation de dominance ou relation de dominance Pareto, entre les solutions :

Définition 2. Une solution y domine une solution z, notée $y \preceq z$, si et seulement si

$$\forall\, i \in \{1, \ldots, n\},\ f_i(y) \leq f_i(z) \ \text{ et } \exists\ i \in \{1, \ldots, n\} \text{ tel que } f_i(y) < f_i(z).$$

Autrement dit, la solution y est au moins identique à z en regard de tous les objectifs et elle est meilleure que z au moins pour un objectif.

Dans l'exemple de la figure 4.1, les solutions A, B, C, D, E ne sont pas dominées par d'autres solutions. Par contre la solution G est dominée par les solutions B et C et la solution H est dominée par les solutions C et D. Les solutions A, B, C, D, E sont incomparables, c'est-à-dire que d'un point de vue multiobjectif, il n'est pas possible d'établir une préférence pour une solution plutôt que pour une autre.

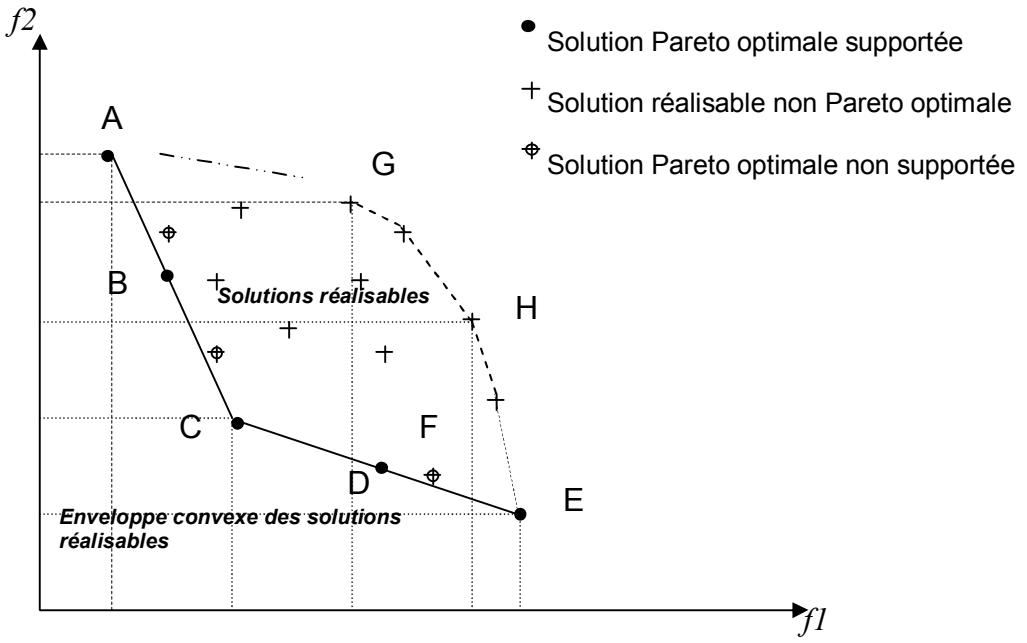


Figure 4.1. Exemple de dominance et de solutions supportées/non supportées.

Définition 3. Pour un problème d'optimisation multiobjectif donné, l'ensemble Pareto optimal, noté $\mathcal{PO}^*$, est défini comme suit :

$$\mathcal{PO}^* = \{x \in \Omega \mid \nexists\, x' \in \Omega, \quad x' \preceq x\}.$$

Remarquons qu'une autre définition équivalente est celle-ci :

Définition 4. Une solution $x^* \in \Omega$ est dite Pareto optimale si, pour chaque $x \in \Omega$, l'une des deux propositions suivantes est vraie :

i) $\forall\, i \in \{1, \ldots, n\}, \quad f_i(x) = f_i(x^*)$,

ii) $\exists\, i \in \{1, \ldots, n\}, \quad f_i(x) > f_i(x^*)$.

Cette définition peut s'interpréter de la manière suivante : une solution x^* est Pareto optimale s'il n'existe pas une solution réalisable x qui améliore la valeur d'un objectif sans détériorer au moins celle d'un autre objectif.
L'exemple de la figure 4.1 montre, par exemple, que les solutions G et H ne sont pas Pareto optimales et que A, B, C, D et E sont des solutions Pareto optimales.

L'ensemble Pareto optimal peut être divisé en deux sous-ensembles : l'ensemble des solutions supportées et l'ensemble des solutions non supportées (figure 4.1). Les solutions supportées sont les sous ensembles des solutions Pareto optimales qui appartiennent à l'enveloppe convexe de l'ensemble des points atteignables dans l'espace objectif. Les solutions Pareto n'appartenant pas à cette enveloppe convexe sont dites non supportées. Ainsi, dans l'exemple de la figure 4.1, les solutions supportées sont A, B, C, D, et E tandis que F est une solution non supportée.
L'ensemble des vecteurs coûts correspondant aux solutions Pareto optimales est appelé la frontière Pareto ou surface de compromis. Formellement, la frontière Pareto se définit comme suit :

Définition 5. Pour un problème d'optimisation multiobjectif donné et un ensemble Pareto optimal $\mathcal{PO}^*$, la frontière Pareto, notée PF^* est définie comme suit :

$$PF^* = \{u = (f_1(x), \ldots, f_n(x)) \mid x \in \mathcal{PO}^*\}.$$

En général, il est peu évident de trouver une expression analytique de la frontière Pareto. De plus les problèmes étudiés, par la suite, étant NP-difficiles dans leur version monoobjectif, ils le sont également dans leur version multiobjectif. Enfin, la taille de l'ensemble Pareto optimal peut être exponentielle, ce qui rend nécessaire de rechercher une approximation de cet ensemble.
Signalons que seul le cas où chaque objectif doit être minimisé est considéré par la suite.

4.2.2. Approches de résolution : l'approche Pareto avec les algorithmes génétiques

Des méthodes exactes telles que la méthode par *Séparation & Evaluation* [Say 99], l'algorithme *A** [Ste 91][Man 96] et la programmation dynamique [Whi 82], à l'origine conçues pour des problèmes monoobjectifs, ont été adaptées au cas multiobjectif. Cependant, ces approches sont destinées à des problèmes de petite taille, et leur efficacité est mise en question, dès que la taille du problème augmente et que le nombre de critères croît.

Les méthodes approchées, telles que les métaheuristiques, tentent d'approcher l'ensemble des solutions optimales. Plusieurs adaptations de métaheuristiques ont été proposées dans la littérature pour le cas multiobjectif et la détermination des solutions Pareto optimales : le recuit simulé [Ulu 93], la recherche tabou [Gan 97] et les algorithmes évolutionnaires (algorithmes génétiques et les stratégies évolutionnaires [Fon 95][Sri 95]). Ces méthodes sont applicables à une large gamme de problèmes et leur efficacité demeure relativement bonne quand la taille et le nombre d'objectifs augmentent.

Nous avons choisi d'utiliser les algorithmes génétiques pour approcher la courbe de Pareto. Ce choix est motivé par le souhait de fournir au décideur un ensemble de solutions parmi lesquelles il pourrait choisir celle qui convient à ses préférences. De plus, les algorithmes génétiques s'adaptent bien à l'approche Pareto du fait qu'ils utilisent une population d'individus.

La conception d'un algorithme génétique pour un problème multiobjectif nécessite de définir une manière d'évaluer les individus et la phase de sélection. En effet, la phase

d'évaluation doit prendre en compte l'ordre partiel défini par la relation de dominance, en affectant un rang à chaque individu. Quant à la phase de sélection, le maintien de la diversité de la population et l'élitisme sont primordiaux.

4.2.2.1. Aperçu sur les algorithmes génétiques

Les algorithmes génétiques sont inspirés des mécanismes d'évolution des populations d'organismes biologiques. Dans la nature, selon les principes de sélection naturelle décrits par Charles Darwin, les êtres vivants sont en compétition pour se nourrir, comme pour se reproduire. Ainsi, au fil des générations, les organismes les "plus forts" (les mieux adaptés) se nourrissent et se reproduisent mieux et finissent par s'imposer en nombre dans la population. Au milieu des années 1970, John Holland dégage de son étude des processus d'adaptation naturelle une théorie de l'adaptation [Hol 75]. Il s'appuie sur cette théorie pour construire des systèmes artificiels capables d'adaptation : c'est la naissance de l'algorithmique génétique. Aujourd'hui, les algorithmes génétiques montrent leur efficacité dans la résolution des problèmes d'optimisation combinatoire, parfois associés à d'autres métaheuristiques. L'algorithme génétique de base peut être écrit comme suit (figure 4.2) :

```
Algorithme : AG_générique
Début
Génération de la population initiale
Evaluation des individus
Tant que  <condition de terminaison non atteinte>
      faire
        sélection des individus
        croisement et mutation des individus
        évaluation de nouveaux individus
        remplacement (mise à jour de la population)
      fait
Fin.
```

Figure 4.2. Algorithme génétique générique.

Un algorithme génétique travaille avec une population de solutions, encore appelées individus, chromosomes ou phénotypes. Dans un algorithme génétique de base, le génotype (codage des individus) est une chaîne de bits. Un individu est donc une suite de valeurs binaires. Cependant, le codage des individus est un facteur prépondérant de l'efficacité d'un algorithme génétique. Ainsi, pour de nombreux problèmes, le codage basé sur l'alphabet binaire n'est pas efficace ; donc, on utilise un codage mieux adapté, spécifique au problème.

Pour que l'algorithme génétique soit efficace, la population initiale (souvent construite aléatoirement) doit présenter une grande diversité de phénotypes, afin d'augmenter la probabilité d'avoir dans la population les caractères (allèles) d'un individu de bonne qualité. Par conséquent, la taille (nombre d'individus) de la population est un paramètre déterminant pour un algorithme génétique.

D'une manière générale, la sélection est un mécanisme stochastique. Une fonction d'évaluation, dénommée fonction mérite, donne le niveau d'adaptation d'un individu. La sélection détermine la probabilité de reproduction d'un individu en fonction de son mérite. Un

individu obtient une probabilité de reproduction d'autant plus grande qu'il est bien adapté. Par exemple, la probabilité de reproduction peut être une fonction linéaire du mérite. C'est la méthode de la "roulette", appelée ainsi car on sélectionne les candidats à la reproduction à l'aide d'une "roulette de casino" sur laquelle chaque individu est représenté par un secteur dont l'angle est proportionnel à son mérite. Une fois les probabilités établies, les candidats à la reproduction sont tirés aléatoirement pour constituer la population des individus sélectionnés.

Dans un algorithme génétique, les opérateurs de reproduction, le croisement et la mutation opèrent sur les individus sélectionnés pour constituer de nouveau individus. La recombinaison correspond à la reproduction sexuée du monde naturel. Ainsi, pour la recombinaison, les individus sont associés par paires. On peut imaginer de nombreuses façons de grouper les individus en paires ; cependant, J. Holland préconise l'association d'un bon individu avec un autre choisi au hasard (uniformément). Quand les individus sont appariés, chaque paire peut être soumise à la recombinaison, selon une certaine probabilité, qui est un paramètre de l'algorithme génétique. La recombinaison, transforme deux individus en deux autres individus. Parmi les opérateurs de recombinaison fréquemment utilisés, on trouve les croisements, dont le principe consiste à échanger certaines fractions de phénotype. Pour le croisement à 1 point, par exemple, un point de coupe est défini au hasard ; puis chaque génotype est coupé ; et les parties coupées sont échangées.

La mutation, généralement appliquée après la recombinaison, pendant la phase de transformation, affecte légèrement le génotype des individus. Elle peut modifier chaque individu selon une probabilité qui est un paramètre de l'algorithme génétique. La mutation, quand elle s'applique, change la valeur d'un bit choisi au hasard dans le génotype de l'individu.

Le remplacement termine le cycle de l'algorithme génétique pour produire une nouvelle génération des solutions d'individus. La nouvelle génération peut occulter complètement les parents ou bien elle ne peut remplacer que quelques parents. D'autres techniques existent, comme par exemple la préservation dans la nouvelle population des k individus les plus performants. Pour la méthode stable (*steady-state*) seuls un ou deux individus sont remplacés à chaque génération.

4.2.2.2. Méthodes d'attribution du rang des solutions ("Ranking")

L'objectif de ces méthodes est d'associer un *rang* à chaque individu de la population. Par la suite, certaines de ces méthodes de rangement sont utilisées dans le cadre multiobjectif et le *mérite* d'un individu est égal à son *rang*.

a/ La méthode NSGA (*Nondominated Sorting Genetic Algorithm*)

Cette procédure a été initialement proposée par Goldberg [Gol 89] et réutilisée par *Srinivas* et *Deb* [Sri 94]. Tous les individus non dominés de la population possèdent le rang 1. Ces individus sont ensuite enlevés de la population, et l'ensemble suivant d'individus non dominés est identifié et on leur attribue le rang 2. Ce processus est itéré, jusqu'à ce que tous les individus de la population aient un rang. Cette méthode a été utilisée dans les algorithmes génétiques pour la résolution de plusieurs problèmes multiobjectifs : arbre recouvrant minimum [Zho 99], conception de réacteurs [Par 98], distribution de l'eau, etc.

b/ La méthode NDS (*Nondominated Sorting*)

En 1993, *Fonseca* et *Fleming* [Fon 95] proposèrent une procédure basée sur la notion d'optimalité Pareto. Dans cette procédure, le rang d'un individu est égal au nombre de solutions qui le dominent dans la population plus un.

Un individu non dominé de la population possède donc le rang 1. Les rangs calculés avec la méthode NDS sont toujours supérieurs à ceux calculés avec la méthode NSGA. Cette méthode induit donc une plus forte pression de sélection.
Cette méthode a été utilisée dans les algorithmes génétiques pour plusieurs problèmes multiobjectifs : gestion de containers, conception d'ailes d'avion, et la synthèse de systèmes distribués embarqués [Tal 99].

c/ La méthode WAR (*Weighted Average Ranking*)

De façon identique aux méthodes NDS et NSGA, cette approche consiste à ordonner les individus de la population selon leurs rangs. Les rangs des individus sont calculés en formant plusieurs listes ordonnées des individus, une par fonction objectif. Le rang d'un individu revient alors à la moyenne de ses rangs par rapport à chaque objectif.

4.2.2.3. Sélection

Plusieurs méthodes de sélection existent.

a/ La méthode de sélection VEGA (*Vector Evaluated Genetic Algorithm*)

Schaffer proposa en 1985 [Zit 99] un algorithme génétique multiobjectif appelé *VEGA* (*Vector Evaluated Genetic Algorithm*). La technique, aussi connue sous l'appellation de sélection parallèle opère par traitement séparé des différents critères. À chaque génération, la population est subdivisée en *n* sous-populations de tailles égales. À chaque sous-population est associée une fonction objectif à optimiser. Les individus d'une même sous-population sont alors sélectionnés suivant uniquement la fonction objectif qui leur est affiliée, en utilisant des méthodes de sélection monoobjectif. Parmi celles-ci, citons, par exemple :

- la roulette, qui est une méthode dans laquelle chaque individu a une probabilité d'être sélectionné proportionnelle à son mérite.
- la sélection par tournois. Dans cette méthode, deux individus sont choisis au hasard et leurs fonctions mérite sont comparées : l'individu le plus adapté est alors sélectionné.

Pour plus de détails, à ce sujet, voir le livre de Goldberg [Gol 94].

b/ La méthode de sélection NPGA (*Niched Pareto Genetic Algorithm*)

Cette technique, proposée par *Horn*, *Nafpliotis* en 1993 et *Goldberg* en 1994 [Gol 94], combine les principes de la sélection par tournoi et de la dominance.
Quand le procédé de sélection par tournoi n'arrive pas à départager les deux individus en compétition, le principe de partage (*sharing)* est alors utilisé. Il consiste à calculer, pour chacun des deux compétiteurs, le nombre d'individus déjà sélectionnés qui leur ressemblent (dans l'espace objectif). L'individu appartenant à une niche de population plus réduite est finalement sélectionné.

4.2.2.4. Elitisme

Le principe de l'élitisme consiste à maintenir une deuxième population de solutions qui contient les meilleures solutions trouvées tout au long de la recherche. A chaque génération, un certain pourcentage des individus de la population sont remplacés par des membres de la population élite. Ces membres sont soit choisis de façon aléatoire, ou suivant un autre critère, comme la période pendant laquelle la solution est restée élite. Pour limiter le nombre de solutions élites, d'autres mécanismes doivent être implémentés, tels que le maintien d'une certaine dissimilarité entre les individus de l'ensemble élite.
L'élitisme intervient dans le processus de sélection. Il consiste à réaliser la sélection des individus, aussi bien dans la population courante que dans la population élite [Par 98].

Dans les algorithmes génétiques multiobjectif, l'élitisme joue un rôle encore plus important que dans le cas monoobjectif et sa mise en œuvre est plus complexe [Zit 99]. En effet, au lieu d'avoir une seule meilleure solution, un ensemble de solutions non dominées sont à manier, dont la taille est comparable à une population. La notion d'élitisme dans le cas multiobjectif, suscite deux questions :

- *Population→Ensemble élite* : Quels sont les individus à maintenir dans l'ensemble élite et pour combien de temps ?

- *Ensemble élite→Population :* Quand et de quelle façon sont réintroduites les solutions élites dans la population ?

Deux approches peuvent se présenter. La première copie systématiquement dans l'ensemble élite les individus de la population non dominés. Une variante plus restrictive consiste à ne tenir compte que des n individus ayant un vecteur objectif qui maximise l'un des n critères.

4.2.2.5. Maintien de la diversité

Dans la résolution d'un problème multiobjectif, il est nécessaire que les solutions trouvées soient Pareto optimales, mais aussi qu'elles soient uniformément réparties dans le sous-espace des solutions Pareto optimales. Pour que ces solutions soient uniformément réparties, il faut utiliser des méthodes de maintien de la diversité. Parmi celles-ci citons : les *niches écologiques*, les fonctions de partage (*sharing*), le *crowding*, les *modèles parallèles* et la *restriction de voisinages* [Tal 99].
Dans le cadre de cette thèse, on s'intéresse particulièrement à la méthode de *partage*, qui a été utilisée dans nos approches de résolution.
La fonction de *partage* sert à déterminer le *voisinage* et *le degré de partage* pour chaque individu de la population. Le partage dégrade le mérite d'un individu par rapport au nombre d'individus similaires dans la population. Le mérite d'un individu x est égal à la fonction mérite f, divisée par son compteur de niche.

$$f'(x) = f(x)/m(x)$$

où $m(x)$ représente le compteur de niche pour un individu x. Le compteur de niche est la somme des fonctions de partage (sh) entre l'individu x et tous les autres individus de la population. La fonction sh dépend de la distance $dist$ entre deux individus de la population. Elle vaut 1 si deux individus sont identiques, 0 s'ils sont différents (à partir d'un seuil donné), et une valeur intermédiaire pour des niveaux de similarité intermédiaires. La distance $dist$ peut être dans l'espace de décision (*distance génotypique*) ou dans l'espace objectif (*distance phénotypique*) [Fon 95], et dépend généralement du problème traité. Une méthode combinant

une distance génotypique et une distance phénotypique a été utilisée dans [Row 96][Mab 01]. Soient $d_1(x_i,x_j)$ la distance de Hamming entre deux individus x_i et x_j, et $d_2(x_i,x_j)$ la mesure de la distance dans l'espace objectif. La distance d_2 représente la somme des différences de coûts correspondant à chaque dimension :

$$d_2(x_i, x_j) = \sum_{k=1}^{n} \left| f_k(x_i) - f_k(x_j) \right|.$$

On a alors

$$m(x) = \sum_{y \in pop} sh(\text{dist}(x,y)).$$

Il existe plusieurs fonctions de partage. La fonction de partage classique proposée dans [Gol 87] est:

$$sh(dist(x, y)) = \begin{cases} 1 - \left(\frac{dist(x, y)}{\gamma} \right)^{\alpha} & si \quad dist(x, y) < \gamma \\ 0 \text{ sinon.} \end{cases}$$

Les paramètres γ et α sont des constantes. La constante γ spécifie le seuil de dissimilarité (taille des niches) ; si la distance entre deux individus est supérieure à γ, ils n'affectent par leur fonction de partage respectivement. La constante α, généralement initialisée à 1, permet de réguler la forme de la fonction de partage. La fonction de partage a été définie de la façon suivante :

$$sh(dist(x_i, x_j)) = \begin{cases} 1 - \frac{d_1(x_i, x_j)}{\gamma_1} & \text{si} \quad d_1(x_i, x_j) < \gamma_1,\ d_2(x_i, x_j) \geq \gamma_2 \\ 1 - \frac{d_2(x_i, x_j)}{\gamma_2}, & \text{si} \quad d_1(x_i, x_j) \geq \gamma_1,\ d_2(x_i, x_j) < \gamma_2 \\ 1 - \frac{d_1((x_i, x_j) d_2(x_i, x_j)}{\gamma_1 \gamma_2} & \text{si} \quad d_1(x_i, x_j) < \gamma_1,\ d_2(x_i, x_j) < \gamma_2 \\ 0 & \text{sinon,} \end{cases}$$

où γ_1, γ_2 correspondent respectivement à la taille des niches dans les deux espaces (décision et objectifs).

4.2.3. Métriques pour l'évaluation des performances

Il existe de nombreuses mesures pour calculer la qualité des courbes de Pareto obtenues par des métaheuristiques. Une des difficultés pour évaluer la qualité d'une courbe de Pareto provient du fait qu'il existe plusieurs mesures qui évaluent un même aspect. Cependant, comme chaque métrique présente à la fois des avantages et des inconvénients, il n'est pas évident de choisir laquelle appliquer.

Nous allons présenter, dans un premier temps, une métrique qui permet de comparer deux courbes de Pareto approchées entres elles. Il s'agit donc d'une métrique relative.
Dans un deuxième temps, nous présentons une métrique absolue, qui permet d'associer à chaque courbe de Pareto approchée une mesure de sa "qualité". Cette métrique permet donc d'ordonner entre elles un nombre quelconque de courbes de Pareto approchées selon leur "qualité".

4.2.3.1. Une métrique relative : la contribution

Cette mesure calcule la proportion de solutions d'un ensemble potentiellement Pareto optimal P_A obtenu par un algorithme A par rapport à l'ensemble P_B obtenu par un algorithme B.

Soit C l'ensemble des solutions potentiellement Pareto optimales communes à A et B :

$$C = P_A \cap P_B$$

Soit W_A (respectivement W_B) l'ensemble des solutions de P_A (respectivement P_B) qui dominent des solutions dans P_B (respectivement P_A). Soit L_A (respectivement L_B) l'ensemble des solutions de P_A (respectivement P_B) dominées par des solutions dans P_B (respectivement P_A). L'ensemble N_A (respectivement N_B) est formé des solutions de P_A (respectivement P_B) n'ayant pas de relation de dominance avec toutes les solutions de P_B (respectivement P_A). Formellement, on a la relation suivante :

$$N_A = P_A \setminus (C \cup W_A \cup L_A)$$

Soit PO^* l'ensemble des solutions *potentiellement Pareto optimales* trouvées par les deux algorithmes A et B :

$$PO^* = C \cup W_A \cup N_A \cup W_B \cup N_B$$

La contribution de l'algorithme A relativement à l'algorithme B, notée Cont(A, B), est le ratio des solutions non dominées produites par A par rapport à PO^* :

$$\text{Cont}(A,B)=\frac{\left(\frac{|C|}{2}+|W_A|+|N_A|\right)}{\left(|C|+|W_A|+|N_A|+|W_B|+|N_B|\right)} = \frac{\left(\frac{|C|}{2}+|W_A|+|N_A|\right)}{|PO^*|}$$

La contribution de l'algorithme B relativement à l'algorithme A est définie d'une manière similaire. La relation suivante est toujours vraie :

$$\text{Cont}(A, B) + \text{Cont}(B, A) = 1$$

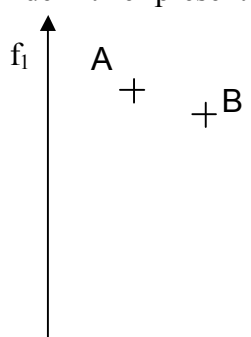
D'autre part, notons que si les deux algorithmes produisent les mêmes solutions, alors :

$$\text{Cont}(A, B) = \text{Cont}(B, A) = ½$$

tandis que si toutes les solutions produites par B sont dominées par les solutions produites par A, on a :

$$\text{Cont}(B, A) = 0$$

Cette mesure permet d'avoir rapidement une idée de l'apport d'un algorithme par rapport à un autre algorithme. Cependant, l'information fournie est moins claire que celle de la métrique C de Zitzler présentée ci-dessus.



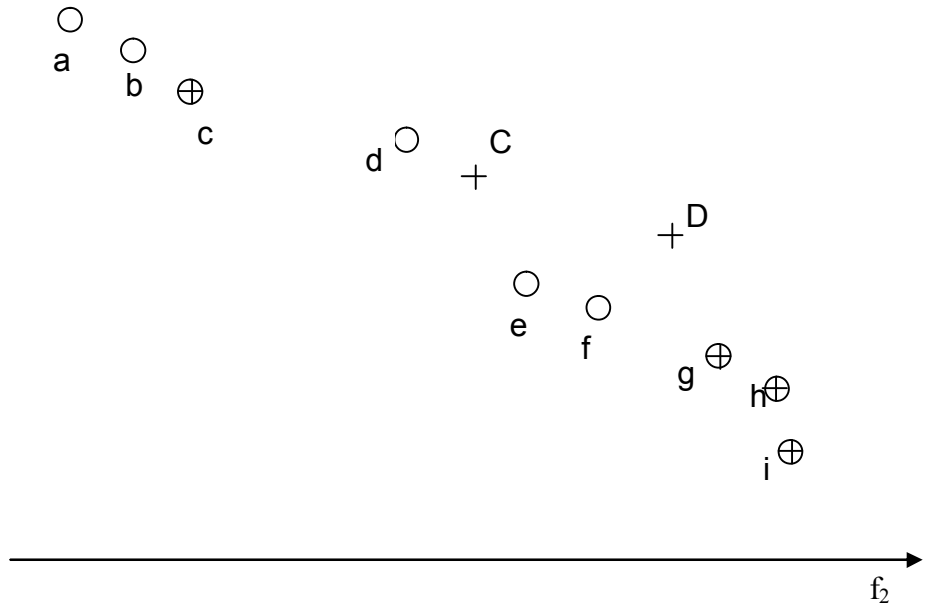


Figure 4.3. Exemple pour la métrique contribution.

L'exemple de la figure 4.3 montre comment est calculée la contribution de deux fronts de Pareto P_A et P_B. Les solutions de P_A sont représentées par des cercles et celles de P_B par des croix.

- Le front de Pareto P_A est composé de P_A ={a,b,c,d,e,f,g,h,i}
- Le front de Pareto P_B est composé de P_B={A,B,c,C,D,g,h,i}
- L'ensemble des solutions communes à P_A et P_B est égal à $C= P_A \cap P_B$ ={c,g,h,i} d'où $|C|$=4
- L'ensemble des solutions de P_A qui dominent des solutions dans P_B : W_A={a,b,e,f} $|W_A|$=4
- L'ensemble N_A des solutions de P_A n'ayant pas de relation de dominance avec toutes les solutions de P_B : N_A={d} $|N_A|$=1
- L'ensemble des solutions de P_B qui dominent des solutions dans P_A : W_B={Ø} d'où $|W_B|$=0
- L'ensemble N_B des solutions de P_B n'ayant pas de relation de dominance avec toutes les solutions de P_A : N_B={C} d'où $|N_B|$=1
- La contribution de P_A par rapport à P_B est égale à Cont (P_A,P_B)=0,7
- La contribution de P_B par rapport à P_A est égale à Cont (P_B,P_A)=0,3

4.2.3.2. Une métrique absolue : la métrique S

Soit P_A la courbe de Pareto approchée obtenue par un algorithme A. La métrique S (voir les figures 4.4 et 4.5), proposée par Zitzler [Zit 99], calcule l'hypervolume de la région multidimensionnelle formée par P_A et un point de référence ; c'est-à-dire la taille de l'espace des objectifs que P_A domine.

L'avantage de cette mesure, dont la signification est intuitive, est qu'elle permet d'ordonner entres elles différentes courbes de Pareto approchées. Cependant, elle nécessite de choisir un point de référence correspondant à une borne supérieure de la région dans laquelle se trouvent tous les points réalisables. Ce choix peut avoir un impact sur l'ordre entre les approximations (voir la figure 4.4). De plus, le temps de calcul est important. Il est de l'ordre de $O(|P_A|^{n+1})$. De ce fait, la mesure est impraticable dans le cas où de nombreux objectifs sont

considérés, ou si le cardinal des ensembles est trop important. Néanmoins, dans leur étude, Knowles et Corne [Kno 02] recommandent l'utilisation de cette mesure.

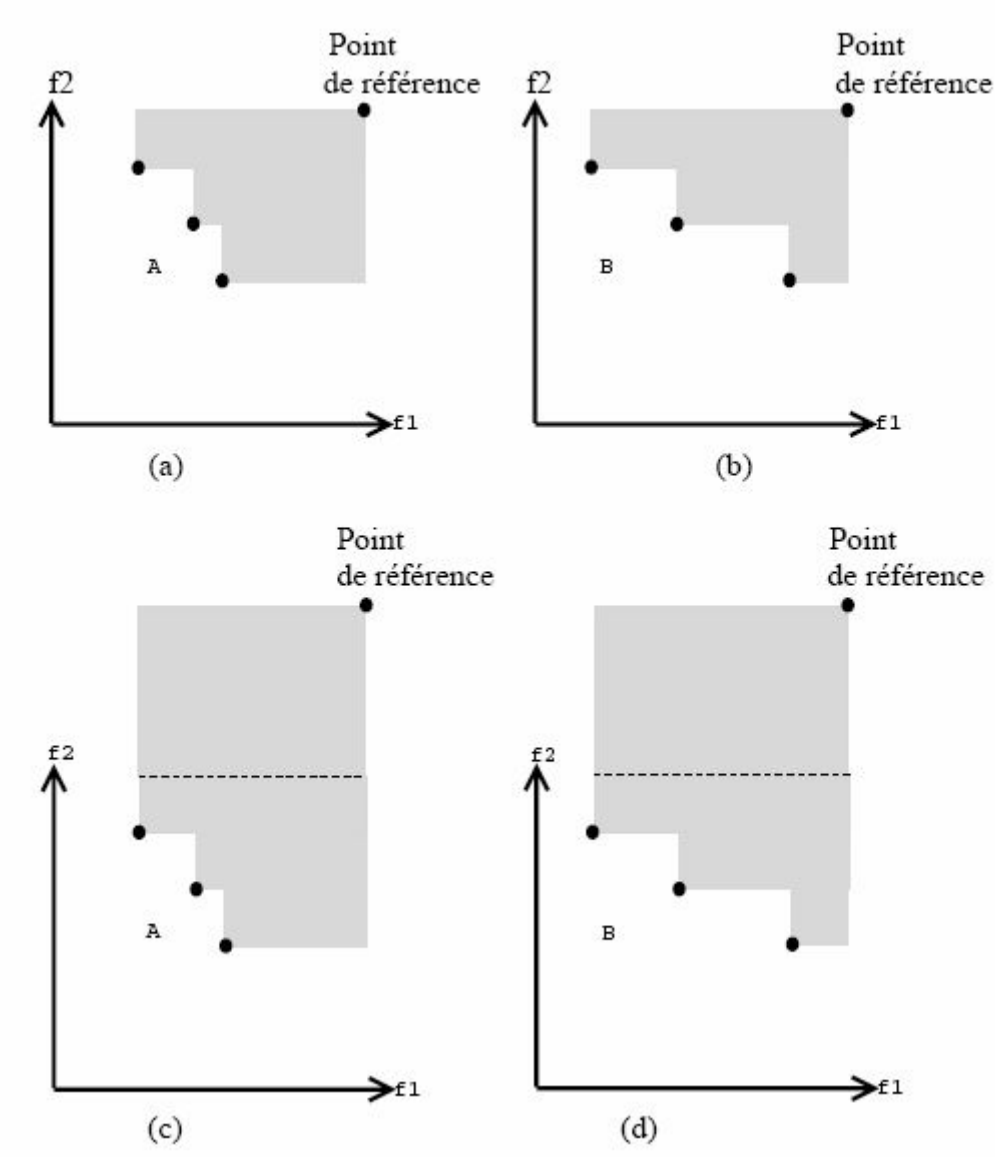


Figure 4.4. La valeur de *S* dépend du choix arbitraire d'un point de référence. Dans les cas (a) et (b), deux courbes de Pareto approchées A et B constituées de 3 solutions sont montrées, avec $S(A) > S(B)$. Dans les cas (c) et (d), pour les mêmes ensembles, on a cette fois-ci $S(A) < S(B)$.

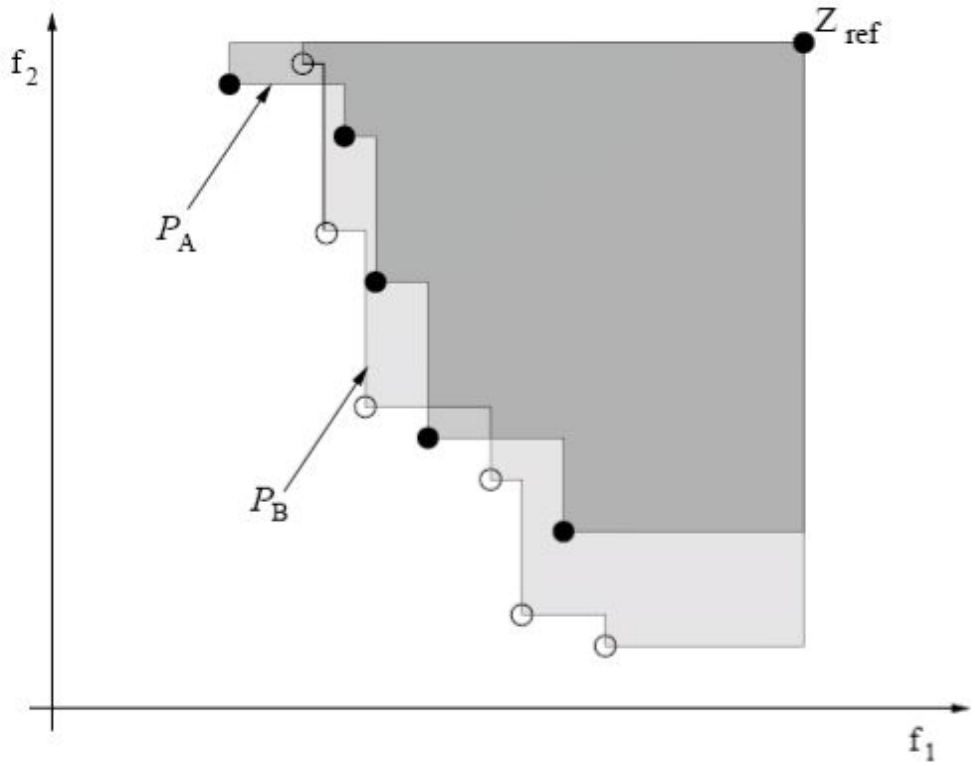


Figure 4.5. Exemple de la métrique *S*, correspondant aux aires de dominance d'ensembles de solutions Pareto par rapport à un point de référence Z_{ref}.

4.3. Formalisation et généralités sur le *flow-shop* multiobjectif

Le problème du *flow-shop*, que nous considérons par la suite fait partie de la classe des problèmes d'ordonnancement. Un problème d'ordonnancement se compose de tâches à exécuter et de machines disponibles pour l'exécution de ces tâches. D'une manière générale, la résolution d'un problème d'ordonnancement consiste à déterminer sur quelle machine doit s'exécuter chaque tâche et la date de début d'exécution de chaque tâche sur chaque machine, de manière à optimiser une certaine fonction objectif.

4.3.1. État de l'art des méthodes d'optimisation pour l'ordonnancement "*flow-shop*"

L'éventail des différents problèmes du *flow-shop* étudiés est très large, surtout si on se rapproche des applications réelles [Tbo 02]. Les problèmes du *flow-shop* sont divisés en deux ensembles : ceux se limitant à deux machines, et ceux avec un nombre de machines *M* variable selon l'instance du problème. La liste des études citées dans ce qui suit n'est pas exhaustive. Dans un premier temps, nous présentons les différentes approches exactes et métaheuristiques proposées dans la littérature portant sur les problèmes du *flow-shop*. Puis nous présentons différentes approches coopératives trouvées dans la littérature.

4.3.1.1. Approches exactes

De nombreuses approches exactes existent dans la littérature pour résoudre des problèmes du *flow-shop* à un seul objectif. Ainsi, le problème du *flow-shop* avec temps de transition dépendants de la séquence (*sequence dependent setup times*), est résolu par la

méthode de *Séparation & d'Evaluation* [Rio 99] et *Branch & Cut* par Rios-Mercado [Rio 98]. Bard, Kohler et Steiglitz [Koh 75] résolvent le problème du *flow-shop* à deux machines par différentes approches, exactes ou heuristiques. Carlier et al. [Car 00] proposent de résoudre de manière optimale un *flow-shop* hybride dans lequel des dates de disponibilité et des durées de latence sont associées aux opérations et aux machines.

Quelques approches exactes ont été proposées pour la résolution de problèmes du *flow-shop* dans un contexte multiobjectif. Par exemple, Sayin et Karabati [Say 99] ont proposé une approche de type *Séparation & Evaluation* pour résoudre un problème bi-objectif du *flow-shop* à deux machines, les critères étant le *makespan* C_{max} et la somme des dates de complétude. Ils ont développé pour cela une procédure de *Séparation & d'Evaluation* résolvant des problèmes avec un seul objectif, le second objectif étant remplacé par une contrainte. Le procédé est itéré avec différentes valeurs pour la contrainte ajoutée, jusqu'à ce que toutes les solutions Pareto soient énumérées.
Sivrikaya-Serifoglu et Ulusoy [Siv 98] proposent une approche par *Séparation & Evaluation*, avec agrégation des mêmes objectifs. Le même problème est résolu par *Séparation & Evaluation* par Yeh [Yeh 99]. Yeh et Allahverdi [Yeh 04] proposent le même type d'approche pour résoudre le même problème, mais sur trois machines.
Bien entendu, les méthodes exactes appliquées aux problèmes du *flow-shop* restent limitées, et, comme pour beaucoup de problèmes d'optimisation, les approches heuristiques sont proposées, afin de résoudre de manière approchée les problèmes de grande taille.

4.3.1.2. Approches par métaheuristiques

Pour la résolution des problèmes du *flow-shop* de grande taille, beaucoup d'heuristiques et de métaheuristiques ont été proposées, aussi bien pour les cas monoobjectif que multiobjectif.
On trouve de nombreux algorithmes basés sur la recherche tabou, comme par exemple les travaux de Pempera [Gra 02], Nowicki et Smutnicki [Now 96], ou Yamada [Yam 02]. Mais la plupart des métaheuristiques classiques ont été utilisées pour ce type de problème, comme les colonies de fourmis [Stü 98a][Yin 04], la recherche par dispersion [Now 03], la recherche locale itérée [Stü 98b], le recuit simulé [Par 98], ou les algorithmes génétiques [Yon 01]. Ces derniers sont d'ailleurs souvent utilisés de manière coopérative avec une autre méthode d'optimisation.

D'une manière générale, les métaheuristiques proposées ne sont pas réellement adaptées à tous les aspects de l'optimisation : robustesse, exploration, intensification, etc. On trouve donc également de plus en plus d'approches hybrides réalisées dans le but de pallier aux manques de chaque méthode.
Ce type d'approches est très largement répandu, essentiellement pour l'optimisation monoobjectif. Ce sont principalement les algorithmes évolutionnaires qui sont hybridés, afin d'être améliorés par des mécanismes de recherche locale qui permettent d'intensifier la recherche sur les bonnes solutions obtenues par l'algorithme à base de population d'individus. Les algorithmes mimétiques (hybridation où l'algorithme génétique utilise la recherche locale comme opérateur) représentent une possibilité pour ce type d'approche, et sont maintenant considérés comme une méthode d'optimisation classique. Un algorithme mimétique a été proposé par Ishibuchi et al. pour résoudre un problème du *flow-shop* bi-objectif [Ish 03].
Yamada et Reeves optimisent la somme des temps de complétude d'un problème du *flow-shop* [Yam R95] en utilisant une recherche Tabou comme mécanisme de recherche locale, de

même que pour l'approche proposée par Zdansky et Pozivil, pour un problème du *flow-shop* hybride [Zda 02].

Feo et al. [Feo 91] ont mis au point la méthode d'optimisation GRASP (*Greedy Randomized Adaptive Search Procedure*). L'algorithme GRASP consiste à réaliser des recherches locales sur des solutions générées de manière gloutonne. Ils l'ont appliqué à un problème du *flow-shop* à une machine avec pénalités sur les avances des tâches (coopération de type (recherche locale+heuristique gloutonne)) [Feo 91]. Dans [Aie 03], Aiex et al. proposent un algorithme GRASP parallèle avec la méthode *path-relinking* comme mécanisme d'intensification.

Peu d'études proposent des coopérations entre une métaheuristique et une méthode exacte pour résoudre des problèmes d'ordonnancement. Nous pouvons citer l'approche hybride entre un algorithme génétique et celle par *Séparation & Evaluation* de Portmann et al. pour résoudre un problème du *flow-shop* hybride [Por 98].

Une deuxième approche coopérative entre une métaheuristique et une méthode exacte a été proposée par Haouari et Ladhari [Hao 03]. Le problème résolu est de type *flow-shop* de permutation, où l'objectif optimisé est la date de terminaison. Dans cette étude, une heuristique permet d'obtenir des ordonnancements initiaux, qui sont ensuite améliorés à la manière d'une recherche locale, mais en réalisant des arbres d'énumération partielle.

En ce qui concerne les approches coopératives multiobjectifs pour l'ordonnancement, les approches proposées sont encore plus rares. Nous n'avons trouvé que deux approches. Une méthode spécifique au problème, qui coopère avec un algorithme de colonies de fourmis de manière séquentielle. Cette coopération est proposée par T'kindt et al. [Tki 02b], qui optimise un problème du *flow-shop* bi-objectif (noté $F2//Lex(C_{max}, \sum C_i)$). Mais, dans cette étude, les objectifs sont traités de manière lexicographique. Le coté multiobjectif du problème est donc transformé en deux problèmes monoobjectifs traités de manières quasi indépendantes. La deuxième est celle de Basseur [Bas 05], qui fait coopérer un algorithme génétique adaptatif (plusieurs opérateurs de mutation évoluent simultanément et il y a une diversification adaptative) et un algorithme exact (*Branch & Cut*) au travers d'une méthode à deux phases appliquée au problème du *flow-shop* de permutation bi-objectif.

4.3.2. Problème du *flow-shop* de permutation multiobjectif

Le problème du *flow-shop* se présente comme un ensemble de N tâches $\{J_1, J_2, ..., J_N\}$ à ordonnancer sur M machines $\{m_1, m_2, ..., m_M\}$. Les machines sont des ressources critiques, dans le sens où une machine ne peut pas être affectée à deux tâches simultanément. Chaque tâche J_i est composée de M opérations consécutives, $J_i=\{t_{i1}, t_{i2}, \ldots, t_{iM}\}$. L'opération t_{ij} représente la $j^{ème}$ opération de la tâche J_i, et elle doit s'exécuter sur la machine m_j. Par conséquent, toutes les tâches ont la même séquence d'usinage sur les machines : d'abord la machine m_1, puis la machine m_2, etc. jusqu'à la machine m_M. La date d'achèvement d'une tâche J_i est égale à la date d'achèvement de la dernière opération dont elle est constituée, c'est-à-dire t_{iM}. Chaque opération t_{ij} possède une durée d'exécution p_{ij}, et chaque tâche J_i est limitée par une date d_i avant laquelle elle doit obligatoirement s'achever.

Le problème bi-objectif que nous considérons par la suite est un problème du *flow-shop à permutation*, noté $F/perm,d_i/(C_{max},T)$ en utilisant la notation de Graham et al. [Gra 79]). Rappelons que le premier champ de la notation désigne le type de problème d'ordonnancement, dans notre cas il s'agit d'un problème du *flow-shop* (*F*). Le deuxième champ désigne les éléments spécifiques à prendre en compte pour le problème étudié. Ici, *perm* indique qu'on étudie un *flow-shop* de permutation, c'est-à-dire que les tâches doivent être ordonnées dans le même ordre sur toutes les machines (voir la figure 4.6, où chaque machine exécute d'abord une opération de la tâche 1, puis une opération de la tâche 2 et enfin une opération de la tâche 3). De plus d_i indique que chaque tâche a une date d_i avant laquelle elle doit s'achever. Enfin, le troisième champ désigne les critères à optimiser. Ici, deux objectifs seront optimisés. Le premier objectif est la date de fin d'ordonnancement notée C_{max}, c'est-à-dire la date de fin de traitement de la dernière tâche sur la dernière machine m_M. Le deuxième objectif est la somme des retards de chaque tâche J_i (différence entre la date de complétude de la tâche J_i et sa date de terminaison prévue d_i, si elle est positive, et 0 sinon), notée *T*.

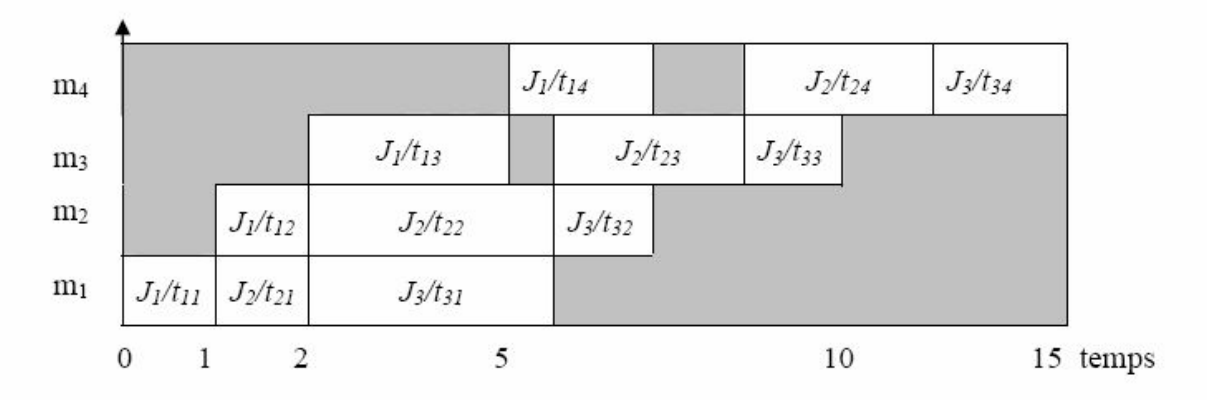


Figure 4.6. Exemple du *flow-shop* de permutation.

Dans la figure 4.6, on voit un exemple d'ordonnancement dans lequel la tâche 2, J_2, constituée des 4 opérations $\{t_{21}, t_{22}, t_{23}, t_{24}\}$ est d'abord exécutée sur la machine 1, puis sur la machine 2 puis sur la machine 3 et enfin sur la machine 4. Avec les valeurs des dates de fin souhaitées d_{J1}=9; d_{J2}= 15et d_{J3}=14, le *makespan* C_{max} est égal à 15 et la somme des retard *T* à 1.

Le premier objectif est très largement étudié dans la littérature. Le deuxième l'est aussi, mais pour les problèmes à une seule machine. Très peu de travaux étudiant cet objectif pour *M* machines existent [Kim 95].

4.4. Proposition d'un algorithme génétique pour le problème du *flow-shop* multiobjectif

Les algorithmes évolutionnaires, et particulièrement les algorithmes génétiques, peuvent être bien adaptés à l'optimisation multiobjectif. En effet, l'utilisation d'une population de solutions permet de pouvoir approcher la courbe de Pareto. Nous proposons, ci-dessous (figure 4.7), le schéma général de l'algorithme génétique implémenté. Dans cet algorithme et dans la suite de ce chapitre, on désignera par PO^+ une population archive où se trouvent l'ensemble des solutions Pareto optimales trouvées et jamais dominées durant toute la recherche. Rappelons que le principe de l'élitisme dans les algorithmes génétiques consiste à

utiliser cette population archive PO^+ en plus de la population courante lors de la phase de sélection des solutions pour la reproduction.

```
Algorithme : AG_Pareto

Début
1.  Générer les individus de la population initiale P de façon aléatoire.
2.  Evaluer les individus de la population P {Calcul des vecteurs coût de
     chaque individu}.
3.  Calculer l'ensemble Pareto Optimal PO+ de la population courante P
    (élitisme).
4.  Tant que le nombre de générations limite n'est pas atteint (critère
              d'arrêt)
    faire
    a. Calculer les probabilités de sélection de chaque individu selon le
       type de sélection adoptée.
    b. Sélectionner tp individus de la population  courante et
       éventuellement de la population PO+(sélection élitiste) suivants
       leurs probabilités, et constituer la population intermédiaire.
    c. Pour i := 1 à tp      /* Phase de croisement */
        faire
    d. Sélectionner une paire d'individus de la population intermédiaire.
       Suivant une probabilité de croisement Pc
       choisir soit
       (d.a) réaliser le croisement et insérer l'individu fils dans la
             nouvelle population ou
       (d.b) insérer l'un des deux individus pères dans la nouvelle
             population.
    e.  Pour i := 1 à tp faire muter l'individu Si de la population avec un
                              probabilité de mutation Pm. /*Phase de
                              mutation */
    f. Evaluer les individus de la nouvelle population.
    g. Mettre à jour PO+ : PO+= Solutions Non Dominé (PO++ Population)
    fait
Retourner PO+
Fin.
```

Figure 4.7. Algorithme génétique Pareto implémenté *AG_Pareto.*

4.4.1. Représentation des solutions

L'application d'un algorithme génétique à un problème donné nécessite, dans un premier temps, une représentation chromosomique des solutions, dans notre cas un ordonnancement des tâches. La séquence de passage des tâches dans les machines étant identique, puisque nous considérons un *flow-shop* de permutation, il suffit donc de représenter la séquence de traitement des tâches sur une seule machine. Par conséquent, un ordonnancement est vu comme une permutation définissant l'ordre de passage des tâches sur chacune des machines. L'exemple de la figure 4.6 est représenté suivant ce codage par un vecteur associant une entrée à chaque tâche. La position d'une tâche dans le chromosome définit son numéro d'ordre dans la séquence (figure 4.8).

1	2	3

Figure 4.8. Représentation chromosomique de la solution de la figure 4.6.

4.4.2. Génération de la population initiale

La procédure de génération de la population initiale consiste en une production aléatoire de plusieurs permutations. L'aspect aléatoire nous garantit une bonne répartition des individus dans l'espace de recherche.

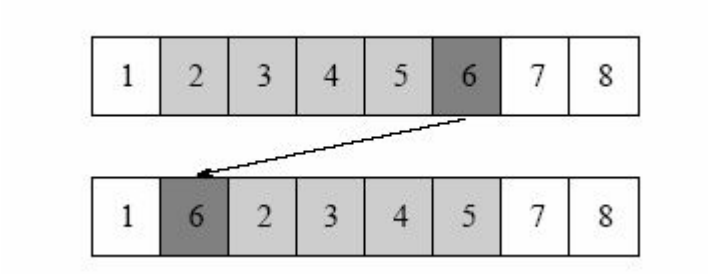


Figure 4.9. Opérateur de mutation.

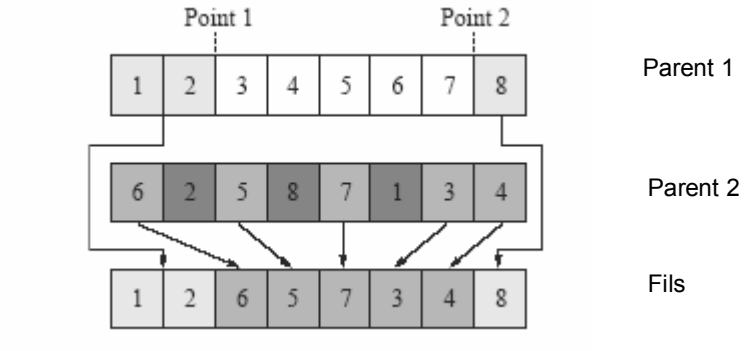


Figure 4.10. Opérateur de croisement.

4.4.3. Opérateurs génétiques

Appliquer un algorithme génétique à un problème donné nécessite aussi le choix des opérateurs génétiques qui serviront à faire évoluer les solutions de la population. Nous nous sommes inspirés des opérateurs définis par *Murata* et *Ishibuchi* [Mur 98]. Les opérateurs de mutation et de croisement sont décrits dans les figures 4.9 et 4.10.
L'opérateur de mutation consiste à choisir de manière aléatoire deux points dans le chromosome. Une rotation est alors effectuée, comme indiquée dans la figure 4.9. L'opérateur de croisement, aussi nommé "*croisement à deux points*", consiste à choisir de manière aléatoire deux points de croisement. Un individu fils est alors obtenu. Il est similaire au niveau de ses extrémités, au *parent1*, le reste des tâches étant prises dans l'ordre rencontré dans le *parent2* (figure 4.10).

4.5. Opérateur de sélection

Le mécanisme de sélection dans les algorithmes génétiques consiste essentiellement à favoriser les solutions les plus intéressantes en terme de qualité. Dans le cadre d'une résolution multiobjectif Pareto, l'opérateur de sélection n'a *a priori* pas de raison d'être modifié.

4.5.1. Stratégies de sélection implémentées

Pour notre étude, nous avons implémenté et testé six stratégies de sélection. Les différences principales entre ces méthodes résident dans la façon dont les individus de la population sont ordonnés, ainsi que l'expression permettant le calcul des probabilités de sélection. Dans la suite, nous nous plaçons dans le cas bi-objectif.

4.5.1.1. Sélection par somme pondérée des objectifs

C'est l'une des premières méthodes utilisées dans le cadre de l'optimisation multiobjectif. Basée sur la transformation du problème bi-objectif en un problème monoobjectif, cette méthode consiste à combiner les différentes fonctions objectifs en une seule fonction, généralement de façon linéaire. Ainsi, soit x_i un individu faisant partie de la population de solutions, et soit $f_1(x_i)$ (respectivement $f_2(x_i)$) son coût vis-à-vis de la première (respectivement seconde) fonction objectif. On crée une nouvelle fonction objectif f de la manière suivante :

$$f(x_i) = \sum_{k\in[1..2]} \lambda_k f_k(x_i)$$

avec $\lambda_k \in [0..1]$ et $\sum_{k\in[1..2]} \lambda_k = 1$.

Le mérite d'un individu est dans ce cas égal à $f(x_i)$:

$$mérite(x_i) = f(x_i)$$

Un individu x_i a alors une probabilité d'être sélectionné égale à :

$$\pi(x_i) = \frac{mérite(x_i)}{\sum_{j\in[1..tp]} mérite(x_j)} = \frac{f(x_i)}{\sum_{j\in[1..tp]} f(x_i)}$$

où tp désigne la taille de la population courante.

4.5.1.2. Sélection parallèle

Dans la sélection parallèle, la moitié des solutions sélectionnées le sont en tenant uniquement compte du premier critère qui est leur temps de terminaison. Les $tp/2$ autres individus sont sélectionnés en tenant uniquement compte du deuxième critère, qui est la somme des retards de toutes les opérations. La procédure ci-dessous (figure 4.11) décrit cette méthode.

```
Algorithme : Sélection_parallèle
Début
- Trier les individus de la population courante selon l'ordre
  croissant de f1 et soit rang1(i) le rang de l'individus i.
- Trier les individus de la population courante selon l'ordre
  croissant de f2 et soit rang2(i) le rang de l'individus i.
- Pour i : = 1 à tp/2
  faire
      Sélectionner un individu de la population courante avec la
      probabilité rang1(i)/∑j∈[1..tp/2]rang1(j);
      Insérer cet individu dans la population intermédiaire
  fait
- Pour i : = 1 à tp/2
  faire
      Sélectionner un individu de la population courante avec la
      probabilité rang2 (i)/∑j∈[1..tp/2]rang2(j) ;
      Insérer cet individu dans la population intermédiaire.
  fait
- Effectuer la reproduction sur la population intermédiaire.
Fin.
```

Figure 4.11. Algorithme réalisant la sélection parallèle : *Sélection_Parallèle.*

4.5.1.3. Sélection NSGA

Dans la sélection *NSGA* (*Non Dominated Sorting Genetic Algorithm*) [Sri 95] les rangs des individus sont calculés d'une manière récursive en commençant par les individus dominants de la population (voir la figure 4.12).
On associe le rang 1 à l'ensemble des individus non dominés E_1 de la population courante.
On associe le rang 2 à l'ensemble des individus E_2 qui ne sont dominés que par des individus appartenant à E_1.
On associe le rang 3 à l'ensemble des individus E_3 qui ne sont dominés que par des individus appartenant à $E_1 \cup E_2$ et ainsi de suite.

Finalement, on associe le rang k à l'ensemble des individus E_k qui ne sont dominés que par des individus appartenant à $E_1 \cup E_2 \cup \ldots E_{k-1}$.

La probabilité de sélection est ensuite affectée à chaque individu en se basant sur le rang, en appliquant par exemple une méthode similaire à celle de Baker [Bak 85] proposée pour les problèmes monoobjectifs. La probabilité qu'un individu de rang k de la population courante soit sélectionné est donnée par l'équation suivante (avec tp la taille de la population, S la pression de sélection, r_i le nombre d'individus de rang i):

$$\pi_k = \frac{S(tp + 1 - R_k) + R_k - 2}{tp(tp - 1)} \qquad (1)$$

$$\text{avec} \quad R_k = 1 + r_k + 2 * \sum_{j \in [1..k-1]} r_j.$$

Une pression de sélection S égale à 1 donne la même probabilité de sélection pour tous les individus ($\pi_k = 1/tp$). Pour des valeurs de S supérieures à 2, les individus de dernier rang

peuvent avoir des probabilités de sélection négatives. Dans ce cas, ils ne sont jamais sélectionnés.

4.5.1.4. Sélection NDS

Dans la sélection *NDS* (*Non Dominated Sorting*) le rang d'un individu est égal au nombre de solutions dominant l'individu plus un.

$$Rang\ (x_i) = |\ x_j \in Population\ /\ x_j \text{ domine } x_i|\ +\ 1$$

La probabilité de sélection est calculée suivant le même procédé que celui de la formule (1).

4.5.1.5. Sélection WAR

Dans la sélection WAR (*Weighted Average Ranking)* on calcule le rang de chaque individu de la population par rapport aux différents objectifs séparément. Le rang d'un individu est alors égal à la somme des rangs calculés :

$$Rang(x_i) = Rang\_makespan(x_i) + Rang\_Retard(x_i)$$

Pour calculer le rang de chaque individu, on commence par ordonner les individus par ordre croissant de la fonction f_1 et par ordre croissant de la fonction f_2. Les probabilités de sélection sont alors calculées comme pour la sélection *NSGA*.

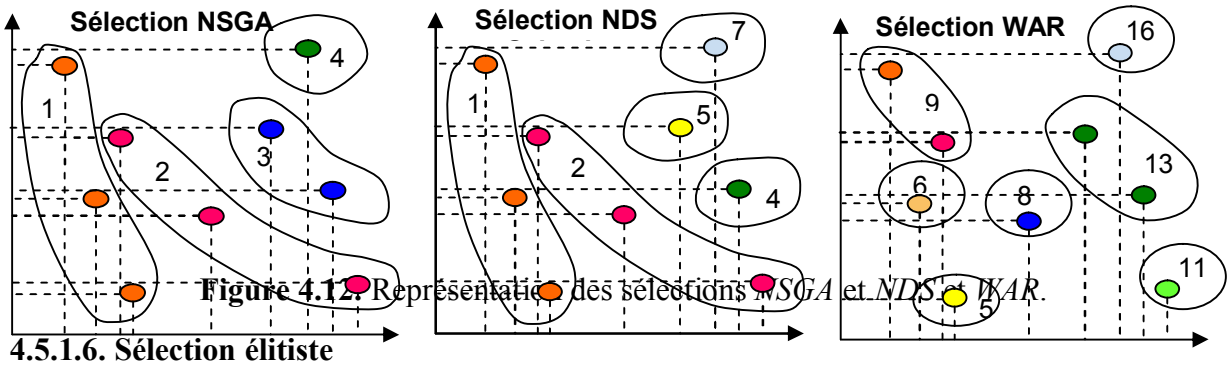


Figure 4.12. Représentation des sélections *NSGA* et *NDS* et *WAR*.

4.5.1.6. Sélection élitiste

La sélection élitiste consiste à maintenir une population archive PO^+ qui contient les meilleures solutions non dominées rencontrées tout au long de la recherche. Cette population participe aux étapes de sélection et de reproduction.
On donne une probabilité *A/tp* de choisir un élément de PO^+, *A/tp* correspondant à la proportion d'individus de PO^+ voulus dans la population totale, et la probabilité qu'un individu x_i de rang k soit sélectionné devient :

$$\pi_k(x_i) = \frac{(tp - A)}{tp} \times \frac{S(tp + 1 - R_k) + R_k - 2}{tp(tp - 1)}.$$

Le paramètre *"A"* détermine de ce fait l'espérance du nombre d'individus sélectionnés à partir de l'ensemble PO^+. L'intérêt de l'élitisme est certain pour améliorer la convergence des solutions. Cependant, une valeur trop grande pour le paramètre *A* implique une convergence rapide au détriment d'une exploration de l'espace des solutions moins performante.

En réalité, toutes les méthodes de sélection gardent une archive contenant les meilleures solutions rencontrées pendant la recherche (ensemble Pareto). La particularité de l'élitisme est de faire participer cette population lors de la phase de sélection (figure 4.14). Le rang des individus est calculé suivant la technique *NSGA*. La procédure suivante (figure 4.13) décrit le processus de sélection d'un individu de la population courante dans le cas de la sélection élitiste.

```
Algorithme : Sélection_élististe
Début
Choisir un nombre aléatoire p compris dans l'intervalle [0, 1]
Si p >(tp - A) / A
alors
      I:=1 ; T := π1
      Tant que  p ≥ T
      faire
            i := i+1; T := T + πi;
            Insérer l'individu sélectionné Si dans la population
            intermédiaire
      fait
Sinon sélectionner de façon aléatoire un individu quelconque de la
      population Pareto (PO+)
fin si
FIN.
```

Figure 4.13. Processus de sélection d'un individu dans le cas de la sélection élitiste.

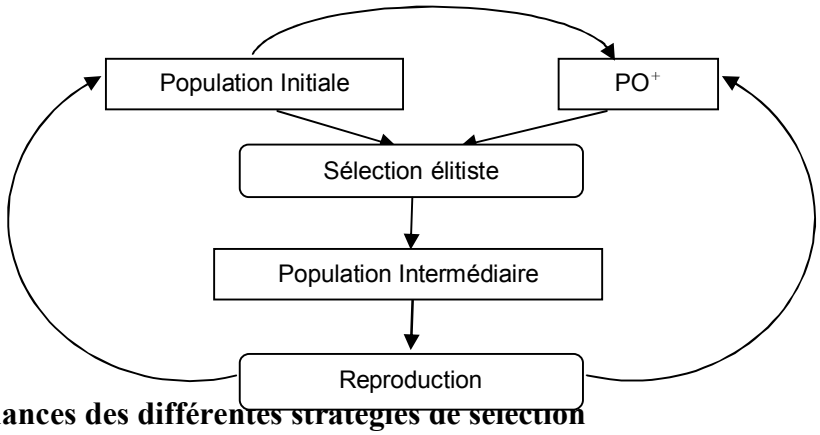


Figure 4.14. Stratégie de sélection élitiste.

4.5.2. Performances des différentes stratégies de sélection

Pour comparer les performances des 6 stratégies de sélection considérées, des tests ont été effectués sur l'instance de *Heller* comportant 20 tâches et 10 machines [Hel 60]. Les paramètres des différentes méthodes sont décrits dans le tableau 4.1. La taille de la population est de 200 individus et le nombre de générations limite est de 15000. Les probabilités de croisement P_c et de mutation P_m sont respectivement égales à 0,8 et 0,7. Ces valeurs ont été fixées suite à plusieurs exécutions et elles correspondent à celles qui donnent les meilleurs résultats. Les tests (figure 4.15) montrent une amélioration de la recherche avec l'introduction de l'élitisme dans la phase de sélection. Les stratégies qui ne sont pas "Pareto spécifiques", représentées par la sélection en utilisant une somme pondérée et par la sélection parallèle, semblent être moins bien adaptées au cas multiobjectif que les autres stratégies de sélection. Les trois stratégies de sélection *NSGA*, *NDS* et *WAR* ont des performances presque identiques, avec une légère suprématie pour les méthodes *NSGA* et *NDS*.

Somme pondérée	$\lambda 1 = 0.5$	$\lambda 2 = 0.5$
NSGA	S = 1.7	
NDS	S = 1.7	
WAR	S = 1.7	
Elitiste	S = 1.7	A = 5

Tableau 4.1. Paramètres des différentes stratégies de sélection.

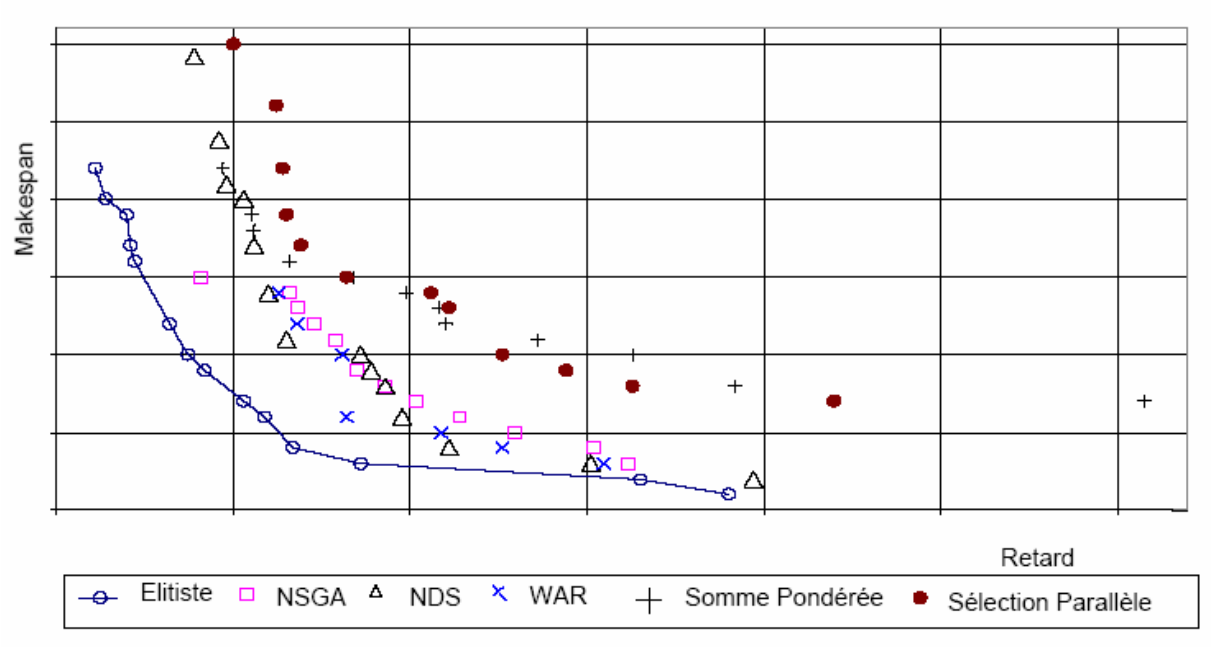
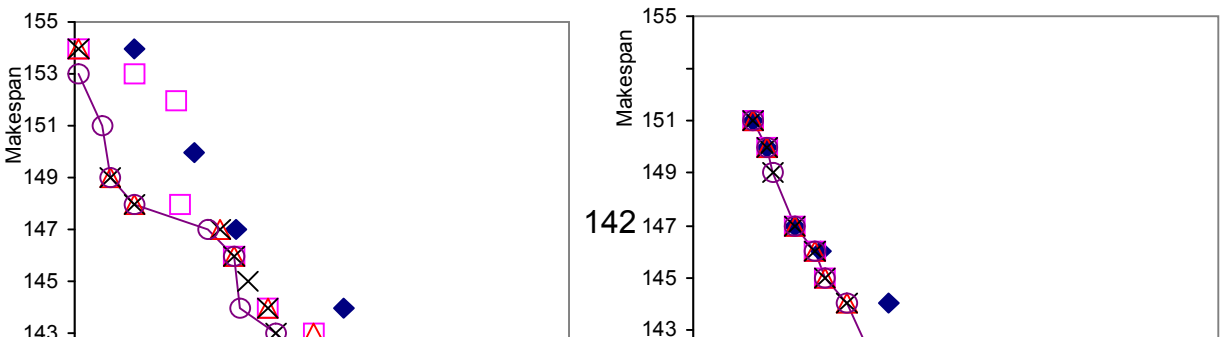


Figure 4.15. Courbes de Pareto obtenues avec différentes stratégies de séléction

4.5.3. Effet du paramètre "*A*" dans la stratégie élitiste

L'apport de l'élitisme étant prouvé expérimentalement, il est donc intéressant de connaître l'impact qu'a le choix du paramètre "*A*" sur l'évolution de la recherche dans l'algorithme génétique. Le choix de la valeur de ce paramètre influe considérablement sur l'équilibre entre l'exploitation et l'exploration. En effet une valeur élevée de "*A*" a pour effet d'intensifier l'exploitation des bonnes solutions trouvées par la recherche. Une valeur "petite" de "*A*" a pour effet de favoriser l'exploration de nouveaux horizons de l'espace de recherche. Le choix d'une valeur adaptée de "*A*" est assez déterminant pour l'efficacité de la recherche. Les figures 4.16 et 4.17 montrent l'évolution de la recherche le long des générations 1000, 3000,… avec des valeurs du paramètre "*A*" = 1 et 4. Pour une valeur de "*A*" petite ("*A*"=1) la convergence vers la frontière Pareto se fait lentement, contrairement aux résultats obtenus avec "*A*" = 4.



4.6. Le maintien de la diversité

Les algorithmes génétiques classiques sont réputés pour être très sensibles quant au choix de la population initiale, ainsi qu'aux mauvais échantillonnages lors de la sélection. Cette fragilité est observable sur le plan de la perte de diversité ou ce que l'on appelle aussi la dérive génétique. Pour pallier à cet inconvénient, nous avons implémenté la méthode de partage (*sharing*).

Le principe du partage [Gol 87] consiste à dégrader le mérite d'un individu par rapport à son mérite réel, pour éviter de trop fortes concentrations d'individus semblables dans une portion réduite de l'espace de recherche. Pour juger de la proximité de deux solutions dans l'espace de recherche, il faut définir une distance entre les individus.

Selon que la distance entre les individus est calculée dans l'espace de décision (la représentation chromosomique de l'individu) ou dans l'espace objectif (adéquation de l'individu), trois possibilités peuvent se présenter.

4.6.1. Partage génotypique

La distance entre individus calcule la différence entre chromosomes représentant les permutations. Ici, la distance entre deux individus x et y est égale au nombre de ruptures d'ordre entre x et y.

$d1(x, y) = |\ \{(i, j)$ / la tâche i précède la tâche j dans la solution x, et la tâche j précède la tâche i dans $y\}|$

4.6.2. Partage phénotypique

La distance entre deux individus est la différence de leur coût. Dans le cas d'un problème bi-objectif on tient compte des deux fonctions objectifs :

$$d2(x,y) = |f_1(x) - f_1(y)| + |f_2(x) - f_2(y)|$$

4.6.3. Partage combiné

Ce dernier cas représente la combinaison des deux premières approches, dans le sens où le calcul de la distance fait intervenir les deux sortes de distances, génotypique et phénotypique. La fonction *sh* prend dans ce cas la forme suivante :

$$sh(x,y)=\begin{cases} 1-\dfrac{d1(x,y)}{\gamma 1} & \text{si d1(x, y)} < \gamma 1,\ \text{d2(x, y)} \geq \gamma 2 \\ 1-\dfrac{d2(x,y)}{\gamma 2} & \text{si d1(x, y)} \geq \gamma 1,\ \text{d2(x, y)} < \gamma 2 \\ 1-\dfrac{d1(x,y)d2(x,y)}{\gamma 1\gamma 2} & \text{si d1(x, y)} < \gamma 1,\ \text{d2(x, y)} < \gamma 2 \\ 0 & \text{sinon} \end{cases}$$

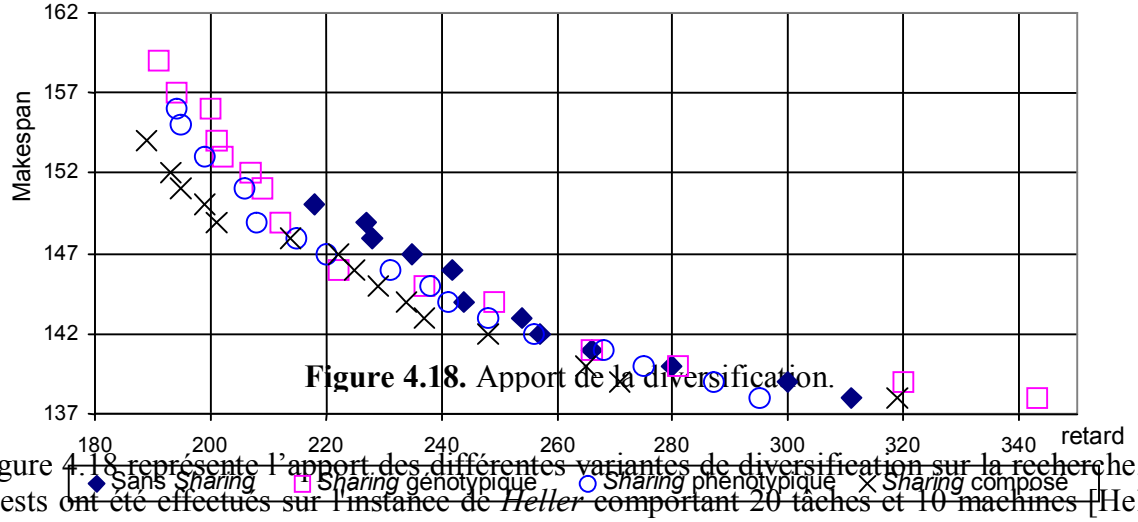


Figure 4.18. Apport de la diversification.

La figure 4.18 représente l'apport des différentes variantes de diversification sur la recherche. Les tests ont été effectués sur l'instance de *Heller* comportant 20 tâches et 10 machines [Hel 60]. Avec un nombre de générations limite de 50000 et en utilisant les mêmes paramètres que ceux décrits précédemment pour la stratégie de sélection élitiste dans la 4.5.1.6. Les paramètres concernant la stratégie de diversification sont : α=0,9, $\gamma 1$=4 et $\gamma 2$=1. L'apport de la diversification génotypique est peu appréciable en comparaison des résultats obtenus par la diversification phénotypique. Cependant, la diversification dans l'espace de décision se distingue par un ensemble de solutions efficaces non trouvées par la diversification phénotypique. La diversification composée présente une meilleure qualité de solutions que les deux précédentes méthodes. A noter que l'apport de la diversification n'apparaît qu'après un nombre de générations considérable, d'où la nécessité de prendre un nombre de générations limite très élevé (> 40000).

4.7. Hybridation avec la recherche locale

Nous avons utilisé la recherche locale, afin d'améliorer les performances de l'algorithme génétique proposé. Nous avons retenu l'algorithme génétique le plus performant quant aux différentes stratégies de sélection, de classement des individus et de maintien de la diversité testées. Cet algorithme utilise l'élitisme comme mode de sélection et le partage *combiné* comme moyen de maintien de la diversité.

Le principe de l'hybridation proposée est de lancer l'algorithme génétique en premier lieu, afin d'approcher la frontière Pareto, après quoi la recherche locale s'occupe d'améliorer les solutions trouvées par l'algorithme génétique, afin de mieux approcher l'ensemble de la courbe de Pareto. Le principe de l'hybridation est simple, une fois l'algorithme génétique

terminé (le nombre de générations limite a été atteint), la recherche locale est alors lancée en ayant pour entrée chacune des solutions de l'ensemble de Pareto obtenu.

L'utilisation de la recherche locale nécessite, premièrement, de définir le voisinage d'une solution. Nous nous sommes inspirés pour cela de l'opérateur de mutation. Le voisinage d'une solution correspond à l'ensemble des permutations obtenues suite au déplacement d'une tâche (rotation dans les deux sens d'une partie de la solution) (figure 4.19). Pour construire l'ensemble des voisins d'une permutation O, on choisit chaque fois une paire de tâches en positions i et j respectivement. L'insertion de la tâche présente à la position i à la position j donne un premier voisin O_1. De même, l'insertion de la tâche présente à la position j à la position i donne un deuxième voisin O_2. De ce fait, le nombre de voisins d'un individu O est égal à $2 \times C_n^2 = N(N-1)$, avec N le nombre de tâches.

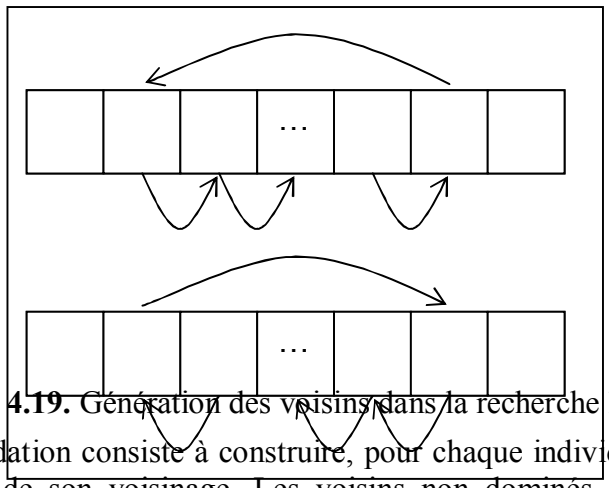
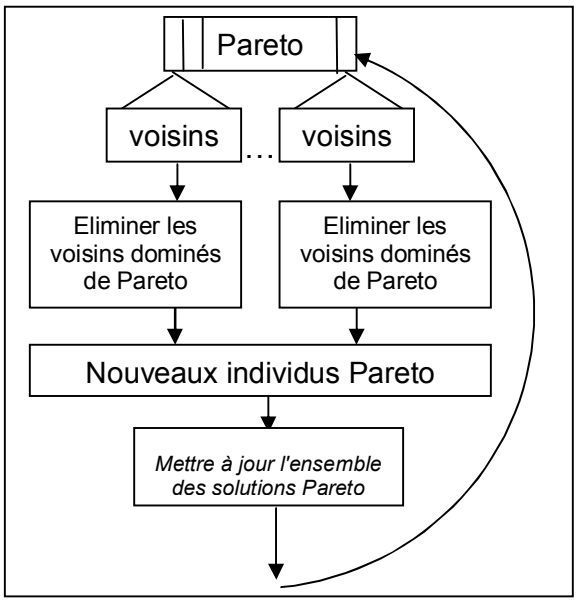


Figure 4.19. Génération des voisins dans la recherche locale.

Le procédé d'hybridation consiste à construire, pour chaque individu de la population Pareto trouvé, l'ensemble de son voisinage. Les voisins non dominés dans la population Pareto sont insérés dans celle-ci et les solutions nouvellement dominées sont supprimées. Ce procédé est itéré, jusqu'à ce qu'aucun voisin d'aucune solution Pareto ne soit inséré dans la population Pareto. La figure 4.20, reprend, sous forme schématique, le procédé de la recherche locale.

Les mesures de performance de l'algorithme génétique hybride obtenu montrent que la recherche locale ne présente aucun intérêt pour les problèmes de petite taille, notamment le problème *Heller* comportant 20 tâches et 10 machines. Cependant, l'apport de l'hybridation (figure 4.21) se fait sentir dès que la taille du problème augmente. Les tests représentés dans la figure 4.21 (somme des retards en abscisse et *makespan* en ordonnée) sont réalisés sur le problème de *Heller* avec 100 tâches et 10 machines.



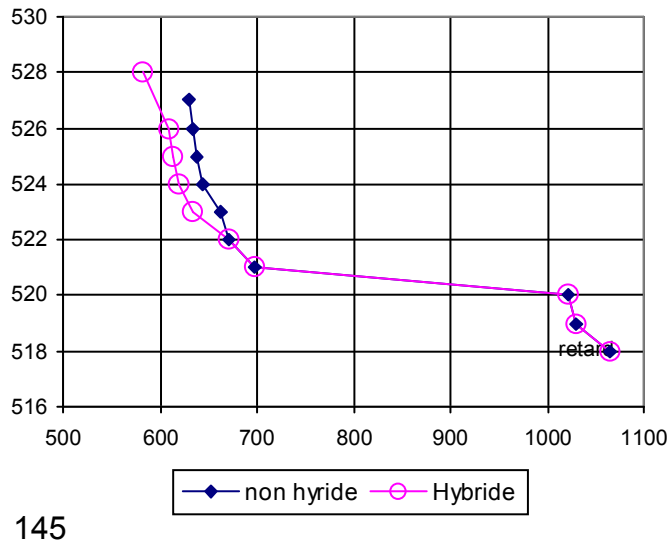


Figure 4.21. Apport de la recherche locale.

Figure 4.20. Recherche locale en action.

D'autres chercheurs ont également utilisé une hybridation de recherche locale avec un algorithme génétique. Dans [Mur 98], la procédure de la recherche locale est lancée suite à chaque étape de reproduction. Les individus de la nouvelle population sont considérés comme solutions de départ pour la recherche locale. Chaque individu est amélioré suivant le procédé de transition d'une solution à une autre solution voisine meilleure. Le processus est arrêté quand aucune amélioration n'est possible. Afin de réduire le temps de calcul induit par l'utilisation de la recherche locale suite à chaque génération, *Murata* et *Ishibuchi* fixent le nombre de voisins d'une solution examinés lors de chaque itération de la recherche locale.

Outre le fait que cette approche est très sensible quant au choix du paramètre *k* du nombre de voisins à explorer lors de chaque itération, l'algorithme est considérablement ralenti. Les tests comparatifs réalisés entre les deux approches d'hybridation montrent une égalité dans la qualité des solutions trouvées ce qui ne justifie en rien le temps d'exécution plus long de la deuxième approche.

4.8. Partie expérimentale

L'étude expérimentale de travaux d'optimisation multiobjectif se heurte généralement au problème d'absence de benchmarks. Dans la section 4.8.1, nous expliquons comment nous avons modifié la procédure de génération d'instances monoobjectif due à Taillard, afin d'introduire le deuxième critère, qui représente la somme des retards des tâches. Une deuxième difficulté consiste à déterminer la qualité des solutions obtenues. Pour cela, nous avons procédé en deux étapes. Dans la section 4.8.2, nous comparons les solutions obtenues par rapport à la borne inférieure du *makespan* et par rapport à une valeur *UB* correspondant à la meilleure solution obtenue par *Taillard* dans le cas monoobjectif. Dans la section 4.8.3, nous procédons à une évaluation de la qualité des fronts Pareto obtenus. Pour cela nous avons utilisé la métrique *contribution* et la métrique *S*. Ces deux métriques comparent le front obtenu par l'algorithme génétique proposé dans sa version de base (avec élitisme-NSGA mais sans la recherche locale, ni le maintien de la diversité) avec celui fourni par l'algorithme génétique adaptatif de Basseur [Bas 05], mentionné dans la section 4.3.1.2.

Les algorithmes proposés ont été écrits en C et testés sur un PC de type Pentium 3 à 1 GHz avec le système d'exploitation Linux. Pour chaque instance, 10 exécutions de 100 minutes sont réalisées.

4.8.1. Génération des instances bi-objectif

Pour nos expérimentations, les benchmarks de Taillard [Tai 93a] offrent un bon support de comparaison. D'une part, ces problèmes sont très référencés dans la littérature [Ben 98][Now 99]. D'autre part, on connaît pour chaque instance, une borne inférieure du *makespan*, notée *LB*, ainsi qu'une borne supérieure pour le *makespan*, notée *UB*, correspondant à la meilleure solution obtenue par Taillard pour cette instance. Pour les instances de base, 8 instances différentes sont proposées pour plusieurs tailles de problèmes. Leur taille varie de 20 à 50 tâches et de 5 à 20 machines. Les temps d'exécution des tâches sur les machines sont choisis aléatoirement (entier entre 1 et 100). Pour notre étude, nous avons dû étendre les instances, en ajoutant à chaque tâche une date limite de fin d'ordonnancement, ceci afin de prendre en compte l'approche bi-objectif. Ces dates ont également été construites aléatoirement, la fourchette des valeurs pouvant être prise par ces dates étant comprise entre

le temps moyen de complétude de la première tâche (soit ((0+100)/2)*M, où M est le nombre de machines de l'instance), et la meilleure date de complétude, comme dans la littérature.

L'algorithme proposé par *Taillard* consiste en la génération d'une instance de problème à partir des trois données d'entrées : N, M et un entier *SEED*. Nous étions donc contraints de modifier ce procédé, afin de fournir des instances comportant des temps de retard. L'algorithme suivant (figure 4.22) décrit notre générateur d'instances du *flow-shop* bi-objectif, où N, M, *SEED*, *UB* sont des variables globales données par l'utilisateur. Cet algorithme génère les mêmes instances que celles de *Taillard*, munies en plus des temps limite l_i.

4.8.2. Qualité des solutions extrêmes obtenues

Le tableau 4.2 présente les résultats de tests effectués sur 8 instances de *Taillard étendu* au cas bi-objectif notées ma_ta*xx*_bi. *MM* désigne le meilleur *makespan* obtenu, *MR* le temps minimal de retard enregistré. Le paramètre *dév* mesure l'écart entre le meilleur *makespan* obtenu et *UB*. |*PO*| indique le nombre d'individus du front Pareto obtenus.

Les résultats du tableau 4.2 montrent la capacité de l'algorithme à trouver des solutions de bonne qualité pour le *makespan*. Les petites déviations sont justifiables. En effet, l'algorithme proposé est indépendant du problème traité, car aucune heuristique ne minimisant le *makespan* ou le retard n'est utilisé. De plus, l'ensemble *Pareto* trouvé est assez dispersé, ce qui nous offre un bon échantillonnage du front de *Pareto*.

```
Algorithme : Génération_inst_biobjectif
Début

fonction unif1: entier
début
      m:=2147483647;   a:=16807;   b:=127773;   c:=2836;
      k:=SEED / b;                              {division entière}
      SEED:=a*(SEED modulo b)-k*c;
      si (SEED < 0) alors SEED:=SEED+m;
      valeur_0_1:=SEED/m ;            {division non entière}
      retourner inf+(valeur_0_1*(sup-inf+1));
fin

fonction unif2 : entier
début
      m:=2147483647;   a:=16807;   b:=127773;   c:=2836;
      k:=SEED2/b;                           {division entière}
      SEED2:=a*(SEED2 modulo b)-k*c;
      si (SEED2 < 0) alors  SEED2:=SEED2+m;
      valeur_0_1:=SEED2/m ;                 {division non entière}
      retourner 12.0*UB/30.0+valeur_0_1*17.0*UB/30.0);
fin

procédure Générer_instance
début
     SEED2 := SEED * 2;
     Pour j := 1 à M
          Pour i := 1 à N
            faire d[i][j] := unif1;
                  {d[i][j] désigne la durée de la opération t_ij }
```

```
            fait
      Pour i := 1 à N
        faire limit[i]=unif2;
              {limit[i] désigne la date limite l_i de la tâche J_i}
        fait
fin;
Fin.
```

Figure 4.22. Procédure de génération des instances du *flow-shop* bi-objectif.

Instance	*Taille Instance (N x M)*	*SEED*	*LB*	*UB*	*MM*	*Dév*	*MR*	*|PO|*	*Nb. de générations*
Ma_ta01_bi	20x5	873654221	1232	**1278**	**1278**	0 %	453	4	50 000
Ma_ta02_bi	20x5	379008056	1290	**1359**	**1359**	0 %	491	6	50 000
Ma_ta11_bi	20x10	587595453	1448	**1582**	**1586**	0.25%	1508	28	80 000
Ma_ta12_bi	20x10	1401007982	1479	**1659**	**1674**	0.9%	1342	21	80 000
Ma_ta21_bi	20x20	479340445	1911	**2297**	**2330**	1.43%	1062	32	200 000
Ma_ta31_bi	50x5	1328042058	2712	**2724**	**2735**	0.4%	3629	11	200 000
Ma_ta41_bi	50x10	1958948863	2907	**3037**	**3126**	2.93%	6653	24	200 000
Ma_ta51_bi	50x20	1539989115	3480	**3886**	**3990**	2.67%	11379	32	300 000

Tableau 4.2: Performance de l'algorithme génétique hybride.

4.8.3. Qualité des fronts de Pareto obtenus

Nous avons évalué les fronts obtenus par notre algorithme génétique de base (avec élitisme-NSGA, mais sans la recherche locale, ni le partage) noté *AGb*, à l'aide des métriques *Contribution* et *S* (voir la section 4.2.3). La métrique *contribution* nécessite de connaître un ensemble de Pareto de "référence" auquel nous nous comparons. Nous avons décidé d'utiliser l'algorithme génétique adaptatif, *AGA,* dû à Basseur [Bas 05], afin de générer cet ensemble de référence. En effet, le choix de cet algorithme est principalement dû au fait que nous utilisons les mêmes objectifs et que *AGA* est postérieur à notre algorithme.
L'algorithme génétique *AGA* est similaire à notre algorithme *AGb* auquel sont ajoutés deux mécanismes favorisant l'adaptativité et la capacité d'exploration des algorithmes génétiques multiobjectif. Le premier mécanisme permet d'utiliser plusieurs opérateurs de mutation simultanément durant l'algorithme génétique, en favorisant les meilleurs d'entre eux. Le deuxième mécanisme influe sur le paramétrage du mécanisme de diversification, en l'adaptant selon la disposition des solutions non-dominées.

Les tableaux 4.3, 4.4 et 4.5 présentent les valeurs minimale (*Min*), maximale (*Max*), moyenne (*Moy*) et l'écart type (*ET*) obtenues pour ces métriques pour l'ensemble des exécutions réalisées pour chaque instance testée. Notons que ces fronts sont obtenus avec la même population initiale pour les deux algorithmes. Dans le cadre de la métrique *S*, nous

utilisons le point de référence *Nadir* des ensembles Pareto comparés. Rappelons que le point *Nadir* est le vecteur des pires valeurs pour chaque objectif.

Instance	***Taille inst.***	***Min***	***Max***	***Moy***	***ET***
Ma_ta01_bi	20x5	0.25	1.00	0.613	0.271
Ma_ta02_bi	20x5	0.42	1.00	0.662	0.201
Ma_ta11_bi	20x10	0.14	0.88	0.571	0.221
Ma_ta12_bi	20x10	0.32	0.89	0.524	0.150
Ma_ta21_bi	20x20	0.22	0.82	0.553	0.154
Ma_ta31_bi	50x5	0.22	1.00	0.711	0.308
Ma_ta41_bi	50x10	0.00	1.00	0.607	0.320
Ma_ta51_bi	50x20	0.00	0.31	0.126	0.114

Tableau 4.3. Evaluation des performances : *Contribution*(*AGA/AGb*). *Contribution*(*AGA/AGb*)+*Contribution*(*AGb/AGA*)=1.

Le tableau 4.3 [Bas 05] montre que, dans la plupart des instances, les fronts obtenus par *AGb* sont moins bons que ceux obtenus par l'algorithme génétique adaptatif *AGA* pour la métrique *Contribution*. En effet, en comparant les fronts obtenus, une moyenne de 45% des solutions Pareto sont fournies par *AGb*. Il est cependant important de signaler que pour l'instance la plus grande (*Ma_ta51_bi*) notre algorithme fournit de meilleures solutions (84% de moyenne). Notons également que les résultats varient beaucoup selon les exécutions, ce qui montre une certaine faiblesse de convergence. Cette disparité conforte l'idée d'hybridation de notre algorithme et éventuellement une parallélisation de celui-ci.

Instance	***Taille inst.***	***Min***	***Max***	***Moy***	***ET***
Ma_ta01_bi	20x5	3612	4393	4166.9	317.8
Ma_ta02_bi	20x5	8935	10657	9995.7	718.8
Ma_ta11_bi	20x10	231224	252835	240980.2	8024.3
Ma_ta12_bi	20x10	155695	172801	164826.7	5534.6
Ma_ta21_bi	20x20	377979	502144	430951.4	40333.0
Ma_ta31_bi	50x5	131781	177186	158228.9	14691.5
Ma_ta41_bi	50x10	409150	774568	593797.7	115172.3
Ma_ta51_bi	50x20	2553558	3131650	2732590.2	191457.9

Tableau 4.4. Evaluation des performances : *S*(*AGb*).

Les tableaux 4.4 et 4.5 réalisent la comparaison à l'aide de la métrique *S*. On peut faire, à peu près, les mêmes remarques que celles à propos de la métrique *Contribution*. Néanmoins,

la mesure permet ici de quantifier plus précisément les différences entre fronts de Pareto obtenus par les deux algorithmes (*S* tient en partie compte de la diversité).

Instance	*Taille inst.*	*Min*	*Max*	*Moy*	*ET*
Ma_ta01_bi	20x5	3837	4449	4247.8	268.0
Ma_ta02_bi	20x5	10594	10948	10770.1	164.7
Ma_ta11_bi	20x10	234171	252950	245016.9	7136.8
Ma_ta12_bi	20x10	158874	173121	166520.8	3806.5
Ma_ta21_bi	20x20	411872	501477	461176.9	38142.1
Ma_ta31_bi	50x5	155403	186970	174577.0	9783.6
Ma_ta41_bi	50x10	467242	806158	639182.3	102171.7
Ma_ta51_bi	50x20	2152415	2714895	2418535.2	171430.6

Tableau 4.5. *S*(*AGA*) [Bas 05].

Afin d'explorer plus de solutions en un temps donné et afin d'améliorer la qualité des résultats, la parallélisation est une alternative envisageable. Nous présentons, dans la suite, une version parallèle de l'algorithme génétique proposé.

4.9. Algorithmes génétiques parallèles

Nous proposons dans cette section une version parallèle de notre algorithme génétique le plus performant, c'est-à-dire celui qui utilise l'élitisme, le partage combiné et la recherche locale.
Le mécanisme de parallélisation que nous avons adopté se base sur le modèle distribué [Tal 95]. Dans ce modèle, appelé aussi *modèle à décomposition*, plusieurs sous-populations sont construites, chaque processus exécute l'algorithme génétique principal sur une sous-population qui lui est associée. A des intervalles réguliers en terme de générations, les processus procèdent à un échange d'individus. Ce phénomène est connu sous l'appellation de *migration* (figure 4.23). Le voisinage d'un individu, la fréquence d'échange, la stratégie du choix de l'individu migrateur et de l'individu à remplacer lors de la réception d'un nouvel individu sont autant de paramètres de l'algorithme.

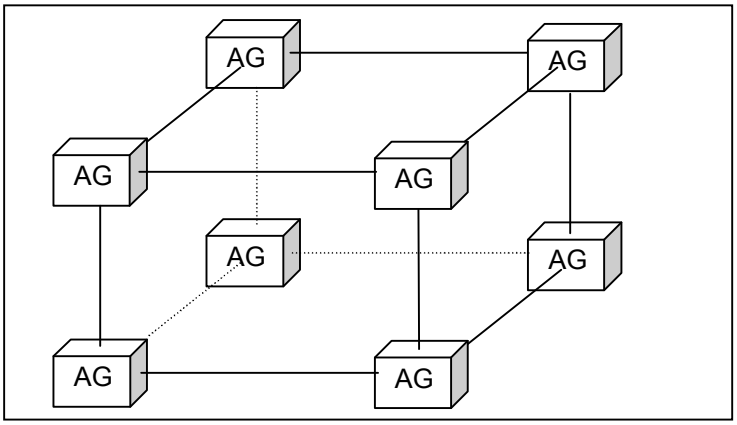


Figure 4.23. Modèle distribute.

Ce modèle présente plusieurs avantages : il est indépendant de l'architecture de la machine (nombre de processeurs). Il donne de bonnes performances pour un nombre de processus restreints. L'inconvénient principal reste le fait que la disposition naturelle des algorithmes génétiques à être parallélisés n'est pas complètement exploitée. L'application de l'algorithme génétique sur chaque sous population étant réalisée de manière séquentielle [Tal 95].
Le taux de diversification et d'intensification est largement affecté par le choix des paramètres de l'algorithme. Quand le taux de connexion entre processus est élevé, ceci tend à favoriser l'implantation des meilleurs individus dans toutes les sous-populations, ce qui du coup accentue l'intensification. Une fréquence d'échange élevée intensifie la coopération entre processus fortement connectés, ce qui débouche sur des sous-populations qui tendent rapidement à être similaires. L'utilisation de plusieurs sous-populations relativement indépendantes a pour effet d'engendrer une dispersion de la recherche dans l'espace, due à une pression de sélection moins importante. Ce qui est d'un grand apport pour la diversification.

Nous avons opté pour une topologie de communication en anneau (figure 4.24). Ce choix est motivé par le souhait de minimiser le taux de communication inter-processus, tout en maintenant l'aspect fortement connexe du graphe. Ce choix garantit aussi qu'un bon individu pourra se propager à toutes les sous-populations après un certain nombre de générations.

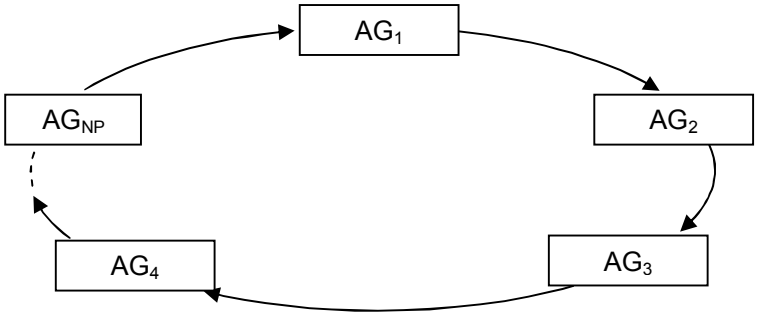


Figure 4.24. Topologie de communication en anneau.

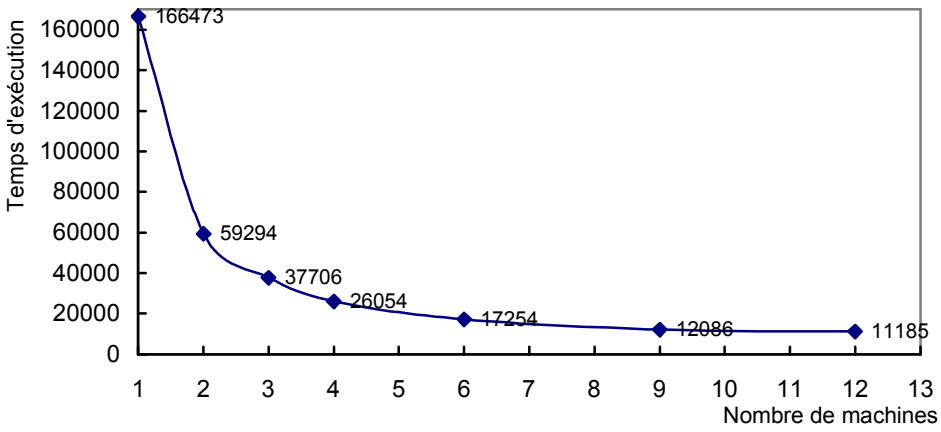


Figure 4.25. Temps d'exécution de l'algorithme génétique parallèle pour l'instance ma_ta21_bi en fonction du nombre de processeurs.

La figure 4.25 montre la variation des temps d'exécution (en secondes) de l'algorithme génétique parallèle, quand le nombre de machines (processus) augmente. Les tests ont été effectués sur un cluster de Stations *SUN ULTRA* 1. Le gain en temps d'exécution étant prouvé expérimentalement, ceci permet d'envisager d'augmenter les tailles des populations, ainsi que le nombre de générations limites, dans le but de trouver des solutions encore meilleures (voir le tableau 4.6).

Comme le montrent les figures 4.26 et 4.27, l'utilisation de populations de grande taille réparties sur différents algorithmes génétiques et l'augmentation du nombre de générations limites améliorent la qualité des solutions trouvées.

L'algorithme est réalisé sous l'environnement PVM (*Parallel Virtual Machine*). Cet environnement permet à des machines de natures différentes et fonctionnant sous des architectures système distinctes de travailler exactement comme si elles faisaient partie d'une même machine à plusieurs processeurs. Aucun équilibrage de charge n'est effectué dans cet environnement, de ce fait la vitesse d'exécution de l'algorithme dépend de la vitesse de la plus faible machine, ainsi que de la charge externe qui circule dans le système.

Afin d'accroître la qualité des solutions obtenues par les différents algorithmes génétiques proposés, nous donnons, dans la suite, un autre générateur de population initiale, de sorte que les individus de celle-ci soient de bonne qualité.

		AG Séquentiel						AG Parallèle					
Instance	**UB**	**MM**	**Dév**	**MR**	**|PO|**	**tp**	**Nb gén**	**MM**	**Dév**	**MR**	**|PO|**	**tp**	**Nb gén**
ma_ta11_bi	1582	1586	0.25%	1508	28	200	80000	1583	0.06%	1431	32	300	300000
ma_ta21_bi	2297	2330	1,43%	1062	32	200	200000	2305	0.34%	1057	29	300	300000

Tableau 4.6. Amélioration de la qualité des solutions pour les instances ma_ta11_bi et ma_ta21_bi. *MM* désigne le meilleur *makespan* obtenu, *MR* le temps minimal de retard enregistré, *dév* mesure l'écart entre le meilleur *makespan* obtenu et *UB*. |*PO*| calcule le nombre d'individus du front de Pareto obtenus, tp : taille de la population et Nb gén : le nombre de générations.

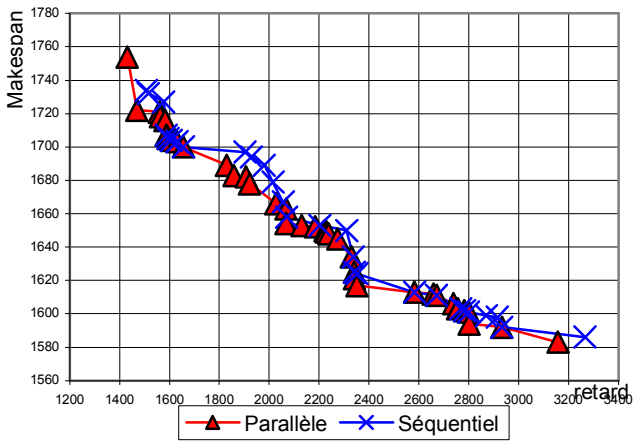


Figure 4.26. Amélioration du front de Pareto par introduction du parallélisme pour l'instance ma_ta11_bi.

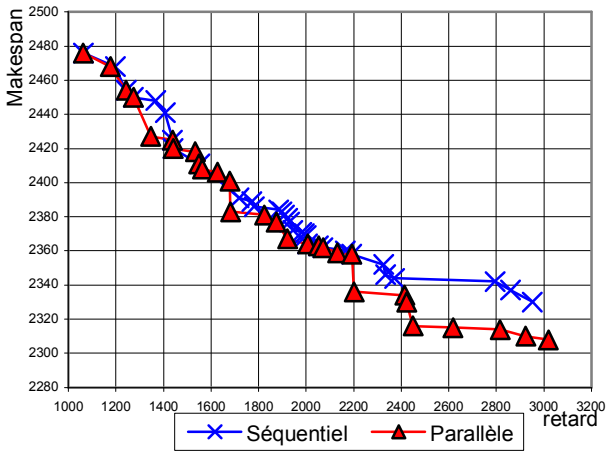


Figure 4.27. Amélioration du front de Pareto par introduction du parallélisme pour l'instance ma_ta21_bi.

4.10. Heuristique *NEH* pour la génération de la population initiale

Nous proposons dans cette section une variante probabiliste de l'heuristique *NEH* due à *Nawaz*, *Escore* et *Ham* [Naw 83], conduisant à la génération d'une population d'individus de bonne qualité. Dans cette variante, une solution est générée par élaboration progressive de plusieurs séquencements partiels. A chaque itération, une tâche J_i, de durée d'exécution d_{ij}, est sélectionnée pour être insérée dans la séquence partielle δ, avec une probabilité égale à :

$$\left(\sum_{j=1}^{M} d_{ij}\right)^{\alpha} \Big/ \sum_{k\in\Phi}\left(\sum_{j=1}^{M} d_{kj}\right)^{\alpha}$$

où Φ correspond à l'ensemble des tâches non encore planifiées. Une fois qu'une tâche est sélectionnée, elle est insérée dans la séquence δ à la position engendrant le meilleur *makespan* ou le meilleur temps de retard. Le nombre α qui figure dans la formule ci-dessus représente un paramètre de contrôle de l'heuristique, dont la valeur est comprise entre]0, +∞ [. Une valeur petite de α conduit à des solutions de modeste qualité bien réparties dans l'espace. Une valeur élevée du paramètre engendre des solutions de bonne qualité concentrées dans des régions attractives de l'espace de recherche. Nous donnons ci-dessous la procédure de génération de la population initiale à base de l'heuristique *NEH* (figure 4.28).

La moitié des individus de la population initiale sont engendrés sur la base de la minimisation du *makespan*, l'autre moitié se base sur la minimisation du temps des retards. Les tests repris dans le tableau 4.7 représentent une comparaison avec l'algorithme génétique séquentiel décrit précédemment. Le paramètre α est fixé à 1.

Pour les deux problèmes des figures 4.29 et 4.30, l'utilisation de l'heuristique *NEH* comme outil de génération de la population initiale fournit de meilleurs résultats. L'analyse des frontières de Pareto produites par chacune des deux approches (figures 4.29 et 4.30) fait apparaître un plus large front dans le cas d'une génération *NEH* de la population initiale.

```
Algorithme : NEH_modifié
Début
Pour chaque individus ind de la population initiale
faire
   Φ := {l'ensemble de tout les tâches} ;
   δ := nil ;
   Pour chaque tâche Ji
      faire calculer les temps de traitements D[i]= somme(D[k])
           /*D représente le vecteur des durées d'exécution des
             tâches*/
      fait
   choisir une tâche Jm avec la probabilité D[m]^α/somme(D[k]^α)
   D[m]:= 0 ; Φ := Φ - Jm ;
   choisir une tâche Jn avec la probabilité D[n]^α/ somme(D[k]^α)
   D[n]:= 0 ; Φ := Φ - Jn ;
   Si Jm→Jn est meilleure que Jn→Jm alors δ = Jm→Jn sinon δ = Jn→Jm
```

```
   Pour k := 3 à N
   faire
     choisir une tâche Jm avec la probabilité D[m]^α/somme(D[k]^α)
     D[m]:= 0 ;  Φ := Φ - Jm ;
     Générer l'ensemble des séquences construites par introduction
     de Jm dans δ à l'une des k positions possibles ;
     Si ind < tp/2  alors δ = la séquence avec le meilleur makespan
     sinon δ = la séquence avec le meilleur temps retard fin si
   fait;
fait;
FIN.
```

Figure 4.28. Algorithme de génération de la population initiale : *NEH_modifié.*

Instance	UB	AG séquentiel avec génération aléatoire de la population initiale						AG séquentiel avec génération de la population initiale avec *NEH_modifié*					
		MM	Dév	MR	|PO|	tp	Nb gén	MM	Dév	MR	|PO|	tp	Nb gén
Ma_ta11_bi	1582	1586	0.25%	1508	28	200	80000	1586	0.25%	1392	37	200	80000
Ma_ta12_bi	1659	1674	0.9%	1342	21	200	80000	1672	0.78%	1342	21	200	80000
Ma_ta21_bi	2297	2330	1.43%	1062	32	200	200000	2303	0.26%	1097	30	200	200000

Tableau 4.7. Comparaison des méthodes de génération aléatoire et *NEH_modifié.*

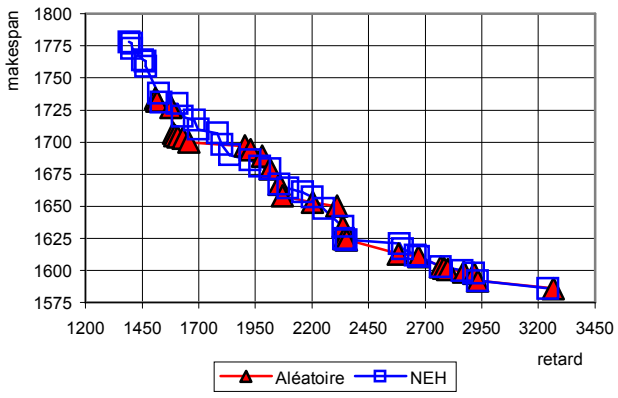


Figure 4.29. Comparaison des fronts de Pareto des approches de génération de la population initiale aléatoire et *NEH* pour le problème ma_ta11_bi.

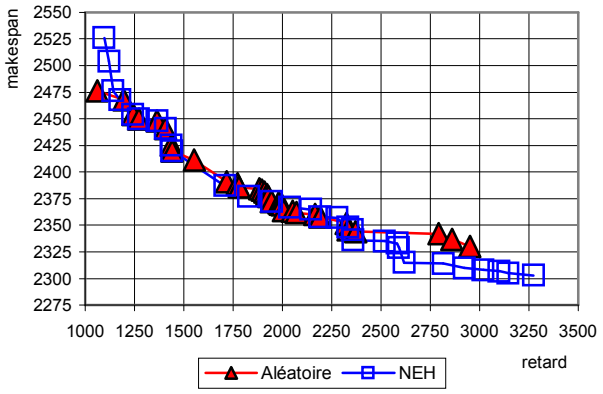


Figure 4.30. Comparaison des fronts de Pareto des approches de génération de la population initiale aléatoire et *NEH* pour le problème ma_ta21_bi.

4.11. Conclusion

Pour la résolution du problème du *flow-shop* bi-objectif, nous sommes partis d'un algorithme génétique générique, auquel nous avons ajouté différents ingrédients, tels que des mécanismes de sélection spécialement adaptés au cas bi-objectif, de l'élitisme, des mécanismes de maintien de la diversité et une hybridation avec une recherche locale. A chaque étape, nous avons illustré l'apport du mécanisme introduit, ce qui nous permet de formuler les conclusions suivantes.

Les stratégies de sélection (NSGA, NDS, WAR) implémentées semblent être bien adaptées au cas multiobjectif. L'efficacité de telles méthodes est améliorée avec l'introduction de l'élitisme lors de la phase de sélection. Cette phase de sélection consiste à réaliser la sélection des individus aussi bien de la population courante que des solutions non dominées trouvées pendant la recherche.
Cependant, le risque de la dérive génétique reste présent. Autrement dit, le risque qu'une population s'homogénéise à cause des erreurs stochastiques, et ainsi converge vers un minimum local. Les stratégies de diversification permettent d'éviter de tels problèmes. Trois variantes de la méthode de partage (*sharing*) ont été développées.
Le partage phénotypique (c'est-à-dire lorsque la distance entre deux individus est calculée dans l'espace objectif) apparaît être le plus intéressant pour obtenir une courbe de Pareto plus large et mieux dispersée ; la diversification génotypique (c'est-à-dire lorsque la distance entre deux individus est calculée dans l'espace de décision), quant à elle, fournit des résultats de meilleure qualité. La combinaison des deux méthodes améliore considérablement la qualité de la courbe de Pareto obtenue.
Ensuite, la recherche locale a été utilisée afin d'améliorer encore les résultats. Nous exécutons l'algorithme génétique en premier, afin d'avoir une première approximation de la courbe de Pareto. La recherche locale améliore ensuite chacune de ces solutions. L'apport de cette hybridation apparaît surtout pour des problèmes de grande taille.

Nous avons proposé, l'extension des instances de *Taillard* [Tai 93] au cas bi-objectif, afin de pouvoir tester l'algorithme génétique que nous proposons. Les tests effectués montrent la capacité de l'algorithme génétique à atteindre des solutions de faible coût en terme de durée d'exécution (*makespan*).

Cependant, l'algorithme génétique ainsi conçu se ralentit à cause des mécanismes de sélection et de diversification. La parallélisation se présente alors comme un moyen intéressant pour surmonter cet inconvénient. La réduction du temps nécessaire pour faire évoluer les solutions permet ainsi d'utiliser une population de plus grande taille et permet de faire davantage d'itérations, ce qui a pour effet d'améliorer la qualité du front de Pareto trouvé.

Nous avons aussi illustré l'impact qu'a le choix de la population initiale sur la qualité de la courbe de Pareto trouvée. Nous avons proposé une nouvelle variante probabiliste de l'heuristique *NEH*, qui permet d'avoir des solutions de départ à la fois de bonne qualité et bien réparties dans l'espace de recherche.

Chapitre 5

Le problème bi-objectif de tournées de véhicules avec des fenêtres horaires

Ce chapitre est consacré aux problèmes de tournées en général et aux problèmes de tournées de véhicules multiobjectifs plus particulièrement. Tout d'abord, nous faisons une présentation générale des problèmes de tournées.
Dans un second temps, une introduction plus détaillée de deux problèmes de tournées particuliers est faite. Ces deux problèmes sont le problème d'élaboration de tournées de véhicules et une extension de celui-ci : le problème de tournées de véhicules avec des fenêtres horaires (Vehicles Routing Problem With Time Windows – VRPTW). Ce dernier est la version monoobjectif du problème multiobjectif étudié au cours de cette thèse. Pour chaque problème, nous présentons un état de l'art des méthodes de résolution.
Nous proposons ensuite des d'algorithmes génétiques Pareto ainsi qu'une étude détaillée des différentes stratégies d'optimisation pour la résolution du VRPTW multiobjectif.
Nous achevons ce chapitre avec une étude de complexité du VRPTW.

Les travaux décrits dans ce chapitre ont fait l'objet de communications à MIC 2001 [Rah 01], EA'2003 [Rah 03a] et Roadef'2003 [Rah 03b]. Une publication au Journal of Combinatorial Mathematics and Combinatorial Computing : JCMCC [Rah 07].

5.1. Introduction

De nombreux secteurs de l'industrie, en particulier dans les transports, sont concernés par des problèmes de logistique et d'organisation complexes. Les problèmes de tournées, qui sont une facette de la logistique, font partie des principaux problèmes étudiés en recherche opérationnelle et modélisent bien de telles situations. En effet, de nombreux enjeux théoriques, mais aussi pratiques et économiques, sont liés à cette famille de problèmes.
Le terme problème de tournées englobe un grand nombre de problèmes très disparates. On peut citer par exemple les transports scolaires, les tournées des transporteurs de fonds, le transport de produits (livraison d'hydrocarbures, collecte de lait, etc.) et autres.
Le but ici n'est pas de présenter tous les problèmes que l'on peut rencontrer, mais plutôt de donner une idée des différents éléments qui peuvent être considérés dans un problème de tournées. Une attention particulière sera donnée au problème de tournées de véhicules avec des fenêtres horaires (VRPTW), dont la version multiobjectif est traitée dans cette thèse.

La section 5.2 de ce chapitre fournit une description générale des caractéristiques que l'on peut rencontrer dans un problème de tournées. La section 5.3 présente le problème de tournées de véhicules (VRP). La section 5.4 est consacrée à une extension du VRP de base, à savoir le VRPTW. La section 5.5 expose les motivations des problèmes de tournées multiobjectifs, ainsi que les différentes méthodes de résolution. La section 5.6 propose une approche Pareto à base d'algorithmes génétiques pour la résolution du VRPTW bi-objectif avec, comme critères retenus, le *nombre de véhicules* et la *distance totale* parcourue par les véhicules. Une étude comparative de plusieurs opérateurs génétiques et de plusieurs mécanismes de sélection et de diversification accompagnera l'implémentation des différents algorithmes de résolution proposés. La section 5.7, traite de la complexité du VRPTW et répond aux deux questions suivantes : est-il NP-complet de trouver une solution réalisable de VRPTW ? Etant donnée une solution réalisable, est-il NP-complet de trouver une bonne approximation de l'optimum d'une instance de VRPTW ?

A noter que tout au long de ce chapitre, on utilisera indifféremment les termes de "tournée" et de "routage" de véhicules.

5.2. Présentation générale des problèmes de tournées

Un problème de tournées consiste à rechercher une tournée ou un ensemble de tournées sur un réseau ou un sous-réseau, sous certaines contraintes en optimisant un (ou plusieurs) objectif(s) fixé(s). L'exemple de problèmes de tournées le plus connu est sans doute le problème du voyageur de commerce. Dans ce problème, étant donné un graphe complet valué $G = (S, A)$, le but est de trouver un circuit de longueur minimale passant exactement par chaque noeud de S.
Du point de vue de la théorie des graphes, une tournée sur un graphe $G = (S, A)$ est un circuit sur un sous-ensemble $H \subseteq S$. Il est possible de définir un problème de tournées en se basant sur les catégories suivantes : le réseau, la demande, la flotte, le coût et l'objectif.

5.2.1. Le réseau

Le réseau est un graphe $G = (S, A)$. Selon les problèmes, les sommets peuvent représenter des villes à visiter, des clients à servir, des points de passage, des dépôts etc.

Les arcs représentent les liaisons existantes entre ces derniers. Les arcs peuvent donc être des routes, des canalisations ou des connexions symboliques etc. Une action de collecte ou de livraison peut être définie le long des liaisons. Le réseau peut être symétrique, les liaisons entre deux noeuds étant représentées par des arêtes, ou asymétrique, une liaison étant symbolisée par un arc. Les réseaux urbains sont typiquement modélisés par des graphes asymétriques, les arêtes représentant les rues à double sens et les arcs les sens uniques par exemple. Les réseaux interurbains sont le plus souvent modélisés par des graphes symétriques. Aux arcs et aux arêtes sont souvent associées des valeurs numériques, qui indiquent les longueurs des liaisons ainsi que des temps de parcours qui peuvent dépendre du type de véhicule utilisé et de la période de traversée. Dans certains cas, les noeuds et les arcs ne peuvent être desservis qu'à certains moments précis, comme par exemple des horaires d'ouverture. Des fenêtres de temps sont alors associées à chaque noeud ou arc. Elles peuvent être de deux sortes : dures ou souples. Dans le premier cas, si la tournée permet que l'on arrive en avance, il faut attendre jusqu'à la borne inférieure de la fenêtre et il est interdit d'arriver en retard. Dans le cas des fenêtres de temps souples, la violation des contraintes est permise, mais entraîne une pénalité pour la fonction objectif.

5.2.2. La demande

La demande peut être associée soit aux nœuds - on parle alors de problèmes de tournées sur les nœuds - soit aux arcs - on parle de problèmes de tournées sur les arcs. Elle peut être fixe ou stochastique, auquel cas elle est donnée par des formules probabilistes. Enfin, la demande peut porter sur un seul produit ou sur plusieurs.
Le plus souvent, on parle de demande lorsque le problème traité est un problème de distribution, c'est-à-dire un problème où il s'agit de livrer à chaque noeud - on parle alors de clients - ou arc, une certaine quantité de marchandise ou de produit. Les problèmes de ramassage, où il s'agit de prendre chez les clients une certaine quantité de marchandise, peuvent être considérés comme équivalents. Cependant, il existe des problèmes dits de collecte et de livraison dans lesquels une quantité de produit doit d'abord être collectée chez un premier client, pour ensuite être livrée chez un autre.
Il est possible d'imposer que la demande soit satisfaite en une seule fois. Si ce n'est pas le cas, la demande d'un client peut être divisée entre plusieurs tournées. Il se peut qu'il ne soit pas possible ou nécessaire de satisfaire toutes les demandes. Par exemple, dans un problème de livraison, certains clients peuvent être ignorés.

5.2.3. La flotte

Les tournées correspondent souvent aux chemins que doit suivre un véhicule ou une flotte de véhicules pour visiter les noeuds et arcs du graphe. La première caractéristique de la flotte est le nombre de véhicules la composant. Ce nombre peut être fixe ou non. Toutefois, même s'il n'y a qu'un seul véhicule, cela ne signifie pas que la solution du problème est une tournée unique. Dans les problèmes de distribution par exemple, il est possible pour un seul véhicule de faire plusieurs tournées si l'ensemble de la demande est trop important pour y pourvoir en une seule fois. Les véhicules composant la flotte peuvent avoir plusieurs caractéristiques, qui limitent ou conditionnent leur utilisation :
- un véhicule doit commencer et terminer sa route en un noeud précis du graphe appelé dépôt. Il se peut que le véhicule ait le choix entre plusieurs dépôts possibles. Le choix du dépôt est alors fixé lors de la résolution du problème.
- les véhicules peuvent avoir une capacité maximale exprimée en termes de poids, de volume etc.

- différentes limitations peuvent être définies, comme la distance ou le temps maximal(e) avant que le véhicule ne retourne à son dépôt. Cela représente par exemple la limite de carburant pour le véhicule ou l'impossibilité humaine ou légale, pour un conducteur, de conduire de manière continue plus d'un certain nombre d'heures.

La flotte peut être homogène, c'est-à-dire composée d'un seul type de véhicules, ou hétérogène. Les différences peuvent alors porter sur la capacité des véhicules, certaines caractéristiques (couverts ou non couverts) ou certaines particularités (véhicules frigorifiques).

5.2.4. Les coûts

Généralement, le coût est fixe pour le véhicule et variable pour son utilisation, en fonction de la distance parcourue ou du temps écoulé. Le coût prend aussi en compte les pénalités de service si les clients sont servis en retard ou de manière incomplète, voire non servis. Un gain peut être associé aux noeuds ou aux liaisons ; le gain étant collecté lors de la visite du noeud ou de la liaison.

5.2.5. L'objectif

De nombreux objectifs différents peuvent être imaginés. Les plus couramment utilisés sont :

- la minimisation de la longueur totale parcourue par les véhicules,
- la minimisation de la durée totale des tournées,
- la minimisation du coût total des tournées,
- la minimisation de la somme des coûts fixes et des coûts variables de l'utilisation des véhicules,
- la minimisation de la taille de la flotte,
- la maximisation de la qualité de service offert,
- la maximisation du profit collecté par la visite chez des clients choisis,
- ...

Ces fonctions objectifs peuvent être calculées sur une période unique, mais aussi sur plusieurs périodes (heure, jour, semaine ...). Il est alors nécessaire d'affecter les véhicules et les visites aux différentes périodes.
Les objectifs cités sont souvent contradictoires. Il est possible de les combiner dans la définition de problèmes de tournées multiobjectifs.

5.3. Le problème d'élaboration de tournées de véhicules (VRP)

5.3.1. Présentation

Introduit par Dantzig et Ramser [Dan 59], le problème d'élaboration de tournées de véhicules (*Vehicle Routing Problem - VRP*), se définit sur un graphe $G = (V, E)$ où l'ensemble des sommets $V = \{1,2, \ldots, n\}$ représente le dépôt (le sommet 1) et les *n-1 clients*. E est l'ensemble des arcs reliant les sommets. On doit servir à chaque client une quantité q_i d'un

bien (i=1,…,n) depuis le dépôt. Pour effectuer les livraisons, une flotte de véhicules est disponible (nombre de véhicules connu et éventuellement en quantité suffisante). Les véhicules sont limités par la quantité Q de biens qu'ils peuvent transporter. Une solution du problème d'élaboration de tournées de véhicules est une collection de tournées.
Chaque client appartient à une et une seule tournée et la demande totale sur chaque tournée ne peut pas dépasser Q. A chaque arc (i, j), on associe la distance entre les sommets i et j. Une solution de longueur minimale (somme des longueurs des tournées de chaque véhicule) est recherchée lors de la résolution du problème. Un exemple de solution du VRP est donné dans la figure 5.1. Le problème a été prouvé NP-difficile [Len 81].

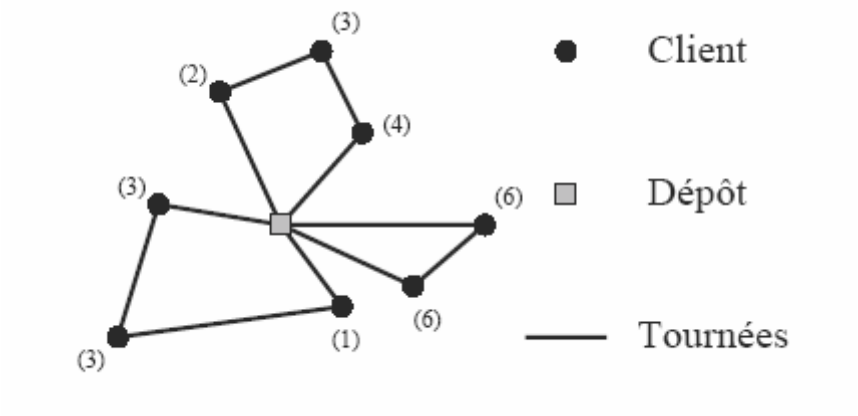


Figure 5.1. Un exemple de solution pour le problème d'élaboration de tournées de véhicules. Les demandes des clients sont indiquées dans les parenthèses et la capacité des véhicules est de 12.

5.3.2. Métaheuristiques pour le VRP

Le but de cette section n'est pas d'exposer toutes les métaheuristiques existantes, ni le plus grand nombre de références pour les problèmes de tournées de véhicules. L'objectif est de présenter brièvement les métaheuristiques les plus utilisées pour le VRP.

Les premières implémentations d'un recuit simulé pour le VRP sont celles de Robusté et al. [Rob 94] et Alfa et al. [Alf 91]. Une implémentation plus évoluée du recuit simulé est celle proposée par Osman [Osm 93].

Durant la dernière décennie, la recherche tabou a de plus en plus été appliquée au VRP. Même si les premières implémentations [Wil 89][Pur 91] n'offraient pas des résultats satisfaisants, les travaux suivants furent de plus en plus efficaces. Parmi ceux-ci, on peut citer les travaux de Taillard [Tai 93], de Gendreau et al. [Gen 94], de Rochat et Taillard [Roc 95], de Xu et Kelly [Xu 96], de Rego et Roucairol [Reg 96], de Toth et Vigo [Tot 03], de Barbarosoglu et Ozgur [Bar 99].

Peu d'études portent sur l'application des algorithmes génétiques au VRP, contrairement aux variantes plus complexes comme le VRP avec fenêtres de temps [Bla 93][Tha 93][Tha 95][Pot 96b]. Une première étude de comparaison d'un algorithme génétique avec des recuits simulés et des recherches tabou a été faite par Van Breedam [Van 96]. Par la

suite, Schmitt [Sch 94][Sch 95] proposa un algorithme génétique. Les résultats de cet algorithme ne sont pas de très bonne qualité, d'autant plus qu'il demande un temps de calcul important. Récemment, deux algorithmes génétiques efficaces ont été proposés pour le VRP. Le premier algorithme est proposé par Baker et Ayechew [Bak 03] et le second par Prins [Pri 04]. La première étude utilisant les colonies de fourmis a été proposée par Kawamura et al. [Kaw 98]. Puis par Reiman et al. [Rei 02][Rei 04].

5.4. Le VRP avec des fenêtres horaires : VRPTW

5.4.1. Présentation et formulation du VRPTW

Le VRP avec des fenêtres horaires (*Vehicle Routing Problem with Time Windows - VRPTW*) [Pot 96a][Brä 01] est une extension du VRP de base, auquel des contraintes de temps sur les clients sont ajoutées. Pour une liste exhaustive sur les nombreuses variantes du VRP, le lecteur peut se reporter à [Aro 96]. Dans ce problème, chaque client fournit deux limites de temps qui représentent la période pendant laquelle il désire être livré. Cette version possède elle même de nombreuses variantes, par exemple il existe deux types de fenêtres temporelles rencontrées dans la littérature [Lan 01] :

- Dans la variante *"Soft time windows"*, le véhicule peut arriver avant la borne inférieure ou après la borne supérieure de la fenêtre de temps. Si le véhicule arrive en avance, il doit attendre pour commencer le service. S'il est en retard, une pénalité est ajoutée à la fonction objectif.
- Dans la variante *"Hard time windows"*, le véhicule peut arriver avant la borne inférieure mais il doit obligatoirement arriver avant la borne supérieure de la fenêtre temporelle pour chaque client. Dans ce cas, la taille de la flotte de véhicules est une variable de décision, car la contrainte temporelle est ici une contrainte forte, jouant sur la faisabilité de l'instance à résoudre.

L'introduction de fenêtres horaires dans un problème de tournées de véhicules complique sensiblement l'énoncé initial. En effet, dans un VRP correctement défini, les demandes des clients sont compatibles avec la capacité des véhicules. Cette compatibilité assure l'existence d'une solution réalisable. Dans un VRPTW, où le service de chaque client est astreint à une fenêtre de temps simple, le nombre de solutions réalisables diminue, surtout si les fenêtres sont étroites. De plus, lorsque les fenêtres de deux clients se chevauchent, elles peuvent introduire une disjonction dans le parcours d'un véhicule. Ce dernier ne peut plus servir les deux clients dans la même tournée, ce qui supprime encore des solutions réalisables.

Nous formulons le VRPTW selon le modèle de [Sol 88]. Cette formulation complète celle du VRP vu dans la section 5.3.1. par des éléments spécifiques à la contrainte relative aux fenêtres de temps. L'ensemble des véhicules (identiques) servant les clients est nommé V. Chaque véhicule possède une capacité $Q \geq 0$ et chaque client $i \in C$ possède une demande $d_i \geq 0$ et un temps de service $s_i \geq 0$, qui correspond au temps nécessaire pour le déchargement. Les véhicules doivent quitter le dépôt durant la fenêtre horaire $[e_1, l_1]$, et y retourner durant la fenêtre horaire $[e_{n+1}, l_{n+1}]$. A chaque client, est associée une fenêtre de temps $[e_i, l_i]$, $i \in C$ pendant laquelle le service doit être fait. Il est possible qu'un véhicule arrive en avance chez un client, c'est-à-dire avant e_i ; dans ce cas le véhicule doit attendre jusqu'à ce que le service commence, on parle alors d'un temps d'attente w_i, $i \in$ C.

L'objectif est de trouver des itinéraires pour l'ensemble des véhicules de façon à minimiser une fonction objectif avec les contraintes suivantes :

- Chaque client doit être livré une et une seule fois.
- La capacité de chaque véhicule doit être respectée.
- Chaque tournée doit partir du dépôt et y retourner.
- Il faut tenir compte des fenêtres horaires pour chaque client.

5.4.2. Le VRPTW et les métaheuristiques

Vu l'importance économique du VRPTW, de nombreuses méthodes de résolution ont été proposées, allant des méthodes exactes aux métaheuristiques. Les algorithmes génétiques et la recherche tabou se sont révélés être parmi les techniques les plus prometteuses dans ce domaine et celles donnant les meilleurs résultats [Brä 01]. C'est pourquoi, parmi la multitude de méthodes existantes, nous nous penchons principalement et de manière chronologique sur l'application des algorithmes génétiques et de la recherche tabou au VRPTW.

Thangiah et al. [Tha 94] ont développé deux métaheuristiques basées sur l'approche à *deux-phases*. Dans un premier temps, une solution initiale est créée, par l'heuristique C*heapest insertion*, qui consiste à choisir, parmi tous les noeuds non insérés jusqu'ici, un noeud dont l'insertion induit la plus basse augmentation de la longueur de la tournée. La deuxième phase applique l'une des procédures de recherche suivantes, qui utilisent la stratégie *λ-interchange* : une procédure de recherche de descente utilisant le mécanisme de sélection "*premier élu"*, ou un algorithme de recuit simulé hybridé avec une recherche tabou.

Thangiah dans [Tha 95a] définit un algorithme nommé *GIDEON* basé sur la méthode "*grappe en premier, route en second*", qui affecte les clients aux véhicules en formant des grappes (clusters) engendrées par un algorithme génétique. Quelque temps après, Thangiah [Tha 95b] a développé une autre approche similaire à *GIDEON*, nommée *GenClust*. Des formes géométriques sont utilisées à la place des grappes (clusters) de clients.

Rochat et Taillard dans [Roc 95], ont proposé une recherche tabou basée sur "la mémoire adaptative" et le voisinage *2-opt* [Pot 95]. La mémoire adaptative est une structure qui rassemble des tournées (sous-routes) des meilleures solutions rencontrées durant la recherche. La sélection et la combinaison de ces routes donnent lieu à de nouvelles solutions, utilisées comme point de départ pour la recherche tabou.

Dans [Pot 96c], les auteurs ont proposé un algorithme dans lequel les opérateurs génétiques sont appliqués directement sur les solutions, ce qui évite le problème de codage des solutions.

Taillard et al. [Tai 97] ont appliqué une recherche tabou au VRPTW avec des contraintes temporelles "molles" (*Soft Time Windows)*.

Chiang et Russel [Chi 97] ont développé une recherche tabou qui utilise une liste tabou dont la longueur varie de manière dynamique.

Un algorithme génétique hybridé avec des heuristiques de construction a été employé dans [Ber 98b]. Les opérateurs de croisement proposés combinent itérativement plusieurs routes r_1 du parent P_1 avec un sous ensemble de clients, formés par les r_2 routes les plus proches du parent P_2, en terme de distance euclidienne entre les clients. Par la suite, les auteurs ont poursuivi leurs travaux en proposant un algorithme génétique qui fait évoluer deux populations en parallèle [Ber 01]. La première population est utilisée pour la minimisation de la distance, et la seconde essaie de minimiser le nombre de contraintes violées.

Dans [Hom 99], les auteurs ont proposé deux métaheuristiques évolutionnaires. D'autres travaux ont été poursuivis par ces auteurs en introduisant quelques modifications au niveau de la population initiale et des opérateurs utilisés.

Dans [Geh 99], les auteurs ont étudié une approche à *deux-phases*. La première phase utilise ES1 (stratégie évolutionnaire) avec une population composée d'un seul individu, pour minimiser le nombre de véhicules utilisés. La deuxième phase consiste en une recherche tabou conçue pour minimiser la distance totale parcourue et qui utilise les mêmes opérateurs de recherche que ES1. Gehring et Homberger [Geh 99] ont introduit par la suite quelques améliorations de cet algorithme parallèle [Geh 01].

Bräysy [Brä 99a][Brä 99b] a complété le travail de Berger et al. [Ber 98b] en proposant plusieurs opérateurs de croisement et de mutation, en testant plusieurs formes d'algorithmes génétiques et en utilisant plusieurs stratégies de sélection. Dans [Brä 00], Bräysy et al. ont proposé un algorithme génétique hybridé avec un algorithme évolutionnaire, qui consiste à utiliser différentes méthodes de recherche locale, ainsi que des heuristiques de construction inspirées de [Sol 87] et de [Tai 97].

Tan et al. [Tan 01] introduisent un algorithme génétique similaire à celui de Zhu [Zhu 00], dans lequel les solutions sont représentées par une suite d'entiers, la population initiale est une combinaison de solutions créées par l'heuristique de Solomon, et de solutions créées avec l'application du voisinage λ *-interchange* aux solutions précédentes. Wee Kit et al. [Wee 01] proposent un algorithme génétique hybridé avec une recherche tabou basée sur les opérateurs de voisinage suivants : *exchange*, *relocate* et *2-opt*.

Cordeau et al. dans [Cor 01], introduisent une procédure de recherche tabou pour le VRPTW et deux de ses extensions.

5.5. Problèmes de tournées multiobjectif

Les problèmes de tournées académiques nécessitent souvent des adaptations pour des applications pratiques. Une des manières de s'approcher des cas réels rencontrés en pratique consiste à considérer plusieurs objectifs. Dans le cas des problèmes de tournées, les objectifs peuvent être classés selon la composante du problème sur laquelle ils portent (voir figure 5.2) : la tournée (coût, *makespan* (minimiser la durée de la plus longue tournée), équilibre, profit (maximiser le gain), risque (minimiser les risques lors du transport de produits dangereux) ...), les noeuds ou arcs (fenêtres de temps, service du client, ...) et les ressources (nombre de véhicules, marchandises, ...).

5.5.1. Applications

Les approches multiobjectifs des problèmes de tournées ont principalement trois motivations :

- la résolution d'un problème réel, dans lequel plusieurs objectifs ont clairement été définis par le décideur,
- l'adaptation ou l'extension de problèmes académiques, dans le but de rendre les problèmes plus en phase avec des cas réels, ou pour étudier des objectifs complémentaires, sans abandonner toutefois l'objectif classique de minimisation de la distance parcourue
- pour généraliser un problème par l'inclusion d'objectifs supplémentaires. Typiquement, il s'agit de modéliser une ou plusieurs caractéristiques du problème, généralement des contraintes, et de les remplacer par des objectifs. Dans la littérature, cette stratégie a été plusieurs fois employée pour s'attaquer au problème d'élaboration de tournées de véhicules avec fenêtres de temps, où les fenêtres de temps, qui sont très contraignantes, sont supprimées et remplacées par un ou plusieurs objectifs [Hon 99][Gei 01][Rah 01][Bar 03][Rah 03].

  Pour le VRPTW, il peut aussi s'agir de traiter simultanément l'objectif de *minimisation de la longueur de la tournée* et de *minimisation du nombre de véhicules utilisés* [Gei 01][Rah 01][Bar 03][Rah 03][Tan 03b]. Ce dernier objectif peut avoir une signification économique : moins il y a de véhicules, moins il y a d'investissement (achats, essence, salaires des conducteurs ...) [Cor 02][Pac 03][Tan 03b].

  La modélisation classique du VRPTW comprend deux objectifs qui sont traités de manière lexicographique. Le premier objectif est la minimisation du nombre de véhicules, puis, pour un nombre de véhicules fixé, le deuxième objectif est la minimisation de la longueur de la solution. La plupart des études multiobjectif portant sur ce problème, à l'exception de l'étude de Hong et Park [Hon 99], comportent ces deux objectifs. Ceux-ci ne sont pas pris en compte de manière lexicographique, comme auparavant, mais sont considérés avec le même niveau de priorité.

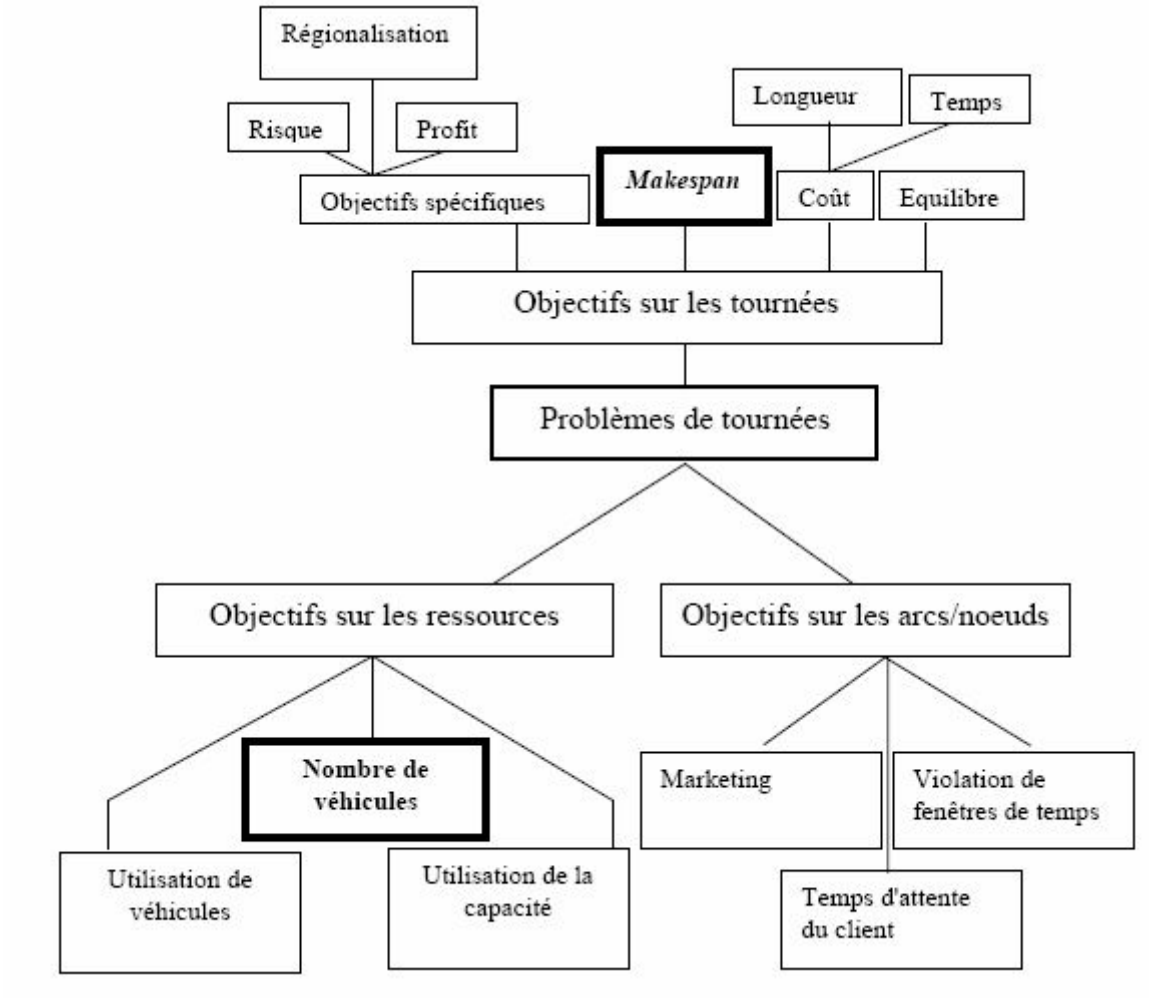


Figure 5.2. Objectifs pour les problèmes de tournées de véhicules.

5.5.1.1. Quelques exemples de cas réels

Compagnie de transport : dans l'étude de El-Sherbeny [Els 01], le problème envisagé comporte huit objectifs fixés par une compagnie de transport belge (fenêtres de temps, contrainte sur l'endroit de chargement, contrainte sur le transfert de matière, contrainte sur l'endroit de déchargement, contrainte sur le temps maximum de livraison pour chaque véhicule, …). Un ensemble de clients demande à se faire livrer une certaine quantité de marchandise depuis un emplacement autre que le dépôt de véhicules.

La livraison se fait donc en deux temps. D'abord, le camion se rend en un lieu pour charger la cargaison, qui est ensuite livrée au destinataire final. Pour chaque lieu de chargement, une fenêtre de temps est définie. La livraison est effectuée par une flotte de camions hétérogènes. Il existe deux types de camions : les camions non-couverts qui n'ont pas de bâche et les camions couverts, qui en ont une. Il n'y a pas de problème de capacité, en effet, si c'est nécessaire, une commande peut être décomposée entre plusieurs véhicules. Un client peut donc être livré à partir de plusieurs véhicules. La résolution doit être effectuée quotidiennement, par rapport aux clients de la journée.

Tournées de cars scolaires : Bowerman et al. [Bow 95] s'intéressent à la planification de tournées de cars scolaires en milieu urbain et notamment dans le comté de Wellington dans

l'Ontario. Selon les auteurs, le problème de tournées de cars est plus complexe que le problème d'élaboration de tournées de véhicules classiques. Ils spécifient le problème de la manière suivante : un groupe d'élèves répartis sur une zone géographique doit recevoir un service de transport public depuis leur lieu de résidence jusqu'à l'école. Le problème est de trouver une série de tournées de cars scolaires qui assure que le service est équitable pour tous les élèves admissibles. Les auteurs proposent un modèle mathématique multiobjectif pour ce problème. Le modèle comporte quatre objectifs qui sont : la minimisation de la distance totale parcourue par les véhicules, la minimisation de la distance de marche des élèves, l'équilibrage de la charge et l'équilibrage des longueurs. Cependant, les auteurs proposent une approche en deux phases, qui ne prend pas en compte l'objectif d'équilibrage des longueurs. Dans la première phase, les élèves sont groupés en *clusters*, qui peuvent être servis par un car unique. Les objectifs sont la longueur des tournées, l'aspect compact des *clusters* (distance de marche) et l'équilibrage de la charge. La seconde phase calcule la tournée des cars sur chacune des grappes définies précédemment. Ce calcul sur une grappe est indépendant de celui effectué sur les autres grappes. Dans cette phase, il s'agit aussi de sélectionner les points d'arrêt du car, de telle sorte que la distance que les élèves doivent parcourir à pied soit minimisée.
Les deux objectifs envisagés dans cette deuxième phase sont : la minimisation de la longueur de la tournée sur chaque grappe et la minimisation de la distance que les élèves doivent parcourir à pied.
Les études de Corberan et al. [Cor 02] et de Pacheco et Marti [Pac 03] traitent le problème du transport d'élèves à leur école et leur retour ; le transport devant être effectué de la manière la plus sûre, la plus économique et la plus commode possible. D'autre part, l'environnement dans lequel le transport s'effectue est aussi particulier. Il s'agit en effet de zones rurales en Espagne, où les distances entre deux points de ramassage et jusqu'aux écoles tendent à être longues. Ainsi, le problème de remplissage des cars n'intervient pas ; car un autobus est rarement plein avant qu'une certaine limite de temps ne soit atteinte. Cette limite vient du fait qu'il ne faut pas qu'un élève passe trop de temps dans un car, et qu'il y ait trop d'iniquité entre le premier élève ramassé sur une tournée et le dernier.

Ramassage des déchets : dans l'étude de Lacomme et al. [Lac 03], des déchets doivent être collectés dans les rues de la ville de Troyes en France. Il s'agit donc d'un problème de tournées sur les arcs. Dans ce cas, tous les véhicules quittent le dépôt à six heures du matin et la collecte des déchets doit être terminée au plus vite car les travailleurs affectés aux véhicules doivent ensuite s'occuper du tri des déchets dans une usine de recyclage.

Transport de marchandises à Singapour : Tan et al. [Tan 03a] proposent un problème bi-objectif défini à partir des capacités des véhicules pour une entreprise de logistique de Singapour présentée dans [Lee 03]. Le modèle utilisé est un VRP avec camions et remorques (*VRPCR - Truck and Trailer Vehicle Routing Problem*) [Cha 02], où la longueur de la solution et la taille de la flotte de camions et de remorques sont prises en compte. Dans le VRPCR, les véhicules sont formés de deux éléments détachables. Certaines localisations sont accessibles par le véhicule complet. Pour d'autres, il est nécessaire de laisser la remorque le temps d'effectuer les livraisons. Dans l'étude de Tan et al. [Tan 03a], des fenêtres de temps sont ajoutées au problème.

Distribution de produits dangereux : Zografos et Androustsopoulos [Zog 04] proposent la modélisation de la distribution de produits dangereux sous la forme d'un problème bi-objectif, où la minimisation du coût ne se fait pas au détriment de la minimisation du risque et vice versa. Le problème est modélisé sous la forme d'un problème d'élaboration de tournées de

véhicules avec fenêtres de temps. L'heuristique proposée a été intégrée à un système d'information géographique pour des opérations logistiques sur les matériaux dangereux.

Un problème de tournées de véhicules multipériodiques : Dans [Mou 04], le problème étudié est un problème de tournées de véhicules multipériodiques où l'équilibrage de charges (sur les véhicules ou les journées), ainsi qu'un objectif de régionalisation visant à générer des *clusters* de clients, sont pris en compte. Lors de ce travail, un problème réel comprenant 6000 clients sur un délai de 20 jours a notamment été traité.

5.5.1.2 Extension du problème

Une autre utilisation de l'optimisation multiobjectif est la possibilité d'étudier des objectifs supplémentaires, sans pour autant abandonner la minimisation du coût de la solution qui est souvent nécessaire. L'ajout d'objectifs a le plus souvent pour but de renforcer le côté réaliste du modèle, en prenant en compte le fait que la plupart des problèmes en logistique sont soumis à des impératifs qui ne sont pas uniquement liés à la quantité de travail. Le VRPTW, présenté en 4.1, est une extension du VRP, où un objectif supplémentaire sert, par exemple, à chercher des solutions qui utilisent le moins de véhicules possible.

Prise en compte des conducteurs : Lee et Ueng [Lee 98] proposent une extension du VRP où un équilibrage entre les tournées est aussi recherché. L'ajout du second objectif est motivé par la volonté de rendre le travail plus équitable entre les conducteurs. D'après les auteurs, à Taiwan, l'affectation des tâches est effectuée manuellement par un décideur qui peut être influencé par des préférences personnelles, des informations insuffisantes ou d'autres facteurs humains. Les conducteurs, en comparant leurs emplois du temps, peuvent découvrir ainsi des anomalies, ce qui entraîne des plaintes et donc une baisse de la qualité du service. Or, les conducteurs jouent un rôle important dans le rendement de leurs compagnies de transport et c'est pourquoi leur bien-être doit être pris en compte.

Prise en compte de la satisfaction des clients : Sessomboon et al. [Ses 98] ont ajouté des objectifs au VRP, pour améliorer le niveau de satisfaction des clients par rapport à la date de livraison. Ribeiro et Lourenço [Rib 01] proposent une extension du VRP avec période dans lequel le coût, l'équilibre entre les tournées et un objectif défini d'un point de vue marketing sont pris en compte. La motivation pour l'objectif d'équilibrage est la même que dans l'étude de Lee et Ueng [Lee 98]. Le problème a été défini par rapport aux problèmes survenant dans l'industrie de la nourriture et de la boisson, où le planning de distribution est ordinairement défini pour une semaine, et où la relation entre le client et le conducteur a une grande importance pour améliorer les ventes et créer une bonne image de la compagnie.

Problème du voyageur de commerce multiobjectif : plusieurs études s'intéressent au problème du voyageur de commerce multiobjectif [Paq 03][Zhe 03]. Dans ce problème, plusieurs coûts sont associés aux arêtes. Par exemple, si le graphe représente un réseau routier, le temps nécessaire pour se rendre d'un point à l'autre peut ne pas être proportionnel à la distance parcourue. Une autre motivation pour l'étude de ce problème vient du fait qu'il s'agit d'une extension d'un problème classique fortement étudié.

5.5.1.3. Généralisation du problème

Une autre manière d'utiliser l'optimisation multiobjectif est de généraliser un problème par l'inclusion d'objectifs supplémentaires. Typiquement, il s'agit de modéliser une ou

plusieurs caractéristiques du problème, généralement des contraintes, et de les remplacer par des objectifs. Dans la littérature, cette stratégie a été plusieurs fois employée pour s'attaquer au problème d'élaboration de tournées de véhicules avec fenêtres de temps, où les fenêtres de temps, qui sont très contraignantes, sont supprimées et remplacées par un ou plusieurs objectifs [Hon 99][Rah 01][Bar 03][Rah 03a]. Pour le VRP avec fenêtres de temps, il peut aussi s'agir de traiter simultanément la minimisation de la longueur de la tournée et la minimisation du nombre de véhicules utilisés [Rah 01][Bar 03][Rah 03a][Tan 03a].

Boffey [Bof 95] cite une liste de problèmes de tournées qu'il qualifie de problèmes implicitement multiobjectifs. Dans ces problèmes, une contrainte est utilisée à la place de ce qui peut naturellement être modélisé par un objectif. Feillet et al. [Fei 05] exposent une classe de problèmes qu'ils appellent problèmes du voyageur de commerce avec profits.
Dans ces problèmes, un profit, associé à chaque client, peut être collecté en visitant le client. Il n'est pas obligatoire de visiter tous les clients. Deux objectifs opposés peuvent être clairement envisagés :
1. Maximiser le profit, et de ce fait visiter le maximum de clients, au prix d'une augmentation de la distance totale parcourue.
2. Minimiser la distance parcourue en visitant peu de clients, au prix d'une diminution du profit.

Lorsque l'objectif est de maximiser le profit sans que la longueur de la tournée dépasse une constante, on parle de problème du voyageur de commerce sélectif. Lorsque le but est de minimiser la longueur de la tournée tout en assurant un profit maximum, le problème est appelé problème du voyageur de commerce avec quota. Cependant, d'un point de vue bi-objectif, il ne s'agit que d'un seul et même problème. Les auteurs ne rapportent qu'une seule étude qui prend en compte le problème sous forme bi-objectif [Kel 88].

5.5.2. Méthodes de résolution

Les techniques employées pour résoudre les problèmes de tournées de véhicules sont multiples. Elles sont brièvement présentées ici, selon deux catégories. La première concerne l'approche de résolution, qui correspond à la manière dont la présence de multiples objectifs est prise en compte. La seconde concerne la méthode d'optimisation employée.

5.5.2.1 Prise en compte du caractère multiobjectif

La plupart des études utilisent des méthodes scalaires pour traiter la présence de plusieurs objectifs. Une majorité utilise la méthode d'agrégation des critères [Koe 89][Sut 90][Lee 98][Cor 02][Paq 03][Zag 04]. Dans [Bar 03], la méthode d'optimisation employée est la métaheuristique des colonies de fourmis et le choix entre les objectifs est pris en compte en utilisant plusieurs types de phéromone. D'autres méthodes scalaires, comme l'approche "but programmé" [Koe 89], l'approche ε-contrainte [Pac 03] sont aussi utilisées. Dans cette dernière étude, les auteurs essaient les différents nombres de véhicules possibles et, pour un nombre de véhicules fixé, cherchent à minimiser la durée de la plus longue tournée (*makespan*). Une approche lexicographique a aussi été proposée [Kel 88]. D'autres études emploient des approches Pareto [Ses 98][Rah 01][Rah 03a][Joz 04].

5.5.2.2 Méthodes d'optimisation

De nombreuses études utilisent des heuristiques spécifiques. Ces méthodes sont le plus souvent couplées avec des approches de résolution scalaires, et notamment des agrégations.

Corboran et al. [Cor 02] utilisent une méthode de recherche par dispersion. Dans [Pac 03], la méthode d'optimisation employée est une recherche tabou. El-Sherbeny [Els 01] utilise le recuit simulé multiobjectif proposé par Ulungu et al. [Ulu 99]. Les autres études emploient des algorithmes évolutionnaires. Dans le cas de Sessomboon et al. [Ses 98], de Lacomme et al. [Lac 03] et Tan et al. [Tan 03a][Tan 03b], l'algorithme génétique est hybridé avec une recherche locale.
Paquete et al. [Paq 03] proposent une méthode de recherche locale en deux phases. Dans un premier temps, le problème est résolu de manière monoobjectif, puis, lorsqu'un optimum local est atteint, les deux objectifs du problème sont combinés à l'aide d'une agrégation, et le problème ainsi obtenu est de nouveau résolu par une recherche locale.

5.6. Proposition d'une approche Pareto pour la résolution du VRPTW bi-objectif

5.6.1. Introduction

Nous nous proposons, dans cette partie, de concevoir un algorithme génétique pour le VRPTW dans sa version multiobjectif (distance parcourue, nombre de véhicules) en utilisant l'approche Pareto [Van 00][Zit 99][Coe 02]. Notre méthode intègre plusieurs enrichissements de l'algorithme génétique de base, dont :

- l'utilisation d'une heuristique pour la génération de la population initiale,
- l'implémentation et le test des opérateurs de croisement *RBX* et *SBX* [Pot 96a],
- l'implémentation et le test des opérateurs de mutation *swap* (*reinsert*) et *One Level Exchange* [Pot 96b],
- l'implémentation et le test des approches de sélection *NSGA* [Sri 95] et *ELITISME* [Lau 01][Deb 02],
- l'introduction de techniques de partage (*sharing*) pour la diversification [Ped 98],
- l'hybridation avec un algorithme de recherche locale.

L'apport de ce travail réside dans la diversité des approches de résolution proposées et implémentées. En effet, l'utilisation de plusieurs combinaisons de stratégies de sélection, d'opérateurs de recombinaison, de techniques de maintien de la diversité et enfin de méthodes d'hybridation permet de mieux cerner le problème, et donc de trouver des solutions de bonne qualité. En particulier, ces algorithmes ont permis d'améliorer la meilleure solution connue pour l'instance R203 de Solomon [Sol 88].

5.6.2. Application des algorithmes génétiques au VRPTW multiobjectif

L'algorithme génétique implémenté est résumé comme suit (figure 5.3) :

```
Algorithme : AG

Début
- Déterminer la population initiale; gen:=1;
- Tant que gen ≤nbgen   /*gen : génération courante*/
   faire
   - Evaluer les individus de la population;
   - Diversification : partage phénotypique:
        Début . Calculer les distances phénotypiques
              . Calculer f'(x) /*f'(x)=f(x)/m(x)*/;
```

```
        Fin
    - Sélection;
    - Copier chaque individu sélectionné dans une population
      intermédiaire;
    - Croisement;
    - Mutation;
    - Remplacer les mauvais individus par les meilleurs;
    - Passer à la génération suivante : gen:=gen+1;
  fait
Fin
```

Figure 5.3. Algorithme génétique pour la résolution du VRPTW bi-objectif.

Cet algorithme utilise différents éléments, que nous détaillons tout au long de cette section.

5.6.2.1. Représentation d'une solution

La représentation chromosomique choisie est celle d'une séquence de clients. Ce choix est motivé par la traduction naturelle d'une tournée (solution) et la simplicité de cette représentation. Avec cette représentation, les clients apparaissent selon leur ordre de visite. Chaque client ne peut y apparaître qu'une seule fois. Le dépôt peut, lui, apparaître plusieurs fois. Son emplacement marque la fin d'une tournée et le début d'une autre. Par exemple, la représentation de la solution de la figure 5.4 est la suivante: 0 1 2 3 0 4 5 0 8 7 6 0.

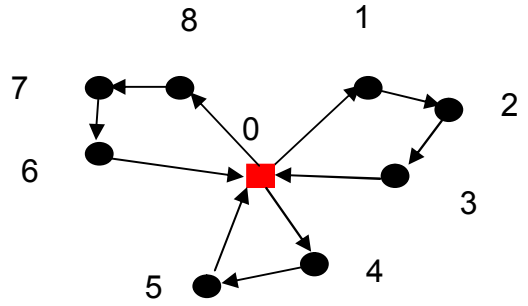


Figure 5.4. Exemple d'une tournée.

5.6.2.2. Génération de la population initiale

Dans notre approche, la population initiale est construite en deux phases : la première consiste, à l'aide d'une heuristique, à trouver une solution initiale réalisable. A partir de cette solution, d'autres sont construites, en utilisant le voisinage *swap* [Or 76] (de l'ordre de 40% de la population initiale totale). Dans la deuxième phase, la population ainsi constituée est complétée par d'autres solutions réalisables construites de façon aléatoire (de l'ordre de 60% de la population initiale totale). Celles-ci garantissent une bonne répartition des solutions dans l'espace de recherche.

L'heuristique développée est inspirée de l'heuristique *Push Forward Input Heuristic* (*PFIH*) introduite par Solomon [Sol 88]. L'heuristique *PFIH* prend un premier client par lequel un véhicule commence son trajet, puis insère les autres clients dans cette route en cours de construction, jusqu'à ce qu'il soit impossible d'y insérer d'autres clients. Dans notre

approche, le premier client à insérer est celui qui est le plus proche du dépôt en terme de distance euclidienne. Une fois le premier client inséré, l'heuristique choisit parmi le reste des clients, celui qui engendre un coût minimal (distance totale minimale) après son insertion. Ce processus de sélection est répété jusqu'à ce qu'il soit impossible d'insérer d'autres clients sur la route en cours. A ce stade, on met fin à la route en cours, et on crée une nouvelle route. Ce processus est itéré jusqu'à ce que tous les clients soient insérés dans la solution. Le nombre de routes obtenu détermine le nombre de véhicules utilisés.

L'amélioration des solutions obtenues suivant l'heuristique *PFIH* se fait en utilisant le voisinage *swap*. Ce dernier est un cas particulier du voisinage *Or-opt* [Or 76]. Il consiste à déplacer un client *c* d'une route *i* vers une route *j* d'une solution. La position d'insertion dans la route *j* est celle qui minimise la distance totale parcourue. La figure 5.5 montre un exemple d'un tel mouvement.

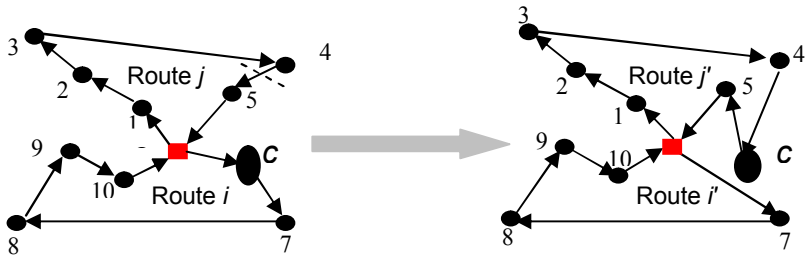


Figure 5.5. Le voisinage *swap*. Le client *c* a été supprimé de la route *i* et mis dans la route *j* à la position 5 on obtient alors les routes *i'* et *j'*.

La procédure de génération de la population initiale est donnée par la figure 5.6. La taille totale de la population est notée t_p et est fixée expérimentalement. Le nombre de solutions construites aléatoirement dépend du nombre de voisins de s_0, il est égal à t_p moins le nombre de voisins de s_0. Expérimentalement t_p varie entre 100 et 150 individus.

```
Algorithme : Gen_pop_initiale

Début
Soit S0 la solution initiale obtenue avec l'heuristique PFIH
modifiée.
Pour chaque route i dans S0 faire
    Pour chaque client k dans la route i faire
      - Chercher à travers toutes les routes j ≠ i la position qui
        minimise la distance parcourue;
      - Si une telle position existe, alors insérer l'individu k
        dans la route j selon cette position et ajouter cette
        solution à la population initiale.
     fait;
fait
/*Génération aléatoire du reste de la population*/
```

```
Etape 1 : Tant qu'il reste des clients non encore insérés, faire
-  Choisir un client de façon aléatoire.
-  Si son insertion n'engendre pas une violation de contraintes
   alors l'insérer dans la route en cours de construction
- S'il n'est plus possible d'insérer d'autres clients dans la
   route, mettre fin à la route en cours, aller à l'étape 1.
fait;
FIN.
```

Figure 5.6. Algorithme de génération de la population initiale.

5.6.2.3. Les opérateurs génétiques

5.6.2.3.1. Opérateurs de croisement retenus

Nous avons implémenté deux types de croisement : *SBX* et *RBX* [Pot 96a].
Le croisement *SBX* (*Sequence Based Crossover*) [Pot 96c], illustré par la figure 5.7, agit sur deux parents. Un lien (arc), choisi aléatoirement, est retiré de chacun des parents. Les clients qui sont servis, avant le point de rupture sur la route du premier parent, sont liés à ceux servis après le point de rupture sur la route du deuxième parent. La nouvelle route ainsi obtenue remplace l'ancienne dans le premier parent. Cet opérateur permet d'obtenir un seul individu fils. En inversant l'ordre des parents, on obtient un autre individu fils. Généralement, les clients ayant des temps de début de livraison les plus rapprochés, sont ordonnés au début d'une route et inversement pour les clients dont les temps de début sont les plus éloignés. On suppose donc que la solution construite est réalisable. Cependant, la nouvelle solution issue de ce croisement n'est pas toujours valide. En effet, des clients peuvent disparaître de la solution, alors que d'autres peuvent être dupliqués. Une réparation est alors appliquée sur l'individu fils. Lorsqu'un client apparaît deux fois dans la même route, l'une des deux copies est supprimée. Lorsqu'un client apparaît deux fois dans deux routes différentes, le client sera gardé dans la nouvelle route et supprimé de l'ancienne. On remarque que ces deux situations sont plutôt simples à résoudre, du fait qu'une solution réalisable (qui respecte les contraintes de temps et de capacité) restera réalisable après la suppression d'un client.
Lorsqu'un client disparaît de la solution, on cherche à l'insérer dans une position qui maintienne la faisabilité de la solution, d'un côté, et qui minimise le coût additionnel de son insertion, d'un autre côté. En réalité, les clients supprimés sont difficiles à manipuler, car rien ne garantit l'existence d'une telle position d'insertion ; c'est la raison pour laquelle souvent les solutions générées ne sont pas réalisables. Pour y remédier, nous appliquons le croisement sur la totalité de la population, à chaque itération de l'algorithme génétique.

Le deuxième opérateur utilisé est le croisement *RBX* (*Route Based Crossover*) [Pot 96b]. Cet opérateur remplace une route du deuxième parent par une route du premier parent. Comme dans le cas *SBX*, il faut réparer la nouvelle solution, en retirant les clients dupliqués et en insérant ceux qui ont disparu de la solution résultante. Ce croisement est illustré par la figure 5.8.

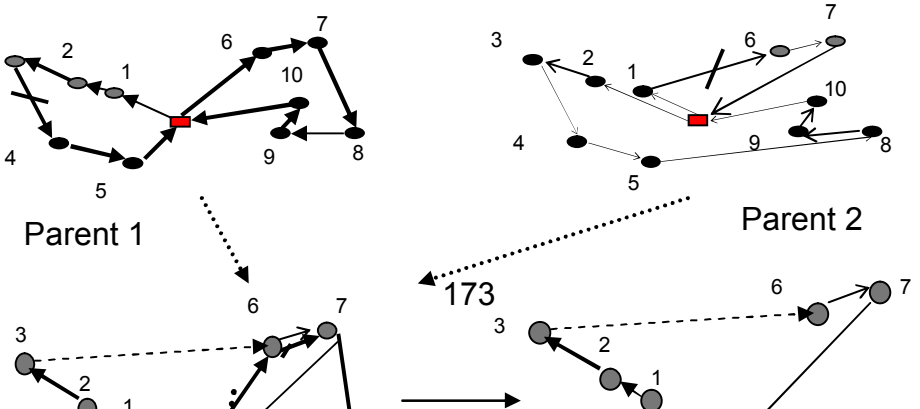


Figure 5.7. Croisement SBX avec réparation.

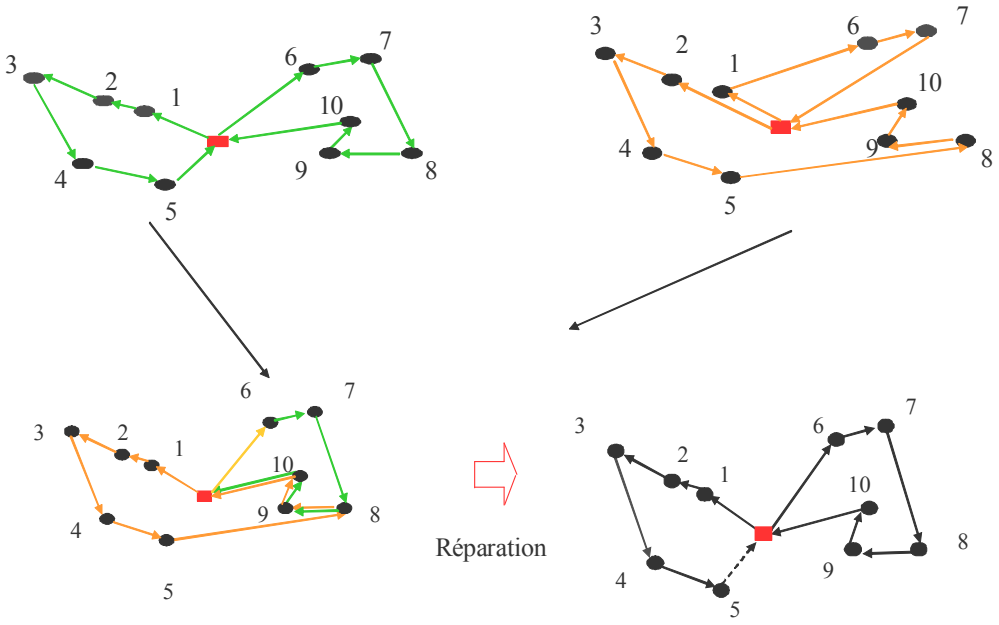


Figure 5.8. Croisement RBX avec réparation.

5.6.2.3.2. Opérateurs de mutation implémentés

Deux opérateurs de mutation ont été utilisés, l'opérateur *swap* (ou *reinsert*) [Or 76] et l'opérateur *One Level Exchange* proposé par [Pot 96a]. Ce dernier choisit une route de la solution, puis essaye de déplacer ses clients pour les insérer ailleurs. Cet opérateur a pour but de réduire le nombre de véhicules utilisés. La procédure correspondante est (figure 5.9) :

```
Algorithme : mutation_OLE
DEBUT
Pour chaque individu ind de la population Faire
- Générer aléatoirement un nombre p∈[0,1]
```

```
- Si (p<pm) alors /*pm probabilité de mutation*/
 . Choisir une route i de ind
 . Pour chaque client c dans la route i, faire
     - Trouver une place d'insertion réalisable pour le client i
       qui minimise le détour (à travers toutes les autres routes
       j≠i)
     - Si une telle position existe, alors insérer le client c à
       cette position
    fait ;
  fin si ;
fait ;
FIN.
```

Figure 5.9. Algorithme pour l'opérateur de mutation *One Level Exchange*.

5.6.2.3.3. Stratégies de sélection et de diversification retenues

Pour montrer les performances des deux méthodes de sélection implémentées, élitisme et NSGA (voir les sections 4.2.2.2 et 4.2.2.4, plusieurs tests ont été réalisés sur plusieurs types d'instances (R102, R203, RC105 et RC204). Ces tests ont permis aussi le réglage des paramètres : le nombre de générations (*nbgen*), la taille de la population (*tp*), la pression de sélection (*S*) et le nombre d'individus sélectionnés à partir de la population archive PO^+ (*A*). Notons que l'utilisation de l'élitisme nécessite de bien régler le paramètre *A*. En effet, une valeur élevée de "*A"* a pour effet d'intensifier l'exploitation des bonnes solutions trouvées par le processus de recherche. Alors qu'une "petite" valeur de *A* favorise l'exploration de nouvelles zones de l'espace de recherche. Ainsi, plusieurs essais ont été effectués afin de trouver la bonne valeur de ce paramètre pour chaque classe d'instances (environ dix tests pour chaque type de croisement utilisé, en modifiant la valeur de *A*, soit au total 30 tests par classe d'instances).

Dans l'algorithme proposé, le maintien de la diversité a été réalisé grâce au principe du *partage* (s*haring*) [Ped 98][Sri 95][Deb 02] vu dans la section 4.2.2.5. La dégradation du mérite *f* des individus appartenant à la même région de l'espace de recherche est réalisée grâce à une fonction de partage *sh*. Celle-ci définit le compteur de niche *m(x)* qui est utilisé pour le calcul du mérite *f'* d'un individu x :

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

$$m(x) = \sum_{y \in pop} sh\big(dist(x,y)\big)$$

La fonction de partage *sh* calcule le degré de similarité d'un individu par rapport au reste de la population. Elle est définie de la manière suivante :

$$sh\big(dist(x,y)\big) = \begin{cases} 1 - \left(\dfrac{dist(x,y)}{\sigma_{share}}\right)^{\alpha} & \text{si } \mathrm{dist(x,y)} < \sigma_{share} \\ 0 & \text{sinon} \end{cases}$$

avec σ_{share} la taille des niches (ou degré de dissimilarité), c'est-à-dire la distance à partir de laquelle deux individus *x* et *y* ne sont pas considérés comme appartenant à la même niche et α une constante permettant de réguler la forme de la fonction *sh*.

La stratégie de partage utilisée dans notre algorithme est dite *phénotypique*. En effet, la distance entre deux individus est calculée dans l'espace objectif. En notant f_1 le critère sur la distance totale parcourue et f_2 le critère sur le nombre de véhicules utilisés, la fonction distance s'écrit :

$$dist(x,y) = |f_1(x)-f_1(y)| + |f_2(x)-f_2(y)|$$

5.6.3. Hybridation de l'algorithme avec la recherche locale

La recherche locale a été utilisée pour améliorer la qualité de la solution. Deux approches ont été utilisées. La première consiste à lancer l'algorithme génétique en premier, afin d'avoir une première approximation de la frontière de Pareto. La seconde approche consiste à lancer la recherche locale à des intervalles de temps réguliers en terme de générations.
Le principe de notre recherche locale consiste à construire pour chaque individu de la population Pareto, l'ensemble de son voisinage (figure 5.10). Les voisins non-dominés dans la population Pareto sont insérés dans celle-ci. Les solutions appartenant à l'ensemble Pareto et dominées par un élément de l'ensemble des voisins sont supprimées de l'archive PO^+. Ce procédé est itéré jusqu'à ce qu'aucun voisin d'aucune solution Pareto ne soit inséré dans la population Pareto. Pour la génération du voisinage d'une solution, la recherche locale développée utilise l'opérateur 2-opt*, de Potvin et al. [Pot 96a]. Il consiste à choisir deux routes et un point de coupure pour chacune. Les clients qui sont servis avant le point de coupure de la première route sont concaténés aux clients servis après le point de coupure de la deuxième route, et inversement (figure 5.11).

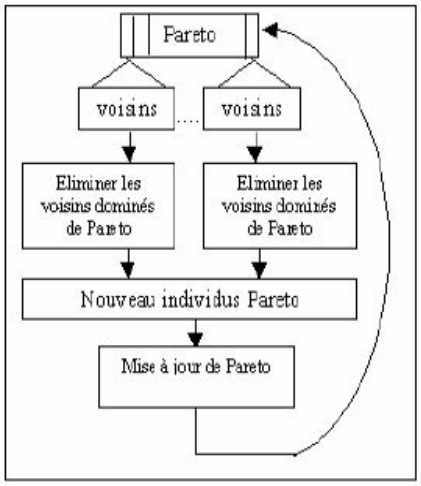
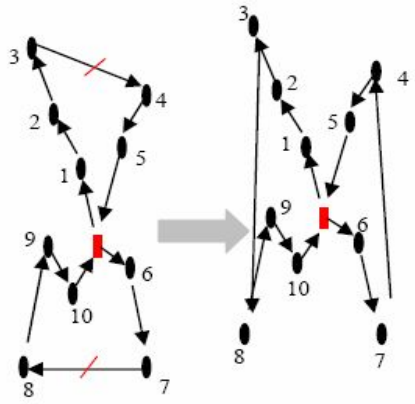


Figure 5.10. Fonctionnement de la Recherche Locale



Les arcs (3,4) et (7,8) sont supprimés de la solution (a). Après l'application de l'opérateur 2-opt*, on obtient la solution (b) tel que : le sommet 3 est lié au sommet 8 et le sommet 7 est lié au sommet 4 créant ainsi les arcs (3,8) et (7,4).

Figure 5.11. Le voisinage 2-opt*

La procédure de recherche locale décrite précédemment est résumée comme suit (figure 5.12):

```
Algorithme : RL
  DEBUT
   Changement = vrai ;
   Tant que (changement == vrai)
   faire
```

```
      - Voisins = Ø ;
      - Changement =faux ;
      - Pour tout individu ind de la population archive PO+ faire
          Pour chaque route i de ind faire
             Pour chaque position coupa dans i faire
                Pour chaque route j ≠ i dans ind faire
                   Pour chaque position coupb dans j faire
                   Début
                        - Appliquer l'opérateur 2-opt* à l'individu
                          ind en considérant les deux points de
                          coupure comme étant coupa et coupb.
                          Appeler l'individu résultant par ind'.
                        - Evaluer ind' (nbre de véhicules, distance,
                          nbre de contraintes violées)
                        - Mettre à jour l'ensemble voisins (si ind'
                          est dominé par l'un des membres de
                          voisins, ne pas l'insérer, s'il existe des
                          membres dominés par ind' alors les
                          supprimer de voisins)
                   Fin ; Fait ; Fait ; Fait ; Fait ; Fait ;
      - mettre à jour PO+ à partir de l'ensemble voisins
      - Si l'ensemble PO+ est modifié alors changement=vrai ;
  fait
FIN.
```

Figure 5.12. Algorithme de recherche locale proposé.

5.6.4. Tests et résultats

Les algorithmes proposés ont été écrits en C/LINUX et testés sur un PC de type Pentium 3 (1 GHz et 128Mo de RAM). Les instances utilisées lors de ces tests sont les 56 instances euclidiennes à 100 clients et un dépôt créées par Solomon [Sol 04] (voir le tableau 5.1). Près de 150 exécutions par classe d'instances ont été réalisées. Ces tests ont permis le réglage empirique des différents paramètres de nos algorithmes (tableau 5.3). De plus, près de 14 combinaisons des différents opérateurs retenus ont été testées. Cette étude comparative a permis d'établir un classement des meilleurs opérateurs par classe d'instances (tableau 5.4).

5.6.4.1. Les benchmarks de la littérature

Les 56 instances de Solomon (voir le tableau 5.1) sont une généralisation pour le VRPTW des instances de 100 clients créées par Christophides, Mingozzi en 1979 pour le VRP. La conception de ces problèmes tests met en valeur plusieurs facteurs pouvant influer sur le comportement des heuristiques de routage : la répartition géographique des clients, le nombre de clients pouvant être servis par le même véhicule et toutes les caractéristiques inhérentes aux fenêtres horaires.

Six classes d'instances ont ainsi été définies : R1, RC1, C1, R2, RC2, C2. La distinction par les lettres R, RC et C est relative à la forme de *répartition géographique* des clients. Les problèmes de classe R1 et R2 prennent en compte des clients uniformément répartis. Les clients des classes C1 et C2 sont groupés en *"cluster"* : des sous-ensembles de clients distincts, proches les uns des autres. Enfin les problèmes de type RC consistent en un mélange de clients uniformément répartis, et de *clusters*.

La distinction entre un problème de type 1 et un problème de type 2 concerne ce que l'on appelle *l'horizon du routage*. Les instances dans les classes R1, RC1 et C1 ont un *horizon court* : c'est-à-dire que les demandes des clients et les capacités des véhicules sont définies de telle manière qu'un véhicule ne peut accepter que très peu de clients sur une tournée, ces problèmes requièrent généralement entre 9 à 19 véhicules. A l'inverse, certaines instances dans les classes R2, RC2 et C2 ont des solutions où une tournée passe par la moitié des clients, 2 à 4 véhicules suffisent dans ce cas. On parle alors d'*horizon lointain*.

Pour chaque famille de problèmes, plusieurs variations existent quant à la forme des fenêtres horaires. D'une instance à une autre, Solomon a fait varier plusieurs facteurs, tels que le pourcentage de clients soumis à des contraintes horaires (25%, 50%, 75%, 100%), la position dans le temps, ainsi que l'étroitesse des fenêtres. Au total, 56 problèmes de 100 clients ont été générés :

Classe	Instances	Description
R1	R101 à R112	Les clients sont uniformément répartis avec un horizon court
C1	C101 à C109	Les clients sont "clustérisés" avec un horizon court
RC1	RC101 à RC108	Un mélange de clients R et C avec un horizon court
R2	R201 à R211	Les clients sont uniformément répartis avec un horizon long
C2	C201 à C208	Les clients sont "clustérisés" avec un horizon long
RC2	RC201 à RC208	Un mélange de clients R et C avec un horizon long

Tableau 5.1. Les benchmarks de Solomon [Sol 88].

Depuis plus d'une dizaine d'années, la plupart des heuristiques nouvelles ont été testées et comparées par rapport à ces instances, devenues, en quelque sorte, une échelle de référence et un laboratoire pour le VRPTW.

5.6.4.2. Réglage des paramètres des algorithmes développés

La détermination des paramètres de nos algorithmes est une étape importante pour juger de la qualité des résultats. A cet effet, près de 150 exécutions ont été effectuées. Pour déterminer la valeur d'un paramètre, on fixe celle des autres paramètres et on fait varier celle-ci. La valeur retenue est celle qui donne la meilleure solution. Plusieurs exécutions sont réalisées sur plusieurs benchmarks. Les meilleures valeurs trouvées sont consignées dans le tableau 5.2.

Classe Insts.	t_p	*Nbgen*	P_c	P_m	*A*	*S*	α	σ_{share}
R_1	130	2000	1	0.5	5	1.8	1	10
R_2	150	1500	1	0.4	7	1.8	1	10
RC_1	100	1500	1	0.1	5	1.8	1	15

RC_2	100	1500	1	0.1	5	1.8	1	10
C_1	100	1500	1	0.1	5	1.8	1	10
C_2	100	1500	1	0.1	5	1.8	1	10

Tableau 5.2. Les paramètres des algorithmes implémentés par classe de problèmes.
Où t_p : taille de la population ; *Nbgen* : nombre limite de générations ; P_c : probabilité de croisement ; P_m : probabilité de mutation ; A : nombre d'individus sélectionnés à partir de PO^+ ; S : pression de sélection ; α : constante de régulation de la forme de la fonction de partage (*sharing*) sh ; σ_{share} : taille de niche (partage phénotypique) appelé aussi degré de dissimilarité.

5.6.4.3. Classement des opérateurs par classe d'instances

Près de 14 combinaisons des différents opérateurs retenus ont été testées. Cette étude comparative a permis d'établir un classement des meilleurs opérateurs par classe d'instances (tableau 5.3) :

Classes Instances	Croisement			Mutation		Sélection		Partage
	RBX	SBX	RBX+ SBX	*Swap*	*One Level Exchange*	Elitisme	NSGA	Phéno.
R_1			x		x	x		x
R_2			x		x	x		x
RC_1		x			x		x	x
RC_2	x			x		x		x
C_1		x					x	
C_2	x			x		x		x

Tableau 5.3. Classement des meilleurs opérateurs par classe d'instances. Le symbole "x" indique la meilleure stratégie par classe d'instances.

5.6.4.4. L'apport de la recherche locale

Les résultats des tests effectués sur les instances de Solomon permettent de remarquer que l'introduction de la recherche locale améliore les résultats obtenus avec l'algorithme génétique. La recherche locale périodique (dans ce cas, une période de 30 itérations) donne de meilleurs résultats que l'approche où la recherche locale est appliquée une fois. Cependant, dans certains cas, l'introduction de la recherche locale n'améliore pas la qualité des solutions. Nous avons constaté aussi que souvent l'amélioration des solutions se fait sur la distance parcourue.

5.6.4.5. Qualité des solutions extrêmes obtenues

Nous avons comparé nos résultats aux meilleures solutions extrêmes trouvées dans la littérature. Devant l'absence de benchmarks bi-objectifs, nous nous sommes restreints à comparer nos résultats à ceux obtenus dans le cas monoobjectif. Les résultats trouvés (tableaux 5.4 et 5.5) sont encourageants. Ils sont très proches des meilleures solutions publiées dans la littérature. Les valeurs moyennes calculées (tableau 5.5) et comparées aux

meilleures de la littérature permettent de constater que nous obtenons un surplus de 6 % de véhicules et qu'en terme de distance moyenne, nos résultats sont meilleurs de 4 %. Il est important de signaler que notre algorithme nous a permis de trouver une nouvelle solution pour l'instance R203 [Sol 88]. Le nombre de véhicules est égal à 3 et la distance est de 939,15, alors qu'elle était à 939,54. La solution ainsi trouvée est donnée par :

Route1 : 27-52-18-45-46-36-64-11-62-88-31-30-76-3-79-78-9-20-66-71-35-34-29-68-12-26-13-58.

Route2 : 50-33-81-65-51-1-69-39-67-23-72-73-21-40-53-87-2-41-22-75-56-74-4-55-25-54-24-80-77-28.

Route3 : 94-95-97-92-98-37-42-57-15-43-14-44-38-86-16-85-59-99-96-6-84-83-8-48-47-49-19-63-90-32-10-70-7-82-17-61-91-100-93-5-60-89.

Le tableau 5.5 donne, pour chaque classe d'instances, en première position, le nombre moyen de véhicules utilisés et, en deuxième position, la distance totale moyenne parcourue par les véhicules. Il est important de signaler que ces résultats sont obtenus par des méthodes approchées et que, parmi elles et pour certaines instances (exemple C1), des solutions optimales sont atteintes.

En comparant les résultats obtenus avec les différentes implémentations des algorithmes génétiques et évolutionnaires, on peut conclure que les algorithmes génétiques sont performants pour le VRPTW, ainsi que les algorithmes à base de colonies de fourmis [Gam 99]. Parmi toutes les méthodes utilisant ce type de métaheuristique pour la résolution de ce problème, l'algorithme génétique de Berger et al. [Ber 01] donne la plus petite valeur moyenne du nombre de véhicules utilisés. Quant au critère de la distance totale parcourue, la plus petite valeur moyenne est trouvée par Gehring et al. [Hom 99], avec un algorithme hybride (ES1/ES2).

Quant à l'utilisation de la recherche tabou pour la résolution du VRPTW, les plus petites valeurs moyennes en terme de distance totale et nombre moyen de véhicules sont obtenues par Cordeau [Cor 01]. De même l'algorithme de recherche tabou de Roachat et al. [Roc 95] arrive à minimiser la distance totale parcourue de façon remarquable. De manière générale, les algorithmes génétiques semblent être plus performants que la recherche tabou. Cependant la combinaison des deux donne de bons résultats [Geh 01].

D'un point de vue global, les méthodes ayant donné de bons résultats, sont celles présentées par [Hom 99][Ber 01] et [Cor 01]. La différence entre ces méthodes, en terme de qualité de solution, est négligeable, seulement 0.5% pour le nombre moyen de véhicules et 1% pour la distance totale moyenne.

Par rapport aux différentes classes d'instances, on remarque que toutes les méthodes présentées arrivent pratiquement à des résultats satisfaisants. Elles donnent de bons résultats sur les classes C1 et C2, où les clients sont localisés dans des grappes. La classe RC2 semble être la plus difficile à résoudre, en ce qui concerne la distance totale parcourue, par exemple la différence entre [Tha 95a] et [Geh 01] est de 25%. De manière générale, les instances de type 2 sont plus compliquées à résoudre que celles de type 1, de par leur nature.

Inst.	Meilleurs résultats connus dans le cas monoobjectif*		Nos résultats		
	Nombre véhicules	Distance totale	Taille front de Pareto	Nombre véhicules	Distance totale
R101	19	1645,79	20	**19**	1656,04
R102	17	1486,12	11	**17**	1496,30
R103	13	1292,68	31	**13**	1296,07
R104	9	1007,24	20	10	1009,07
R105	14	1377,11	17	**14**	1378,45
R106	12	1251,98	12	**12**	1438,32
R107	10	1104,66	31	11	1250,56
R108	9	960,88	21	**9**	1114,33
R109	11	1194,73	12	**11**	1274,76
R110	10	1118,59	22	**10**	1270,28
R111	10	1096,72	23	**10**	1215,59
R112	9	982,14	22	**9**	1027,20
R201	4	1252,37	11	**4**	1307,35
R202	3	1191,70	15	4	1141,84
R203	3	939,54	**21**	**3**	**939,15**
R204	2	825,52	22	3	**762,96**
R205	3	994,42	31	**3**	998,60
R206	3	906,14	30	**3**	941,39
R207	2	893,33	29	**2**	898,76
R208	2	726,75	18	**2**	826,14
R209	3	909,16	21	**3**	1009,87
R210	3	939,34	21	**3**	1020,82
R211	2	892,71	23	3	**808,78**
RC201	4	1406,91	23	**4**	1496,41
RC202	3	1367,09	22	4	1338,53
RC203	3	1049,62	21	**3**	1126,44
RC204	3	798,41	19	**3**	859,65
RC205	4	1297,19	20	**4**	1586,22
RC206	3	1146,32	17	**3**	1245,10
RC207	3	1061,14	11	**3**	1173,31
RC208	3	828,14	13	**3**	965,69
RC101	14	1696,94	14	15	**1342,77**
RC102	12	1554,75	13	14	**1226,88**
RC103	11	1261,67	12	12	**1069,80**
RC104	10	1135,48	12	11	**1015,66**
RC105	13	1629,44	15	14	**1239,28**
RC106	11	1424,73	21	13	**1135,78**
RC107	11	1230,48	22	12	**1032,72**
RC108	10	1139,82	21	**10**	1141,36
C101	10	828,94	22	**10**	828,94
C102	10	828,94	12	**10**	828,94
C103	10	828,06	23	**10**	828,06
C104	10	824,78	31	**10**	824,09
C105	10	828,94	29	**10**	828,94
C106	10	828,94	30	**10**	828,94
C107	10	828,94	25	**10**	828,94
C108	10	828,94	23	**10**	828,94
C109	10	828,94	17	**10**	830,00

C201	3	591,56	12	**3**	591,55
C202	3	591,56	11	**3**	699,95
C203	3	591,17	14	**3**	594,74
C204	3	590,60	15	**3**	765,79
C205	3	588,88	22	**3**	680,84
C206	3	588,49	23	**3**	717,96
C207	3	588,29	31	**3**	623,86
C208	3	588,32	23	**3**	623,86

Tableau 5.4. Comparaison des résultats obtenus aux meilleurs publiés. Le symbole "*" désigne les meilleurs résultats connus et obtenus avec des heuristiques dans le cas monoobjectif [Sol 88]. Les temps CPU sont de l'ordre de 1h.

Nous remarquons que les meilleurs résultats de la littérature (tableau 5.5) ne sont pas produits par un seul algorithme mais par des algorithmes hybrides. Il n'y a pas d'algorithme universel pour la résolution du VRPTW. Gambardella donne dans [Gam 99] une idée des temps de calcul nécessaires pour la résolution du VRPTW par des algorithmes approchés. Les temps d'exécution ne peuvent pas être directement comparés, pour différentes raisons. D'abord, les auteurs ont utilisé différents ordinateurs; en second lieu, quelques méthodes (TB par exemple) ont été conçues pour résoudre d'autres problèmes (plus difficiles que le VRPTW) alors que des adaptations spécifiques pour le VRPTW pourraient être plus rapides. Ainsi, il compare six algorithmes de résolution :

- MACS-VRPTW [Gam 99]: avec colonies de fourmis sur UltraSparc 1 167MHz, 70 Mflop/s (exécutions arrêtées après 100, 300, 600, 1200 et 1800 secondes); (le temps CPU est de l'ordre de 30 minutes approximativement).

- RT : Rochat et Taillard [Roc 95] avec de la recherche tabou sur Silicon Graphics computer 15 Mflop/s (le temps CPU est de l'ordre de 2 h approximativement);

- SW : Shaw et al. [Gam 99] exploration de grand voisinage sur Sun UltraSparc 63 Mflop/s; le temps CPU est de l'ordre de 1 h approximativement).

- KPS : Kilbey et al. [Gaml 99] avec recherche locale guidée sur DEC Alpha 25Mflops/s; le temps CPU est de l'ordre de 50 minutes approximativement).

- CW : Cordone et Wolfer-Calvo [Gam 99] avec k-Exchange réduction sur Pentium 18 Mflop/s; le temps CPU est de l'ordre de 22 minutes approximativement).

- TB : Taillard et al. [Tai 97] avec recherche tabou 10 Mflop/s Sun Sparc 10; le temps CPU est de l'ordre de 4 h approximativement).

Classes d'instances	R1	R2	C1	C2	RC1	RC2
Auteurs (ordre chronologique)	NV/DT	NV/DT	NV/DT	NV/DT	NV/DT	NV/DT

Nos résultats	12,08 1285,58	3.00 968,69	**10,00** 828,41	**3,00** 591,16	18.00 1150,53	3,37 1223.91
Thangiah et al. [Tha 94] (AG)	12.33 1227.42	3.00 1005.00	10.00 830.89	3.00 640.86	12.00 1391.13	3.38 1173.38
Rochat et al. [Roc 95] (RT)	12.25 1208.50	2.91 961.72	10.00 828.38	3.00 589.86	11.88 1377.39	3.38 1119.59
Thangiah [Tha 95a] (AG)	12.75 1300.25	3.18 1124.28	10.00 892.11	3.00 749.13	12.50 1474.13	3.38 1411.13
Potvin et Bengio [Pot 96a] (AG)	12.58 1296.83	3.00 1117.64	10.00 838.11	3.00 590.00	12.13 1446.25	3.38 1368.13
Taillard et al. [Tai 97] (RT)	12.17 1209.35	2.82 980.27	10.00 828.38	3.00 589.86	11.50 1389.22	3.38 1117.44
Chiang et al. [Chi 97] (RT)	12.17 1204.19	2.73 986.32	10.00 828.38	3.00 591.42	11.88 1397.44	3.25 1229.54
Berger et al. [Ber 98b] (AG)	12.58 1261.58	3.09 1030.01	10.00 834.61	3.00 594.25	12.13 1441.35	3.50 1284.25
Bräysy [Brä 99b] (AG)	12.58 1272.34	3.09 1053.65	10.00 857.64	3.00 624.31	12.13 1417.05	3.38 1256.80
Homberger et al. [Hom 99] (ES1/ES2)	11.92 1228.06	2.73 969.95	10.00 828.38	3.00 589.86	11.63 1392.57	3.25 1144.43
Gambardella [Gam 99] (Colonies de fourmis)	12.00 1217.73	2.73 967.75	10.00 828.38	3.00 589.86	11.63 1382.42	3.25 1129.19
Bräysy et al. [Brä 00] (AG+AE)	12.42 1213.86	3.09 978.00	10.00 828.75	3.00 591.81	12.13 1372.20	3.38 1170.23
Berger et al. [Ber 01] (AG)	11.92 1221.10	2.73 975.43	10.00 828.48	3.00 589.93	11.50 1389.89	3.25 1159.37
Tan et al. [Tan 01] (AG)	13.17 1227	5.00 980	10.11 861	3.25 619	13.50 1427	5.00 1123
Wee Kit et al.[Wee 01] (AG+RT)	12.58 1203.32	3.18 951.17	10.00 833.32	3.00 593.00	12.75 1382.06	3.75 1132.79
Cordeau et al. [Cor 01] (RT)	12.08 1210.14	2.73 969.57	10.00 828.38	3.00 589.86	11.50 1389.78	3.25 1134.52
Gehring et al. [Geh 01] (ES1+RT)	12.00 1217.57	2.73 961.29	10.00 828.63	3.00 590.33	11.50 1395.13	3.25 1139.37

Tableau 5.5. Valeurs moyennes par classes d'instances obtenues en monoobjectif pour les critères nombre de véhicules et distance totale parcourue.
Avec AG : algorithme génétique, RT : recherche tabou, AE : algorithme évolutionnaire, ES : stratégie évolutionnaire et NV/DT : nombre de véhicules moyen/distance totale moyenne.

5.7. Complexité de VRPTW

5.7.1. Motivations et complexité

Concernant la complexité de VRPTW, deux questions se posent :

- Est-il NP-complet de trouver une solution réalisable de VRPTW ?
- étant donnée une solution réalisable, est-il NP-complet de trouver une bonne approximation de l'optimum d'une instance de VRPTW ?

Il n'est pas difficile de voir que la réponse à la première question est positive. Il est en effet NP-complet de trouver une solution réalisable d'un problème de VRPTW.
Dans cette section, nous allons surtout nous occuper de la complexité de VRPTW au regard de l'approximation, tel que reflété par la deuxième question. Pour cela, nous allons d'abord poser VRPTW comme un problème monoobjectif avant de montrer comment le problème ainsi réduit se situe par rapport à l'approximation. Remarquons que le front Pareto comme

défini précédemment peut être muni d'une relation d'équivalence selon laquelle deux solutions sont considérées comme équivalentes dès lors que leurs fonction objectif ont les mêmes valeurs. Ainsi, il est inutile de calculer tout le front Pareto. Un ensemble de solutions représentatif de toutes les classes d'équivalence suffit, puisque seules les valeurs des objectifs nous intéressent plus particulièrement, et c'est ce que nous entendrons désormais par calcul d'un front Pareto. Pour voir comment VRPTW peut être réduit à un problème monoobjectif que nous noterons VRPTW(k), il suffit de fixer un entier k et de chercher, pour une instance donnée I de VRPTW, une plus courte tournée de I que l'on peut faire avec k véhicules ou moins en respectant les contraintes de fenêtres de temps. En faisant varier k de 1 à N, on sera en mesure de trouver le front Pareto de notre instance. La réciproque est évidente, puisqu'un front Pareto donne automatiquement une solutions optimale de VRPTW(k) pour tout k. Pour le cas particulier où k=1, VRPTW(1) contient le problème bien connu du voyageur de commerce (TSP) [Ang 02]. Or, il est bien connu que le TSP n'admet pas de rapport d'approximation constant en temps polynomial à moins que N=NP et on conclut donc que VRPTW(k) n'en admet pas non plus. Cependant, il est aussi bien connu que le TSP euclidien admet un rapport d'approximation égal à 3/2, et il se trouve que le cas euclidien a un intérêt particulier pour nous, puisque toutes nos instances étaient euclidiennes. La possibilité existe donc que le VRPTW(k) euclidien admette un rapport d'approximation constant. Le résultat principal de cette section est donné par le théorème suivant :

Théorème 5.1 : *Pour tout $\varepsilon \succ 0$ et pour un véhicule de capacité infinie, il n'existe pas de rapport d'approximation $r \leq N^{1-\varepsilon}$ pour le VRPTW(1) euclidien, où N est le nombre de clients.*

Preuve : La réduction se fait à partir du problème d'existence d'un chemin hamiltonien commençant à partir d'un sommet x_0 dans un graphe G. Le problème est évidemment NP-complet. Soit donc (G,x_0) une instance d'un tel problème, où G est d'ordre n. Pour tout entier $p \geq 4$, nous allons construire une instance I_p du VRTW(1) euclidien, telle que si G admet un chemin hamiltonien commençant en x_0 alors la valeur de la solution optimale de I_p ne dépasse pas $b(I_p)$, et réciproquement, si G n'admet pas de chemin hamiltonien commençant en x_0, alors la valeur de la solution optimale I_p dépassera nécessairement $\frac{p-1}{2}.b(I_p)$. À partir de là, nous conclurons que le VRPTW(1) euclidien ne peut avoir de rapport d'approximation inférieur à $(p-1)/2$. Le paramètre p peut être vu comme un amplificateur dans notre réduction. Pour un entier $p \geq 4$ quelconque, considérons donc un ensemble I_p de $(2p+1)n$ sites clients étiquetés (x,i), pour $x \in V(G)$ et $0 \leq i \leq 2p$. Le site $(x_0,0)$ désigne le dépôt. Les sites (x,i) ayant le même i sont dits appartenir à la classe i. Pour toute paire d'entiers $0 \leq i,j \leq 2p$, posons :

$$d_C(i,j) = \begin{cases} |j-i| & si\ |j-i| \leq p \\ 2p+1-|j-i| & \text{sinon} \end{cases}$$

(en d'autres termes, $d_C(i,j)$ est la distance de i à j au sens de la théorie des graphes sur le cycle induit imaginaire C=0,1,2,…2p+1,0).
Définissons la distance d'une paire de sites (x,i) et (y,j) comme :

$$c_{(x,i),(y,j)} = \begin{cases} Md_C(i,j) & si \quad d_C(i,j) \le p-1 & eti \ne j \\ M(p-1) & si \quad d_C(i,j) \quad p & eti \ne j \\ 1 & si \quad i = j & et(x,y) \in E(G)\text{'} \\ 2 & si \quad i = j & et(x,y) \notin E(G) \end{cases}$$

où $M \ge n$. Le temps de parcours entre deux sites (x,i) et (y,j) est définie de même comme : $t_{(x,i),(y,j)} = c_{(x,i),(y,j)}$.

De plus, comme p et $2p+1$ sont relativement premiers entre eux, la suite d'entiers (modulo $(2p+1)$) : $2p$, $3p$, $4p$,…, $(2p+1)p$, $(2p+2)p$, est en bijection avec l'ensemble 0,1,…,$2p$.

Maintenant pour tout site (x,i) avec, disons $i = kp \bmod(2n+1)$ pour un certain entier $2 \le k \le 2p+2$, définissons la borne inférieure de la fenêtre de temps comme $\alpha((x,i)) = i(M+n-1)$. De plus, pour tout site (x,i) avec, disons $i = kp \bmod(2n+1)$ pour un certain $2 \le k \le 2p$, définissons la borne supérieure de la fenêtre de temps comme :

$$\beta((x,i)) = \begin{cases} 2p(M+n-1) + (k-2)\left[M(p-1) + 2(n-1)\right] & six \ne x_0 \\ i(M+n-1) & six = x_0 \end{cases}$$

Les autres sites (x,i) avec $i = 2p \bmod(2p+1)$ où *i= 0(2p+1)* ont des bornes supérieures infinies. Posons $b(I_p) = (2p+1)(M+n-1)$.

Nous affirmons que si *G* possède un chemin hamiltonien commençant en x_0 alors I_p possède une tournée de longueur $b(I_p)$, sinon I_p ne contient pas de tournée de longueur inférieure à $\frac{p-1}{2}.b(I_p)$.

Remarquons d'abord que les bornes des fenêtres de temps sont serrées pour les clients (x_0,i), puisque $\alpha(x_0,i) = \beta(x_0,i)$. Ces temps sont séparés par des intervalles T_i, $1 \le i \le 2p$ de longueur $M+n-1$ chacun, durant lesquels le véhicule a juste le temps de visiter un certain nombre de sites de la même classe (disons la classe *i*-1) en temps *n*-1 et de passer à (x_0,i). Le même schéma se répète, le véhicule visitant un certain nombre de sites de la classe *i* avant de passer à $(x_0,i+1)$ etc) jusqu'à ce que tous les sites (x_0,i) soient visités, ce qui arrive au temps *2p(M+n-1)* exactement.

Maintenant, on distingue deux cas, selon que *G* possède ou ne possède pas de chemin hamiltonien commençant en x_0. Dans le premier cas, suivant le schéma de parcours qui vient d'être décrit, le véhicule a la possibilité de visiter tous les sites de la même classe *i* en temps *n*-1 (en utilisant le chemin hamiltonien de *G*) avant de passer à la classe *i+1* à chaque étape. En prenant les classes *i* modulo *(2p+1)*, on obtient bien une tournée de longueur $(2p+1)(M+n-1)$.

D'autre part, si *G* n'a pas de chemin hamiltonien, alors pour toute classe de sites, le véhicule ne peut visiter tous les sites de la classe *i*-1 et passer à (x_0,i) durant l'intervalle de temps T_{i-1} alloué à cet effet. Ainsi, au temps *2p(M+n-1)*, lorsque tous les sites (x_0,i) auront été visités, il restera encore des sommets à visiter dans chaque classe *i*. Or, les bornes supérieures des

fenêtres de temps montrent que le véhicule doit visiter ensuite la classe *3p(mod(2p+1))* immédiatement après le temps *2p(M+n-1)* (puisque le plus court chemin d'un site de la classe *2p* à un autre site de la classe 3*p* est de longueur *M(p-1)* et ne passe aucune autre classe). Les mêmes contraintes de bornes supérieures imposées sur les temps de visite des sites (x,i), où $i = kp$, $2 \le k \le 2p$, forcent le véhicule à visiter ensuite les sites de la classe *4p(mod(2p+1))*, puis les sites de la classe *5p(mod(2p+1))*…, etc, jusqu'à la classe *2p.p*(*mod(2p+1)*…Par conséquent, la longueur d'une telle tournée doit être supérieure ou égale à $2p(M+n-1)+M(p-1)(2p-2) \ge \frac{p-1}{2} b(I_p)$, comme on l'a affirmé. D'où, il ne peut exister de rapport d'approximation inférieur à $\frac{p-1}{2}$, comme indiqué. Mais il y a plus. Le nombre de sites de notre instance est $N = (2p+1)n$. En posant $2(p+1) = n^k$ pour un entier *k* arbitraire et en remplaçant dans *N*, on obtient: $\frac{p-1}{2} = \frac{N^{1-\frac{1}{k+1}} - 3}{4}$ et on conclut que VRPTW(1) ne possède pas de rapport d'approximation inférieur à $\frac{N^{1-\frac{1}{k+1}} - 3}{4}$ pour tout entier *k*, ce qui est équivalent à l'énoncé du théorème.

Remarquons que l'instance construite dans la démonstration de notre théorème admet toujours une solution réalisable : il suffit pour le véhicule de visiter les sites (x_0,i) dans l'ordre croissant de leurs bornes inférieures puis de visiter tous les autres sites dans l'ordre croissant de leurs bornes supérieures en résolvant les conflits de manière arbitraire. Ceci montre clairement que ce n'est pas seulement l'"infaisabilité" du problème qui rend VRPTW difficile, mais bien son approximation, même si les deux questions sont étroitement liées.

Notre théorème suggère assez clairement que tout rapport d'approximation raisonnable du VRPTW(1) euclidien devrait être fonction des fenêtres de temps. Or, un examen attentif de notre preuve (moyennant quelques changements mineurs pour que les bornes supérieures soient finies) montre que l'on a le résultat suivant (où *i* parcourt les sites) :

Théorème 5.2 : *Il n'existe pas de rapport d'approximation* $r \prec \frac{\max_i \{\beta(i)\}}{\max_i \{\alpha(i)\}}$ *pour le VRPTW(1) euclidien à moins que P=NP*

Les deux résultats indiquent que le VRPTW(1) euclidien est fortement intraitable pour l'approximation, contrairement à son homologue (TSP). Ceci nous amène à poser la conjecture suivante :

Conjecture : *VRPTW(1) Euclidien et VRPTW(k) Euclidien sont équivalents pour l'approximation.*

5.7.2. Résolution de VRPTW(1) par une méthode de coupes

Tenant compte de la similarité entre TSP et VRPTW(1) telle que reflétée dans la preuve du théorème 1, nous allons pousser l'analogie plus loin en explorant la possibilité de résoudre VRPTW(1) en adaptant les méthodes disponibles pour le TSP. La méthode des coupes de la programmation en nombres entiers est l'une de ces possibilités et nous allons

simplement montrer comment l'utiliser dans notre contexte. Dans toute la suite, nous supposerons que les distances et les temps de parcours sont tous positifs (strictement) et les durées de service sont négligeables.

Nous allons d'abord formuler notre problème comme un problème de programmation mathématique. Considérons donc *N* sites numérotés 0,1,…,*N*-1 avec une distance c_{ij} et un temps de parcours t_{ij} entre chaque paire de sites *ij*. Le site 0 désigne le dépôt. On duplique le site 0 en créant un site *N*, avec les distances et temps de parcours de *N* à tout autre site *j* définis comme c_{0j} et t_{0j} respectivement, et $c_{0n} = t_{0n} = 0$. Nous associons une variable de décision x_{ij} à chaque paire de sites *ij*, et une variable y_i pour tout site (client) *i*. Notons *t* le minimum de tous les t_{ij}. Ainsi, *t* est le temps de parcours minimum entre deux sites. VRPTW(1) se formule alors comme le programme mathématique suivant :

$$\text{Min} \sum_{ij} c_{ij} x_{ij} \ ,$$

Sous les contraintes :

(i) $y_0 = 0 \ ; y_i \geq 0$ pour tout site *i* ; $x_{0n} = 1$

(ii) $\alpha(i) \leq y_i \leq \beta(i)$ pour tout site *i*

(iii) $\left| y_i - y_j \right| \geq t_{ij} . x_{ij}$ pour toute paire de sites *ij*

(iv) $y_j \leq \max\left\{ y_i, y_j, y_k \right\} - t + M_1 (1 - x_{ij} x_{jk})$ pour tout triplet de sites *i,j,k*, où M_1 est une borne supérieure quelconque des y_j. On pourra prendre $M_1 = \sum_{ij} t_{ij}$.

(v) Le vecteur $x = (x_{ij})_{ij}$ est le vecteur caractéristique d'un cycle hamiltonien dans le graphe complet des sites.

Expliquons d'abord les contraintes avant de les linéariser. Chaque variable y_j désigne le temps de visite du client *j*. La condition (ii) spécifie les contraintes relatives aux fenêtres de temps. La condition (iii) stipule que le temps entre deux visites successives est au moins aussi long que le temps de parcours entre les deux sites correspondants. La condition (iv) assure que les temps de visite sont croissants dans le sens du parcours. Plus exactement, si *ij* et *jk* sont des arêtes du cycle hamiltonien, alors y_j n'est pas le maximum de $\left\{ y_i, y_j, y_k \right\}$ (le lecteur devra se convaincre que cette condition portant sur tous les triplets suffit à assurer la monotonie des y_j le long du parcours).

Linéarisons maintenant les contraintes (iii) et (iv).

Linéarisation de (iii) :

Introduisons deux variables y_{ij}^+, y_{ij}^- et six contraintes pour chaque contrainte de (iii) comme suit : $y_{ij}^+ \geq y_i \ ; \ y_{ij}^+ \geq y_j \ ; \ y_{ij}^- \leq y_i \ ; \ y_{ij}^- \leq y_j \ ; \ y_{ij}^+ + y_{ij}^- = \ y_i + y_j \ ; \ y_{ij}^+ - y_{ij}^- \geq t_{ij} . x_{ij}$

Comme nous voulons en plus que y_{ij}^+ soit égal au maximum de y_i et y_j, nous associons une grande valeur de pénalité (*M*) par unité de $y_{ij}^+ - (\frac{y_i + y_j}{2})$. Ainsi, la fonction objectif devient :

$(\sum_{ij} c_{ij}x_{ij}) + M\sum_{ij}(y_{ij}^{+} - \frac{y_i + y_j}{2})$. De cette façon, pour toute solution optimale, on aura : $y_{ij}^{+} = \max\{y_i, y_j\} \pm \varepsilon$, comme on le souhaitait, ou ε est la précision avec laquelle nos calculs sont effectués. Pour cela, on pourra choisir *M* plus grand que $\frac{1}{\varepsilon}\sum_{ij} t_{ij}$

Linéarisation de (iv) :

Le terme quadratique $x_{ij}x_{jk}$ peut être linéarisé par l'introduction d'une variable réelle z_{ijk} et les contraintes associées : $0 \le z_{ijk} \le 1$, $z_{ijk} \le x_{ij}$; $z_{ijk} \le x_{jk}$, $z_{ijk} \ge x_{ij} + x_{jk} - 1$

En plus, nous ajoutons une variable supplémentaire y_{ijk} pour chaque triplet *ijk* avec les contraintes suivantes: $y_j \le y_{ijk}$, $y_i \le y_{ijk}$, $y_k \le y_{ijk}$, et nous associons une grande pénalité à y_{ijk} . La condition (iii) devient alors : $y_j \le y_{ijk} - t + M_1(1 - z_{ijk})$, et la fonction objectif devient :

$$\sum_{ij} c_{ij}x_{ij} + M\sum_{ij}(y_{ij}^{+} - \frac{y_i + y_j}{2}) + M\sum_{ijk} y_{ijk}$$

Toutes ces transformations sont linéaires et n'introduisent pas de variables entières autres que les x_{ij} . Il nous reste donc un noyau dur de contraintes entières (v) communes au TSP, qui sont irréductibles. Ce sont toutefois des contraintes qui ont été bien étudiés dans la littérature sur le TSP. Padberg et Rao en particulier [Pad 03] ont distingué trois types d'inégalité (induisant des facettes du polyèdre des tournées du voyageur de commerce) qui pourraient être utilisées pour relâcher les contraintes d'intégrité des x_{ij} . Le polyèdre résultant (ayant un nombre exponentiel de facettes) pourrait toujours contenir des sommets fractionnaires, mais on peut espérer que l'approximation soit assez bonne pour nous permettre d'obtenir d'excellentes bornes inférieures pouvant être efficacement utilisées dans une approche branch-and-bound, par exemple. Les solutions fractionnaires générées peuvent aussi donner de bonnes solutions approchées à travers un arrondi aléatoire (randomized rounding).

Nous allons présenter maintenant ces trois types de contraintes à titre indicatif. Le lecteur intéressé pourra consulter la référence [Gro 88] pour une revue détaillée de ces méthodes.

Contraintes de degré :

Elles spécifient que dans tournée (un cycle hamiltonien), chaque sommet est de degré 2. $x(\delta(i)) = 2$ pour tout site (sommet) *i*.

Contraintes de 2-couplage :

Pour éliminer les solutions fractionnaires qui peuvent être induites par les contraintes de degré, on ajoute les contraintes suivantes :

$x(\delta(W) \setminus F) + |F| - x(F) \ge 1$ pour tout $W \subseteq N$ et $F \subseteq \delta(W)$ avec $|F|$ impair.

Contraintes d'élimination de sous-tours:

Ces contraintes spécifient qu'une tournée doit être connexe. Pour éliminer les sous-tours qui peuvent résulter des contraintes de 2-couplages, on ajoute alors les contraintes suivantes :

$$x(\delta(W)) \geq 2 \text{ pour tout } W \subseteq N$$

Comme il a été observé dans [Gro 88], tous ces ensembles de contraintes (bien que de tailles exponentielles) peuvent être séparés en temps polynomial, c'est à dire, pour un vecteur x donné, on peut décider en temps polynomial si x est une solution faisable pour toutes ces contraintes et, s'il ne l'est pas, on peut trouver une contrainte violée par x. Il est bien que le problème de la séparation est équivalent au problème de l'optimisation dans la programmation convexe et on peut donc optimiser sur toutes les contraintes précédentes.

5.8. Conclusion

Notre contribution a porté sur l'adaptation des algorithmes génétiques hybrides au VRPTW bi-objectif. L'intérêt de ce travail réside dans la diversité des approches de résolution implémentées. En effet, plusieurs combinaisons d'opérateurs de recombinaison, de stratégies de sélection, de techniques de maintien de la diversité et enfin de méthodes d'hybridation avec la recherche locale, ont été proposées et testées.

Cette multitude de méthodes nous a permis, en fonction des classes d'instances de Solomon [Sol 04], de mieux cerner le problème, de classifier ces différentes approches, de déterminer les valeurs des différents paramètres des algorithmes utilisés et enfin de trouver des solutions de bonne qualité. En effet, les résultats obtenus pour la classe C1 sont de qualité égale aux solutions fournies avec des méthodes exactes. La déviation du nombre de véhicules cumulé, pour les 56 instances de Solomon, ne dépasse pas 6% par rapport aux meilleurs résultats publiés [Sol 88], tandis que l'écart calculé par rapport à la distance totale parcourue est meilleur de 4 %. Il est important de signaler qu'une nouvelle solution a été trouvée pour le problème R203 (le nombre de véhicules est de 3, la distance parcourue est de 939,15).

Les résultats de complexité obtenus indiquent clairement que le problème considéré est fortement intraitable pour l'approximation, excluant jusqu'à l'existence d'un rapport d'approximation inférieur à N^{α}, pour tout $\alpha \prec 1$, à moins que P=NP, où N est le nombre de sites. Ainsi, le mieux que l'on puisse espérer à cet égard, est d'obtenir un rapport d'approximation qui soit fonction des fenêtres de temps $[\alpha(i),ß(i)]$, sans que ce rapport ne soit, là encore, inférieur à $(\max_i ß(i)/\max_i \alpha(i))$ (à moins que P=NP).
Par ailleurs nous avons indiqué comment les puissantes méthodes de coupes de la programmation linéaires, qui ont fait leurs preuves dans le traitement du TSP, peuvent être appliquées dans un contexte de fenêtres de temps.

Conclusion et perspectives

Dans cette thèse, nous avons abordé le domaine de la coopération entre des méthodes d'optimisation, dans le cadre de la résolution de problèmes NP-difficiles de l'optimisation combinatoires *mono* et *multiobjectifs*. Nous nous sommes particulièrement intéressés aux métaheuristiques séquentielles et hybrides, ainsi qu'à leur parallélisation.

Dans le cadre *monoobjectif*, nous avons proposé différentes métaheuristiques hybrides, ainsi que des schémas de coopération génériques pour la résolution de différents problèmes classiques de l'optimisation combinatoire : un problème d'ordonnancement de tâches sur une architecture parallèle, le problème de couverture d'ensembles et un problème de bioinformatique, le repliement de protéines.

Le problème d'ordonnancement de tâches sur une architecture parallèle est fondamental pour la conception d'algorithmes parallèles. Nous avons considéré un problème d'ordonnancement statique formé d'un ensemble de tâches (sans duplication et sans préemption) à ordonnancer sur une architecture multiprocesseur à mémoire distribuée, avec comme fonction de coût la minimisation de la date de terminaison de la dernière tâche (*makespan*).
Nous avons proposé deux algorithmes de recherche tabou, ainsi qu'une version parallèle de cette approche. Le premier algorithme est une méthode *Tabou* dans sa forme classique. Le second est une amélioration du premier, dans la mesure où il intègre des étapes d'intensification et de diversification. Le troisième algorithme est un algorithme parallèle d'une portée générale. En effet, d'une part, il est indépendant de l'architecture sur laquelle il s'exécute, et d'autre part, aucune hypothèse n'est faite sur la structure de l'application. Il s'agit d'un algorithme synchrone. Il est obtenu en particularisant le rôle de l'un des processus (le maître), qui est chargé de distribuer des solutions initiales et des stratégies entre les différents processus esclaves. Chacun de ces processus esclaves applique alors un algorithme de recherche tabou sur la solution initiale qui lui a été attribuée par le maître, et selon la stratégie choisie par ce dernier également. Les solutions trouvées par les esclaves sont ensuite envoyées au processus maître qui sélectionne la meilleure d'entre elles.
D'après les tests effectués sur les différents algorithmes développés, nous avons remarqué que les résultats obtenus dépendaient de la nature du graphe de précédence (durées d'exécution des tâches, coûts de communication entre paires de tâches, densité du graphe). En effet, lorsque les coûts de communication sont faibles, les tâches se répartissent assez bien sur les différents processeurs. Dans le cas contraire, elles ont tendance à s'exécuter sur un nombre restreint de processeurs. Par ailleurs, nous avons montré que le problème est NP-complet même pour des cas particuliers qui nous intéresse. Nous avons également montré que le problème est dur au regard de l'approximation dans le sens qu'il n'admet pas de schéma d'approximation polynomial à moins que P=NP. Ainsi, nous avons adopté une approche plus pratique à ce problème.

Au travers du problème de couverture d'ensembles, deux algorithmes parallèles à base de colonies de fourmis et deux hybridations avec la recherche locale ont été proposés. Dans un premier temps, nous avons proposé un algorithme de colonies de fourmis simple. Cet algorithme a ensuite été hybridé avec une recherche locale. Les deux types d'hybridations qui ont été mises en œuvre sont l'hybridation *HRH* (*High-Level Relay Hybrid*) et l'hybridation

HCH (High Co-evolutionary Hybrid). Dans les hybridations de type *HRH*, chaque métaheuristique qui intervient dans ce processus d'hybridation reste inchangée et autonome. Les méthodes *HRH*-hybridées sont exécutées en séquence. Au contraire, dans les hybridations *HCH*, les algorithmes hybridés, dont la structure interne n'est pas modifiée, sont exécutés simultanément et coopèrent pour résoudre le problème. Les tests effectués ont montré que la deuxième hybridation était meilleure que la première. Les résultats obtenus ont ensuite été améliorés en utilisant différentes implémentations parallèles de l'algorithme hybride le plus performant. Ces implémentations parallèles ont permis de réduire le temps de recherche ; cependant nous avons remarqué que l'efficacité de la parallélisation dépendait de la taille du problème traité. Pour le problème de couverture d'ensembles, la valeur de l'accélération (*speedup)* n'était pas uniformément distribuée par rapport aux instances traitées. Cette valeur augmentait proportionnellement avec la taille de l'instance traitée. L'augmentation de l'accélération s'explique par le fait que le temps de communication, entre le processus maître et les processus esclaves, est très élevé par rapport au temps de traitement sur chacun des processus esclaves pour les instances de petite taille. Par ailleurs, pour les instances de grande taille, les temps de communication sont faibles.

Dans le cadre de la coopération entre une méthode exacte et une métaheuristique, nous avons proposé un certain nombre d'algorithmes. Ces derniers font coopérer la programmation par contraintes avec les colonies de fourmis. Ces algorithmes ont été développés sous l'environnement de la programmation par contraintes *Mozart-Oz* et appliqués au problème de repliement de protéines.
Deux modèles à base de la programmation par contraintes ont été proposés. Le premier, simple et de base, est insuffisant, même hybridé avec une recherche locale basée sur le voisinage *Pull-Move*. Un autre modèle, plus élaboré, a été proposé. Ce modèle introduit de nouvelles contraintes basées sur la notion de "cadre". Il ne donne des résultats satisfaisants que pour les protéines de taille inférieure à 50. Pour pallier à ce manque, nous avons proposé un schéma de coopération entre la programmation par contraintes et les colonies de fourmis. Cette coopération permet une meilleure recherche de la solution, en réduisant l'espace de recherche exploré par la programmation par contraintes de deux manières. D'une part, il est possible d'exploiter les renseignements quant à une solution déjà calculée. Ces renseignements sont exprimés par des contraintes additionnelles : après qu'une solution ait été trouvée, la contrainte supplémentaire qu'une prochaine solution doit être meilleure est prise en considération. Avec cette contrainte supplémentaire, l'arbre de recherche peut devenir considérablement plus petit. D'autre part, les valeurs de taux de phéromone calculés par les fourmis, sont utilisées pour explorer l'arbre de recherche. Il s'agit en fait de choisir l'ordre des directions à explorer pour chaque monomère, et ainsi commencer par explorer les directions les plus prometteuses (c'est-à-dire la branche sur laquelle la valeur de la phéromone est la plus importante). Les résultats de cette coopération entre une métaheuristique et la programmation par contraintes sont satisfaisants. Le temps de calcul de la programmation par contraintes est d'autant plus réduit que la recherche par colonies de fourmis est de qualité.
Enfin, cette étude a permis de constater que ces algorithmes génériques peuvent être appliqués à d'autres problèmes d'optimisation, moyennant, éventuellement, des modifications spécifiques à la problématique. En effet, la programmation par contraintes permet l'ajout de contraintes au problème initial, sans que cela n'altère le code déjà développé, pour la coopération avec les colonies de fourmis.

Dans le cas *multiobjectif*, nous avons proposé des algorithmes génétiques ayant pour objectif d'approcher la frontière de Pareto. A cet effet, plusieurs mécanismes ont été introduits dans notre algorithme : l'élitisme, l'hybridation, le parallélisme, la diversification génotypique,

la diversification phénotypique, etc. Nos algorithmes ont été testés sur les problèmes du *flow-shop* ainsi que le problème de tournées de véhicules avec des fenêtres horaires.

Le problème du *flow-shop* à permutation bi-objectif a été traité, avec comme objectif la minimisation du temps de terminaison (*makespan*) et la somme des retards de toutes les tâches. Nous avons utilisé pour sa résolution des algorithmes génétiques. Les tests effectués sur cinq stratégies de sélection montrent que les stratégies Pareto (NSGA, NDS, WAR) sont celles qui fournissent les meilleurs résultats. Ces stratégies partagent en commun le fait de s'appuyer sur la notion de dominance, afin de classer entres elles des solutions de la population.
D'une manière générale, les algorithmes génétiques sont souvent critiqués pour leur lenteur. L'hybridation de la méthode avec la recherche locale offre alors une solution à ce problème. L'algorithme génétique proposé est lancé en premier, fournissant une première approximation de la frontière de Pareto, la recherche locale est ensuite lancée avec pour entrée chacune des solutions de l'ensemble Pareto fourni par l'algorithme génétique. L'effet de cette hybridation, du type HRH, n'est remarquable que pour des problèmes de grande taille.
Pour réduire les temps de réponse et augmenter la taille de la population, un modèle d'algorithme génétique parallèle a été proposé. Il se base sur une approche distribuée. Dans ce cas, plusieurs algorithmes génétiques sont lancés, chacun ayant pour entrée une sous-population initiale d'individus. A des périodes de temps déterminées, une migration circulaire est effectuée entre les différents processus de recherche.
Les algorithmes génétiques sont aussi connus pour leur sensibilité quant au choix de la population initiale. Nous avons alors proposé une adaptation probabiliste de l'heuristique *NEH* [Naw 83]. Ainsi, la moitié des individus de cette population ont été construits de façon à présenter de bons temps de terminaison (*makespan*), l'autre moitié étant construits afin d'offrir de bons temps de retard.
L'absence de benchmarks pour les problèmes multiobjectifs est souvent une difficulté à laquelle doit faire face le concepteur d'algorithmes d'optimisation. Nous avons proposé dans cette thèse un générateur d'instances pour le problème *flow-shop* bi-objectif. Ce générateur se présente comme une adaptation de l'algorithme de *Taillard,* souvent référencé dans la littérature [Tai 93a]. Nous estimons que ces instances fourniront une bonne base de comparaison pour les travaux futurs. Une batterie de tests ont été effectués, démontrant une certaine habilité de nos algorithmes à trouver des solutions de faible compromis (solutions extrêmes).

Enfin, nous avons étudié le problème de tournées de véhicules avec des fenêtres horaires. Notre contribution a porté sur l'adaptation d'algorithmes génétiques hybrides au VRPTW bi-objectif (distance parcourue, nombre de véhicules), en utilisant l'approche Pareto. Nous avons enrichi l'algorithme génétique de base par l'introduction des éléments suivants :

- une heuristique pour la génération de la population initiale,
- des opérateurs de croisement (*RBX* et *SBX*),
- des opérateurs de mutation (*swap* (*reinsert*) et *One Level Exchange*),
- des approches de sélection (*NSGA* et *ELITISME*),
- des techniques de partage *(sharing)* pour la diversification,
- un algorithme hybride avec la recherche locale.

Les algorithmes développés ont été testés sur les instances de Solomon [Sol 88]. Les solutions extrêmes de la courbe de Pareto ont été comparées aux meilleurs résultats monoobjectifs de la littérature. Il en ressort que nos solutions sont de bonne qualité pour les instances de la classe C1 et mitigées pour les autres classes du fait de leur nature et de leur complexité. Cependant,

nos courbes de Pareto offrent aux décideurs un éventail de solutions, leur permettant ainsi de faire un choix de compromis.
Par ailleurs, les résultats de complexité obtenus indiquent clairement que le problème considéré est fortement intraitable pour l'approximation, excluant jusqu'à l'existence d'un rapport d'approximation inférieur à N^{α}, pour tout $\alpha \prec 1$, à moins que P=NP, où N est le nombre de sites. Ainsi, le mieux que l'on puisse espérer à cet égard, est d'obtenir un rapport d'approximation qui soit fonction des fenêtres de temps $[\alpha(i),ß(i)]$, sans que ce rapport ne soit, là encore, inférieur à $(\max_i ß(i)/\max_i \alpha(i))$ (à moins que P=NP).
Par ailleurs nous avons indiqué comment les puissantes méthodes de coupes de la programmation linéaires, qui ont fait leurs preuves dans le traitement du TSP, peuvent être appliquées dans un contexte de fenêtres de temps.

Parmi les extensions possibles de ces travaux, aussi bien dans le cas monoobjectif que dans le cas multiobjectif, on peut citer :

- l'adaptation des algorithmes génériques développés entre la programmation par contraintes et la métaheuristique des colonies de fourmis à d'autres problèmes d'optimisation ;

- la prise en charge et le développement d'outils de résolution de problèmes d'optimisation multiobjectif comprenant plus de deux objectifs ;

- Dans le cadre de l'hybridation de métaheuristiques avec la recherche locale, il serait intéressant de développer des techniques d'exploration de grands voisinages de tailles exponentielles en temps polynomial. On peut notamment penser à la programmation dynamique, qui a été utilisée avec succès pour le problème de voyageur de commerce, pour explorer des voisinages de grande taille [Ang 02][Con 02] ;

- Il serait intéressant d'explorer d'autres pistes de coopération entre des méthodes exactes et des métaheuristiques ;

- La proposition d'autres techniques de parallélisation des méthodes hybrides est possible ;

- Le développement de métaheuristiques adaptatives (méthodes mimétiques, opérateurs de mutation spécialisés et adaptatifs, etc.) est un axe à explorer pour la résolution de problèmes d'optimisation combinatoire d'une manière plus efficace ;

- Enfin, il est sans doute possible de mener une analyse plus rigoureuse des résultats obtenus dans le cadre de l'optimisation multiobjectif, en utilisant les différentes métriques de la littérature et en utilisant des tests statistiques [Tai 03].

Nous travaillons, actuellement, sur l'amélioration des approches de coopération proposées entre la programmation par contraintes et les colonies de fourmis ainsi que sur l'exploration d'autres types de coopération.

Références bibliographiques

[Ade 04] B. Adenso-Diaz. Restricted neighborhood in the tabu search for flow-shop problem. European Journal of Operational Research, 62, 27-37, 1992.

[Aie 03] M. Aiex, S. Binato, M.G.C. Resende. Parallel GRASP with path-relinking for job shop scheduling. Parallel Computing, 29, 393-430, 2003.

[Alf 91] A.S. Alfa, S.S. Heragu, M. Chen. A 3-opt based simulated annealing algorithm for vehicle routing problems. Computers and Industrial Engineering, 21, 635-639, 1991.

[And 97] E. Andersson, E. Housos, E. Kohl, D. Wedelin. Crew pairing optimisation. Operations Research in the Airline Industry. Kluwer Scientific Publishers, 1997.

[Ang 02] E. Angel, E. Bampis, L. Gourvès. A dynasearch neighborhood for the bicriteria travelling salesman problem. X. Gandibleux, M. Sevaux, K. Sörensen and V. T'kindt, Editors, Lecture Notes in Economics and Mathematical Systems, Springer, 535, 153-176, 2002.

[Ang 05] E. Angel, E. Bampis, M. Rahoual, H. Bouchema. Ants metaheuristic and constraint programming cooperation for the folding protein problem. Proceedings of the 6th Metaheuristics International Conference (MIC'2005), Vienna, 57-62, Aout 2005.

[Aro 96] L.D. Aronson. Algorithms for vehicule routing – a survey. Technical Report 96-21, Delft University of Technology, 1996.

[Bäc 00] T. Bäck, D.B. Fogel, Z. Michalewicz. Evolutionary computation 1: Basic algorithms and operators. Institute of Physics Publishing, 2000.

[Bac 01] R. Backofen. The protein structure prediction problem: A constraint optimisation approach using a new lower bound. Constraints, 6, 223-255, 2001.

[Bac 96] V. Bachelet, P. Preux, E.G. Talbi. Parallel hybrid metaheuristics: Application to the quadratic assignment problem. Parallel Optimization Colloquium POC96, Versailles, France, 233-242, Mars 1996.

[Bac 98] V. Bachelet, Z. Hafidi, P. Preux, E.G. Talbi. Vers la coopération de métaheuristiques parallèles. Calculateurs Parallèles, Réseaux et Systèmes Répartis, 10(2), 211-223, 1998.

[Bak 85] J.E. Baker. Adaptive selection methods for genetic algorithms. In Proceedings of the First International Conference on Genetic Algorithms and their Applications, J.J. Grefenstette editor, Hillsdale, New Jersey, USA, 101-111. July 1985.

[Bak 03] B.M. Baker, M.A. Ayechew. A genetic algorithm for the vehicle routing problem. Computers and Operations Research, 30, 787-800, 2003.

[Bal 80a] E. Balas. Cutting planes from conditional bounds: A new approach to set covering. Mathematical Programming, 12, 19-36, 1980.

[Bal 80b] E. Balas, A. Ho. Set covering algorithms using cutting planes and subgradient optimization: A computational study. Mathematical Programming, 12, 37-60, 1980.

[Bal 94] S. Baluja. Population-based incremental learning: A method for integrating genetic search based function optimization and competitive learning. Technical Report CMU-CS-94-163, Carnegie Mellon University, Pittsburgh, PA, USA, 1994.

[Bal 95] S. Baluja, R. Caruana. Removing the genetics from the standard genetic algorithm. In A. Prieditis and S. Russel, editors, Proceedings of the International Conference on Machine Learning (ML-95). Morgan Kaufmann Publisher, San Mateo, CA, 38-46, July 1995.

[Bal 96] E. Balas, M.C. Carrera. A dynamic subgradient-based branch and bound procedures for set covering. Operations Research, 44, 875-890, 1996.

[Bap 01] P. Baptiste. Combining operations research and constraint programming to solve real-life scheduling problems. ERCIM News, 44, 53-53, 2001.

[Bar 98] R. Barták. On-line guide to constraint programming. Prague, http://kti.mff.cuni.cz/~bartak/constraints/, 1998.

[Bar 99] R. Barták. Constraint programming: A survey of solving technology. In AIROnews Journal, 4, 7-11, 1999.

[Bar 03] B. Baràn, M. Schaerer. A multiobjective ant colony system for vehicle routing problem with time windows. In IASTED International Conference on Applied Informatics, Innsbruck, Austria, 97-102, February 2003.

[Bas 05] M. Basseur. Conception d'algorithmes coopératifs pour l'optimisation multiobjectif : Application aux problèmes d'ordonnancement de type *flow-shop*. Thèse de doctorat de l'université de Lille, juin 2005.

[Bea 83] J.E. Beasley. Route-first cluster-second methods for vehicle routing. Omega, 11, 403-408, 1983.

[Bea 87] J.E. Beasley. An algorithm for set covering problem. European Journal of Operational Research, 31, 85-93, 1987.

[Bea 90] J.E. Beasley. OR-Library: Distributing test problems by electronic mail. Journal of the Operational Research Society, 41, 1069-1072, 1990.

[Bea 92] J. E. Beasley, K. Jornsten. Enhancing an Algorithm for Set Covering Problem. European Journal of Operational Research, 58, 293-300, 1992.

[Bea 96] J.E. Beasley, P.C. Chu. A genetic algorithm for the set covering problem. European Journal of Operational Research, 95(5), 393-404, 1996.

[Ben 89] D. Benhamamouch, G. Plateau. Task allocation in distributed computing systems. Rapport technique, Université Paris-Nord, LIPN, Villetaneuse, 1989.

[Ben 98] M. Ben-Daya, M. Al-Fawzan. A tabu search approach for the flow-shop scheduling problem. European Journal of Operational Research, 109, 88-95, 1998.

[Ber 98a] B. Berger, T. Leighton. Protein folding in the hydrophobic-hydrophilic (HP) model is NP complete. Proceedings of the Second Annual International Conference on Computational Molecular Biology (RECOMB'98), New York, 30-39, March 1998.

[Ber 98b] J. Berger, M. Salois, R. Begin. A hybrid genetic algorithm for the vehicle routing problem with time windows. Lecture Notes in Artificial Intelligence 1418, AI'98 Advances in Artificial Intelligence, Vancouver, 114–127, 1998.

[Ber 01] J. Berger, M. Barkaoui, O. Bräysy. A Parallel hybrid genetic algorithm for the vehicle routing problem with time windows. Working paper, Defense Research Establishment Valcartier, Canada, 2001.

http://neo.lcc.uma.es/radi-aeb/Webvrp/data/articles/hybrid2.pdf,

[Bil 99] J.C. Billaut, V. T'Kindt. Les problèmes d'ordonnancement d'atelier multiobjectif. Rapport Interne n° 206, Ecole d'ingénieur en informatique pour l'industrie, Université François Rabelais, 1999.

[Bla 87] J. Blazewicz. Selected topics in scheduling theory. Discrete Applied Mathematics, 31, 1-60, 1987

[Bla 93] J.L. Blanton, R.L. Wainwright. Multiple vehicle routing with time and capacity constraints using genetic algorithms. In S. Forrest, editor, Proceedings of the Fifth International Conference on Genetic Algorithms, Morgan Kaufmann Publisher, San Mateo, CA, 452-459, June 1993.

[Bof 95] B. Boffey. Multiobjective routing problems. Journal Top, 3(2), 167-220, 1995.

[Bok 81] S. H. Bokhari. On the mapping problem. IEEE Transactions on Computers, C-30(3), 207-214, 1981.

[Bol 88] S.W. Bollinger, S.F. Midkiff. Processor and link assignment in multicomputers using simulated annealing. Proceedings of the International Conference Parallel Processing. The Pennsylvania State University, University Park, PA, USA, 1-7, August 1988.

[Bou 94] P. Bouvry. Placement de tâches sur ordinateurs parallèles à mémoire distribuée, Thèse de doctorat, de l'université de Grenoble, octobre 1994.

[Bow 95] R. Bowerman, B. Hall, P. Calamai. A multi-objective optimization approach to urban school bus routing: Formulation and solution method. Transportation Research Part A, 29, 197-123, 1995.

[Brä 99a] O. Bräysy. A hybrid genetic algorithm for the vehicle routing problem with time windows. Licentiate thesis, University of Vaasa, Finland, 1999.

[Brä 99b] O. Bräysy. A new algorithm for the vehicle routing problem with time windows based on the hybridization of a genetic algorithm and route construction heuristics. Research papers 227, University of Vaasa, Finland, 1999.

[Brä 00] O. Bräysy, J. Berger, M. Barkaoui. A new hybrid evolutionary algorithm for the vehicle routing problem with time windows. Presented at the Route 2000 Workshop, Skodsborg, Denmark, August 2000.

[Brä 01] O. Bräysy, M. Gendreau. Genetic algorithms for the vehicle routing problem with time windows. Internal Report STF42A01021, SINTEF (Foundation for Scientific and Industrial Research at the Norwegian Institute of Technology) Applied Mathematics, Department of Optimisation, Oslo, Norway, 2001.

[Bra 91] S.B. Brah, J.L. Hunsucker. Branch and Bound algorithm for the flow-shop with multiple processors. European Journal of Operational Research, 51, 88-89, 1991.

[Bro 87] G.B. Brown, G.W. Graves, D. Ronen. Scheduling ocean transportation of crude oil. Management Science, 33, 335-346, 1987.

[Bul 98] B. Bullnheimer, G. Kotsis, C Strauss. Parallelization strategies for the ant system. In R. De Leone, A. Murli, P. Pardalos, and G. Toraldo, editors, High Performance Algorithms and Software in Nonlinear Optimization, Applied Optimization, Kluwer Academic Publishers, Dordrecht, NL, 24, 87-100, 1998.

[Cap 00] A. Caprara, P. Toth, M. Fischetti. Algorithms for the set covering problem. Annals of Operations Research, 98, 353-371, 2000.

[Cap 99] A. Caprara, M. Fischetti, P. Toth. A heuristic method for the set covering problem. Operations Research, 47, 730-743, 1999.

[Car 00] J. Carlier, E. Neron. An exact method for solving the multi-processor flow-shop. RAIRO - Recherche Opérationnelle, 34(1), 1-25, 2000.

[Cat 01] M.S.F. Catalano, F. Malucelli. Parallel randomized heuristics for the set covering problem. Practical Parallel Computing Huntington, New York, Nova Science Publishers, 113-132, 2001.

[Cha 93a] J. Chakrapani, J. Skorin Kapov. Connection machine implementation of a tabu search algorithm for the traveling salesman problem. Journal of Computing and Information Technology, 1(1), 29-36, 1993.

[Cha 93b] J. Chakrapani, J. Skorin Kapov. Massively parallel tabu search for the quadratic assignment problem. Annals of Operations Research, 41, 327-341. 1993.

[Cha 02] I.M. Chao. A tabu search method for the truck and trailer problem. Computers and Operations Research, 29, 22-51, 2002.

[Cha 03] V. Chandrum, A. Datta Shama, V.S. Anil Kumar. The algorithms of folding proteins on lattices. Discrete Applied Mathematics, 127(1), 145-161, 2003.

[Chi 97] W.C. Chiang, R.A. Russell. A Reactive tabu search metaheuristic for the vehicle routing problem with time windows. Informs Journal on Computing 9, 417-430, 1997.

[Chu 98] P.C. Chu. Constraint handling in genetic algorithms: The set partitioning problem. Journal of Heuristics, 4, 323-357, 1998.

[Chv 79] V. Chvàtal. A greedy heuristic for the set covering problem. Mathematics of Operations Research, 4(3), 13-235, 1979.

[Chv 83] V. Chvátal. Linear Programming. Freeman, New York. 1983.

[Coe 02] C.A.C. Coello, D.A. Van Veldhuizen, G.B. Lamont. Evolutionary algorithms for solving multi-objective problems. Kluwer Academic Publishers, New York, 2002.

[Cof 72] E.G. Coffman, R.L. Graham. Optimal scheduling for two processor systems. Acta Informatica, 1, 200-213, 1972.

[Col 91a] J.Y. Colin. Problème d'ordonnancement avec délais de communication : complexité et algorithmes. Thèse de doctorat, Université Paris VI, Novembre 1991.

[Col 91b] J.Y. Colin, P. Chrétienne. CPM scheduling with small interprocessor communication delays. Operations Research, 39(3), 680-684, 1991.

[Col 92] A. Colorni, M. Dorigo, V. Maniezzo. Distributed optimization by ant colonies. Proceedings of ECAL'91 – First European Conference on Artificial Life, edited by F. Varela and al., Elsevier Publishing, Paris, France, 134-142, 1992.

[Col 02] Y. Collette, P. Siarry. Optimisation multiobjectif. Eyrolles 2002.

[Con 67] R. Conway, W. Maxwell, L. Miller. Theory of scheduling. Addison-Wesley, 1967.

[Con 02] R.K. Congram, C.N. Potts, S.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.

[Coo 98] W.J. Cook, W.H. Cunningham, W.R. Pulleyblank, A. Schrijver. Combinatorial Optimization. John Wiley and Sons, New York, 1998.

[Cor 01] J.F. Cordeau, G. Laporte, A. Mercier. A unified tabu search heuristic for vehicle routing problems with time windows. Journal of the Operational Research Society, 52, 928-936, 2001.

[Cor 02] A. Corberan, E. Fernandez, M. Laguna, R. Marti. Heuristic solutions to the problem of routing school buses with multiple objectives. Journal of the Operational Research Society, 53, 427-435, 2002.

[Cos 93] M. Cosnard. Architectures et algorithmes parallèles. Inter-edition, 1993.

[Cre 98] P. Crescenzi, D. Goldman, C. Piccolboni, M. Yannakanis. On the complexity of protein folding. Journal of Computational Biology 5(3), 423-465, 1998.

[Dan 59] G.B. Dantzig, J.H. Ramser. The truck dispatching problem. Management Science, 6, 80-91, 1959.

[Das 81] M.S. Daskin. A hierarchical objective set covering model for emergency medial service vehicle deployment. Transportation Science, 15, 137-151, 1981.

[Deb 01] K. Deb. Multi-objective optimization using evolutionary algorithms. John Wiley & Sons, Chichester, UK, 2001.

[Deb 02] K. Deb, A. Pratap, S. Agarwal, T. Meyarivan. A fast and elitist multiobjective genetic algorithm: NSGA-II. IEEE Transaction on Evolutionary Computation, 6(2), 82-197, 2002.

[Dew 89] A. De Werra, A. Hertz. Taboo search techniques: A tutorial and an application to neural networks. OR Spektrum, 11, 131-141, 1989.

[Dil 95] K.A. Dill, S. Bornberg, K. Yue, K. Fiebig, D. Yee, P. Thomas, H. Chan. Principles of Protein Folding – a perspective from simple exact models. Protein Science, 4, 561-602, 1995.

[Doe 04] K. Doerner, R. F. Hartl, G. Kiechle, M. Lucka, M. Reimann. Parallel ant systems for the capacitated vehicle routing problem. In J. Gottlieb and G. R. Raidl, editors, Proceedings of EvoCOP 2004, Lecture Notes in Computer Science, 3004, 72-83. Springer-Verlag, 2004.

[Dor 92] M. Dorigo, V. Maniezzo, A. Colorni. An investigation of some properties of an "Ant algorithm". Proceedings of Parallel Solving from Nature (PPSN'92), Elsevier Publishing, Brussels, Belgium, 509-520, September 1992.

[Dor 96] M. Dorigo, V. Maniezzo, A. Colorni. Ant system: Optimization by a colony of cooperating agents. IEEE Transactions on Systems, Man and Cybernetics-Part B, 26(1), 29-41, 1996.

[Dor 97] M. Dorigo, L.M. Gambardella. Ant colony system: A cooperative learning approach to the TSP. IEEE Transactions on Evolutionary Computation, 1, 53-66, 1997.

[Dor 04] M. Dorigo, T. Stützle. Ant colony optimization. MIT Press, 2004.

[Dré 03] J. Dréo, A. Pétrowski, P. Siarry, E. Taillard. Métaheuristiques pour l'optimisation difficile. Eyrolles, 2003.

[Dun 90] R. Duncay. A survey of parallel computer architectures. Computer, 23(2), 5-16, 1990.

[Ehr 02] M. Ehrgott, X. Gandibleux (editors). Multiple criteria optimization: State of the art annotated bibliographic surveys. Kluwer Academic Publishers, Boston, 2002.

[Els 01] N. El-Sherbeny. Resolution of a vehicle routing problem with multi-objective simulated annealing method. Thèse de doctorat, Faculté Polytechnique de Mons, Belgique, décembre 2001.

[Ere 99] A.V. Eremeev. A genetic algorithm with a non-binary representation for the set covering problem. Proceedings of Operations Research (OR'98). Springer-Verlag, 175-181, September 1999.

[Ere 00] A.V. Eremeev, A.A. Kolokov, L.A. Zaozerskaya. A hybrid algorithm for set covering problem. Proceedings of International Workshop "Discrete Optimization Methods in Scheduling & Computers - Aided Design", Minsk, 123-129, October 2000.

[Fal 91] E. Falkenauer. A genetic algorithm for job shop. Proceedings of IEEE International Conference on Robotics and Automation, 1, 824-829, June 1991.

[Fal 97] I. De Falco, R. Del Balio, E. Tarantin. An analysis of parallel heuristics for task allocation in multicomputers. Computing, 59, 259-275, 1997.

[Fei 05] D. Feillet, P. Dejax, M. Gendreau. Traveling salesman problem with profits. Transportation Science, 39(2), 188-205, 2005.

[Feo 91] T.A. Feo, K. Venkatraman, J.F. Bard. A GRASP for a difficult single machine scheduling problem. Computers and Operations Research, 18, 635-643, 1991.

[Fis 90] M.L. Fisher, P. Kedia. Optimal solution of set covering/partitioning problems. Mathematical Programming, 36, 674-688, 1990.

[Fon 95] C.M. Fonseca, P.J. Fleming. Multiobjective genetic algorithms made easy: Selection, sharing and mating restrictions. In First IEE/IEEE International Conference On Genetic Algorithms in Engineering Systems: Innovations and Applications, Sheffield, UK, 45-52, October 1995.

[Fuj 98] K. Fujita, N. Hirokawa, S. Akagi, S. Kimatura, H. Yokohata. Multi-objective optimal design of automotive engine using genetic algorithm. In Proceedings of DETC'98 - ASME Design Engineering Technical Conferences, Atlanta, Georgia, 11-22, September 1998.

[Gam 99] L.M. Gambardella, E.D. Taillard, G. Agazzi. MACS-VRPTW: A multiple ant colony system for vehicle routing problems with time windows. In New Ideas in Optimization, D. Corne, M. Dorigo and F. Glover, editors, 63-76, Mc Graw-Hill, London, 1999.

[Gan 97] X. Gandibleux, N. Mezdaoui, A. Freville. A tabu search procedure to solve multi objective combinatorial optimization problems. In R. Caballero, F. Ruiz, and R. Steuer, editors, Advances in Multiple objective and Goal Programming, Lecture Notes in Economics and Mathematical Systems, 455, 291-300, Springer-Verlag, 1997.

[Gar 78] M.R. Garey, R.L. Graham, D.S. Johnson. Performance guarantees for scheduling algorithms. Operations Research, 26(1), 3-21, 1978.

[Gar 79] M.R. Garey, J.S. Johnson. Computers and intractability: A guide to the theory of NP-Completeness. W.H. Freeman and Co., San Francisco, 1979.

[Geh 99] H. Gehring, J. Homberger. A Parallel hybrid evolutionary metaheuristic for the vehicle routing problem with time windows. In Proceedings of EUROGEN99 - Short Course on Evolutionary Algorithms in Engineering and Computer Science, Reports of the Department of Mathematical Information Technology, Series A. Collections, n° A 2/1999, K. Miettinen, M. Mäkelä and J. Toivanen (eds), 57-64, University of Jyväskylä, Finland. May 1999.

[Geh 01] H. Gehring, J. Homberger. Parallelization of a two-phase metaheuristic for routing problems with time windows. Asia-Pacific Journal of Operational Research, 18, 35-47, 2001.

[Gen 94] M. Gendreau, A. Hertz, G. Laporte. A tabu search heuristic for the vehicle routing problem. Management Science, 40, 1276-1290, 1994.

[Glo 77] F. Glover. Heuristics for integer programming using surrogate constraints. Decision Sciences, 8, 156-166, 1977.

[Glo 86] F. Glover. Future path for integer programming and links to artificial intelligence. Computers and Operations Research, 13(5), 533-549, 1986.

[Glo 89] F. Glover. Taboo search - Part I. ORSA Journal of Computing, 1(3), 190-206, 1989.

[Glo 90] F. Glover. Tabu search - Part II. ORSA. Journal on Computing, 2(1), 4-32, 1990.

[Glo 95] F. Glover, F. Tseng. Tabu search. Optimization Techniques and Applications, World Scientific Press, 2, 1416-1423, 1995.

[Gol 87] D.E. Goldberg, J. Richardson. Genetic algorithms with sharing for multimodal function optimization. In Grefenstette, editor, Proceedings of the Second Int. Conf. on Genetic Algorithms (ICGA'2), Morgan Kaufmann Publishers, 41-49, San Mateo, CA, October 1987.

[Gol 89] D.E. Goldberg. Genetic algorithms in search, optimization, and machine learning. Addison-Wesley, 1989.

[Gol 94] D.E. Goldberg. Algorithmes génétiques: Exploitation, optimisation et apprentissage automatique. Adison - Wesley 1994.

[Gon 78] T. Gonzalez, S. Sahni. Flowshop and job-shop schedules: Complexity and approximation. Operational Research, 26(1), 36-52, 1978.

[Gou 05] L. Gourvès. L'optimisation multicritère et l'approximation. Thèse de doctorat, de l'université d'Evry-Val d'Essonne. Novembre 2005.

[Gra 79] R.L. Graham, E.L. Lawler, J.K. Lenstra, A.H.G. Rinooy Kan. Optimization and approximation in deterministic sequencing and scheduling theory: A survey. Annals of Discrete Mathematics, 5, 287-326, 1979.

[Gra 02] J. Grabowski, J. Pempera. New block properties for the permutation flow-shop problem with application in tabu search. Journal of the Operational Research Society, 52(2), 210-220, 2002.

[Gre 99] P. Greistorfer. Hybrid genetic taboo search for a cyclic scheduling problem. In S. Voss, S. Martello, I.H. Osman, C. Roucairol, editors, Meta-heuristics advances and trends in local search paradigms for optimization, Boston Kluwer Academic Publishers, 213-229, 1999.

[Gre 04] H.J. Greenberg, W.E. Hart, G. Lancia. Opportunities for combinatorial optimization in computational biology. Informs Journal on Computing, 16(3), 211-231, 2004.

[Gro 88] M. Groetscehl, L. Lovasz, A.Shrijver. Geometric Algorithms and Combinatorial Optimization, Springer Verlag, Berlin Heidelberg, 1988.

[Had 00] R. Hadji, M. Rahoual, E.G. Talbi, V. Bachelet. Ant colonies for the set covering problem. From Ant Colonies to Artificial Ants: Second International Workshop on Ant Algorithms ANTS'2000, Brussels, Belgium, 63-66, September 2000.

[Haj 92] P. Hajela, C.Y. Lin. Genetic search strategies in multicriterion optimal design. Structural Optimization, 4, 99-107, 1992.

[Han 86] P. Hanssen. The steepest ascent mildest descent heuristic for combinatorial programming. In Congress on Numerical Methods in Combinatorial Optimization, Capri, Italy, March 1986.

[Hao 03] M. Haouari, T. Ladhari. A branch and bound based local search for the flowshop problem. Journal of the Operational Research Society, 54(10), 1076-1084, 2003.

[Har 68] P.E. Hart, N.J. Nilsson, B. Raphael. A formal basis for the heuristic determination of minimum cost paths. IEEE Transactions on Systems Science and Cybernetics, 4, 100-107, 1968.

[Hel 60] J. Heller. Some numerical experiments for a MxJ flow-shop and its decision theoretical aspects. Operational Research, 8, 178-184, 1960.

[Hen 96] J.L. Hennessy, D.A Patterson. Architecture des ordinateurs : une approche quantitative. International Thomson Publishing France, 1996.

[Hey 83] A.M. Hey, N. Christofides. Algorithms for the set covering problem using graph theory. Working Paper, Imperial College, University of London, 1983.

[Hol 75] J. Holland. Adaptation in natural and artificial systems. Michigan Press Univ., Ann Arbor, MI, USA. 1975.

[Hol 92] J. Holland. Les algorithmes génétiques. Pour la Science, 179, 44-51, 1992.

[Hom 99] J. Homberger, H. Gehring. Two evolutionary meta-heuristics for the vehicle routing problem with time windows. Infor 37, 297-318, 1999.

[Hon 99] S-C. Hong, Y-B. Park. A heuristic for a bi-objective vehicle routing with time window constraints. International Journal of Production Economics, 62, 249-258, 1999.

[Hu 61] T.C. Hu. Parallel sequencing and assembly line problems. Operations Research, 9(6), 841-848, 1961.

[Ish 03] H. Ishibuchi, T. Yoshida, T. Murata. Balance between genetic search and local search in memetic algorithms for multiobjective permutation flowshop scheduling. IEEE Transaction on Evolutionary Computation, 7(2), 204-223, 2003.

[Jac 93] L.W. Jacobs, M.J. Brusco. A simulated annealing approach to the cyclic staff scheduling problem. Naval Research Logistics, 40, 69-84, 1993.

[Jac 95] L.W. Jacobs, M.J. Brusco. A local search heuristic for large set covering problem. Naval Research Logistics, 42, 1129-1140, 1995.

[Jia 03] T.Z. Jiang, Q. Cui, G. Shi, S. Ma. Protein Folding simulations of the hydrophobic-hydrophilic model by combining tabu search with genetic algorithms. Journal of Chemical Physics, 119(8), 4592-4596, 2003.

[Joz 04] N. Jozefowiez. Modélisation et résolution approchée de problèmes de tournées multi-objectif. Thèse de doctorat de l'université de Lille, décembre 2004.

[Kap 94] A. Kapsalis, G.D. Smith, V.J. Rayward-Smith. A unified paradigm for parallel genetic algorithms. Evolutionary computing. Ed. T.C. Forgarty. AISB Workshop, 131-149. Leeds, U.K., April 1994.

[Kaw 98] H. Kawamura, M. Yamamoto, T. Mitamura, K. Suzuki, A. Ohuchi. Cooperative search on pheromone communication for vehicle routing problems. IEEE Transactions on Fundamentals, 81, 1089-1096, 1998.

[Kel 88] C.P. Keller, M. Goodchild. The multiobjective vending problem: A generalization of the traveling salesman problem. Environment and Planning B: Planning and Design, 15, 447-460, 1988.

[Kim 95] Y-D. Kim. Minimizing total tardiness in permutation flowshops. European Journal of Operational Research, 33, 541-551. 1995.

[Kno 02] J. Knowles, D. Corne. On metrics for comparing nondominated sets. In Congress on Evolutionary Computation (CEC'2002), Piscataway, New Jersey, 1, 711-716, May 2002.

[Koe 89] Y. Park, C. Koelling. An interactive computerized algorithm for multicriteria vehicle routing problems. Computers and Industrial Engineering, 16, 477-490, 1989.

[Koh 75] W. H. Kohler, K. Steiglitz. Exact, approximate, and guaranteed accuracy algorithms for the flow-shop problem n/2/F/ $\overline{F}$. Journal ACM, 22(1), 106-114, 1975.

[Kök 03] M. Köksalan, A.B. Keha. Using genetic algorithms for single-machine bicriteria scheduling problems. European Journal of Operational Research, 145(3), 543-556, 2003.

[Kon 89] J.C. Konig, D. Trystram. Ordonnancement du graphe deux pas pour le calcul parallèle. Acad. Sci., Paris, Ser. I 309, 8, 569-572, 1989.

[Kra 99] N. Krasnogor, W.E. Hart, J. Smith, D.A. Pelts. Protein structure prediction with evolutionary algorithms. Proceedings of the 1999 International Genetic and Evolutionary Computation Conference (GECCO'99), Orlando, Florida, USA, 488-495, July 1999.

[Kwo 99] Y.K. Kwok, I. Ahmad. Benchmarking and comparison of the task graph scheduling algorithms. Journal of Parallel and Distributed Computing, 59(3), 381-422, 1999.

[Lac 03] P. Lacomme, C. Prins, M. Sevaux. Multiobjective capacitated arc routing problem. In C. M. Fonseca and al., editors, Evolutionary Multi-criterion Optimization, Lecture Notes in Computer Science 2632, Springer-Verlag, 550-564, 2003.

[Lan 01] A. Landrieu. Logistique inverse et collecte des produits techniques en fin de vie, tournées de véhicules avec contraintes. Thèse de doctorat de l'institut National Polytechnique de Gronoble, septembre 2001.

[Lau 01] M. Laumanns, E. Zitzler, L. Thiele. On the effects of archiving elitism and density based selection in evolutionary multi-objective optimization, In Eckart Zitzler and al., editors, Evolutionary Multi-criterion Optimization, Lecture Notes in Computer Science 1993, Springer Verlag, 181-196, 2001.

[Law 66] E.L. Lawler, D.E. Wood. Branch-and-bound methods: A survey. Operations Research 14(4), 699-719. 1966.

[Lee 98] T.R. Lee, J.H. Ueng. A study of vehicle routing problem with load balancing. International Journal of Physical Distribution and Logistics Management, 29, 646-648, 1998.

[Lee 03] L.H. Lee, K.C Tan, K. Ou, Y.H. Chew. Vehicle capacity planning system (VCPS): A case study on vehicle routing problem with time windows. IEEE Transaction on Systems, Man and Cybernetics: Part A (Systems and Humans), 33(2), 169-178, 2003.

[Leg 99] G. Leguizamon, Z. Michalewicz. A new version of Ant System for subset problems. Proceedings of the 1999 Congress on Evolutionary Computation, Washington, 1459-1464, July 1999.

[Len 81] J.K. Lenstra, A.H.G. Rinnooy Kan. Complexity of vehicle routing and scheduling problem. Networks, 11, 221-227, 1981.

[Les 03] N. Lesh, M. Mitzenmacher, S. Whitesides. A complete and effective move set for simplified protein folding. Proceedings of the Seventh Annual International Conference on Computational Molecular Biology, Berlin, Germany, 188-195, April 2003.

[Les 04] L. Lessing, I. Dumitrescu, T. Stützle. A comparison between ACO Algorithms for the set covering problem. In M. Dorigo, L. Gambardella, F. Mondada, T. Stützle, M. Birratari, and C. Blum, editors, ANTS'2004, Fourth International Workshop on Ant Algorithms and Swarm Intelligence, Lecture Notes in Computer Science 3172, 1-12, 2004.

[Lia 01] F. Liang, W.H. Wong. Evolutionary Monte-Carlo Algorithm for protein folding simulations. Journal of Chemical Physics 115(7), 3374-3380, 2001.

[Lo 85] V.M. Lo. Task assignment to minimize completion time. In Proc. of 5th Int. Conf. on Distributed Computing Systems, Denver, Colorado, 239-336, May 1985.

[Lo 88] V.M. Lo. Heuristic algorithms for task assignment in distributed systems. IEEE Transaction on Computer, 37(11), 1384-1397, 1988.

[Lor 94] L.A.N. Lorena, F.B. Lopes. A surrogate heuristic for set covering problems. European Journal of Operational Research, 79, 138-150, 1994.

[Lus 01] I.J. Lustig, J.F. Puget. Program does not equal program: Constraint programming and its relationship to mathematical programming. Interfaces 31(6), 29-53. 2001.

[Lut 94] E. Lutton. P. Martinez. A genetic algorithm of 2D geometric primitives in images. In Pattern Recognition, Conference a Computer Vision and Image Processing. Proceedings of the 12th IAPR International Conference, Jerusalem, 1, 526-528, October 1994.

[Mab 01] M.H. Mabed, M. Rahoual, E.G. Talbi, C. Dhaenens. Algorithmes génétiques multiobjectifs pour les problèmes de Flowshop. Proceedings of MOSIM'01, 848-849, Troyes, avril 2001.

[Man 96] L. Mandow, E. Millan. Goal programming and heuristic search. In R. Caballero, F. Ruiz, R. Stauer, editors, Second Int. Conf. on Multi-objective Programming and Goal Programming MOPG'96, Spain, Springer-Verlag, 48-56, May 1996.

[Mar 98] E. Marchiri, A. Steenbeek. An iterated heuristic algorithm for the set covering problem. Proceedings of WAE'98, Saarbrüken, Germany, 1-3, August 1998.

[Mar 00] E. Marchiri, A. Steenbeek. An evolutionary algorithm for large scale set covering problems with application to airline crew scheduling. Real World Applications of Evolutionary Computing. Lecture Notes in Computer Science 1083, Springer-Verlag, 367-381, 2000.

[Meh 02] V. Mehra, W. Nadler, P. Grassberger. Growth algorithm for lattice heteropolymers at low temperatures. Hsu, H.P., e-print cond-mat/0208042, 2002.

[Mit 93] H. Mitra. P. Ramanathan. A genetic approach for scheduling non pre-emptive tasks with precedence and deadline constraints. In Proceedings of the 26th Hawaii International Conference on System Sciences, 556-564, January 1993.

[Mou 04] M. Mourgaya. The periodic vehicle routing problem: Planning before routing. Thèse de doctorat, de l'université de Bordeaux 1, Juillet 2004.

[Moz 04] http://www.mozart-oz.org

[Mun 93] A. Munier, J.C. Kong. A heuristic for a scheduling problem with communication delays. Rapport de Recherche n° 871, université de Paris Sud, 1993.

[Mur 98] T. Murata, H. Ishibuchi. A multi-objective genetic local search algorithm and its application flow-shop scheduling. IEEE Transaction Systems, Man and Cybernetics, Part C: Application and Reviews, 28(3), 392-403, 1998.

[Naw 83] M. Nawaz, E.E. Jr. Enscore, I. Ham. A heuristic algorithm for the m-machine, n-job flowshop sequencing problem. OMEGA, 11, 91-95, 1983.

[Nem 88] G. Nemhauser, L. Wolsey. Integer and combinatorial optimization. Wiley, 1988.

[Now 96] E. Nowicki, C. Smutnicki. A fast tabu search algorithm for the flow-shop problem. European Journal of Operational Research, 91, 160-175, 1996.

[Now 99] E. Nowicki. The permutation flow-shop with buffers: A tabu search approach. European Journal of Operational Research, 116(1), 205-219, 1999.

[Now 03] E. Nowicki, C. Smutnicki. Some aspects of scatter search in the flow-shop problem. European Journal of Operational Research, 169(2), 654-666, 2006.

[Or 76] I. Or. Traveling salesman-type combinatorial problems and their relation to the logistics of regional llood banking. Ph.D. thesis, North-western University, Evanston, Illinois, December 1976.

[Osm 93] I. H. Osman. Metastrategy simulated annealing and tabu search algorithms for the vehicle routing problem. Annals of Operations Research, 41, 421-451, 1993.

[Pac 03] J. Pacheco, R. Marti. Tabu search for a multi-objective routing problem. Rapport technique TR09-2003, Université de Valence, 2003.

[Pad 82] M.W. Padberg, M.R. Rao. Odd minimum cut-sets and b-matchings; Math of Operations Research 7, 67-80, 1982.

[Pap 82] C.H. Papadimitriou, K. Steiglitz. Combinatorial optimization: Algorithms and complexity. Prentice-Hall, 1982.

[Pap 98] C.H. Papadimitriou, K. Steiglitz. Combinatorial optimization: Algorithms and complexity. Dover Publications, July 1998.

[Paq 03] L. Paquete, T. Stützle. A two-phase local search for the bi-objective traveling salesman problem. In C. M. Fonseca and al., editors, Evolutionary Multi-criterion Optimization, Lecture Notes in Computer Science 2632, 295-310, 2003.

[Par 98] S. Parthasarathy, C. Rajendran. Scheduling to minimize mean tardiness and weighted mean tardiness in flowshop and flowline-based manufacturing cell. Computers and Industrial Engineering, 34(2), 531-546, 1998.

[Paz 89] J.L. Pazat. Outils pour la programmation d'un multiprocesseur à mémoires distribuées. Thèse de doctorat, Université de Bordeaux I, Février 1989.

[Pea 90] J. Pearl. Heuristique : stratégies de recherche intelligente pour la résolution de problèmes par ordinateur. Cépadues Edition, 1990.

[Ped 98] J.P. Pedroson, Niche search: An application in vehicle routing. IEEE International Conference on Evolutionary Computation, Anchorage, Alaska, IEEE publisher, 1, 177-182, May 1998.

[Pic 92] C. Picouleau. Etude de problèmes d'optimisation dans les systèmes distribués. Thèse de doctorat de l'université Paris 7, octobre 1992.

[Por 98] M.C. Portmann, A. Vignier, D. Dardihac, D. Dezalay. Branch and bound crossed with GA to solve hybrid flowshops. European Journal of Operational Research, 107(2), 389-400, 1998.

[Pot 95] J.Y. Potvin, J.M. Rousseau. An exchange heuristic for routing problems with time windows. Journal of the Operational Research Society 46, 1433-1446, 1995.

[Pot 96a] J.Y. Potvin, J.M. Rousseau, B.L. Carcia. The vehicule routing problem with time windows part I : Tabu search. Informs Journal on Computing, 8(2), 165-172, 1996.

[Pot 96b] J.Y. Potvin, S. Bengio. The vehicle routing problem with time windows part II: Genetic search. Informs Journal on Computing, 8, 339-370, 1996.

[Pot 96c] J.Y. Potvin, D. Dubé, C. Robillard. A hybrid approach to vehicle Routing using neural networks and genetic algorithms. Applied Intelligence 6, 241-252, 1996.

[Pri 04] C. Prins. A simple and effective evolutionary algorithm for the vehicle routing problem. Computers and Operations Research, 31, 1985-2002, 2004.

[Pur 91] V. M. Pureza, P. M. França. Vehicle routing problems via tabu search metaheuristic. Technical report CRT-347, Centre de Recherche sur les Transports, Montréal, Canada, 1991.

[Rah 95] M. Rahoual. Parallel branch and bound algorithms for a class of scheduling problem. ETFA'95 INRIA/IEEE, Paris, 2, 405-416, October 1995.

[Rah 97] M. Rahoual. Application of a genetic and hybrid algorithms to the static scheduling of processes on a parallel architecture. 2nd International Conference on Metaheuristics MIC'97, Sophia-Antipolis, France, 137-142, July 1997.

[Rah 98] M. Rahoual, J.C. Konig, Application de méta-heuristiques au problème d'ordonnancement statique de processus sur architectures parallèles. Revue Calculateurs Parallèles, 10(6), 699-725, 1998.

[Rah 01] M. Rahoual, B. Kitoun, M. Mabed, V. Bachelet, F. Benameur. Multicriteria genetic algorithms for the vehicule routing problem with time windows. In 4th Metaheuristic International Conference MIC'2001, Porto, 527-532, July 2001.

[Rah 03a] M. Rahoual, W. Djoukhdjoukh. Métaheuristiques hybrides Pareto pour le problème de tournées de véhicules avec fenêtres horaires. Proreedings of 6th International Conference on Artificial Evolution, Université de Provence, 380-395, October 2003.

[Rah 03b] M. Rahoual, W. Djoukhdjoukh. Utilisation d'algorithmes génétiques pour le problème de routage de véhicules avec fenêtres horaires multiobjectif. Actes du congrès Roadef 2003, Université d'Avignon, 37-38, 37-38, février 2003.

[Rah 04] M. Rahoual, R. Saad. Scheduling with taboo search on parallel architectures. Journal Combinatorial Mathematics and Combinatorial Computing, JCMCC 51, 65-88, 2004.

[Rah 05] M. Rahoual, M.H. Mabed, C. Dhaenens, E.G. Talbi. A comparative study of different optimization techniques for a bi-criteria flow shop problem. Journal Combinatorial Mathematics and Combinatorial Computing JCMCC, accepté en 2005.

[Rah 07] M. Rahoual, R. Saad. Complexity of the Euclidian Vehicle Routing Problem Problem with Time Windows with regard to Approximation. Journal Combinatorial Mathematics and Combinatorial Computing, JCMCC, 2007.

[Raj 95] C. Rajendran. Heuristics for scheduling in flowshop with multiple objectives. European Journal of Operational Research, 82, 540-555, 1995.

[Ray 87] V.J. Rayward-Smith. UET scheduling with unit interprocessor communication delays. Discrete Applied Mathematics, 18, 55-71, 1987.

[Reg 96] C. Rego, C. Roucairol. A parallel tabu search algorithm using ejection chains for the vehicle routing problem. In I. H. Osman and J. P. Kelly, editors, Meta-heuristics: Theory and Applications, Kluwer, Boston, USA, 661-675, 1996.

[Rei 04] M. Reimann, K. Doerner, R. F. Hartl. D-Ants: Savings based Ants divide and conquer the vehicle routing problem. Computers and Operations Research, 31, 563-591, 2004.

[Rib 01] R. Ribeiro, H.R. Lourenço. A multi-objective model for a multi-period distribution management problem. In 4th Metaheuristic International Conference MIC'2001, Porto, 91-102, July 2001.

[Rio 99] R. Z. Rios-Mercado, J. F. Bard. A branch-and-bound algorithm for permutation flow-shops with sequence-dependent setup times. IIE Transactions, 31(8), 721-731, 1999.

[Rob 94] F. Robusté, C.F. Daganzo, R. Souleyrette. Implementing vehicle routing models. Transportation Research B, 24, 263-286, 1990.

[Roc 95] Y. Rochat, E.D. Taillard. Probabilistic diversification and intensification in local search for vehicle routing. Journal of Heuristics, 1, 147-167, 1995.

[Rou 94] C. Roucairol. Algorithmes parallèles : analyse et conception. Hermes 1994.

[Sak 84] M. Sakarovitch. Optimisation combinatoire : méthodes mathématiques et algorithmiques. Hermann, 1984.

[Say 99] S. Sayin, S. Karabati. A bicriteria approach to the two-machine flow-shop scheduling problem. European Journal of Operational Research, 113, 435-449, 1999.

[Sch 94] M. Schwehm. Massively parallel genetic algorithms. Massively Parallel Processing Applications and Development, L. Dekker, W. Smith and J.C. Zuidervaart, editors, Elsevier Science B.V., 505-512, 1994.

[Sch 98] A. Schrijver. Theory of linear and integer programming. John Wiley and Sons. 1998.

[Ses 98] W. Sessomboon, K. Watanabe, T. Irohara, K. Yoshimoto. A study on multi-objective vehicle routing problem considering customer satisfaction with due-time (the creation of Pareto optimal solutions by hybrid genetic algorithm). Transaction of the Japan Society of Mechanical Engineers, 64(617), 370-376, 1998.

[Sha 98] P. Shaw. Using constraint programming and local search methods to solve vehicle routing problems. In Principles and Practice of Constraint Programming - CP98, Lecture Notes in Computer Science, M. Maher and J.F. Puget (eds), 417-431, Springer-Verlag, New York. 1998.

[She 85] C.C. Shen, W.H. Tsai. A graph matching approach to optimal task assignment in distributed computing systems using a minimax criterion. IEEE Transactions on Computers, C-34(3), 197-203, 1985.

[Shm 02] A. Shmygelska, R. Aguirre-Hernandez, H.H. Hoos. An ant colony Optimization algorithm for the 2D HP protein folding problem. In Proc. of ANTS 2002, LNCS 2463, 40-52, 2002.

[Shm 03] A. Shmygelska, H.H. Hoos. An improved ant colony optimization algorithm for the 2D HP protein folding problem. Proc. of the 16th Canadian Conference on Artificial Intelligence (AI'2003), LNCS 2671, 400-417, Springer Verlag, 2003.

[Shm 05] A. Shmygelska, H.H. Hoos. An ant colony optimisation algorithm for the 2D and 3D hydrophobic polar protein folding problem. BMC Bioinformatics, 6-30, 2005.

[Sin 87] J.B. Sinclair. Efficient computation of optimal assignments for distributed tasks. Journal on Parallel and Distributed Computing, 4, 342-362, 1987.

[Siv 98] F. Sivrikaya-Serifoglu, G. Ulusoy. A bicriteria two-machine permutation flowshop problem. European Journal of Operational Research, 107(2), 414-430, 1998.

[Sol 02] M. Solar, V. Parada, R. Urrutia. A parallel genetic algorithm to solve the set-covering problem. Computers and Operations Research, 29(9), 1221-1235, 2002.

[Sol 87] M. Solomon. Algorithms for the vehicle routing and scheduling problems with time window constraints. Operations Research, 35, 254-265, 1987.

[Sol 88] M. Solomon, J. Desrochiers. Time window constrained routing and scheduling Problems. Transportation Science, 22(1), 1-13, 1988.

[Sol 04] http://web.cba.neu.edu/~msolomon

[Sri 94] M. Srinivas, L.M. Patnaik. Genetic algorithms: A survey. IEEE Computer, 27(6), 17-26, 1994.

[Sri 95] N. Srinivas, K. Deb. Multiobjective optimisation using non-dominated sorting in genetic algorithms. Evolutionary Computation, 2(8), 221-248, 1995.

[Sri 96] J. Sridhar, C. Rajendran. Scheduling in flowshop and cellular manufacturing systems with multiple objectives - A genetic algorithmic approach. Production Planning & Control, 7, 374-382, 1996.

[Ste 91] B.S. Stewart, C.C. White. Multiobjective A*. Journal of the ACM, 38(4), 775-814, 1991.

[Stü 98] T. Stützle. An ant approach for the flow-shop problem. In Proceedings of the 6th European Congress on Intelligent Techniques and Soft Computing (EUFIT'98), Aachen, Germany, 3, 1560-1564, September 1998.

[Sut 90] C. Sutcliffe, J. Board. Optimal solution of a vehicle routing problem: Transporting mentally handicapped adults to an adult training centre. Journal of the Operational Research Society, 41, 61-67, 1990.

[Tai 93] E.D. Taillard. Parallel iterative search methods for vehicle routing problem. Networks, 23, 661-673, 1993.

[Tai 93a] E.D. Taillard. Benchmarks for basic scheduling problems. European Journal of Operational Research, 64, 278-285, 1993.

[Tai 94] E. Taillard. Parallel taboo search techniques for the job shop scheduling problem. ORSA Journal of computing, 6(2), 108-117, 1994.

[Tai 97] E. Taillard, P. Badeau, M. Gendreau, F. Guertin, J.Y. Potvin. A tabu search heuristic for the vehicle routing problem with soft time windows. Transportation Science, 31, 170-186, 1997.

[Tai 03] E. Taillard. Statistical test for comparing success rates. The Fifth Metaheuristics International Conference MIC2003. Kyoto, Japan, August 2003.

[Tal 01] E.G. Talbi, M. Rahoual, M.H. Mabed, C. Dhaenens. New genetic approach for multicriteria optimization problems: Application to the flow-shop. Lecture Notes in Computer Science, LNCS 1993, Springer Verlag, 416-428, 2001.

[Tal 91a] E.G. Talbi, P. Bessière. A parallel genetic algorithm for the graph partitioning problem. ACM International Conference on Supercomputing. Cologne, Germany, June 1991.

[Tal 91b] E.G. Talbi, T. Muntean. Méthode de placement statique des processus sur les architectures parallèles. Technique et Science Informatique, 10(5), 355-373, 1991.

[Tal 92] E.G. Talbi. Etude expérimentale d'algorithmes de placement de processus. La Lettre du Transputer et des Calculateurs Distribués, 1-26, 1992.

[Tal 93] E.G. Talbi. Allocation de processus sur les architectures parallèles à mémoire distribuées. Thèse de doctorat de l'institut naional polytechnique de Grenoble, octobre 1993.

[Tal 95] E.G. Talbi. Algorithmes génétiques parallèles : techniques et application. Dans Parallélisme et Applications Irrégulières, Ed. C. Roucairol et al., Hermes, 29-48. 1995.

[Tal 98] E.G. Talbi, Z. Hafidi, J-M. Geib. A parallel adaptive taboo search approach. Journal Parallel Computing, 24(14), 2003-2019, 1998.

[Tal 99] E.G. Talbi. Métaheuristiques pour l'optimisation combinatoire multiobjectifs : Etat de l'art. Rapport interne, Université de sciences et Technologies de Lille, France 1999.

[Tal 02] E.G Talbi. A taxonomy of hybrid metaheuristics. Journal of Heuristcs, 8(5), 541-564, 2002.

[Tan 87] R. Tanese. Parallel genetic algorithms for a hypercube. Proc. of The Second Int. Conf. On Genetic Algorithms, MIT, Cambridge, 177-183, July 1987.

[Tan 01] K.C. Tan, L.H. Lee, K. Ou. Hybrid genetic algorithms in solving vehicle routing problems with time sindows constraints. Asia-Pacific Journal of Operational Research, 18, 121-130, 2001.

[Tan 03a] K.C. Tan, T.H. Lee, Y.H. Chew, L.H. Lee. A hybrid multiobjective evolutionary algorithm for solving truck and trailer vehicle routing problems. In Proceedings of the 2003 Congress on Evolutionary Computation (CEC'2003), Canberra, Australia, IEEE Press, 2134-2141, December 2003.

[Tan 03b] K.C. Tan, T.H. Lee, Y.H. Chew, L.H. Lee. A multiobjective evolutionary algorithm for solving vehicle routing problem with time windows. In IEEE International Conference on Systems, Man and Cybernetics, Washington, 361-366, October 2003.

[Teg 96] J. Teghem. Programmation linéaire. Ellipses-Marketing, 1996.

[Tha 93] S.R. Thangiah. Vehicle routing with time windows using genetic algorithms. Technical Report SRU-CpSc-TR-93-23, Slippery Rock University, Splippery Rock, PA, USA, 1993.

[Tha 94] S.R. Thangiah, I. Osman, T. Sun. Hybrid genetic algorithm, simulated annealing and tabu search methods for vehicle routing problems with time windows. Working Paper UKC/IMS/OR94/4, Institute of Mathematics and Statistics, University of Kent, Canterbury. 1994.

[Tha 95] S.R. Thangiah. An adaptive clustering method using a geometric shape for vehicle routing problems with time windows. In L.J. Eshelman, editor, Proceedings of the Sixth International Conference on Genetic Algorithms, Pittsburgh, Morgan Kaufmann, 536-543, July 1995.

[Tha 95a] S.R. Thangiah. Vehicle routing with time windows using genetic algorithms. In Application Handbook of Genetic Algorithms: New Frontiers, L. Chambers (ed), CRC Press, Boca Raton. 2, 253-277, 1995.

[Tki 00] V. T'kindt, J.C. Billaut. L'ordonnancement Multiobjectif. Presses universitaires de Tours, Décembre 2000.

[Tki 02a] V. T'kindt, J.C. Billaut. Multicriteria scheduling, theory, models and algorithms, Springer, Berlin, 2002.

[Tki 02b] V. T'kindt, N. Monmarché, F. Tercinet, D. Laügt. An ant colony optimization algorithm to solve a 2-machine bicriteria flowshop scheduling problem. European Journal of Operational Research, 142, 250-257, 2002.

[Tot 03] P. Toth, D. Vigo. The granular tabu search and its application to the vehicle routing problem. Informs Journal on Computing, 15, 333-346, 2003.

[Ung 93] R. Unger, J. Moult. A genetic algorithm for three dimensional protein folding simulation. Journal of Molecular Biology, 231(1), 75-81, 1993.

[Vae 98] R.J.M. Vaessens, E.H. Aarts, J.K. Lenstra. A local search template. Compu. Oper. Res. 25, 969-979, 1998.

[Van 96] A. Van Breedam. An analysis of the effect of local improvement operators in genetic algorithms and simulated annealing for the vehicle routing problem. RUCA Working Paper 96/14, Université d'Anvers, Belgique, 1996.

[Van 00] D.A. Van Veldhuizen, G.B. Lamont. Multiobjective evolutionary algorithms: Analysing the state of the art. Evolutionary Computation, 8(2), 125-147, 2000.

[Var 96] T.A. Varvarigou, V. P. Roychowdhury, T. Kailath, E. Lawler. Scheduling in and out forests in the presence of communication delays. IEEE Trans. on Parallel and Distributed Syst., 7(10), 1065-1074, 1996.

[Vas 84] F.J. Vasko, J.R. Wilson. Using a facility location algorithm to solve large set covering problems. Operations Research Letters, 3, 85-90, 1984.

[Vas 87] F.J. Vasko, F.E. Wolf, K. L. Stott. Optimal selection of ingot sizes via set covering problem. Operations Research, 35, 346-353, 1987.

[Vas 02] F.J. Vasko, F.E. Wolf. A heuristic concentration approach for weighted set covering problems. Locator: Electronic Publication of Location Analysis 2, 1-14. 2002.

[Vas 05] F.J. Vasko, P.J. Knolle, D.S. Spiegel. An empirical study of hybrid genetic algorithms for the set covering problem. Journal of Operational Research Society, 1-11, 2005

[Vel 99] D.A.Veldhuizen. Multiobjective evolutionary algorithms: classification, analyses, and new innovations. Dissertation Presented to the Graduate School of Engineering of the Air Force Institute of Technology. 1999.

[Wed 95] D. Wedelin. An algorithm for large 0-1 integer programming with application to airline crew scheduling. Annals of Operational Research, 57, 283-301, 1995.

[Wee 01] H. Wee Kit, A. Chin, A. Lim. Hybrid search algorithm for the vehicle routing Problem with time windows. International Journal on Artificial Intelligence Tools, 10(3), 431-449, 2001.

[Whi 82] D.J. White. The set of efficient solutions for multiple objective shortest path problems. Computers and Operations Research, 9, 101-107, 1982.

[Wil 89] J.A.G. Willard. Vehicle routing using r-optimal tabu search. Master's thesis, The Management School, Imperial College, London, 1989.

[Wu 84] C.L. Wu, T. Feug. Interconnection networks for parallel and distributed processing. IEEE Computer Society Press, Washington, 1984.

[Xu 96] J. Xu, J.P. Kelly. A network flow-based tabu search heuristic for the vehicle routing problem. Transportation Science, 30, 379-393, 1996.

[Xu 03] J. Xu, M. Li, D. Kim, Y. Xu. Protein threading by linear programming. Pacific Symposium in Biocomputing (PSB), Lihue, Hawaii, 264-275, January 2003.

[Yam 02] T. Yamada. A pruning pattern list approach to the permutation flowshop scheduling problem. Essays and Surveys in Metaheuristics, 12(1), 641-651, 2002.

[Yeh 04] W.C. Yeh, A. Allahverdi. A branch-and-bound algorithm for the three-machine flowshop scheduling problem with bicriteria of makespan and total flowtime. International Transactions in Operational Research, 11, 323. 2004.

[Yeh 99] W.C. Yeh. A new branch-and-bound approach for the n/2/flowshop/αF + ßCmax flowshop scheduling problem. Computers and Operations Research, 26(13), 1293-1310, 1999.

[Yin 04] K.C. Ying, C.J. Liao. An ant colony system for permutation flow-shop sequencing. Computers and Operational Research, 31(5), 791-801, 2004.

[Yon 01] Z. Yong, N. Sannomiya. An improvement of genetic algorithms by search space reductions in solving large-scale flowshop problems. EEJ Transactions on Electronics, Information and Systems, 121-C(6), 1010-1016, 2001.

[Yue 93] K. Yue, K.A. Dill. Sequence-structure relationships in proteins and copolymers. Physical Review E, 48(3), 2267-2278, 1993.

[Zda 02] M. Zdansky, J. Pozivil. Combination genetic/tabu search algorithm for hybrid flowshops optimization. In Proceedings of Algorithmy 2002, Conference on Scientific Computing, Vysoke Tatry, Podbanske, 230-236, September 2002.

[Zhe 03] Y. Zhenyu, L. Zhang, K. Lishan, L. Guangming. A new MOEA for multi-objective TSP and its convergence property analysis. In C. M. Fonseca and al., editors, Evolutionary Multi-criterion Optimization, Second International Conference, EMO 2003, Faro, Portugal. Lecture Notes in Computer Science, 2632, Springer-Verlag, 342-354, April 2003.

[Zhu 00] K.Q. Zhu. A New Genetic Algorithm for VRPTW. National University of Singapore, 2000.

[Zit 99] E. Zitzler, L. Thiele. Multiobjective evolutionary Algorithms: A comparative case study and the strength Pareto approach. IEEE Transaction on Evolutionary Computation, 3(4), 257-271. 1999.

[Zog 04] K.G. Zografos, K.N. Androustsopoulos. A heuristic algorithm for solving hazardous materials distribution problems. European Journal of Operational Research, 152, 507-519, 2004.