Université des Sciences et Technologies de Lille

U.F.R. d'I.E.E.A.

Numéro d'ordre : 3633 Année : 2005

MÉMOIRE DE THÈSE

pour obtenir le grade de

DOCTEUR DE L'U.S.T.L.

DISCIPLINE : INFORMATIQUE

présentée et soutenue publiquement,

le 21/06/2005, par

Matthieu BASSEUR

Titre :

Conception d'algorithmes coopératifs pour l'optimisation multi-objectif : Application aux problèmes d'ordonnancement de type Flow-shop.

Jury :

Président :	Philippe MATHIEU	Professeur, USTL Lille 1
Rapporteurs :	Jin-Kao HAO	Professeur, université d'Angers
	Patrick SIARRY	Professeur, université de Paris XII
Examinateurs :	Arnaud FRÉVILLE	Professeur, université de Valenciennes
	Kenneth SORENSEN	Assistant professor, université d'Anvers, Belgique
	Jacques TEGHEM	Professeur, faculté poltechnique de Mons, Belgique
Directeur :	El-Ghazali TALBI	Professeur, USTL Lille 1

Matthieu BASSEUR
Laboratoire d'informatique Fondammentale de Lille
Bâtiment M3, Cité Scientifique
59655 Villeneuve d'ascq CEDEX
basseur@lifl.fr

Remerciements

Je tiens à remercier El-Ghazali Talbi et Franck Seynhaeve pour avoir encadré ma thèse pendant ces années. Je tiens également à remercier toute l'équipe OPAC, qui m'ont de diverses manières aidé à la réalisation de mon travail. Merci également aux différents aux membres du LIFL que j'ai eu le plaisir de cotoyer, pour les enseignements, ou à d'autres occasions.

Je tiens à remercier M. Philippe Mathieu de m'avoir fait l'honneur d'accepter d'être président du jury.

De même, je remercie chaleureusement M. Jin-Kao Hao et M. Patrick Siarry d'avoir accepté d'être les rapporteurs de cette thèse. Merci, pour l'attention qu'ils ont porté à mon travail, ainsi que pour les conseils et améliorations qu'ils m'ont suggéré, et qui m'ont permis de réaliser le document final.

Merci également à M. Arnaud Fréville, M. Kenneth Sörensen et M. Jacques Teghem d'avoir accepté d'être les examinateurs de cette thèse et pour l'intérêt qu'ils lui ont porté.

Je tiens à remercier spécialement Laetitia pour ses précieux conseils et ses relectures. Sans elle, je serais sûrement encore submergé par la multitude de formalités administratives qui m'ont assailli ces derniers mois. Je tiens également à remercier Clarisse pour l'aide qu'elle m'a apporté, ainsi que Nicolas pour avoir partagé mes discussions philosophiques ou existentielles. Je remercie Grégory, un 'ancien' de l'équipe pour m'avoir encouragé à garder l'esprit créatif, par ses problèmes 'scientifico-phylosophico-intuitifs', durant les discussions agrémentant quelques repas bien copieux le midi. Je remercie également Julien et Benjamin, qui m'ont souvent aidé par leurs relectures et leur intérêt pour mon travail.

Enfin je voudrais remercier également tout ceux qui m'ont indirectement permis d'en arriver là. Tout d'abord ma famille, pour la confiance qu'ils m'ont toujours accordé, leur soutient et leurs encouragements. Je remercie également mes amis, qui ont toujours été là pour me changer les idées aux moments les plus difficiles. Parmi ceux-ci, les plus "anciens", Fabrice et Loïc, avec qui j'ai réalisé ma préparation physique pour ces difficiles années de thèse ; les amis du Club Pongiste Lyssois, pour les moments inoubliables, de tennis de table, ou de toute autre sorte.

Enfin, je tiens à remercier tout particulièrement Clémentine pour tout ce qu'elle m'a apporté ces deux dernières années, par sa sérénité, son soutient, mais aussi par sa simple présence auprés de moi, si agréable et réconfortante.

Résumé

De nombreuses techniques ont été mises au point pour la résolution exacte ou approchée des problèmes d'optimisation. Actuellement, un nombre croissant d'approches coopératives entre ces méthodes voient le jour. Nous présentons différents schémas de coopération dans le but de classifier les différentes approches coopératives rencontrées dans la littérature. Puis nous proposons différentes coopérations pour résoudre un problème de flow-shop bi-objectif.

Pour résoudre de manière approchée ce problème d'optimisation, nous proposons en premier lieu un algorithme génétique adaptatif, spécialisé dans l'exploration de l'espace de recherche et l'optimisation du paramétrage. Ensuite, nous proposons de faire coopérer cet algorithme avec des méthodes dédiées à l'intensification de la recherche. Nous proposons une coopération avec un algorithme de type recherche mimétique, puis avec un algorithme de *path relinking*, et avec une méthode exacte bi-objectif.

Mots clés

Optimisation, coopération, multi-objectif, métaheuristique, méthode exacte, flow-shop.

Abstract

Many methods has been established to solve optimization problems exactly or approximatively. Nowadays, we can observe a growing interest about cooperative approaches between these methods. We present different cooperation schemes in order to classify the cooperatives approaches found in the literature. Then, we propose different cooperation mechanisms in order to solve a bi-objective Flow-shop problem.

To solve approximatively this optimization problem, we propose an adaptive genetic algorithm in a first time. This algorithm is specialized to explore of the search space, and to optimize automatically the parameters. Then, we propose cooperation scheme between this algorithm and other methods specialized in the intensification of the search. We propose cooperations with a mimetic algorithm, a path relinking algorithm, and a bi-objective exact method.

Keywords

Optimization, cooperation, multiobjective, metaheuristics, exact method, flow-shop.

Table des matières

Chapitre 1

Introduction

Cette thèse s'inscrit dans le domaine de l'optimisation combinatoire multi-objectif. Elle a été réalisée au sein de l'équipe d'Optimisation PArallèle Coopérative ($OPAC$) du Laboratoire d'Informatique Fondamentale de Lille ($LIFL$), dans le cadre du projet $INRIA$ 'DOLPHIN', sous la direction du professeur E. G. Talbi.

Le but de ce mémoire est de proposer des méthodes coopératives pour la résolution de problèmes d'optimisation combinatoire multi-objectif. Résoudre un problème d'optimisation combinatoire consiste à trouver une solution qui optimise une fonction appelée fonction *objectif*. Pendant longtemps, la plupart des travaux d'optimisation étaient dédiés à l'optimisation d'une seule fonction objectif. Or la plupart des problèmes rencontrés dans l'industrie (mécanique, télécommunications, environnement, transport...) sont de nature multi-objectif. Les problèmes d'optimisation rencontrés en pratique sont en fait rarement mono-objectif. Il y a généralement plusieurs objectifs contradictoires à satisfaire simultanément (coût de revient à maximiser/qualité à maximiser, temps de réalisation à minimiser/charge de travail à minimiser, etc). Pendant longtemps, les méthodes de résolution des problèmes multi-objectif consistaient principalement à les transformer en problèmes mono-objectif. Depuis quelques années, des travaux prenant en compte directement chacun des objectifs ont été réalisés. Pour cela ils utilisent la notion d'optimalité Pareto, définie initialement dans des travaux en économie au $XIX^{ème}$ siècle [Par96]. Contrairement à l'optimisation mono-objectif, la solution d'un problème multi-objectif n'est pas une solution unique, mais un ensemble de solutions, connu comme l'ensemble des solutions *Pareto optimales*. Ces solutions offrent des compromis entre les différents objectifs optimisés, et le choix d'une de ces solutions est alors réalisé par un décideur. Parmi les problèmes d'optimisation classiques de la littérature, les problèmes d'ordonnancement sont particulièrement de nature multi-objectif, tant le nombre d'objectifs que l'on peut envisager est important. Les problèmes d'optimisation multi-objectif sont considérés comme étant particulièrement difficiles.

Les méthodes d'optimisation des problèmes mono-objectif ou multi-objectif sont très variées. Certains problèmes d'optimisation peuvent être résolus par des méthodes exactes simples qui permettent de trouver à coup sûr la (ou les) meilleure(s) solution(s), ou *solution(s) optimale(s)*. Mais la plupart des problèmes étudiés en optimisation appartiennent

à la classe des problèmes NP-difficiles. Cette classe rassemble des problèmes pour lesquels il n'existe pas d'algorithme qui fournisse la solution optimale en un temps d'exécution polynomial en fonction de la taille de la donnée[1]. Il existe une quantité innombrable de problèmes de ce type dans la littérature. Parmi les plus célèbres, on peut citer le problème du voyageur de commerce, le problème du sac à dos. D'une manière générale, la résolution de problèmes multi-objectif est très complexe, et combine parfois plusieurs objectifs, chacun NP-difficile. Ces problèmes sont très rarement résolus de manière optimale, et il est même difficile d'approcher les solutions optimales.

Pour la résolution des problèmes d'optimisation, il existe de nombreuses méthodes génériques de résolution. Ces méthodes se classent en deux catégories bien distinctes. D'une part, les méthodes exactes, cherchant à trouver de manière certaine la solution optimale en considérant l'ensemble des solutions possibles. D'autre part, les méthodes heuristiques, qui se contentent de rechercher une solution "de bonne qualité". Dans le cadre de l'optimisation de problèmes NP-difficiles, les méthodes exactes ne résolvent pas les problèmes en temps polynomial, ce qui limite la taille des problèmes solubles par ce type d'approche. Cette limite est variable selon les problèmes, mais toujours présente. C'est pour la résolution approchée des problèmes de grande taille qu'ont été introduites les méthodes heuristiques. Parfois on utilise également ces méthodes pour des applications temps réel qui ne permettent pas d'utiliser une méthode exacte, qui devient alors trop coûteuse en temps de calcul.

Les métaheuristiques sont des méthodes génériques de résolution approchée des problèmes d'optimisation. Elles permettent, avec une même méthode, d'envisager une résolution approchée de nombreux problèmes d'optimisation différents, en ayant un minimum d'adaptations à réaliser pour chaque problème.

Au fil des années, la puissance de calcul disponible n'a cessé de croître, ce qui a permis de mettre au point des métaheuristiques de plus en plus complexes. Depuis quelques années, un nombre croissant de méthodes d'optimisation de la littérature proposent de faire coopérer plusieurs métaheuristiques entre elles. Actuellement, les coopérations s'effectuent aussi bien entre métaheuristiques qu'entre métaheuristiques et méthodes exactes.

Ces coopérations permettent d'obtenir des méthodes d'optimisation efficaces sur des problèmes de plus en plus difficiles, et en particulier sur les problèmes multi-objectif. L'intérêt de ces approches coopératives est de permettre à différentes méthodes d'optimisation d'allier leur atouts, dans le but d'améliorer les performances globales obtenues par chacune d'elles. Actuellement, poussées par les performances générales de tels algorithmes, un nombre croissant d'études proposent ce type d'approche.

Dans cette thèse, nous proposons des schémas de coopération entre méthodes d'optimisation, dans le cadre particulier de l'optimisation multi-objectif. Lors de l'optimisation d'un problème, deux buts sont contradictoires : l'amélioration des solutions existantes (intensification) et la découverte de nouvelles régions de recherche (diversification). L'idée des coopérations réalisées par la suite est de combiner des méthodes spécialisées dans chacun de ces deux buts. Dans un premier temps, l'objectif principal a été d'établir une méthode initiale efficace, afin de servir de base aux coopérations futures.

[1] Cette propriété des problèmes NP-difficiles, bien que généralement admise, n'a pas été démontrée.

La méthode de base pour les coopérations que nous réalisons dans cette thèse est spécialisée pour la diversification. Pour cela, nous avons donc mis au point un algorithme génétique multi-objectif. En effet, les algorithmes génétiques, ainsi que les autres algorithmes basés sur une population de solutions, permettent de faire évoluer en parallèle plusieurs solutions, en envisageant simultanément différentes régions de l'espace de recherche, ce qui leur permet d'être efficaces en terme de diversification. De plus, les algorithmes à base de population de solutions sont très souvent utilisés pour l'optimisation multi-objectif, les méthodes à solution unique ne permettant pas directement de découvrir un ensemble de solutions optimales.

Nous avons fait coopérer cet algorithme génétique multi-objectif de diverses manières avec des méthodes orientées vers l'intensification de la recherche. En premier lieu, les coopérations ont été réalisées avec d'autres métaheuristiques. Puis, les coopérations avec une méthode exacte multi-objectif ont étés envisagées. Les coopérations établies dans cette thèse ont été réalisées de diverses manières, et sont répertoriées dans l'annexe 1.

L'organisation de cette thèse est la suivante : dans le chapitre 2 nous proposons une taxinomie des méthodes d'optimisation coopératives. Dans un premier temps nous présentons les problèmes d'optimisation en général, puis les différentes classes de méthodes de résolution existantes. Nous proposons ensuite une taxinomie des méthodes d'optimisation coopératives. À partir de cette taxinomie, nous avons classifié les méthodes coopératives de la littérature, en se restreignant aux coopérations entre métaheuristiques et méthodes exactes, les coopérations entre métaheuristiques ayant déjà fait l'objet de classifications.

Dans le chapitre 3, nous présentons, dans un premier temps, les problèmes d'optimisation multi-objectif, ainsi que les différentes approches existantes pour leur résolution. La comparaison des solutions d'un problème multi-objectif n'est pas triviale. Nous présentons donc différentes métriques permettant d'évaluer la qualité d'un résultat, et nous nous proposons d'en extraire deux pertinentes, qui nous permettront de réaliser nos évaluations. Ensuite nous présentons un problème d'ordonnancement bi-objectif de type *Flow-shop* $BOFSP$, qui servira de problème test pour les coopérations proposées par la suite. Nous avons réalisé un état de l'art des approches de résolution des problèmes d'ordonnancement, en particulier les approches coopératives concernant les problèmes de Flow-shop.

Dans le chapitre 4, nous définissons les différents opérateurs d'un algorithme génétique multi-objectif, sélectionnés selon une étude réalisée dans léquipe par Hakim Mabed. Nous avons ensuite établi un Algorithme Génétique Adaptatif (AGA), où nous proposons deux mécanismes favorisant l'adaptivité et la capacité d'exploration des algorithmes génétiques multi-objectif. Le premier mécanisme permet d'utiliser plusieurs opérateurs de mutation simultanément en favorisant les meilleurs d'entre eux. La méthode que nous proposons est spécialement adaptée au cas multi-objectif. Le deuxième mécanisme que nous avons mis au point influe sur le paramétrage du mécanisme de diversification en l'adaptant selon la disposition des solutions Pareto d'un problème multi-objectif. Nous montrons ensuite l'intérêt apporté par nos mécanismes, à l'aide notamment des métriques de performances sélectionnées dans le chapitre 3. Nous comparons notamment l'algorithme génétique adaptatif AGA avec sa version non-adaptative, mais aussi avec un algorithme génétique classique performant de l'optimisation multi-objectif : $NSGA\ II$ [DAPM00]. Les résultats obtenus montrent l'intérêt des mécanismes présentés dans ce chapitre, mais aussi la nécessité

d'utiliser AGA de manière coopérative avec une autre méthode d'optimisation.

Les chapitres suivants sont consacrés à la mise au point de coopérations entre AGA et d'autres méthodes d'optimisation. Les coopérations ont été envisagées avec trois méthodes d'optimisation, qui feront l'objet des trois derniers chapitres de cette thèse :

- PLS, une recherche locale Pareto.
- $MOPR$, algorithme de *path relinking* multi-objectif coopérant avec PLS.
- TPM, méthode exacte bi-objectif en deux phases.

Dans le chapitre 5, nous proposons différentes méthodes coopératives s'appuyant sur deux briques de base : l'algorithme génétique adaptatif présenté dans le chapitre précédent, et une méthode d'intensification de la recherche que nous proposons dans ce chapitre : la recherche locale Pareto PLS. Les différentes coopérations que nous proposons sont AMA, un algorithme coopératif de type *mimétique*. Puis, nous définissons deux méthodes de coopération entre AMA et AGA. La première ($AGA+AMA$) s'exécute de manière séquentielle, et les transitions sont prédéfinies. La deuxième ($AGMA$) effectue des transitions adaptatives durant l'exécution. En dernier lieu, nous proposons des modèles parallèles, basés sur le modèle coopératif de $AGMA$. Nous montrons, par les expérimentations réalisées, l'efficacité de chacune de ces coopérations, chaque apport réalisé améliorant les résultats obtenus.

Dans le chapitre 6, nous proposons $PRMO$, un algorithme de *path relinking* appliqué au cas multi-objectif. Les algorithmes de *path relinking* ont fait leur preuves en optimisation mono-objectif. Dans ce chapitre, nous proposons de les adapter au cas multi-objectif. Nous définissons des mécanismes multi-objectif pour le *path relinking*, ainsi qu'une méthode de "raccordement" entre deux solutions du $BOFSP$. Puis nous réalisons une coopération entre $PRMO$, AGA et PLS, afin d'intensifier et de diversifier la recherche. Les résultats obtenus montrent les capacités de ce type d'approche.

Dans le chapitre 7, après avoir présenté une méthode de résolution exacte multi-objectif appliquée au Flow-shop bi-objectif dans l'équipe dans le cadre de la thèse de Julien Lemesre, nous proposons trois méthodes coopératives entre métaheuristiques et méthodes exactes. La première est une approche de résolution exacte, les deux autres sont heuristiques, l'une explorant des voisinages larges, l'autre s'appuyant sur le partitionnement des solutions. Nous montrons que ces coopérations permettent d'améliorer le temps de résolution pour l'approche exacte, et la qualité des approximations pour les approches heuristiques. Nous discutons ensuite des résultats obtenus par les différentes approches afin de conclure sur l'intérêt de telles coopérations.

Enfin, nous terminons par une conclusion générale, en reprenant les différentes contributions de cette thèse afin de les analyser pour en donner les apports et les limites. Les différentes conclusions nous amèneront à donner quelques perspectives de recherche.

Chapitre 2

Méthodes d'optimisation coopératives

Dans ce chapitre, nous proposons une taxinomie des coopérations entre méthodes d'optimisation. Puis nous classifions les coopérations entre méthodes de résolution exacte et approchée, peu nombreuses dans la littérature. Le contenu de ce chapitre a été soumis pour publication dans la revue ACM Computing Surveys [BJT05].

2.1 Introduction

Les méthodes de résolution des problèmes d'optimisation sont très diverses, et sont destinées à leur résolution soit de manière exacte, soit de manière approchée (ou heuristique). Les méthodes exactes ont l'avantage de fournir une garantie sur le résultat obtenu, qui est prouvé comme étant optimal, mais très souvent on ne peut garantir de trouver la solution optimale en un temps "raisonnable". Les heuristiques, quant à elles, permettent de fournir très rapidement des solutions intéressantes, mais n'offrent pas la garantie de trouver la solution optimale, ni même une solution proche de la solution optimale.

Afin de rendre les algorithmes plus performants et robustes, de nombreux travaux proposent de combiner différentes heuristiques et/ou méthodes exactes. Depuis quelques années, de nombreuses approches coopératives ont vu le jour dans la littérature. Le but de ces approches est de combiner les avantages des différentes méthodes d'optimisation. Ainsi, de nombreuses coopérations entre méthodes heuristiques ont vu le jour, ainsi que plus récemment des coopérations entre méthodes de résolution exactes et heuristiques.

Dans la section 2.2, nous présenterons l'optimisation dans un sens général, ainsi que les différentes méthodes de résolution des problèmes d'optimisation. L'objectif sera de donner quelques définitions, notations et termes utiles par la suite, ainsi que de présenter les différentes classes de résolution des problèmes d'optimisation.

Dans la section 2.3, nous proposerons une taxinomie des méthodes coopératives. Puis, nous classifierons les différentes méthodes coopératives trouvées dans la littérature, en distinguant d'une part les approches de coopération entre méthodes heuristiques (section 2.4), puis celles entre heuristiques et méthodes exactes, moins répandues dans la littérature (section 2.5). Enfin nous conclurons sur la classification réalisée dans la section 2.6.

2.2 Les méthodes d'optimisation

Dans cette section, nous donnons quelques notions de base de l'optimisation. Nous commençons dans un premier temps par définir ce qu'est un problème d'optimisation, puis nous présentons d'autres termes découlant des problèmes d'optimisation. Nous abordons ensuite les différentes méthodes qui ont été mises au point afin de résoudre les problèmes d'optimisation de manière exacte ou approchée.

2.2.1 Problème d'optimisation

Dans ce qui suit, on se limite à l'optimisation mono-objectif. Le cas multi-objectif sera introduit dans le prochain chapitre. Un *problème d'optimisation* consiste à trouver, parmi un ensemble de solutions potentielles, une solution optimale au regard d'un critère donné.

De manière plus formelle, à chaque instance d'un tel problème, est associé un ensemble $\mathcal{S}$ des solutions potentielles et une fonction de coût f, qui associe à chaque solution potentielle $s \in \mathcal{S}$ une valeur numérique $f(s)$. Résoudre l'instance $(\mathcal{S}, f)$ du problème d'optimisation consiste à trouver une solution $s^* \in \mathcal{S}$ qui minimise (ou maximise) la valeur de la fonction

de coût f. Une telle solution est appelée *solution optimale*, ou *optimum global*. Elle n'est pas forcément unique. Voici une définition, pour le cas de la minimisation (un problème de maximisation peut être défini de manière similaire) [PS82] :

Définition 1 *Une instance d'un problème de minimisation est un couple* $(\mathcal{S}, f)$ *où* $\mathcal{S}$ *est un ensemble de solutions potentielles ou configurations, et* f *une fonction* $f : \mathcal{S} \mapsto \mathbb{R}$. *Le but est de trouver* $s^* \in \mathcal{S}$ *tel que* $f(s^*) \leq f(s)$ *pour tout élément* $s \in \mathcal{S}$.

2.2.1.1 Variables de décision

Dans les problèmes d'optimisation, les *variables de décision* sont des variables pour lesquelles des valeurs sont à choisir. Cet ensemble de variables est appelé *vecteur de décision*. Soit un problème d'optimisation avec n variables de décision. Le vecteur de décision $\overrightarrow{x}$ est représenté par :

$$\overrightarrow{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \tag{2.1}$$

Les différentes valeurs possibles prises par les variables de décision $x_i \in \mathbb{R}$ *ou* $\mathbb{N}$ constituent l'ensemble des solutions envisageables.

Notons que l'on peut généralement distinguer deux branches de l'optimisation. Un problème d'optimisation est dit continu, si les domaines de définition des différentes variables de décision sont continues (souvent dans $\mathbb{R}$). Un problème d'optimisation est dit combinatoire si les domaines de définition des variables de décision sont discrets (souvent les variables sont binaires ou dans $\mathbb{N}$).

2.2.1.2 Espace décisionnel/objectif

L'ensemble des $n-uplets$ de valeurs réelles (ou entières, binaires ...) composant le vecteur de décision est un espace de dimension n. L'ensemble des valeurs pouvant être prises par le vecteur de décision constitue l'espace de recherche de la méthode d'optimisation. Deux espaces euclidiens sont considérés en optimisation :

- L'espace *décisionnel*, de dimension n, n étant le nombre de variables de décision.
- L'espace *objectif*, l'ensemble de définition de la (des) fonction(s) objectif(s). Généralement, cet espace est défini dans $\mathbb{R}$ en optimisation mono-objectif, mais dans le cas avec nb_{obj} objectifs, il sera de dimension nb_{obj}, on parlera alors de problème multi-objectif.

Dans le cadre des algorithmes évolutionnaires, faire varier les variables de décision peut être apparenté à faire varier les gènes d'un individu. Dans ce cas, l'espace décisionnel peut s'appeler également espace *génotypique*. L'espace objectif peut également être appelé espace *phénotypique*.

La valeur dans l'espace objectif d'une solution est généralement appelée *coût*, ou *fitness*.

2.2.1.3 Contraintes

Dans la plupart des problèmes d'optimisation, des restrictions sont imposées par les caractéristiques du problème, ou par les données d'une instance de problème. Ces restrictions doivent être satisfaites afin de considérer une solution comme acceptable. Cet ensemble de restrictions, ou *contraintes*, décrit les dépendances entre les variables de décision et les paramètres du problème. On formule usuellement ces contraintes c_i par un ensemble d'inégalités, ou d'égalités de la forme :

$$c_i(\overrightarrow{x}) \geq 0 \qquad i = 1, \ldots, m \tag{2.2}$$

2.2.1.4 Attribut, but, critère, objectif

En optimisation, on rencontre les termes attribut, but, critère ou objectif. Ces différents termes font souvent l'objet de différenciations dans la communauté de la recherche opérationnelle, même si la frontière entre les définitions de ces termes n'est pas toujours nette.

En général, le terme objectif est employé lorsque l'on mesure une notion modélisée qui a une nature quantitative (coût, distance, ...), tandis que le terme critère correspond plus à une notion qualitative (appartenance à une classe, relation entre différents objets, ...). Dans ce manuscrit, le terme objectif sera utilisé indifféremment pour les deux cas.

En recherche opérationnelle, on parle également d'*attribut* pour désigner un aspect, une propriété, ou une caractéristique (ex : rouge, bleu ou noir). Un *but* désigne une cible pouvant potentiellement être atteinte par un objectif ou un critère.

2.2.2 Méthodes de résolution exacte

Nous nous intéressons maintenant aux différentes méthodes de résolution des problèmes d'optimisation. En premier lieu nous présentons les différentes classes de méthodes exactes. L'intérêt des méthodes exactes est d'apporter l'assurance d'obtenir une solution optimale. Pour cela, elles doivent parcourir l'ensemble de l'espace de recherche, ou au moins avoir l'assurance de n'écarter aucune solution ayant le potentiel d'être meilleure que la solution optimale trouvée par l'algorithme.

Les méthodes de résolution les plus répandues sont certainement celles construisant un arbre de recherche afin d'explorer l'ensemble de l'espace de recherche. Ces méthodes sont appelées *Branch & **, '*' pouvant être remplacé par *Bound*, *Cut*, *Price* ou *Cut & Price*) (ces méthodes sont également appelées méthodes de séparation). Elles utilisent des bornes pour éliminer de la recherche des ensembles de solutions ayant une certaine structure. D'autres méthodes sont spécifiques à un problème donné, ou moins générales, comme la programmation dynamique, la méthode du simplexe, et la programmation linéaire en nombre entiers.

2.2.2.1 Branch & Bound

Le *Branch & Bound* ($B\&B$ - séparation et évaluation) est une large classe d'algorithmes d'où sont dérivées le *Branch, Cut & Price*. Il utilise la stratégie *diviser pour régner*, en partitionnant l'espace des solutions en sous-problèmes pour les optimiser chacun individuellement.

L'idée de base est de construire un arbre où les solutions se forment de manière incrémentale lorsqu'on s'enfonce dans l'arbre, les solutions intermédiaires étant appelées *solutions partielles*. Afin de résoudre plus rapidement le problème, il faut définir une borne supérieure et inférieure sur la solution partielle z, et affiner ces deux bornes par la suite jusqu'à les égaliser :

- Construction d'une borne supérieure : pour trouver une borne supérieure, il suffit de donner une solution réalisable pour le problème traité. Si l'objectif est de minimiser, l'optimum du problème ne pourra qu'être inférieur à la valeur de cette solution en ce point.
- Construction d'une borne inférieure : l'idée est de résoudre le problème que l'on obtient à partir du problème partiel initial en abandonnant les contraintes sur les variables. Comme on optimise sur un ensemble réalisable plus large, l'optimum ainsi obtenu ne pourra qu'être inférieur à l'optimum du problème initial.

La racine de l'arbre correspond à une solution partielle vide, les feuilles correspondent à des solutions réalisables (une feuille par solution réalisable). Les autres noeuds correspondent à des solutions partielles. D'une manière classique, on fixe une variable de décision à chaque 'étage' de l'arbre. La profondeur de l'arbre, nécessaire pour atteindre les solutions complètes, correspond au nombre de variables de décision à fixer.

Décrivons maintenant de manière générale la méthode du $B\&B$ appliquée à la résolution des problèmes de programmation linéaire en nombres entiers. Elle consiste en trois étapes principales :

- Résoudre la relaxation linéaire : consiste à la construire une borne supérieure sur la solution partielle z. Soit la solution optimale trouvée est réalisable, dans ce cas on a une borne inférieure du sous-problème traité. Cette borne peut être meilleure que la borne inférieure globale, et ainsi la remplacer. On peut ensuite couper la branche explorée, la solution optimale du sous-problème étant trouvée. Dans le cas où la solution optimale n'est pas réalisable (partielle) et que les bornes trouvées ne permettent pas de "couper", on doit "brancher".
- "Brancher" sur une variable : La seconde idée de la méthode de $B\&B$ est d'opérer une séparation : la région réalisable va être séparée en sous-régions. Cette séparation nécessite le choix d'une variable de séparation. Le choix de cette variable est heuristique. Différents choix sont possibles et de ce choix peut dépendre l'efficacité de la méthode de résolution.
- Coupe : Dans le cas où la borne inférieure de la solution optimale non réalisable du sous-problème traité est inférieure à la borne supérieure globale (i.e. la meilleure solution réalisable trouvée), on peut arrêter la recherche dans cette branche. En effet, on est certain que toute solution réalisable de ce sous-problème ne sera pas meilleure

que l'optimal global courant. On peut également couper lorsque le sous-problème linéaire obtenu est non réalisable.

L'efficacité d'un $B\&B$ dépend en grande partie de la borne inférieure et de la stratégie de parcours de l'arbre (profondeur d'abord, largeur d'abord...). Une borne inférieure la plus précise possible est nécessaire afin de couper rapidement un maximum de branches de l'arbre, et la stratégie d'exploration doit permettre de découvrir rapidement de bonnes solutions réalisables, afin d'avoir une bonne borne supérieure, toujours dans le but de couper un maximum de branches.

2.2.2.2 Branch & Cut

Padberg et Rinaldi |PR91| ont amélioré l'idée du $B\&B$ basé sur la Programmation Linéaire (PL) en décrivant une méthode utilisant des inégalités renforçant la relaxation par PL (inégalités valides pour l'enveloppe convexe des solutions entières). Ils ont appelé cette technique *Branch & Cut* ($B\&C$). Depuis, beaucoup d'implémentations ont été réalisées en utilisant cette approche.

Cette méthode nécessite la transformation du problème d'optimisation en problème de programmation linéaire en nombres entiers. Malheureusement, il est généralement difficile, voire impossible, d'énumérer toutes les inégalités résultant du problème d'optimisation,même si on se restreint à celles décrivant l'enveloppe convexe au niveau de la région optimale.

Une fois l'éventuelle transformation du problème effectuée, on utilise le fait que le problème de programmation linéaire en réels est plus facile à résoudre que celui en entiers. L'algorithme de $B\&C$ utilise cette spécificité afin d'accélérer la recherche.

Le principe de base est le même que pour le $B\&B$, mais au lieu de 'brancher' sur une variable pour descendre dans l'arbre de recherche, on peut rechercher également des 'plans de coupes' qui permettent de restreindre l'espace des solutions réalisables. Au départ de l'algorithme, on se restreint à un petit ensemble d'inégalités du problème que l'on résout en nombres réels. Puis, si la solution optimale obtenue pour le problème en réels est :

- entière : on coupe la branche étudiée, et on met à jour éventuellement la valeur de la solution optimum.
- non entière : on applique un algorithme de séparation et des heuristiques pour extraire un ensemble d'inégalités violées. Si cet ensemble est vide, une solution entière avec ce coût est recherchée, sinon il faut mettre à jour l'ensemble des inégalités et continuer l'algorithme. Le but est d'exclure de manière 'intelligente' la solution optimale non entière de l'espace de recherche, à l'aide de plans de coupes.
- Si le problème résolu ne possède pas de solution réalisable, on coupe la branche.

Les méthodes de génération de plans de coupes peuvent être réalisées de manières diverses comme par combinaison linéaire des inégalités du problème de PL |Gom63|, en se servant de l'algorithme du simplexe défini par Dantzig, ou de manière spécifique au problème (comme par exemple pour le problème du sac à dos |CJP83|).

Quand la génération de plan de coupe échoue, et que le sous-problème ne peut être

élagué selon la valeur d'un objectif, il faut alors "brancher". L'opération de branchement est réalisée en spécifiant un ensemble d'hyperplans qui divisent le sous-problème courant de telle sorte que la solution optimale réelle courante ne soit pas réalisable dans aucun des sous-problèmes de relaxation PL générés (par exemple, si x_i, pour la solution optimale réelle, vaut 2.7, on pourra "brancher" en ajoutant la contrainte $x \leq 2$ pour une branche et $x \geq 3$ pour l'autre).

2.2.2.3 Branch & Price

Les inégalités d'un programme linéaire en nombres entiers peuvent être vues comme une matrice, chaque colonne correspondant à une variable, et chaque ligne correspondant à une inégalité. Comme pour les plans de coupe, les colonnes de la matrice correspondant aux inégalités prises en compte peuvent être définies implicitement si le nombre de variables est grand. Si une colonne n'est pas présente dans la matrice courante, alors la variable correspondante prend implicitement la valeur zéro. Le processus de génération dynamique des variables est appelé couramment *pricing*. Cette méthode peut être vue comme une génération de plans de coupe pour le dual du PL courant. De plus, les algorithmes de $B\&B$ résolvant des PLs en générant les variables dynamiquement quand cela est nécessaire sont appelés algorithmes de *Branch & Price* ($B\&P$). Une étude générale de ce type de méthode est proposée par Barnhart *et al.* [BJN+98].

2.2.2.4 Branch, Cut & Price

Lorsque les variables et les plans de coupe sont engendrés dynamiquement durant l'algorithme de $B\&B$, on appelle cette technique *Branch, Cut & Price* ($B\&C\&P$)[1]. Il existe une certaine symétrie entre la génération de coupes et de variables. Cependant, même si les méthodes de $B\&C$ et de $B\&P$ sont proches, combiner les deux approches requiert la mise en place de méthodes sophistiquées.

2.2.2.5 Autres méthodes exactes

En dehors des méthodes d'énumération décrites précédemment, de nombreuses approches exactes différentes existent, et utilisent les spécificités du problème traité pour résoudre le problème d'optimisation. C'est le cas de l'algorithme du simplexe, de la programmation dynamique, de l'algorithme A^* (voir le livre de Cormen *et al.* [CLR90]). D'autres approches exactes sont entièrement spécifiques au problème traité, comme l'algorithme de Johnson pour l'ordonnancement [Joh54]. Elle sont généralement appliquées aux problèmes peu difficiles, même si l'on en rencontre aussi pour la résolution de problèmes *NP-difficiles*.

[1] Un aperçu des différents travaux répertoriés utilisant cette approche est consultable à l'adresse : `http ://branchandcut.org/reference.html`.

2.2.3 Méthodes de résolution approchée

Les méthodes de résolution approchée sont généralement utilisées là où les méthodes exactes échouent. En effet, une résolution exacte nécessite de parcourir l'ensemble de l'espace de recherche, ce qui devient irréalisable lorsque l'on veut résoudre de gros problèmes. Dans ce cas, une exécution partielle de l'algorithme exact permet rarement d'obtenir une solution de bonne qualité. Des méthodes de résolution approchée ont été mise au point afin de procurer rapidement des solutions de bonne qualité mais non optimales. Ces méthodes sont regroupées en différentes classes génériques de résolution, qu'on appelle métaheuristiques.

Parmi ces méthodes de résolution approchée, on distingue deux classes : celles se basant sur une solution unique et celles faisant évoluer une population de solutions.

2.2.3.1 Métaheuristiques à solution unique

Les métaheuristiques à solution unique sont basées sur un algorithme de recherche de voisinage qui commence avec une solution initiale, puis l'améliore pas à pas en choisissant une nouvelle solution dans son voisinage. Les plus classiques sont la *recherche locale*, le *recuit simulé* et la *recherche tabou*.

2.2.3.1.1 Recherche locale

Les métaheuristiques dites de recherche locale, ou de descente, sont très anciennes et doivent leur succès à leur rapidité et leur simplicité [Pap76, PS82]. A chaque pas de la recherche, ces méthodes progressent vers une solution voisine de meilleure qualité.

La descente s'arrête quand tous les voisins candidats sont moins bons que la solution courante, c'est-à-dire lorsqu'un optimum local est atteint. On distingue différents types de descentes en fonction de la stratégie de génération de la solution de départ et du parcours du voisinage : la descente déterministe (en général on choisit la solution apportant la plus grande amélioration), la descente stochastique (choix aléatoire parmi les solutions améliorant la solution initiale) et la descente vers le premier meilleur.

2.2.3.1.2 Recuit simulé

Le recuit simulé est une technique d'optimisation de type *Monte-Carlo* généralisé à laquelle on introduit un paramètre de température qui sera ajusté pendant la recherche [KGV83]. La méthode du recuit simulé, appliquée aux problèmes d'optimisation (son origine est inspirée des méthodes de simulation de Metropolis en mécanique statistique - années 50), considère une solution initiale et recherche dans son voisinage une autre solution de façon aléatoire. L'originalité de cette méthode est qu'il est possible de se diriger vers une solution voisine de moins bonne qualité avec une probabilité non nulle. Ceci permet d'échapper aux optima locaux. Au début de l'algorithme, un paramètre T, apparenté à la température, est déterminé et décroît tout au long de l'algorithme pour tendre vers 0. De la valeur de ce paramètre va dépendre la probabilité d'acceptation des solutions détériorantes (plus la température T est élevée, plus cette probabilité sera forte).

2.2.3.1.3 Recherche Tabou

La recherche Tabou (*Tabu Search*) a été introduite par Glover [Glo86] et a montré ses performances sur de nombreux problèmes d'optimisation. Les idées de base de la recherche Tabou se retrouvent également dans le travail de Hansen [Han86]. Elle peut n'avoir aucun caractère stochastique et utilise la notion de mémoire pour éviter de tomber dans un optimum local.

Le principe de l'algorithme est le suivant : à chaque itération, le voisinage (complet ou sous-ensemble de voisinage) de la solution courante est examiné et la meilleure solution est sélectionnée, même si elle est moins bonne que la solution courante. Afin d'éviter le phénomène de cyclage entre plusieurs solutions, la méthode interdit les mouvements aboutissant à une solution récemment visitée. Pour cela, une liste taboue contenant les attributs des dernières solutions visitées est tenue à jour. Chaque nouvelle solution considérée enlève de cette liste la solution la plus anciennement visitée. Ainsi, la recherche de la solution courante suivante se fait dans le voisinage de la solution courante actuelle sans considérer les solutions appartenant à la liste taboue. Dans certains cas, on mémorise les mouvements réalisés plutôt que les solutions complètes, essentiellement dans le but de mémoriser le moins d'informations possibles.

En appliquant ce principe, la méthode autorise de remonter vers des solutions qui semblent moins intéressantes mais qui ont peut être un meilleur voisinage. Afin d'augmenter l'efficacité de ces méthodes, on l'utilise souvent de manière complémentaire avec des mécanismes de diversification, d'intensification ou d'aspiration.

2.2.3.2 Métaheuristiques à base de population

Les méthodes d'optimisation à base de population de solutions améliorent, au fur et à mesure des itérations, une population de solutions. L'intérêt de ces méthodes est d'utiliser la population comme facteur de diversité. De plus, elles sont très adaptées et très largement utilisées pour l'optimisation multi-objectif.

2.2.3.2.1 Algorithmes génétiques

Les algorithmes génétiques sont des algorithmes dits "évolutionnaires". La classe des algorithmes évolutionnaires regroupe les algorithmes génétiques, les stratégies d'évolution, la programmation évolutionnaire et la programmation génétique [MS96].

Les algorithmes évolutionnaires sont des techniques d'optimisation itérative et stochastique, inspirées par des concepts issus de la théorie de l'évolution de Darwin [Dar59]. Un algorithme évolutionnaire simule un processus d'évolution sur une population d'individus, dans le but de les faire évoluer vers les optimums globaux du problème d'optimisation considéré. Au cours du cycle de simulation, trois opérateurs principaux interviennent : recombinaison, mutation et sélection. La recombinaison et la mutation créent de nouvelles solutions candidates, tandis que la sélection élimine les candidats les moins prometteurs.

Les algorithmes génétiques sont des méthodes basées sur les principes de la génétique. Holland exposa les principes de ces algorithmes pour permettre aux ordinateurs "d'imiter les êtres vivants en évoluant" pour rechercher la solution à un problème [Hol75]. Il expliqua

d'abord comment ajouter de l'intelligence dans un programme informatique avec les croisements (échange du matériel génétique) et la mutation (source de la diversité génétique).

Comme pour l'ensemble des algorithmes évolutionnaires, un certain nombre d'opérateurs génétiques sont nécessaires pour la définition d'un algorithme génétique. Un opérateur d'initialisation pour définir la population d'individus qui va permettre de "lancer" l'évolution ; un opérateur de sélection qui tend à augmenter l'importance des bonnes solutions par rapport aux mauvaises (principe du Darwinisme) ; un opérateur de croisement ainsi que de mutation, afin de permettre la génération de nouveaux individus. Ces opérateurs constituent la base d'un algorithme génétique, et peuvent être complétés par d'autres opérateurs génétiques, notamment par un opérateur de diversification, introduit dans le but de maintenir une certaine diversité dans la population, qui tend à s'uniformiser avec le temps.

2.2.3.2.2 Programmation génétique

Récemment, un nouveau paradigme génétique fait son apparition, la programmation génétique. L'idée sous-jacente à cette approche est qu'il n'est pas nécessaire d'utiliser des codages linéaires (les simples vecteurs de l'algorithme génétique traditionnel) pour soumettre des programmes à une évolution génétique. Le principal promoteur de ce paradigme est Koza [Koz92].

La programmation génétique consiste à faire évoluer une population constituée d'un grand nombre de programmes. La plupart des algorithmes de programmation génétique travaillent avec une population modélisée sous forme d'arbres. Le principe est le même qu'en algorithmique génétique classique, mais les opérateurs de croisement et de mutation sont différents. En effet, ils travaillent directement sur la structure d'arbre du programme.

2.2.3.2.3 Algorithmes mimétiques

Les algorithmes mimétiques ressemblent très fortement aux *AGs* 'classiques'. Ils ont été mis au point afin d'accélérer la convergence des *AGs*, réputée pour être lente. L'idée principale est d'inclure un mécanisme de recherche locale dans le déroulement de l'*AG*, en remplaçant l'un de ses opérateurs génétiques. Pour cette raison, certains les classifient comme étant des algorithmes génétiques hybrides, faisant coopérer recherche locale et algorithme évolutionnaire. Ces algorithmes sont aussi parfois appelés 'recherche locale génétique' (*genetic local search*).

Le premier a avoir employé le terme d'algorithme mimétique est Moscato en 1989 [Mos89] pour un algorithme optimisant le problème du voyageur de commerce. Une bonne introduction aux algorithmes mimétiques, ainsi que quelques applications, peuvent être trouvées dans [CDG99].

2.2.3.2.4 Colonies de fourmis

Le système de fourmis (*Ants System - AS*) est une méthode d'optimisation proposée par Dorigo [DMC91, DMC96, Dor92], basée sur des observations faites sur les fourmis. En se déplaçant du nid à la source de nourriture et vice-versa (ce qui, dans un premier temps, se fait essentiellement de façon aléatoire), les fourmis déposent au passage sur le sol une

substance odorante appelée phéromone, ce qui a pour effet de créer une piste chimique. Les fourmis peuvent sentir ces phéromones qui ont un rôle de marqueurs de chemin : quand les fourmis choisissent leur chemin, elles ont tendance à choisir la piste qui porte la plus forte concentration de phéromones, ce qui peut donc s'apparenter à la recherche de bonnes solutions à un problème d'optimisation.

Le système de fourmis a été employé avec succès sur de nombreux problèmes (voyageur de commerce, affectation quadratique, ...) mais les auteurs ont remarqué que l'AS n'a pas un comportement très exploratoire ce qui a conduit les auteurs à utiliser des hybridations du système de fourmis avec des mécanismes de recherche locale.

2.2.3.2.5 Essaim de particules

Les algorithmes d'optimisation par essaim de particules (*Particle Swarm Optimization - PSO*) ont été introduits en 1995 par Kennedy et Eberhart comme une alternative aux algorithmes génétiques standards [KE95]. Ces algorithmes s'inspirent des essaims d'insectes (ou des bancs de poisson,s ou des nuées d'oiseaux) et de leurs mouvements coordonnés. En effet, tout comme ces animaux se déplacent en groupe pour trouver de la nourriture ou éviter les prédateurs, les algorithmes à essaim de particules recherchent des solutions pour un problème d'optimisation.

Dans ces algorithmes, une particule décide de son prochain mouvement en fonction de sa propre expérience, qui est dans ce cas la mémoire de la meilleure position qu'elle a rencontrée, et en fonction de son meilleur voisin (au sens de l'essaim). Les nouvelles vitesse et direction de la particule sont définies en fonction de trois tendances : la propension à suivre son propre chemin, sa tendance à revenir vers sa meilleure position atteinte et sa tendance à aller vers son meilleur voisin.

2.2.3.2.6 Recherche par dispersion - *Path relinking*

La recherche par dispersion (*Scatter Search*) est une méthode d'optimisation relativement ancienne décrite par Glover [Glo77]. Elle contraste avec d'autres algorithmes évolutionnaires, tels que les algorithmes génétiques, par l'utilisation de conceptions stratégiques là où d'autres approches utilisent des principes stochastiques. Ces algorithmes, de type évolutionnaire, consistent à exploiter la connaissance du problème pour créer de nouvelles solutions issues de combinaisons de solutions existantes. Ces solutions sont stockées dans un ensemble de référence, relativement petit en comparaison aux autres approches évolutionnaires.

La recherche par dispersion combine plusieurs mécanismes :
- Une méthode de génération de solutions avec mécanisme pour assurer leur diversité.
- Une méthode d'amélioration d'une solution en une ou plusieurs solutions améliorées.
- Un ensemble de solutions de référence comprenant les solutions rencontrées les plus intéressantes, en terme de qualité ou de diversité.
- Une méthode de génération de sous ensembles de l'ensemble de référence, afin de créer des solutions combinées.
- Une méthode de combinaison de solutions, permettant de créer de nouvelles solutions formées par combinaison de solutions sélectionnées par la méthode précédente.

Les algorithmes de *path relinking* sont une variante des algorithmes de recherche par dispersion, et Glover est également à l'origine de cette méthode [Glo96]. Cette méthode est une variante de la recherche par dispersion. Elle consiste à explorer les chemins permettant la connexion de solutions de qualité entre elles. Les solutions rencontrées sur les chemins explorés sont des combinaisons de la solution 'initiale' et de la solution 'but', et les meilleures d'entre elles sont sélectionnées pour participer à un mécanisme d'intensification comme, par exemple, une recherche locale.

2.2.3.3 Autres méthodes heuristiques

Nous avons énuméré les grandes familles de métaheuristiques pour l'optimisation approchée, basées sur un seul individu, ou sur une population entière d'individus. Évidemment, d'autres heuristiques plus spécifiques, ou des classes de métaheuristiques moins répandues, existent dans la littérature. Nous pouvons citer notamment les métaheuristiques à base de population appelées systèmes immunitaires (*immune systems*), basées sur des rapprochements entre le fonctionnement du système immunitaire des êtres vivants et l'optimisation. Des applications informatiques du système immunitaire pour l'optimisation peuvent être trouvées dans un livre de Corne *et al.* [CDG99]. D'autres approches sont basées sur la logique floue (systèmes flous - *fuzzy systems*). Elle peuvent être appliquées à divers types de métaheuristiques. Dans [DPST03], un recensement des différentes classes de métaheuristiques est effectué.

Enfin, on trouve de nombreuses heuristiques qui 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 ce type d'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 un problème de voyageur de commerce, par exemple, un algorithme glouton basique serait de visiter à chaque itération la ville non visitée la plus proche.

Actuellement, une proportion grandissante des approches heuristiques sont de type coopératif. Elles consistent à coupler les particularités de chacune d'elles. La puissance de calcul actuelle des ordinateurs permet d'exploiter efficacement plusieurs heuristiques coopératives simultanément ou séquentiellement. Ce type d'approche est l'objet de la section suivante, où nous nous proposons de les classifier.

2.3 Schéma global des méthodes coopératives

Durant ces dernières années, de nombreuses recherches sur la coopération[2] de métaheuristiques ont été menées dans le domaine de l'optimisation combinatoire. Actuellement, les meilleurs résultats obtenus sont issus de ce type d'approche, en particulier sur les problèmes réels. Les coopérations étaient à l'origine essentiellement réalisées entre différentes métaheuristiques. Mais, actuellement, de plus en plus de travaux proposent la coopération entre métaheuristiques et méthodes exactes comme le montre la figure 2.1. Ces travaux

[2] On utilisera indifféremment les termes coopération ou hybridation.

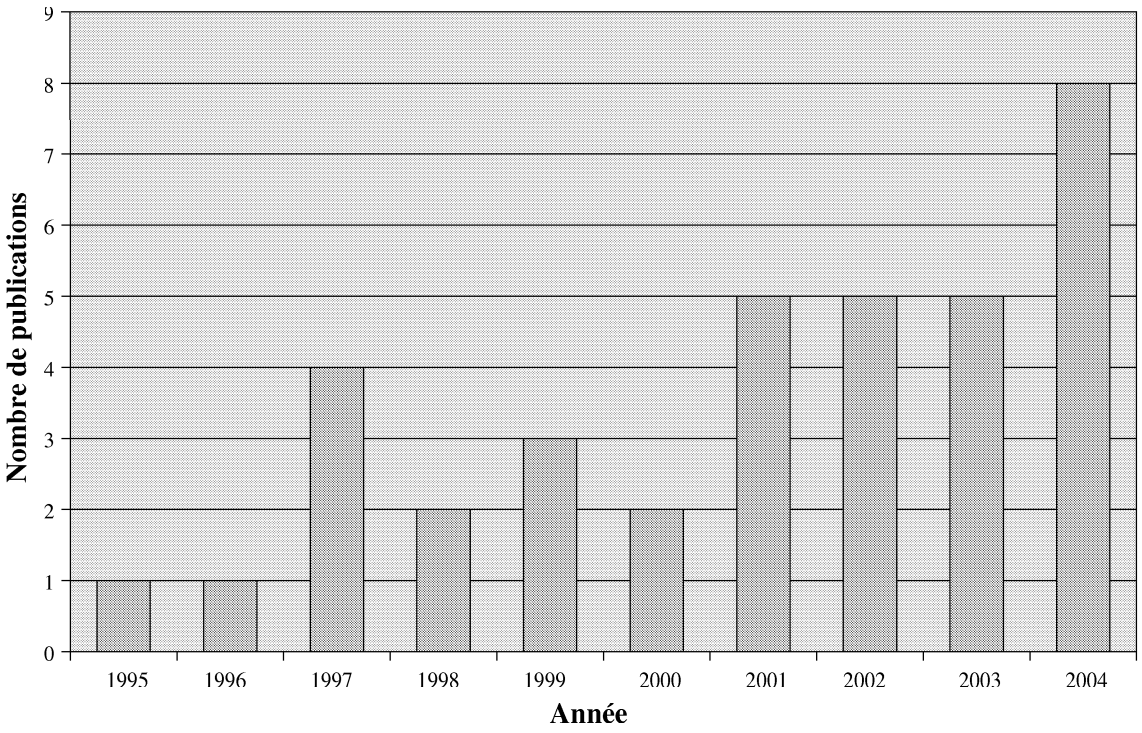


FIG. 2.1 – Évolution du nombre de publications portant sur les coopérations entre méthodes exactes et métaheuristiques.

sont généralement efficaces, car les deux méthodes coopérant ont alors des particularités bien différentes, et associent leurs avantages afin d'obtenir de bons résultats.

On trouve, parmi ces méthodes une grande diversité sur les techniques mises en oeuvre afin de faire coopérer les méthodes d'optimisation. Dans cette section, une taxinomie de ces méthodes est proposée afin de les classifier, en distinguant les approches hybrides "classiques" faisant coopérer des métaheuristiques de celles faisant intervenir une méthode exacte.

Parmi les taxinomies existantes dans divers domaines, on trouve des modèles de classification hiérarchique et des modèles de classification à plat. La taxinomie que nous proposons, ici, est composée des deux modèles. La figure 2.2 représente la structure de cette taxinomie. Nous avons repris et complété la structure générale de la taxinomie des métaheuristiques hybrides réalisée antérieurement [Tal02].

Nous ferons lors de cette classification une distinction entre les approches coopératives de plusieurs métaheuristiques et les coopérations entre méthodes exactes et métaheuristiques. Pour cela, nous détaillerons des exemples pour les différentes classes de coopérations pour les deux types d'approches. On parlera dans la suite de méthode d'optimisation pour tout ce qui désigne une méthode exacte ou métaheuristique.

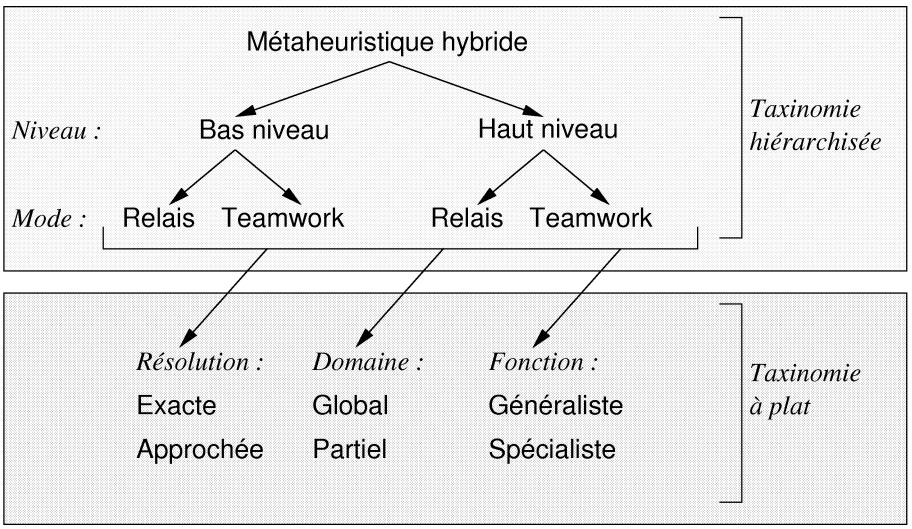


FIG. 2.2 – Structure de notre taxinomie des méthodes coopératives.

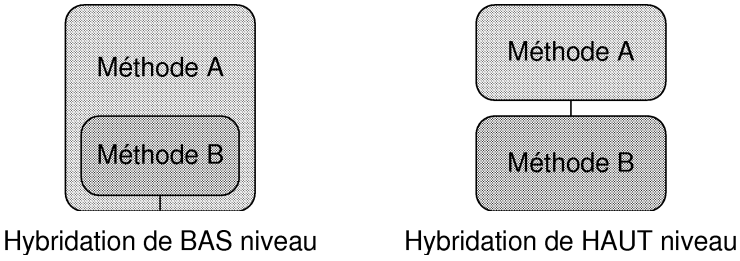


FIG. 2.3 – Le niveau d'hybridation. Dans l'hybridation de bas niveau, la méthode est modifiée, ici la méthode B devient un élément fonctionnel de A. Pour l'hybridation de haut niveau, les méthodes A et B ne sont pas modifiées.

2.3.1 Classification hiérarchique

Dans cette section, nous commençons par présenter les deux propriétés discriminantes que sont le niveau et le mode de coopération. Puis nous détaillerons les différentes classes hiérarchiques découlant de ces propriétés, en donnant notamment des exemples détaillés.

2.3.1.1 Les propriétés discriminantes

2.3.1.1.1 Niveau de coopération

La première étape de la classification consiste à distinguer les méthodes suivant leur niveau de coopération. On distingue les hybridations de bas niveau et les hybridations de haut niveau. La figure 2.3 illustre la notion de niveau pour la coopération de deux méthodes d'optimisation (parfois plus de deux méthodes d'optimisation sont hybridées).

L'hybridation de bas niveau modifie les éléments fonctionnels qui constituent une méthode d'optimisation. Dans cette classe de coopération, une fonction interne d'une méthode d'optimisation est remplacée par une (autre) méthode d'optimisation.

Au contraire, 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 méthodes d'optimisation.

2.3.1.1.2 Mode de coopération

Le mode de coopération distingue les coopérations en mode relais des hybrides en mode *teamwork*.

En mode relais, les méthodes d'optimisation coopératives opèrent les unes après les autres dans un ordre prédéterminé. Hormis la première, chaque méthode impliquée dans la coopération reçoit en entrée le résultat produit par la précédente, à la manière d'un pipeline.

La classe des recherches hybrides en mode *teamwork* intègre les modèles d'optimisation coopérative où plusieurs agents coopèrent en parallèle. Chaque méthode hybridée (agent) conduit une recherche dans un espace de solutions donné. Très souvent, la coopération en mode *teamwork* fait intervenir au moins une méthode d'optimisation à base de population de solutions.

2.3.1.2 Les classes hiérarchiques

Selon la classification hiérarchique, nous obtenons, à partir des deux propriétés discriminantes que sont le niveau et le mode de coopération, quatre classes détaillées ci-dessous. Nous aurions pu développer le classement hiérarchique en fonction d'autres discriminants (ceux de la classification à plat) mais nous préférons l'arrêter là, pour éviter d'augmenter le nombre de classes. Nous proposons pour chaque classe des exemples de coopération trouvés dans la littérature, pour la coopération de métaheuristiques et pour la coopération entre métaheuristiques et méthodes exactes (que nous appellerons par ailleurs respectivement coopérations méta/méta et méta/exacte).

2.3.1.2.1 La classe LRH - *Low-level Relay Hybrid*

La classe de coopération de bas niveau en mode relais (**LRH**) regroupe les coopérations constituées d'une méthode d'optimisation à solution unique dans laquelle est insérée une (autre) méthode d'optimisation. Les métaheuristiques utilisant une population de solutions ne sont pas vraiment adaptées à ce type de coopération en mode relais (bien que cela reste possible, si celle-ci remplace un mécanisme d'une méthode d'optimisation à solution unique).

Coopération méta/méta de type LRH

Martin *et al.* ont réalisé une coopération de type **LRH** qui combine un recuit simulé et une méthode de descente déterministe pour résoudre le problème du Voyageur de Commerce [MOF92]. Le principe général de leur algorithme hybride est de limiter la recherche

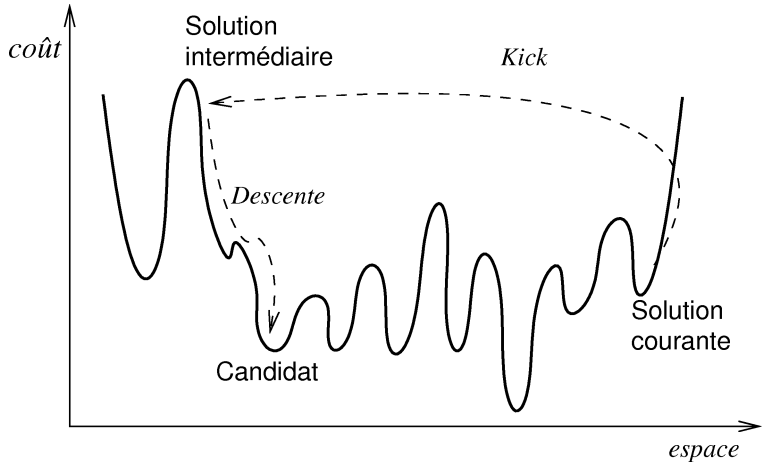


FIG. 2.4 – Un exemple d'hybridation **LRH** dans lequel un recuit simulé intègre une recherche locale. La figure donne une représentation schématique de la fonction coût (ligne continue) et montre les étapes de la génération d'un nouveau candidat du recuit simulé (flèches en pointillé).

du recuit simulé à l'espace des optima locaux par l'intégration de la méthode de descente au recuit simulé. La coopération se déroule comme suit (voir Fig. 2.4) : on part d'une solution courante qui est un optimum local, on la perturbe jusqu'à obtenir une solution intermédiaire suffisamment éloignée dans l'espace de recherche. On effectue ainsi un saut dans l'espace de recherche que les auteurs appellent le *kick*. Puis, cette solution intermédiaire est optimisée par une méthode de descente jusqu'à l'obtention d'un optimum local qui devient un candidat du recuit simulé. Enfin, si le candidat est rejeté (suivant un ou plusieurs critère(s) de qualité), on revient à la solution courante ; s'il est accepté, il devient la nouvelle solution courante. On se retrouve ainsi avec une solution courante qui est un optimum local qui diffère du précédent de façon significative. Ce principe est connu sous le nom d'algorithme du kangourou (*kangaroo algorithm* [**?**, Fle93]

Ainsi, cette métaheuristique hybride permet de sauter par dessus un grand nombre de barrières de coût en un seul pas. Cependant, cette technique d'hybridation peut facilement conduire à une recherche qui cycle si on ne prend pas garde au choix du *kick* et de la méthode de descente. En effet, on conçoit aisément que si le *kick* et la descente sont basés sur le même opérateur de voisinage, alors la descente risque de retourner à la solution courante. Pour cela, il est intéressant d'utiliser deux notions de voisinage différentes pour cet algorithme hybride.

Coopération méta/exacte de type LRH

Ce type de coopération est plus facilement envisageable en ayant une méthode heuristique au service d'une méthode exacte. Un exemple de ce type de coopération est proposé par Augerat *et al.* [ABB+98]. Dans cette étude, un algorithme de *Branch & Cut* est proposé pour résoudre un problème de tournées de véhicules avec contraintes de capacité. La gé-

nération de plans de coupe est une partie cruciale déterminant l'efficacité des algorithmes de $B\&C$. Les auteurs partent de la remarque que les inégalités du programme linéaire traitant des contraintes de capacités sont celles qui sont à l'origine des meilleures plans de coupes, ce qui permet les meilleures améliorations de la borne lorsque ces contraintes sont ajoutées à la formulation du programme linéaire relaxé. Trois approches heuristiques, une constructive, une gloutonne, et une Tabou, sont envisagées pour extraire un ensemble pertinent de contraintes de capacité violées du problème relaxé.

2.3.1.2.2 La classe LTH - *Low-level Teamwork Hybrid*

La classe de coopération de bas niveau en mode *teamwork* (**LTH**) regroupe les coopérations constituées d'une méthode d'optimisation à population de solutions pour laquelle un opérateur agissant sur les solutions de manière individuelle ou globale est remplacé par une méthode d'optimisation.

Coopération méta/méta de type LTH

Dans la conception d'une métaheuristique, on recherche l'équilibre entre deux objectifs contradictoires : l'exploration et l'exploitation (également appelés diversification et intensification). L'exploration est nécessaire, car elle apporte une information (partielle) de l'espace de recherche dans sa globalité qui fournit une approximation de la qualité de l'optimum global. L'exploitation n'est pas moins importante, car une recherche autour de la solution courante tend à produire une meilleure solution.

D'une manière générale, les métaheuristiques basées sur une population de solutions sont plutôt performantes pour l'exploration de l'espace de recherche alors qu'elles le sont beaucoup moins pour l'exploitation des solutions qu'elles ont trouvées. On retrouve souvent les propriétés inverses pour les métaheuristiques à solution unique. C'est pourquoi, beaucoup d'heuristiques à base de population de solutions classiques ont été **LTH**-hybridées avec des méthodes à solution unique qui sont performantes en terme d'exploitation. Ainsi, la coopération **LTH** réalise l'équilibre exploration/exploitation en hybridant deux algorithmes complémentaires, l'un optimisant localement, l'autre optimisant d'une manière globale. Les algorithmes mimétiques présentés dans la section précédente sont des algorithmes coopératifs **LTH** (coopération AG/méthode de recherche locale). Leur efficacité les a rendus très populaires et ils sont devenus maintenant standards.

On peut par exemple utiliser un algorithme génétique pour l'optimisation globale et ajouter à ses opérateurs standards une méthode d'optimisation à solution unique. Ainsi, au lieu d'utiliser un opérateur aveugle, c'est-à-dire qui ne tient pas compte de la *fitness* de l'individu, on prend un opérateur, dit *lamarckien*, qui est une heuristique locale, qui optimise l'individu et restitue à l'AG un individu amélioré. Les opérateurs standards des AGs les plus souvent remplacés ou complétés sont la mutation et la recombinaison. Les heuristiques utilisées en remplacement de l'opérateur de mutation sont généralement des méthodes de descente [SG87], des recherches taboues [FF94], ou des recuits simulés [BHS89]. Les opérateurs de recombinaison classiques fonctionnent sans tenir compte d'aucune spécificité du problème. Certains opérateurs de recombinaison intègrent des caractéristiques propres au problème à résoudre par l'intermédiaire d'algorithmes gloutons [Gre87]. La figure 2.5 montre un exemple de ce type classique de coopération, où les opérateurs de mutation et

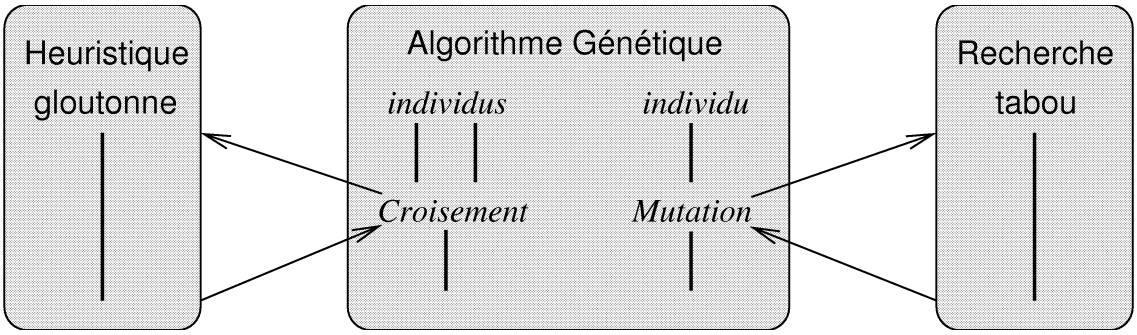


FIG. 2.5 – Hybridation de bas niveau en mode *teamwork*. Ici, une recherche tabou et une heuristique gloutonne remplacent les opérateurs standard de l'algorithme génétique, respectivement la mutation et la recombinaison.

de recombinaison ont été remplacés par des heuristiques.

D'autre métaheuristiques ont été hybridées de la même manière, comme les systèmes à colonies de fourmis [TG97], la programmation génétique [OO95], et les recherches par dispersion [CMMT97].

Coopération méta/exacte de type LTH

Dans cette classe de coopération, diverses approches sont proposées. D'une part, des approches de type "algorithmes mimétiques" peuvent être rencontrées dans la littérature. Ainsi Cotta *et al.* proposent une approche basée sur la coopération entre AGs et $B\&B$ [CANT95]. L'algorithme exact est incorporé dans l'AG pour servir d'opérateur de recombinaison. L'opérateur résultant explore de manière intelligente les solutions potentielles issues de la recombinaison de deux parents, afin d'en extraire la plus intéressante grâce à un algorithme de $B\&B$.

Jahuira *et al.* proposent différentes coopérations entre AGs et méthodes exactes appliquées au problème du voyageur de commerce [Jah02, JCV03]. La coopération est introduite dans les opérateurs génétiques. L'opérateur de recombinaison est remplacé par un algorithme de $B\&B$, d'arbre de recouvrement minimal et de méthode de *backtracking* (méthode utilisée pour reconstruire le chemin emprunté pour la résolution du problème en partant de la fin). De plus, dans cette étude, la population initiale est engendrée à l'aide d'un arbre de recouvrement minimal engendré de manière exacte.

Des algorithmes hybrides différents de ceux de type "algorithme mimétique" ont été proposés, comme l'approche de type **LTH**, proposée par Kostikas & Fragakis [KF04]. Un algorithme de Programmation Génétique (PG) est incorporé dans un algorithme de $B\&B$ basé sur un problème de Programmation Linéaire Mixte (PLM). L'architecture coopérative employée ici utilise la PG pour générer l'expression dédiée à la sélection des noeuds à explorer. L'algorithme de PG exploite les caractéristiques du PLM étudié pour faire évoluer la méthode de sélection de noeud. La méthode générée remplace alors l'opérateur de sélection par défaut de l'algorithme de $B\&B$ pour le reste de l'exécution.

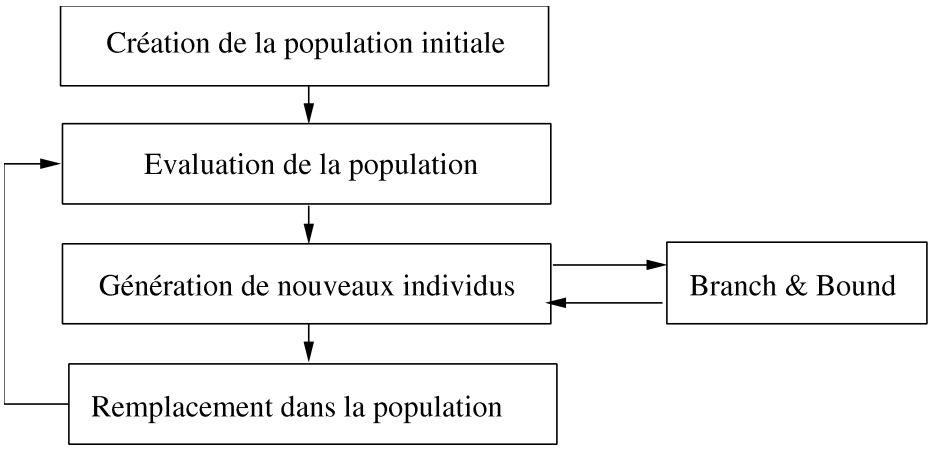


FIG. 2.6 – Schéma général de l'*AG* hybride de Cotta *et al.* [CANT95] : un exemple de coopération de type **LTH**.

Les algorithmes de recherche sur large voisinage (*large neighborhood search*) sont des algorithmes de descente qui utilisent des voisinages larges pour améliorer l'efficacité de l'algorithme. La méthode d'exploration du voisinage peut être aussi bien heuristique qu'exacte. Un tour d'horizon de ces méthodes est proposé dans [AEOP02]. Les algorithmes de recherche sur large voisinage, faisant intervenir des méthodes exactes pour l'exploration du voisinage, sont des algorithmes typiquement **LTH**. La méthode exacte intervient durant la recherche pour trouver la (les) meilleure(s) solution(s) dans un sous espace de l'espace de recherche global. Plusieurs études utilisent ce principe dont celle de Bent et Van Hentenryck pour le problème de voyageur de commerce asymétrique [BV04], ou celle de Shaw pour le problème de routage de véhicules [Sha98].

2.3.1.2.3 La classe HRH - *High-level Relay Hybrid*

Dans les coopérations de haut niveau en mode relais, les heuristiques et/ou méthodes exactes hybridées conservent leur intégrité. Les méthodes coopératives de type **HRH** sont exécutées en séquence. Par exemple, on sait que les algorithmes à base de population de solutions ne parviennent pas à ajuster finement les solutions proches des bons optima. Mais, à l'inverse, leur force réside dans la capacité de trouver rapidement des régions de bonne qualité, même pour des espaces de recherche très vastes ou très complexes. Une fois ces régions repérées, il peut être intéressant de poursuivre la recherche en affinant les solutions performantes qui s'y trouvent ; pour cela, on applique une recherche itérative à solution unique sur la (les) solution(s) trouvées par la première méthode. De même, on utilise également très souvent des heuristiques à convergence très rapide (souvent de type gloutonnes) afin de fournir de bonnes solutions initiales à un algorithme d'optimisation. Ces deux mécanismes peuvent être exécutées en séquence (voir Fig. 2.7).

Coopération méta/méta de type HRH

De nombreux auteurs ont conçu des coopérations de type **HRH** à partir de méthodes

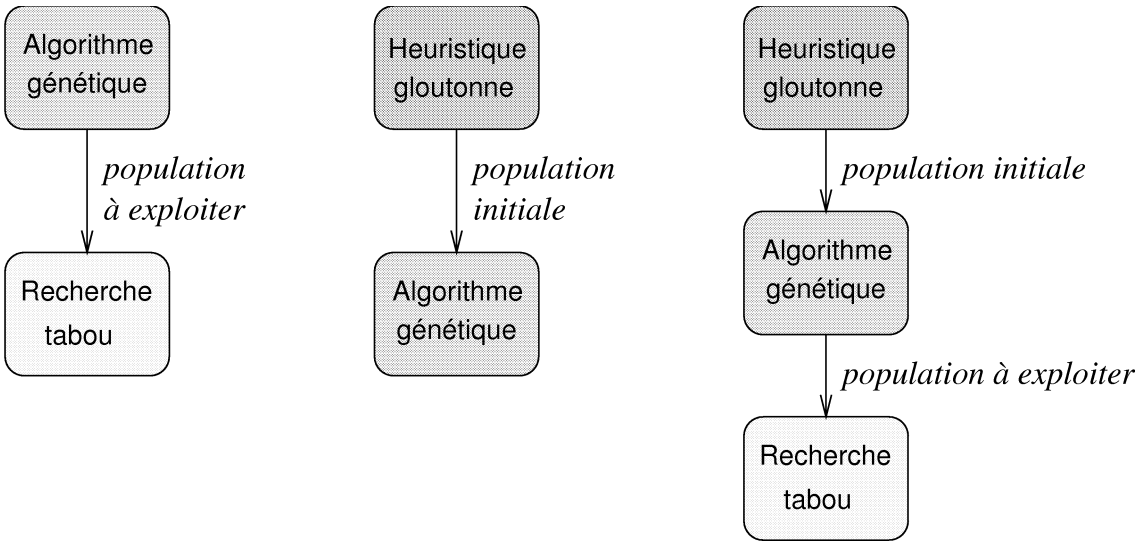


FIG. 2.7 – Hybridation de haut niveau en mode relais. La figure montre 3 instances (classiques dans la littérature) du schéma d'hybridation **HRH**. Ce modèle ne limite pas le nombre d'heuristiques exécutées séquentiellement.

d'optimisation diverses. Dans |TMS94, MG95|, les auteurs ont utilisé, respectivement, un recuit simulé et une recherche tabou pour optimiser la population issue d'un AG. Dans |Nis94|, une méthode de descente améliore les solutions fournies par une heuristique Stratégie Évolutionnaire (*Evolution Strategy* - SE). Dans [LKH91], l'algorithme proposé commence par un recuit simulé ; il est suivi d'un algorithme génétique pour parfaire les solutions trouvées. Une étude expérimentale sur la résolution du Problème de Partitionnement de Graphes (*Graph Partitioning Problem*) avec une coopération **HRH**, impliquant un AG suivi d'une recherche Tabou, montre que l'approche coopérative fournit des solutions meilleures que l'algorithme AG ou la recherche Tabou seuls |TMS94|.

Coopération méta/exacte de type HRH

Cette classe de coopération inclut notamment les algorithmes exacts utilisant des heuristiques afin d'obtenir des solutions initiales, ou simplement des bornes. Kleipeis *et al.* proposent une coopération entre un α Branch & Bound ($\alpha B\&B$) et un AG hybridé avec un algorithme de *Conformational Space Annealing* (CSA) pour la prédiction de structure de protéine |KPF03|. L'algorithme de $\alpha B\&B$ est un algorithme général d'optimisation basé sur le $B\&B$, qui peut être appliqué à une large classe de problèmes d'optimisation non-linéaires ayant deux fonctions différentiables. L'algorithme de CSA est une méthode stochastique employant différents éléments des AGs et du recuit simulé. Dans cet algorithme, les auteurs alternent des exécutions de $\alpha B\&B$ et de CSA. Cet algorithme hybride a été parallélisé dans un modèle maître-esclave (figure 2.8).

Dans |BV04|, Bent et Van Hentenryck proposent un algorithme hybride en deux phases pour le problème de tournées de véhicules avec fenêtres de temps. Dans un premier temps

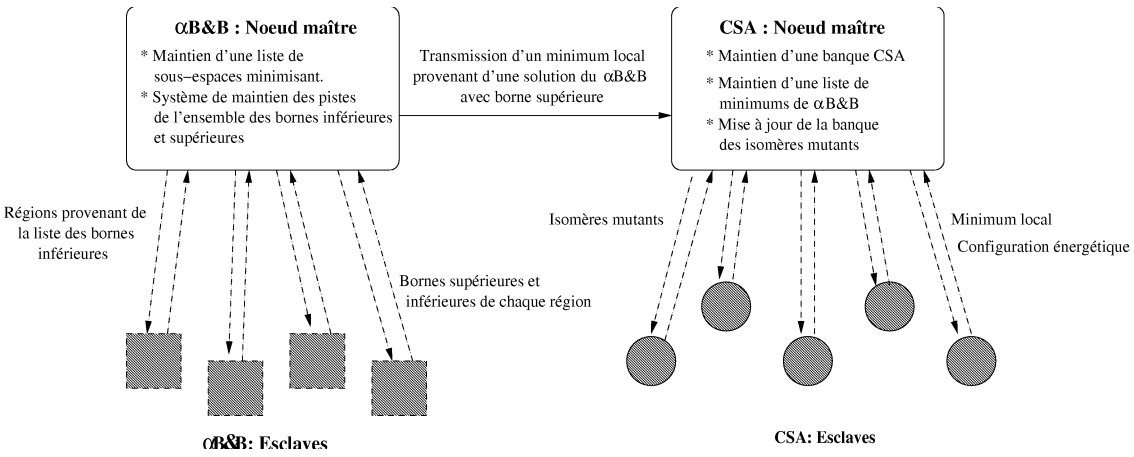


FIG. 2.8 – Exemple de coopération **HRH** pour la prédiction de structure de protéines [KPF03].

l'algorithme minimise le nombre de véhicules par un algorithme de recuit simulé. Puis le coût de chaque tournée est minimisé par une recherche exacte sur un large voisinage pouvant relocaliser un grand nombre de clients.

Un autre exemple de ce type de coopération est proposé par Portmann *et al.* [PVDD98] pour résoudre un problème de Flow-shop hybride.

2.3.1.2.4 La classe HTH - *High-level Teamwork Hybrid*

Dans les hybridations de haut niveau *teamwork*, la structure interne des métaheuristiques hybridées n'est pas modifiée. Ces dernières sont exécutées simultanément et coopèrent pour résoudre le problème.

Un exemple d'hybridation **HTH** est le modèle des algorithmes génétiques en îles. Dans ce modèle, la population est partagée en petites sous-populations géographiquement éloignées. Un *AG* fait évoluer chaque sous-population et les individus peuvent migrer d'une sous-population à une autre suivant différents paramètres (nombre, fréquence et stratégie des remplacements...). La coopération peut également se réaliser entre différentes méthodes d'optimisations, et même entre une méthode exacte et une métaheuristique.

Coopération méta/méta de type HTH

Tanese [Tan87] propose un *AG* en îles où les sous-populations sont disposées dans un hypercube de dimension 4. Des migrations ont lieu périodiquement entre les sous-populations voisines sur l'hypercube (voir Fig. 2.9). Les individus à migrer sont choisis de façon probabiliste parmi les meilleurs individus des sous-populations ; puis, ils migrent et remplacent les individus les moins bons dans la population qui les accueille.

Cohoon *et al.* proposent un hybride **HTH** inspiré de la théorie des « équilibres ponctuels » [CHMR87]. Leur méthode est expérimentée sur une topologie en grille pour un problème de placement linéaire. Ils observent que l'algorithme avec migration est plus

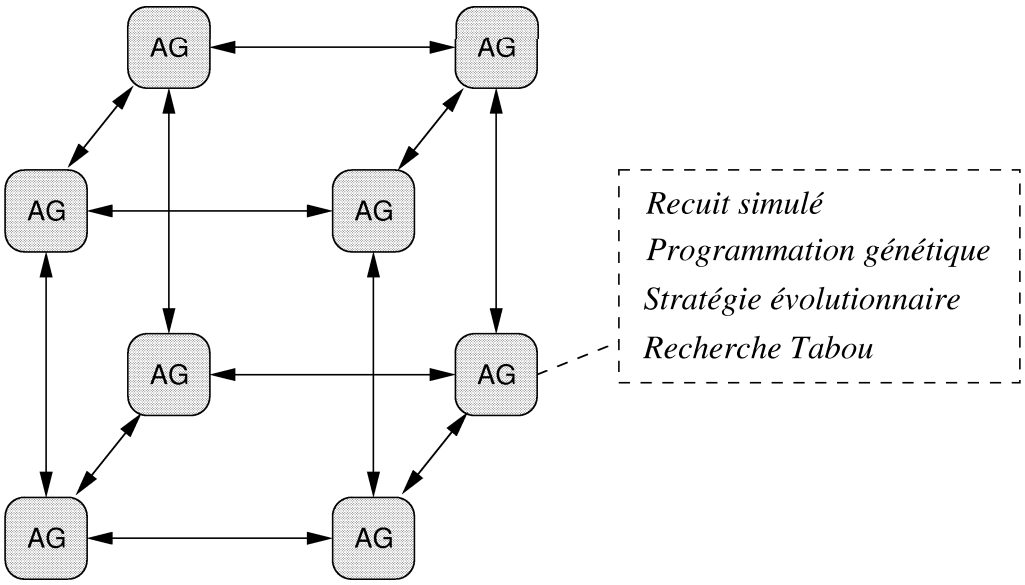


FIG. 2.9 – Le modèle d'algorithme génétique en îles est un exemple de coopération de haut niveau *teamwork*. Ce modèle a été aussi employé avec différentes topologies pour des recuits simulés, des programmations génétiques, des stratégies évolutionnaires et des recherches tabous.

performant que celui sans migration et l'*AG* standard.

L'hybridation **HTH** a été appliquée également à d'autres métaheuristiques, telles que les recuits simulés |FBT95|, les programmations génétiques |KA95|, les stratégies d'évolution |VBSK90| et les recherches tabous |FBTV94|.

Coopération méta/exacte de type HTH

Ce type de coopération est assez difficile à mettre en œuvre entre une méthode exacte et une heuristique. En effet, les deux approches ne permettent pas de résoudre les mêmes types de problèmes. Donc, lors d'une coopération de ce type, il paraît indispensable que les deux approches traitent des parties différentes du problème, tout en étant indépendantes l'une envers l'autre.

Dans |CDP02|, Chabrier *et al.* proposent une coopération entre recherche locale et algorithme de génération de colonnes pour le problème de tournées de véhicules. Les deux méthodes d'optimisation coopérent en parallèle en se procurant mutellement diverses informations, comme la montre le schéma de la figure 2.10.

2.3.2 Classification à plat

Les critères discriminants retenus pour la classification à plat des coopérations entre métaheuristiques proposée par Talbi *et al.* |Tal02| sont : l'homogénéité des méthodes hybridées, leurs domaines d'application et la nature (généraliste ou spécialiste) de leurs fonc-

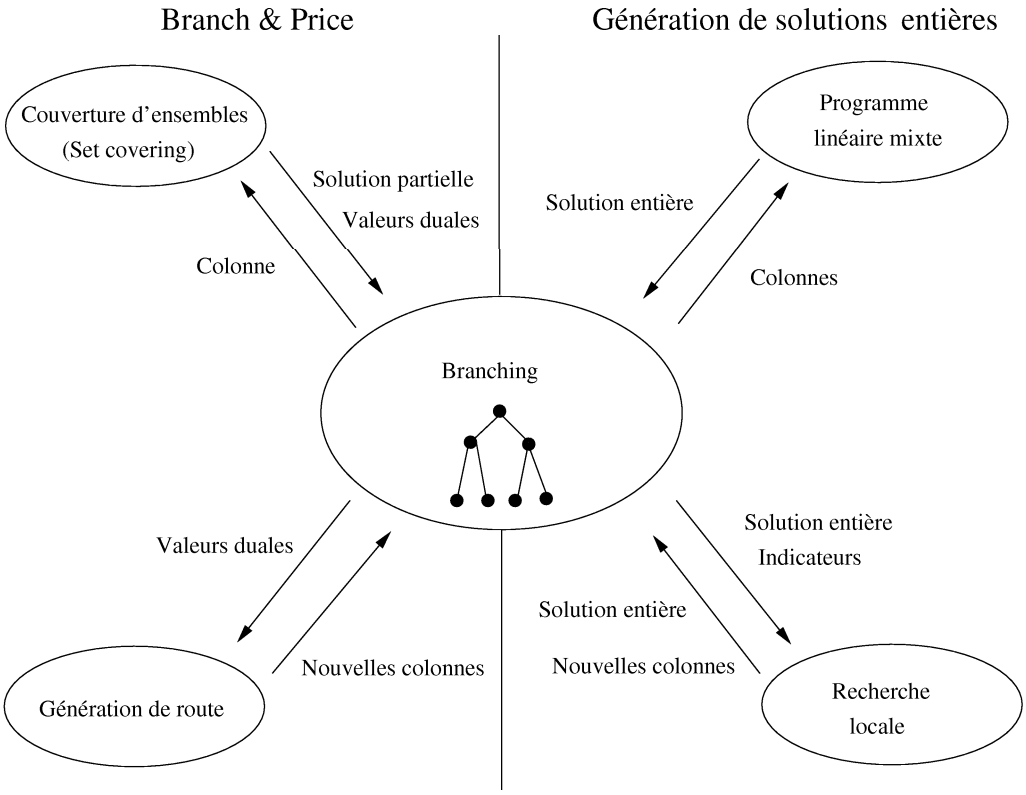


FIG. 2.10 – Exemple de coopération **HTH** pour le problème de tournées de véhicules [CDP02].

tions. Ici nous retiendrons les mêmes discriminants, tout en supprimant celui concernant l'homogénéité des méthodes d'optimisation, qui indiquait si les méthodes d'optimisation hybridées étaient les mêmes ou non. D'une part, pour les coopérations de type méta/méta, seules celles de type parallèle sont concernées. D'autre part, ce type de coopération n'est pas envisageable pour les coopérations de type méta/exacte. De plus, il est intéressant de savoir si l'approche coopérative dans sa globalité est une approche exacte ou heuristique. Nous avons donc ajouté un discriminant *résolution*, qui indique si l'approche générale de la méthode coopérative est exacte ou approchée.

2.3.2.1 Approche de résolution

Ce discriminant concerne essentiellement les coopérations de type méta/exacte. En effet, les coopérations méta/méta sont toutes des méthodes de résolution approchées.

Les approches coopératives exactes utilisent les heuristiques afin d'accélérer l'énumération des solutions en trouvant de bonnes bornes, en offrant des solutions initiales, en définissant des plans de coupes prometteurs...

Par exemple, dans l'approche proposée par Chabrier *et al.* [CDP02], une recherche locale intervient dans un algorithme de *Branch & Cut* afin de générer des nouvelles colonnes permettant de définir des plans de coupes pertinents. Malgré l'utilisation d'une heuristique dans l'approche coopérative, l'approche globale reste exacte.

Dans [BCK01], Burke *et al.* décrivent une méthode coopérative méta/exacte approchée. Dans cette étude, la méthode exacte est incluse dans un algorithme de recherche locale pour explorer de manière exacte les régions prometteuses de l'espace de recherche ; l'approche globale est donc heuristique.

2.3.2.2 Domaine d'application

Le domaine d'application des heuristiques coopératives permet de différencier deux classes de coopération : les coopérations globales et les coopérations partielles.

On appelle coopération globale une coopération pour laquelle toutes les méthodes hybridées sont appliquées à la totalité de l'espace de recherche. L'objectif de ce type de coopération est d'explorer l'espace de recherche plus intensément. Par exemple, une coopération méta/méta de type **HTH** globale entre plusieurs recherches Tabou est présentée dans [CTG97]. Chaque recherche Tabou fait un certain nombre d'itérations, puis diffuse sa meilleure solution ; alors, la meilleure solution de toutes les recherches Tabou devient leur nouvelle solution initiale. Dans le cadre de la coopération méta/exacte, Portmann *et al.* proposent une coopération où une heuristique calcule des solutions initiales dans le but d'offrir de bonnes bornes pour le lancement de la méthode exacte [PVDD98]. Les deux méthodes coopératives travaillent donc sur le même espace de recherche.

Pour la coopération partielle, le problème à résoudre est décomposé en sous-problèmes, chacun ayant un espace de recherche propre. Ainsi, chaque métaheuristique et/ou méthode exacte de la coopération résout un sous-problème dans son propre espace de recherche. Cependant, il est généralement difficile de découper un problème en sous-problèmes indé-

pendants sans lever quelques contraintes. Ainsi, pour que la coopération partielle produise une solution réalisable, les méthodes hybridées doivent communiquer pour tenter de respecter les contraintes levées par la décomposition. Par exemple, ce type de coopération a été appliqué à un recuit simulé et à une recherche tabou pour le problème de tournées de véhicules [Tai93b].

Une part importante des coopérations de type méta/exacte est partielle étant donné que l'espace exploré est trop large pour la méthode exacte seule. Mais des coopérations globales sont réalisées en vue d'accélérer une méthode exacte qui est déjà "praticable" sur les problèmes étudiés.

Pour le problème d'affectation quadratique, la méthode *Mimausa*, mise en oeuvre par Mautor et Michelon [MM97], construit à chaque itération un sous-problème au problème initial qu'il résout par $B\&B$ (coopération partielle). De même, Palpant *et al.* proposent une coopération partielle pour la résolution du problème d'ordonnancement de projet avec contraintes de ressources (*Resource-Constrained Project Scheduling Problem*). Dans cette étude, un sous-problème est généré à chaque itération de recherche locale, puis résolu de manière exacte. Cette solution générée sert alors de point de départ pour la construction du reste de la solution.

T'kindt *et al.* proposent de résoudre un problème de Flow-shop bi-objectif à 2 machines par une approche méta/exacte [TMTL02]. Les deux objectifs sont le *makespan* (C_{max}), et la somme des dates de complétions ($\sum C_i$), à résoudre dans un ordre lexicographique (C_{max}, puis $\sum C_i$). Le premier objectif est soluble en temps polynomial par l'algorithme de Johnson [Joh54]. Un algorithme de colonies de fourmis optimise le second objectif en ne considérant que les solutions optimales pour le *makespan*. Ici, l'espace exploré par la métaheuristique est plus restreint que celui exploré par la méthode exacte.

Une autre approche partielle est proposée par Maniezzo, qui propose de résoudre le problème d'affectation quadratique par un algorithme coopératif **LRH** également à base de colonies de fourmis [Man99] . Il propose d'utiliser une borne inférieure du problème afin de rendre plus efficace cet algorithme. La borne inférieure est calculée en résolvant de manière exacte le problème linéaire en nombre réels associé (calcul de bornes pour le *Branch & Cut & Price*). D'une manière générale, beaucoup d'approches transforment le problème initial afin de pouvoir appliquer une méthode exacte et de fournir des informations à la métaheuristique. On considère dans ce cas que ces approches sont de type partiel.

Les coopérations méta/exacte globales sont généralement des approches exactes. Le cas le plus couramment rencontré dans la littérature est celui où la métaheuristique permet de fournir des bornes et/ou des solutions initiales pour débuter un algorithme de $B\&B$, par exemple [PVDD98].

Nous n'avons pas trouvé de référence de coopération proposant une approche exacte de type partielle. Cela paraît difficilement imaginable puisque, pour résoudre de manière exacte un problème, la méthode exacte doit opérer dans tout l'espace de recherche. Il faut donc imaginer un algorithme où la méthode approchée travaille sur un espace restreint de l'espace de recherche, cas peu courant, surtout si l'approche de résolution doit être exacte.

2.3.2.3 Uniformité du problème traité

On différencie également deux types de coopérations selon que les méthodes d'optimisation traitent le même problème d'optimisation ou non. Pour les coopérations généralistes, toutes les heuristiques (et/ou méthodes exactes) hybridées traitent le même problème d'optimisation. À l'inverse des coopérations généralistes, les coopérations spécialistes combinent des méthodes qui s'attaquent à des problèmes différents.

La plupart des approches coopératives trouvées dans la littérature sont généralistes. Cependant on peut trouver quelques approches spécialistes.

Pour les coopérations méta/méta, un cas d'hybridation **HRH** spécialiste est l'utilisation d'une heuristique pour améliorer une autre heuristique. Le principe est qu'une heuristique optimise le paramétrage de l'autre. Quelques études ont considéré ce principe : un AG pour optimiser un RS et des méthodes de bruitage [Kru93], un AG pour des colonies de fourmis [AAC95] et un AG pour un AG [SM90].

Pour les coopérations méta/exactes, nous avons vu dans la partie précédente, que l'algorithme de T'kindt *et al.* [TMTL02] proposait une coopération de type partielle, la métaheuristique n'étant appliquée que sur les solutions optimales pour le premier objectif du problème, trouvées par la méthode exacte. Dans cette étude, la méthode exacte optimise un objectif (le C_{max}), tandis que l'algorithme de colonies de fourmis optimise un second objectif (la somme $\sum C_i$). Les problèmes traités par ces deux méthodes d'optimisation sont donc différents et l'approche est de type spécialiste.

2.3.3 Implémentation

Dans la plupart des cas, les travaux proposant des métaheuristiques hydrides sont des programmes séquentiels. Cependant, les algorithmes hybrides parallèles deviennent très intéressants lorsque les problèmes traités sont de grandes tailles. Ainsi, il existe quand même un certain nombre d'études portant sur des algorithmes hybrides parallèles (voir la taxinomie de Talbi [Tal02]). Mais les algorithmes de coopérations méta/exacte parallèles sont peu nombreux. En effet, les grandes différences entre les deux types d'algorithmes rendent difficile la mise en oeuvre d'une architecture parallèle. Nous avons trouvé deux approches de coopérations méta/exactes parallèles, proposées par Cotta et Troya [CT03] et par Kleipeis *et al.* [KPF03].

Nous avons donc défini plusieurs critères permettant de classifier les méthodes coopératives, classification qui fait l'objet des deux sections suivantes.

2.4 Classification des coopérations Méta/Méta

Depuis quelques années déjà, de nombreuses études proposent de faire coopérer deux ou plusieurs métaheuristiques. Une classification de ces études suivant cette taxonomie est proposée dans [Tal02]. A peu près tous les types d'approches ont été proposés pour ce type de coopération. Leur efficacité surclassant d'une manière générale les métaheuristiques non

hybridées fait que les métaheuristiques hybrides sont devenues maintenant assez classiques dans le domaine de l'optimisation. Nous ne ferons donc pas ici une classification de ces méthodes, mais nous nous référons à [Tal02] pour un état de l'art de ces approches.

2.5 Classification des coopérations Méta/Exacte

Les études de coopérations méta/exacte étant moins courantes, leur classification restait jusqu'à récemment assez inutile. Mais leur nombre grandissant la rend aujourd'hui plus utile afin de voir les différents types d'approches proposées. Un état de l'art de ces coopérations a d'ailleurs été proposé récemment par Stützle et Dumitrescu [DS03]. Ils distinguent cinq types d'approches des coopérations méta/exacte et détaillent une application trouvée dans la littérature pour chaque type d'approche. La classification proposée n'est pas de type 'taxinomie', mais sépare les approches selon la méthode utilisée pour la coopération. Ces classes sont :

- Utilisation d'un algorithme exact pour explorer des larges voisinages dans un algorithme de recherche locale.
- Utilisation des solutions de bonne qualité afin de réduire l'espace de recherche de la méthode exacte.
- Exploitation des bornes de la méthode exacte pour une heuristique constructive.
- Utilisation des informations fournies par les relaxations des problèmes linéaires pour orienter un algorithme de recherche locale ou constructif.
- Utilisation d'une méthode exacte pour une fonction spécifique de la métaheuristique.

L'état de l'art de Stützle et Dumitrescu survole les grandes 'branches' de la coopération méta/exacte, mais les exemples proposés utilisent, dans la plupart des cas, des recherches locale ou Tabou pour métaheuristique.

Notre taxinomie propose une approche plus générale de la classification, qui permet de regrouper les coopérations méta/méta et méta/exacte. De plus, la classification proposée dans cette section permet d'inclure facilement des nouveaux schémas de coopération.

Après avoir défini la grammaire de notre classification, ainsi que certaines notations, nous proposerons un tableau récapitulatif des coopérations méta/exacte trouvées dans la littérature.

2.5.1 Grammaire et vocabulaire de la classification

Nous avons repris la grammaire proposée dans [Tal02], puis nous l'avons modifiée pour permettre de prendre en compte les approches méta/exacte. En particulier, la grammaire détaillée ici ne distingue pas les modèles homogènes des modèles hétérogènes (inutile pour les coopérations méta/exacte - et cette information peut se retrouver dans la grammaire en regardant les deux métaheuristiques hybridées) ; par contre l'approche de résolution (approchée ou exacte), y est notamment ajoutée.

Pour les différentes métaheuristiques qui sont rencontrées dans cette taxinomie, nous utilisons les abréviations suivantes :

$< algorithme\ coopératif > \longrightarrow < conception >< implémentation >$
$< conception > \longrightarrow < hiérarchique >< plat >$
$< hiérarchique > \longrightarrow < LRH > | < LTH > | < HRH > | < HTH >$
$< LRH > \longrightarrow$ LRH $(< métaheuristique >(< exact >)|(< exact >(< métaheuristique >))$
$< LTH > \longrightarrow$ LTH $(< métaheuristique >(< exact >)|(< exact >(< métaheuristique >))$
$< HRH > \longrightarrow$ HRH $((< métaheuristique >+< exact >)|(< exact >+< métaheuristique >))$
$< HTH > \longrightarrow$ HTH $((< métaheuristique >+< exact >)|(< exact >+< métaheuristique >))$
$< plat > \longrightarrow (< résolution >, < optimisation >, < fonction >)$
$< résolution > \longrightarrow$ exacte | approchée
$< optimisation > \longrightarrow$ globale | partielle
$< fonction > \longrightarrow$ générale | spécialiste
$< implémentation > \longrightarrow$ séquentielle | parallèle $< type\ parallèle >$
$< type\ parallèle > \longrightarrow$ statique | dynamique | adaptative
$< métaheuristique > \longrightarrow$ RS | AG | AM | SE | PG | RN | AD | RL | RLI | RT | HG | CF | RD | HS | CSA | $< algorithme\ coopératif >$
$< exact > \longrightarrow$ B&B | αB&B | B&C | B&P | B&C&P | PL | PLM | PD | AS | RB | MS | $< algorithme\ coopératif >$

FIG. 2.11 – Grammaire pour la coopération de méthodes d'optimisation (exactes ou approchées).

- RS : Recuit simulé (*Simulated Annealing*)
- AG : Algorithme génétique (*Genetic Algorithm*)
- AM : Algorithme mimétique (*Memetic Algorithm*)
- SE : Stratégie évolutionnaire (*Evolution Strategy*)
- PG : Programmation génétique (*Genetic Programming*)
- RN : Réseau de neurones (*Neural Network*)
- AD : Algorithme de descente (*Descent Walk*)
- RL : Recherche Locale (*Local Search*)
- RLI : Recherche Locale Itérée (*Iterated Local Search*)
- RT : Recherche Tabou (*Tabu Search*)
- HG : Heuristique gloutonne (*Greedy Heuristic*)
- CF : Colonies de fourmis (*Ant Colonies*)
- RD : Recherche par dispersion (*Scatter Search*)
- HS : Heuristique Spécifique
- CSA : *Conformational Space Annealing*

Pour les méthodes exactes, nous utiliserons les abréviations suivantes :

- B&B : *Branch & Bound*
- αB&B : α *Branch & Bound*
- B&C : *Branch & Cut*
- B&P : *Branch & Price*
- B&C&P : *Branch & Cut & Price*
- PL : Programmation Linéaire (*Linear Programming*) (exemple : solveur CPlex, ...)
- PLM : Programmation Linéaire Mixte (*Linear Programming*) (exemple : solveur CPlex, ...)

- PD : Programmation Dynamique (*Dynamic Programming*)
- PC : Programmation par Contraintes (*Constraint Programming*)
- AS : Algorithme du Simplexe (*Simplex Search*)
- RB : Recherche par But (*Search Goal*)
- MS : Méthode spécifique au problème

2.5.2 Classification

Voici un tableau récapitulatif des coopérations méta/exacte de la littérature. Notons que les coopérations peuvent être multiples.

TAB. 2.1: Bibliographie annotée des coopérations méta/exacte.

Référence	**Design**	**Problème d'optimisation**
[ABCC99]	LTH(PD(RLI)) (approchée, partielle, générale) séquentiel	Voyageur de Commerce
[ANS96]	HRH(B&P + HS) (approchée, partielle, générale) séquentielle	Partitionnement large échelle
[BCK01]	LRH(RL(PD)) (approchée, partielle, générale) séquentielle	Voyageur de commerce asymétrique
[BD01]	HRH(RT + PL) (approchée, partielle, générale) séquentielle	Coupes irrégulières
[BGS00]	HRH(HG + LTH(RT(B&B))) (approchée, partielle, générale) séquentielle	Design réseaux de vols directs
[BH03]	LTH(AG(PC)) (approchée, partielle, générale) séquentielle	Fonctions continues
[BKY02]	HRH(HRH(HG + RL) + B&C) (exacte, globale, générale) séquentielle	Routage de véhicules avec fenêtres de temps
[BV04]	HRH(RS + LTH(RL(B&B))) (approchée, partielle, générale) séquentielle	Routage de véhicules avec fenêtres de temps
[CANT95]	LTH(AG(B&B)) (approchée, partielle, générale) parallèle statique	Voyageur de commerce
[CCJ+97]	HRH(RL + PL) (approchée, partielle, générale)	Problème de fabrication de câbles à fibre optique

suite à la page suivante ⟶

TAB. 2.1 : Bibliographie annotée des coopérations méta/exacte (suite).

Référence	Design	Problème d'optimisation
	séquentielle	(ordonnancement)
[CDP02]	HTH(B&P+HRH(PLM+RL)) (exacte, globale, spécialiste) séquentielle	Routage de véhicules
[CPv02]	LTH(RL(PD)) (approchée, partielle, générale) séquentielle	Flow-shop à une machine (somme pondérée des retards)
[CS05]	HRH(RT + RD) (approchée, partielle, générale) séquentiellele	Fonctions continues à multi-minima
[CT03]	LTH(AG(B&B)) (approchée, partielle, générale) parallèle statique	Emploi du temps
[DFK+00]	LTH(RT(B&B)) (approchée, partielle, générale) séquentielle	Tournées de véhicules
[DQ01]	HTH(CF+HTH(B&B + HS)) (approchée, partielle, générale) séquentielle	Design d'accès au réseau local
[FR04]	HRH(PL + AG) (approchée, partielle, générale) séquentielle	Affectation généralisée
[FRW01]	LRH(B&B + AG) (exacte, globale, générale) séquentielle	Max-SAT
[HL03]	HRH(HS + B&B) (approchée, partielle, générale) séquentielle	Flow-shop
[Jah02]	LRH(MST(LTH(AG(B&B|ACM))) (approchée, globale, générale) séquentielle	Voyageur de commerce
[Joz04]	HTH(AG + B&C) (approchée, partielle, générale) séquentielle	Tournée de véhicules multi-objectif
[KF04]	LTH(PLM(PG)) (exacte, globale, spécialiste) séquentielle	Jeux de données MIPLIB3
[KLM+04]	HRH(PL + AM) (approchée, partielle, générale) séquentielle	Problème de l'arbre de Steiner
[KPF03]	HRH(αB&B + CSA) (approchée, partielle, spécialiste) parallèle statique	Prédiction de structure de protéines

suite à la page suivante ⟶

TAB. 2.1 : Bibliographie annotée des coopérations méta/exacte (suite).

Référence	Design	Problème d'optimisation
[Man99]	LRH(CF(PL)) (approchée, globale, spécialiste) séquentielle	Affectation quadratique
[MM97]	HRH(B&B + RT) (approchée, partielle, générale) séquentielle	Affectation quadratique
[PAM04]	HTH(RL+PC) (approchée, partielle, générale) séquentielle	Ordonnancement de projets avec contrainte de ressources
[PG99]	HTH(RL+PC) (approchée, partielle, générale) séquentielle	Problème du voyageur de commerce
[PRRL97]	HRH(AD + B&B) (exacte, globale, générale) séquentielle	Affectation quadratique
[PVDD98]	HRH(HS + (HTH(B&B+AG))) (exacte, globale, générale) séquentielle	Flow-shop hybride
[RR97]	HRH(HS+B&B) (approchée, partielle, générale) séquentielle	Problème *p-median*
[Sha98]	LTH(RL(B&B)) (approchée, partielle, générale) séquentielle	Routage de véhicules
[SW03]	HRH(B&B + RN) (approchée, partielle, générale) séquentielle	Ordonnancement de diffusion d'informations bi-objectif
[TMTL02]	HRH(MS + CF) (approchée, partielle, spécialiste) séquentielle	Flow-shop biobjectif sur 2 machines
[UYI04]	LRH(RLI(AS)) (approchée, globale, spécialiste) séquentielle	Variante du problème de découpe à une dimension
[VH01]	HRH(AS + RT) (approchée, partielle, générale) séquentielle	0-1 Sac à dos multi-dimensionnel

2.6 Conclusion

Les méthodes d'optimisation sont divisées en deux grandes classes, selon qu'elles résolvent les problèmes de manière exacte ou approchée. Au fur et à mesure des années et de l'augmentation exponentielle de puissance de calcul des ordinateurs, ces méthodes d'optimisation sont devenues de plus en plus évoluées.

Cette puissance de calcul disponible a abouti depuis quelques années à de nombreux travaux tentant de faire coopérer plusieurs de ces méthodes d'optimisation entre elles. Dans un premier temps, la coopération était réalisée entre méthodes d'optimisation approchées. Mais actuellement, de plus en plus de coopérations entre méthodes exactes et approchées voient le jour.

Nous avons présenté brièvement les différentes méthodes de résolution, approchées ou exactes. Puis nous avons proposé de faire une classification de ces méthodes, en particulier selon leur mode et leur niveau de coopération. Enfin, nous avons recensé et classifié les différentes approches de coopération méta/exacte que nous avons trouvé dans la littérature. Ces approches ne sont pas encore très nombreuses, mais elles sont de plus en plus courantes.

Nous avons recensé 36 articles traitant explicitement de la coopération entre méthodes d'optimisation exactes et métaheuristiques. La classification réalisée montre que beaucoup des coopérations recensées sont réalisées entre deux méthodes indépendantes exécutées en séquence (coopérations **HRH**). Ce type de coopération concerne 18 articles. Pour ce qui est de la classification à plat, 23 des 36 articles proposent des approches heuristiques, partielles et générales.

Avec cette classification, on remarque que quelques voies de coopérations peuvent encore être explorées ou approfondies, comme les coopérations spécialistes, ou les coopérations exactes sur un domaine partiel. De plus, il ressort que très peu de travaux traitent des coopérations dédiées à l'optimisation multi-objectif ou proposent des modèles parallèles.

Chapitre 3

Le problème de flow-shop multi-objectif

Dans ce chapitre, nous présenterons dans un premier temps les problèmes d'optimisation multi-objectif, ainsi que les différentes approches existant pour leur résolution, et les métriques permettant d'évaluer la qualité d'un résultat. Puis nous présenterons un problème de Flow-shop bi-objectif, qui sera étudié dans les chapitres suivants. Nous faisons un état de l'art des approches de résolution des problèmes d'ordonnancement, en particulier les approches coopératives concernant des problèmes de Flow-shop.

3.1 Introduction

De nombreux secteurs de l'industrie (mécanique, chimie, télécommunications, environnement, transports, ...) sont concernés par des problèmes complexes de grande dimension. Ces problèmes d'optimisation, aux enjeux souvent importants, sont de surcroît très souvent de nature multi-objectif. D'une manière générale, très peu de problèmes d'optimisation rencontrés en pratique sont mono-objectif. Il existe en général plusieurs critères permettant d'évaluer une solution à un problème réel (coût de production, qualité, coût d'entretien, équilibre des tâches à réaliser,...). Ces critères doivent être optimisés simultanément et sont souvent contradictoires. L'optimisation multi-objectif s'intéresse à la résolution de ce type de problèmes. Elle possède ses racines au $19^{ème}$ siècle dans les travaux de Edgeworth et Pareto [Edg81, Par96]. Le constat que la plupart des problèmes réels sont de nature multi-objectif a provoqué une intensification des études portant sur ce type de problèmes depuis la fin des années 1980. Ces problèmes ont la particularité d'être beaucoup plus difficiles à traiter que leur équivalents mono-objectif. Les approches exactes multi-objectif sont très vite limitées, et les métaheuristiques "classiques" peu efficaces. Des métaheuristiques dédiées à la résolution de ces problèmes multi-objectif ont été mises au point, mais la difficulté de ce type de problèmes encourage l'utilisation de méthodes d'optimisation coopératives pour leur résolution.

La section 3.2 est dédiée à l'optimisation multi-objectif. Nous définirons en premier lieu quelques termes dédiés aux problèmes multi-objectif. Puis nous présenterons les différents types d'approches de résolutions (approchée ou exacte) trouvées dans la littérature. De plus, la comparaison des solutions trouvées n'étant pas triviale pour le cas multi-objectif, nous présenterons différents indicateurs de performances afin d'évaluer l'efficacité des algorithmes. Dans la section 3.3 nous présentons la classe des problèmes d'ordonnancement. Auparavant, nous ferons un tour d'horizon des études réalisées pour la résolution des problèmes d'ordonnancement, et plus spécialement ceux de type Flow-shop. Dans un dernier temps, nous présenterons le problème de Flow-shop bi-objectif que nous traiterons dans la suite de cette thèse. Enfin, nous ferons un récapitulatif dans la section 3.4.

3.2 L'Optimisation multi-objectif

Dans cette section, nous commencons par définir quelques notions de l'optimisation multi-objectif, puis nous présentons les différentes méthodes de résolution existantes dans ce domaine.

3.2.1 Définitions

Définissons dans un premier temps la notion de problème d'optimisation multi-objectif, ainsi que les caractèristiques des solutions d'un tel problème.

3.2.1.1 Problème d'optimisation multi-objectif

L'optimisation multi-objectif consiste à optimiser plusieurs composantes d'un vecteur de fonctions de coût, chaque composante de ce vecteur correspondant à un objectif. Un Problème d'Optimisation Multi-objectif (*Multiobjective Optimization Problem - MOP*), peut être défini comme suit[1] :

$$(MOP) = \begin{cases} min\ F(x) = (f_1(x), f_2(x), \ldots, f_n(x)) \\ t.q.\ x \in D \end{cases} \tag{3.1}$$

$n \geq 2$ est le nombre de fonctions objectifs, $x = (x_1, x_2, \ldots, x_r)$ le vecteur de variables de décision, D l'espace des solutions réalisables (espace décisionnel), et $F(x)$ le vecteur des n fonctions objectifs à optimiser. L'ensemble $O = F(D)$ correspond à l'ensemble des points atteignables dans l'espace objectif, et $f_x = (f_1(x), f_2(x), \ldots, f_n(x))$ est un point de l'espace objectif (Fig.3.1).

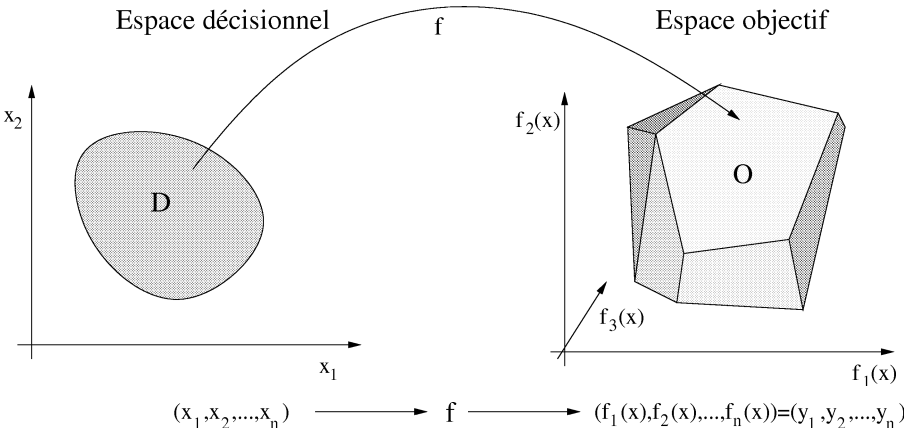


FIG. 3.1 – Problème d'optimisation multi-objectif : exemple avec 2 variables de décision et 3 fonctions objectifs.

3.2.1.2 Solution d'un problème d'optimisation multi-objectif

Pour les solutions de problèmes multi-objectif, 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). On parle alors de *solutions de compromis*.

L'optimisation multi-objectif consiste à définir des méthodes efficaces pour la résolution de problèmes où tous les objectifs sont pris en compte. Son but est de trouver des solutions offrant un bon compromis entre les différentes fonctions objectifs à optimiser. Un *décideur* choisit alors parmi les différentes solutions proposées par l'optimiseur la solution de compromis qui lui convient le mieux.

[1] On parle ici de minimisation des objectifs, le parallèle avec les problèmes de maximisation se fait aisément.

La notion d'optimalité la plus généralement admise est celle introduite par Edgeworth en 1881 [Edg81], généralisée plus tard par Pareto en 1896 [Par96]. Le terme le plus employé pour s'y référer est celui d'*optimum de Pareto*. Donc, il convient de manipuler des populations de solutions dites Pareto optimales. L'ensemble des solutions Pareto optimales est aussi appelé frontière Pareto. Il convient de définir la notion de Pareto optimalité, ainsi que d'autres notions s'y rapportant[2] :

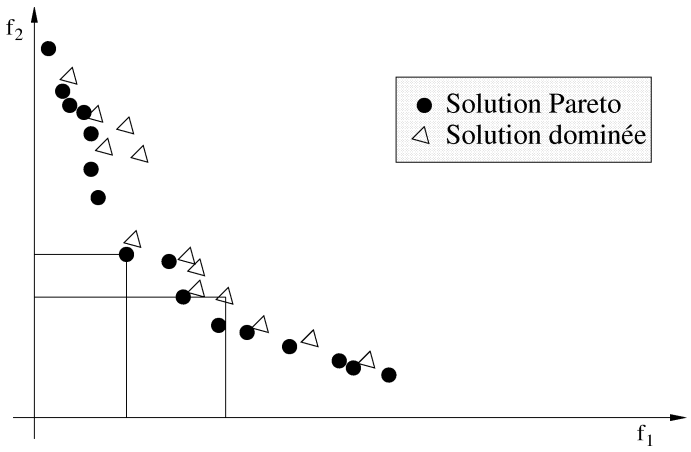


FIG. 3.2 – Solutions Pareto optimales (cas pour la minimisation de 2 objectifs).

Définition 2 *Une solution x_i domine une solution x_j si et seulement si :*
$\forall k \in [1..n], f_k(x_i) \leq f_k(x_j)$ *et* $\exists k \in [1..n]$ *tq* $f_k(x_i) < f_k(x_j)$,
où n est le nombre d'objectifs à optimiser. On notera alors $x_j \preceq x_i$.

Le fait qu'une solution x_i domine une solution x_j sera noté $x_i \preceq x_j$. Si x_i est meilleur que x_j pour tous les objectifs, on notera alors $x_i \prec x_j$. Lorsqu'on a ni $x_i \preceq x_j$, ni $x_j \preceq x_i$, on notera alors $x_i \sim x_j$.

Définition 3 *Une solution est dite Pareto optimale si elle n'est dominée par aucune autre solution admissible.*

Définition 4 *Pour un problème d'optimisation multi-objectif donné $F(x)$, l'ensemble Pareto optimal $\mathcal{PO}^*$ est défini comme suit :*

$$\mathcal{PO}^* = \{x \in D \mid \nexists x' \in D, F(x') \preceq F(x)\} \tag{3.2}$$

L'image de l'ensemble Pareto optimal dans D est appelée frontière Pareto, ou surface de compromis. L'allure de cette frontière prend des formes différentes selon que les objectifs doivent être minimisés ou maximisés. Un exemple avec deux critères est représenté sur la figure 3.3.

[2] On considère ici que l'on veut minimiser les objectifs, mais le problème resterait inchangé si l'on voulait en maximiser un certain nombre.

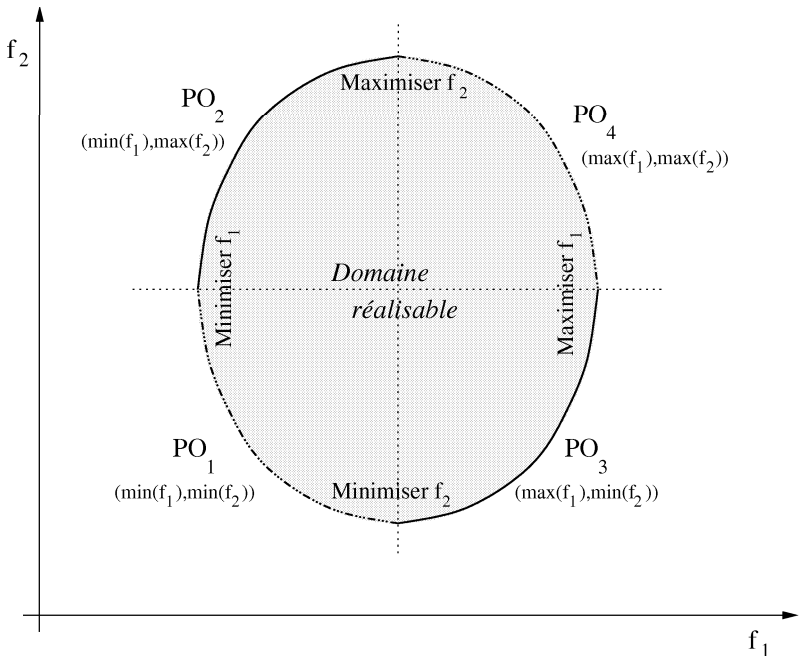


FIG. 3.3 – Allure de la frontière Pareto selon que l'on considère une maximisation ou une minimisation des différents objectifs (cas avec deux objectifs).

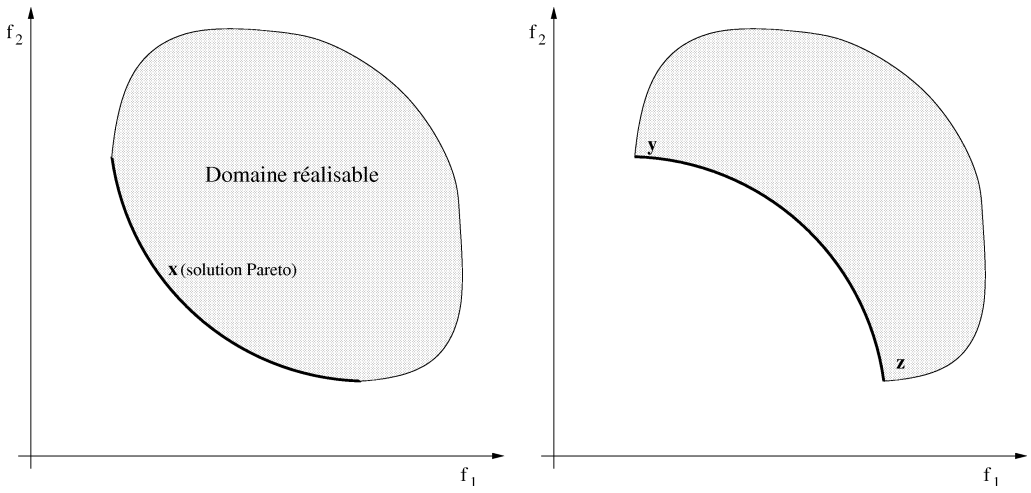
De plus, l'ensemble Pareto optimal peut être divisé en deux sous-ensembles : l'ensemble des solutions supportées et l'ensemble des solutions non-supportées. Les solutions supportées sont les solutions Pareto optimales dont le point correspondant dans D se trouve sur l'enveloppe convexe de O. Les solutions Pareto n'appartenant pas à l'enveloppe convexe sont dites non-supportées. La figure 3.4 montre les cas extrêmes. La figure 3.5 représente un cas plus général de l'optimisation combinatoire multi-objectif.

Les premières approches de l'optimisation multi-objectif n'utilisaient pas ces notions de dominance et de solutions Pareto optimales. Elles ramenaient les problèmes multi-objectif à des problèmes mono-objectif. Cette approche utilisant les notions de dominance entre les solutions a été proposée pour la première fois par Goldberg [Gol89]. Dans la prochaine section, nous présentons les différentes manières d'aborder la résolution d'un problème multi-objectif, en utilisant directement ou non la notion d'optimalité Pareto. Puis nous présentons les manières d'appliquer ces méthodes aux techniques classiques d'optimisation mono-objectif tels que les AGs ou $B\&B$.

3.2.2 Approches de résolution multi-objectif

Nous présentons ici quelques approches proposées au fil des années pour la résolution de problèmes multi-objectif à l'aide de (méta)heuristiques. La qualité d'une méthode (méta)heuristique multi-objectif est définie par plusieurs critères :

– Les solutions trouvées doivent être intéressantes du point de vue de la qualité.



(a) Une frontière Pareto convexe (b) Une frontière Pareto concave

FIG. 3.4 – Cas de la minimisation de deux objectifs. (a) Frontière Pareto convexe - Cas extrême : toutes les solutions Pareto optimales sont supportées. (b) Frontière Pareto concave - Cas extrême : seules les deux solutions x et y situées aux extrémités du front Pareto optimal sont supportées.

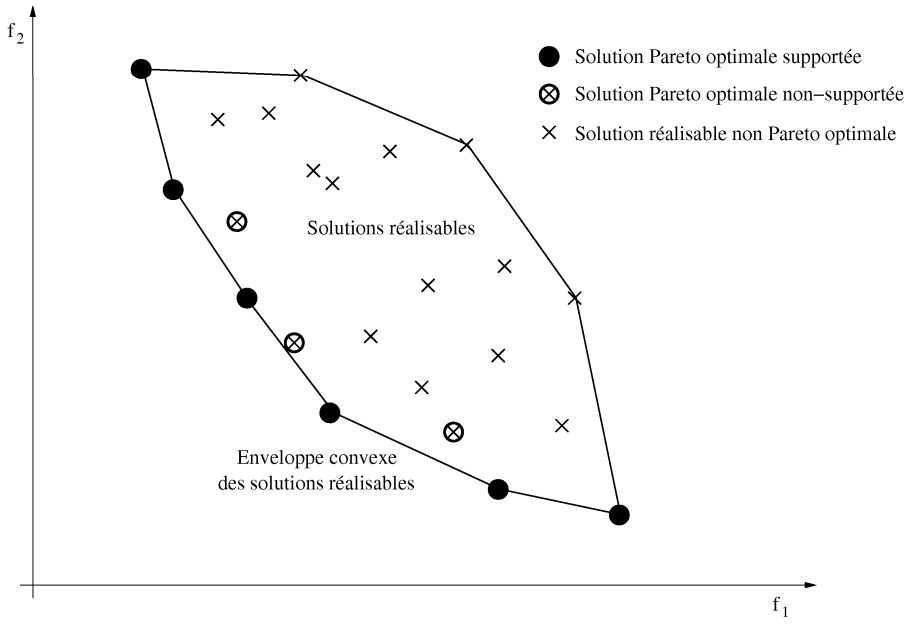


FIG. 3.5 – Solutions supportées/non-supportées - exemple en optimisation combinatoire à deux objectifs (minimisation).

– L'ensemble des solutions trouvé doit être assez grand pour fournir un choix suffisamment large au décideur.
– Dans le même but, les solutions trouvées doivent être réparties le mieux possible le long de la frontière Pareto.

Afin d'évaluer la qualité des fronts proposés par les différentes approches des indicateurs de performance ont été mis au point. Ceux-ci sont présentés dans la section 3.2.3.

Les méthodes d'optimisation multi-objectif peuvent être classées en trois catégories :
– Les approches scalaires, qui transforment le problème multi-objectif en problème mono-objectif.
– Les approches non scalaires et non Pareto, qui gardent l'approche multi-objectif, mais en traitant séparément chacun des objectifs.
– Les approches Pareto, qui utilisent les notions de dominance Pareto.

Un recensement des différentes approches de résolution des problèmes multi-objectif est proposé par Collette et Siarry [CS03].

3.2.2.1 Les approches scalaires

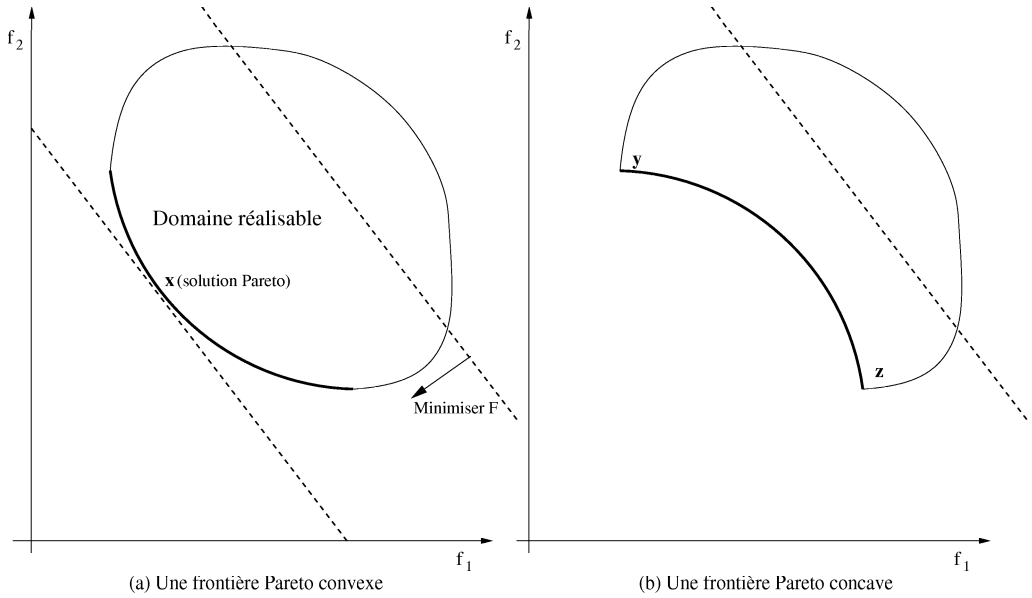
A l'origine, les problèmes multi-objectif étaient usuellement transformés en problèmes mono-objectif. Pour de telles approches, une fonction objectif est créée de telle sorte qu'elle permet de traiter le problème d'optimisation multi-objectif comme un problème d'optimisation classique à un seul objectif, et donc d'utiliser les méthodes déjà existantes pour traiter de tels problèmes. Plusieurs approches différentes ont été mises au point pour transformer les problèmes multi-objectif en problèmes mono-objectif : les méthodes d'agrégation, les méthodes avec vecteur cible, et les méthodes ϵ-contraintes.

3.2.2.1.1 Méthodes d'agrégation

C'est l'une des premières méthodes employée pour résoudre les problèmes multi-objectif. C'est aussi l'une des plus largement employées [IM98, AOBH96, EE99, GHGL99]. La transformation en problème mono-objectif se réalise en définissant une fonction de coût unique comme étant la somme pondérée des différentes fonctions objectifs du problème initial :

$$F(x) = \sum_{i=1}^{n} \delta_i f_i(x) \tag{3.3}$$

En définissant le problème de cette manière, la représentation du domaine réalisable de $F(x)$ devient linéaire, comme le montre la figure 3.6(a). Donc, pour une agrégation donnée, il n'existe qu'une seule valeur optimale pour le problème (la solution x sur la figure 3.6(a)). Généralement, les approches proposées résolvent le problème en utilisant différentes valeurs pour les poids δ_i. Ainsi, on peut découvrir l'ensemble des solutions Pareto supportées du problème initial. Mais, dans le cas d'une frontière de Pareto concave, les solutions non supportées sont alors négligées. La figure 3.6(b) représente un cas extrême où seules deux solutions Pareto optimales peuvent être trouvées.



(a) Une frontière Pareto convexe (b) Une frontière Pareto concave

FIG. 3.6 – La méthode d'agrégation.

3.2.2.1.2 Méthodes avec vecteur cible

Dans les méthodes de ce type, un ensemble de buts (ou cibles) que l'on désire atteindre est défini. La méthode d'optimisation essaie alors de minimiser la différence entre les solutions envisagées et les buts. Ces méthodes, bien que travaillant par agrégation des objectifs, permettent de générer des solutions non-supportées. Différentes approches sont envisageables pour ce type de méthode, comme celles du but à atteindre [WLK92], du but programmé [MPT00], ou du min-max [Coe98].

3.2.2.1.3 Méthode ϵ-contrainte

Dans cette méthode, une seule fonction objectif est optimisée, les autres fonctions objectif sont remplacées par des contraintes. Le problème d'optimisation associé s'écrit alors :

$$(MOP_\epsilon) = \begin{cases} min\ f_k(x) \\ t.q. \\ x \in \Omega \\ f_j(x) \leq \epsilon_j,\ j = 1, \ldots, n, j \neq k \end{cases} \tag{3.4}$$

L'ensemble Pareto optimal peut alors être obtenu en faisant varier les valeurs de ϵ_j, et en ne résolvant ainsi qu'un ensemble de problèmes d'optimisation mono-objectif. Des exemples d'applications de ces méthodes peuvent être trouvés dans les travaux de Lee [Lee97] et de Schott [Sch95].

3.2.2.2 Les approches non-Pareto et non-scalaires

Ces méthodes ne transforment pas le problème multi-objectif en un problème mono-objectif mais elle n'utilisent pas non plus la notion de dominance Pareto.

3.2.2.2.1 Sélection parallèle

Cette méthode a certainement été la première proposant un algorithme génétique pour la résolution de problèmes multi-objectif [Sch85]. L'algorithme proposé, *VEGA* (*Vector Evaluated Genetic Algorithm*), sélectionne les individus selon chaque objectif de manière indépendante (sélection parallèle).

3.2.2.2.2 Méthode lexicographique

Cette méthode, proposée par Fourman [Fou85], classe les objectifs en fonction d'un ordre d'importance proposé par le décideur. Ensuite les fonctions objectif sont traitées dans cet ordre pour obtenir l'optimum. Le principe est schématisé sur la figure 3.7. L'optimisation séquentielle des différents objectifs aboutit à la découverte d'une seule solution optimale. Des variantes de la méthode peuvent être définies afin d'en découvrir plusieurs, mais la méthode reste assez inappropriée pour obtenir ou approcher le front Pareto correspondant à un problème multi-objectif.

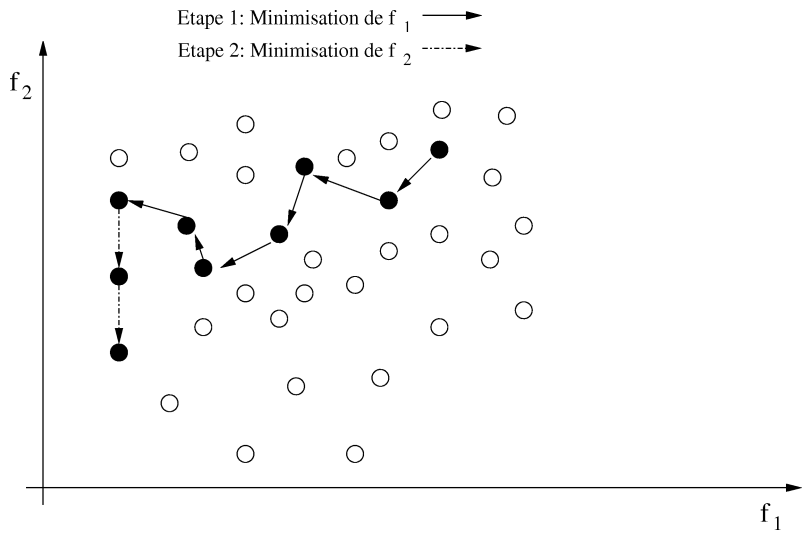


FIG. 3.7 – La méthode lexicographique : La fonction $f1$ est optimisée, puis la fonction $f2$, en ne tenant compte que des solutions de coût optimal pour la fonction $f1$.

3.2.2.2.3 Méthode avec genres

Proposée par Allenson pour traiter un problème bi-objectif, cette méthode utilise la notion de genre (masculin ou féminin), et d'attracteur [All92]. En affectant un objectif à chaque genre, l'auteur espère minimiser les deux objectifs simultanément, car un genre sera toujours jugé d'après l'objectif qui lui a été associé.

Allenson utilise un algorithme génétique classique, mais lors de la phase de sélection, le principe d'attracteur est introduit pour éviter que des individus trop éloignés du point de vue de leur *fitness* puissent être croisés et pour permettre à des individus de mauvaise *fitness* de se reproduire. Dans l'étude d'Allenson, où l'on se trouve dans un cas bi-objectif, les deux genres sont mâle et femelle. La reproduction n'est alors permise qu'uniquement entre un individu mâle et un individu femelle, le genre étant affecté aléatoirement à la création de l'individu.

L'utilisation des genres est une manière de définir des sous-populations séparées par rapport à chaque objectif. La différence avec la sélection parallèle et lexicographique est que la reproduction impose une restriction de voisinage, évitant la reproduction aléatoire entre individus.

L'inconvénient des trois méthodes non Pareto et non scalaires présentées dans cette section est qu'elles tendent à générer des solutions qui sont largement optimisées pour certains objectifs et très peu pour les autres objectifs. Les solutions de compromis sont négligées.

3.2.2.3 Les approches Pareto

Les approches Pareto utilisent directement la notion de dominance Pareto. Goldberg [Gol89] a été le premier à proposer ce type d'approche pour résoudre les problèmes proposés par Schaffer [Sch85]. Ce concept a été introduit en premier lieu dans un algorithme génétique.

D'une manière générale, la plupart des approches Pareto trouvées dans la littérature sont incluses dans des algorithmes évolutionnaires, et une large part de ces algorithmes évolutionnaires sont des algorithmes génétiques. Dans le chapitre 4, nous développons la notion d'algorithme génétique multi-objectif.

Des approches non-évolutionnaires ont néanmoins été proposées, comme par exemple Czyzak et Jaszkiewicz pour le recuit simulé [CJ98]. D'une manière générale, les approches Pareto permettent de ne pas favoriser un objectif plutôt qu'un autre, mais les traitent de manière équitable, ce qui fournit une aide précieuse au décideur.

3.2.3 Évaluation des performances

La comparaison d'algorithmes pour la résolution exacte des problèmes d'optimisation est triviale. En effet, il suffit de comparer la taille et le nombre des problèmes résolus par chaque méthode, en tenant compte éventuellement du temps nécessaire pour la découverte de la (des) solution(s) optimale(s). Dans le cas de méthodes approximatives, la comparaison de deux solutions obtenues reste triviale pour l'optimisation mono-objectif (on compare les valeurs trouvées), mais pas pour l'optimisation multi-objectif. En effet, l'évaluation des performances de solutions d'un algorithme multi-objectif n'est pas triviale, puisque celui-ci fournit un ensemble de solutions non dominées et donc non-comparables entre elles. Il est donc nécessaire d'utiliser des indicateurs de performance afin d'évaluer ces algorithmes.

De nombreuses mesures pour calculer l'efficacité des méthodes d'optimisation pour

les problèmes multi-objectif ont été proposées. Cependant, chacune présente à la fois des avantages et des inconvénients. Dans cette section, les plus couramment utilisées sont décrites. Cependant, il est possible de trouver des études comme celle de Knowles et Corne [KC02], qui comparent plusieurs métriques.

Nous avons séparé les métriques en deux ensembles : celles qui offrent une mesure absolue par rapport à l'ensemble des solutions Pareto optimales PO^*, et celles qui n'ont pas besoin de PO^* et qui offrent une mesure relative. Les mesures peuvent servir à quantifier deux aspects des résultats. En effet, si l'objectif de la méthode n'est pas de rechercher une unique solution en prenant en compte les préférences du décideur, mais de générer un ensemble de solutions potentiellement Pareto (cas *a posteriori*), alors le but de la méthode est double. Premièrement, la méthode doit converger vers le front Pareto optimal ; deuxièmement elle doit trouver des solutions tout le long de cette frontière. La figure 3.8 montre les deux buts d'une méthode d'optimisation *a posteriori*. La figure 3.9 représente un ensemble de solutions répondant idéalement aux deux buts exposés ci-dessus. La figure 3.10 montre une approximation qui répond au premier but, mais qui est mauvaise par rapport au second. La figure 3.11 illustre le cas opposé.

Dans la suite de cette section, $\mathcal{P}_A$ représente l'ensemble des solutions potentiellement Pareto optimales trouvé par un algorithme A.

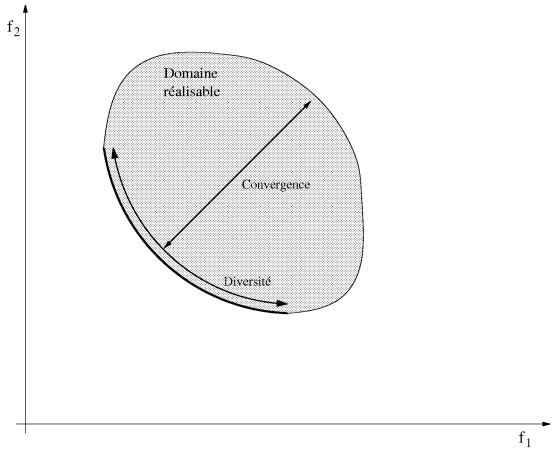


FIG. 3.8 – Les deux buts de l'optimisation multi-objectif.

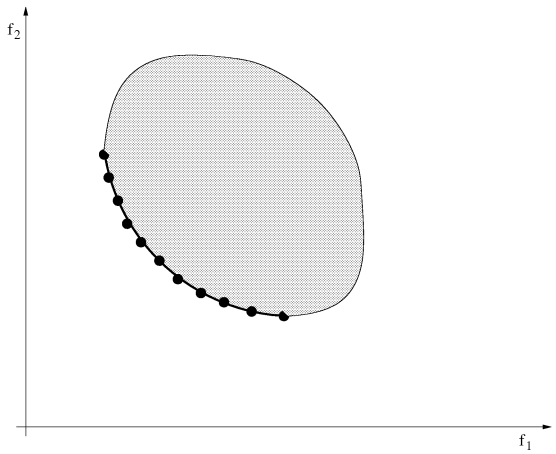


FIG. 3.9 – Une disposition idéale des solutions.

3.2.3.1 Ensemble PO^* connu

3.2.3.1.1 Proportion d'erreur

Cette mesure [VL00b] compte le nombre de solutions de $\mathcal{P}_A$ qui n'appartiennent pas à PO^*, soit :

$$ER(A) = \frac{\sum_{i=1}^{|\mathcal{P}_A|} e_i}{|\mathcal{P}_A|} \tag{3.5}$$

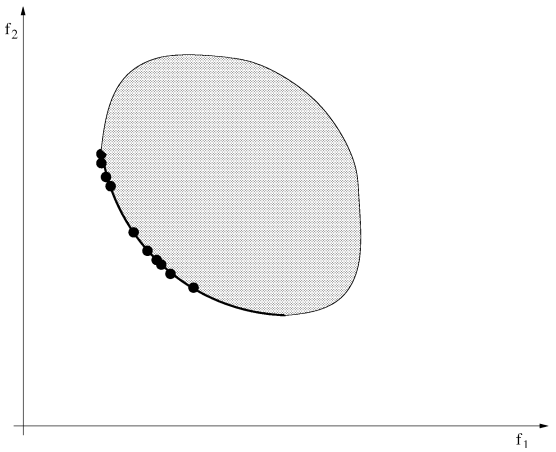


FIG. 3.10 – Solution de bonne qualité en terme de convergence, mais mauvaise pour la diversité.

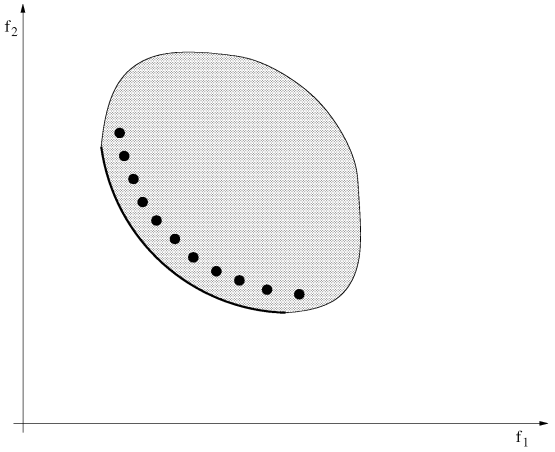


FIG. 3.11 – Solution de bonne diversité, mais de mauvaise qualité pour la convergence.

où $e_i = 1$ si la $i^{ème}$ solution de $\mathcal{P}_A$ appartient à PO^*, sinon $e_i = 0$.

Le désavantage de cette méthode est que, si aucune solution de $\mathcal{P}_A$ n'appartient à PO^*, elle n'apporte aucune information quant à la proximité relative de $\mathcal{P}_A$ par rapport à PO^* puisque dans ce cas, quelle que soit la distance séparant $\mathcal{P}_A$ de PO^*, on a $ER(A) = 0$.

3.2.3.1.2 Distance générationnelle

Cette mesure [VL00b] calcule la distance moyenne entre les solutions de $\mathcal{P}_A$ et celles de PO^*. Elle se calcule selon la formule suivante :

$$GD(A) = \frac{(\sum_{i=1}^{|\mathcal{P}_A|} d_i^p)^{\frac{1}{p}}}{|\mathcal{P}_A|} \tag{3.6}$$

Pour $p = 2$, le paramètre d_i est la distance Euclidienne (dans l'espace des objectifs) entre la solution $i \in \mathcal{P}_A$ et le membre le plus proche de PO^* :

$$d_i = \min_{k=1}^{|\mathcal{P}^*|} \sqrt{\sum_{j=1}^{n} (f_j^i - f_j^k)^2} \tag{3.7}$$

où f_j^i est la valeur de la $j^{ème}$ fonction objectif de i, et f_j^k est la valeur de la $j^{ème}$ fonction objectif de la $k^{ème}$ solution de PO^*.

La difficulté avec cette méthode est que, s'il existe un ensemble $\mathcal{P}_A$ pour lequel il y a une fluctuation importante dans les distances, la métrique peut ne pas retourner la véritable distance. Dans un tel cas, le calcul de l'écart type de la mesure GD est nécessaire. D'autre part, si les fonctions objectifs ne sont pas de même amplitude, elles doivent être normalisées avant le calcul.

3.2.3.1.3 Erreur maximale à la surface de compromis

Cette métrique permet de mesurer la distance entre PO^* et l'ensemble $\mathcal{P}_A$. Elle calcule en fait la plus grande distance minimale entre les solutions de $\mathcal{P}_A$ et les solutions les plus proches de $\mathcal{P}^*$. Sa définition pour un problème à deux objectifs est la suivante :

$$EM(A) = \max_j(\min_i(|f_1^i(x) - f_1^j(x)|^p + |f_2^i(x) - f_1^j(x)|)^{\frac{1}{p}}) \tag{3.8}$$

3.2.3.2 Ensemble PO^* inconnu

Lorsque l'ensemble PO^* est connu, les mesures calculent le plus souvent la distance entre l'approximation et PO^*. Lorsque l'ensemble PO^* est inconnu, elles permettent le plus souvent de comparer de manière relative deux approximations. Il existe cependant des mesures qui calculent une évaluation de la qualité d'une seule approximation. Les mesures évaluent la qualité des approximations soit par rapport à la convergence, soit par rapport à la diversification, ou par rapport aux deux buts en même temps.

3.2.3.2.1 Mesures évaluant la convergence

La métrique C

Cette mesure, proposée par Zitzler [Zit99], calcule la proportion de solutions d'un ensemble potentiellement Pareto optimal $\mathcal{P}_B$ dominées par des solutions d'un ensemble potentiellement Pareto optimal $\mathcal{P}_A$:

$$C(A,B) = \frac{|\{b \in \mathcal{P}_B | \exists a \in \mathcal{P}_A : a \prec b\}|}{|\mathcal{P}_B|} \tag{3.9}$$

$C(A,B) = 1$ signifie que toutes les solutions trouvées par l'algorithme B sont dominées par celles trouvées par l'algorithme A. Tandis que $C(A,B) = 0$ indique qu'aucune solution engendrée par B n'est dominée par une solution trouvée par A. Comme la relation de dominance n'est pas symétrique, $C(A,B)$ n'est pas forcément égal à $1-C(A,B)$. Il est donc nécessaire de calculer $C(A,B)$ et $C(B,A)$. Il est intéressant de noter que les cardinalités de $\mathcal{P}_A$ et $\mathcal{P}_B$ ne doivent pas forcément être identiques.

Contribution

Cette mesure [MTR00] se base également sur la comparaison de deux ensembles Pareto $\mathcal{P}_A$ et $\mathcal{P}_B$. Soit $\mathcal{P}_{AB}$ l'ensemble des solutions non-dominées de $\mathcal{P}_A \cup \mathcal{P}_B$. Cette mesure de contribution, mise au point dans notre équipe, calcule la proportion de l'ensemble $\mathcal{P}_{AB}$ représenté par les solutions de l'ensemble $\mathcal{P}_A$ ou de l'ensemble $\mathcal{P}_B$.

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

$$C = \mathcal{P}_A \cap \mathcal{P}_B \tag{3.10}$$

Soit W_A (respectivement W_B) l'ensemble des solutions de $\mathcal{P}_A$ (respectivement $\mathcal{P}_B$) qui dominent des solutions dans $\mathcal{P}_B$ (respectivement $\mathcal{P}_A$). Soit L_A (respectivement L_B) l'ensemble des solutions de $\mathcal{P}_A$ (respectivement $\mathcal{P}_B$) dominées par des solutions dans $\mathcal{P}_B$ (respectivement $\mathcal{P}_A$). L'ensemble des solutions N_A (respectivement N_B) de $\mathcal{P}_A$ (respectivement $\mathcal{P}_B$) n'ayant pas de relation de dominance avec toutes les solutions de $\mathcal{P}_B$ (respectivement $\mathcal{P}_A$) est alors :

$$N_A = \mathcal{P}_A \setminus (C \cup W_A \cup L_B) \tag{3.11}$$

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

$$PO = C \cup W_A \cup N_A \cup W_B \cup N_B \tag{3.12}$$

La contribution de l'algorithme A relativement à B, notée $Contribution(A, B)$, est le rapport des solutions non dominées de PO produites par A, au terme $|PO|$:

$$Contribution(A,B) = \frac{\frac{|C|}{2} + |W_A| + |N_A|}{|C| + |W_A| + |N_A| + |W_B| + |N_B|} = \frac{\frac{|C|}{2} + |W_A| + |N_A|}{|PO|} \tag{3.13}$$

La contribution de l'algorithme B relativement à l'algorithme A est définie d'une manière similaire. Notons que si les deux algorithmes produisent les mêmes solutions, alors on a $Contribution(A, B) = Contribution(B, A = 1/2)$

Si toutes les solutions produites par B sont dominées par les solutions produites par A, alors on a $Contribution(B, A) = 0$. De plus, dans le cas général, on a : $Contribution(A, B) + Contribution(B, A) = 1$.

Cette mesure permet d'avoir rapidement une idée de l'apport d'un algorithme par rapport à un autre algorithme.

3.2.3.2.2 Mesures évaluant la diversité

La métrique *Spacing*

Cette métrique, proposée par Schott [Sch95], calcule la distance relative entre deux solutions consécutives de $\mathcal{P}_A$, de la manière suivante :

$$S = \sqrt{\frac{1}{|\mathcal{P}_A|} \sum_{i=1}^{|\mathcal{P}_A|} (d_i - \bar{d})^2} \tag{3.14}$$

où

$$d_i = \min_{k \in \mathcal{P}_A \wedge k \neq i} \sum_{m=1}^{n} |f_m^i - f_m^k|$$

et $\bar{d}$ est la valeur moyenne de la distance précédente $\bar{d} = \frac{\sum_{i=1}^{|\mathcal{P}_A|} d_i}{|\mathcal{P}_A|}$. La distance d_i est la valeur minimale de la somme des différences absolues des valeurs des fonctions objectifs

entre la $i^{ème}$ solution et toutes les autres solutions de l'ensemble. Il est à noter que cette distance est différente de la distance *Euclidienne* minimale entre deux solutions.

Cette métrique calcule les écarts type des différentes valeurs de d_i. Ainsi, si les solutions sont uniformément espacées, la distance correspondante sera faible. Donc, plus un algorithme trouve un ensemble de solutions potentiellement Pareto optimales pour lequel cette mesure est faible, meilleur il est.

Métrique *Maximum spread*

Zitzler [Zit99] définit une métrique mesurant la longueur de la diagonale d'une "hyper-boîte" formée par les valeurs des fonctions objectifs extrêmes de l'ensemble potentiellement Pareto optimal généré :

$$D = \sqrt{\frac{1}{|\mathcal{P}_A|}\sum_{m=1}^{n}\left(\frac{\max_{i=1}^{|\mathcal{P}_A|} f_m^i - \min_{i=1}^{|\mathcal{P}_A|} f_m^i}{F_m^{\max} - F_m^{\min}}\right)^2} \tag{3.15}$$

où $F_m^{\max}$ et $F_m^{\min}$ sont le maximum et le minimum pour le $m^{ème}$ objectif. Le problème de cette mesure est qu'elle ne fournit aucune information sur la distribution exacte des solutions de compromis.

Entropie

Nous avons mis au point cette mesure afin d'évaluer la diversité d'une approximation $\mathcal{P}_A$ produite par un algorithme A par rapport à la diversité d'une approximation $\mathcal{P}_B$ produite par un algorithme B [BST02, MTR00]. On notera PO l'ensemble des solutions non dominées de l'union de $\mathcal{P}_A$ et $\mathcal{P}_B$.

Dans cette mesure, une niche est associée à chaque solution. Les solutions présentes dans chaque niche sont considérées comme voisines de la solution associée à la niche. L'entropie est alors donnée par la mesure :

$$E(A/B) = \frac{-1}{\log \gamma}\sum_{i=1}^{C}\frac{1}{N_i}\frac{n_i}{C}\log\frac{n_i}{C} \tag{3.16}$$

où N_i est le nombre de solutions de $\mathcal{P}_A \cup PO$ se trouvant dans la niche de la $i^{ème}$ solution de $\mathcal{P}_A \cup PO$, C est le cardinal de $\mathcal{P}_A \cup PO$, n_i est le nombre de solutions de l'ensemble $\mathcal{P}_A$ dans la niche de la $i^{ème}$ solution de $\mathcal{P}_A \cup PO$, et $\gamma = \sum_{i=1}^{C}\frac{1}{N_i}$ représente la somme des coefficients affectés à chaque solution.

Cette mesure permet donc une estimation de la diversité relative entre deux approximations. Toutefois, l'interprétation des résultats n'est pas toujours évidente.

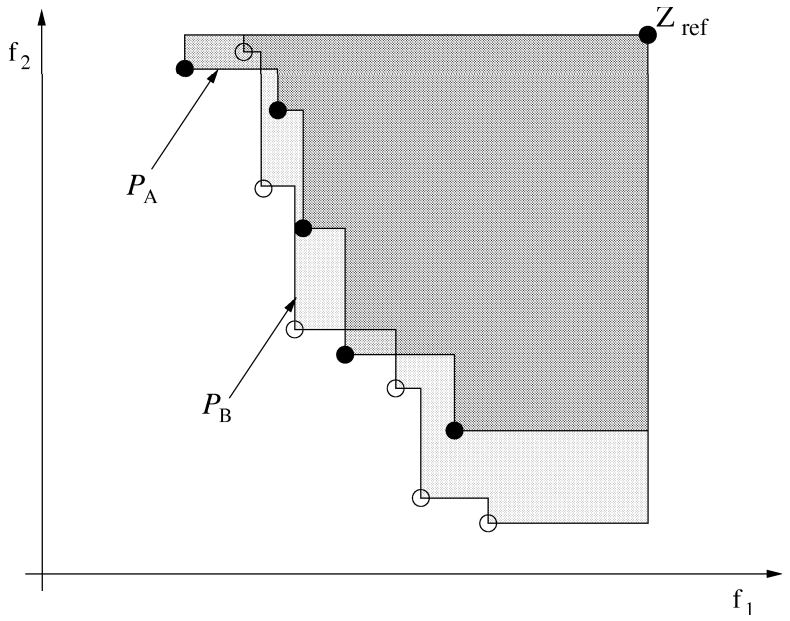


FIG. 3.12 – Métrique S, correspondant aux aires de dominance d'ensembles de solutions Pareto par rapport à un point de référence Z_{ref}.

3.2.3.2.3 Mesures évaluant la convergence et la diversité

Volume de l'espace dominé par un vecteur de référence

La mesure S, proposée par Zitzler [Zit99], calcule l'hypervolume de la région multidimensionnelle fermée par $\mathcal{P}_A$ et un point de référence, c'est-à-dire la taille de l'espace des objectifs que $\mathcal{P}_A$ domine.

Dans leur étude [KC02], Knowles et Corne recommandent l'utilisation de cette mesure, dont un exemple est illustré sur la figure 3.12.

Zitzler propose également une métrique D permettant de comparer deux fronts Pareto $\mathcal{P}_A$ et $\mathcal{P}_B$, s'appuyant sur S. Après avoir calculé les solutions non-dominées PO de l'union des deux ensembles $\mathcal{P}_A$ et $\mathcal{P}_B$, on calcule la valeur $D(\mathcal{P}_A, PO)$ qui équivaut à $S(PO - \mathcal{P}_B)$. La valeur obtenue correspond au volume dominé par PO_1, mais pas par PO_2. Il reste ensuite à comparer les valeurs $D(\mathcal{P}_A, PO)$ et $D(\mathcal{P}_B, PO)$.

3.2.3.3 Mesures utilisées

Pour nos expérimentations, nous connaîtrons certaines frontières Pareto optimales (pour les petites instances de problème). Dans ce cas, nous comparerons les temps nécessaires à la découverte de ces fronts optimaux. Mais, dans la plupart des cas, nous aurons affaire à des problèmes pour lesquels nous ne connaissons pas la frontière Pareto optimale.

Pour l'évaluation des différentes approximations Pareto calculées lors des expérimentations, nous utiliserons deux mesures : la métrique *Contribution* et la métrique S.

La contribution permet d'avoir facilement une idée de la supériorité d'un algorithme par rapport à un autre. Les chiffres sont très facilement interprétables et la mesure n'est sujette à aucun paramétrage. Cependant, cette mesure ne permet pas réellement de quantifier la différence d'efficacité des deux algorithmes (si l'on ne tient compte que de la comparaison de deux fronts entre eux - voir les Figures 3.13 et 3.14).

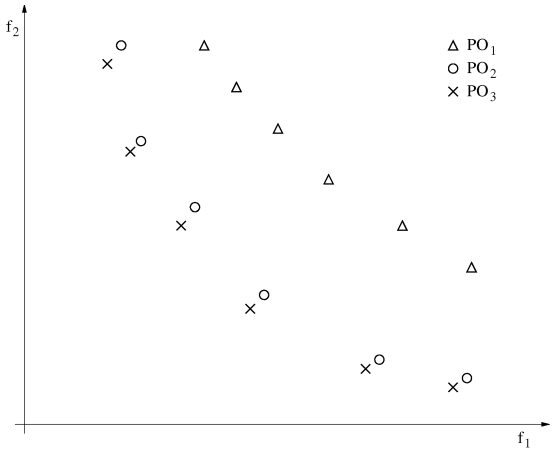


FIG. 3.13 – Limites de la métrique *Contribution* : ici, $Cont(PO_1, PO_3) = Cont(PO_2, PO_3) = 0$. Pourtant l'écart de performance entre PO_2 et PO_3 est beaucoup moins important que celui entre PO_1 et PO_3.

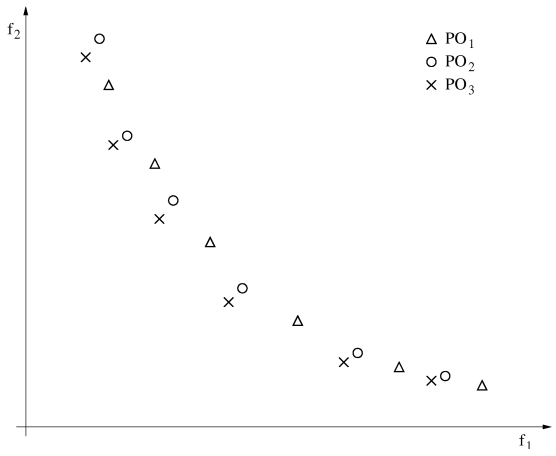


FIG. 3.14 – Ici, $Cont(PO_1, PO_3) = 0,5$ et $Cont(PO_2, PO_3) = 0$. Pourtant les qualités de PO_1 et PO_2 paraissent équivalentes.

Afin de compléter les informations fournies par la mesure de contribution, nous utiliserons la métrique S. L'avantage de cette mesure, dont la signification est intuitive, est qu'elle offre un ordre total entre différentes approximations. Cependant, elle nécessite, pour choisir le point de référence, la définition d'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. D'où l'intérêt d'utiliser cette mesure avec la contribution, et non seule. Nous avons utilisé la métrique S et pas son extension D, car S permet de comparer directement plusieurs ensembles de solutions, et de quantifier le rapport d'aire de dominance entre les fronts Pareto.

Lors de nos évaluations de performance ultérieures, nous prendrons comme point de référence le point *Nadir*[3] de l'ensemble des ensembles Pareto comparés.

[3]Le point *Nadir* est le vecteur des pires valeurs pour chaque objectif.

3.3 Le flow-shop multi-objectif

Le Flow-shop, qui nous servira par la suite de problème de test, fait partie de la classe des problèmes d'ordonnancement. Un problème d'ordonnancement se compose de tâches à réaliser et de machines disponibles pour l'exécution de ces tâches. La résolution d'un problème d'ordonnancement consiste à déterminer :

- l'ordre dans lequel chaque machine exécute les tâches,
- le placement des tâches dans le temps, c'est-à-dire les instants de début d'exécution de chaque tâche sur chaque machine.

Dans un premier temps, nous allons rappeler les différentes classes de problèmes d'ordonnancement rencontrés dans le domaine de l'optimisation. Puis nous présenterons le problème qui nous concernera dans la suite de cette thèse.

3.3.1 Formalisation

3.3.1.1 Les problèmes d'ordonnancement

3.3.1.1.1 Notations

Deux notations existent pour référencer les problèmes d'ordonnancement. La plus ancienne a été proposée par Conway *et al.* [CMM67]. Une notation plus récente, généralement utilisée dans la littérature, a été introduite par Graham *et al.* [GLLK79]. Cette dernière est découpée en trois champs : $\alpha/\beta/\gamma$.

- Le champ α décrit la structure du problème.
- Le champ β contient l'ensemble des contraintes du problème.
- Le champ γ contient la liste des objectifs à optimiser.

Une description très complète des valeurs possibles prises par ces différents champs est proposée par T'kindt et Billaut [TB02]. Nous reprenons cette énumération dans les tables 3.1, 3.2 et 3.3. Un recensement des différents problèmes d'ordonnancement rencontrés actuellement ainsi que leurs méthodes de résolution, est proposé par Lee *et al.* [LLP97] et Pinedo [Pin95].

3.3.1.1.2 Les différentes classes de problèmes d'ordonnancement

Un problème d'ordonnancement peut se concevoir de la manière suivante : connaissant les tâches à exécuter et les machines qui peuvent exécuter ces tâches, il faut déterminer les ordres de passage sur chaque machine des différentes tâches et leurs dates de début sur chacune de ces machines. Il existe différentes classes de problèmes d'ordonnancement. L'ensemble de ces classes de problèmes d'ordonnancement est détaillé dans la table 3.1. Détaillons les problèmes d'ordonnancement "classiques" :

- Ateliers à machine unique ($\alpha_1 = \emptyset$) : on ne dispose que d'une machine, en exemplaire unique. Il s'agit d'un atelier dans lequel une seule machine pose un réel problème d'ordonnancement.
- Flow-shop ($\alpha_1 = F$) : on dispose de plusieurs machines dans l'atelier. Les tâches doivent passer sur toutes les machines, et toujours dans le même ordre (*ex : chaîne de montage*).
- Job-shop ($\alpha_1 = J$) : même problème que le Flow-shop, sauf que chaque job doit parcourir les machines dans un ordre qui lui est propre (*ex : monter les composants*

sur une carte-mère d'ordinateur).

- Open-shop ($\alpha_1 = O$) : ici, les machines sont parcourues dans un ordre quelconque par les jobs (*ex : coloration de tissu en industrie*).
- Ateliers de type mixte ($\alpha_1 = X$) : certains jobs ont un parcours de machines fixé, mais d'autres jobs n'en ont pas.
- Les problèmes d'ordonnancement et d'affectation avec étages. Les machines sont regroupées par étage, chaque étage étant voué à la réalisation d'une seule opération. Il peut n'y avoir qu'un seul étage ($\alpha_1 = P, Q, R$), ou plusieurs étages ($\alpha_1 = FH, JG, OG$ selon le type du problème).
- Les problèmes d'ordonnancement et d'affectation généralisée. Chaque opération ne peut être effectuée que sur un ensemble de machines qui lui est propre ($\alpha_1 = \{P, Q, R, S, O\}AG$).

TAB. 3.1 – Le champ α.

Champ α : $\alpha = \alpha_1\alpha_2$ ou $\alpha = \alpha_1\alpha_2, (\alpha_3\alpha_4^{(l)})_{l=1}^{\alpha_2}$			
Sous-champ α_1		Sous-champ α_2	
Valeur	Signification	Valeur	Signification
$\emptyset$	Machine unique	$\emptyset$	Le nombre de machines ou de pools est quelconque
P, Q, R	Machines parallèles identiques, proportionnelles ou quelconques	$1, 2, 3, etc.$	Le nombre de machines ou de pools est égal à $1, 2, 3, etc.$
F, J, O, X	Flow-shop, Job-shop, Open-shop, Mixed Shop		
FH	Flow-shop Hybride		
OG	Open-shop Généralisé		
JG	Job-shop Généralisé		
$\{P, Q, R\}AG$	Machines parallèles (de type P ou Q ou R) avec affectation générale		
SAG	Problème d'atelier avec affectation générale		
OAG	Open-shop avec affectation générale		
(uniquement lorsque α_1=FH)			
Sous-champ α_3		Sous-champ α_4	
Valeur	Signification	Valeur	Signification
1	Il n'y a qu'une machine dans le pool l	M	Le nombre de machines est quelconque
P, Q, R	Il y a des machines parallèles identiques, proportionnelles ou quelconques dans le pool l	$1, 2, 3, etc.$	Le nombre de machines est égal à $1, 2, 3, etc.$

3.3.1.1.3 Les contraintes

On s'intéresse ici aux contraintes explicites - celles qui ne sont pas directement implicites par la nature même du problème (par exemple, pour le problème du Flow-shop, il est implicite qu'un job ne peut s'exécuter sur une machine s'il est encore en traitement sur la machine précédente). Un recensement des différentes contraintes classiques rencontrées dans les problèmes d'ordonnancement est proposé par V. T'Kindt et J-C. Billaut [TB02], et présenté par la table 3.2.

TAB. 3.2: Le champ β.

Le champ β			
Valeur	Signification	Valeur	Signification
$prec$	Il y a des contraintes de précédence quelconques entre opérations	$tree$	Il y a des contraintes de précédence sous forme d'arbre entre opérations
$chains$	Il y a des contraintes de précédence sous forme de chaînes entre opérations	$in-tree$	Il y a des contraintes de précédence sous forme d'arbre entrant entre opérations
$out-tree$	Il y a des contraintes de précédence sous forme d'arbre sortant entre opérations	$sp-graphe$	Il y a des contraintes de précédence sous forme d'un graphe série-parallèle entre opérations
r_i	Les travaux ont des dates de disponibilité différentes	s_i	Les travaux ont une date de début souhaitée
$p_{i,j}=p$	Les travaux ont le même temps de traitement	$p_i \in [\underline{p_i};\overline{p_i}]$	Les temps opératoires des travaux sont à déterminer et appartiennent à l'intervalle $[\underline{p_i};\overline{p_i}]$
d_i	Les travaux ont des dates de fin au plus tard souhaitée	$d_i=d$	Les travaux possèdent une date de fin au plus tard souhaitée qui est commune
d_i $inconnues$	Les travaux ont des dates de fin au plus tard souhaitée qui sont à déterminer	$d_i=d$ $inconnue$	Les travaux ont des dates de fin au plus tard souhaitée qui est commune et à déterminer
$\tilde{d}_i$	Les travaux possèdent une date de fin au plus tard impérative	a_{j_1,j_2}	Il y a un décalage minimum à respecter entre deux opérations successives d'un travail autorisé
$split$	Le découpage des opérations en lots pour une exécution simultanée sur plusieurs machines est autorisée	$over$	Le recouvrement de deux opérations successives d'un travail est autorisé

suite à la page suivante ⟶

Le champ β			
pmtn	Les opérations peuvent être interrompues pour être reprises ultérieurement, éventuellement sur une autre machine	$no-wait$	Pour un travail, dès qu'une de ses opérations est terminée, l'opération suivante commence
block	L'atelier dispose d'une zone de stockage de capacité limitée entre chaque machine, ce qui peut conduire à des blocages sur les machines	*recrc*	Un travail peut être amené à utiliser plusieurs fois la même machine
batch	Les opérations peuvent être regroupées en paquets lors de leur exécution	$s-batch$	Les opérations peuvent être regroupées en paquets et sont exécutées en série dans chaque paquet
$p-batch$	Les opérations peuvent être regroupées en paquets et sont exécutées en parallèle dans chaque paquet	*permu*	On ne considère que l'ensemble des ordonnancements de permutation (valable uniquement pour les problèmes de Flow-shop)
blcg	Les machines doivent terminer leurs traitements en même temps	$unavail_j$	Les machines peuvent avoir des périodes, connues, d'indisponibilité
R_{sd} (R_{nsd})	Il y a des temps de réglage après le traitement de chaque opération. Ces temps sont fonctions (resp. ne sont pas fonction) de la séquence des opérations sur chaque machine	S_{sd} (S_{nsd})	Il y a des temps de réglage avant le traitement de chaque opération. Ces temps sont fonctions (resp. ne sont pas fonction) de la séquence des opérations sur chaque machine
nmit	Une machine ne peut être laissée volontairement inactive alors qu'elle peut traiter une opération	$no-idle$	Sur chaque machine le traitement des opérations s'enchaîne sans temps d'inactivité de la machine

TAB. 3.3 – Le champ γ.

Champ γ			
Objectif	Signification	Objectif	Signification
C_{max}	Date d'achèvement de l'ordonnancement	$\overline{C}$ ($\overline{C}^w$)	Date d'achèvement moyenne (pondérée) des travaux. Représente également les encours moyens
T_{max}	Retard maximum des travaux par rapport à leur date de fin au plus tard souhaitée	$\overline{T}$ ($\overline{T}^w$)	Retard moyen (pondéré) des travaux par rapport à leur date de fin au plus tard souhaitée
L_{max}	Le plus grand décalage des travaux par rapport à leur date de fin au plus tard souhaitée	$\overline{U}$ ($\overline{U}^w$)	Nombre de travaux en retard (pondéré) par rapport à leur date de fin au plus tard souhaitée
E_{max}	Avance maximum des travaux par rapport à leur date de fin au plus tard souhaitée	$\overline{E}$ ($\overline{E}^w$)	Avance moyenne (pondérée) des travaux par rapport à leur date de fin au plus tard souhaitée
F_{max}	Temps de présence maximum	–	Aucun critère. Problème de faisabilité
f_{max}	Fonction générique de coût maximum. Elle est généralement supposée croissante en fonction des dates de fin des travaux		

3.3.1.1.4 Les objectifs

Afin d'évaluer les ordonnancements, on dispose d'un certain nombre de critères. Le plus classique est sans doute l'objectif mesurant la date d'achèvement de l'ensemble des travaux, noté C_{max} et souvent appelé *makespan*. Les différents objectifs peuvent être formulés sous forme de fonction objectif à optimiser (comme minimiser le *makespan* par exemple), ou sous forme de contrainte à ne pas violer (par exemple, interdire les jobs en retard). La table 3.3 présente les différents objectifs généralement rencontrés dans les études de problèmes d'ordonnancement.

3.3.1.2 Les problèmes de flow-shop

On s'intéresse ici aux problèmes de type $F/\beta/\gamma$. L'éventail des différents problèmes de Flow-shop est très large, surtout si on se rapproche des applications réelles. On trouve un recensement des nombreux problèmes différents de type Flow-shop dans [Esp98, TB02].

Les problèmes de Flow-shop sont principalement divisés en deux ensembles : ceux se limitant à une ou deux machines, et ceux avec un nombre de machines M variable selon l'instance du problème. En effet, de nombreuses études considèrent les problèmes de type $F1/\beta/\gamma$ ou $F2/\beta/\gamma$, pour lequel une grande part des objectifs sont NP-difficiles. Pour M machines, tous les problèmes de type $F/\beta/\gamma$ sont fortement NP-difficiles, d'après les résultats de Tanaev *et al.* [TSS94].

3.3.1.2.1 Objectifs L'objectif très majoritairement étudié pour le Flow-shop à N jobs et M machines est le C_{max}. On définit $C_{max} = max_{i\in[1..N]}(C_i)$, avec C_i la date de fin de travail du job i sur la dernière machine. Ce problème est NP-difficile pour $M > 2$. Pour le cas $M = 2$, l'algorithme de Johnson le résout en temps polynomial, en $\theta(N * log(N))$ [Joh54].

3.3.1.2.2 Contraintes

La contrainte généralement utilisée pour l'optimisation du C_{max} consiste à autoriser uniquement les ordonnancements de permutation, c'est-à-dire les ordonnancements pour lesquels les jobs sont exécutés dans la même séquence sur toutes les machines (cela correpond à une ligne de production, par exemple). En effet, d'une manière générale, les ordonnancements de permutation sont souvent plus intéressants en terme de qualité, et plus faciles à traiter dans le cadre de l'optimisation, que les ordonnancements où l'ordre d'exécution des jobs varie selon les machines.

D'autres contraintes sont parfois ajoutées aux problèmes classiques, afin de se rapprocher ou de modéliser des problèmes réels, comme des contraintes de précédence entre les jobs, ou des périodes d'indisponibilité des machines...

3.3.1.3 État de l'art des méthodes d'optimisation pour l'ordonnancement

La liste des études auxquelles nous faisons référence dans cette section n'est pas exhaustive. Les problèmes d'ordonnancement ont fait l'objet de nombreuses études, il serait très fastidieux de toutes les énumérer. Nous nous contentons ici de survoler les différentes méthodes utilisées pour optimiser différents problèmes d'ordonnancement. Dans un premier temps, nous présentons les différentes approches exactes et métaheuristiques proposées dans la littérature. Devant l'étendue des approches recensées, nous nous restreindrons aux études portant sur les problèmes de Flow-shop. Puis nous présenterons différentes approches coopératives trouvées dans la littérature. Afin de garder un panel large des coopérations possible, nous avons étendu nos recherches aux problèmes d'ordonnancement en général.

3.3.1.3.1 Approches exactes

Quelques approches exactes ont été proposées pour la résolution de problèmes de Flow-shop dans un contexte multi-objectif. Par exemple, Sayin et Karabati ont proposé une approche de type $B\&B$ pour résoudre un problème bi-objectif de Flow-shop à deux machines [SK99]. Les objectifs proposés pour l'optimisation sont le *makespan* (C_{max}) et la

somme des dates de complétion (appelée aussi *Flowtime*) $\bar{C}$. Ils ont développé pour cela une procédure de $B\&B$ 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. Cette approche est connue sous le nom de méthode *ϵ-contrainte*.

Sivrikaya-Serifoglu et Ulusoy [SU98] proposent une approche par $B\&B$, avec agrégation des mêmes objectifs (*makespan* et *Flowtime*). D'après la notation de Graham *et al.* le problème est de type $F2//\alpha F_{max} + \beta C_{max}$ [GLLK79]. Le même problème est résolu par $B\&B$ par Yeh [Yeh99]. Yeh et Allahverdi proposent le même type d'approche, mais sur trois machines [YA04].

Notons que de nombreuses approches exactes existent dans la littérature pour résoudre des problèmes de Flow-shop à un seul objectif. De nombreux problèmes de Flow-shop différents ont été traités. Ainsi, le problème de Flow-shop avec temps de transition dépendants de la séquence (*sequence dependent setup times*), est résolu par $B\&B$ [RMB99] et $B\&C$ [RMB98] par Rios-Mercado et Bard. Kohler et Steiglitz résolvent le problème de Flow-shop à deux machines par différentes approches, exactes ou heuristiques [KS75]. Carlier & Néron proposent de résoudre de manière optimale le Flow-shop hybride [CN00]. Des contraintes de disponibilité et des durées de latence sont associées aux opérations et aux machines.

Comme pour beaucoup de problèmes dits difficiles, les méthodes exactes appliquées aux problèmes de Flow-shop restent limitées, et comme pour beaucoup de problèmes d'optimisation, les approches heuristiques sont utilisées afin de résoudre de manière approchée les problèmes de grande taille.

3.3.1.3.2 Approches par métaheuristiques

Pour la résolution des problèmes de Flow-shop de grande taille, beaucoup d'heuristiques et de métaheuristiques ont été implémentées, aussi bien pour les cas mono-objectif que multi-objectif.

On trouve de nombreuses recherches Tabou, souvent efficaces pour l'optimisation mono-objectif, comme par exemple les travaux de Pempera [GP02], Nowicki et Smutnicki [NS96], ou Yamada [Yam02]. Mais la plupart des métaheuristiques classiques ont été utilisées pour ce type de problème, comme les colonies de fourmis [YL04, Stü98a], la recherche par dispersion [NS03], la recherche locale itérée [Stü98b], le recuit simulé [PR98], les systèmes à logique floue [HW00], les systèmes immunitaires [DEO03], ou les algorithmes génétiques [YS01]. Ces derniers sont d'ailleurs souvent utilisés de manière coopérative avec une autre méthode d'optimisation.

Beaucoup d'études portant sur l'optimisation approchée de problèmes de Flow-shop multi-objectif sont des algorithmes génétiques. Murata *et al.* proposent une telle approche [MIT96]. Tagami et Kawabe proposent un *AG* multi-objectif avec partitionnement des solutions non-dominées pour optimiser le *makespan* et le retard maximal [TK98]. Néanmoins, d'autres approches ont été envisagées pour résoudre les problème de Flow-shop bi-objectif. Par exemple, Armentano et Claudio proposent une recherche Tabou [AC04], et Ruiz-Torres *et al.* proposent différentes variantes de recuit simulé [REB97].

D'une manière générale, les métaheuristiques proposées ne sont pas réellement adaptées à tous les aspects de l'optimisation : robustesse, exploration, intensification... On trouve donc également de plus en plus d'approches hybrides réalisées dans le but de pallier aux manques de chaque méthode d'optimisation.

3.3.1.3.3 Approches coopératives

Ce type d'approche est assez répandu, mais essentiellement pour l'optimisation mono-objectif. Ce sont principalement les algorithmes évolutionnaires qui sont hybridés, afin d'être améliorés par des mécanismes de recherches locales . Les algorithmes mimétiques (hybridation de type **LTH(AG(RL))**) représentent une possibilité souvent exploitée pour ce type d'approche. Un algorithme mimétique a été proposé par Ishibuchi *et al.* pour résoudre un problème de Flow-shop bi-objectif [IYM03]. Burke et Smith ont appliqué cette méthode à un problème de maintenance [BS00], Kragelund pour un problème d'emploi du temps [Kra97]. Yamada et Reeves optimisent la somme des temps de complétion du problème de Flow-shop [YR95] en utilisant une recherche Tabou comme mécanisme de recherche locale, de même que pour l'approche proposée par Zdansky et Pozivil, pour un problème de Flow-shop hybride [ZP02].

D'autres approches de type algorithme mimétique ont été proposées, avec un algorithme glouton comme mécanisme de recherche locale, comme pour les travaux de Davis pour un problème de job-shop [Dav85] et de Heijligers et Jess pour un problème particulier d'ordonnancement [HJ95].

Des approches coopératives de différents types ont été proposées pour résoudre les problèmes d'ordonnancement. Gonçalves *et al.* ont proposé une coopération de type **HRH** entre un *AG* et une recherche locale pour résoudre un problème de Job-shop [GMR02]. Reeves et Yamada résolvent un problème de Job-shop étendu par une approche coopérative entre un *AG* et un *path-relinking* (hybridation de type **LTH**) [RY98]. Dans cette étude, la recombinaison des solutions est réalisée à l'aide de générations de chemins entre elles. Yu et Liang réalisent une coopération de type **HTH** entre un *AG* et un réseau de neurones [YL01]. Feo *et al.* 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. *GRASP* a été appliqué à un problème de Flow-shop à une machine avec pénalités sur les avances des jobs [FVB91]. La coopération est de type **HRH(RL+HG)**. Dans [ABR03], Aiex *et al.* proposent un algorithme *GRASP* parallèle avec *path-relinking* comme mécanisme de diversification (coopération de type **HRH(RL+HRH(PR+HG)**). Kim *et al.* ont proposé d'hybrider un recuit simulé, une recherche Tabou et un *AG* par une coopération de type **LTH(AG(HRH(RS+RT)))** pour résoudre un problème de maintenance.

Peu d'études proposent des coopérations méta/exacte pour résoudre des problèmes d'ordonnancement. Nous pouvons citer l'approche hybride entre un *AG* et un *B&B* de Portmann *et al.* pour résoudre un problème de Flow-shop hybride [PVDD98].

Une deuxième approche coopérative méta/exacte a été proposée par Haouari et Ladhari [HL03]. Le problème résolu est de type Flow-shop de permutation, où l'objectif optimisé est la date de complétion. L'approche coopérative est de type **HRH(HS + B&B)**. 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 de $B\&B$ développés partiellement.

En ce qui concerne les approches coopératives multi-objectif pour l'ordonnancement, les approches proposées sont encore plus rares. Nous n'avons trouvé qu'une seule approche de type **HRH(MS + CF)**, proposée par T'kindt *et al.* [TMTL02], qui optimise un problème de Flow-shop bi-objectif (noté $F2//Lex(C_{max}, \sum C_i)$). Mais dans cette étude les objectifs sont traités de manière lexicographique, donc le coté multi-objectif du problème est transformé en deux problèmes mono-objectif traités de manière quasi indépendantes.

Il existe beaucoup d'approches coopératives pour résoudre les problèmes d'ordonnancement. Cependant, on peut remarquer, d'une part, que très peu font coopérer (méta)-heuristiques et méthodes exactes, et d'autre part, que ces études concernent généralement des problèmes mono-objectif.

3.3.2 Le Flow-shop de permutation multi-objectif

Bien qu'étant naturellement multi-objectif, les problèmes de Flow-shop ont souvent fait l'objet d'études sous forme mono-objectif, l'objectif étudié étant généralement le plus "classique" (date de complétion, ou somme des temps de complétion pour le cas sur plusieurs machines, somme ou maximum des retards pour le cas sur une machine). Nous allons décrire, dans un premier temps, le problème d'optimisation bi-objectif que nous étudierons par la suite. Ensuite, nous décrirons les jeux de données qui seront utilisés pour les expérimentations. Puis nous ferons quelques remarques d'ordre général sur cette version particulière du problème de Flow-shop.

3.3.2.1 Définition du problème

La variante du Flow-shop étudiée ici est une version simple parmi les nombreuses extensions existantes pour ce problème. Un problème de Flow-shop consiste à ordonnancer N jobs sur M machines, sachant que chaque machine ne peut traiter qu'un seul job à la fois. Chaque job doit être traité un certain temps sur chaque machine. De plus, tous les jobs ont la même séquence d'usinage. On peut donc comparer les machines à une chaîne de production, sachant que tous les jobs utilisent les machines dans le même ordre (contrainte de *permutation*).

On peut donc représenter un job J_i par un ensemble $J_i = \{t_{i1}, t_{i2}, ..., t_{iM}\}$ de tâches à réaliser. La tâche t_{ij} représente la $j^{ème}$ tâche du job J_i et requiert la machine m_j. Enfin, à chaque tâche t_{ij} est associée une durée d'exécution p_{ij}.

Le problème d'ordonnancement que nous allons étudier précisément est le Flow-shop de permutation de type $F/permu, d_i/(C_{max}, \bar{T})$ (d'après la notation définie par Graham *et al.* [GLLK79]). Le premier champ de la notation désigne le type de problème d'ordonnancement (ici, le Flow-shop, F). Le deuxième champ désigne les contraintes spécifiques à prendre en compte pour le problème étudié. Ici, *permu* indique qu'on étudie le Flow-shop de permutation (les jobs doivent être ordonnés dans le même ordre sur toutes les machines

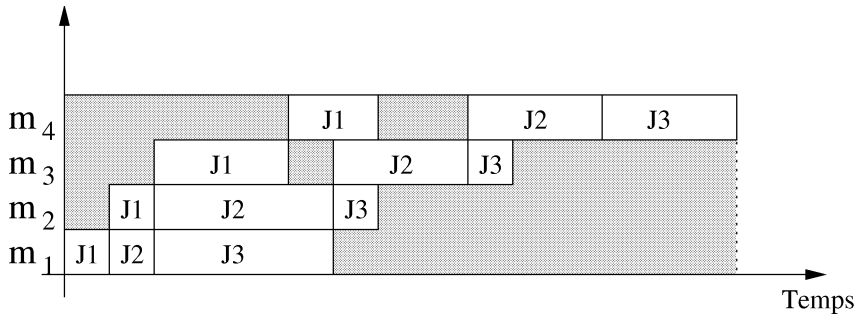


FIG. 3.15 – Flow shop de permutation. Exemple à 3 jobs et 4 machines.

(figure 3.15)). d_i indique la date de fin désirée du job i. Enfin, le troisième champ désigne les critères à optimiser. Ici, deux critères seront optimisés : la date de fin d'ordonnancement (C_{Max}), et la somme des retards ($\bar{T}$), retard de chaque job J_i par rapport à la date prévue d_i).

La date de fin d'ordonnancement peut se calculer de la manière suivante : soit une permutation $\Pi_1, \Pi_2, \ldots, \Pi_N$. Les dates de complétion $C(\Pi_i, j)$, du $i^{éme}$ job de Π sur la machine j, sont calculées de la manière suivante :

$$\begin{array}{lcl} C(\Pi_1,1) & = & p_{\Pi_1,1} \\ C(\Pi_i,1) & = & C(\Pi_{i-1},1) + p_{\Pi_i,1} \quad pour\ i = 2,\ldots,N \\ C(\Pi_1,j) & = & C(\Pi_1,j-1) + p_{\Pi_1,j} \quad pour\ j = 2,\ldots,M \\ C(\Pi_i,j) & = & max\{C(\Pi_{i-1},1), C(\Pi_1,j-1)\} + p_{\Pi_i,j} \\ & & pour\ i = 2,\ldots,N; j = 2,\ldots,M \end{array} \tag{3.17}$$

La valeur de l'objectif C_{max} correspond à la valeur suivante :

$$C(\Pi_N, M) \tag{3.18}$$

La somme des retards $\bar{T}$ peut se calculer à partir des différentes valeurs des $C(\Pi_i, M)$:

$$\bar{T} = \sum_{i=1}^{N} (Max(0, C(\Pi_i, M) - d_i)) \tag{3.19}$$

Nous traiterons ici des ordonnancement de permutation. La figure 3.16 représente les solutions optimales pour le Flow-shop classique et pour le Flow-shop de permutation (ces solutions sont optimales pour les deux critères. Ce cas est possible, étant donnée la petite taille du problème).

On voit sur cette figure que la solution optimale de permutation (*ici de 15 pour le makespan et de 1 pour le retard*) peut être moins bonne que la solution optimale classique (*ici de 14 pour le makespan et de 0 pour le retard*). D'après le résultat de Conway *et*

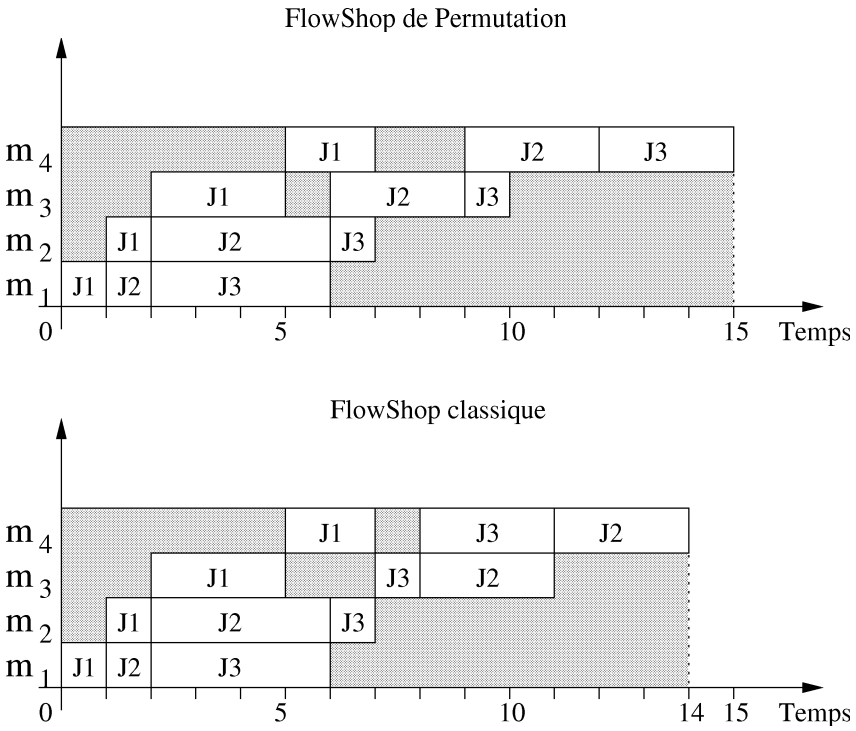


FIG. 3.16 – Flow-shop de permutation/classique. Valeurs des dates de fin souhaitées : $d_{J_1} = 9$, $d_{J_2} = 15$, $d_{J_3} = 14$.

al. [CMM67], il existe une solution de permutation optimale pour le problème général pour toute instance à 3 machines ou moins, pour l'objectif C_{max}. Au delà de trois machines, il n'existe pas forcément une solution de permutation optimale pour le problème de Flow-shop classique.

Le C_{max} est un objectif très largement étudié dans la littérature. La somme des retards (appelé aussi *retard moyen*) est aussi beaucoup étudiée, mais pour les problèmes à une seule machine. Très peu de travaux étudiant cet objectif pour M machines ont été proposés [Kim95].

3.3.2.2 Jeu de données

Pour nos expérimentations, nous prendrons comme les jeux de données de base les instances mono-objectif proposées par Taillard [Tai93a]. Pour les instances de base, 10 problèmes différents sont proposés pour plusieurs tailles de problèmes. Leur taille varie de 20 jobs à 500 jobs et de 5 à 20 machines. Les temps d'exécution des jobs sur les machines sont engendrés aléatoirement (entiers de 0 à 100).

Pour notre étude, nous avons dû étendre les instances, en ajoutant à chaque job une date de fin d'ordonnancement désirée (*due date*), ceci afin de prendre en compte l'approche bi-objectif. Ces dates ont été définies de manière aléatoire, la fourchette des valeurs pouvant

être prise par ces dates étant comprise entre le temps moyen de complétion du premier job, et la meilleure date de complétion de la littérature.

On définit le temps moyen de complétion du premier job comme étant le temps moyen d'un travail sur une machine multiplié par le nombre de machines, soit $\frac{0+100}{2} * M$, où M est le nombre de machines de l'instance.

La table 3.4 répertorie les instances bi-objectif générées[4]. La table donne également un récapitulatif des meilleures valeurs obtenues pour l'objectif C_{max} parmi les nombreuses études mono-objectif de la littérature. Ces valeurs sont régulièrement mises à jour sur la page web de Taillard[5]. Notons que ces valeurs ont été obtenues par des études mono-objectif, et que les algorithmes utilisés sont généralement très spécifiques à ce problème.

TAB. 3.4 – Instances bi-objectif pour le Flow-shop. La colonne [Tai93a] représente l'instance mono-objectif correspondante, N le nombre de jobs de l'instance et M le nombre de machines. Les trois dernières colonnes donnent la meilleure valeur obtenue dans la littérature pour le C_{max}, la référence correspondante, et si cette valeur est l'optimale prouvée.

Nom du jeu de données	[Tai93a]	N	M	Borne du C_{Max}	Ref	Exact
$ta\_20\_5\_01$	tai001	20	5	1278	[Tai93a]	OUI
$ta\_20\_5\_02$	tai002	20	5	1359	[Tai93a]	OUI
$ta\_20\_10\_01$	tai011	20	10	1582	[Tai93a]	OUI
$ta\_20\_10\_02$	tai012	20	10	1659	[Tai93a]	OUI
$ta\_20\_20\_01$	tai021	20	20	2297	[Tai93a]	OUI
$ta\_50\_5\_01$	tai031	50	5	2724	[Tai93a]	OUI
$ta\_50\_10\_01$	tai041	50	10	2991	[Vae95]	OUI
$ta\_50\_20\_01$	tai051	50	20	3850	[NS03]	NON
$ta\_100\_5\_01$	tai061	100	5	5493	[Tai93a]	OUI
$ta\_100\_10\_01$	tai071	100	10	5770	[NS96]	OUI
$ta\_100\_20\_01$	tai081	100	20	6202	[NS03]	NON
$ta\_200\_10\_01$	tai091	200	10	10862	[Vae95]	OUI
$ta\_200\_20\_01$	tai101	200	20	11195	[Vae96]	NON
$ta\_500\_20\_01$	tai111	500	20	26059	[Vae96]	NON

3.3.2.3 Étude du problème

Il existe des travaux permettant d'analyser la structure du paysage des solutions d'un problème d'optimisation. Mais peu d'études ont été proposées afin d'analyser particulièrement les paysages de problèmes multi-objectif. Cependant, afin de donner une idée de l'intérêt d'un opérateur de voisinage, on peut calculer le taux de variation du coût des individus par rapport à leur voisinage, afin d'évaluer la rugosité du paysage, ou encore de calculer le taux d'optima locaux pour évaluer la complexité du paysage [Bac99]. En engendrant aléatoirement une population de solutions, on peut également évaluer le niveau

[4] celles-ci sont disponibles à l'adresse suivante : `http ://www.lifl.fr/~basseur`.
[5] `http ://ina.eivd.ch/collaborateurs/etd/problemes.dir/ordonnancement.dir/ordonnancement.html`

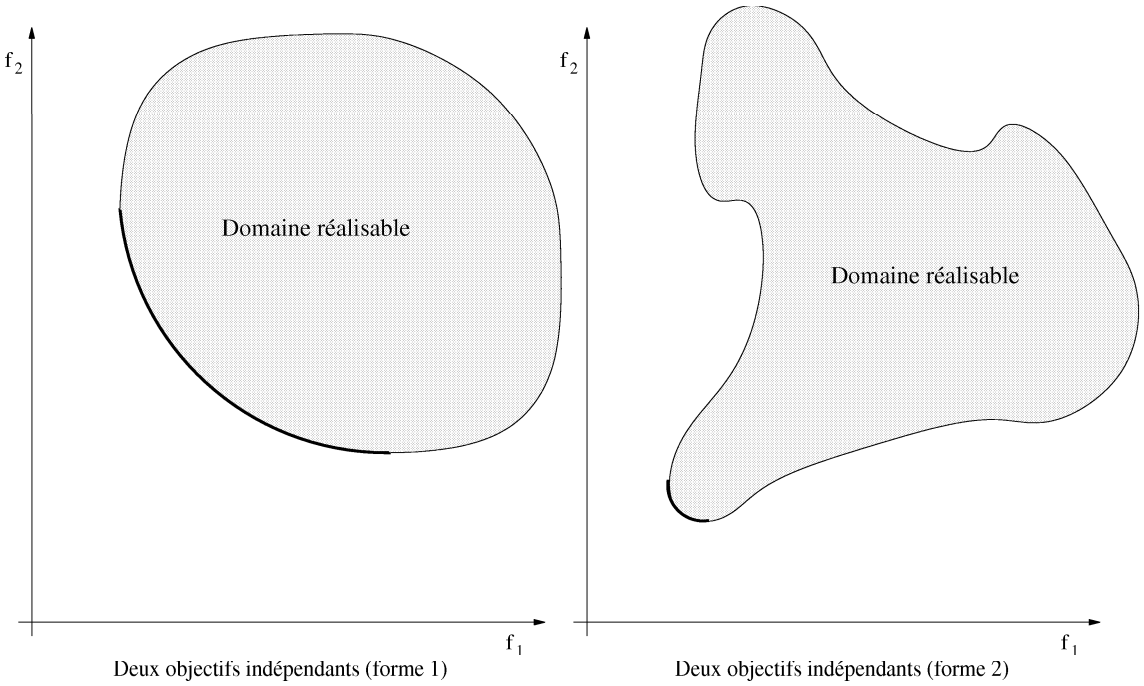


FIG. 3.17 – Lorsque les objectifs ne sont pas corrélés, l'allure et la taille du front Pareto optimal peuvent être diverses. Rien n'empêche que la valeur optimale pour un objectif soit aussi très bonne sur les autres objectifs.

de corrélation des différents objectifs. Selon l'allure du paysage phénotypique des solutions engendrées, on peut évaluer le niveau et la nature de la corrélation des objectifs (figures 3.17 et 3.18).

Nous avons engendré 100000 individus de manière aléatoire pour les instances bi-objectif que nous traiterons dans les chapitres suivants. L'allure des solutions obtenues prend toujours à peu près la même forme circulaire, mais légèrement allongée (voir exemples des figures 3.19, 3.20, 3.21 et 3.22). Cette forme indique une légère corrélation entre les objectifs. En effet, il semble naturel que, lorsque l'objectif C_{max} se dégrade, la somme des retards ait tendance à se dégrader également, les derniers jobs se terminant alors plus tard. Les fronts Pareto que nous obtiendrons seront donc *a priori* assez peu étendus au vu du paysage phénotypique de l'ensemble des solutions réalisables.

3.4 Conclusion

Les problèmes d'optimisation multi-objectif sont réputés comme difficiles. En effet, l'espace phénotypique engendré par le problème est de dimension supérieure à 1, ce qui le complexifie de manière non négligeable, puisqu'il n'existe alors plus de relation d'ordre totale pour le coût des solutions, et qu'il existe par conséquent un ensemble de solutions optimales à découvrir. Cet ensemble offre des compromis entre les différents objectifs op-

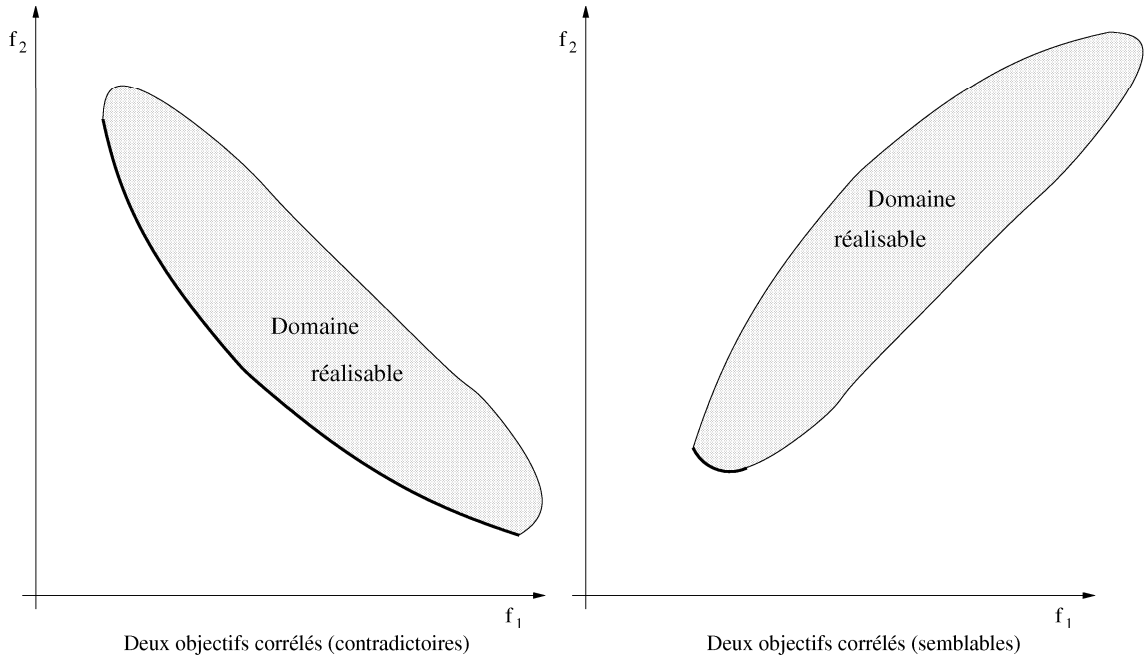


FIG. 3.18 – Lorsque les objectifs sont corrélés, le front Pareto optimal est très large si les objectifs sont en opposition, et très restreint si les objectifs sont semblables.

timisés.

Parmi les différentes approches de résolution, celles utilisant directement la notion de dominance Pareto sont les plus naturelles, bien qu'elles soient plus complexes à mettre en oeuvre. Afin d'évaluer la pertinence des résultats d'une méthode, il est nécessaire d'utiliser des opérateurs de performance, la comparaison de fronts Pareto étant souvent impossible à faire de manière directe (sauf lorsque l'on obtient le front Pareto optimal). Nous utiliserons par la suite les métriques S et $Contribution$, en plus des comparaisons graphiques et mono-objectif.

Dans la deuxième partie de ce chapitre, nous avons présenté le problème de Flow-shop bi-objectif que nous traiterons par la suite. Il fait partie de la classe des problèmes d'ordonnancement, constituée d'une multitude de problèmes différents. Ces problèmes se différencient par leur type (*Open-shop, Job-shop, Flow-shop...*), leurs contraintes, et les objectifs optimisés. Ces derniers sont nombreux pour les différents types de problèmes d'ordonnancement, qui sont naturellement multi-objectif. Le problème de Flow-shop que nous traiterons dans la suite consiste à optimiser deux objectifs classiques pour le Flow-shop : le temps de terminaison et la somme des retards (ou retard moyen). D'après la classification de Graham *et al.* [GLLK79] introduite dans ce chapitre, ce problème est de type $F/permu, d_i/(C_{max}, \bar{T})$. Nous travaillerons sur des jeux de données mono-objectif classiques, que nous avons adaptés au cas bi-objectif. Une analyse préliminaire a montré une certaine liaison des deux objectifs, ceux-ci étant légèrement corrélés.

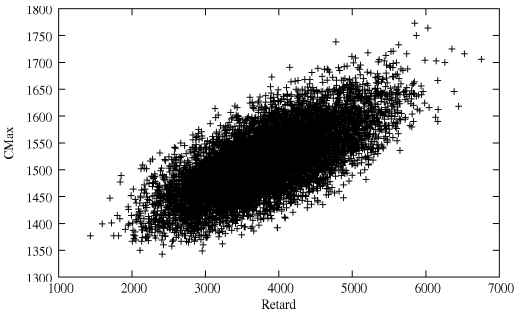


FIG. 3.19 – Paysage phénotypique de 100000 individus engendrés aléatoirement pour le problème *ta_20_5_01*.

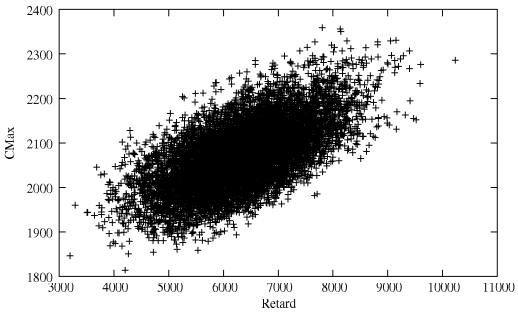


FIG. 3.20 – Paysage phénotypique de 100000 individus engendrés aléatoirement pour le problème *ta_20_10_02*.

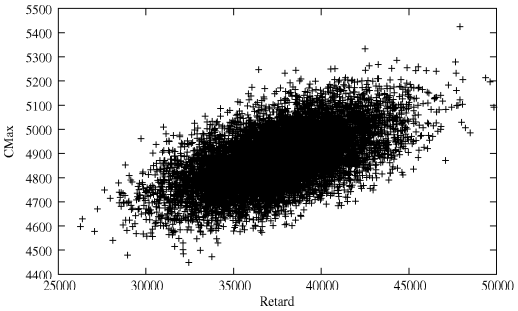


FIG. 3.21 – Paysage phénotypique de 100000 individus engendrés aléatoirement pour le problème *ta_50_20_01*.

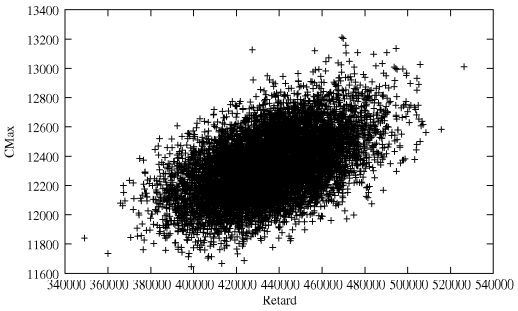


FIG. 3.22 – Paysage phénotypique de 100000 individus engendrés aléatoirement pour le problème *ta_200_10_01*.

Chapitre 4

Algorithme génétique adaptatif pour le flow-shop multi-objectif

Dans ce chapitre, nous définissons les différents opérateurs d'un algorithme génétique multi-objectif. Puis nous proposons l'algorithme AGA, utilisant deux mécanismes favorisant l'adaptivité et la capacité d'exploration des algorithmes génétiques multi-objectif. Le premier mécanisme permet d'utiliser plusieurs opérateurs de mutation simultanément durant l'AG, en favorisant les meilleurs d'entre eux. Le deuxième mécanisme influe sur le paramètrage du mécanisme de diversification, en l'adaptant selon la disposition des solutions non-dominées. Nous montrons ensuite l'intérêt apporté par ces mécanismes. Ce chapitre a fait l'objet d'une publication à la conférence CEC'02 [BST02] et d'une soumission pour publication à une revue [BST05a].

4.1 Introduction

Les algorithmes évolutionnaires, et particulièrement les algorithmes génétiques, sont bien adaptés à l'optimisation multi-objectif. En effet, l'utilisation d'une population de solutions permet de pouvoir approcher un ensemble Pareto optimal en une seule exécution. Les méthodes coopératives présentées dans les chapitres suivants utiliseront toujours l'algorithme génétique adaptatif comme base de la recherche. Le but de l'algorithme génétique présenté dans ce chapitre est bien sûr d'offrir une bonne approximation de la frontière Pareto exacte, mais surtout d'obtenir une population la plus diversifiée possible, en explorant d'une manière la plus intelligente possible l'espace des solutions. Le but, par la suite, sera d'allier la capacité d'exploration de l'algorithme génétique multi-objectif présenté, avec la vitesse de convergence d'autres méthodes d'optimisation.

Les algorithmes génétiques souffrent, d'une manière générale, de la présence de nombreux paramètres qui déterminent leur efficacité. Ces paramètres influent sur la vitesse de convergence, la capacité d'exploration, le temps de recherche... Notre but sera de faire en sorte de rendre automatique le calcul de certains paramètres, en établissant des mécanismes qui favorisent l'exploration de l'espace des solutions.

Deux des mécanismes principaux qui déterminent la capacité d'exploration de l'espace des solutions d'un algorithme génétique multi-objectif sont les opérateurs de mutation et de croisement, ainsi que la méthode de diversification. Nous proposons dans ce chapitre des mécanismes s'appliquant à ces opérateurs, afin de rendre l'algorithme génétique adaptatif et capable d'effectuer une exploration efficace de l'espace de recherche.

Dans la section 4.2 nous décrivons les différents opérateurs d'un algorithme génétique multi-objectif. Puis, dans la section 4.3, nous proposons des mécanismes de mutation et de diversification adaptative adaptés au cas multi-objectif afin de mettre au point un algorithme génétique adaptatif (AGA). Nous proposons une implémentation de ces mécanismes pour le Flow-shop bi-objectif dans la section 4.4. Enfin, nous discuterons des résultats expérimentaux dans la section 4.5, en comparant AGA avec sa version non-adaptative, mais aussi avec un algorithme classique de l'optimisation multi-objectif : $NSGA\ II$ [DAPM00]. Nous conclurons dans la section 4.6.

4.2 Description générale d'un algorithme génétique multi-objectif

Dans le chapitre 2, nous avons présenté très brièvement les algorithmes génétiques. Nous décrivons ici plus précisément chacun de leurs mécanismes génétiques. La figure 4.1 représente le fonctionnement itératif simplifié d'un AG.

4.2.1 Algorithmes génétiques multi-objectif

Les approches Pareto utilisent directement la notion de Pareto dominance entre les solutions. Cette idée a été introduite initialement par Goldberg [Gol89]. Ces approches

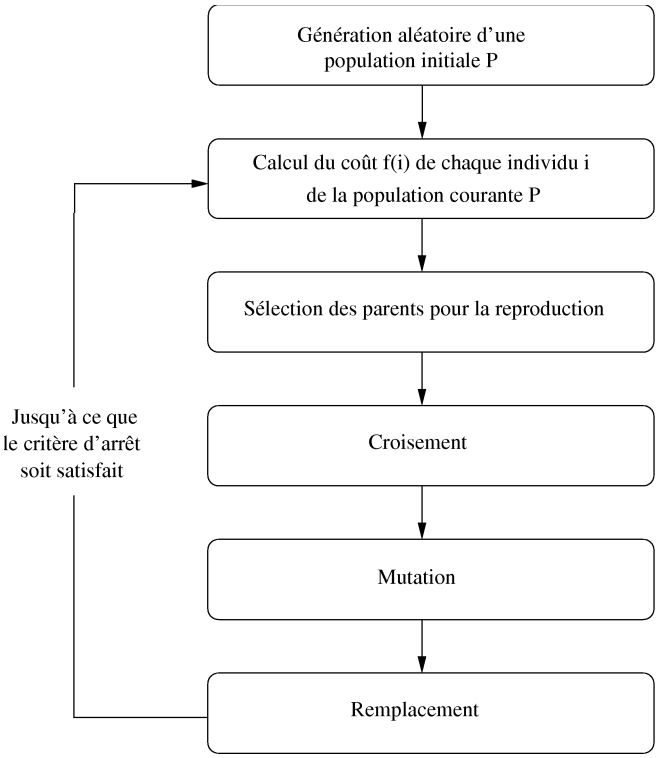


FIG. 4.1 – Fonctionnement général d'un *AG* de base.

permettent de traiter les différents objectifs de manière équitable, et de ne négliger aucune solution potentiellement intéressante pour le décideur (i.e. solution faisant partie de l'ensemble Pareto optimal). L'utilisation de ce critère de dominance a depuis permi de définir de nombreuses méthodes différentes et tend à prendre de l'ampleur au fil des années. L'utilisation de la dominance est principalement à la base du développement des algorithmes évolutionnaires multi-objectif, bien adaptés à la notion de Pareto dominance qui implique l'utilisation de solutions potentiellement optimales. Le domaine connaît un intérêt grandissant depuis le milieu des années 90, avec l'apparition de plusieurs techniques et applications utilisant des approches Pareto. Actuellement, l'intérêt porté à ces méthodes est très important comme le montre la publication de livres dédiés à ce sujet [Deb01, CVL02] ou le nombre important de publications de tout genre référencées sur la page de Coello Coello, `http ://delta.cs.cinestav.mx/~coello/EMOO`, qui comporte plus de 1900 références.

Coello Coello *et al.* [CVL02] proposent une classification simple en deux générations de méthodes. Dans la première génération, ils rangent tous les algorithmes évolutionnaires qui prennent en compte la présence de plusieurs objectifs par des approches scalaires ou des approches non scalaires et non Pareto. Les méthodes de première génération comprennent aussi les premières études utilisant des approches Pareto : *pure ranking Pareto* |Gol89|, *Multi-Objective Genetic Algorithm* (MOGA) |FF93|, *Nondominated Sorting Genetic Algorithm* (NSGA) |SD94|, *Niched-Pareto Genetic Algorithm (NPGA)* |HN93, HNG94| et NPGA 2 [EMH01]. Les méthodes de seconde génération utilisent toutes des approches

Pareto. Elles se différencient des méthodes de première génération par l'inclusion de populations secondaires et de méthodes avancées pour la génération de solutions qui sont à la fois non-dominées et uniformément distribuées. Parmi ces méthodes, on peut citer : *Pareto Archived Evolution Strategy* (PAES) [KC00], *Pareto Archived Envelope-based Selection Algorithm* (PESA) [CKO00], *Pareto Envelope-based Selection Algorithm-II* (PESA-II) [CJKO01], *Strength Pareto Evolutionary Algorithm* (SPEA) [ZT99], *Strength Pareto Evolutionary Algorithm 2* (SPEA2) [ZTL01], *Nondominated Sorting Genetic Algorithm II* (NSGA II) [DAPM00], *Multi-objective Messy Genetic Algorithm* (MOMGA) [VL00a], *Multi-objective Messy Genetic Algorithm II* (MOMGA II) [ZVL01], *Micro Genetic Algorithm for Multiobjective Optimization* [CT01].

Pour pouvoir développer un *AG* multi-objectif et résoudre efficacement un problème, il faut déterminer différents opérateurs qu'on retrouve dans les *AGs* mono-objectif. Certains opérateurs sont semblables dans les deux cas, d'autres nécessitent une interprétation propre aux problèmes multi-objectif.

4.2.2 Description des opérateurs génétiques multi-objectif

Les opérateurs génétiques multi-objectif sont, pour la plupart, semblables à leurs homologues mono-objectif. Nous survolons rapidement les différents mécanismes que l'on retrouve dans un *AG* multi-objectif. Le point principal différenciant un *AG* multi-objectif d'un *AG* mono-objectif réside dans l'affectation de l'efficacité, qui n'est pas triviale dans le cas multi-objectif.

Parmi ces mécanismes, deux ne font pas partie du schéma de base des *AGs*, mais interviennent dans une grande part des *AGs* actuels : la diversification et l'élitisme.

4.2.2.1 Représentation des solutions

La représentation des solutions doit être de préférence complète, c'est-à-dire que toutes les solutions possible du problème doivent pouvoir être codées à l'aide de cette représentation. En effet, si une solution ne peut pas être représentée, l'algorithme génétique ne pourra jamais la trouver. De même, il est préférable d'éviter de permettre la redondance de codage d'une même solution, ainsi que le codage de solutions non-réalisables. Cependant, dans certains cas, on utilise quand même ce type de codages, lorsqu'il est difficile d'obtenir des solutions réalisables.

4.2.2.2 Affectation de l'efficacité

L'affectation de l'efficacité aux individus se fait par l'intermédiaire d'une *fonction d'évaluation*, qui quantifie la qualité de chaque solution par rapport au problème.

En optimisation multi-objectif, l'efficacité des solutions est définie par plusieurs fonctions d'évaluation. Ces fonctions d'évaluation peuvent être transformées en une seule, comme pour la méthode d'agrégation des objectifs. Mais aujourd'hui de nombreux algorithmes évolutionnaires multi-objectif utilisent la notion de Pareto dominance pour affecter

l'efficacité des individus. Cette idée a été introduite initialement par Goldberg [Gol89]. On parle alors du *rang* d'un individu, et de méthode de *ranking*. Le rang ainsi affecté à un individu lui sert alors de *fitness* pour la sélection.

Une méthode proposée par Bentley et Wakefield pour affecter des rangs aux individus consiste à les classer selon chaque objectif, le rang final de l'individu étant la somme de ses places sur chaque objectif [BW97]. Quelques méthodes de *ranking* utilisant la notion de Pareto dominance ont été proposées dans la littérature, comme le rang de dominance, où le rang d'un individu correspond au nombre d'individus le dominant plus 1 [FF93]. Le compte de dominance calcule le nombre de solutions que l'individu considéré domine [ZT99]. Une dernière méthode, la profondeur de dominance, consiste à diviser la population en plusieurs ensembles d'individus qui ne se dominent pas entre eux [DAPM00]. La couche E_0 des individus non-dominés de la population P prend alors le rang 1, la couche E_1 des individus non-dominés de la population P moins les éléments de E_0 prend le rang 2... La figure 4.2 montre un exemple de cette méthode de ranking.

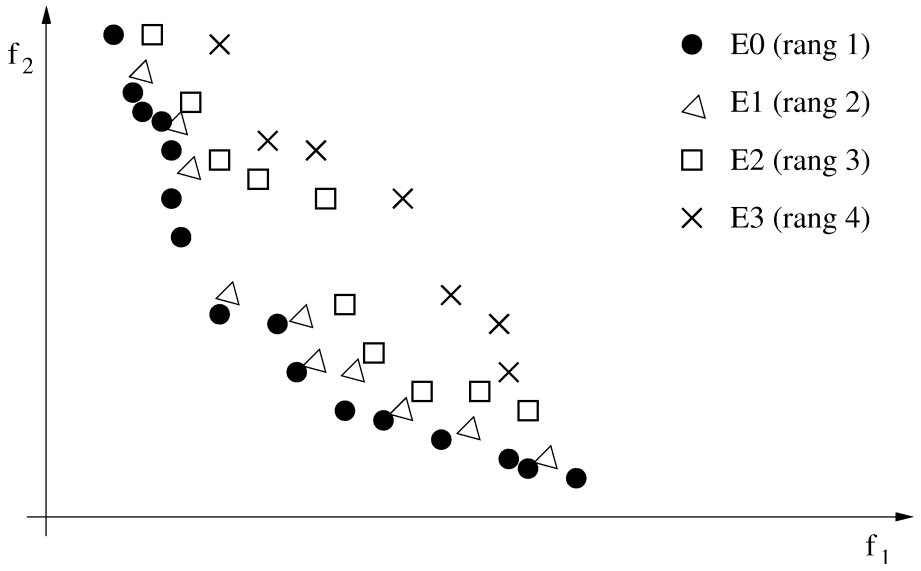


FIG. 4.2 – Profondeur de dominance.

4.2.2.3 Opérateur de sélection

Les mécanismes de sélection dans les algorithmes génétiques consistent essentiellement à favoriser les solutions les plus intéressantes en terme de convergence[1]. Dans le cadre d'une approche de résolution multi-objectif Pareto, l'opérateur de sélection n'a *a priori* pas de raison d'être modifié, les probabilités de sélection des individus dépendant de l'affectation de l'efficacité réalisée.

[1] Et éventuellement en terme de diversité, si un mécanisme de diversification est utilisé.

4.2.2.4 Opérateurs de mutation et de croisement

Les opérateurs génétiques permettent de créer de nouvelles solutions. Lors de la phase de reproduction, deux types d'opérateurs génétiques sont utilisés : la mutation et le croisement. La mutation modifie le code génétique d'un individu, le croisement crée une solution fille (souvent deux), héritant d'une partie du caractère génétique de deux solutions dites "parents".

La qualité des nouvelles solutions dépend grandement des opérateurs utilisés. En ce qui concerne les méthodes d'optimisation multi-objectif, il est nécessaire d'avoir des opérateurs performants pour l'ensemble des objectifs. Dans certains cas, il est préférable d'utiliser plusieurs opérateurs, chacun étant spécialisé pour un objectif.

4.2.2.5 Remplacement des anciennes solutions

Le remplacement des solutions peut être total ou partiel. Généralement, les meilleures solutions remplacent les plus mauvaises ; il en résulte une amélioration de la population. Lorsque la nouvelle population n'est constituée que de nouvelles solutions, on parle d'algorithme génétique générationnel, comme celui que nous présenterons par la suite.

4.2.2.6 Mécanisme de diversification

Des mécanismes de diversification sont incorporés dans la plupart des algorithmes génétiques actuels. Le but est de préserver la diversité de la population évoluant, et ainsi d'éviter une uniformisation de la population. Ce mécanisme modifie généralement l'algorithme global de manière à favoriser les solutions originales au dépend des solutions largement représentées dans la population. La diversification peut intervenir pendant l'évaluation des solutions ou pendant la phase de remplacement.

Le plus souvent, ces mesures améliorent ou détériorent la probabilité de sélection des individus. Les différentes approches prennent en compte une notion de voisinage. Il est possible de définir le voisinage d'une solution de trois manières différentes. Comme suggéré par |ZLB04|, les trois catégories sont basées sur celles utilisées en estimation de densité statistique |Sil86|.

Les méthodes à noyau définissent le voisinage d'un point i par rapport à une fonction K qui prend en paramètre la distance entre le point et un autre point. D'une manière générale, les distances d_{ij} entre i et les autres points j sont calculées et la somme des $K(d_{ij})$ effectuée. Cette somme représente la densité estimée pour l'individu i. Le *fitness sharing* |GR87| , utilisé notamment dans $MOGA$ |FF93|, $NSGA$ |SD94| et $NPGA$ |HN93|, est certainement la technique la plus populaire en calcul évolutionnaire.

Les techniques de voisin le plus proche calculent la distance entre un point donné et son $k^{iéme}$ voisin le plus proche pour estimer la densité dans son voisinage. Cette technique a notamment été utilisée dans $SPEA2$ |ZTL01|.

Une troisième catégorie utilise les histogrammes. Dans cette technique, l'espace est divisé en voisinages par une hypergrille. La densité autour d'une solution est estimée par

le nombre de solutions se trouvant dans la même case de la grille. Cette technique a été employée dans $PAES$ [KC00].

4.2.2.7 Élitisme

L'élitisme consiste à garder les meilleures solutions rencontrées au cours de la recherche. Cette sauvegarde peut être réalisée directement dans la population principale, comme pour $NSGA\ II$ [DAPM00], qui maintient forcément les solutions non-dominées dans sa population. Mais le plus souvent l'élitisme consiste à maintenir une population secondaire, appelée archive, qui est ajoutée à l'algorithme génétique. Son premier rôle consiste à éviter que les solutions non-dominées trouvées lors de la recherche ne soient perdues à cause de la nature stochastique des algorithmes génétiques. Une autre utilisation de cette archive est l'inclusion de solutions de l'archive dans la population intermédiaire lors de la phase de sélection. Cette méthode permet une amélioration de la convergence vers la frontière de Pareto.

4.3 Mécanismes adaptatifs pour les AGs

Lors de l'évaluation d'une méthode d'optimisation il est généralement nécessaire de mettre en place au préalable une série de tests déstinés à définir la valeur de différents paramètres influant sur la qualité des résultats. Ces tests sont particulièrement nombreux pour les AGs, soumis à de nombreux paramètres. Dans cette section, nous proposons des mécanismes permettant d'une part d'éviter ces phases de mise au point des paramètres, tout en gardant comme objectif principal l'exploration de l'espace de recherche. Nous nous sommes concentrés sur deux paramètres importants de la définition d'un AG.

Le premier paramètre est le choix des opérateurs de mutation et de croisement, qui influe grandement sur l'efficacité de l'AG. Souvent, le concepteur de l'algorithme dispose de plusieurs opérateurs génétiques intéressants, et doit souvent mettre en place une longue phase de tests pour déterminer le plus prometteur. Donc, nous proposons une méthode permettant l'utilisation d'un ensemble complet d'opérateurs génétiques durant l'algorithme, tout en favorisant les meilleurs d'entre eux. La méthode mise en place est inspirée du travail de Hong *et al.* [HWC00], que nous avons étendu au cas multi-objectif.

Ensuite, nous proposons de définir de manière dynamique un paramètre crucial de la méthode de diversification par *sharing* : la taille des niches. Ce paramètre définit la zone d'influence d'une solution, à partir de laquelle une autre solution est considérée comme trop proche. La méthode que nous proposons est inspirée du travail de Horn *et al.* [HNG94].

4.3.1 Mutation adaptative

Dans beaucoup d'études antérieures, un opérateur s'impose suite à une série de tests éprouvants. De plus, dans beaucoup de cas, l'opérateur sélectionné, bien que meilleur que les autres candidats, ne permet pas d'avoir toutes les qualités des opérateurs "recalés". Dans certains cas, un opérateur peut être le plus performant sur un ensemble d'instances,

et moins performant sur d'autres instances. L'efficacité d'un opérateur au regard d'un autre peut même varier au cours d'une même recherche.

Hong *et al.* [HWC00] sont partis de ces remarques pour proposer un AG appelé $DGMA$ (*Dynamic Mutation Genetic Algorithm*) permettant l'utilisation simultanée de plusieurs opérateurs de mutation, tout en favorisant l'utilisation de ceux apportant les meilleurs résultats. Ainsi, tout en favorisant les meilleurs opérateurs, les moins performants restent susceptibles d'être utilisés, dans une moindre mesure. Après avoir présenté brièvement leur approche, nous en proposons une application au cas multi-objectif.

4.3.1.1 Approche mono-objectif

Hong *et al.* [HWC00] ont proposé un mécanisme permettant l'utilisation simultanée de n opérateurs de mutations $M_1 \dots M_n$, en affectant des probabilités de sélection $P_{M_1} \cdots P_{M_n}$ à chacun d'eux.

Au départ de l'algorithme, l'application des différentes mutations s'effectue avec équi-probabilité. Mais ensuite la probabilité de sélection des opérateurs de mutation dépend des résultats obtenus à la génération précédente. Pour cela, à la fin de chaque génération, on calcule le taux de progression de chaque opérateur, les opérateurs ayant fait progresser le plus les solutions se voyant octroyer une probabilité de sélection plus forte à la génération suivante. Voici le déroulement général de l'algorithme de mutation dynamique :

1. Création de la population initiale. Initialisation de $P_{M_1} \cdots P_{M_n}$ à $\frac{1}{n * p_{mutation}}$ (n étant le nombre d'opérateurs de mutation et $p_{mutation}$ le probabilité de mutation d'un individu).
2. Faire une boucle de l'AG, en appliquant les opérateurs de mutation suivant leur probabilité de sélection. À chaque fois qu'il est décidé d'appliquer une mutation à un individu, un opérateur de mutation est choisi parmi les n opérateurs disponibles, le choix s'effectuant selon les valeurs $P_{M_1} \cdots P_{M_n}$.
3. Calculer les coûts des solutions avant et après mutation.
4. Pour chaque opérateur de mutation M_i, calculer son coefficient de progression $Progress(M_i)$.
5. Ajuster les P_{M_i} en fonction des valeurs $Progress(M_i)$.
6. Retourner à l'étape 2 tant que l'AG n'est pas terminé.

Pour le calcul des P_{M_i}, Hong *et al.* utilisent une valeur minimale δ pour éviter de perdre définitivement un opérateur de mutation. Ainsi, chacun des n opérateurs aura toujours une probabilité de sélection valant au minimum δ (sachant que δn ne dépasse pas 1) :

$$P_{M_i} = \frac{Progress(M_i)}{\sum_{j=1}^{n} Progress(M_j)} * (1 - n * \delta) + \delta \qquad (4.1)$$

La valeur $Progress$ est le gain moyen, en terme de coût, obtenu par l'application de l'opérateur de mutation M_i. L'équation 4.3 représente le gain π de la solution p issue de

l'application d'un opérateur de mutation sur la solution a, dans le cas d'une minimisation de la fonction objectif f.

$$Progress(M_i) = \frac{\sum \pi(M_{ia})}{\|M_i\|} \tag{4.2}$$

$$\pi(M_{ia}) = Max(f(p), f(a)) - f(p) \tag{4.3}$$

M_{ia} représente l'application de la mutation M_i à un individu a et $\|M_i\|$ le nombre de fois où la mutation M_i a été sélectionnée.

4.3.1.2 Application au cas multi-objectif

Pour l'application de cette méthode au cas multi-objectif, il faut redéfinir le calcul de la valeur $\pi(M_{ia})$. Nous proposons différentes méthodes.

L'idée principale des approches proposées est d'utiliser la notion de Pareto dominance pour évaluer l'effet d'une mutation. La première approche compare la solution p avec la solution initiale a. Puis nous proposons une approche utilisant la notion de *ranking* pour évaluer l'effet d'une mutation sur un individu.

4.3.1.2.1 Approche simple par relation de dominance

Pour notre première approche, nous proposons d'affecter un progrès $\pi(M_{ia})$ de 1 à l'opérateur mutation M_i lorsque celui-ci, appliqué à une solution a, permet d'engendrer une solution p dominant a. Dans le cas où a domine p, $\pi(M_{ia})$ vaudra 0. Dans les cas où a et p sont incomparables en terme de Pareto dominance, on donne à $\pi(M_{ia})$ une valeur intermédiaire (voir figure 4.3). Le principe général de l'approche reste le même que l'approche de Hong *et al.*, seule la formulation de $\pi(M_{ia})$ change. L'équation 4.4 représente ce calcul pour le cas bi-objectif.

$$\pi(M_{ia}) = \begin{cases} 0 & \text{si } p \preceq a \\ 1 & \text{si } a \preceq p \\ \frac{1}{2} & \text{sinon} \end{cases} \tag{4.4}$$

Le tableau 4.1 illustre ces calculs dans un exemple avec 4 opérateurs de mutation et $\delta = 0.05$.

Pour les problèmes à plus de deux objectifs, il semble naturel de calculer des valeurs de π intermédiaires selon le nombre d'objectifs améliorés par l'application d'un opérateur de mutation i. La formulation générale de $\pi(M_{ia})$ prend alors la forme suivante, pour le cas à N objectifs :

$$\pi(M_{ia}) = \frac{\sum_{k=1}^{N} \delta_k}{N} \quad \text{avec} \quad \delta_k = \begin{cases} 0 & \text{si } f_k(p) > f_k(a) \\ 1 & \text{si } f_k(p) \leq f_k(a) \end{cases} \tag{4.5}$$

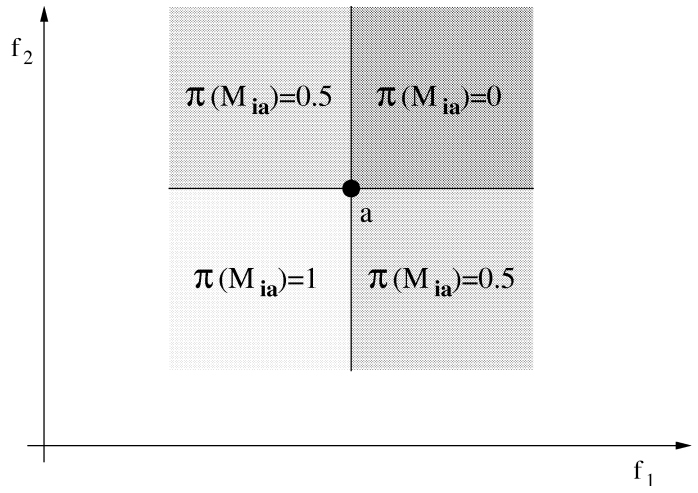


FIG. 4.3 – Mutation adaptative par relation de dominance. Calcul du gain $\pi(M_{ia})$ en fonction de la relation de dominance entre un individu a et p : cas bi-objectif.

TAB. 4.1 – Mise à jour des P_{M_i} en fonction des relations de dominance entre a et p.

Mutation	$a \preceq p$	$p \preceq a$	$a \sim p$	$Progress(M_i)$	P_{M_i}
M_1	1	6	4	3/11	0.23
M_2	0	7	0	0	0.05
M_3	3	1	7	6.5/11	0.43
M_4	4	9	0	4/13	0.29

4.3.1.2.2 Approche par *ranking*

L'approche présentée précédemment a l'avantage d'être très simple à mettre en oeuvre, puisqu'il suffit de comparer le coût des solutions avant et après application de l'opérateur de mutation, et ceci pour chaque individu concerné dans la population, afin de pouvoir mettre à jour les probabilités de sélection de chaque opérateur de mutation pour la génération suivante de l'*AG*. Cependant, une comparaison entre ces deux seules solutions n'est pas suffisante pour évaluer l'effet d'une mutation. D'une part, si la solution générée par la mutation domine la solution initiale, le calcul de $\pi(M_{ia})$ par relation de dominance ne permet pas de quantifier ce progrès. De plus, le fait que les deux solutions ne se dominent pas entre elles ne permet pas de pouvoir conclure sur la qualité de la mutation effectuée (voir la figure 4.4).

Ce problème d'appréciation de la qualité de la nouvelle solution créée peut être résolu en confrontant cette solution à la population entière de l'*AG*. Ce mécanisme dévaluation est plus "lourd", mais lorsque l'algorithme évolutionnaire multi-objectif utilise déjà la notion de *ranking*, le problème ne se pose plus. En effet, il suffit alors de comparer le rang de la solution avant et après application de l'opérateur de mutation, pour en évaluer son effet.

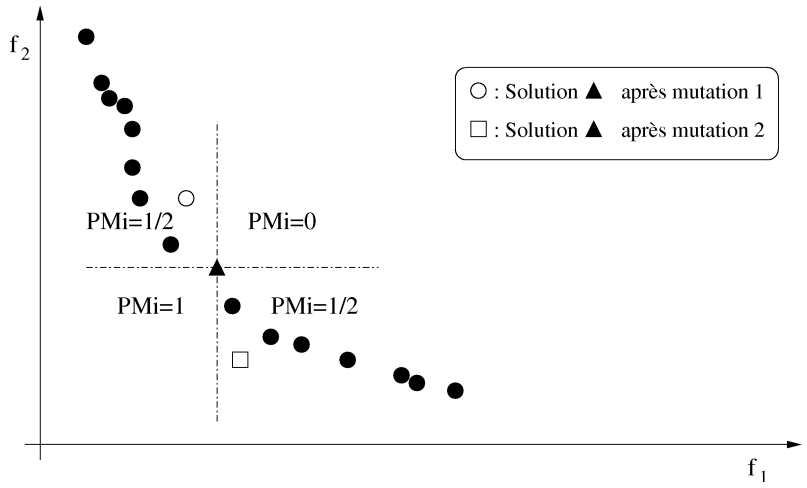


FIG. 4.4 – Mutation adaptative : faiblesse de l'approche par relation de dominance. Dans cet exemple bi-objectif, la valeur calculée pour $\pi(M_{ia})$ avec la relation de dominance est la même dans les deux cas. Pourtant, la vision globale du front montre que la mutation 2 est de loin la plus intéressante.

Soit p une solution issue de l'application de l'opérateur de mutation M_i sur la solution a, R_p et R_a leurs rangs respectifs. La formule 4.6 représente alors le nouveau calcul de $\Pi(M_{ia})$.

k est un paramètre permettant de favoriser plus ou moins les opérateurs efficaces. Plus la valeur de k est grande, plus les opérateurs performants auront une grande probabilité de sélection durant l'algorithme.

$$\Pi(M_{ia}) = \left(\frac{R_a}{R_p}\right)^k \tag{4.6}$$

En considérant le rang des solutions initiales et mutées, on compare les solutions par rapport à toute la population courante. Ainsi, il est plus facile d'évaluer l'intérêt d'une mutation, en particulier lorsque les solutions engendrées ne se dominent pas entre elles.

4.3.1.2.3 Approche par *ranking* élitiste

L'intérêt des améliorations en terme de rang n'est pas le même pour les solutions à rang faible que pour celles à rang élevé. En effet, les progrès permettant de découvrir une solution de rang 1 sont plus importants que ceux permettant de découvrir une solution de rang 2, ... (Voir figure 4.5). Afin de favoriser au mieux les mutations permettant la découverte de solutions de faible rang, nous avons introduit dans la formule 4.6 un facteur d'élitisme $C_{I_{M_i}}$:

$$\Pi(I_{M_i}) = C_{I_{M_i}} * \left(\frac{R_{I_{M_i}}}{R_I}\right)^k \text{ avec } C_{I_{M_i}} = \frac{1}{R_{I_{M_i}}} \tag{4.7}$$

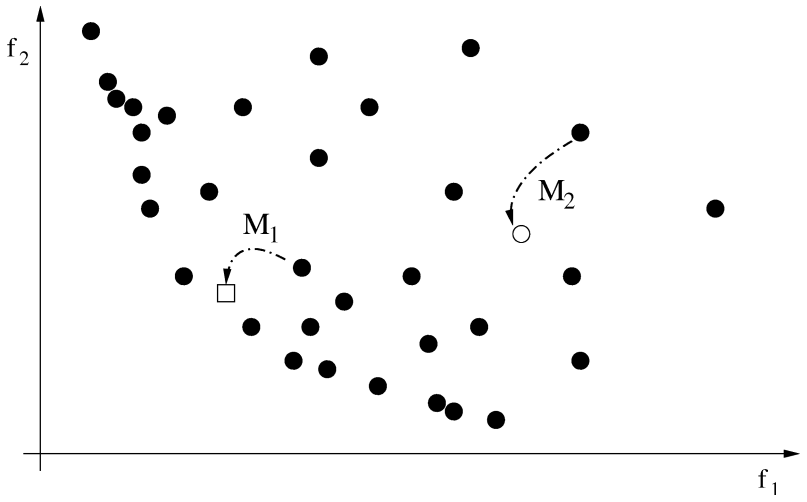


FIG. 4.5 – Mutation adaptative : notion d'élitisme. Les deux mutations font progresser l'individu sélectionné en terme de rang. Cependant l'apport réalisé par la mutation M_1 est plus conséquent que celui apporté par M_2.

Pour cette version du calcul de $\Pi(I_{M_i})$, nous avons donc besoin de modifier légèrement la formulation de $Progress(M_i)$:

$$Progress(M_i) = \frac{\sum \Pi(I_{M_i})}{\sum C_{I_{M_i}}} \tag{4.8}$$

Nous utiliserons cette version élitiste de la mutation adaptative pour nos expérimentations. Notons que l'utilisation du rang des solutions pour le calcul des $\Pi(I_{M_i})$ est applicable sans modification à des problèmes à plus de deux objectifs.

4.3.2 *Fitness sharing* adaptatif

Dans cette section, nous présentons en premier lieu le *fitness sharing*, mécanisme de diversification basé sur la distance entre le point considéré et le reste de la population. Puis nous proposons une méthode permettant de définir dynamiquement durant l'*AG* le paramètre principal de cette mesure de diversité.

4.3.2.1 Le *sharing*

La diversification réalisée avec la méthode de *sharing* (ou une autre méthode) est une partie importante de l'algorithme génétique (voir figures 4.6 et 4.7). Elle permet d'éviter de perdre la diversité dans l'ensemble des solutions. Le maintien de cette diversité permet d'approcher le front de Pareto dans toute son étendue.

Le *sharing* a été initialement proposé par Goldberg et Richardson |GR87|. Son principe consiste à dégrader le coût des individus appartenant à des régions de l'espace de recherche

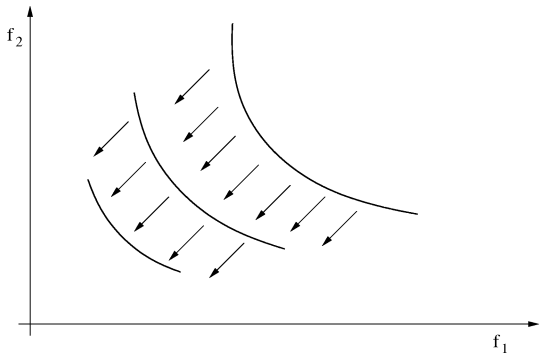


FIG. 4.6 – Progression du front de Pareto sans maintien de diversité.

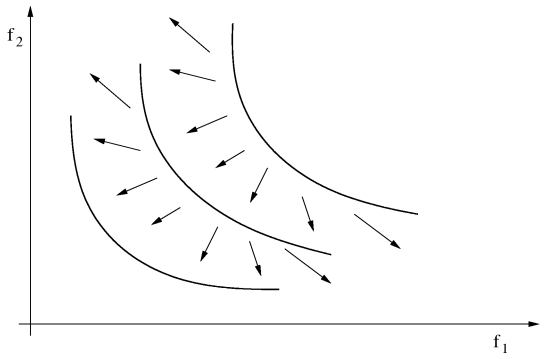


FIG. 4.7 – Progression du front de Pareto avec maintien de diversité.

où il y a une grande concentration de solutions. Pour cela on crée des niches écologiques autour de chaque individu. Le coût d'un individu est alors pénalisé pour chaque solution se trouvant dans sa niche.

La pénalisation du coût des solutions se réalise à l'aide d'une *fonction de partage*. Cette notion de partage a été présentée également par Goldberg et Richardson. Elle consiste à déterminer le voisinage et le degré de partage pour chaque individu de la population. On dégrade le coût d'un individu en fonction du nombre d'individus similaires dans la population. Le nouveau coût f' est alors égal à :

$$f'(x) = \frac{f(x)}{m(x)} \tag{4.9}$$

$m(x)$ est le compteur de niche de l'individu x. Il vaut la somme des fonctions de partage (sh) entre l'individu x et la population entière P :

$$m(x) = \sum_{y \in P} sh(dist(x,y)) \tag{4.10}$$

$dist(x,y)$ représente un calcul de distance entre deux solutions x et y. La fonction de partage doit posséder les propriétés suivantes :

- $0 \leq sh(x) \leq 1$
- $sh(0) = 1$
- $lim_{x \to \infty} sh(x) = 0$

Celle utilisée généralement est la fonction de partage proposée par Goldberg et Richardson [GR87] :

$$sh(dist(x,y)) = \begin{cases} 1 - (\frac{dist(x,y)}{\sigma_{share}})^{\alpha} & \text{si } dist(x,y) < \sigma_{share} \\ 0 & \text{sinon} \end{cases} \tag{4.11}$$

La distance entre les solutions peut être calculée de deux manières différentes :
- Dans l'espace décisionnel : les différences se font au niveau du codage des individus. On parle alors de *Sharing Génotypique*.
- Dans l'espace des objectifs : les différences se font au niveau du *fitness* des individus. On parle alors de *Sharing Phénotypique*.

La fonction de partage possède également deux paramètres : σ_{share} et α. σ_{share} est la taille des niches. Ainsi, si la distance entre deux individus est supérieure à σ_{share} ils n'affectent pas leur coûts respectifs. Et si un individu se trouve éloigné d'une distance supérieure à σ_{share} de tous les individus de la population, son compteur de niche vaudra 1 (somme de 0, plus 1, car l'individu est identique à lui-même), et son coût restera inchangé. Le deuxième paramètre est α, et permet de réguler la forme de la fonction de partage, c'est-à-dire le calcul de pénalisation lorsque deux solutions se trouvent dans la même niche.

La définition du paramètre σ_{share} est une partie primordiale déterminant l'efficacité de la diversification réalisée durant l'*AG*. En effet, l'utilisation d'un σ_{share} trop petit peut provoquer une convergence des solutions vers un sous-ensemble de la frontière Pareto optimale, car le mécanisme de diversification influe alors trop peu sur la sélection. Par contre, l'utilisation d'un σ_{share} trop grand gène l'évolution de l'algorithme, car les niches sont facilement surpeuplées, de telle sorte que même les bonnes solutions risquent d'être très pénalisées au moment de la sélection.

4.3.2.2 Taille adaptative des niches pour le *Fitness Sharing* phénotypique

La méthode que nous proposons est inspirée du travail de Horn *et al.* [HNG94]. Il faut prendre en compte le fait que l'on ne connaît pas l'allure du front de Pareto optimal exact, donc les calculs du σ_{share} optimal aura pour but de répartir équitablement les solutions de la population en fonction du front Pareto courant.

Horn *et al.* proposent de considérer les ensembles Pareto optimaux comme étant continus. Ainsi on peut envisager une répartition idéale des solutions sur cet espace, chaque solution se trouvant à une distance σ_{share} de ses voisines. Si l'on considère des niches de rayon $\frac{\sigma_{share}}{2}$ autour de chaque solution, on peut décrire une disposition idéale des solutions afin que les niches ne se superposent pas (figure 4.8). Ainsi, si l'on considère une population de N individus, et que l'on connaît l'aire $Aire_{pareto}$ de la frontière de Pareto[2], on peut calculer l'aire optimale $Aire_{niche}[\sigma_{share}]$ de l'intersection des niches avec le front de Pareto.

$$Aire_{niche}[\sigma_{share}] \simeq \frac{Aire_{pareto}}{N} \tag{4.12}$$

Dans le cas où les niches sont des hypercubes, on a l'égalité $Aire_{niche}[\sigma_{share}] = \frac{Aire_{pareto}}{N}$, mais dans le cas d'hypersphères, cette égalité est vérifiée à un facteur près α. Cette valeur vaut le rapport entre le volume d'une hypershère de dimension $n-1$ et de rayon r et un hypercube de dimension $n-1$ et de côté $2r$. Le volume d'un hypercube de dimension n est de $(2r)^n$, celui d'une hypersphère est :

[2]Le terme *Aire* est utilisé, car dans un espace à n dimensions (n attributs.), la surface du front est de dimension $n-1$.

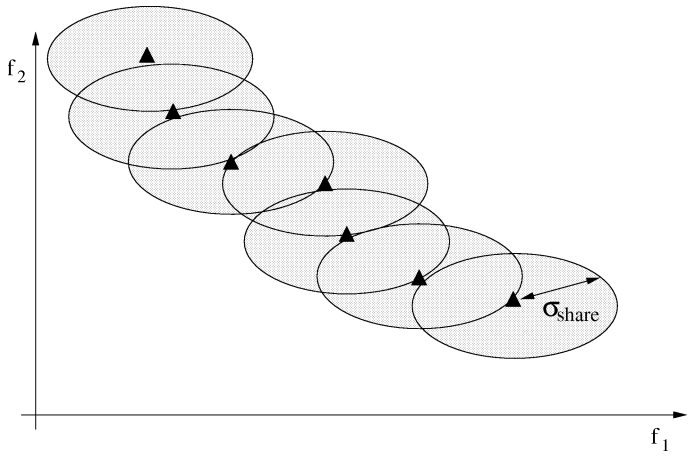


FIG. 4.8 – σ_{share} pour une disposition idéale des solutions dans l'espace.

$$Volume_{HyperSphère} = \begin{cases} \frac{\pi^p}{p!} * r^n & \text{si } n = 2p \\ \frac{2^n.\pi^p.p!}{n!} * r^n & \text{si } n = 2p+1 \end{cases} \quad (4.13)$$

Donc α vaut :

$$\alpha \begin{cases} \frac{\pi^p}{2^n.p!} & \text{si } n = 2p \\ \frac{\pi^p.p!}{n!} & \text{si } n = 2p+1 \end{cases} \quad (4.14)$$

TAB. 4.2 – Valeur du coefficient α selon le nombre d'objectifs envisagés.

n	α	n	α
1	1	5	0.16
2	0.79	6	0.08
3	0.52	7	0.037
4	0.31	8	0.016

Les valeurs de α sont proches de 1 tant que n est petit (Voir tableau 4.2). Mais lorsque n augmente, α tend vers 0. Donc, si le nombre d'objectifs est grand, l'aire d'une niche hypersphérique devient négligeable par rapport à l'aire de la niche hypercubique équivalente.

On peut donc approcher $(\sigma_{share})^{n-1}$. On a en effet $(\sigma_{share})^{n-1} \approx \frac{Aire_{pareto}}{N}$, car $Volume_{niche}[\sigma_{share}] = (\sigma_{share})^n$, et donc l'intersection de $Volume_{niche}[\sigma_{share}]$ avec le front est approché par $(\sigma_{share})^{n-1}$ [3]. Pour le cas bi-objectif, on a :

$$\sigma_{share} \approx \frac{Aire_{pareto}}{N} \quad (4.15)$$

[3] En considérant que les niches sont des hypercubes.

Il nous reste à obtenir une approximation de $Aire_{Pareto}$. Fonseca et Fleming proposent une méthode pour approcher cette aire [FF98]. On maximise l'espace des solutions non-dominées par un hypercube de dimension le nombre d'objectifs n. Cet hypercube est défini par deux points opposés m et M, de coordonnées respectives correspondant aux plus mauvaises (respectivement meilleures) valeurs trouvées pour chaque objectif (figure 4.9).

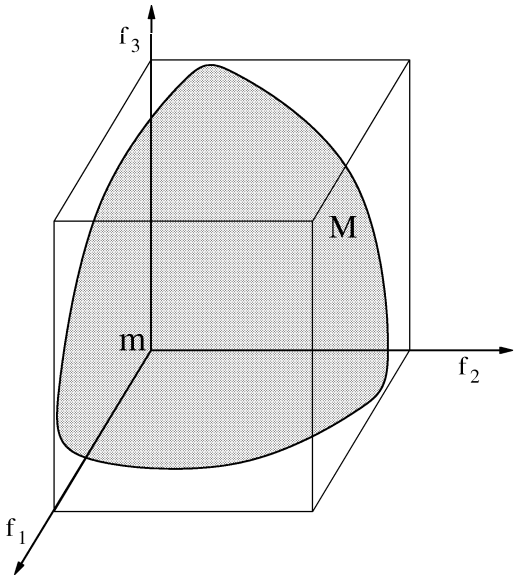


FIG. 4.9 – Approximation de l'espace des solutions Pareto optimales par un hypercube : exemple dans le cas de la minimisation de trois objectifs.

L'aire formée par les faces de l'hypercube contenant le point m constitue alors une borne maximale pour l'aire du front de Pareto de l'espace des solutions. L'aire, de dimension $N-1$, vérifie donc l'égalité suivante :

$$Max_{Aire_{Pareto}} = \sum_{i=1}^{n} \prod_{j=1, j\neq i}^{n} (M_j - m_j) \tag{4.16}$$

On peut donc en déduire une borne maximale de cette aire dans le cas bi-objectif :

$$Max_{Aire_{Pareto}} = |M_1 - m_1| + |M_2 - m_2| \tag{4.17}$$

Malheureusement, on ne peut définir une borne minimale de l'aire de la surface de Pareto. Cette borne n'est calculable que si l'espace des solutions est continu sur la frontière de Pareto. En supposant cette hypothèse vérifiée, on peut définir cette aire comme étant le chemin le plus direct joignant les deux points m et M. Cette borne Min vaut, dans le cas bi-objectif (Min vaut tout simplement la distance entre m et M. Si $n > 2$, alors cette borne Min est nulle) :

$$Min_{Aire_{Pareto}} = \sqrt{|M_1 - m_1|^2 + |M_2 - m_2|^2} \tag{4.18}$$

En reprenant la formule de Horn *et al.* [HNG94] et la borne maximale de l'aire Pareto définie par Fonseca, on a donc une borne maximale pour le rayon de niche optimal :

$$\sigma_{share} \leq \sqrt[n-1]{\frac{\sum_{i=1}^{n} \prod_{j=1,j\neq i}^{n} (M_j - m_j)}{n}} \tag{4.19}$$

Appliqué au cas bi-objectif, on a donc un encadrement de la valeur optimale de σ_{share} (en considérant toujours que l'espace des solutions est monotone sur le front). Cet encadrement est borné selon les deux formes extrêmes du front, comme le montre la figure 4.10. Les valeurs de ces bornes sont représentées par les équations 4.20 et 4.21.

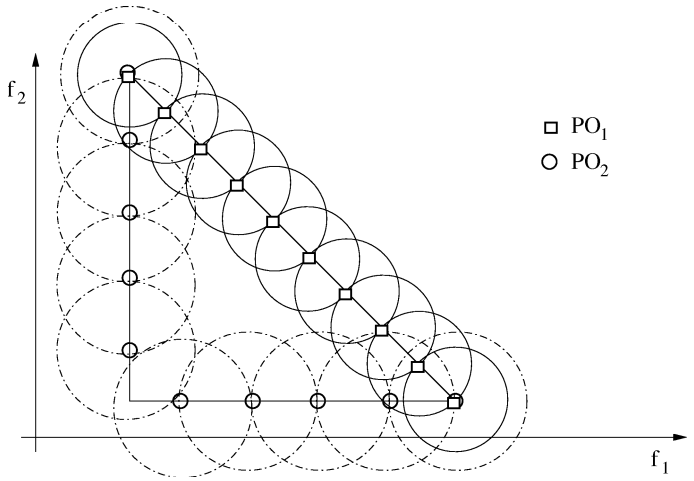


FIG. 4.10 – Taille idéale des niches variable en fonction de l'allure du front de Pareto : représentation des deux cas extrêmes dans le cas bi-objectif.

$$\sigma_{share} \leq \frac{|M_1 - m_1| + |M_2 - m_2|}{N} \tag{4.20}$$

$$\sigma_{share} \geq \frac{\sqrt{|M_1 - m_1|^2 + |M_2 - m_2|^2}}{N} \tag{4.21}$$

Pour le calcul de σ_{share} pour le *sharing* génotypique, les approximations précédentes ne peuvent évidemment pas être utilisées (en particulier, on ne peut associer de valeur au code génétique d'un individu, donc il y a impossibilité de trouver les extremums). Cependant, il existe tout de même des méthodes tentant de calculer ce paramètre [Gol89].

4.4 Implémentation

Nous avons appliqué au Flow-shop bi-objectif les mécanismes présentés dans la dernière section. Ici, nous détaillons les différents mécanismes et opérateurs utilisés dans *AGA*. Afin

de déterminer les autres opérateurs utilisés dans AGA, nous nous sommes appuyés sur un travail de comparaison des opérateurs génétiques multi-objectif classiques, qui a été réalisé antérieurement à cette thèse par Talbi *et al.* [TRMD01].

4.4.1 Codage des individus

Dans le cas général du Flow-shop, il faut représenter une solution par la suite des jobs s'exécutant sur chaque machine. Mais comme on considère ici le Flow-shop de permutation (chaque machine exécute la même séquence de jobs - voir figure 3.15), il suffit de présenter le séquencement des jobs sur une seule machine, ce qui rend la représentation beaucoup plus simple (voir figure 4.11).

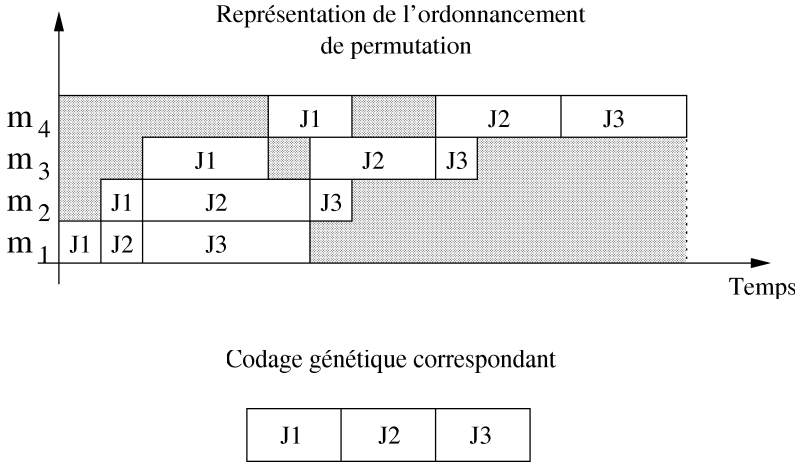


FIG. 4.11 – Représentation chromosomique pour le Flow-shop de permutation.

4.4.2 Initialisation

L'initialisation de la population courante d'un AG est généralement aléatoire, et parfois issue d'un algorithme glouton et stochastique. Pour notre problème, il paraît peu envisageable de mettre au point un algorithme glouton capable de fournir de bonnes solutions pour les deux objectifs.

4.4.3 Sélection

Nous utilisons la méthode de *ranking* de profondeur de dominance, détaillée dans la section 4.2.2.2. Soit r_i le rang de l'individu i, N la taille de la population. Soit rg_k le nombre d'individus de la population ayant le rang k. On définit R_k :

$$R_k = 1 + rg_k + 2\sum_{i=1}^{k-1} rg_i \tag{4.22}$$

La probabilité qu'un individu de rang n soit sélectionné est alors la suivante (S étant un paramètre définissant la pression de sélection) :

$$p_n = \frac{S(N+1-R_n)+R_n-2}{N(N-1)} \tag{4.23}$$

Cette méthode de ranking, dans sa version avec un nombre d'objectifs quelconque est lourde en temps de calcul. En effet, la complexité de la méthode générale la plus efficace de la littérature est en $\Theta(n^2\ log\ n)$ [DAPM00], ce qui la rend peu praticable pour les AGs. Cependant, dans le cas bi-objectif, il est possible d'attribuer les rangs des différentes solutions en $\Theta(n\ log\ n)$ en moyenne et en $\Theta(n^2)$ dans le pire des cas. Il suffit pour cela d'effectuer un tri sur les objectifs, comme indiqué dans l'algorithme 4.1, qui est illustré dans la figure 4.12. Nous utilisons donc cette méthode de *ranking* pour notre application, qui est bi-objectif. Nous préconisons une méthode de *ranking* moins coûteuse pour les problèmes à plus de deux objectifs.

Algorithme 4.1 Fonction de *ranking* rapide.

Q : ensemble des solutions non traitées.
$tri(A)$: cette fonction trie les solutions de l'archive A de manière croissante selon le premier objectif ; si plusieurs solutions ont la même valeur pour le premier objectif, elles sont triées par rapport au second objectif de manière décroissante. Les solutions sont notées s_i avec $i = 1, \ldots, |A|$.
$tri(A)$.
$Q \leftarrow A$.
$r \leftarrow 1$.
tant que $Q \neq \emptyset$ **faire**
  $s \leftarrow s_i$ tel que $s_i \in Q$ et $\nexists\ j \in Q$ tel que $s_j \in Q$ avec $j < i$.
  $s.rang \leftarrow r$.
  $meilleur \leftarrow f_2(s)$.
  $Q \leftarrow Q \backslash \{s\}$.
  pour $i = 1, \ldots, |A|$ **faire**
    si $((s_i \in Q) \wedge (f_2(s_i) < meilleur))$ **alors**
      $s_i.rang \leftarrow r$.
      $meilleur \leftarrow f_2(s)$.
    fin si
  fin pour
  $r \leftarrow r + 1$.
fin tant que

4.4.3.1 Élitisme

L'élitisme mis en place dans notre AG consiste à garder, en plus de la population courante, une population archive PO^* où se trouvent l'ensemble des solutions Pareto optimales trouvées et jamais dominées durant toute la recherche. Cette population participe à la sélection et donc à la reproduction. A est un paramètre influant sur l'intensité de l'élitisme.

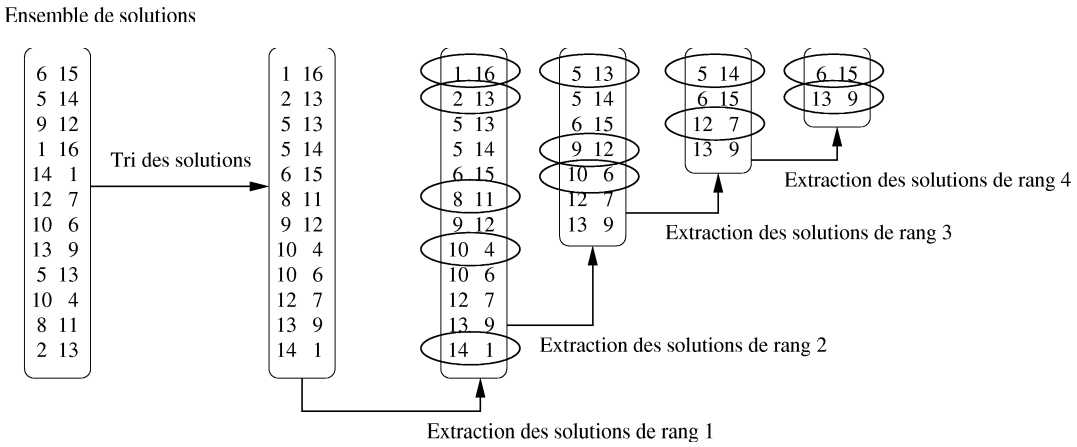


FIG. 4.12 – Application de la méthode de *ranking* rapide bi-objectif. On trie les solutions, afin d'extraire les solutions non-dominées en temps linéaire. Cette extraction se réalise en parcourant les solutions et en sélectionnant celles qui font décroître le deuxième objectif.

On donne une probabilité A/N de choisir un élément de PO^*, A/N correspondant à la proportion d'individus de PO^* voulus dans la population totale, et la probabilité qu'un individu de rang n soit sélectionné devient :

$$p_n = \frac{N-A}{A} \cdot \frac{S(N+1-R_n)+R_n-2}{N(N-1)} \tag{4.24}$$

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. Dans notre cas, et dans la perspective des coopérations à venir pour améliorer la convergence de l'algorithme, il est préférable de prendre une faible valeur de A.

4.4.4 Opérateurs génétiques

Nous présentons ici les opérateurs de croisement et de mutation utilisés dans AGA appliqué au $BOFSP$. Pour la mutation, nous en présentons plusieurs, dans le but d'appliquer le mécanisme de mutation adaptative.

4.4.4.1 Opérateur de croisement

L'opérateur de croisement que nous utilisons est le croisement deux points proposé par Ishibuchi et Murata [IM98]. Cet opérateur consiste au choix aléatoire de deux points de croisement. L'individu fils est alors défini en conservant les extrémités de chromosome du premier parent, et en complétant avec les jobs manquants, suivant leur ordre d'apparition dans le deuxième parent.

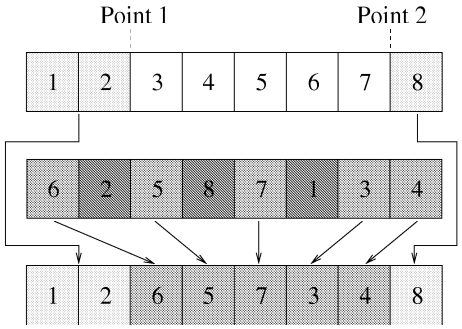


FIG. 4.13 – Opérateur de croisement.

4.4.4.2 Opérateurs de mutation

L'opérateur de mutation que nous utilisons dans notre algorithme de base est celui utilisé par Ishibuchi et Murata dans leur étude sur le Flow-shop [IM98]. Cet opérateur est couramment utilisé pour le Flow-shop, et donne de bons résultats [RY98]. On définit aléatoirement deux points sur l'individu et on réalise une permutation des gènes entre ces deux points (figure 4.14). Soit Π_N l'espace des solutions de taille N. Soit une solution $\pi \in \Pi_N$, la formule 4.25 est associée à l'opérateur, communément appelé $Shift$.

$$Shift(i,j,\pi) : [1..N],[1..N],\Pi_N \mapsto \Pi_N \quad \begin{cases} \pi_k \mapsto \pi_{k+1} & si \quad i \leq k < j \\ \pi_j \mapsto \pi_i & \\ \pi_k \mapsto \pi_k & sinon \end{cases} \tag{4.25}$$

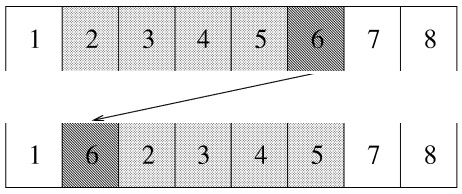


FIG. 4.14 – Opérateur de mutation $Shift$.

Beaucoup d'autres opérateurs pourraient être utilisés. Lors de nos premières expérimentations sur le Flow-shop bi-objectif, nous avons remarqué que deux opérateurs de mutation au moins se révélaient intéressants : le *Shift* et le *Exchange*.

Dans le cadre de l'application du mécanisme de mutation adaptative au $BOFSP$, définissons d'autres opérateurs qui sont utilisés dans AGA. Ces opérateurs sont tout d'abord $Exchange$ (figure 4.15), mais aussi $Random$ (figure 4.16) et $Inverse$ (figure 4.17). La définition de ces opérateurs est détaillée dans les équations 4.26, 4.27 et 4.28. D'autres opérateurs peuvent être envisagés, comme l'échange de deux jobs voisins, la permutation de blocs. Nous n'avons pas choisi nécessairement les opérateurs les plus performants, mais

plutôt des opérateurs qui engendrent des voisinages assez différents les uns des autres, dans un souci de gain en terme de diversification de la population.

$$Exchange(i,j,\pi) : [1..N],[1..N],\Pi_N \mapsto \Pi_N \quad \begin{cases} \pi_i \mapsto \pi_j \\ \pi_j \mapsto \pi_i \\ \pi_k \mapsto \pi_k \quad si \quad k \neq i,j \end{cases} \tag{4.26}$$

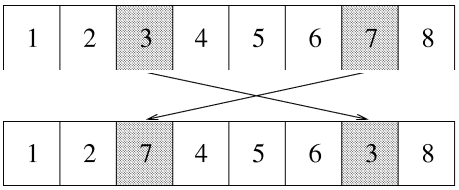


FIG. 4.15 – Opérateur de mutation *Exchange*.

$$Random(i,j,\pi) : [1..N],[1..N],\Pi_N \mapsto \Pi_N \quad \begin{cases} si \quad i \leq k \leq j, \quad \pi_k \mapsto \pi_{k'} \\ \quad tq \ \forall l \in [i \ldots j], \pi_{k'} \neq \pi_l \\ \pi_k \mapsto \pi_k \quad sinon \end{cases} \tag{4.27}$$

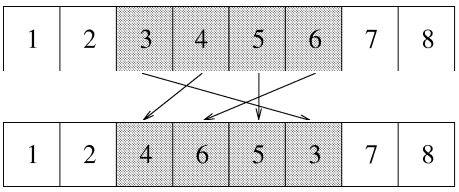


FIG. 4.16 – Opérateur de mutation *Random*.

$$Inverse(i,j,\pi) : [1..N],[1..N],\Pi_N \mapsto \Pi_N \quad \begin{cases} \pi_k \mapsto \pi_{j-(k-i)} & si \quad i \leq k \leq j \\ \pi_k \mapsto \pi_k & sinon \end{cases} \tag{4.28}$$

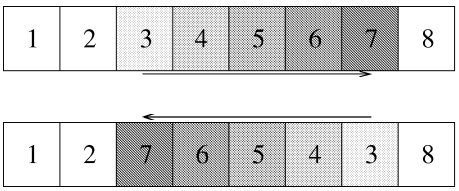


FIG. 4.17 – Opérateur de mutation *Inverse*.

En ce qui concerne la méthode de *ranking* utilisée pour le mécanisme de mutation adaptative, nous prendrons celle utilisée pour l'AG : la profondeur de dominance. Cela permet d'éviter certains calculs redondants. Ainsi, pour évaluer le rang d'une solution mutée S_{M_i} sans recalculer les rangs des autres solutions, il suffira de chercher rg, le rang maximal des solutions dominant S_{M_i}. Ainsi le rang de la solution S_{M_i} prendra la valeur $rg+1$.

4.4.5 Diversification

Nous utilisons la méthode de diversification par *Sharing combiné* qui consiste à considérer en même temps le *Sharing génotypique* et le *Sharing phénotypique*. Des expérimentations réalisées sur le problème que nous étudions ici ont montré que le sharing phénotypique donne de meilleurs résultats que le sharing génotypique, mais celui-ci apporte plus de diversité [TRMD01]. La méthode proposée combine les deux. La fonction de partage est alors modifiée, pour prendre la forme suivante :

$$sh(dist(x,y)) = \begin{cases} 1-(\frac{d_1(x,y)}{\sigma_{share}}) & \text{si } d_1(x,y) < \sigma_{share}, d_2(x,y) \geq \sigma'_{share} \\ 1-(\frac{d_2(x,y)}{\sigma'_{share}}) & \text{si } d_1(x,y) \geq \sigma_{share}, d_2(x,y) < \sigma'_{share} \\ 1-(\frac{d_1(x,y)d_2(x,y)}{\sigma_{share}\sigma'_{share}}) & \text{si } d_1(x,y) < \sigma_{share}, d_2(x,y) < \sigma'_{share} \\ 0 & \text{sinon} \end{cases} \quad (4.29)$$

La mise en place d'un tel mécanisme nécessite la mise en oeuvre de deux opérateurs de distances, l'un dans l'espace objectif d_1, l'autre dans l'espace décisionnel d_2.

La distance phénotypique entre deux individus est assez naturelle puisque le coût des individus est représenté par deux nombres :

$$d_1(x,y) = (\sum_{i=1}^{N} |x_i - y_i|^p)^{\frac{1}{p}} \quad (4.30)$$

Cette définition de la distance est paramétrée par p. Si $p=2$, cette formule définit la distance *Euclidienne*, si $p=1$, c'est la distance de *Manhattan* qui est définie. Il parait naturel de définir p à 2, mais Horn *et al.* montrent l'intérêt de définir $p<2$ [HNG94]. En effet, plus p est petit, plus le front de Pareto sera dense sur les régions diagonales du front et clairsemé sur les régions parallèles aux axes, ce qui avantage la découverte de bons compromis (la forme des niches est schématisée dans la figure 4.18). Donc, c'est la distance de *Manhattan* qui a été finalement préférée à la distance *Euclidienne*.

Un second avantage de cette distance est qu'elle est peu coûteuse en temps de calcul (pas de calcul de racine ni de puissance). Comme le nombre de calculs de distances à chaque génération est élevé (tous les couples parmi les n solutions, soit un calcul en $\Theta(n^2)$), il est intéressant d'utiliser un calcul le moins coûteux possible.

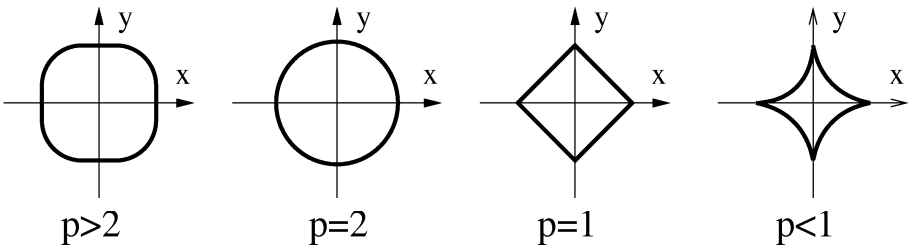


FIG. 4.18 – Formes des niches dans un espace à 2 objectifs, en fonction de p.

La distance génotypique, quant à elle, est calculée à partir du code génétique de deux individus. Il parait naturel d'avoir un calcul de distance en rapport avec l'opérateur de mutation utilisé afin de pénaliser les solutions proches en nombre de mutations à réaliser pour transformer l'une en l'autre. Mais ce calcul de distance n'est pas trivial pour l'opérateur de base de AGA (le *Shift*) |RY98|. Nous proposons néanmoins un calcul de distance basé sur cet opérateur dans le chapitre suivant, mais ce calcul est trop complexe pour l'insérer dans le mécanisme de diversification de AGA.

Nous avons donc défini un opérateur simple permettant d'estimer la ressemblance de deux individus. Elle correspond au nombre de ruptures d'ordre entre les deux codes génétiques :

$$d_2(x,y) = \|(i,j) \text{ tq } x = [\ldots,i,\ldots,j,\ldots] \text{ et } y = [\ldots,j,\ldots,i,\ldots]\| \qquad (4.31)$$

4.4.6 Remplacement

Le mécanisme de remplacement est générationnel, c'est à dire que la population est entièrement remaniée à chaque génération de l'AG . D'autre algorithmes, pour lesquels à une génération correspond un seul individu modifié dans la population, sont dit de type *steady state*.

4.4.7 Description générale de l'algorithme AGA

L'algorithme 4.2 détaille les différentes étapes de ce qui a été décrit dans cette section. Le mécanisme le plus coûteux de l'algorithme est le mécanisme de diversification par *sharing combiné*, pour lequel on doit calculer la distance séparant chaque couple de solutions de la population P. La distance, la plus coûteuse à calculer est bien sur la distance génotypique. Pour une population de p individus de taille N, la complexité globale de AGA est en $O(Np^2)$, puisque le calcul de distance que nous effectuons est en $O(N)$. Ceci montre qu'il est important d'utiliser un calcul de distance peu complexe.

Algorithme 4.2 *AGA*

Création d'une population initiale P aléatoire.
Initialisation des probabilités de sélection P_{M_i} des i opérateurs de mutations à $1/i$.
tant que le temps d'exécution n'est pas atteint **faire**
  Évaluation des solutions de P.
  Élitisme : Calculer PO^*, l'ensemble des solutions non-dominées de P.
  Calculer la valeur de σ_{share} en fonction de PO^*.
  Construction d'une population P' par sélection $NSGA$ élitiste avec diversification par Sharing combiné.
  Croisement.
  Mutation adaptative : Sélection de l'opérateur à appliquer à chaque individu à muter en fonction des valeurs P_{M_i}. Évaluation du fitness de chaque individu N muté pour calculer $\pi(M_{iN})$.
  Calculer la valeur de $Progress(M_i)$ pour les i opérateurs de mutation.
  Calculer les valeurs P_{M_i} pour la génération suivante de AGA.
  Remplacement : $P \leftarrow P'$.
fin tant que
Retourner PO^*.

4.5 Résultats expérimentaux

Intéressons nous maintenant aux résultats obtenus lors de nos expérimentations. Dans un premier temps, nous évaluerons les apports des mécanismes adaptatifs introduits dans ce chapitre en comparant l'AG de base avec AGA. L'AG de base est défini comme AGA, hormis les deux mécanismes présentés dans la section 4.3. Puis nous comparerons AGA avec un algorithme classique de l'optimisation multi-objectif : $NSGA\ II$. Cela nous permettra de conclure sur l'intérêt de AGA pour les hybridations présentées dans les chapitres suivants.

4.5.1 Protocole d'expérimentation

4.5.1.1 Tests réalisés

Comme indiqué lors du chapitre précédent, nous avons réalisé nos tests sur les jeux de données de Taillard [Tai93a] étendus au cas bi-objectif. Les expérimentations ont été réalisées sur un `Pentium 4` à $1.7GHz$. 10 exécutions de 100 minutes pour chaque instance ont été réalisées. Pour chaque algorithme testé, les populations initiales sont les mêmes (cette propriété est également valable pour les expérimentations des prochains chapitres).

4.5.1.2 Paramètres

4.5.1.2.1 Taille de la population

La taille de la population est un paramètre influant beaucoup l'efficacité globale de l'AG. Horn montre que, pour une convergence rapide de l'algorithme, tout en gardant une bonne

répartition sur le front, il faut que le nombre d'individus non-dominés de la population représente environ 10% de la population totale [HNG94]. Donc, dans l'algorithme implanté, le nombre d'individus de la population courante devrait être 10 fois plus important que le nombre d'individus de l'archive Pareto. Cependant, un nombre trop grand d'individus rend l'algorithme bien trop lent, surtout pour les grosses instances de problèmes, pour lesquelles la convergence est plus longue et pour lesquelles les fronts de Pareto sont souvent bien plus grands. Nous avons donc décidé d'avoir une taille de population constante de 150 individus pour nos expérimentations, pour chaque approche testée.

4.5.1.2.2 Pourcentages de mutation et de croisement

Les valeurs que nous utilisons pour ces paramètres sont $p_{mutation} = 0.35$ et $p_{crossover} = 0.25$. Nous utilisons ces valeurs, de manière constante, pour toutes nos expérimentations futures. Cependant, il existe différentes méthodes proposées pour définir dynamiquement la valeur de ces paramètres [SD94].

4.5.1.2.3 Pression de sélection

Là encore, cette valeur peut être définie de manière dynamique. Horn propose de faire varier ce paramètre en fonction du taux d'individus non-dominés dans la population courante [HNG94]. Pour notre cas, elle a été fixée expérimentalement à 1.5.

4.5.1.2.4 σ_{share} dans l'espace génotypique

Dans notre cas, nous avons laissé ce paramètre constant. Les distances génotypiques entre différentes solutions d'une population sont moins sujettes à de grandes variations que les distances phénotypiques. En effet, l'ordre de grandeur des distances génotypiques que l'on rencontrera durant l'algorithme ne dépend que du nombre de jobs. De plus, les diverses expérimentations montrent que le nombre de solutions Pareto optimales augmente à peu près proportionnellement avec le nombre de jobs et de machines. Donc prendre un σ_{share} constant sur tous les problèmes n'est pas ou peu pénalisant. Pour le problème présent, la taille des niches au niveau génotypique a été fixée à 4. Cette valeur permet de pénaliser les solutions identiques, ainsi que les solutions 'voisines', c'est-à-dire distantes l'une de l'autre d'une seule application d'un opérateur de mutation. Cette valeur reste également constante pour toutes nos expérimentations.

4.5.1.2.5 Valeur de σ_{share} dans l'espace phénotypique

Pour implémenter le mécanisme de diversification phénotypique adaptative, il reste à définir la notion de distance utilisée pour nos tests.

La distance utilisée ici n'est pas la distance *Euclidienne* mais la distance de *Manhattan*. Donc le calcul de la borne supérieure de σ_{share} s'en trouve changé. En effet, avec le calcul des distances utilisé ici, quelle que soit l'allure du front, son aire est définie par ses deux extrémités et vaut $|M_1 - m_1| + |M_2 - m_2|$ (comme le montre la figure 4.19, sur les deux allures extrêmes du front, l'aire optimale des niches ne change pas). Donc, la valeur de σ_{share} dans le cas présent est :

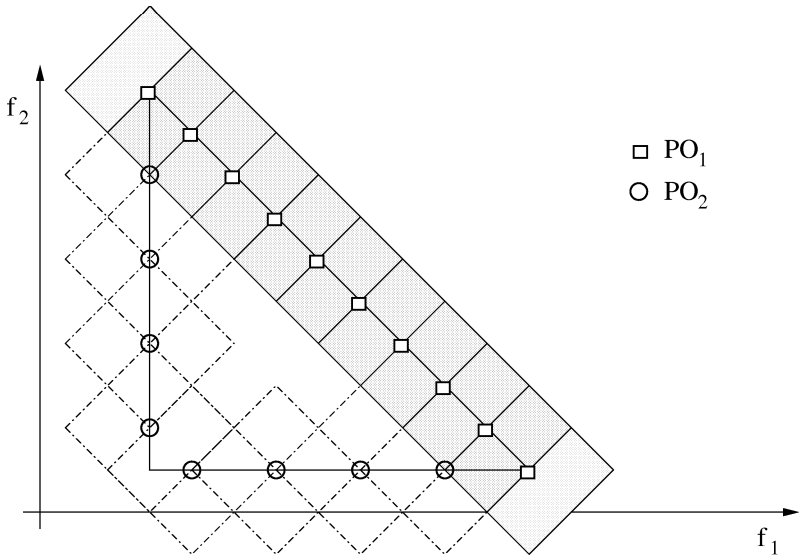


FIG. 4.19 – Distance de *Manhattan* : Deux ensembles Pareto PO_1 et PO_2, aux formes extrêmes sont représentés. La taille idéale des niches reste constante quelle que soit l'allure des ensembles Pareto, ce qui n'est pas le cas pour d'autres calculs de distances, comme la distance *Euclidienne* (voir figure 4.10). Ici, σ_{share} se calcule donc uniquement en fonction des valeurs extrêmes de l'ensemble de Pareto.

$$\sigma_{share} = \frac{|M_1 - m_1| + |M_2 - m_2|}{N} \tag{4.32}$$

Enfin, dans un but d'équité entre les différents objectifs pour la diversification, il est utile de normaliser les coûts des différentes solutions, M_1 et M_2 valent alors 1, m_1 et m_2 valent 0. Donc la formule définitive utilisée pour la taille des niches dans l'espace phénotypique est :

$$\sigma_{share} = 2/N \tag{4.33}$$

4.5.1.2.6 Valeur de k pour la mutation adaptative

Cette valeur a été fixée expérimentalement à 2.

4.5.2 Apports des mécanismes introduits

Nous avons évalué nos fronts à l'aide des métriques S et $Contribution$ appliquées aux ensembles Pareto obtenus par les algorithmes avec les mêmes populations initiales. En premier lieu, nous avons évalué les apports des mécanismes présentés dans ce chapitre par rapport à l'AG de base.

Les tableaux 4.3, 4.4 et 4.5 présentent les valeurs minimales (Min), maximales (Max),

TAB. 4.3 – Évaluation des performances : comparaison entre AGA et AG_{base} avec la métrique *Contribution*.

Instance	Contribution(AGA/AG$_{base}$)			
	Min	Max	Moy	ET
ta_20_5_01	0.25	1.00	**0.613**	0.271
ta_20_5_02	0.42	1.00	**0.662**	0.201
ta_20_10_01	0.14	0.88	**0.571**	0.221
ta_20_10_02	0.32	0.89	**0.524**	0.150
ta_20_20_01	0.22	0.82	**0.553**	0.154
ta_50_5_01	0.22	1.00	**0.711**	0.308
ta_50_10_01	0.00	1.00	**0.607**	0.320
ta_50_20_01	0.00	0.31	**0.126**	0.114

moyennes (Moy) et l'écart type (ET) obtenus pour ces métriques pour l'ensemble des exécutions réalisées pour chaque instance testée. Puis nous avons observé les résultats obtenus par rapport aux bornes sur chaque objectif (tableaux 4.6 et 4.7).

4.5.2.1 Comparaison avec le cas multi-objectif

Le tableau 4.3 montre que, dans la plupart des instances, les fronts obtenus par AGA sont meilleurs que ceux obtenus par l'AG de base pour la métrique *Contribution*. En effet, en comparant les fronts obtenus sur toutes les instances, exceptées la dernière, une moyenne de 59.2% des solutions Pareto sont fournies par AGA.

Seule la plus grosse instance testée ($ta\_50\_20\_01$) donne des résultats décevants (12.6% de moyenne). En fait, il semble que l'exploration plus 'intelligente' de l'espace de recherche permet d'améliorer les résultats généraux, mais, sur cette plus grosse instance, la convergence étant plus longue à obtenir, la diversification engendre un retard supplémentaire pour la convergence de AGA. Notons également que les résultats varient beaucoup selon les exécutions, ce qui montre une certaine faiblesse de convergence et conforte l'idée d'une coopération avec d'autres méthodes d'optimisation.

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 pour la *Contribution*. Néanmoins, la mesure permet ici, de quantifier plus précisément les différences entre les ensembles Pareto obtenus par les deux algorithmes. La colonne *Apport* de la table 4.5 est le rapport des aires de dominance obtenues par AGA par rapport à l'AG de base, c'est-à-dire $S(AGA)/S(AG_{base})$. De plus, en observant les écarts types des différentes exécutions réalisées des deux méthodes, on remarque que, malgré un manque certain de robustesse des deux méthodes, AGA est plus stable pour toutes les instances.

La métrique S tient compte en partie de la diversité du front, ce que ne fait pas la métrique *Contribution*. Or, on observe dans les résultats une différence plus importante du point de vue de la métrique S par rapport à la métrique *Contribution*, ce qui indique un ap-

TAB. 4.4 – Évaluation des performances de AG_{base} avec la métrique S.

Instance	$S(AG_{base})$			
	Min	**Max**	**Moy**	**ET**
ta_20_5_01	3612	4393	**4166.9**	317.8
ta_20_5_02	8935	10657	**9995.7**	718.8
ta_20_10_01	231224	252835	**240980.2**	8024.3
ta_20_10_02	155695	172801	**164826.7**	5534.6
ta_20_20_01	377979	502144	**430951.4**	40333.0
ta_50_5_01	131781	177186	**158228.9**	14691.5
ta_50_10_01	409150	774568	**593797.7**	115172.3
ta_50_20_01	2553558	3131650	**2732590.2**	191457.9

TAB. 4.5 – Évaluation des performances : comparaison de AGA par rapport à AG_{base} avec la métrique S.

Instance	S(AGA)				
	Min	**Max**	**Moy**	**ET**	**Apport**
ta_20_5_01	3837	4449	4247.8	268.0	**1.94%**
ta_20_5_02	10594	10948	10770.1	164.7	**7.75%**
ta_20_10_01	234171	252950	245016.9	7136.8	**1.68%**
ta_20_10_02	158874	173121	166520.8	3806.5	**1.03%**
ta_20_20_01	411872	501477	461176.9	38142.1	**7.01%**
ta_50_5_01	155403	186970	174577.0	9783.6	**10.33%**
ta_50_10_01	467242	806158	639182.3	102171.7	**7.64%**
ta_50_20_01	2152415	2714895	2418535.2	171430.6	**-11.49%**

port au niveau de la diversité. C'est le cas pour les instances, *ta_20_5_02*, *ta_20_20_01*, *ta_50_5_02* et *ta_50_10_02*, où la *Contribution* indique un petit apport, mais la métrique S indique un progrès plus conséquent. Pour l'intance *ta_50_20_01* la *Contribution* indique une nette dominance de AG_{base}, dominance moins prononcée pour la métrique S. Donc, pour cette instance, AGA donne des résultats peu intéressants en terme de convergence, mais procure une population mieux diversifiée. Les figures 4.20 et 4.21 sont des exemples de fronts obtenus par les deux algorithmes, montrant l'apport de AGA en terme de diversité. La figure 4.22 montre également que, lorsque le temps d'exécution augmente, AGA continue de progresser, contrairement à l'AG de base.

Les différentes expérimentations menées sur AGA ont montré que l'opérateur de mutation *Shift* est le plus performant pour toutes les instances traitées. Pour les petits problèmes à 20 jobs, il est légèrement plus performant que les opérateurs *Exchange* et *Random*. Lorsque la taille des problèmes augmente, cette différence tend à s'accentuer. La figure 4.23 montre l'évolution des probabilités de sélection de chaque opérateur durant une exécution de AGA sur l'instance *ta_100_5_01*.

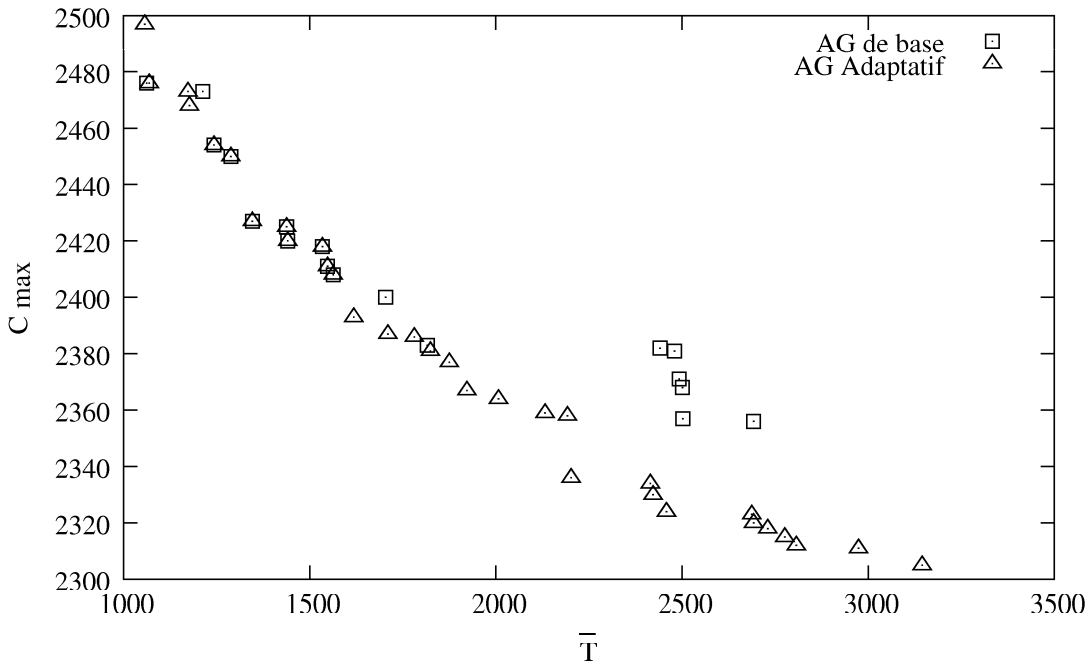


FIG. 4.20 – Résultat sur l'instance *ta*_20_20_01. Il y a un apport intéressant de *AGA* en terme de convergence et de diversité.

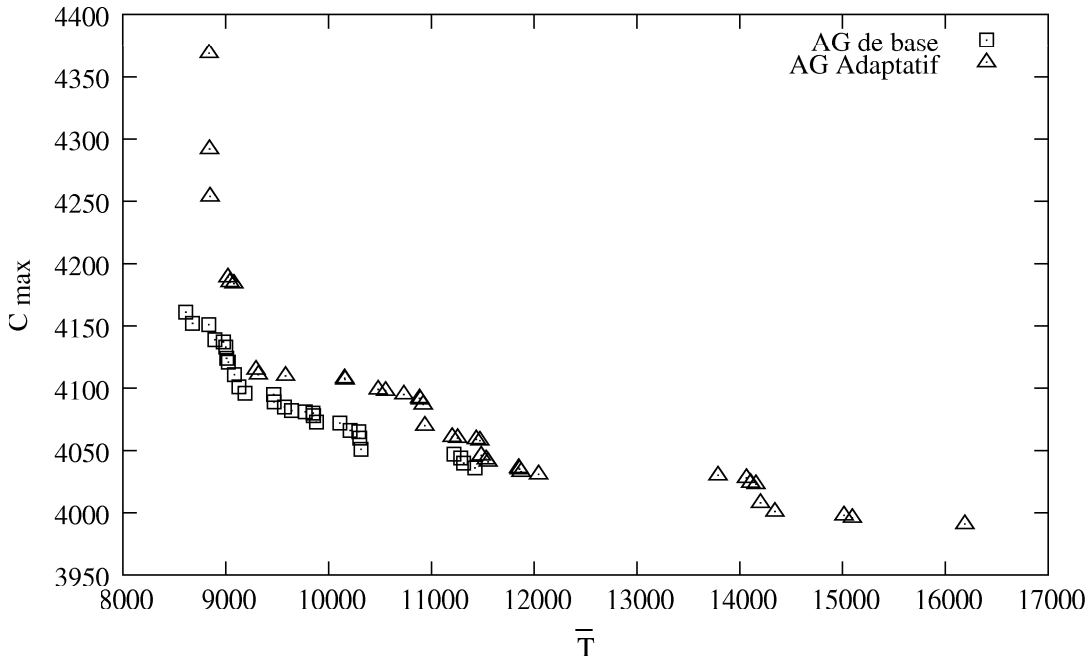
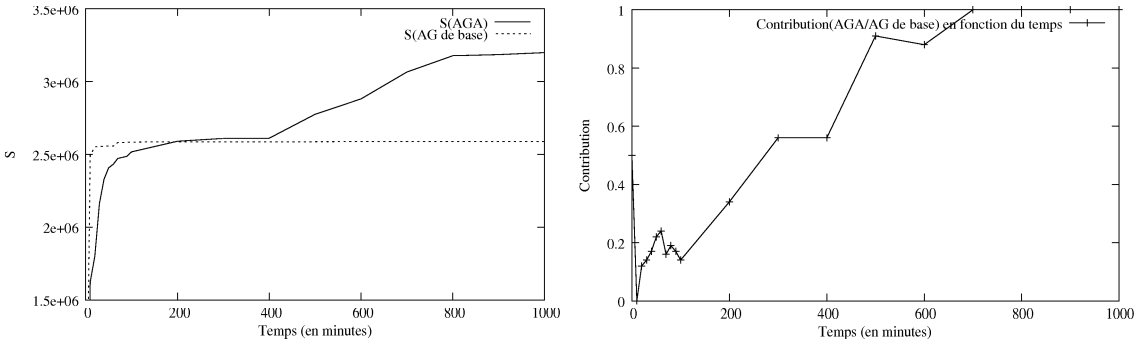


FIG. 4.21 – Résultat sur l'instance *ta*_50_20_01. Bien que le front obtenu par *AGA* soit faible en terme de convergence, la population obtenue est très diversifiée et offre de bonnes perspectives de convergence pour une hybridation.

FIG. 4.22 – Comparaison de AGA avec l'AG de base en fonction du temps : AGA est de plus en plus efficace lorsque le temps augmente, du point de vue des métriques S et $Contribution$. Sur cet exemple, AGA et AG_{base} ont été exécutés pendant 1000 minutes sur l'instance $ta\_50\_20\_01$.



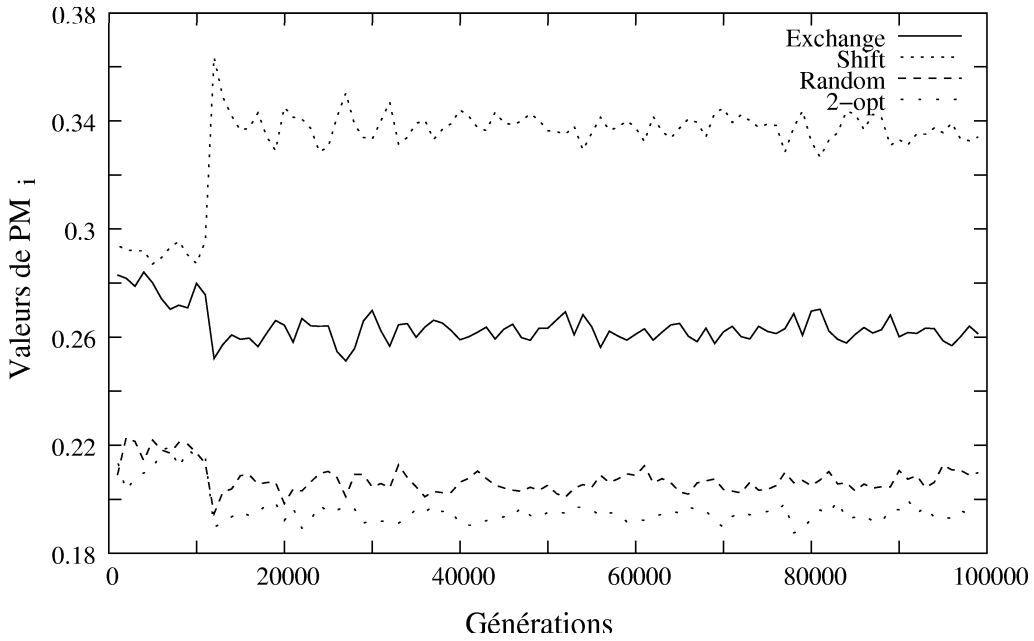


FIG. 4.23 – Évolution des PM_i durant l'exécution de AGA.

TAB. 4.6 – Meilleurs C_{max} obtenus par AGA. Comparaison par rapport aux meilleures bornes de la littérature.

Instance	LB	Min	Max	Moy	ET	Min_{Dev}	Moy_{Dev}
$ta\_20\_5\_01$	1278	1278	1278	1278.0	0.0	0%	0%
$ta\_20\_5\_02$	1359	1359	1365	1361.4	2.9	0%	0.18%
$ta\_20\_10\_01$	1582	1586	1617	1596.6	10.9	0.25%	0.92%
$ta\_20\_10\_02$	1659	1672	1682	1676.2	3.4	0.78%	1.04%
$ta\_20\_20\_01$	2299	2307	2382	2344.0	24.4	0.44%	2.05%
$ta\_50\_5\_01$	2724	2724	2735	2729.3	4.3	0%	0.19%
$ta\_50\_10\_01$	2991	3063	3126	3088.4	24.7	2.41%	3.26%
$ta\_50\_20\_01$	3850	3957	4056	3990.1	30.3	2.78%	3.64%

TAB. 4.7 – Meilleures valeurs obtenues pour le retard total par AGA.

Instance	Min	Max	Moy	ET
$ta\_20\_5\_01$	452	470	457.3	7.5
$ta\_20\_5\_02$	469	491	480.0	11.0
$ta\_20\_10\_01$	1224	1280	1241.0	25.5
$ta\_20\_10\_02$	1275	1365	1311.4	31.7
$ta\_20\_20\_01$	1031	1062	1054.8	12.1
$ta\_50\_5\_01$	3355	3723	3521.3	118.0
$ta\_50\_10\_01$	5751	6621	6140.8	304.9
$ta\_50\_20\_01$	7806	9409	8758.6	469.7

4.5.2.2 Comparaison par rapport aux bornes

Afin d'évaluer l'efficacité de notre algorithme par rapport aux études mono-objectif, les tables 4.6 et 4.7 représentent les valeurs minimales, maximales, moyennes et l'écart type des meilleures valeurs obtenues pour chaque objectif, pour les différents tests réalisés. Pour le C_{max}, nous avons comparé les résultats obtenus avec les meilleurs valeurs de la littérature. Min_{Dev} (respectivement Moy_{Dev}) représente l'écart mimimal (respectivement moyen) entre l'optimal de la littérature en mono-objectif LB et la borne trouvée par AGA. Pour les petits problèmes, les résultats fournis par AGA sont proches des optimums mono-objectif du C_{max}. Cependant, pour les plus grands problèmes testés, l'algorithme s'éloigne des optimums mono-objectif. Pour la somme des retards, nous n'avons pas de référence mono-objectif à proposer à titre de comparaison, donc la table 4.7 est donnée à titre indicatif.

4.5.3 Comparaison avec NSGA II

Afin de mieux évaluer *AGA*, nous avons implémenté un des algorithmes classiques de l'optimisation multi-objectif, proposé par Deb *et al.* : *NSGA II* (*Non-dominated Sorting Genetic Algorithm II*). Cette méthode est un algorithme génétique avec approche Pareto évolué, qui utilise l'élitisme et une méthode de diversification dite par *crowding* |DAPM00|. Nous avons réalisé une série de tests pour comparer *NSGA II* avec *AGA*. Pour nos expérimentations, *NSGA II* et *AGA* utilisent la même taille de population ainsi que les mêmes populations initiales.

TAB. 4.8 – Comparaison *NSGA II*/*AGA* : Métrique *Contribution* (*C*(*AGA*/*NSGA II*)).

	Cont(AGA/NSGA II)			
Instance	**Min**	**Max**	**Moy**	**ET**
*ta*_20_5_01	0.33	1.00	0.900	0.214
*ta*_20_5_02	0.83	1.00	0.946	0.070
*ta*_20_10_01	0.30	0.67	0.501	0.110
*ta*_20_10_02	0.36	0.69	0.544	0.099
*ta*_20_20_01	0.13	0.93	0.392	0.279
*ta*_50_5_01	0.00	1.00	0.441	0.344
*ta*_50_10_01	0.00	1.00	0.551	0.389
*ta*_50_20_01	0.00	0.20	0.027	0.059

TAB. 4.9 – Comparaison *NSGA II*/*AGA* : Métrique *S*. Apport représente le rapport *S*(*AGA*)/*S*(*NSGA II*.

	S(NSGA – II)				
Instance	**Min**	**Max**	**Moy**	**ET**	**Apport**
*ta*_20_5_01	378	4368	3010.6	1278.3	41.09%
*ta*_20_5_02	7438	9608	8492.3	566.9	26.82%
*ta*_20_10_01	236497	255661	243414.6	6727.3	0.66%
*ta*_20_10_02	155909	171811	161958.7	3899.0	2.82%
*ta*_20_20_01	271141	504444	455221.3	75095.7	1.31%
*ta*_50_5_01	147901	184711	171980.9	11385.3	1.51%
*ta*_50_10_01	416038	742794	573043.9	103437.4	11.54%
*ta*_50_20_01	2780145	3372471	3078976.8	209967.4	-21.45%

Les tables 4.8 et 4.9 représentent les résultats obtenus par *NSGA II* pour les métriques *S* et *Contribution*. *NSGA II* est un algorithme très élitiste, si bien que la convergence de l'algorithme est très rapide pour un *AG*. La convergence de *AGA* est plus lente, mais l'exploration de l'espace y est mieux réalisée. Donc, à long terme, *AGA* procure de meilleurs résultats.

Pour la métrique *Contribution*, *AGA* est meilleur sur les deux plus petites instances,

et moins bon sur la plus grande, les deux algorithmes étant à peu prés équivalents pour les autres instances.

Pour la métrique S, $NSGA\ II$ donne de meilleurs résultats uniquement sur l'instance à 50 jobs et 20 machines, et est battu par AGA sur toutes les autres instances. Là encore, AGA est plus performant pour la métrique S que pour la contribution, ce qui indique une meilleure diversité des solutions obtenues.

4.5.4 Comparaison par rapport aux ensembles Pareto optimaux

Lorsque cela est possible, il est plus intéressant d'évaluer un algorithme par rapport aux ensembles Pareto optimaux. L'évaluation du temps de calcul nécessaire pour obtenir la totalité des solutions Pareto optimales est un critère de comparaison efficace, car très facile à interpréter. Néanmoins, on ne peut généralement connaître ces fronts exacts que pour les problèmes de petites tailles. Dans le chapitre 7, nous détaillons une méthode bi-objectif exacte, appelée méthode deux phases, qui nous a permis d'obtenir les ensembles Pareto optimaux pour les cinq plus petites instances (nous verrons également par la suite que l'ensemble Pareto optimal semble avoir été atteint pour l'instance $ta\_50\_5\_01$). Le tableau 4.10 montre le temps nécessaire à AGA pour la découverte de ces ensembles Pareto exacts. Nous remarquons que, bien qu'AGA offre une bonne approximation de l'ensemble Pareto exact des instances traitées, il a énormément de mal à converger vers l'ensemble Pareto optimal. Lorsque le front exact est atteint, le temps nécessaire pour y parvenir est généralement très grand. Notons que, lors des tests effectués sur l'algorithme $NSGA\ II$, aucun ensemble Pareto optimal n'a été atteint.

TAB. 4.10 – Temps de résolution exact de AGA (minimum, maximum, moyen et écart type). Les exécutions infructueuses ont été stoppées après une semaine de calcul.

Instance	$\mathbf{T_{Min}}$	$\mathbf{T_{Max}}$	**Moyenne**	**ET**
$ta\_20\_5\_01$	2'43"	40h57'51"	7h49'51"	9h54'17"
$ta\_20\_5\_02$	8h57'56"	*Non atteint*	X	X
$ta\_20\_10\_01$	*Non atteint*	*Non atteint*	X	X
$ta\_20\_10\_02$	*Non atteint*	*Non atteint*	X	X
$ta\_20\_20\_01$	*Non atteint*	*Non atteint*	X	X
$ta\_50\_5\_01$	*Non atteint*	*Non atteint*	X	X

4.6 Conclusion

Dans ce chapitre, nous avons proposé un algorithme génétique avec approche Pareto pour la résolution du Flow-shop bi-objectif. Nous nous sommes appuyés sur un algorithme de base, utilisant des opérateurs classiques efficaces pour ce problème. Ensuite, nous nous sommes concentrés sur l'élaboration de mécanismes permettant d'améliorer l'exploration

de l'espace de recherche, en proposant des méthodes adaptatives afin de rendre l'algorithme efficace sur les différentes instances traitées.

En premier lieu, nous avons établi une méthode permettant l'utilisation de plusieurs opérateurs de mutation de manière simultanée durant le déroulement de l'algorithme génétique. Cette méthode permet d'une part de diminuer le nombre d'optima locaux qui bloquent très souvent la convergence des algorithmes génétiques et d'avoir une population plus facilement diversifiée. D'autre part, elle permet d'éviter une phase de tests pour déterminer l'opérateur de mutation le plus efficace parmi une liste de candidats potentiels. Cette phase de tests, nécessaire lorsqu'on utilise un seul opérateur de mutation, est souvent fastidieuse et ne garantit pas l'utilisation du meilleur opérateur pour l'ensemble des instances possibles du problème donné. Bien évidemment, le mécanisme proposé est applicable à d'autres opérateurs génétiques, et en particulier aux opérateurs de croisement.

Ensuite, nous avons proposé une méthode de diversification adaptative, toujours dans le but de favoriser l'exploration et de rendre l'algorithme génétique adaptatif. Nous avons proposé un moyen de calculer de manière dynamique un paramètre crucial déterminant l'efficacité de la diversification par *sharing* : la taille des niches. Ce calcul dynamique de la taille des niches permet également d'éviter une phase de tests, ou l'utilisation d'une valeur peu adéquate pour l'instance traitée.

Enfin, nous avons réalisé des expérimentation sur le problème du Flow-shop bi-objectif, afin d'évaluer l'intérêt de tels mécanismes. La comparaison de l'algorithme génétique adaptatif par rapport à sa version de base a montré des résultats encourageants, les ensembles Pareto obtenus étant généralement meilleurs et plus diversifiés. Nous avons comparé dans un deuxième temps cet algorithme avec $NSGA\ II$, un algorithme génétique classique en optimisation multi-objectif. Les résultats obtenus sont encore une fois probants. Cependant, la convergence de notre algorithme reste lente, l'utilisation de cette méthode de manière coopérative avec des méthodes à convergence rapide semble inéluctable pour obtenir les meilleurs résultats possibles. Les méthodes coopératives font l'objet des chapitres suivants, où les approches proposées tentent d'allier la capacité d'exploration de AGA avec la capacité d'intensification d'autres méthodes d'optimisation.

Chapitre 5

Algorithme génétique coopératif

Dans ce chapitre, nous proposons différentes méthodes coopératives s'appuyant sur deux briques de base : l'algorithme génétique adaptatif présenté dans le chapitre précédent et une recherche locale Pareto (PLS), présentée dans ce chapitre. Quatre coopérations ont été réalisées : AMA, AMA + AGA, AGMA et PAGMA. Nous montrons l'efficacité de chacune de ces coopérations, chaque apport réalisé améliorant les résultats obtenus. Ce chapitre a fait l'objet de publications dans les conférences CEC'02 [BST02] et CESA'03 [BST03], et d'un article soumis à la revue IEEE Transactions on Evolutionary Computation [BST05a].

5.1 Introduction

Dans le chapitre précédent, nous avons présenté des mécanismes afin de mettre en oeuvre un algorithme génétique adaptatif. Les mécanismes décrits ont permis d'obtenir un algorithme assez performant, mais qui cherche avant tout à diversifier la recherche, par l'intermédiaire du mécanisme de diversification adaptatif ainsi que par l'application de plusieurs opérateurs de mutation différents.

Dans le chapitre 2, nous avons remarqué que beaucoup d'approches coopératives trouvées dans la littérature mettent en oeuvre une méthode de recherche à solution unique et une méthode de recherche à base de population, afin d'allier la capacité d'intensification avec la capacité d'exploration de ces algorithmes.

Dans le cadre de l'optimisation multi-objectif avec approche Pareto, l'utilisation constante d'une population de solutions paraît primordiale, étant donné que la recherche s'oriente vers la découverte d'un ensemble de solutions non-dominées. Dans ce chapitre, nous proposons donc des approches coopératives souvent de type **HTH**, où les différentes méthodes utilisées gardent cette notion d'approche Pareto à base de population.

Afin d'intensifier la recherche effectuée par AGA, nous proposons en premier lieu une recherche locale Pareto (PLS), basée sur une population de solutions. Les coopérations mises en œuvre utilisent toutes l'algorithme AGA comme méthode d'exploration de l'espace de recherche, et PLS comme mécanisme d'intensification.

Le graphe d'agrégation des différentes méthodes réalisées à partir de AGA et PLS est schématisé sur la figure 5.1. Tout d'abord nous présentons AMA, un algorithme coopératif de type *mimétique*. Puis, nous proposons deux méthodes de coopération entre AMA et AGA. La première ($AGA+AMA$) s'exécute de manière séquentielle, et les transitions sont prédéfinies. La deuxième ($AGMA$) effectue des transitions adaptatives durant l'exécution. Enfin, nous proposons des modèles parallèles $PAGMA$, basés sur le modèle coopératif de $AGMA$.

Dans la section 5.2 nous présenterons, dans un premier temps, la recherche locale Pareto avant de proposer différents types de coopérations séquentielles AMA, $AGA + AMA$ et $AGMA$. Nous évaluerons l'efficacité des différentes approches proposées. Dans la section 5.3, nous aborderons la coopération parallèle de métaheuristiques de type "modèle en îles", en proposant et évaluant différents schémas de coopérations parallèles. Enfin, nous conclurons ce chapitre dans la section 5.4.

5.2 Algorithme génétique coopératif séquentiel

Nous proposons dans cette section différentes méthodes de coopération menant en dernier lieu à une coopération adaptative entre AGA et une recherche de type algorithme mimétique. Pour cela, nous commencerons par présenter une méthode de recherche locale Pareto, basée sur une population de solutions, puis, à partir de cette recherche locale, nous présenterons un algorithme mimétique. Enfin nous ferons coopérer AGA avec cet algorithme mimétique en proposant une coopération classique et une autre se réalisant

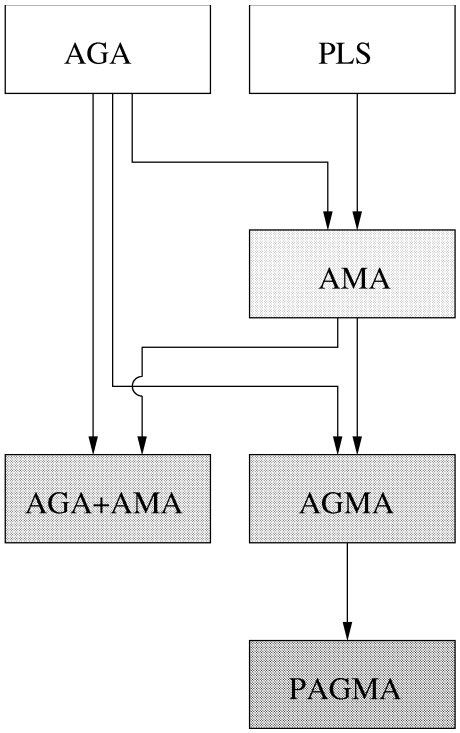


FIG. 5.1 – Graphe d'agrégation des méthodes d'optimisation présentées dans ce chapitre. Plus une méthode est grisée, plus cette méthode résulte de nombreuses coopérations.

de manière adaptative durant l'exécution. Nous terminons cette section en présentant les résultats obtenus sur le Flow-shop bi-objectif pour chaque version.

5.2.1 Recherche locale Pareto

Les méthodes de recherche locale sont généralement des algorithmes à solution unique. Mais, dans le cadre de l'optimisation multi-objectif, l'utilisation d'une population de solutions permet de pallier l'absence de relation d'ordre total entre les solutions en maintenant un ensemble de solutions non-dominées durant la recherche locale. Nous proposons donc un algorithme de Recherche Locale Pareto (*Pareto Local Search* - PLS) appliquant les principes de la recherche locale à un ensemble de solutions non-dominées.

5.2.1.1 Description

La méthode PLS est une recherche locale basée sur une population de solutions décrite dans l'algorithme 5.1.

La figure 5.2 montre le déroulement d'une itération de PLS. L'ensemble Pareto progresse parallèlement à chaque pas, jusqu'à ce que la génération du voisinage n'apporte aucune nouvelle solution. Afin d'éviter les calculs redondants, on ne génère pas le voisinage

Algorithme 5.1 *PLS*.

Génération d'un ensemble initial de solutions non-dominées PO.
répéter
  $S' \leftarrow PO$.
  Génération du voisinage PN_x pour chaque solution x de S'.
  Soit PO l'ensemble des solutions non-dominées de $S' \cup_x PN_x$.
jusqu'à PO=S' (la population a atteint un optimum local).
Retourner l'ensemble des solutions non-dominées PO.

des solutions qui étaient déjà présentes à l'itération précédente (soit les solutions noires non barrées sur la figure 5.2).

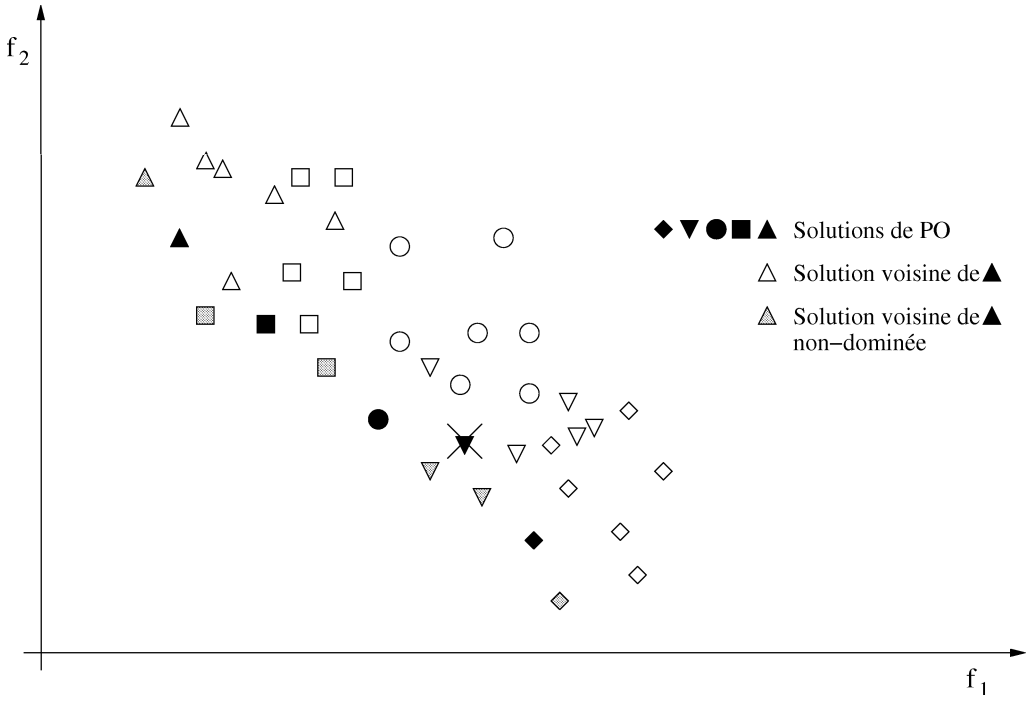


FIG. 5.2 – Recherche locale Pareto : On génère le voisinage V (solutions blanches et grises) de chaque solution de l'ensemble Pareto courant PO (solutions noires), puis on affecte à PO les solutions non-dominées de $V \cup PO$ pour l'itération suivante de PLS, si de nouvelles solutions ont été découvertes (solutions grises ou noires, non barrées).

L'approche de PLS est assez naturelle. Une autre méthode envisageable consiste à d'agréger les objectifs pour réaliser une recherche locale classique. Les résultats obtenus sont bien meilleurs avec PLS. Cependant, pour de grandes instances, ou lorsque les ensembles de solutions Pareto manipulées sont trop grands, PLS devient très coûteux en temps de calcul. Dans ces cas, il faut peut être envisager des recherches locales partielles (modification du critère d'arrêt, exploration d'un voisinage partiel...), ou une sélection de certaines solutions non-dominées pour la recherche locale à l'aide d'une méthode de *clustering*.

5.2.1.2 Implémentation

Le voisinage que nous utiliserons lors de nos expérimentations sera celui engendré par l'opérateur *Shift*, le plus performant d'après les résultats de AGA.

Pour le problème de Flow-shop bi-objectif, les expérimentations montrent que les ensembles Pareto obtenus sont de taille raisonnable (jamais plus de 200 individus), nous permettant de ne pas modifier l'algorithme de base et de réaliser la recherche locale Pareto complète, même pour les grandes instances.

5.2.2 Algorithme mimétique adaptatif

Les algorithmes mimétiques présentés dans la section 2.2.3.2 sont des métaheuristiques hybrides de type **LTH**. Ils sont connus pour converger plus rapidement que les algorithmes génétiques standards. Nous proposons ici un algorithme mimétique standard basé sur AGA et PLS.

5.2.2.1 Description

Les algorithmes mimétiques tentent d'allier les avantages en terme d'exploration des métaheuristiques à base de population avec ceux en terme d'intensification des algorithmes de descente. La méthode de type mimétique la plus classique consiste à remplacer l'opérateur de mutation habituel par un algorithme de recherche locale. L'idée est donc, à partir de l'algorithme AGA, de mettre au point un algorithme mimétique en remplaçant les opérateurs de mutation utilisés initialement par l'algorithme PLS présenté précédemment. PLS est basé sur une population de solutions, il a donc été nécessaire de s'éloigner un peu du schéma classique des algorithmes mimétiques pour mettre en oeuvre notre algorithme. Nous avons gardé l'ensemble de l'exécution de AGA, en ajoutant à la fin de chaque génération une sélection de quelques individus qui participeront à la recherche locale PLS (voir algorithme 5.2). Les individus sélectionnés sont en fait issus de croisements d'individus de la population courante de AGA. La coopération proposée est de type **LTH(AGA(PLS))**.

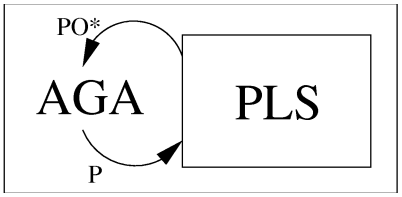


FIG. 5.3 – Algorithme mimétique adaptatif : le principe consiste à itérer des recherches de PLS sur des individus issus de croisements sur la population courante P de AGA, favorisant l'exploration de nouvelles zones de recherche. On met à jour l'archive Pareto PO^* de AGA en fonction des solutions obtenues par PLS.

Dans cet algorithme, PLS utilise les solutions de la population courante P fournies par AGA. Ces solutions ont profité des différents mécanismes de AGA, ce qui leur permet

Algorithme 5.2 *AMA*.

Création d'une population initiale aléatoire.
tant que le temps d'exécution n'est pas atteint **faire**
  Réaliser une génération de *AGA* (avec une population P et une archive Pareto PO^*).
  Appliquer l'opérateur de croisement à des couples de solutions sélectionnées aléatoirement dans la population courante de *AGA*.
  Calculer l'ensemble des solutions non dominées PO' de ces solutions créées.
  /* *PLS* */
  répéter
    $S' \leftarrow PO$.
    Génération du voisinage PN_x pour chaque solution x de S'.
    Soit PO l'ensemble des solutions non-dominées de $S' \cup_x PN_x$.
  jusqu'à PO=S' (la population a atteint un optimum local).
  Mettre à jour PO^* (PO^* = Solutions non-dominées de $PO \cup PO^*$).
fin tant que
Retourner l'ensemble Pareto PO^*.

d'acquérir une certaine diversité. Quant à *AGA*, il utilise les efforts d'intensification fournis par *PLS*, par l'intermédiaire de l'élitisme de l'*AG*, qui insère des solutions de l'archive Pareto dans le mécanisme de sélection de *AGA*. Cette recherche de diversité allié à un effort d'intensification engendre une bonne interaction entre les deux méthodes *AGA* et *PLS* (voir figure 5.3).

5.2.2.2 Implémentation

Pour l'implémentation de cet algorithme pour le Flow-shop bi-objectif, nous avons repris les algorithmes mis au point pour appliquer *AGA* et *PLS* au flow-shop bi-objectif, sans modifier le paramétrage.

5.2.3 Algorithme génétique/mimétique adaptatif

L'algorithme génétique *AGA* et l'algorithme mimétique *AMA* sont deux méthodes d'optimisation qui ne présentent pas les mêmes qualités. *AGA* est censé offrir une bonne exploration du paysage des solutions. Utilisé de manière coopérative et adaptative avec *AMA*, cela doit permettre d'obtenir de meilleurs résultats que la recherche mimétique seule. En effet, *AMA*, la première version de coopération entre *AGA* et *PLS* que nous venons de détailler, reste un algorithme qui favorise l'intensification. Les recherches locales étant réalisées de manière systématique à toutes les générations, cela laisse peu de possibilités à la partie *AG* de *AMA* d'exprimer ses capacités exploratrices. Nous proposons de faire coopérer *AGA* avec *AMA*, de manière adaptative.

5.2.3.1 Description

Une hybridation classique de ce type de métaheuristique consiste à les exécuter en séquence : exploration (AGA), puis intensification (AMA). Une telle hybridation (que l'on appellera $AGA + AMA$) est de type **HTH(AGA+AMA)**, que l'on peut considérer également de type **HTH(AGA+LTH(AGA(PLS)))**, en développant la totalité de la coopération. AGA fournit donc une population de solutions non-dominées à AMA, qui tente d'améliorer les solutions essentiellement à l'aide du mécanisme de recherche locale. Nous avons voulu améliorer ce schéma classique d'hybridation suite à deux observations :

- La capacité exploratrice de la première métaheuristique n'est exploitée qu'en première phase de l'algorithme. Il semble plus intéressant de pouvoir y revenir après une phase d'intensification. Les deux algorithmes devraient être exécutés de manière alternée afin de tirer le meilleur parti de cette coopération.
- Il n'est pas aisé de définir manuellement le moment adéquat pour achever la première métaheuristique et lancer la deuxième. Est-il plus intéressant d'allouer un temps équitable entre les deux méthodes hybridées ou faut-il en favoriser une ?

Afin de répondre à ces interrogations, nous proposons de définir des règles de transition entre les deux métaheuristiques. Selon que ces règles soient valides ou pas, on exécutera l'une ou l'autre métaheuristique. De plus, le problème de lenteur de convergence étant lié à AGA, il semble nécessaire de construire ces règles en fonction de la vitesse de convergence de AGA, afin d'accélérer la convergence en exécutant AMA lorsque la progression est trop lente. Le concept général de l'Algorithme Génétique/Mimétique Adaptatif ($AGMA$), décrit dans l'algorithme 5.3, consiste à calculer, durant le déroulement de AGA, une valeur de progrès P_{PO*} correspondant au taux de modification subi par l'archive Pareto PO^* durant les dernières générations. Si ce taux devient plus petit qu'une valeur de seuil α, alors on lance une génération de recherche mimétique sur la population courante. Une fois la génération de recherche mimétique terminée, PO^* et α sont mis à jour, et AGA est relancé avec son ancienne population, mais avec une archive Pareto *a priori* améliorée.

Comme pour AGA+AMA, $AGMA$ est une coopération de type **HTH(AGA+AMA)**. Le schéma général de ces deux coopérations est illustré dans les figures 5.4 et 5.5 (C étant une règle de transition et $\bar{C}$ son contraire).

Algorithme 5.3 *$AGMA$.*

Création d'une population initiale aléatoire.
tant que le temps d'exécution n'est pas atteint **faire**
  Réaliser une génération de AGA.
  Mise à jour de PO^* et de P_{PO^*}.
  si $P_{PO^*} < \alpha$ **alors**
    Réaliser une génération de AMA sur la population de AGA (Algorithme 5.2).
    Mise à jour de PO^* et de P_{PO^*}.
  fin si
fin tant que
Retourner PO^*.

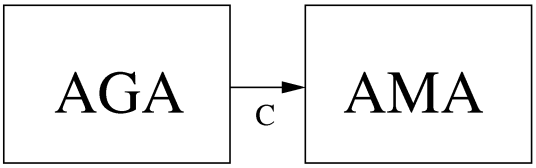


FIG. 5.4 – Schéma de coopération de $AGA + AMA$. La transition entre AGA et AMA est prédéfinie. Elle est réalisée selon le critère C ne dépendant pas du déroulement de l'algorithme.

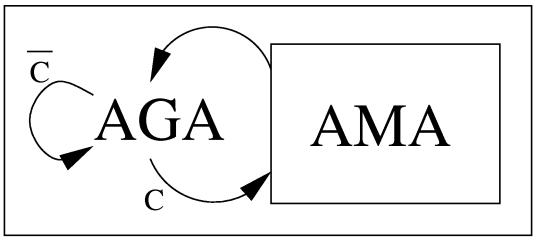


FIG. 5.5 – Schéma de coopération de $AGMA$. La transition entre AGA et AMA dépend d'un critère qui est vérifié selon le déroulement de la recherche.

5.2.3.2 Implémentation

Quelques choix ont été nécessaires pour l'implémentation de $AGMA$:

- Individus sélectionnés pour la recherche locale : les algorithmes mimétiques classiques sélectionnent certains individus de la population courante afin d'y appliquer un algorithme de descente. Ces algorithmes sont appliqués à chaque individu de manière séquentielle. Dans notre cas, PLS implique de réaliser la recherche locale de manière parallèle sur l'ensemble des individus sélectionnés. Une portion de ces individus sont issus de la sélection élitiste et font partie de l'archive Pareto, et ces individus se trouvent généralement déjà dans un optimum local. De plus, PLS ne garde durant la recherche que les individus non-dominés, la population participant à la recherche locale est donc en grande partie issue des individus de l'archive Pareto et ne peut être améliorée. Afin d'éviter ce problème et de faire participer à la recherche locale la structure de solutions moins bonnes, nous avons choisi de réaliser des croisements entre les individus sélectionnés. Les solutions ainsi créées servent alors de population initiale pour PLS. Ainsi on évite de réaliser des recherches sur des individus de l'archive se trouvant dans des optimums locaux, et on apporte encore un petit plus au niveau de l'exploration de l'espace de recherche grâce aux croisements réalisés. On évite également d'avoir des recherches locales successives semblables, lorsque PLS est lancée plusieurs fois consécutivement.
- État de la population courante après PLS : à la fin de chaque recherche locale, on met à jour l'archive Pareto. Quant à la population courante de AGA, on ne la modifie pas (approche *lamarckienne*). En effet, laisser intacte la population courante permet de ne pas perdre l'effort d'exploration fourni par AGA en remplaçant des solutions de la population par une population de solutions issues de PLS qui n'offrent aucune garantie de diversité (il est possible que la population Pareto issue de la recherche locale n'aie qu'un seul individu à la source !). Cependant les solutions Pareto de l'archive trouvée par PLS participeront quand même à l'opérateur de sélection de AGA par le biais de l'élitisme, ces solutions influeront en effet l'orientation des recherches à long terme, ce qui est indispensable.
- Paramètres des règles : pour le seuil correspondant à la vitesse de convergence mini-

male de l'archive Pareto AGA, nous maintenons une archive contenant le nombre de modifications subies par l'archive Pareto durant les N dernières générations de l'algorithme. Pour nos expérimentations, nous avons établi que la vitesse de convergence minimale est de 10 modifications sur les 500 dernières générations. Il est certain que l'implémentation de ce mécanisme pour d'autres problèmes nécessite de redéfinir ces valeurs, selon la quantité de modifications potentiellement subies par l'archive Pareto (est-ce que les meilleures solutions sont souvent améliorées ?) et la cardinalité de celui-ci (un front Pareto dense est susceptible de subir plus de modifications durant la recherche qu'un front Pareto contenant peu de solutions). Une solution pourrait être de définir la règle de transition selon un opérateur de performance. En optimisation mono-objectif, cette règle correspond au rapport entre la solution optimale actuelle et celle connue il y a N générations. Pour le multi-objectif, on peut comparer par exemple les aires de dominance (métrique S) de l'ancienne et de la nouvelle archive Pareto.

Au fur et à mesure que $AGMA$ converge, il semble naturel qu'il soit fait de plus en plus souvent appel à AMA. En effet, le taux des modifications effectuées à l'archive Pareto par $AGMA$ baisse avec le temps (un exemple explicite est proposé dans la figure 5.6). Ceci est une propriété intéressante, car au début de l'algorithme il est primordial d'étendre la recherche afin de ne pas négliger des zones de l'espace de recherche. Mais, plus l'algorithme converge, plus il est primordial d'effectuer des phases d'intensification de la recherche sur les régions de l'espace de recherche qui se sont révélées intéressantes.

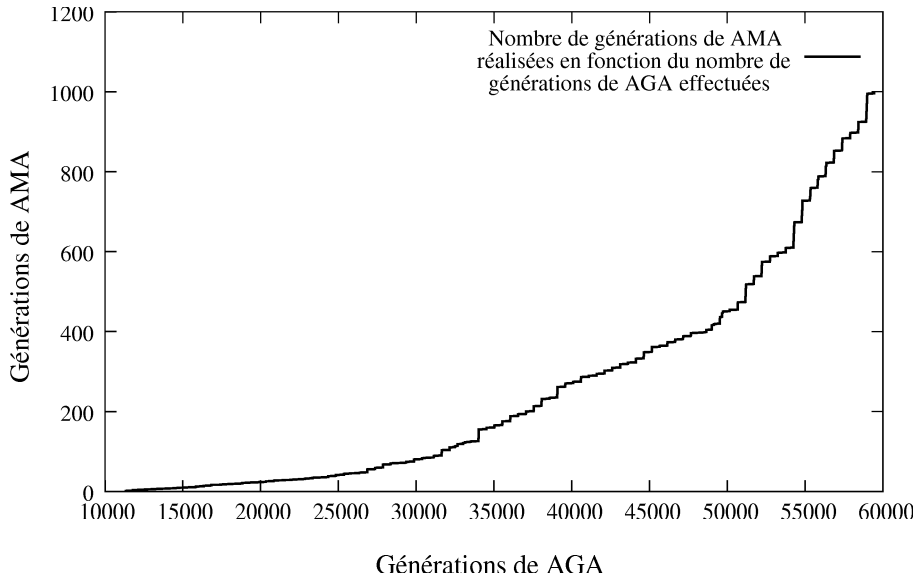


FIG. 5.6 – Balance AGA/AMA : la fréquence des recherches mimétiques réalisées augmente durant le déroulement de AGA (exemple pour l'instance $ta\_50\_20\_01$).

5.2.4 Expérimentations

Les résultats procurés par les différentes méthodes hybrides sont bien au delà de ceux obtenus par AGA seul. En effet, l'utilisation de PLS comme accélérateur de convergence permet d'obtenir les fronts exacts sur les petits problèmes en des temps bien en deçà de ceux alloués aux tests réalisés pour AGA.

Dans un premier temps, nous comparons AMA avec $AGMA$, afin de justifier le maintien de l'algorithme génétique dans l'approche globale. Puis nous montrerons l'apport de l'approche adaptative $AGMA$ par rapport à l'approche hybride simple $AGA + AMA$.

5.2.4.1 Comparaison $AGMA/AMA$.

Pour comparer $AGMA$ et AMA nous allons dans un premier temps évaluer le temps nécessaire pour découvrir le front de Pareto optimal pour les petites instances du problème. Puis nous utiliserons les indicateurs de performances S et $Contribution$ afin d'évaluer les ensembles de solutions non-dominées trouvés pour les instances plus grandes.

Le tableau 5.1 représente les temps d'exécution nécessaires pour trouver les ensembles Pareto optimaux avec AMA. $\mathbf{T_{Min}}$, $\mathbf{T_{Max}}$ et $\mathbf{T_{Moy}}$ représentent respectivement les temps minimum, maximum et moyen nécessaires à l'algorithme pour trouver la totalité des solutions Pareto optimales. **ET** est l'écart type calculé entre les temps des différentes exécutions sur une même instance (10 tests par instance).

TAB. 5.1 – Temps de recherche de AMA pour trouver les ensembles Pareto optimaux.

Instance	$\mathbf{T_{Min}}$	$\mathbf{T_{Max}}$	$\mathbf{T_{Moy}}$	ET
$ta\_20\_5\_01$	2”	42”	15”	14”
$ta\_20\_5\_02$	8”	4’25”	1’14”	1’24”
$ta\_20\_10\_01$	2’27”	23’50”	9’46”	6’02”
$ta\_20\_10\_02$	10’20”	64’36”	30’35”	18’47”
$ta\_20\_20\_01$	3’49”	67’24”	25’10”	20’40”
$ta\_50\_5\_01$	26’03”	385’30”	155’03”	120’18”

Le tableau 5.2 représente le temps qui a été nécessaire à la découverte des ensembles Pareto optimaux par $AGMA$. Notons que, pour l'ensemble des tests réalisés, les deux algorithmes ont toujours fini par découvrir le front de Pareto exact en un temps 'raisonnable'. Cette performance est une différence primordiale avec AGA, ce qui montre l'apport important du mécanisme d'intensification, sans lequel les algorithmes évolutionnaires restent peu efficaces.

La comparaison des résultats de $AGMA$ avec ceux de AMA présentés dans les tableaux 5.1 et 5.2 nous apporte plusieurs enseignements. Tout d'abord, si l'on compare les temps moyens d'exécution, on remarque que $AGMA$ est en moyenne plus rapide que AMA, hormis pour la petite instance qu'est $ta\_20\_5\_01$, pour laquelle les temps d'exécution moyens sont très petits. Les apports sont variables mais assez conséquents pour les

instances $ta\_20\_10\_02$ et $ta\_20\_20\_01$, où les temps sont divisés par deux environ. Si on compare maintenant les temps extrêmes obtenus pour les différentes exécutions réalisées, on remarque que AMA est toujours plus rapide que $AGMA$ dans le meilleur des cas, et toujours moins rapide que $AGMA$ dans le pire des cas. Cette particularité se reflète dans la valeur des écarts type calculés sur les différentes instances, ce qui correspond à une évaluation de la robustesse de l'algorithme testé[1]. En effet, les petites valeurs d'écart type calculées pour $AGMA$ montrent une certaine régularité dans les résultats obtenus. Donc les résultats indiquent que $AGMA$ offre un avantage au niveau de la robustesse, en plus de l'amélioration générale des résultats obtenus.

Notons que les différentes instances des tables 5.1 et 5.2 ont toutes été résolues de manière exacte dans le chapitre 7, excepté l'instance $ta\_50\_5\_01$. Cependant, des longues exécutions ont montré que $AGMA$ et AMA trouvent toujours le même ensemble de solutions. On peut conjecturer que cet ensemble constamment trouvé est l'ensemble Pareto optimal.

TAB. 5.2 – Temps de recherche de $AGMA$ pour trouver les ensembles Pareto optimaux.

Instance	$\mathbf{T_{Min}}$	$\mathbf{T_{Max}}$	$\mathbf{T_{Moy}}$	**ET**
$ta\_20\_5\_01$	10”	34”	20”	6”
$ta\_20\_5\_02$	40”	1’41”	1’01”	18”
$ta\_20\_10\_01$	5’02”	14’27”	9’08”	2’59”
$ta\_20\_10\_02$	4’45”	36’25”	18’29”	10’04”
$ta\_20\_20\_01$	7’59”	21’03”	12’26”	4’48”
$ta\_50\_5\_01$	38’09”	365’16”	139’38”	106’25”

5.2.4.2 Comparaison $AGMA/AGA + AMA$.

Les tests de l'algorithme coopératif $AGA + AMA$ ne sont pas aisés puisqu'il faut définir quel est le moment opportun pour lancer l'hybridation par AMA. Faut-il lancer l'algorithme mimétique tôt dans la recherche ou en toute fin de recherche ? Pour évaluer la performance de $AGMA$ par rapport à $AGA + AMA$, nous avons réalisé nos expérimentations sur 200 hybridations (recherches locales PLS). Afin d'avoir des expérimentations les plus équitables possibles, nous avons, pour les expérimentations de $AGA + AMA$, réalisé approximativement le même nombre total de générations de AGA que pour nos expérimentations pour l'algorithme $AGMA$. Ainsi, le temps et le nombre de générations des différents tests effectués sont à peu près identiques. Les temps d'exécution varient entre 1 minute (instance $ta\_20\_5\_01$) et 3 heures (instance $ta\_50\_20\_01$).

Le tableau 5.3 présente les valeurs moyennes obtenues respectivement pour les indicateurs de $Contribution$ et S. Les résultats obtenus pour la métrique $Contribution(AGMA/AGA+AMA)$ sont supérieurs à 0.5 en moyenne pour chaque instance traitée. Donc, selon cette

[1] Une bonne “robustesse” indique que les résultats de l'algorithme stochastique testé sont peu sensibles aux choix aléatoires réalisés durant les expérimentations.

métrique, $AGMA$ est plus performant que $AGA + AMA$. De plus, pour les deux plus grosses instances traitées, les valeurs moyennes de contribution sont de 0.92 et 0.984, on peut donc en déduire qu'une grande majorité des solutions obtenues par $AGMA$ dominent celles obtenues par $AGA + AMA$.

Les calculs effectués pour la métrique S confirment ceux réalisés pour la métrique *Contribution*. En effet, pour les deux plus grosses instances, l'aire de dominance de $AGMA$ dépasse de plus de 20% en moyenne celle de $AGA + AMA$. Pour les autres instances, la différence est moindre, mais toujours à l'avantage de $AGMA$. Notons tout de même une grosse différence sur les instances $ta\_20\_5\_02$ et $ta\_20\_20\_01$ (respectivement 9.61% et 14.57%).

TAB. 5.3 – Comparaison $AGMA/AGA + AMA$: métriques *Contribution* et S.

Instance	Contribution (AGMA/AGA + AMA)	S		
		AGMA	AGA + AMA	Apport
$ta\_20\_5\_01$	**0.657**	4778.1	4707.8	**1.49%**
$ta\_20\_5\_02$	**0.739**	6437.9	5873.5	**9.61%**
$ta\_20\_10\_01$	**0.751**	322121.5	304436.7	**5.81%**
$ta\_20\_10\_02$	**0.732**	180378.2	172749.7	**4.42%**
$ta\_20\_20\_01$	**0.754**	506460.0	442050.9	**14.57%**
$ta\_50\_5\_01$	**0.690**	146462.6	141386.0	**3.59%**
$ta\_50\_10\_01$	**0.920**	1249924.3	1041016.6	**20.07%**
$ta\_50\_20\_01$	**0.984**	2954461.3	2451656.3	**20.51%**

Le tableau 5.4 décrit les meilleures valeurs obtenues par $AGMA$ et $AGA+AMA$ pour chaque objectif :

- UB (*Upper Bound*) désigne la meilleure valeur obtenue pour le makespan pour l'ensemble des études mono-objectifs de la littérature.
- C_1 et T_1 sont les meilleurs C_{Max} et T obtenus avec $AGA + AMA$.
- C_2 et T_2 sont les meilleurs C_{Max} et T obtenus avec $AGMA$.
- $Dèv$ est l'écart entre C_2 et UB, ce qui correspond à l'écart entre notre meilleure solution obtenue par $AGMA$ pour le C_{max} par rapport à la meilleure solution de la littérature mono-objectif.

En observant les résultats du tableau 5.4, nous pouvons remarquer qu'avec $AGMA$, on obtient la solution optimale pour le C_{max} pour les six premières instances, en moins de 200 hybridations (soit, pour ces instances, quelques minutes). Par contre, la version coopérative simple $AGA + AMA$ n'atteint ces optima que sur deux instances.

Les figures 5.7 et 5.8 montrent des exemples de résultats obtenus pour deux différentes instances. Trois ensembles Pareto sont représentés sur ces figures : celui correspondant à $AGMA$, celui correspondant à $AGA + AMA$, et pour indication celui correspondant à $AGA + PLS$ (soit une recherche locale et une seule en fin d'AG). $AGA + PLS$ est beaucoup moins efficace que $AGA + AMA$, ce qui confirme l'intérêt d'une recherche de type mimétique par rapport à une recherche locale simple. Par contre les solutions Pareto

TAB. 5.4 – Évaluation des performances : approche mono-objectif (meilleurs résultats obtenus en gras).

Instance	UB	C_1	C_2	Dév	T_1	T_2
*ta*_20_5_01	1278	**1278**	**1278**	**0%**	**452**	**452**
*ta*_20_5_02	1359	**1359**	**1359**	**0%**	491	**469**
*ta*_20_10_01	1582	1586	**1582**	**0%**	**1224**	**1224**
*ta*_20_10_02	1659	1672	**1659**	**0%**	**1275**	**1275**
*ta*_20_20_01	2297	2308	**2297**	**0%**	1097	**1031**
*ta*_50_5_01	2724	2729	**2724**	**0%**	3364	**3231**
*ta*_50_10_01	2991	3063	**3025**	**1.14%**	**4636**	4706
*ta*_50_20_01	3850	3933	**3904**	**1.27%**	7667	**7214**

obtenues par $AGMA$ sont meilleures que celles obtenues par $AGA+AMA$, ce qui confirme graphiquement l'intérêt de l'utilisation d'une transition adaptative entre AGA et AMA.

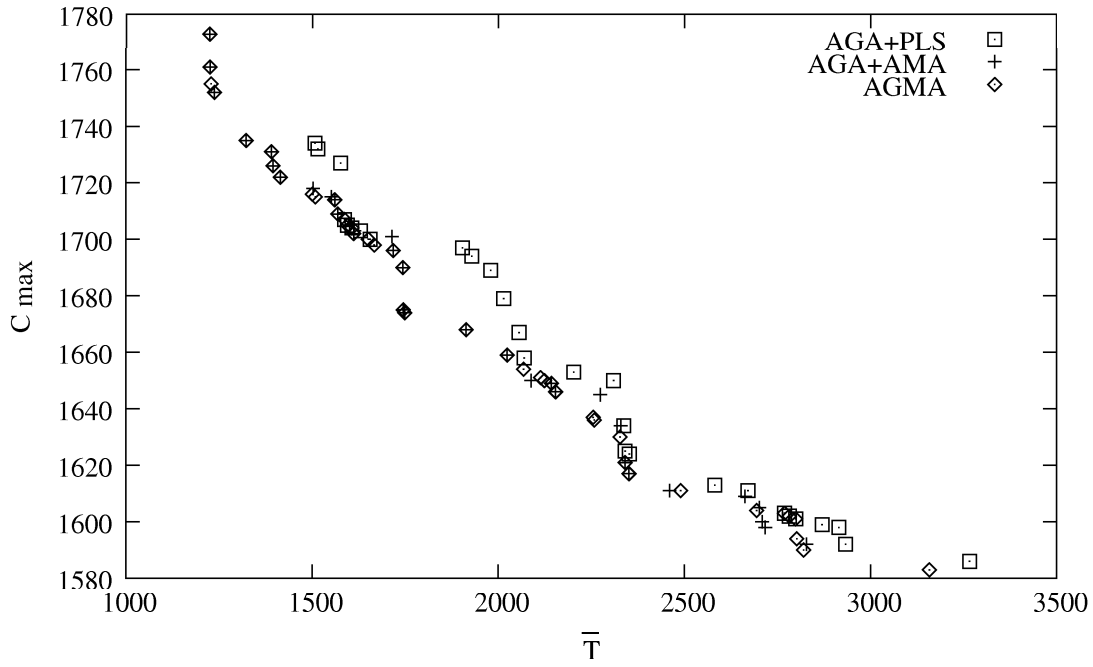


FIG. 5.7 – Comparaison $AGMA/AGA+AMA$: exemple d'ensembles Pareto obtenus pour l'instance *ta*_20_10_01.

Les différents résultats que nous venons d'exposer nous montrent tout d'abord que la méthode coopérative $AGMA$ est beaucoup plus efficace que AGA grâce notamment au mécanisme d'intensification PLS. La différence entre les deux algorithmes est grande puisque, là où AGA ne découvre pas les ensembles Pareto optimaux, $AGMA$ les découvre tout le temps, et en un temps largement acceptable.

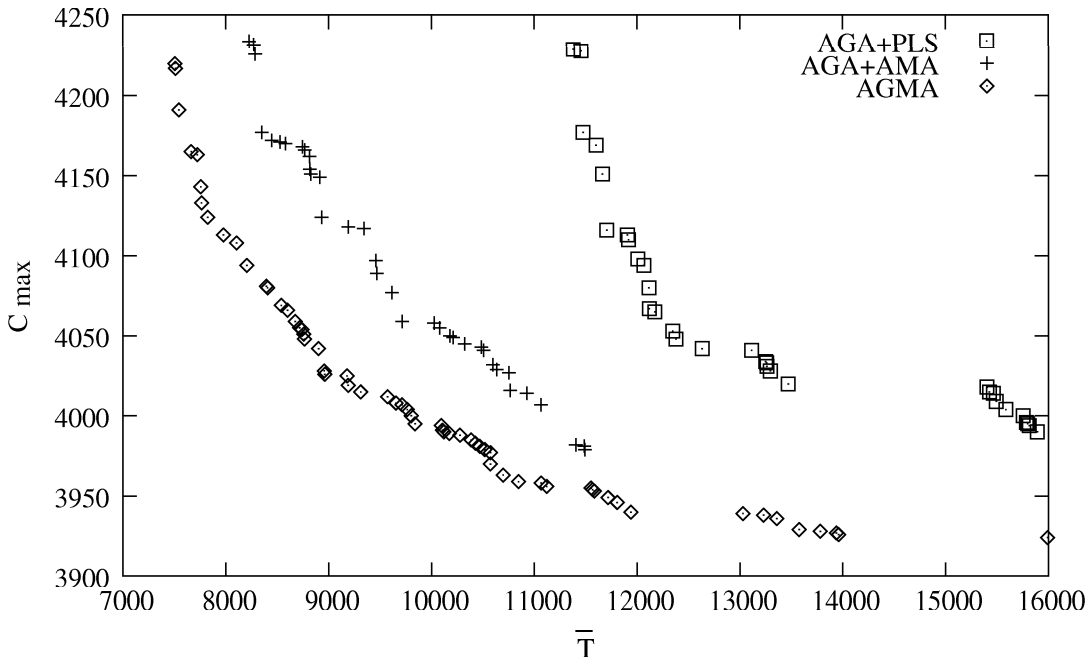


FIG. 5.8 – Comparaison $AGMA/AGA+AMA$: exemple d'ensembles Pareto obtenus pour l'instance $ta\_50\_20\_01$.

Nous avons également montré l'efficacité de $AGMA$ sur deux aspects :
- En comparant $AGMA$ avec AMA, nous avons montré l'intérêt de garder l'approche AGA, bien qu'elle n'intervienne pas directement dans la progression de l'archive Pareto.
- En comparant $AGMA$ avec $AGA+AMA$, nous avons montré l'intérêt du mécanisme de transition adaptative entre les deux méthodes d'optimisation.

5.3 Coopération parallèle

Parmi les méthodes coopératives pour l'optimisation de la littérature, celles dédiées au calcul parallèle forment une classe incontournable. Le modèle le plus connu est certainement celui de type "modèle en îles". Ce modèle de coopération de type **HTH** consiste à exécuter de manière parallèle plusieurs méthodes d'optimisation, en réalisant à certains moments de la recherche des migrations d'individus ou d'informations entre les différents processeurs.

La plupart du temps, ce type de coopération se fait avec plusieurs méthodes d'optimisation de même type. Cependant, on peut faire coopérer de cette manière différentes méthodes d'optimisation, ou encore utiliser des paramétrages différents d'une même méthode d'optimisation.

Dans cette section, nous abordons les coopérations de type "modèle en îles". Nous avons choisi de paralléliser l'algorithme $AGMA$, qui est le plus performant réalisé, et qui de plus

gère automatiquement les transitions entre AGA et AMA. Dans un premier temps, nous proposons un modèle en îles dédié à la coopération parallèle sur quelques machines, et nous évaluons l'apport de ce type de coopération. Ensuite nous présentons des implémentations dédiées au calcul massivement parallèle, ainsi que quelques résultats obtenus.

5.3.1 Coopération sur une machine parallèle

Dans un premier temps nous proposons une coopération entre plusieurs $AGMA$ s'exécutant chacun indépendamment. Seules quelques migrations d'individus permettront à chaque $AGMA$ de profiter des bonnes solutions des autres. Donc la coopération décrite ci-dessous est de type **HTH(AGMA+AGMA)***. Si l'on détaille toutes les coopérations réalisées, l'algorithme coopératif global est de type **HTH(HTH(AGA+LTH(AGA(PLS))) , HTH(AGA+LTH(AGA(PLS))))***.

5.3.1.1 Description

Nous utilisons un modèle maître-esclave, où le processeur maître centralise l'ensemble des solutions non-dominées des processeurs esclaves (voir figure 5.9). Les communications s'opèrent après chaque génération de AMA, ce qui correspond au moment où l'archive Pareto du processus concerné vient de subir les plus grandes modifications grâce au mécanisme d'intensification de AMA. Ainsi le moment où ont lieu les communications dépend de l'évolution de l'algorithme $AGMA$ (voir figure 5.10).

Le déroulement des algorithmes des processus maîtres est détaillé dans l'algorithme 5.4, celui des processus esclaves dans l'algorithme 5.5.

Algorithme 5.4 $PAGMA$: processeur maître.

Lancer k processus esclaves.
$PO^* \leftarrow \emptyset$.
tant que critère d'arrêt non atteint **faire**
  Attendre de recevoir l'archive Pareto PO_i d'un processeur P_i.
  $PO^* \leftarrow$ solutions non-dominées de $PO^* \cup PO_i$.
  Envoyer PO^* au processeur P_i.
fin tant que
Retourner PO^*.

5.3.1.2 Implémentation

Pour nos expérimentations, nous avons utilisé un super-calculateur à architecture parallèle $IBMRS6000/SP$ installé au Centre de Ressources Informatiques de l'Université des Sciences et Technologies de Lille. Nous avons réalisé nos expérimentations sur ce super-calculateur, en utilisant 8 processeurs Power4 à 1,1GHz. La machine dispose de 16 Go RAM.

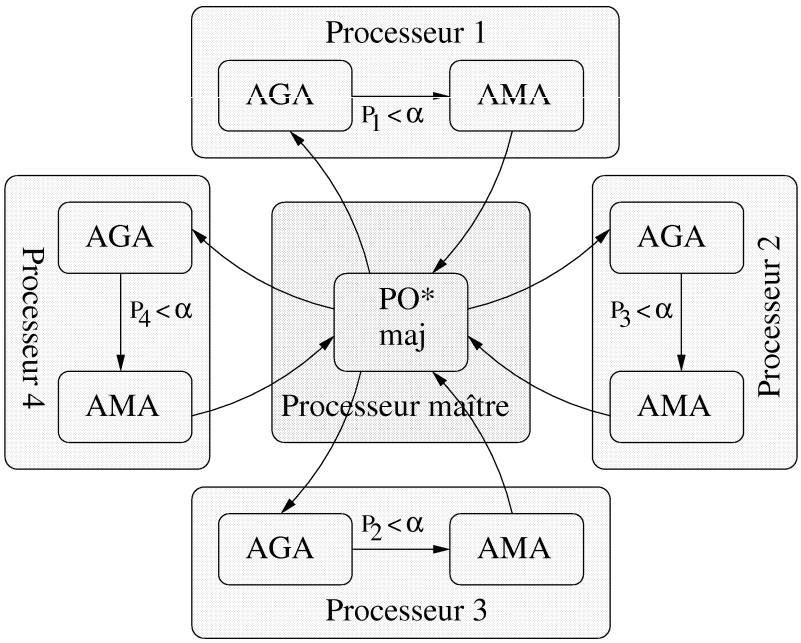


FIG. 5.9 – Modèle parallèle : exemple avec 5 machines. Le processeur centralise les différentes archives Pareto des processeurs, en calculant à chaque communication l'ensemble des solutions non-dominées. Il renvoie ensuite au processeur esclave le résultat de l'union des ensembles.

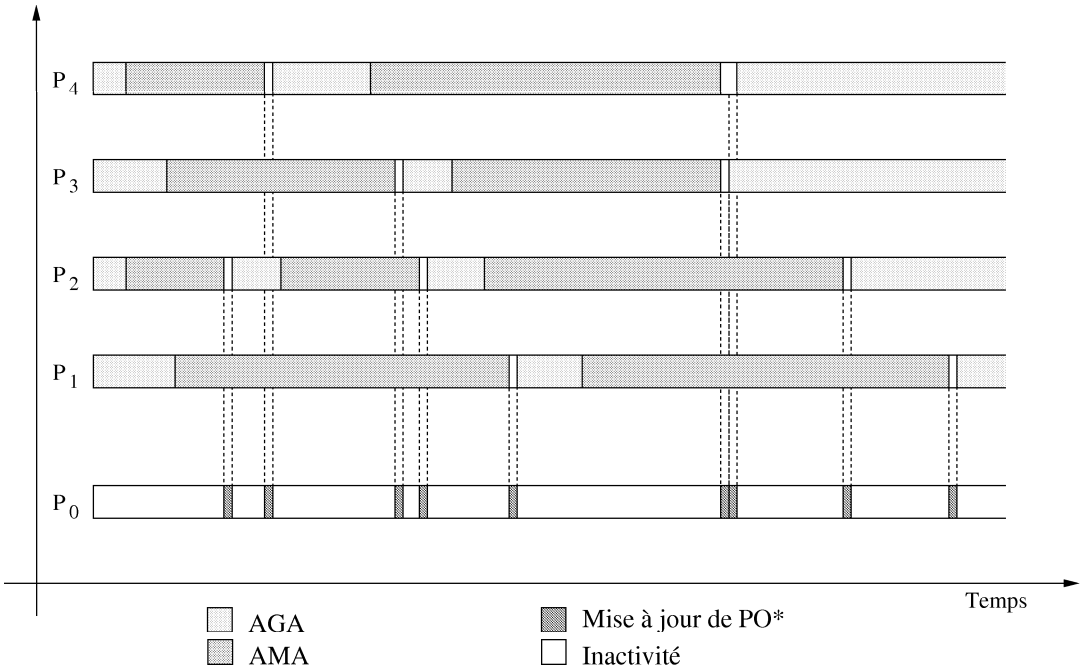


FIG. 5.10 – Déroulement de $PAGMA$: exemple sur 5 processeurs. P_0 est le processeur maître, qui met à jour l'archive Pareto PO^* en fonction des archives Pareto secondaires qu'il reçoit.

Algorithme 5.5 $PAGMA$: processeur esclave P_i.

Création d'une population initiale aléatoire.
tant que critère d'arrêt du processus maître non atteint **faire**
  Réaliser une génération de AGA.
  Mise à jour de PO_i et de P_{PO_i}.
  si $P_{PO_i} < \alpha$ **alors**
    Réaliser une génération de MA sur la population population courante de AGA (Algorithme 5.2).
    Mise à jour de PO_i et de P_{PO_i}.
    Envoyer P_{PO_i} au processus maître.
    Recevoir P_{PO_i} du processus maître.
  fin si
fin tant que

Le critère d'arrêt choisi a été de stopper l'exécution quand le nombre total de recherches locales effectuées atteint 1000 (soit de quelque minutes à quelques heures de calcul) pour les instances jusqu'à $ta\_50\_20\_01$, afin de pouvoir comparer avec une exécution séquentielle de $AGMA$. Pour les plus grosses instances, nous avons stoppé les exécutions au bout de 24 heures, le but étant d'aller le plus loin possible dans la recherche.

5.3.1.3 Expérimentations

Nous avons comparé dans un premier temps l'approche coopérative parallèle de $AGMA$ avec des exécution séquentielles similaires de 1000 recherches locales.

Comme le montrent les figures 5.11 et 5.12, le modèle de coopération parallèle en îles permet d'améliorer les résultats, en ne réalisant pas plus de recherches locales, à l'origine de la convergence de l'archive Pareto.

Les tableaux 5.5 et 5.6 détaillent les valeurs obtenues pour les métriques S et $Contribution$ pour les deux approches. D'une manière générale, les valeurs obtenues confirment l'intérêt de la coopération parallèle. En effet, les aires de dominance moyenne calculées sont améliorées sensiblement. De plus, la quasi-totalité des solutions Pareto obtenues par la fusion des ensembles Pareto de $AGMA$ et $PAGMA$ sont obtenues par $PAGMA$ (en moyenne 96.9% pour l'instance $ta\_50\_10\_01$ et 87% pour l'intance $ta\_50\_20\_01$ - table 5.6). De plus, si l'on observe l'écart type entre les différentes exécutions effectuées, on remarque que $PAGMA$ permet de diminuer cet écart type de 92.8% pour l'instance $ta\_50\_10\_01$ et de 83.4% pour l'instance $ta\_50\_20\_01$. En conclusion, en plus d'apporter une amélioration au niveau de la convergence, $PAGMA$ offre surtout une robustesse des résultats nettement plus importante que $AGMA$.

Nous avons également profité de ce modèle parallèle pour traiter les instances plus grandes du $BOFSP$, soit les instances $ta\_100\_5\_01$, $ta\_100\_10\_01$, $ta\_100\_20\_01$ et $ta\_200\_10\_01$. Les résultats obtenus pour ces instances se trouvent dans l'annexe 8.

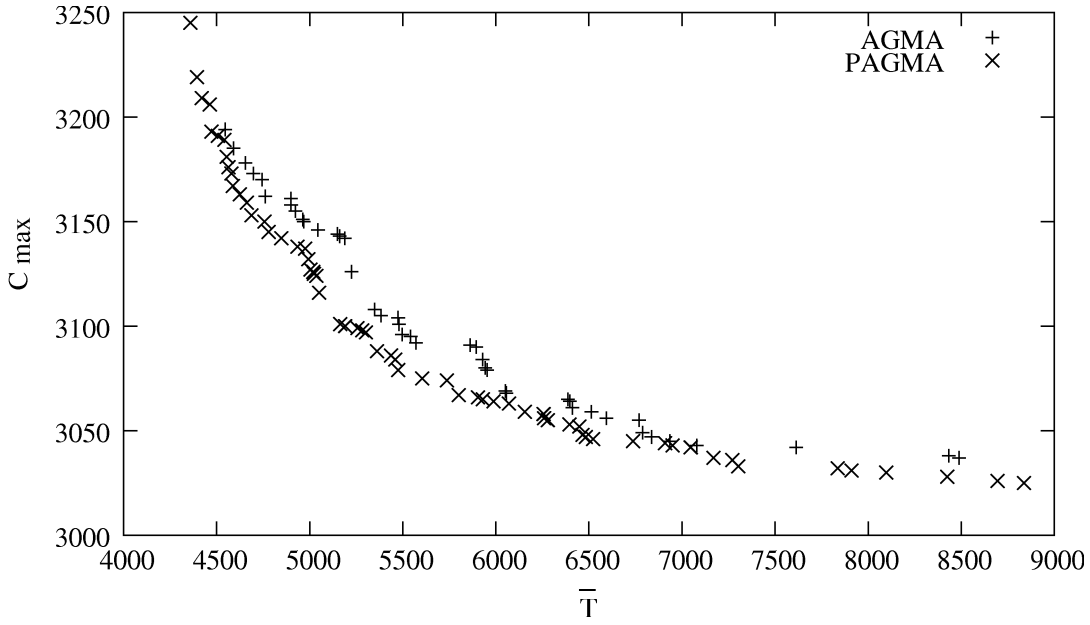


FIG. 5.11 – Résultat obtenu sur l'instance $ta\_50\_10\_01$ par $AGMA$ et $PAGMA$.

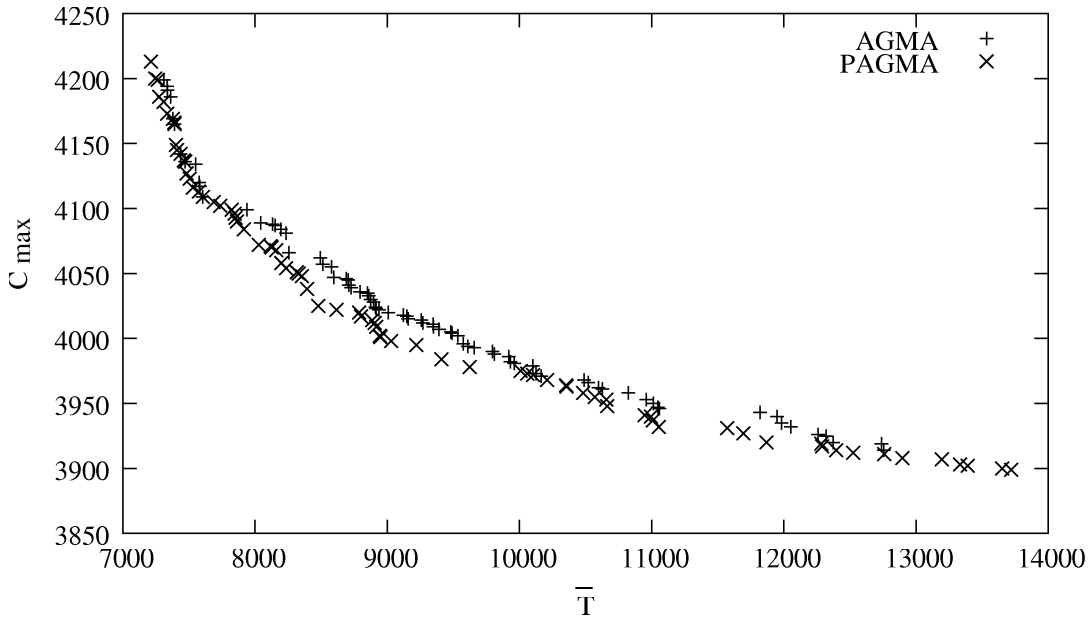


FIG. 5.12 – Résultat obtenu sur l'instance $ta\_50\_20\_01$ par $AGMA$ et $PAGMA$.

TAB. 5.5 – Comparaison $PAGMA/AGMA$: métrique S.

Instance	S(AGMA)					
	Min	**Max**	**Moy**		**ET**	
*ta*_50_10_01	776839	926870	858405.8		53501.6	
*ta*_50_20_01	2891579	3033353	2976029.6		41821.3	
Instance	**S(PAGMA)**					
	Min	**Max**	**Moy**	**Apport**	**ET**	**Apport**
*ta*_50_10_01	950313	963496	958079.2	11.61%	3831.5	-92.8%
*ta*_50_20_01	3065968	3088282	3079267.2	3.47%	6945.1	-83.4%

TAB. 5.6 – Comparaison $PAGMA/AGMA$: métrique *Contribution*.

Instance	Contribution(PAGMA/AGMA)			
	Min	**Max**	**Moyenne**	**ET**
*ta*_50_10_01	0.87	1.00	0.969	0.049
*ta*_50_20_01	0.70	1.00	0.870	0.092

5.3.2 Coopération parallèle pair à pair

5.3.2.1 Motivations

Pour des instances de problèmes d'optimisation de dimension importante, la résolution sera donc toujours limitée par les ressources utilisées. L'utilisation du déploiement à large-échelle de type pair à pair (*peer to peer* - $P2P$) nous permet alors d'aboutir à des solutions satisfaisantes en un temps raisonnable. Les systèmes de calcul $P2P$ tels que `XtremWeb` [FGNC01] ou `Seti@Home` [ACK+02] permettent la conception et le développement d'un environnement de calcul $P2P$.

5.3.2.2 Description

Nous avons envisagé la coopération parallèle $P2P$ à deux niveaux différents. La première approche est toujours celle du modèle en îles, qui a fait ses preuves sur de nombreux problèmes. Le modèle insulaire est déployé sur des clients, où à chaque $AGMA$ correspond un client particulier. Donc la coopération décrite ici est de même type que celle correspondant à $PAGMA$.

La deuxième approche consiste à paralléliser la partie de $AGMA$ la plus lourde en temps de calcul : la recherche locale PLS. En effet, lorsque la taille des problèmes augmente, il devient de plus en plus coûteux de réaliser une recherche locale complète. Donc, lors de la phase d'hybridation par AMA, les recherches locales PLS, initiées à quelques solutions, sont soumises à un *dispatcher*. Ce dernier aura la charge de les déployer sur les *workers*. Chaque *worker* réalise alors une recherche de voisinage avant de renvoyer son résultat. Ensuite, il attend un nouveau travail, c'est-à-dire un nouveau voisinage à explorer. Donc la coopération décrite ici est de type **HTH(AGA+LTH(AGA(LTH(**PLS_i**,**PLS_i**)*)))**, où PLS_i est une génération de voisinage de PLS. Une combinaison de ces approches est présentée dans la figure 5.13.

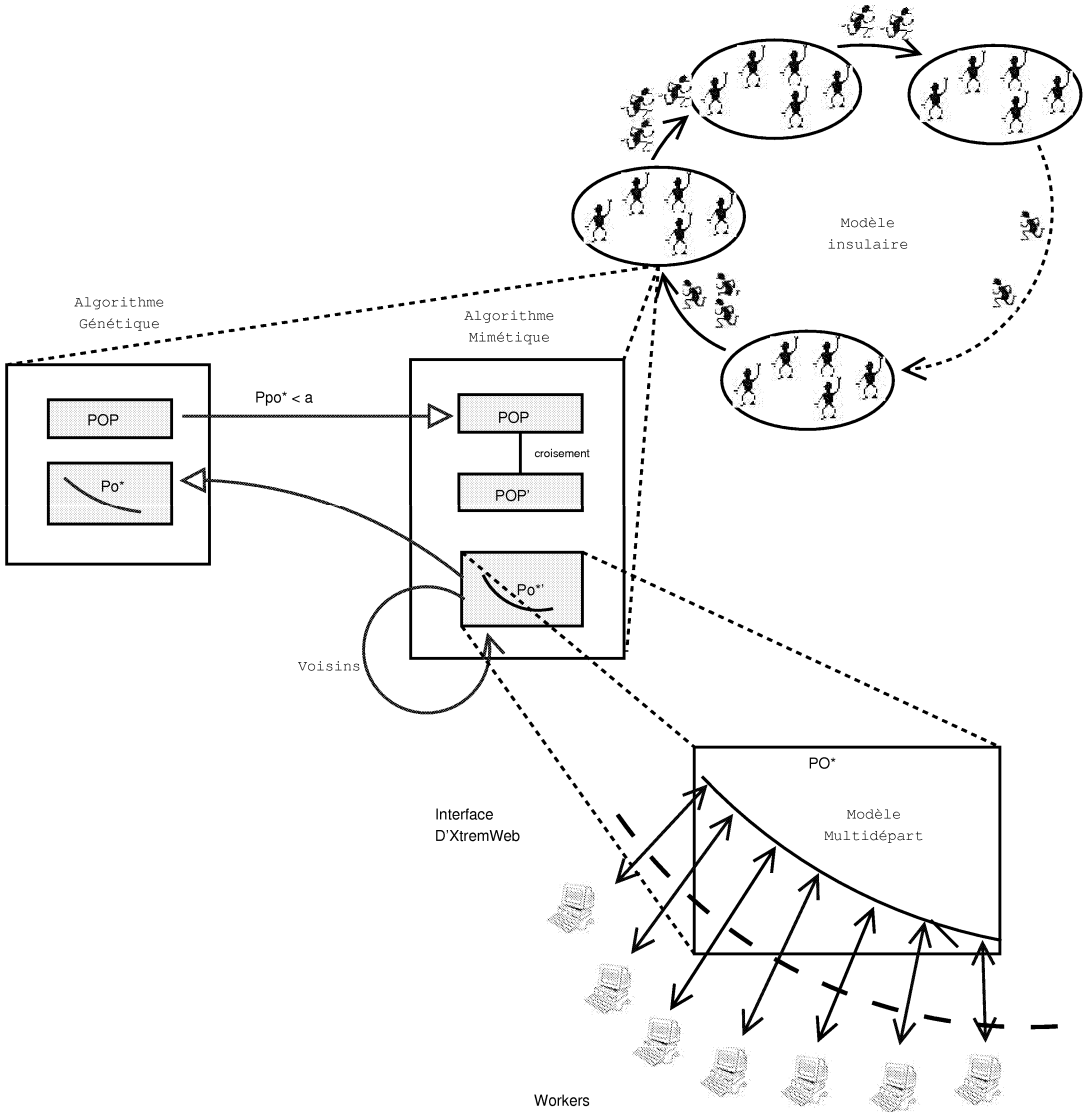


FIG. 5.13 – Illustation d'un modèle $P2P$ pour $AGMA$. Le schéma illustre une coopération de type modèle en îles, où chaque île utilise une flotte de *workers* pour réaliser les recherches locales PLS.

5.3.2.3 Implémentation

Pour nos expérimentations, nous avons considéré trois versions de déploiement parallèle de $AGMA$.

1. *Version 1* : C'est la version parallèle $PAGMA$ présentée dans la section précèdente.
2. *Version 2* : Même principe, sauf que la phase d'hybridation est déployée de manière distribuée sur un ensemble de *workers* `XtremWeb` suivant un modèle de vol de cycles et le déploiement est à l'initiative des workers.
3. *Version 3* : Cette version utilise le modèle insulaire et multidépart, trois $AGMA$ déployés sur des machines clientes coopèrent suivant le modèle insulaire. Chaque $AGMA$ est une implémentation de la version 2.

Dans nos expérimentations, nous avons considéré l'instance $ta\_200\_10\_01$. Les paramètres du modèle insulaire sont fixés de la manière suivante : le processus de migration intervient toutes les 10 générations, le nombre de migrants à chaque opération de migration est fixé à 20 si la taille de l'archive Pareto de l'$AGMA$ est supérieure à 20 et à la totalité dans le cas contraire. La taille de la population dans chaque $AGMA$ est fixée à 100.

Dans une application $P2P$, il est indispensable d'avoir un mécanisme de tolérance aux pannes. Les applications $P2P$ auxquelles on s'intéresse nécessitent un calcul intensif de plusieurs jours, voire plusieurs semaines. Le réseau et les machines n'étant pas fiables, l'exécution de ces applications n'est pas à l'abri des pannes matérielles et/ou logicielles. Dans $AGMA$ $P2P$, chaque client aura la charge de gérer le problème de pannes en utilisant le mécanisme de *checkpointing*, sauvegarde/restauration des données.

5.3.2.4 Expérimentations

La plate-forme matérielle d'expérimentation est le réseau d'enseignement de Polytech'Lille composé de 120 PCs sous Linux Debian. Les caractéristiques de ces PCs sont données dans le tableau 5.7.

TAB. 5.7 – PCs utilisés pour les expérimentations $P2P$.

Processeur	Nombre
AMD Duron(tm) Processor	14
Celeron (Coppermine)	14
Intel(R) Celeron(R)CPU 2.00 GHz	8
Intel(R) Celeron(R)CPU 2.20 GHz	28
Intel(R) Celeron(R)CPU 2.40 GHz	21
Intel(R) Celeron(R)CPU 1400 MHz	7
Pentium III (Katmai)	28
Total	120

Les résultats obtenus sont encore probants. La seule instance traitée pour le moment est l'instance $ta\_200\_10\_01$ pour laquelle le modèle 2, et surtout le modèle 3, améliorent

les résultats obtenus par $PAGMA$.

Le parallélisme à grande échelle basé sur l'utilisation des systèmes $P2P$ s'avère aujourd'hui un moyen potentiel pour disposer d'une puissance de calcul très importante. A notre connaissance, aucun travail de recherche n'a été publié sur les métaheuristiques parallèles hybrides à grande échelle. Les résultats obtenus pour les différentes versions parallèles montrent un apport non-négligable, en terme de qualité, mais aussi en terme de robustesse. Des expérimentations plus poussées sont à réaliser.

5.4 Conclusion

Dans ce chapitre, nous avons étudié les coopérations entre métaheurisques, et leurs applications pour le Flow-shop bi-objectif. Le but était de faire coopérer avec AGA des méthodes dites d'intensification. Nous avons dans un premier temps présenté PLS, une recherche locale Pareto sur laquelle nous avons basé toutes nos coopérations.

La première voie de coopération que nous avons explorée est très répandue puisqu'il s'agit des algorithmes de type mimétique. AMA, que nous avons appliqué au Flow-shop bi-objectif, a montré son efficacité. Nous avons néanmoins montré qu'il s'avérait intéressant de garder des périodes d'exécutions de AGA entre les différentes générations de recherche mimétique. Ceci à aboutit à un algorithme $AGMA$, coopératif entre AGA et AMA, où les transitions entre les deux algorithmes sont réalisées de manière adaptative durant l'exécution. Les expérimentations réalisées ont montré l'intérêt de $AGMA$ par rapport à AMA. De plus, la comparaison de $AGMA$ avec sa version sans mécanisme de transition adaptative ($AGA + AMA$) a montré l'intérêt de ce mécanisme.

Enfin, nous avons testé quelques coopérations parallèles afin notamment de s'attaquer aux instances de $BOFSP$ les plus grandes. Les différents modèles testés ont permis, en plus de pouvoir réaliser des recherches conséquentes sur les grandes instances, d'obtenir un algorithme global beaucoup plus robuste que la version séquentielle. Ce type de coopération est très prometteur, surtout avec l'avènement du calcul $P2P$. Les modèles coopératifs parallèles présentés ici pourraient être améliorés en utilisant des stratégies de communication plus évoluées.

D'une manière générale, les résultats obtenus se rapprochent globalement des valeurs optimales connues en optimisation mono-objectif. Les méthodes coopérant dans ce chapitre sont néanmoins assez classiques, même pour l'optimisation multi-objectif. Il semble intéressant de réaliser ce type de coopération avec une méthode plus récente et moins répandue, comme le *path relinking*, en l'adaptant au cas multi-objectif.

Chapitre 6

Path relinking multi-objectif coopératif

Dans ce chapitre, nous proposons MOPR, un algorithme de path relinking appliqué au cas multi-objectif. Puis nous réalisons une coopération entre MOPR, AGA et PLS, afin d'intensifier et de diversifier la recherche. Les résultats obtenus montrent les capacités de ce type d'approche. Ce travail a été publié dans la conférence EMO'05 [BST05c], et soumis à la revue Journal of Heuristics [BST05b].

6.1 Introduction

Les algorithmes évolutionnaires de recherche par dispersion, et leur forme généralisée, les algorithmes de *Path Relinking* (PR) - parfois traduit "chemin de laison" - ont prouvé leur efficacité en optimisation mono-objectif. Il nous a semblé intéressant d'évaluer leur efficacité dans le cas multi-objectif et coopératif. Dans cette section, nous proposons d'adapter ce concept à l'optimisation multi-objectif Pareto, dans le but d'établir une coopération avec AGA. L'algorithme de PR Multi-Objectif ($MOPR$) sera dédié à l'intensification de la recherche sur les solutions découvertes par AGA. Pour atteindre ce but, l'algorithme sera proposé sous une forme hybridée avec PLS.

Plusieurs questions classiques se posent lors de l'implémentation d'un algorithme de PR. Nous proposons, dans cette section, des réponses adaptées à l'optimisation multi-objectif et au problème étudié dans cette thèse : le Flow-shop bi-objectif. A notre connaissance, un algorithme de PR a été proposé par Yamada et Reeves [RY98] pour résoudre un problème de Flow-shop, mais dans une version mono-objectif (optimisation du C_{max}). Il existe peu d'applications multi-objectif d'un algorithme de *path relinking*. Cependant, nous pouvons citer le travail de Beausoleil sur la recherche par dispersion multi-objectif [Del01].

Nous proposons ici un algorithme de *path relinking* appliqué au cas multi-objectif. La méthode que nous proposons explore les chemins possibles en sélectionnant les solutions selon des critères de dominance. Pour appliquer cette méthode au $BOFSP$, il a fallu définir une notion de distance pour ce problème, ainsi qu'une méthode d'énumération des solutions à explorer. Afin de réaliser nos expérimentations et de justifier l'intérêt de cette approche, nous avons réalisé une coopération de $MOPR$ avec l'algorithme génétique AGA et la recherche locale Pareto PLS.

Dans la section 6.2, nous proposons un algorithme de *path relinking* multi-objectif appliqué au Flow-shop bi-objectif, puis nous proposons une approche coopérative pour cet algorithme dans la section 6.3. Dans la section 6.4 nous détaillons les expérimentations réalisées et nous concluons dans la section 6.5.

6.2 *Path relinking* multi-objectif pour le $BOFSP$

Le concept général des algorithmes de PR est schématisé dans la figure 6.1. Deux solutions sont sélectionnées parmi une population (composée, si possible d'individus déjà de bonne qualité). Puis, on "relie" ces deux solutions par applications successives d'un opérateur de voisinage. Ces deux solutions sont couramment appelées solutions *initiale* (*initiating solution*) et de *guidage* (*guiding solution*). Durant l'algorithme de "raccordement", les solutions intermédiaires sont évaluées, et on applique généralement un algorithme de recherche locale sur les meilleures d'entre elles.

D'une manière générale, beaucoup de questions se posent lors de la réalisation d'un tel algorithme. Des réponses ont été proposées dans le cadre de l'optimisation mono-objectif, mais celles-ci doivent être reconsidérées pour le cas multi-objectif. Nous avons les méca-

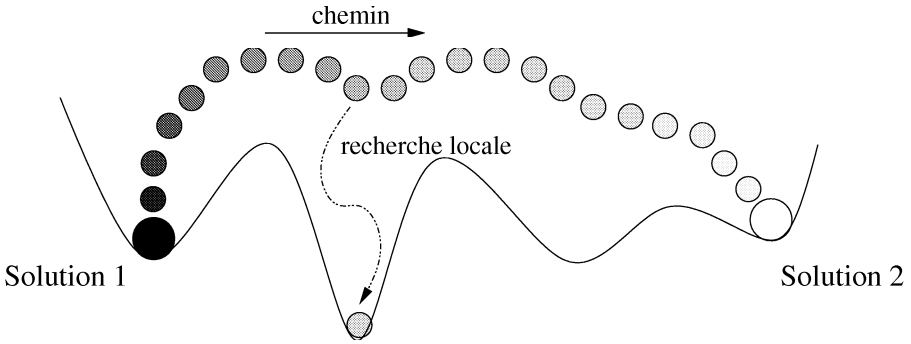


FIG. 6.1 – *Path Relinking* : le concept.

nismes suivants à définir :

- structure du voisinage utilisée ;
- mesure de la distance entre les solutions (corrélée ou non, avec le voisinage). ;
- critère de sélection pour le choix des solutions initiale et de guidage ;
- critère de sélection du (des) chemin(s) à parcourir ;
- amélioration éventuelle des solutions parcourues. Choix des solutions pour l'application du mécanisme d'intensification ;

Dans la suite de cette section, des propositions sont faites afin de répondre à ces difficultés, mais les questions restent bien entendu ouvertes à des mécanismes plus évolués.

6.2.1 Opérateur de voisinage

Comme pour beaucoup d'algorithmes d'optimisation, l'opérateur de voisinage influe grandement la qualité des résultats obtenus par l'algorithme de PR. Afin de réaliser l'opération de "raccordement" de la solution de guidage à partir de la solution initiale, il est nécessaire de pouvoir engendrer la totalité (ou tout au moins une partie) des voisins permettant de s'approcher de la solution de guidage. L'itération de ce mécanisme de rapprochement permet à terme de construire un, plusieurs, ou tous les chemins reliant les deux solutions. Afin de garantir l'atteignabilité des solutions entre elles, il est nécessaire de définir un opérateur de distance correspondant à la notion de voisinage définie. Si S est la solution courante du chemin en construction et G la solution de guidage, les solutions S' faisant potentiellement partie d'un chemin devront vérifier la relation $d(S', G) < d(S, G)$ ($d(x, y)$ étant la distance entre x et y). Ainsi, si l'ensemble des valeurs prises par d est fini, la taille des chemins engendrés sera finie.

L'algorithme AGA utilise un mécanisme de sélection et d'évaluation de plusieurs opérateurs de voisinage. Les résultats expérimentaux obtenus ont montré que l'opérateur *shift* est le plus performant pour les instances bi-objectif testées (comme pour l'exemple du problème $ta\_100\_5\_01$ de la figure 4.23). Nous avons donc choisi d'utiliser cet opérateur de voisinage pour définir les chemins. La méthode de "raccordement" des solutions fait en partie l'objet de la section 6.2.2.

Dans le cadre d'une hybridation d'un algorithme de $MOPR$ avec AGA, on peut éventuellement faire le choix de l'opérateur de voisinage en fonction des résultats obtenus par l'ensemble de ces opérateurs durant le déroulement de l'AG, en rendant donc l'algorithme global adaptatif. Cependant il est nécessaire de définir des notions de distance pour chaque opérateur pouvant potentiellement être sélectionné.

6.2.2 Distance dans l'espace décisionnel

La notion de distance entre deux solutions S_1 et S_2 suivant un opérateur de voisinage $\mathcal{V}$ correspond au nombre minimal d'applications de $\mathcal{V}$ pour transformer S_1 en S_2. Le principe du PR consiste généralement à engendrer les chemins minimaux reliant S_1 à S_2.

Dans le cadre d'un espace de permutations comme pour le problème du Flow-shop, Reeves et Yamada [RY98] remarquent qu'il est trivial de calculer le nombre minimal d'échanges nécessaires pour relier deux solutions, ce qui correspond à la distance liée à l'opérateur *Exchange*. En effet, pour effectuer ce calcul, il suffit de parcourir la permutation en réalisant des échanges là où les jobs sont mal placés. Reeves et Yamada précisent que ce calcul n'est pas aisé pour l'opérateur *Shift*. Ils font également la remarque que l'opérateur *Shift* est généralement plus performant que l'opérateur *Exchange*. Dans leur étude, ils utilisent donc plusieurs mesures de distance indépendantes de l'opérateur de voisinage *Shift*.

Nous proposons ici une mesure de distance correspondant exactement au nombre minimal d'applications de l'opérateur *Shift* nécessaire pour relier deux solutions.

- Soit $s_{max}(S_1, S_2)$ la plus longue séquence de jobs commune à S_1 et S_2.
- Soit $d_{perm}(S_1, S_2)$ la distance génotypique entre les deux solutions S_1 et S_2.

$d_{perm}(S_1, S_2)$ est définie par :

$$d_{perm}(S_1, S_2) = N - s_{max} \tag{6.1}$$

Un exemple de calcul de distance entre deux solutions S_1 et S_2, obtenue grâce au calcul de leur plus longue sous-chaîne commune est détaillé dans la figure 6.2. Dans cet exemple, trois applications de l'opérateur $Shift$ sont nécessaires pour relier S_1 à S_2 - les jobs 2, 4 et 6. Ceux-ci correspondent aux jobs qui ne font pas partie de la plus longue sous-chaîne commune à S_1 et S_2.

Montrons que $d_{perm}(S_1, S_2)$ est bien une notion de distance, qui correspond au nombre minimal d'applications de l'opérateur *Shift* nécessaire pour relier S_1 à S_2.

Voici quelques notations et rappels de notations :

- Dans cette preuve, on parlera indifféremment de *job* ou de *lettre* d'une permutation. La notion de distance définie ici est valable quel que soit le problème de permutation traité. De même, on utilisera indifféremment les termes de chaîne de caractères ou de permutation de jobs.
- $Shift(i, j, X)$ correspond à la permutation engendrée en déplacant le $i^{ème}$ de la permutation X à une nouvelle position j.

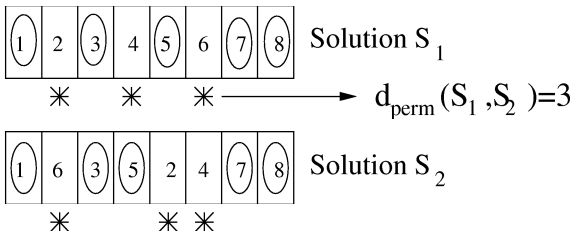


FIG. 6.2 – Calcul de distance entre deux solutions S_1 et S_2. Les jobs marqués d'une '*' sont les jobs à déplacer pour relier S_2 à S_1.

- $pos_X(i)$ est la position de l'élément i dans la permutation X.
- $LSCC(X,Y)$ est la fonction calculant la plus longue sous chaîne commune à X et Y (les lettres communes n'étant pas forcément consécutives).
- $supp(X,i)$ est la fonction qui supprime la $i^{\text{ème}}$ lettre de la chaîne X.
- $add(X,j,y)$ est la fonction qui ajoute la lettre y devant la $j^{\text{ème}}$ lettre de la chaîne X.

Soient X et Y deux chaînes de caractères. Afin de montrer que $d_{perm}(X,Y)$ est bien une distance, commençons par montrer trois lemmes :

Suppression d'une lettre dans une chaîne Y :

Lemme 1 *Pour toutes chaînes X, Y et $\forall i \in [1..|Y|]$, on a :* $\|LSCC(X,Y)\| \geq \|LSCC(X,supp(Y,i))\| \geq \|LSCC(X,Y)-1\|$.

Posons $Y = \mathcal{V}x\mathcal{W}$ avec $\mathcal{V},\mathcal{W}$ sous-chaînes de Y et x une lettre de Y. Soit le nouveau mot formé (après suppression d'une lettre) $Y' = \mathcal{V}\mathcal{W}$.

La longueur de la plus longue sous-chaîne commune à X et Y ne peut augmenter en supprimant un élément de Y. Deux cas sont envisageables :

- Si x appartient à toutes les plus longues sous-chaînes communes à X et Y, dans ce cas $LSCC(X,Y') \geq LSCC(X,Y)-1$ (on a en effet une sous-chaîne commune égale à $LSCC(X,Y)$ moins la lettre x) .
- sinon, on a $\|LSCC(X,Y')\| = \|LSCC(X,Y)\|$.

Dans les deux cas, le lemme 1 est vérifié.

Ajout d'une lettre dans une chaîne Y :

Lemme 2 *Pour toutes chaînes X, Y, $\forall j \in [1..|Y|]$ et $\forall y$, on a $LSCC(X,Y)+1 \geq LSCC(X,add(Y,j,y)) \geq LSCC(X,Y)$.*

La longueur de la plus longue sous-chaîne commune à X et Y ne peut diminuer en ajoutant un élément dans Y. De même que pour le lemme 1, deux cas sont envisageables :

- soit l'élément ajouté peut être inséré dans au moins une des $LSCC$. Dans ce cas, on a $\|LSCC(X,Y'')\| = \|LSCC(X,Y)\|+1$.
- soit $\|LSCC(X,Y'')\| = LSCC\|(X,Y)\|$.

Le lemme 2 est donc vérifié.

Application de l'opérateur d'insertion à une permutation Y :

Lemme 3 *Pour toute permutation X et Y, on a :*

$$d_{perm}(X,Y) - 1 \leq d_{perm}(X, Shift(i,j,Y)) \leq d_{perm}(X,Y) + 1$$

L'opérateur d'insertion, appliqué aux chaînes, consiste à supprimer une lettre d'une chaîne pour la ré-introduire à un autre endroit. Donc, soient deux permutations X et Y. On pose $Shift(i,j,Y) = add(supp(Y,i), y, Y_i)$, et :

$$Y' = supp(Y,i) \tag{6.2}$$

$$Y'' = add(Y', j, Y_i) \tag{6.3}$$

On a donc :

$$Y'' = Shift(i,j,Y) \tag{6.4}$$

Mais d'après les lemmes 1 et 2 on a également :

$$LSCC(X,Y) \geq LSCC(X,Y') \geq LSCC(X,Y) - 1 \tag{6.5}$$

$$LSCC(X,Y') + 1 \geq LSCC(X,Y'') \geq LSCC(X,Y') \tag{6.6}$$

D'après (6.4) (6.5) (6.6), on a donc $\|LSCC(X,Y)-1\| \leq \|LSCC(X,Shift(i,j,Y))\| \leq \|LSCC(X,Y)+1\|$. En appliquant cette relation au calcul de d_{perm}, on a donc bien le lemme 3 vérifié.

On peut "s'approcher d'une solution" :

Prouvons maintenant qu'il existe toujours un moyen de "s'approcher" d'une solution :

Lemme 4 *Pour toute permutation X et Y de taille N, avec $d_{perm}(X,Y) \neq 0$ on a :*

$$\exists\ i,j \in [0..N] \quad \| \quad d_{perm}(Shift(i,j,X),Y) = d_{perm}(X,Y) - 1$$

Preuve : Soient deux permutations X et Y. Si $X \neq Y$, alors $\exists x \in Y$ tel que $x \notin LSCC(X,Y)$. Notons y (resp. z) le plus proche prédécesseur (resp. successeur) de x dans X tel que $y \in LSCC(X,Y)$ (resp. $z \in LSCC(X,Y)$).

Posons $Y' = Shift(x, pos_Y(y)+1, Y)$.

Soit $U = LSCC(X, Y)$. U peut s'écrire U_1yzU_2. Soit $U' = LSCC(X, Y')$. On a $U' = U_1yxzU_2$ (car x se trouve maintenant entre y et z aussi bien pour X que pour Y). On a bien $|U'| = |U| + 1$.

Donc quelque soient X, Y tels que $X \neq Y$, on peut trouver une mutation par insertion permettant d'augmenter la taille de la plus longue sous chaîne commune aux deux individus. Appliqué à d_{perm}, on a donc le lemme 4 vérifié.

Remarquons que l'on peut choisir de placer l'élément x indifféremment dans l'intervalle $[pos_Y(y)+1, pos_Y(z)])$. Les positions $[pos_Y(y)+1, pos_Y(z)]$ correspondent à l'ensemble des possibilités de placement pour la lettre x qui permettent de s'approcher de la solution Y. La figure 6.3 montre un exemple de l'ensemble des places possibles. Ainsi, en itérant ceci pour tous les jobs n'appartenant pas à $LSCC(X, Y)$, on obtient l'ensemble total des voisins de X permettant de s'approcher de Y comme le montre la figure 6.6 (en ne considérant qu'une seule plus longue sous-chaîne commune, bien entendu).

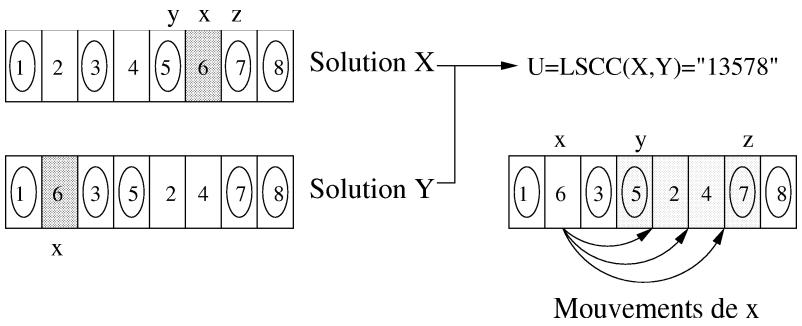


FIG. 6.3 – [Recherche des positions acceptables pour placer un job. Le job x ne fait pas partie de $LSCC(X, Y)$. On cherche y et z dans X, les places disponibles pour x se trouvent alors dans l'intervalle $[pos_Y(y) + 1, pos_Y(z)]$.

À l'aide des lemmes 3 et 4, prouvons que d_{perm} est bien une notion de distance. Nous avons trois propriétés à établir :

- **LSCC est symétrique :** $(d_{perm}(X, Y) = d_{perm}(Y, X))$. Ceci est vérifié de façon triviale par la définition même de la plus longue sous-chaîne commune.
- **Séparation de l'espace par** d_{perm} : $(d_{perm}(X, Y) = 0 \leftrightarrow X = Y)$. Ceci est également trivial grâce à la définition de d_{perm}. En effet, $d_{perm}(X, Y) = 0 \leftrightarrow |LSCC(X, Y)| = N$, et pour cela, on a forcément $LSCC(X, Y) = X = Y$.
- **Inégalité triangulaire :** : $(\forall\{X, Y, Z\}, d_{perm}(X, Z) \leq d_{perm}(X, Y) + d_{perm}(Y, Z))$.

Nous allons démontrer par récurrence cette propriété. Le principe de la preuve est schématisé dans la figure 6.4. On prouve que pour une permutation X quelconque, l'inégalité triangulaire est vérifiée pour toute permutation à une distance donnée n. Le but étant de propager l'inégalité triangulaire aux permutations à une distance $n + 1$, jusqu'à recouvrir tout l'espace de recherche.

Démonstration par récurrence :

Soient X et Z deux permutations. Soit $\mathcal{P}_n$ la proposition $\mathcal{P}_n$ suivante : "$\forall Y$ tel que

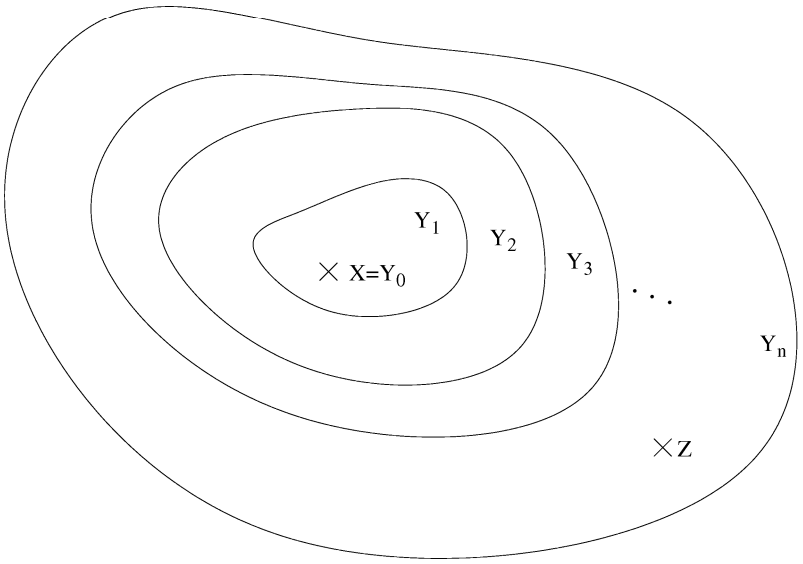


FIG. 6.4 – Principe de la preuve de l'inégalité triangulaire. Les solutions de l'ensemble Y_i se situent à une distance i de la solution initiale X.

$d_{perm}(X,Y) \leq n$, alors on a l'inégalité suivante : $d_{perm}(X,Z) \leq d_{perm}(X,Y)+d_{perm}(Y,Z)$"

D'après la propriété de séparation de d_{perm}, $\mathcal{P}_0$ est vrai. Supposons $\mathcal{P}_n$ vrai.

Soit Y tel que $d_{perm}(X,Y) \leq n+1$.

Si $d_{perm}(X,Y) \leq n$, alors d'après l'hypothèse de récurrence, l'inégalité est vérifiée. Le cas restant est celui où $d_{perm}(X,Y) = n+1$. On a montré précédemment qu'il existait au moins une mutation $Shift(a,b,Y)$ permettant de diminuer la distance entre deux solutions X et Y. Soit $Y' = perm(a,b,Y)$. On a montré que $d_{perm}(X,Y') = n$. Donc d'après l'hypothèse de récurrence, on a $d_{perm}(X,Z) \leq d_{perm}(x,y') + d_{perm}(y',z)$. Donc, avec les propriétés suivantes, provenant des lemmes 3 et 4 et de l'hypothèse de récurrence :

$$d_{perm}(X,Y) = d_{perm}(X,Y') + 1$$
$$d_{perm}(X,Z) \leq d_{perm}(X,Y') + d_{perm}(Y',Z)$$
$$d_{perm}(Y,Z) \geq d_{perm}(Y',Z) - 1$$

On en déduit :

$$d_{perm}(X,Y) + d_{perm}(Y,Z) \geq d_{perm}(X,Y') + 1 + d_{perm}(Y',Z) - 1$$
$$d_{perm}(X,Y) + d_{perm}(Y,Z) \geq d_{perm}(X,Z)$$

L'hypothèse de récurrence est donc bien vérifiée au rang $n+1$. Donc d_{perm} vérifie bien l'inégalité triangulaire. Par conséquent, l'équation 6.1 est bien la notion de distance correspondant à l'opérateur de voisinage $Shift$.

Le calcul de la plus longue sous-chaîne commune à deux chaînes de caractères est un problème bien connu de la communauté de la génomique. Ce problème est une application classique de la programmation dynamique [CLR90]. Nous avons implémenté cet algorithme afin de calculer, dans l'espace décisionnel, la distance entre deux solutions par rapport à l'opérateur de voisinage *Shift*. La complexité de l'algorithme de calcul de la plus longue sous-chaîne commune est en $\mathcal{O}(N^2)$.

Ce calcul pour les deux solutions S_1 et S_2 de la figure 6.2 se trouve dans le tableau 6.1. Le calcul de $s_{max}(S_1, S_2)$ est initialisé à 0 dans la partie supérieure gauche du tableau t, puis en incrémentant successivement les deux indices i et j du tableau, on incrémente la valeur de la plus longue sous-chaîne commune "courante" (c'est-à-dire $max(t[i-1][j], t[i][j-1])$) lorsque $S_1[i] = S_2[j]$. Il est assez facile de trouver une chaîne commune de taille maximale. Pour retrouver l'ensemble des plus longues sous-chaînes communes à S_1 et S_2 on parcourt les cases du tableau correspondant à des valeurs communes de S_1 et S_2 (en gras sur l'exemple), en forçant les indices à augmenter tout le temps. Ainsi on obtient toutes les plus longues sous-chaînes communes à S_1 et S_2, dans notre exemple les chaînes $'13578'$, '13478' et '12478'.

Pour notre application au $MOPR$, on se contentera d'explorer les chemins résultant d'une valeur possible pour la plus longue sous-chaîne commune. Les chemins engendrés seront tous minimaux. On peut engendrer la totalité des chemins minimaux en tenant compte de toutes les plus longues sous-chaînes communes possibles, et de tous les chemins possibles pour placer les jobs mal placés.

TAB. 6.1 – Calcul des plus longues sous-chaînes communes aux solutions S_1 et S_2 de la figure 6.2 par programmation dynamique. Les cases en gras indiquent une valeur commune à S_1 et S_2.

$s_{max}(S_1, S_2)$	S_2	1	2	3	4	5	6	7	8
S_1	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*
1	*0*	**1**	1	1	1	1	1	1	1
6	*0*	1	1	1	1	1	**2**	2	2
3	*0*	1	1	**2**	2	2	2	2	2
5	*0*	1	1	2	2	**3**	3	3	3
2	*0*	1	**2**	2	2	3	3	3	3
4	*0*	1	2	2	**3**	3	3	3	3
7	*0*	1	2	2	3	3	3	**4**	4
8	*0*	1	2	2	3	3	3	4	**5**

6.2.3 Solutions initiales du $MOPR$

En optimisation mono-objectif on choisit en général de "bonnes" solutions pour réaliser l'algorithme de PR, en terme de convergence ou de diversité. Mais on doit définir également quelle similarité on désire avoir entre les deux solutions sélectionnées (similarité au

niveau de l'espace décisionnel). On peut encourager la sélection de solutions distantes entre elles afin de favoriser l'exploration de l'espace de recherche, ou encourager la sélection de solutions proches pour intensifier la recherche autour des bonnes solutions.

Dans le cadre multi-objectif, les mêmes questions se posent. Il semble toujours nécessaire de sélectionner de bonnes solutions, éventuellement non-dominées durant toute la recherche. Pour la définition du degré de similarité à avoir entre les solutions sélectionnées, plusieurs possibilités s'offrent aux concepteurs de l'algorithme. Comme en optimisation mono-objectif, on peut définir le choix des couples de solutions en fonction de la distance les séparant dans l'espace génotypique. On peut également choisir de favoriser la sélection de solutions éloignées ou proches dans l'espace des objectifs tout en ne sélectionnant que de bonnes solutions en terme de convergence. Faut-il choisir deux solutions de la même zone du front de Pareto afin d'espérer améliorer les solutions non-dominées de cette zone, ou faut-il au contraire choisir des solutions opposées l'une à l'autre, afin de combiner les qualités de chacune d'elles pour trouver de bonnes solutions de compromis (voir figure 6.5) ?

Dans notre cas, on sélectionnera comme solutions initiales du $MOPR$ uniquement les solutions non-dominées durant la recherche AGA, c'est-à-dire celles appartenant à l'archive Pareto. Parmi ces solutions Pareto fournies par AGA, le choix se fera aléatoirement. Ainsi, les deux types d'exploration schématisés dans la figure 6.5 seront en partie réalisés. Après chaque génération de $MOPR$, on remet bien entendu à jour l'archive Pareto afin de prendre en compte les derniers progrès réalisés par l'algorithme.

6.2.4 Génération du voisinage

D'après la section 6.2.2, nous utilisons le voisinage engendré par l'opérateur $Shift$ pour générer les solutions permettant de construire les chemins reliant les solutions initiales aux solutions de guidage. Le but étant de trouver des chemins reliant ces solutions, il est nécessaire de ne générer que les voisins qui permettent de réduire la distance restant à parcourir pour atteindre la solution de guidage.

La première étape pour réduire cette distance, consiste à définir un ensemble de jobs "bien placés", en calculant une plus longue sous-chaîne commune aux deux solutions envisagées. Puis, durant l'algorithme, il convient de déplacer un à un les jobs "mal placés" dans le but d'augmenter à chaque mouvement la taille de la plus longue sous-chaîne commune. Il faut donc calculer l'ensemble des positions possibles pour les jobs "mal placés", qui augmentent la taille de cette plus longue sous-chaîne commune, comme décrit dans la preuve de la section 6.2.2. La figure 6.6 montre les voisins à engendrer pour l'exemple de la figure 6.2.

6.2.5 Chemins engendrés

Nous pouvons maintenant trouver l'ensemble des mouvements à réaliser sur S_1 pour se rapprocher de la solution de guidage S_2. Il est nécessaire de définir maintenant quels sont les chemins à explorer parmi les chemins envisageables.

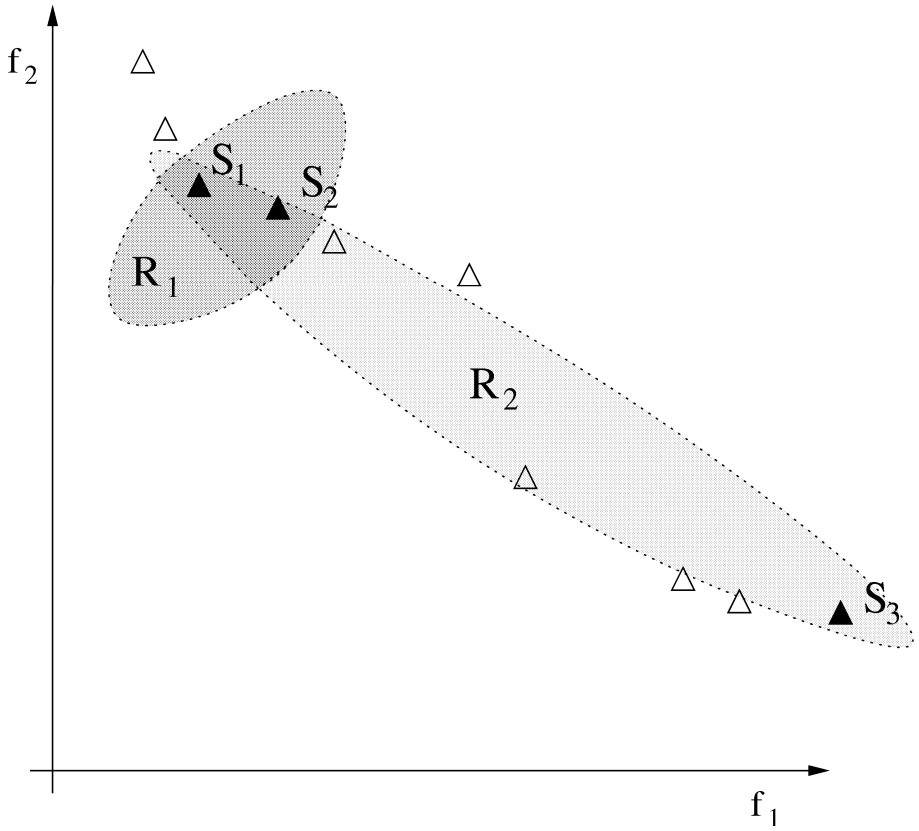


FIG. 6.5 – Allure de l'espace des solutions explorées par $MOPR$ en fonction des solutions initiales choisies (cas bi-objectif). Le choix des solutions S_1 et S_2, proches dans l'espace décisionnel, favorise l'intensification sur une région précise de l'espace de recherche (R_1). Le choix des solutions S_1 et S_3 éloignées dans l'espace décisionnel favorise la découverte de solutions de compromis entre les deux solutions choisies (R_2).

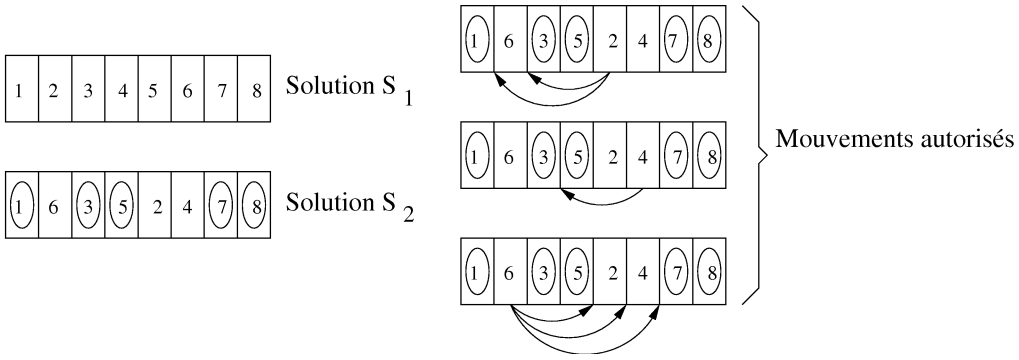


FIG. 6.6 – Voisins envisagés durant $MOPR$. La plus longue sous-chaîne commune est représentée par les jobs encerclés. Les mouvements possibles à appliquer à S_2 pour s'approcher de S_1 sont explorés pour les jobs "mal placés" 2, 4 et 6. Il y a en tout 6 mouvements possibles permettant de se rapprocher de S_1.

Une première approche possible est d'engendrer tous les chemins possibles. Lorsqu'elle est applicable, cette méthode est idéale puisqu'elle permet d'obtenir toutes les bonnes solutions se trouvant entre les solutions initiale et de guidage. Mais, bien évidemment, le nombre de chemins à explorer croît exponentiellement selon la distance initiale calculée entre les deux solutions considérées. L'exploration systématique est donc envisageable uniquement lorsque la sélection des solutions initiales favorise la similarité au niveau génotypique (dans ce cas, une petite distance sépare la solution initiale de celle de guidage). Dans les autres cas, il est nécessaire d'explorer seulement une partie des chemins possibles afin de rester dans des temps de calculs raisonnables.

Une stratégie diamétralement opposée à la précédente consiste à sélectionner uniquement le meilleur mouvement possible à chaque pas de l'algorithme, pour ne définir au final qu'un seul chemin. Beaucoup d'études sur le *path relinking* mono-objectif proposent ce type d'approche. Les autres proposent des approches intermédiaires.

Dans notre cas, l'exploration de tous les chemins possibles est peu envisageable. Les tests effectués ont montré que le nombre de solutions à explorer est trop grand dès que la distance de départ entre la solution initiale et la solution de guidage dépasse 20, ce qui est très fréquent pour les instances de 50 jobs ou plus. Nous avons donc choisi de ne définir qu'un seul chemin par couple de solutions du $MOPR$, mais des travaux futurs pourront explorer des voies intermédiaires utilisant par exemple la notion de Pareto optimalité pour sélectionner l'ensemble des chemins à explorer.

La figure 6.7 schématise la méthode utilisée pour la sélection du chemin à parcourir. On engendre l'ensemble des voisins V de la solution initiale S_2 tels que $d_{perm}(S_1, V_i) < d_{perm}(S_1, S_2)$ (S_1 étant la solution de guidage). On sélectionne ensuite une solution parmi l'ensemble V généré pour itérer le mécanisme. Pour choisir cette solution, on réalise une agrégation aléatoire sur les différents objectifs, et le choix se porte alors sur la solution ayant la meilleure *fitness*.

6.3 $MOPR$ coopératif

Le principe de la coopération est le même que pour l'algorithme $AGA+AMA$. On exécute AGA jusqu'à ce qu'une condition d'arrêt prédéfinie soit vérifiée. Puis, on sélectionne les solutions initiales de PLS à partir des solutions Pareto obtenues par une génération de chemin de $MOPR$. L'algorithme 6.1 détaille les différentes étapes de $MOPR$.

La coopération entre $MOPR$ et PLS est de type **HTH(MOPR+PLS)**. Lors de la recherche de chemin durant l'algorithme de $MOPR$, les solutions non-dominées durant la recherche sont archivées. Une fois l'algorithme de $MOPR$ fini, une recherche PLS est lancée, avec cette archive comme population initiale (voir Fig. 6.8).

L'algorithme global, $AGA+MOPR$, est de type **HTH(AGA+HTH(MOPR+PLS))**. Nous nous sommes contentés d'une implémentation "classique", où AGA fournit une population de départ au mécanisme d'intensification qui est proposé dans cette section (voir l'algorithme 6.2). De cette manière, la stratégie de coopération entre AGA et $MOPR$ est similaire à celle réalisée pour l'algorithme coopératif $AGA + AMA$.

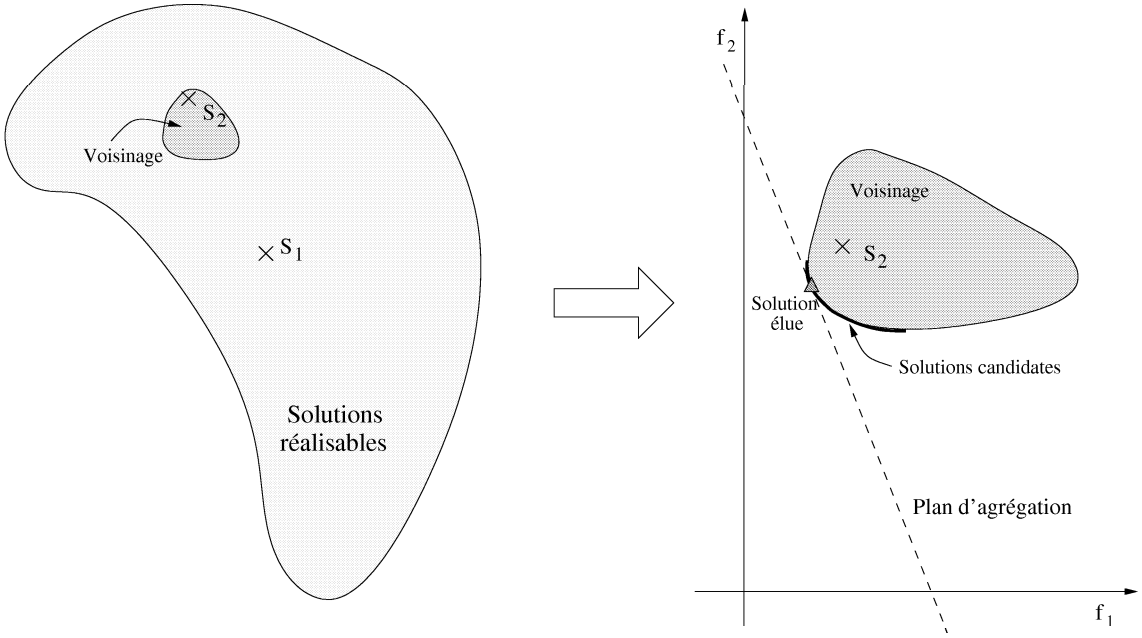


FIG. 6.7 – Exploration du voisinage dans l'algorithme $MOPR$: on engendre l'ensemble des solutions voisines de S_2 permettant de se rapprocher de S_1. Les solutions non-dominées de cet ensemble sont candidates à la sélection pour définir un chemin. Le choix de la solution se fait alors par une agrégation aléatoire des objectifs.

Algorithme 6.1 $MOPR$

Création d'une population initiale de solutions non-dominées PO.
Choix aléatoire de deux solutions S_1 et S_2 de PO.
Calculer $d(S_1, S_2)$ la distance génotypique entre les solutions S_1 et S_2.
$PO' \leftarrow \emptyset$.
tant que $S_1 \neq S_2$ **faire**
  Générer les i voisins S_{i2} de S_2 vérifiant $\forall k \in [0..i[,\ d(S_1, S_{k2}) < d(S_1, S_2)$.
  Choisir aléatoirement des poids pour les différents objectifs à optimiser.
  Évaluer les solutions S_{i2} selon les poids définis.
  Soit S_* la meilleure solution de l'ensemble des S_{i2}.
  $S_2 \leftarrow S_*$.
  $PO' \leftarrow$ solutions non-dominées de $PO' \cup S_{i2}$.
fin tant que
Retourner PO'.

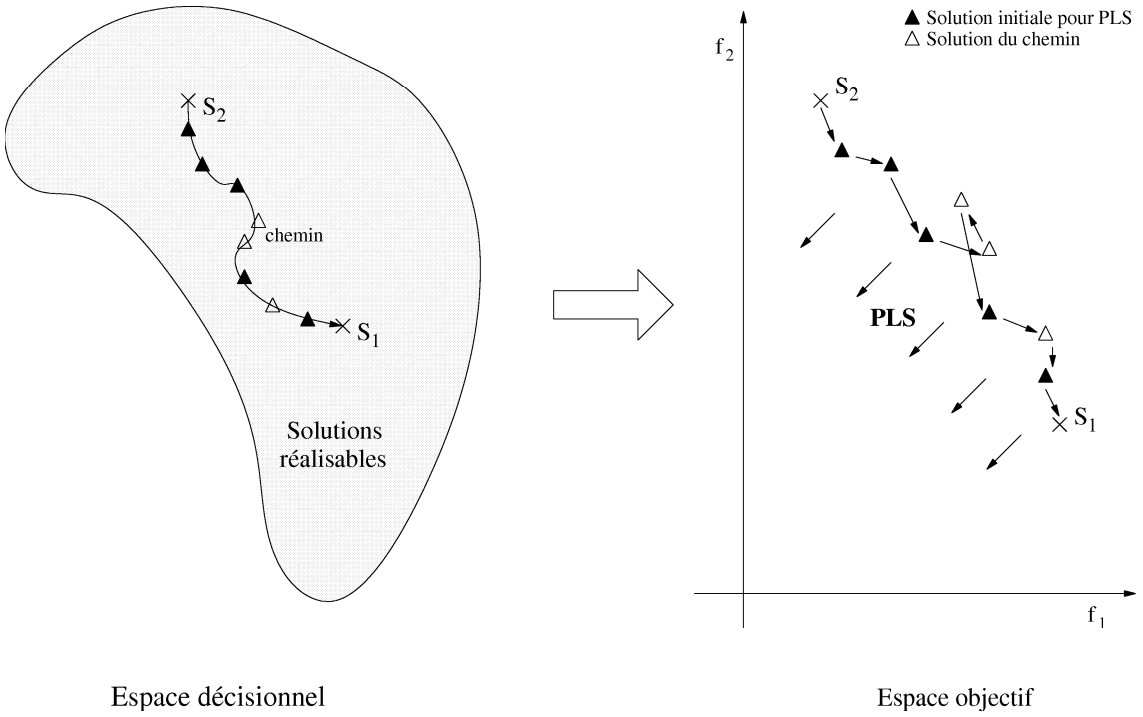


FIG. 6.8 – Coopération entre $MOPR$ et PLS : les solutions non-dominées du chemin défini lors de $MOPR$ servent de solutions initiales pour PLS.

Algorithme 6.2 $AGA + MOPR$

Création d'une population initiale aléatoire P_1.
tant que le temps d'exécution n'est pas atteint **faire**
  Exécuter AGA, initialisé avec P_1 (Algorithme 4.2).
fin tant que
$P_2 \leftarrow$ Archive Pareto de AGA.
tant que le temps d'exécution n'est pas atteint **faire**
  Exécuter $MOPR$, initialisé avec P_2 (Algorithme 6.1).
  $P_3 \leftarrow$ Archive Pareto de $MOPR$.
  Exécuter PLS, initialisé avec P_3 (Algorithme 5.1).
  Mettre à jour P_2.
fin tant que

6.4 Expérimentations

Nous avons réalisé les mêmes expérimentations que dans les sections 4.5.2 et 5.2.4. Nous avons évalué en premier lieu l'apport de l'hybridation par $MOPR$ sur les résultats de AGA. Pour évaluer cet apport, nous avons comparé sur des exécutions de 600 minutes les résultats de AGA seul, de $MOPR + PLS$ seul, avec les approches coopératives entre AGA et $MOPR + PLS$ consacrant une proportion de temps variable à $MOPR + PLS$ (10% ou 90%).

Nous avons réalisé quelques tests sur ces différentes approches. Des exemples de résultats obtenus sont sur les figures 6.9 et 6.10. Nos expérimentations ont montré qu'il était plus intéressant de consacrer une large part de temps à l'exécution de $MOPR + PLS$. Mais comme le montrent les figures 6.9 et 6.10, AGA permet toujours d'apporter un plus par rapport à $MOPR + PLS$ seul.

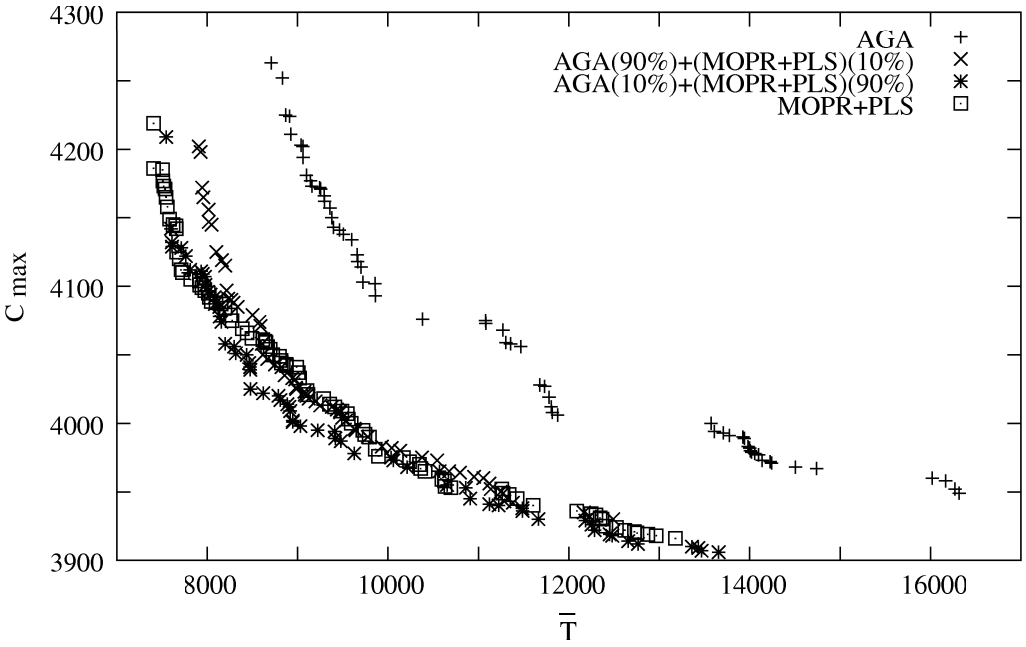


FIG. 6.9 – .
Résultat obtenu sur l'instance *ta_50_20_01* par AGA, $MOPR + PLS$ et $AGA + MOPR$.

Les résultats obtenus, pour les métriques S et $Contribution$ sur les différentes instances, sont répertoriés dans les tables 6.2, 6.3 et 6.4. Ces tables correspondent respectivement aux calculs d'aire de dominance pour AGA, d'aire de dominance pour $AGA + MOPR$, et de $Contribution$ de $AGA + MOPR$ par rapport à AGA.

Pour l'instance *ta_100_5_01*, les progrès réalisés sont très faibles. Cependant, pour cette instance, nous obtenons toujours la borne minimale pour le C_{max}, qui est la valeur optimale prouvée pour cet objectif. De plus, les ensembles Pareto que nous obtenons sur les différentes exécutions sont très semblables entre eux, ce qui nous pousse à penser que

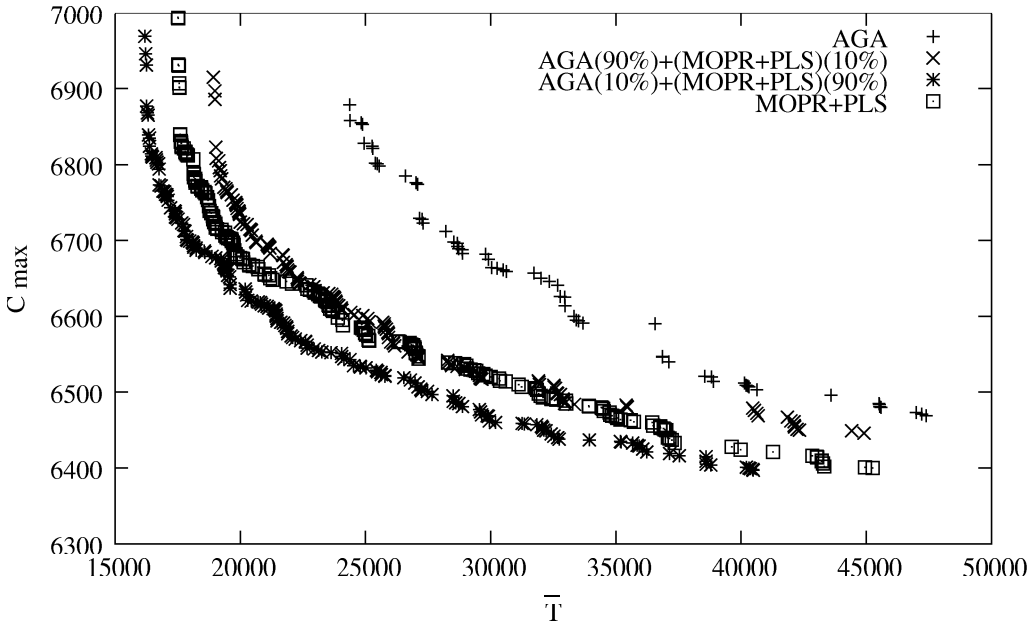


FIG. 6.10 – .
Résultat obtenu sur l'instance *ta_100_20_01* par AGA, $MOPR+PLS$ et $AGA+MOPR$.

TAB. 6.2 – $S(AGA)$.

Instance	S(AGA)			
	S_{Min}	S_{Max}	Moyenne	Déviation
ta_50_10_01	516521	828610	709023.6	93279.2
ta_50_20_01	1841592	2476210	2170418.6	205943.7
ta_100_5_01	431760	446264	439799.2	5526.3
ta_100_10_01	5414203	6956336	6218845.8	464814.8
ta_100_20_01	17167424	20139584	19117894.4	866011.0

les résultats obtenus sont très proches du front Pareto optimal. Pour les autres instances, l'apport de $MOPR$ est considérable. L'apport, en terme d'aire de dominance, varie en effet entre 27.7% et 45.2%. Le tableau 6.4 montre que, pour ces mêmes instances, l'approche hybride domine en totalité AGA pour toutes les exécutions réalisées.

6.5 Conclusion

Dans ce chapitre, nous avons proposé une coopération originale entre AGA et un algorithme de *path relinking* multi-objectif $MOPR$. Nous avons utilisé la notion de dominance Pareto dans le mécanisme de sélection de solutions de $MOPR$.

Dans le cadre de la conception de cet algorithme, nous avons également proposé un

TAB. 6.3 – $S(AGA + MOPR)$

Instance	S(AGA + MOPR)				
	S_{Min}	S_{Max}	Moyenne	Déviation	Apport
$ta\_50\_10\_01$	870448	1065523	988079.6	67443.2	39.4%
$ta\_50\_20\_01$	2853166	3113992	3020664.0	93279.6	39.2%
$ta\_100\_5\_01$	445695	446878	446323.0	286.2	1.5%
$ta\_100\_10\_01$	7813940	8148625	7978420.0	93335.8	28.3%
$ta\_100\_20\_01$	26209816	28745238	27752835.2	690416.0	45.2%

TAB. 6.4 – $Contribution(AGA + MOPR/AGA)$

Instance	Cont(AGA + MOPR/AGA)			
	$Cont_{Min}$	$Cont_{Max}$	Moyenne	Déviation
$ta\_50\_10\_01$	1.00	1.00	1.000	0.000
$ta\_50\_20\_01$	1.00	1.00	1.000	0.000
$ta\_100\_5\_01$	0.71	1.00	0.836	0.085
$ta\_100\_10\_01$	1.00	1.00	1.000	0.000
$ta\_100\_20\_01$	1.00	1.00	1.000	0.000

opérateur de mesure de distance entre solutions, correspondant au voisinage de l'opérateur de mutation *Shift*. Grâce à cet opérateur, les solutions intermédiaires entre la solution initiale et la solution de guidage sont construites directement sans réévaluer la distance.

Nous avons aussi mis au point des méthodes multi-objectif définissant les différentes fonctionalités d'un algorithme de *path relinking*. Puis nous avons effectué des expérimentations qui ont montré l'intérêt de ce type d'approche. Cependant, on peut imaginer d'utiliser par la suite des mécanismes plus évolués pour établir un algorithme de *path relinking* multi-objectif encore plus performant, ou encore s'inspirer du mécanisme de coopération établi dans $AGMA$ afin de rendre plus efficace la coopération entre AGA et $MOPR$.

Chapitre 7

Coopération méta/exacte

Dans ce chapitre, nous présentons trois méthodes coopératives méta/exacte. La première est une approche de résolution exacte, les deux autres sont heuristiques, l'une explorant des voisinages larges, l'autre s'appuyant sur le partitionnement des solutions. Nous montrons que les deux types de coopérations permettent d'améliorer soit le temps de résolution pour l'approche exacte, soit la qualité des approximations pour les approches heuristiques. Ce travail a fait l'objet d'un chapitre de l'ouvrage "Real-world Multi-objective System Engineering" [BST05d], d'une publication à la conférence WEA'04 [BLDT04], et a été soumis pour publication dans la revue SIAM Journal on Optimization [BST05e].

7.1 Introduction

D'une manière générale, les approches de résolution exactes des problèmes d'optimisation difficiles sont réservées aux petits problèmes, les approches heuristiques étant dédiées aux problèmes de plus grande taille. Cependant, les approches heuristiques peuvent résoudre de manière optimale les petites instances, avec parfois un gain de temps très significatif, comme nous l'a montré $AGMA$ appliqué au Flow-shop bi-objectif. Les approches heuristiques peuvent également fournir des informations précieuses à la méthode exacte, informations qui permettront de guider l'exploration de l'espace des solutions réalisables. Les coopérations avec approches de résolution exactes, quant à elles, ne peuvent être appliquées aux grandes instances de problèmes. Cependant, elle peuvent permettre à une méthode heuristique d'affiner la recherche, par le biais d'une coopération entre les deux types d'approches.

Dans ce chapitre, nous proposons d'explorer les deux approches de résolutions, exactes et heuristiques. Nous utilisons pour cela les métaheuristiques coopératives présentées précédemment et une méthode de résolution exacte appelée *méthode à deux phases* [UT95].

En premier lieu, nous présentons donc dans la section 7.2 une approche de résolution exacte, dédiée au cas bi-objectif. Appliquée au Flow-shop, elle permet de confirmer les résultats obtenus par les coopérations présentées dans le chapitre précédent. Puis, dans la section 7.3 nous réalisons différentes coopérations méta-exacte. Une approche coopérative exacte est présentée, dans laquelle la métaheuristique permet d'accélérer l'énumération de l'ensemble de l'espace de recherche. Puis, deux coopérations méta/exacte approchées sont présentées, la méthode à deux phases étant utilisée en mécanisme de post-optimisation. Nous discuterons ensuite des résultats obtenus par les différentes approches avant de conclure dans la section 7.4 sur l'intérêt de telles coopérations.

7.2 Méthode exacte bi-objectif appliquée au flow-shop bi-objectif

Les méthodes de résolution exactes pour les problèmes multi-objectif ne sont pas très nombreuses. La plus couramment utilisée consiste à agréger les objectifs de différentes manières, et de résoudre de manière exacte chaque agrégation [AN79, Hen85, HR94]. Mais cette méthode ne permet de trouver que les solutions Pareto optimales supportées. D'autres méthodes ont été expérimentées, mais elles se basent généralement sur différentes manières de combiner les différents objectifs en un seul (voir section 3.2.2), comme par exemple pour la méthode ϵ-contrainte [MS95]. La méthode 'Pareto', consistant à maintenir un ensemble entier de solutions non-dominées servant de bornes, semble trop lourde pour résoudre les problèmes multi-objectif de manière exacte.

Une méthode appelée méthode à deux phases (*Two Phases Method* - TPM), a été proposée en 1995 par Ulungu et Teghem pour la résolution exacte de problèmes d'optimisation bi-objectif [UT95]. La méthode à deux phases est une combinaison de l'approche par agrégation et de l'approche 'Pareto'. Cette méthode était dédiée en premier lieu à la résolution d'un problème de sac à dos bi-objectif, mais le concept très général permet de

pouvoir l'adapter sous certaines conditions à de nombreux problèmes.

Nous décrivons dans cette section l'application de cette méthode au problème de Flow-shop bi-objectif, réalisée dans le cadre du travail de thèse de Lemesre [LDT05].

7.2.1 Description de la méthode à deux phases

La recherche effectuée par TPM consiste en deux phases distinctes. La première phase est destinée à la découverte des solutions optimales supportées, la deuxième phase cherche les solutions non-supportées en restreignant l'espace de recherche à l'aide des solutions trouvées durant la première phase.

7.2.1.1 La première phase

La première phase consiste à trouver l'ensemble des solutions supportées en réalisant des recherches récursives sur des agrégations des deux objectifs C_1 et C_2 de la forme $\lambda_1 C_1 + \lambda_2 C_2$. Dans un premier temps on recherche les deux extrémités de l'ensemble Pareto optimal (qui sont forcément des solutions supportées) en réalisant une recherche avec ordre lexicographique des objectifs ($lex(C_1, C_2)$, puis $lex(C_2, C_1)$).

Puis, de manière récursive, on recherche les solutions supportées se trouvant entre deux solutions supportées trouvées z^r and z^s selon la direction de recherche normale à la droite (z^r, z^s) (voir figure 7.1), définie par : $\lambda_1 = z_2^{(r)} - z_2^{(s)}$, $\lambda_2 = z_1^{(s)} - z_1^{(r)}$. Ensuite, toute nouvelle solution supportée découverte engendre deux nouvelles recherches (voir figure 7.2), jusqu'à ce que les recherches soient infructueuses.

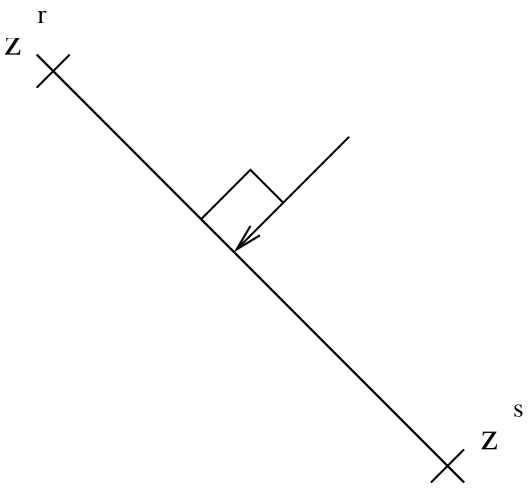


FIG. 7.1 – 1*ère* phase : orientation de la recherche entre deux solutions.

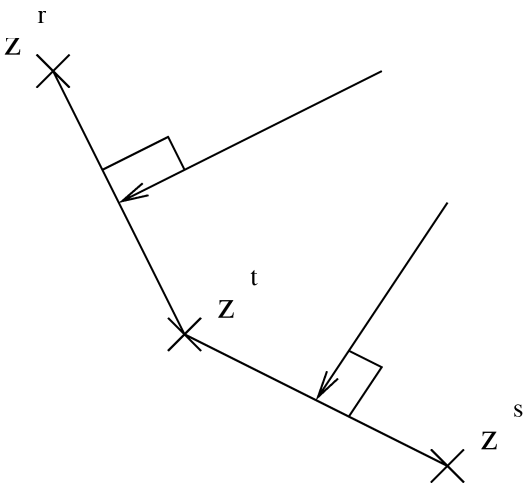


FIG. 7.2 – 1*ère* phase : nouvelles recherches.

7.2.1.2 La deuxième phase

La seconde phase consiste à trouver les solutions non supportées. Les solutions recherchées se trouvent alors forcément dans la zone définie par le triangle $z^r z^s Y$, où Y est le point $[z_1^{(s)}, z_2^{(r)}]$. La figure 7.3 montre cette zone. La recherche effective est donc effectuée dans la région dominant le point Y, en contraignant les valeurs des deux objectifs.

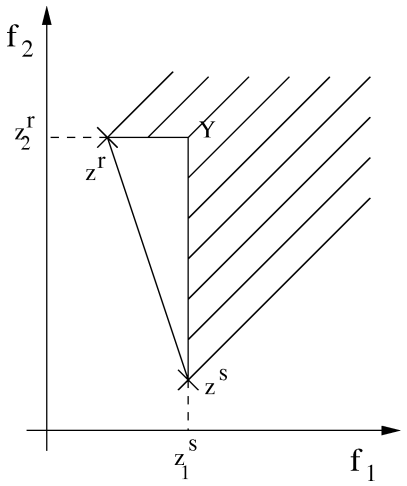


FIG. 7.3 – 2*ème* phase : recherche des solutions non supportées avec restriction de l'espace de recherche.

7.2.1.3 Caractéristiques de la méthode à deux phases

Afin de limiter le temps de la recherche, on pourrait vouloir limiter l'approche à l'exécution de la première phase de l'algorithme, qui permet de découvrir toutes les solutions Pareto optimales supportées. Cependant, ce type d'approche n'est pas conseillé lorsque la proportion de solutions non-supportées est trop importante. Dans le cas du Flow-shop bi-objectif, il para^indispensable d'effectuer les deux phases de la recherche, comme le montre la figure 7.4 illustrant l'ensemble Pareto optimal pour le problème *ta*_20_5_02 pour lequel seules les 2 extrêmes sont des solutions supportées.

TPM est dédiée à la base à la résolution de problèmes bi-objectif. Cependant des travaux existent pour étendre cette approche aux problèmes à N objectifs |Ten03|.

7.2.2 Application de TPM au $BOFSP$

La méthode à deux phases permet de résoudre un problème bi-objectif de manière exacte sans avoir à explorer la totalité de l'espace de recherche. Seules les régions pertinentes sont visitées. Pour pouvoir appliquer cette méthode, il est nécessaire d'utiliser une méthode exacte mono-objectif pour résoudre de manière optimale les différentes agrégations envisagées. De plus, $BOFSP$ a la particularité d'avoir la recherche de chacun de ses

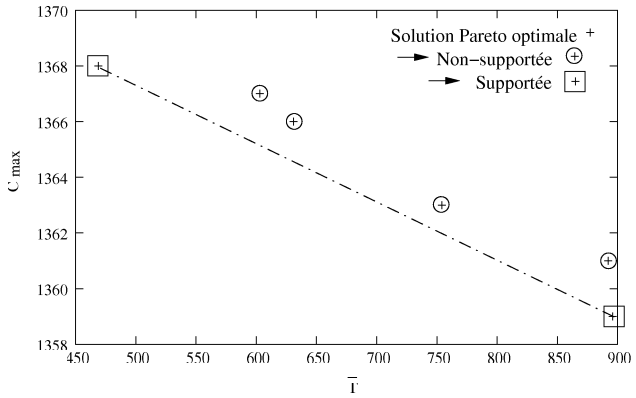


FIG. 7.4 – Importance de la recherche des solutions Pareto optimales non-supportées. Dans le cas du problème *ta_20_5_02*, seules les deux solutions extrêmes sont supportées. Les autres solutions Pareto optimales peuvent offrir de bons compromis au décideur.

objectifs NP-difficile, ce qui rend la première phase difficile à résoudre. Des mécanismes spécifiques sont nécessaires afin de réduire le temps de calcul de la première phase.

Nous présentons donc en un premier temps les bornes que nous utilisons pour la méthode exacte mono-objectif. Ensuite, nous décrivons l'algorithme de $B\&B$ utilisé. Puis nous présentons les modifications apportées à la méthode afin de l'adapter au problème du $BOFSP$.

7.2.2.1 Méthode exacte mono-objectif pour le $BOFSP$

Pour les deux objectifs que nous considérons, il n'existe pas de méthode exacte performante connue autre que le $B\&B$. Cette méthode a donc été implémentée, en utilisant une borne pour chaque objectif. Le C_{max} étant très largement étudié, nous utilisons une borne existante, l'autre borne a été proposée dans le cadre de cette étude [LDT05].

7.2.2.1.1 Borne pour le *makespan*

Lenstra *et al.* [BJLR78] ont proposé une borne inférieure pour le problème de Flow-shop à M machines de type $F/permu/C_{max}$. Cette borne est utilisée pour notre application au Flow-shop bi-objectif. Le calcul de la borne consiste premièrement à considérer un problème avec *lags* (temps de transport) à deux machines de type $F2/l_j, permut/C_{max}$. Pour le calcul de notre borne, le *lag* correspond à la somme des temps d'exécution sur les machines $[k+1 \dots l-1]$. Soit $P^*(M_k, M_l)$ une solution optimale de ce problème (calculée en temps polynomial [Jac56, Joh54, Mit59]). La borne calculée vaut alors :

$$LB = \max_{(1<k<l\leq m)} (P^*(M_k, M_l) + \min_{((i,j)\in J^2, i\neq j)} (Arr_{i,k} + Dep_{j,l})) \tag{7.1}$$

avec $Arr_{i,k} = \sum_{h=1}^{k-1} p_{i,h}$ et $Dep_{j,l} = \sum_{h=l+1}^{m} p_{j,h}$.

7.2.2.1.2 Bornes pour le retard total

Deux bornes complémentaires ont été proposées pour le retard optimal ($(\sum T_i)^*$). Chacune d'elles est meilleure pour une partie non-négligeable des solutions partielles envisagées.

La première borne est celle proposée par Kim [Kim95], la deuxième est proposée par Lemesre *et al.* [LDT05]. Soient les définitions suivantes :

- d_i^{EDD} est la $i^{ème}$ date de fin en considérant un ordonnancement au plus tôt (*Earliest Due Date* - (EDD)).
- $x^+ = max(x, 0)$.
- $d_{i,j}$ représente la date de fin d'un job i sur une machine j. Cette date est calculée en fonction de la date de fin réelle du job i et de ses temps d'exécution sur les différentes machines. Cette valeur est définie par : $d_{i,j} = d_{i,m} - \sum_{k=j+1}^{m}(p_{i,k})$.
- $E^{\bar{S}}_{(k,l)}$ est le temps minimal pour passer d'une machine k à une machine l. Ce temps est calculé en sommant les temps d'exécution sur l'intervalle des machines $[k, l]$ des jobs restant à ordonnancer ($\bar{S}$). Cette valeur est définie par : $E^{S}_{(k,l)} = min_{i\in\bar{S}}(\sum_{j=k}^{l} p_{i,j})$.

La première borne consiste à séparer les temps d'exécution des temps de fin des jobs, puis à ordonnancer selon les dates de fin croissante en affectant les durées d'exécution dans l'ordre croissant. Soit $\prod^{\bar{S}}_{(i,k)}$ la somme des i plus petits temps d'exécution sur la machine k pour les jobs de $\bar{S}$. Alors, le $i^{ème}$ job ordonnancé de $\bar{S}$ ne peut se terminer sur la machine l avant la date $C^s_k + \prod^{\bar{S}}_{(i,k)} + E^{S}_{(k,l)}$. Nous avons, $\forall S$:

$$(\sum T_i)^* \geq \max_{1\leq k<l\leq m}\left(\sum_{\substack{i=1\\ i\in\bar{S}}}^{|\bar{S}|}\left(C^s_k + \prod_{(i,k)}^{\bar{S}} + E^{\bar{S}}_{(k,l)} - d_{i,l}^{EDD,\bar{S}}\right)^+\right) + \sum_{\substack{i=1\\ i\in S}}^{|S|}\left(C_{i,m} - d_i\right)^+ \quad (7.2)$$

avec $d_{i,l}^{EDD,\bar{S}}$ la $i^{ème}$ date de fin en considérant un ordonnancement au plus tôt en considérant les jobs de $\bar{S}$, et $\sum_{\substack{i=1\\ i\in S}}^{|S|}(C_{i,m} - d_i)^+$ est le retard engendré par les jobs déjà ordonnancés.

La deuxième borne proposée consiste à ordonnancer les jobs suivant les dates de fin croissante. Puis, on calcule le retard engendré par chaque job en le limitant au temps d'exécution de ce job. Soit $P^{\bar{S}}_{(i,k)}$ la somme des temps d'exécution des i premiers jobs, dans $\bar{S}$, ordonnancés au plus tôt sur la machine k.

$$(\sum T_i)^* \geq \max_{1\leq k<l\leq m}\left(\sum_{\substack{i=1\\ i\in\bar{S}}}^{|\bar{S}|}\left(min\left((C^s_k + P^{\bar{S}}_{(i,k)} + E^{\bar{S}}_{(k,l)} - d_{i,l}^{EDD,\bar{S}})^+, p_{i,l}^{EDD}\right)\right)\right) + \sum_{\substack{i=1\\ i\in S}}^{|S|}\left(C_{i,m} - d_i\right)^+ \quad (7.3)$$

Les preuves d'existence de ces bornes sont détaillées dans [LDT05].

7.2.2.1.3 Algorithme de *branch & bound* pour le $BOFSP$

Avec les bornes que nous venons de décrire, les agrégations réalisées pour le $BOFSP$

peuvent être résolues de manière optimale à l'aide d'un algorithme de $B\&B$. La recherche est réalisée "en profondeur d'abord", en privilégiant le noeud le plus prometteur. Ainsi, cela permet de trouver une bonne solution rapidement.

Le calcul du retard est beaucoup plus important lors du placement des derniers jobs en fin d'ordonnancement. Donc, à la différence des méthodes "classiques", l'algorithme de construction des solutions peut choisir de placer un nouveau job soit en début soit en fin d'ordonnancement. Ainsi, lors de l'exploration d'un noeud, deux ensembles de solutions partielles sont construits selon que l'on ajoute un job en début ou en fin d'ordonnancement. Une fois ces deux ensembles construits, l'algorithme explore l'ensemble comportant le plus petit nombre de solutions partielles prometteuses[1].

7.2.2.2 Apports à la méthode TPM initiale.

TPM est une méthode générique qui, appliquée au Flow-shop bi-objectif, peut être améliorée.

7.2.2.2.1 Calcul des solutions extrêmes

La première étape de la première phase consiste à calculer les solutions Pareto optimales extrêmes. Cette partie de la recherche se révèle très coûteuse en temps de calcul pour le problème du Flow-shop bi-objectif. Dans notre cas, cette étape est résolue par un algorithme de $B\&B$ mono-objectif. Mais, durant la recherche mono-objectif, les solutions de coût égal à la meilleure courante doivent être prise en compte. En effet, en cas d'égalité sur le premier critère, il est nécessaire de conserver la solution de meilleur coût sur l'autre objectif (tri lexicographique des objectifs). Dans le cas du Flow-shop bi-objectif, de nombreuses solutions sont de même coût, ce qui rend cette recherche des extrêmes très longue car cela entraîne l'exploration de nombreux noeuds soit-disant "prometteurs", mais qui sont en fait de coût identique à la solution optimale courante. La solution retenue pour éviter ce problème est de réaliser la recherche en deux étapes : en premier lieu, on cherche une solution optimale S^* pour l'un des objectifs, puis on lance une deuxième recherche sur le deuxième objectif, en contraignant les solutions à être au moins aussi bonnes que S^* sur le premier objectif.

7.2.2.2.2 Intervalles de recherche

L'intérêt majeur de la première phase est de trouver des solutions supportées qui permettront par la suite de réduire l'espace de recherche pour la seconde phase. Cependant, lorsque plusieurs solutions supportées sont très proches les unes des autres (voir figure 7.5), il est peu intéressant de les chercher toutes lors de la première phase, car cette recherche est très coûteux en temps de calcul. Donc, la première phase est exécutée seulement entre des solutions supportées suffisamment éloignées entre elles. La deuxième phase se chargera alors de trouver naturellement les solutions supportées restantes.

[1] C'est-à-dire l'ensemble pour lequel il y a le plus grand nombre de coupures apportées pas les différentes bornes.

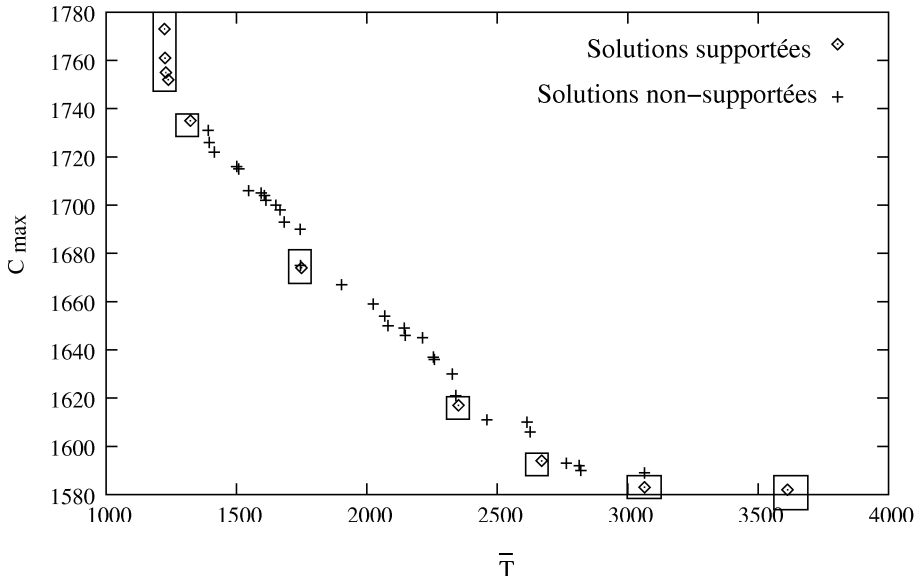


FIG. 7.5 – Ensemble Pareto optimal pour l'instance *ta_20_10_01* du *BOFSP*. Seulement 10 solutions sont supportées parmi les 42 solutions composant l'ensemble Pareto optimal. De plus, 5 de ces solutions supportées sont consécutives.

7.2.3 Résultats

Nous avons testé TPM dans les mêmes conditions que les autres algorithmes, mais avec une seule exécution par instance, l'approche étant déterministe. Les fronts exacts calculés lors des expérimentations se trouvent dans l'annexe 8.

La table 7.1 présente les temps de calcul nécessaires pour la découverte de la totalité des solutions Pareto optimales. S (respectivement $\bar{S}$) est le nombre de solutions supportées (respectivement non-supportées) qui ont été obtenues. L'instance *ta_20_20_01* n'a été résolue que par une version parallèle, présentée dans [LDT05]. La résolution de cette instance a nécessité environ 10 jours sur l'$IBMRS6000/SP$ utilisé pour $PAGMA$ (voir section 5.3.1.2). L'instance *ta_50_5_01* n'a pas pu être résolue de manière optimale, bien que les différentes approches hybrides aboutissent toujours aux mêmes solutions, qui semblent donc optimales. Mais TPM ne nous permet pas de conclure de manière définitive sur l'optimalité de ces solutions.

Les temps de calcul de TPM sont très grands, hormis pour les deux plus petites instances. Dans le chapitre précédent, nous avons montré que $AGMA$ découvrait l'ensemble Pareto optimal pour l'ensemble des exécutions réalisées, et ceci dans un temps raisonnable. En effet, pour les 2 instances à 20 jobs et 10 machines, cet ensemble optimal est toujours découvert en quelques minutes avec $AGMA$ contre plus d'un jour pour TPM. Ceci renforce l'idée que $AGMA$ peut aider TPM à accélérer sa recherche exacte, en lui fournissant les informations nécessaire pour guider les recherches. Ce principe fait l'objet de la première méthode coopérative méta/exacte présentée dans la section suivante.

TAB. 7.1 – Résultats de TPM.

Instance	Temps	S	$\bar{S}$
$ta\_20\_5\_01$	12”	3	1
$ta\_20\_5\_02$	9’11”	2	4
$ta\_20\_10\_01$	38h46’	10	32
$ta\_20\_10\_02$	25h17’	8	23
$ta\_20\_20\_01$	Parallèle	8	23

Nous remarquons également que, pour $AGMA$, l'instance $ta\_20\_10\_01$ est plus facile à résoudre que l'instance $ta\_20\_10\_02$ ($9'08$ en moyenne pour la première contre $18'29$ pour la seconde). Or, pour TPM ce constat est inversé. Cet exemple montre que TPM et $AGMA$ ne sont pas efficaces sur les mêmes instances, que ces méthodes sont bien antagonistes et qu'il peut être intéressant de les faire coopérer.

7.3 Approches coopératives méta/exacte

Dans le chapitre 2, nous avons présenté différentes approches pour les coopérations méta/exacte. Ces approches ont été réparties en différentes classes. Dans cette section, nous proposons deux types d'approches bien différentes. En effet, dans un premier temps, nous proposons une coopération réalisée dans le cadre d'une approche de résolution exacte. Puis nous présentons deux approches coopératives heuristiques.

Les différentes approches de coopération que nous avons réalisées ont été établies de la manière la plus simple possible et peuvent certainement être améliorées en les complexifiant. Le but ici est de proposer différentes approches et d'en évaluer les apports dans le cadre du $BOFSP$.

7.3.1 Approche coopérative exacte

Dans la classification présentée dans le chapitre 2, nous avons recensé quelques coopérations méta/exacte réalisées dans le cadre d'une approche d'optimisation exacte [PVDD98, PRRL97, KF04, FRW01, CDP02, BKY02]. La manière la plus naturelle d'accélérer une méthode exacte de type $B\&B$ par une méthode heuristique est de chercher de bonnes solutions initiales, afin d'avoir de bonnes bornes dès le lancement de la recherche exacte [PVDD98]. C'est l'approche que nous avons retenue.

7.3.1.1 Description

Dans cette coopération, la métaheuristique est lancée dans un premier temps, puis les valeurs trouvées sont utilisées pour améliorer l'initialisation de la méthode exacte. La coopération est de type **LRH**($AGMA$+TPM). Ici, la particularité de ce type de coopéra-

tion est que l'on fournit à la méthode exacte un ensemble de solutions, à la différence des approches trouvées dans la littérature de même type, mais dans un cadre mono-objectif, qui se basent sur une solution unique (voir figure 7.6). Le but ici est de déterminer si ce type d'approche est fructueuse dans le cadre multi-objectif.

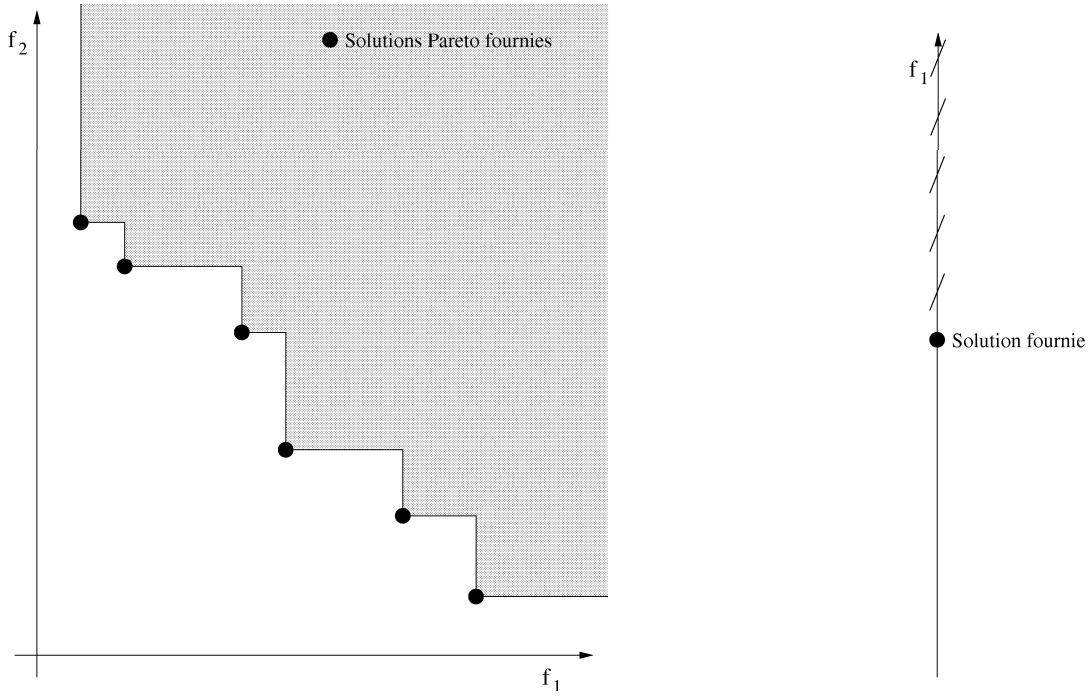


FIG. 7.6 – Approche coopérative exacte : distinction mono-objectif/multi-objectif.

7.3.1.2 Implémentation

Dans cette approche, nous exécutons TPM dans son ensemble, mais en utilisant les solutions Pareto optimales fournies par $AGMA$ comme valeur initiale des différents $B\&B$ effectués. De cette manière, une part non négligeable des noeuds jadis explorés durant le déroulement du $B\&B$ peuvent maintenant être coupés grâce à l'utilisation des solutions de $AGMA$ comme bornes inférieures initiales. Pour l'application au $BOFSP$, aucun paramétrage n'est nécessaire pour définir cette coopération.

7.3.1.3 Expérimentations

Une question importante se pose pour cette coopération, comme pour beaucoup de coopérations : quel est le meilleur moment pour arrêter le déroulement de $AGMA$ pour lancer TPM ? Pour les petites instances, comme celles à 20 jobs et 5 machines, la méthode exacte originelle est déjà très rapide. Pour les autres intances à 20 jobs, on remarque que

la métaheuristique aboutit très rapidement au front exact, en comparaison à TPM. Donc, afin d'évaluer les apports potentiels de ce type de coopération, nous avons, dans un premier temps, exécuté TPM avec le front Pareto optimal comme valeur d'entrée.

Pour évaluer le temps global de recherche, nous avons alors additionné le temps maximal de découverte du front optimal par $AGMA$ (sur les 10 exécutions effectuées) avec le temps de calcul de TPM utilisant le front 'exact' en entrée.

Le tableau 7.2 présente les résultats obtenus pour cette coopération. $T(AGMA)$ est le temps maximal de découverte du front optimal par $AGMA$, $TPMbornes$ est le temps de résolution exacte de l'instance par TPM en utilisant le front exact en borne d'entrée, T_{coop} est le temps total de la coopération, $Apport_{TPM}$ est le gain de temps de TPM en lui fournissant ces bornes, $Apport$ est le gain de temps de l'approche coopérative par rapport à TPM seule.

TAB. 7.2 – Résultats de l'approche coopérative exacte.

Instance	**T(AGMA)**	**TPM_{bornes}**	**T_{coop}**	**TPM**	**$Apport_{TPM}$**	**Apport**
$ta\_20\_5\_01$	34"	8"	42"	12"	33.0%	-350%
$ta\_20\_5\_02$	1'41"	1'33"	3'14"	9'11"	83.1%	65.8%
$ta\_20\_10\_01$	14'27"	29h04'	29h19'	38h46'	25.0%	24.0%
$ta\_20\_10\_02$	36'25"	17h58'	18h34'	25h17'	30.0%	27.6%

La colonne $Apport_{TPM}$ nous montre que, effectivement, fournir les bornes dès l'initialisation de TPM permet d'accélérer la recherche exacte. Cette accélération est supérieure à 25% pour toutes les instances, et même de 83.1% pour l'instance $ta\_20\_5\_02$.

Comparons maintenant le temps global de l'approche coopérative par rapport à TPM. Le gain reste assez conséquent sur les 3 dernières instances (gain de temps de 39.1% en moyenne). Pour la première instance, la résolution exacte par TPM est très rapide, donc le lancement d'une métaheurisque hybride complexe comme $AGMA$ en pré-traitement ne fait que ralentir la recherche.

En conclusion, le temps nécessaire pour la coopération est en général plus petit que l'approche par TPM uniquement. Cependant, l'apport n'est pas suffisant pour permettre de résoudre de manière exacte des problèmes plus grands tels que $ta\_50\_5\_01$ ou $ta\_50\_10\_01$.

Notons également que, dans le cadre d'applications réelles, la coopération est dédiée à la recherche d'un ensemble Pareto optimal inconnu. Dans ce cas, il faut bien sûr déterminer quand arrêter le déroulement de la métaheuristique. Divers critères d'arrêt peuvent être définis, mais celui du seuil de convergence utilisé dans $AGMA$ semble être adapté (mais avec une valeur de seuil plus faible).

7.3.2 Approche heuristique à "large voisinage"

Les algorithmes de recherche à base de voisinage (comme les méthodes de descente, par exemple) sont une classe très répandue d'heuristiques qui consistent à améliorer les solutions en explorant le "voisinage" de la solution courante, dans le but de l'améliorer. Ahuja *et al.* font remarquer que l'efficacité de ce type d'algorithmes dépend grandement du choix de(s) l'opérateur(s) de voisinage [AEOP02]. De plus, d'une manière générale, plus un voisinage est grand, plus les résultats obtenus par ce type d'algorithmes sont bons. Néanmoins, la génération d'un voisinage très large est beaucoup plus coûteuse en temps de calcul.

Les algorithmes de recherche dits "à large voisinage" (*Very Large Neighborhood Search - VLNS*) consistent à explorer des voisinages très larges en utilisant des mécanismes permettant l'exploration de ce voisinage dans un temps raisonnable. Dans [AEOP02], quelques techniques d'exploration de ces voisinages sont décrites. Dans cette section, nous proposons d'utiliser une méthode exacte de type *B&B* comme méthode d'exploration de large voisinage.

7.3.2.1 Description

La coopération réalisée ici est de type **HTH(***Méta+Exact***)**, où *Méta* désigne une métaheuristique et *Exact* une méthode exacte. Le principe de cette coopération est schématisé figure 7.7.

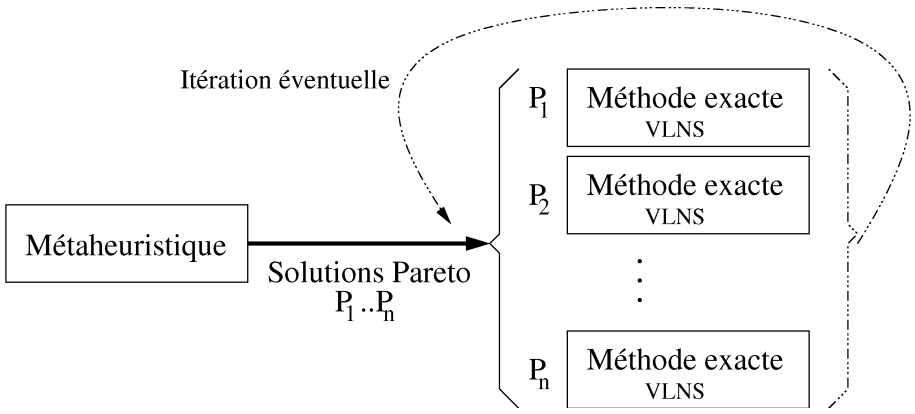


FIG. 7.7 – Principe de la coopération *VLNS*. Les recherches de large voisinage sont effectuées sur chaque individu non-dominé fourni par la métaheuristique. On peut éventuellement itérer le processus jusqu'à convergence.

L'idée principale de l'approche est de réduire l'espace des solutions envisagées par la méthode exacte, par exemple ici un *B&B*, en coupant les branches de l'arbre du *B&B* lorsque la solution en construction "s'éloigne" trop d'une solution initiale fournie par une métaheuristique. Durant la recherche, les seules solutions à explorer seront donc celles qui seront distantes de moins de δ_{max} de la solution initiale. Pour la mise en oeuvre d'une

telle coopération, il est nécessaire de définir au préalable une notion de distance entre les solutions.

La figure 7.8 montre un exemple de construction de solutions envisageables par l'approche $VLNS$.

7.3.2.2 Implémentation

Pour l'application de cette méthode de coopération au $BOFSP$, nous avons sélectionné comme métaheuristique de base celle qui nous a donné les meilleures solutions jusqu'ici. Donc, les solutions initiales de l'approche $VLSN$ seront fournies par la métaheuristique $PAGMA$. L'approche $VLSN$ sera réalisée par l'algorithme TPM.

Pour calculer la distance des solutions partielles à la solution initiale, il semble naturel de se référer à l'opérateur *Shift*. Cependant, il est nécessaire de calculer cette distance de manière incrémentale durant l'exploration de l'arbre de $B\&B$. Afin de ne pas rendre trop grande la complexité algorithmique du calcul de distance, nous utiliserons comme opérateur de distance $Shift_A$ correspondant à l'opérateur *Shift* mais en se restreignant uniquement aux insertions "vers l'avant" (les jobs sont déplacés uniquement vers l'avant). Le calcul de distance correspond alors au nombre de ruptures d'ordre comme l'exemple de la figure 7.8.

La taille du voisinage exploré par l'opérateur $Shift_A$ est en $\Theta(N^2)$, et l'on peut évaluer la taille de l'espace exploré par TPM à $\Theta(N^{2d_{max}})$ (pour $d_{max} << N$). Cette complexité implique de limiter de manière drastique la valeur d_{max} pour les instances comportant beaucoup de jobs.

7.3.2.3 Expérimentations

Pour nos expérimentations, nous avons été confrontés au fait que cette approche est très gourmande en temps de calcul. Ainsi, nous avons dû fixer la valeur de d_{max} à 2, ce qui est très faible (en effet, une valeur de 1 correspond à une itération de recherche locale). De plus, nous n'avons pas pu itérer le processus de recherche sur un large voisinage.

Nous n'exposons pas ici les améliorations obtenues, étant donné qu'elle sont minimes. Pour les instances à 50 jobs, aucune amélioration n'a été obtenue, pour des tests ayant duré plusieurs heures. Pour les plus grandes instances, seules quelques nouvelles solutions ont été obtenues (voir figure 7.9), et ce dans des temps très grands (de l'ordre de plusieurs jours d'exécution).

Cette approche, bien qu'ayant fait ses preuves pour d'autres problèmes et d'autres approches de résolution, n'est pas efficace dans notre cas. Cependant, il pourrait être intéressant de l'utiliser de manière coopérative, en n'envisageant par exemple qu'une partie de l'ordonnancement total, comme cela est réalisé dans la prochaine méthode coopérative que nous présentons.

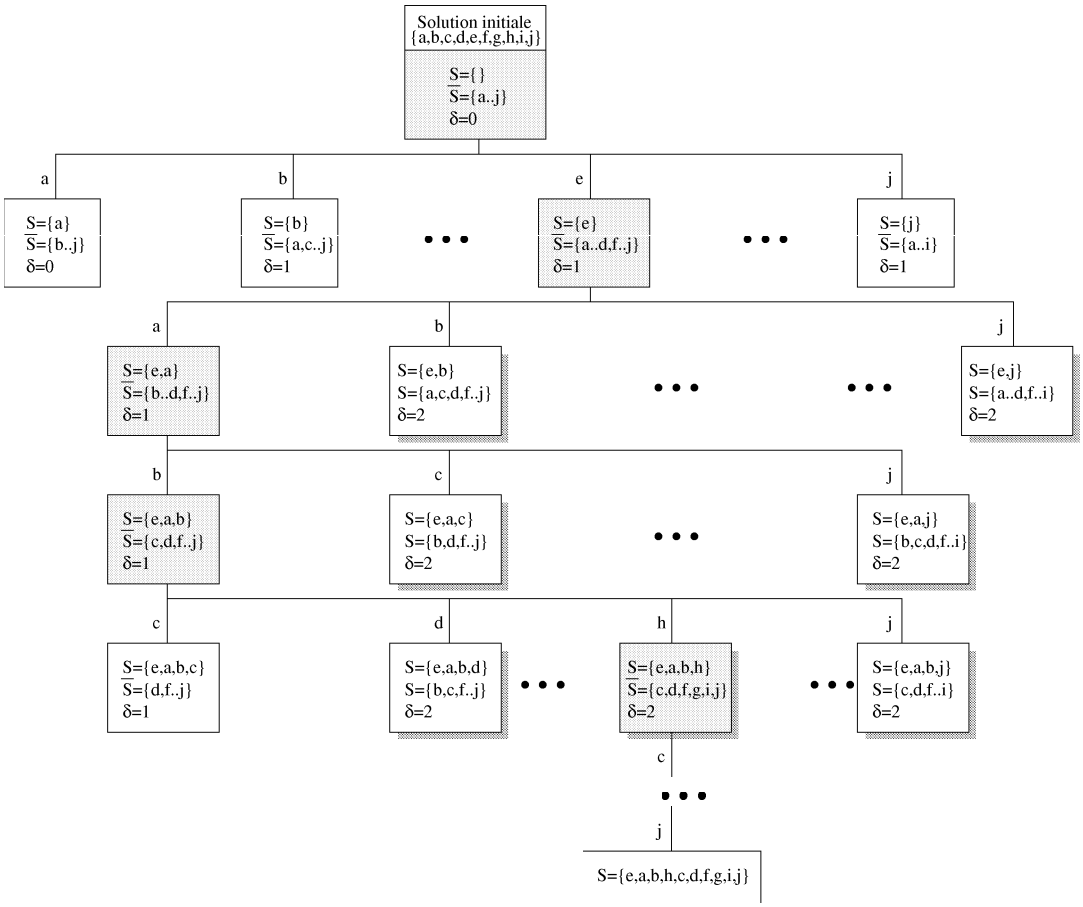


FIG. 7.8 – Déroulement de TPM avec approche $VLNS$. On considère une solution initiale $abcdefghij$. Pour cet exemple, nous utilisons les ensembles S et $\bar{S}$ correspondant respectivement aux jobs faisant partie ou non de la solution partielle. Au départ de l'exécution d'un $B\&B$, S est vide et $\bar{S}$ contient les jobs $a, b, c, d, e, f, g, h, i, j$. Durant la construction des solutions, on utilise une valeur δ initialisée à 0, et incrémentée à chaque fois que l'ordre d'apparition des jobs de la solution initiale n'est pas respecté. Lorsque $\delta \geq \delta_{max}$, une seule solution est alors explorée dans la branche courante, celle qui consiste à placer les jobs restant dans leur ordre d'apparition dans la solution initiale. Sur cet exemple, où la valeur δ_{max} a été fixée à 2, les valeurs de S, $\bar{S}$ et δ durant l'exploration de l'arbre sont détaillées à chaque noeud. À chaque étage de l'arbre, seul le noeud grisé est développé. Lorsque le seuil δ_{max} est atteint, les noeuds sont ombragés. À partir de ce point, une seule solution est explorée, comme la solution $e, a, b, h, c, d, f, g, i, j$ sur l'exemple. Les autres solutions que l'on peut construire avec pour base e, a, b, h sont trop éloignées de la solution initiale.

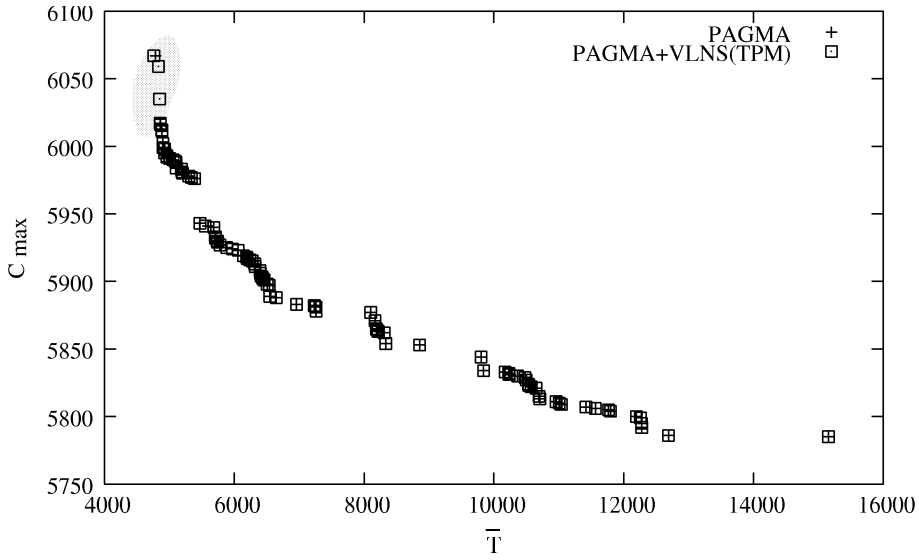


FIG. 7.9 – Coopération $PAGMA/VLNS$: Résultat sur l'instance $ta\_100\_10\_01$. Seules 2 nouvelles solutions (se trouvant dans la région grisée) ont été découvertes en 1 itération de la recherche sur un voisinage large.

7.3.3 Approche partitionnement

Les méthodes d'optimisation par partitionnement sont généralement utilisées lorsque les instances de problèmes traitées sont très grandes. Elles permettent d'utiliser des méthodes d'optimisation impraticables sur l'ensemble de l'instance. C'est le cas de l'ensemble des méthodes d'optimisation exactes, qui sont toutes limitées aux problèmes de petite taille. Nous proposons d'utiliser ce principe afin de pouvoir appliquer TPM aux instances du $BOFSP$ de grande taille.

7.3.3.1 Description

Comme l'approche précédente, la coopération réalisée est de type **HTH**($Méta$+$Exact$), où $Méta$ désigne une métaheuristique et $Exact$ une méthode exacte. Le principe de cette coopération, que nous appelerons $SPPOBB$ (*Simple Partitioning Post Optimization Branch & Bound*), est schématisée figure 7.10.

Cette coopération est, comme la précédente, destinée à améliorer les solutions fournies par la métaheuristique, en se servant de leur structure génotypique pour limiter la taille de l'espace de recherche de la méthode exacte. La méthode coopérative présentée ici est également heuristique, et consiste à optimiser de manière exacte des zones des solutions initiales. On définit des zones de recherche sur les solutions, puis l'espace de recherche associé à ces zones est exploré de manière exacte, puis on remet à jour l'ensemble des solutions Pareto optimales afin de réitérer le processus sur une autre zone des individus.

La structure du paysage exploré, que l'on peut assimiler à une exploration de voisinage,

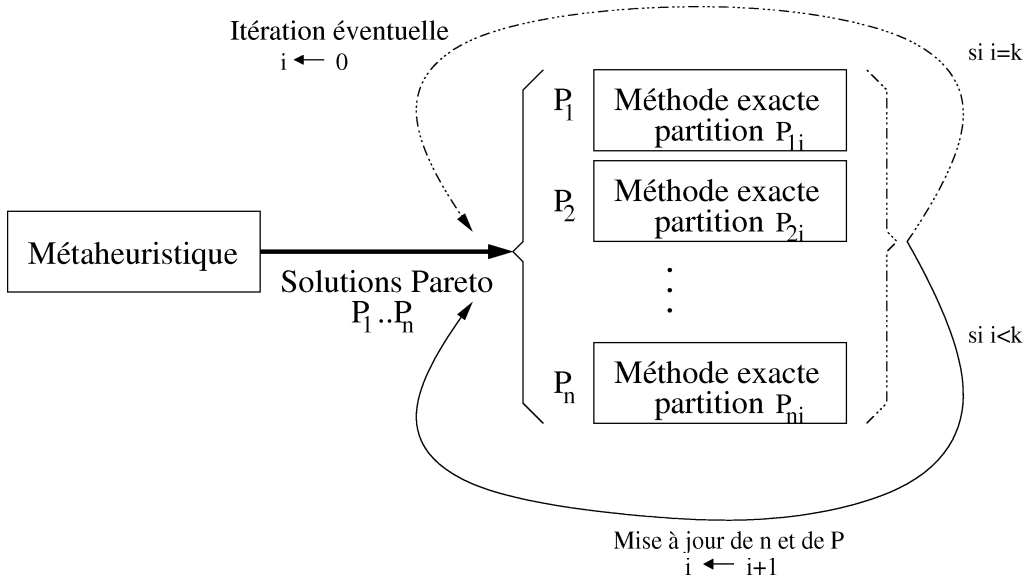


FIG. 7.10 – Principe de la coopération $SPPOBB$. Une population initiale est générée par une métaheuristique. Puis on optimise partition par partition l'ensemble des solutions, en mettant à jour l'ensemble Pareto P ainsi que sa taille n après chaque recherche (P_{ki} correspond à la $ième$ partition de la solution k). On peut éventuellement itérer le processus une fois toutes les partitions explorées.

semble moins adéquate que pour la méthode précédente, puisque l'opérateur permet un mélange libre entre deux points définis des solutions initiales. Cependant, ce partitionnement permet de pouvoir construire une grande partie des ordonnancements dès le départ de la recherche exacte. Ainsi l'estimation des bornes est précise dès le départ de la construction de l'arbre de recherche.

Soit p le nombre de partitions envisagées pour chaque solution, s leur taille et N le nombre de jobs de l'instance. L'algorithme 7.1 détaille le déroulement de $SPPOBB$, et la figure 7.11 schématise le déroulement de l'optimisation d'une partition.

7.3.3.2 Implémentation

Nous utilisons une fois de plus $PAGMA$ pour coopérer avec TPM, afin d'obtenir les résultats les plus performants possibles.

TPM, appliquée au $BOFSP$, est bien adaptée à l'optimisation par partitions réalisée par la suite. En effet, la méthode à deux phases que nous avons implémentée explore l'arbre de recherche en plaçant les jobs aussi bien en début qu'en fin d'ordonnancement de la solution partielle explorée. Donc, si nous définissons une partition allant du $X_i^{ème}$ au $X_j^{ème}$ job, la méthode commencera par placer les jobs $0..X_i-1$ en début d'ordonnancement et les jobs $X_j+1..N$ en fin d'ordonnancement, ceci avant d'explorer de manière exacte les possibilités d'ordonnancement restantes grâce à TPM.

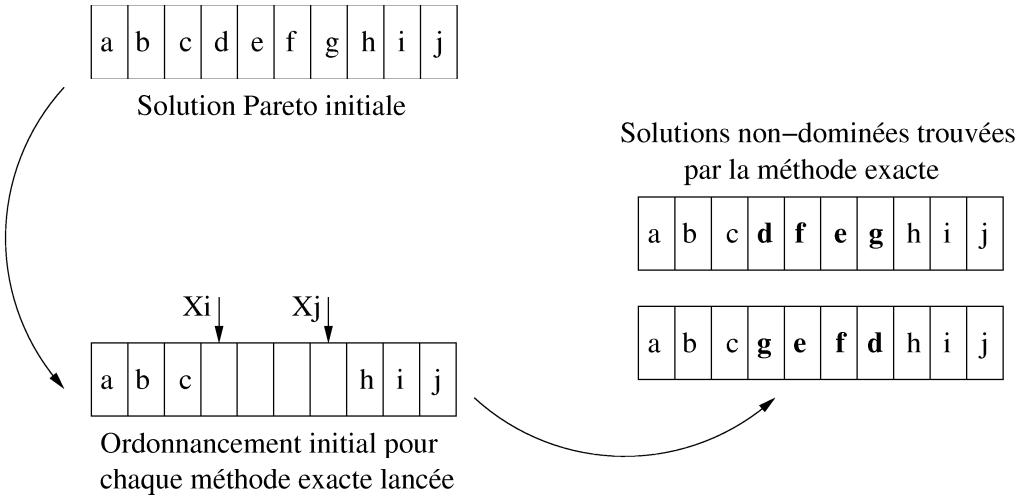


FIG. 7.11 – Optimisation d'une partition avec l'algorithme $SPPOBB$. On itère ensuite le processus sur l'ensemble des solutions non-dominées durant la recherche.

Algorithme 7.1 Algorithme $SPPOBB$.

Création d'une population initiale de solutions non-dominées $P[0]..P[k]$ avec une méthode heuristique ou métaheuristique.
pour i de 0 à $p-1$ **faire**
  $PO = \{\}$.
  pour j de 0 à k **faire**
    Optimiser une partition de P[j] de manière exacte.
    On cherche le placement optimal des jobs $N*i/p$ à $N*i/p+s$ de la solution initiale $P[j]$.
    On obtient une population de solutions non-dominées P'.
    $PO = PO \cup P'$
  fin pour
  Soit P l'ensemble des solutions non-dominées de PO.
fin pour

La figure 7.12 montre un exemple d'exploration d'un arbre de $B\&B$. Dans notre cas, TPM lancera plusieurs arbres de recherches pour la même partition, mais ces arbres seront tous initialisés de la même manière, en plaçant dès le départ les jobs ne faisant pas partie de la partition à optimiser. Dans cette figure, nous considérons une solution initiale $a, b, \ldots, i, j$, faisant partie des solutions non-dominées obtenue dans un premier temps par une méthode d'optimisation approchée ($AGMA$ dans notre cas). La taille des partitions est ici de 4, pour un ordonnancement complet de 10 jobs. On pose $X_i = 4$ et $X_j = 7$, ce qui correspond aux jobs d, e, f, g dans l'ordonnancement initial. Lors de chaque recherche lancée par TPM, la première phase de la recherche consistera à fixer les trois jobs a, b et c en début d'ordonnancement et les jobs h, i et j en fin d'ordonnancement (précédés d'un $'-'$ sur la figure). Ensuite la recherche est lancée pour les jobs restant à ordonnancer. Sur l'exemple, 5 solutions ont finalement été explorées, après avoir profité des règles de coupes habituelles :

- a,b,c,-j,-i,-h,d,-e,f,g correspondant à l'ordonnancement abcdfgehij.
- a,b,c,-j,-i,-h,d,-g,e,f correspondant à l'ordonnancement abcdefghij (la solution initiale).
- a,b,c,-j,-i,-h,d,-g,f,e correspondant à l'ordonnancement abcdfeghij.
- a,b,c,-j,-i,-h,g,-d,-e,f correspondant à l'ordonnancement abcgfedhij.
- a,b,c,-j,-i,-h,g,-d,-f,e correspondant à l'ordonnancement abcgefdhij.

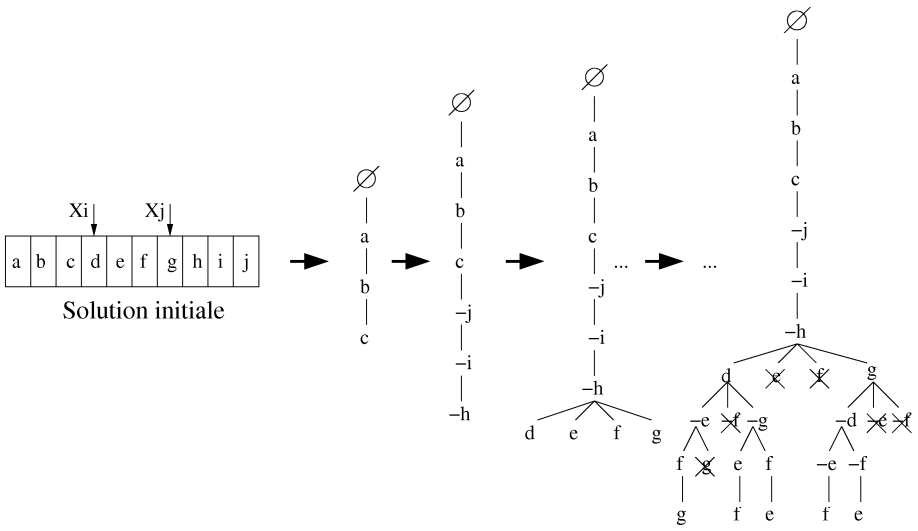


FIG. 7.12 – Exploration exacte d'une partition de l'ordonnancement complet.

7.3.3.3 Expérimentations

Nous avons testé $SPPOBB$ sur les instances de Flow-shop bi-objectif pour lesquelles nous n'obtenons pas l'ensemble Pareto optimal, c'est-à-dire les instances à 50 jobs ou plus, excepté les instances à 5 machines, pour lesquelles il semble que $PAGMA$ obtienne l'ensemble Pareto optimal.

7.3.3.3.1 Paramétrage

Deux paramètres principaux influent sur le temps d'exécution de l'algorithme, ainsi que sur la qualité des résultats obtenus. Nous les avons établis afin d'équilibrer ces deux critères antagonistes :

- Taille des partitions : pour les instances qui nous intéressent ici, le nombre de solutions Pareto obtenues par $AGMA$ varie entre une vingtaine et deux cents individus. Afin de limiter le temps nécessaire à l'exécution de TPM à quelques secondes ou quelques minutes par partition, nous avons limité la taille des partitions à 15 jobs pour les problèmes à 10 machines, et 12 jobs pour les problèmes à 20 machines.
- Nombre de partitions par solution : le nombre de partitions envisagées doit être suffisant pour permettre de couvrir au moins une fois chaque job par l'approche TPM. De plus, il semble intéressant de superposer les partitions deux à deux pour permettre à un même job de pouvoir être déplacé plusieurs fois. Ainsi, un job se trouvant en début de séquence dans l'ordonnancement initial peut être déplacé en fin de séquence par des mouvements successifs réalisés durant l'optimisation de différentes partitions, comme le montré l'exemple de la figure 7.13. D'autre part, cette superposition des partitions ne doit pas être trop importante, car plus on considère de partitions, plus le temps de calcul total sera grand. Afin de prendre en compte ces deux points, nous avons choisi de considérer 8 partitions pour les problèmes à 50 jobs, 16 partitions pour les instances à 100 jobs, et 32 partitions pour les instances à 200 jobs.

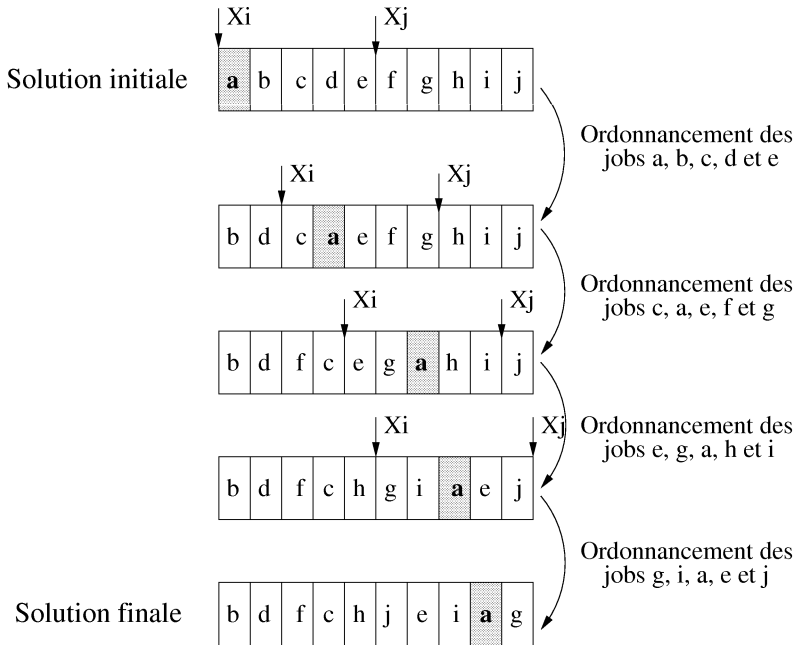


FIG. 7.13 – Évolution du génotype d'une solution, partition après partition.

Des variantes de cette approche peuvent être imaginées, afin de limiter le temps global d'exécution. En effet, il pourrait être intéressant de ne considérer qu'une partie des solutions, en choisissant, avec une méthode de *clustering* par exemple, un sous-ensemble de

solutions à améliorer. Le *clustering* pourrait permettre d'éviter d'optimiser des partitions très semblables pour différentes solutions aux génotypes proches, ou de gérer les cas où les ensembles Pareto fournis par la métaheuristique sont très grands.

7.3.3.3.2 Résultats

Nous avons à évaluer les apports de l'hybridation de $AGMA$ par TPM. Ici, nous comparons l'ensemble des solutions Pareto avant et après hybridation. Dans un premier temps, nous pouvons observer graphiquement les résultats de $SPPOBB$ (figures 7.14 et 7.15). Ces deux exemples montrent que des progrès ont été réalisés, et de manière considérable pour le problème à 200 jobs et 10 machines.

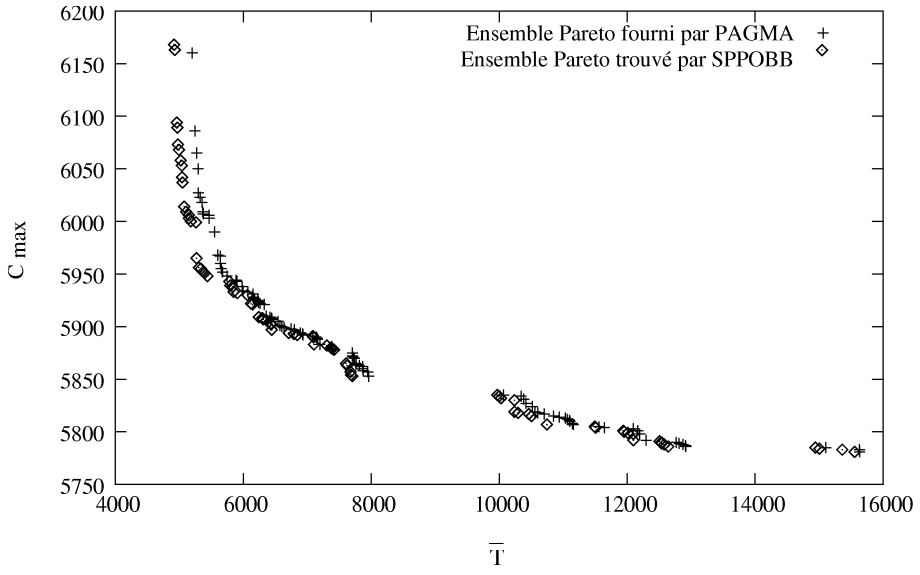


FIG. 7.14 – Apport de $SPPOBB$ sur l'instance 100 jobs/10 machines ($ta\_100\_10\_01$).

Nous avons évalué ces progrès à l'aide des métriques habituelles S et $Contribution$. Ici, nous évaluons l'apport de $SPPOBB$ sur les ensembles Pareto obtenus par $PAGMA$. Ainsi la contribution minimale $C(SPPOBB/PAGMA)$ est de 0.5, ce qui ne correspond ici à aucune amélioration apportée, puisque ici le front Pareto ne peut que être amélioré. On peut calculer le taux de solutions Pareto initiales améliorées, qui correspond à $2 * C(SPPOBB/PAGMA) - 1$. Ainsi, si l'on se réfère aux résultats obtenus (table 7.3), on en déduit une amélioration moyenne de 18.8% des solutions pour l'instance $ta\_50\_10\_01$ et de 4.8% des solutions pour l'instance $ta\_50\_20\_01$. Pour le problème $ta\_50\_10\_01$, la méthode exacte TPM nous a permis de trouver une seule solution Pareto optimale, celle ayant le C_{max} optimal. Nous avons représenté cette solution sur la figure 7.16, ainsi qu'un ensemble Pareto obtenu par $PAGMA$. L'estimation de l'allure du front de Pareto optimal nous encourage à penser que les solutions obtenues par $PAGMA$ sont proches des solutions optimales. Cette proximité peut expliquer en partie le peu d'améliorations obtenues par $SPPOBB$ sur cette instance. La figure montre également que le front de Pareto obtenu par $PAGMA$ manque encore de diversité sur les extrémités. La visualisation des améliorations

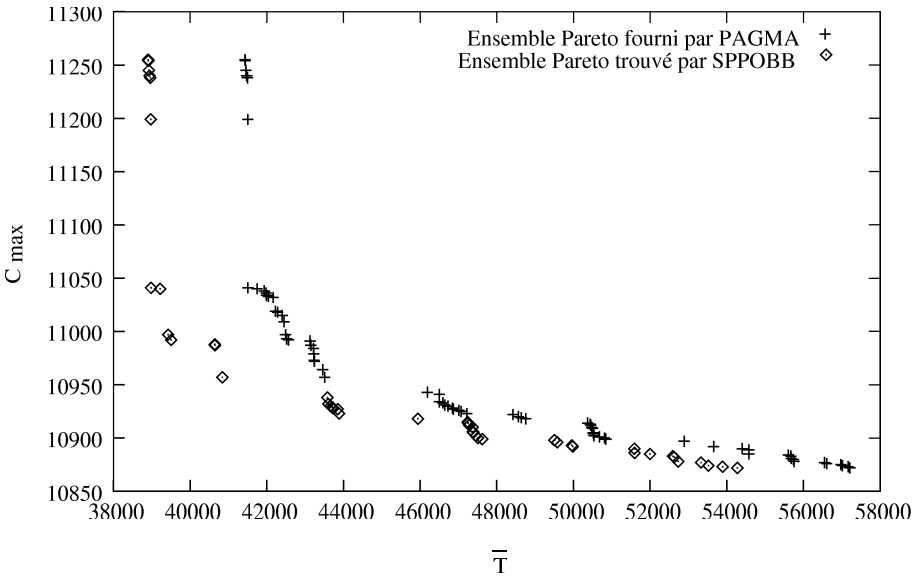


FIG. 7.15 – Apport de *SPPOBB* sur l'instance 200 jobs/10 machines (*ta*_200_10_01).

TAB. 7.3 – Évaluation de l'approche coopérative par partitionnement (métrique *Contribution*) : $C(SPPOBB/PAGMA)$.

Instance	$\mathbf{C_{Min}}$	$\mathbf{C_{Max}}$	**Moyenne**	**ET**
*ta*_50_10_01	0.54	0.63	0.594	0.026
*ta*_50_20_01	0.51	0.55	0.525	0.015
*ta*_100_10_01	0.96	1.00	0.986	0.015
*ta*_100_20_01	0.73	0.96	0.876	0.062
*ta*_200_10_01	1.00	1.00	1.000	0.000

apportées par *SPPOBB* sur ces instances par rapport à la métrique S confirme qu'il y a eu peu d'apports de la coopération (table 7.4).

Si on observe les résultats obtenus sur les instances plus grandes (à 100 ou 200 jobs), on remarque que les progrès réalisés sont beaucoup plus conséquents. Pour l'instance *ta*_100_10_01 une moyenne de 97.2% des solutions Pareto initiales ont été améliorées. Pour l'instance *ta*_100_20_01 une moyenne de 75.2% des solutions Pareto initiales ont été améliorées. Enfin, pour l'instance *ta*_200_10_01, tous les tests effectués ont permis d'améliorer la totalité des solutions Pareto initiales. Si on observe les résultats en terme d'aire de dominance, celle-ci a été augmentée d'environ 1% en moyenne pour les instances *ta*_100_10_01 et *ta*_100_20_01. L'apport le plus important est réalisé sur l'instance *ta*_200_10_01, où l'aire de dominance est augmentée de plus de 13% en moyenne, comme nous l'avions remarqué graphiquement sur la figure 7.15.

La table 7.5 montre les temps de calcul nécessaires pour réaliser l'approche *SPPOBB*

TAB. 7.4 – Évaluation de l'approche coopérative par partitionnement (métrique S) : $S(SPPOBB)/S(PAGMA)$.

Instance	S_{Min}	S_{Max}	Moyenne	ET
$ta\_50\_10\_01$	0.02%	0.46%	0.185%	0.122%
$ta\_50\_20\_01$	0.01%	0.27%	0.095%	0.095%
$ta\_100\_10\_01$	0.75%	2.10%	1.199%	0.387%
$ta\_100\_20\_01$	0.28%	1.92%	0.970%	0.412%
$ta\_200\_10\_01$	8.35%	15.57%	13.094%	1.974%

TAB. 7.5 – Temps d'exécution de $SPPOBB$.

Instance	T_{Min}	T_{Max}	Moyenne	ET
$ta\_50\_10\_01$	3h16'	5h26'	4h19'	42'
$ta\_50\_20\_01$	5h33'	6h19'	5h49'	25'
$ta\_100\_10\_01$	28h11'	52h44'	35h54'	8h05'
$ta\_100\_20\_01$	37h16'	65h02'	50h25'	7h58'
$ta\_200\_10\_01$	63h24'	122h45'	90h53'	17h16'

sur les différentes instances. Ces temps restent longs étant donnée l'utilisation d'une approche exacte. Cependant, les temps nécessaires restent réguliers, malgré l'approche exacte par $B\&B$, connue pour être très peu stable en temps de calcul, même pour des instances de même taille.

La méthode coopérative de type **HTH**($PAGMA$+TPM) présentée ici est très facilement parallélisable, étant donné qu'une résolution exacte est associée à chaque solution non-dominée découverte.

7.4 Conclusion

Dans ce chapitre, nous avons proposé plusieurs approches coopératives méta/exacte que nous avons appliquées au $BOFSP$. Pour cela nous avons, dans un premier temps, appliqué une méthode exacte bi-objectif au $BOFSP$, TPM, en y apportant quelques mécanismes spécifiques au problème étudié.

La première méthode coopérative que nous avons proposée est une approche exacte de type **LRH**. Nous avons expérimenté cette approche avec $AGMA$ et TPM. Les résultats ont montré que l'utilisation d'une métaheuristique exécutée en pré-traitement d'une méthode exacte peut apporter un gain au niveau du temps de résolution exact.

Ensuite, nous avons proposé deux approches coopératives heuristiques. Les deux approches sont de type **HTH**, chaque élement d'une population Pareto fournie par une métaheuristique étant améliorée par une recherche de type *branch & bound*. Nous avons

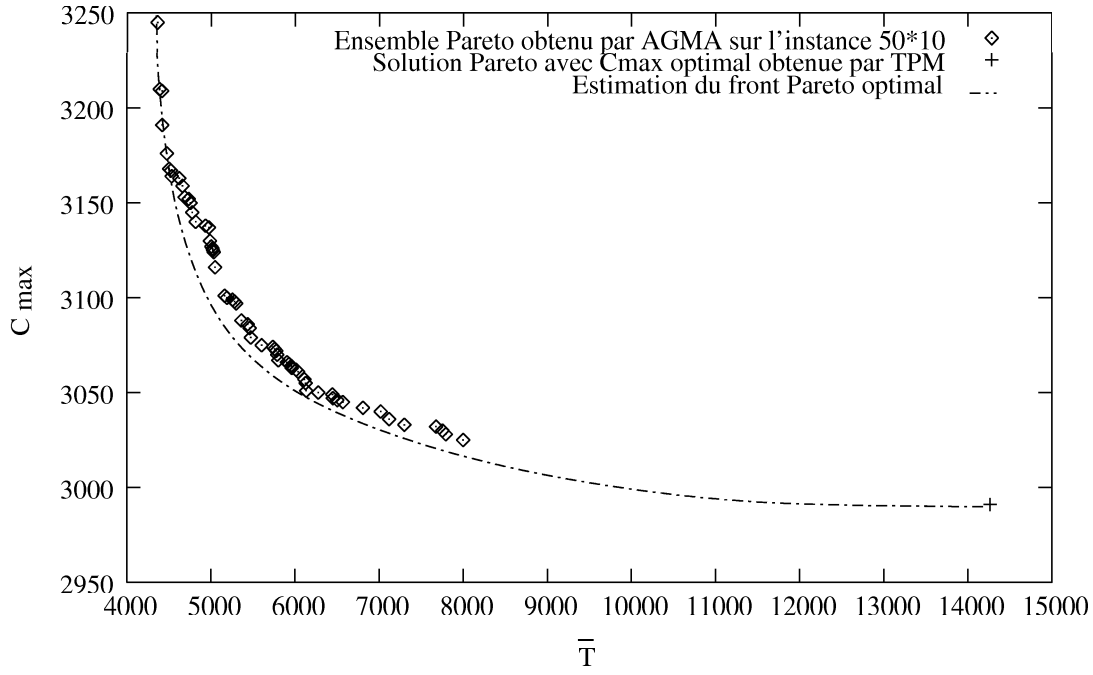


FIG. 7.16 – L'obtention de la solution optimale $lex(C_{max}, T)$ par TPM permet d'évaluer le chemin à parcourir pour atteindre l'ensemble Pareto optimal.

appliqué ces coopérations à TPM avec les résultats obtenus par $PAGMA$. La première approche, consistant à explorer un voisinage large des solutions initiales, a abouti à de très légères améliorations, réalisées en un temps excessif. La deuxième approche, consistant à optimiser les solutions Pareto région par région, a abouti à des résultats beaucoup plus intéressants, et a permis d'améliorer les meilleures solutions que nous avions obtenues jusque là.

Il semble intéressant de combiner les deux approches coopératives heuristiques proposées, afin de combiner la recherche sur voisinage avec le partitionnement. De plus, il semble intéressant, par la suite, d'évaluer les coopérations de type *low-level*, en remplaçant par exemple un opérateur génétique d'une métaheuristique par une méthode exacte. Enfin, les coopérations heuristiques présentées dans ce chapitre peuvent faire très facilement l'objet de coopérations parallèles.

Chapitre 8

Conclusions et perspectives

Dans cette thèse, nous avons abordé le domaine de la coopération entre méthodes d'optimisation dans le cadre de la résolution de problèmes multi-objectif difficiles. Afin d'évaluer les différents mécanismes de coopération, nous avons réalisé nos expérimentations sur un problème de Flow-shop bi-objectif. Deux raisons principales nous ont poussés à expérimenter nos méthodes sur un problème multi-objectif. En premier lieu, les coopérations entre méthodes d'optimisation sont actuellement assez répandues, mais beaucoup moins étudiées dans le cadre multi-objectif. De plus, les problèmes multi-objectif sont connus pour être particulièrement difficiles, ce qui pousse à utiliser des méthodes d'optimisation complexes et coopératives pour leur résolution.

Dans un premier temps, nous avons donc proposé une taxinomie des méthodes coopératives afin de recenser les différents types de coopérations rencontrés dans la littérature. Celles-ci sont principalement divisées en quatre classes résultantes de deux propriétés discriminantes. La première est le niveau de coopération, bas niveau si une fonction interne d'une méthode est remplacée par une autre méthode d'optimisation, haut niveau s'il n'y a pas de relation directe entre les fonctions internes aux méthodes. La deuxième propriété discriminante est le mode de coopération, de type relais ou *teamwork*, selon que la coopération se réalise en *pipe-line* ou de manière parallèle. Nous avons ensuite réalisé une classification des méthodes coopératives trouvées dans la littérature, en se restreignant aux coopérations méta/exacte. Nous en avons conclu que très peu de travaux traitent des coopérations méta/exacte dédiées à l'optimisation multi-objectif, ou proposent des modèles parallèles.

Nous avons ensuite présenté les problèmes d'optimisation multi-objectif. Nous avons en premier lieu présenté les différentes approches de résolution de ces problèmes. Afin d'évaluer la pertinence des résultats d'une méthode, nous avons choisi d'utiliser les métriques S et $Contribution$, pour compléter les comparaisons graphiques et mono-objectif. Dans la deuxième partie de ce chapitre, nous avons présenté le problème de Flow-shop bi-objectif $BOFSP$. Il fait partie de la classe des problèmes d'ordonnancement, constituée d'une multitude de problèmes différents. Ces problèmes se différencient par leur type, leurs contraintes, et les objectifs à optimiser. Ces derniers sont nombreux pour les différents types de problèmes d'ordonnancement, qui sont naturellement multi-objectif. Le problème

de Flow-shop que nous avons traité consiste à optimiser deux objectifs classiques pour le Flow-shop : le temps de terminaison et la somme des retards.

Dans le chapitre 4, nous avons proposé AGA, un algorithme génétique adaptatif avec approche Pareto pour la résolution du Flow-shop bi-objectif. Cet algorithme est à la base de toutes les coopérations que nous avons présentées dans cette thèse. Pour mettre au point AGA, nous nous sommes appuyés sur un algorithme de base, utilisant des opérateurs classiques efficaces pour ce problème. Ensuite, nous nous sommes concentrés sur l'élaboration de deux mécanismes permettant d'améliorer l'exploration de l'espace de recherche, en proposant des méthodes adaptatives, rendant l'algorithme efficace sur les différentes instances traitées. En premier lieu, nous avons établi une méthode permettant l'utilisation de plusieurs opérateurs de mutation de manière simultanée durant le déroulement de l'algorithme génétique. Applicable à d'autres fonctionnalités des AGs, ce mécanisme permet d'une part d'extraire automatiquement un opérateur efficace parmi une liste de candidats, et il permet d'autre part, par l'utilisation de plusieurs opérateurs simultanément, d'obtenir une exploration plus efficace de l'espace de recherche. Ensuite, nous avons proposé une méthode de diversification adaptative, en calibrant automatiquement son paramétrage, toujours dans le but de favoriser l'exploration et de rendre l'algorithme génétique adaptatif. En appliquant ces mécanismes au $BOFSP$, nous avons montré ses apports, essentiellement réalisés au niveau de la diversité des solutions obtenues.

Nous avons expérimenté ensuite différentes coopérations possibles à réaliser avec l'algorithme génétique adaptatif AGA. En premier lieu, dans les chapitre 5 et 6, nous avons étudié les coopérations entre métaheuristiques, et leurs applications pour le Flow-shop bi-objectif. Le but était de faire coopérer avec AGA des méthodes dites d'intensification. L'approche multi-objectif nécessitant la considération d'un ensemble de solutions non-dominées, la plupart des coopérations que nous avons proposées sont en mode *teamwork* (voir l'annexe 1). Ces coopérations sont effectuées entre AGA et diverses méthodes d'intensification : la recherche locale Pareto PLS, l'algorithme de *path relinking* multi-objectif $MOPR$, et la méthode exacte bi-objectif TPM.

Nous avons, dans un premier temps, présenté PLS, une recherche locale Pareto sur laquelle nous avons basé toutes nos coopérations méta/méta. La première voie de coopération que nous avons explorée dans le chapitre 5 est très répandue puisqu'il s'agit des algorithmes de type mimétique. Nous avons donc proposé l'algorithme mimétique AMA dans un premier temps, puis $AMA + AGA$ et $AGMA$, dérivés de AMA. Ces deux algorithmes ont montré qu'il s'avérait intéressant de garder des périodes d'exécution de AGA entre les différentes générations de recherche mimétique AMA. La comparaison de $AMA + AGA$ et $AGMA$ a montré de surcroît l'intérêt de la mise en place d'un mécanisme de transition adaptative entre les méthodes d'optimisation. Dans la dernière partie de ce chapitre, nous avons testé la coopération parallèle $PAGMA$, ainsi que des modèles *peer to peer*, afin notamment de s'attaquer aux instances de Flow-shop bi-objectif les plus grandes.

Dans le chapitre 6, nous avons proposé une coopération originale entre l'algorithme génétique AGA et $PRMO$, un algorithme de *path relinking* multi-objectif. Dans le cadre de la conception de $PRMO$, nous avons proposé un opérateur de mesure de distance entre solutions du Flow-shop, correspondant au voisinage de l'opérateur de mutation *Shift*. Nous avons aussi défini des méthodes multi-objectif permettant d'établir les différentes

fonctionnalités d'un algorithme de *path relinking* multi-objectif. Puis nous avons effectué des expérimentations qui ont montré l'intérêt de ce type d'approche.

Dans le dernier chapitre, nous avons proposé plusieurs approches coopératives entre métaheuristiques et méthodes exactes. Pour cela, nous avons dans un premier temps appliqué une méthode exacte bi-objectif au Flow-shop, la méthode à deux-phases (TPM), en y apportant quelques mécanismes spécifiques au problème étudié. Puis nous avons proposé trois coopérations méta/exacte. La première méthode coopérative que nous avons définie est une approche exacte de type **LRH**. Les résultats ont montré que l'utilisation d'une métaheuristique exécutée en pré-traitement d'une méthode exacte peut apporter un gain au niveau du temps de résolution exact. Ensuite, nous avons proposé deux approches coopératives heuristiques. Les deux approches sont de type **HTH**, chaque élément d'une population Pareto fournie par une métaheuristique étant améliorée par une recherche de type *branch and bound*. Nous avons appliqué ces coopérations à la méthode exacte TPM avec les résultats obtenus par l'algorithme parallèle $PAGMA$. La première approche, consistant à explorer un voisinage large des solutions initiales a abouti à de très légères améliorations, réalisées en un temps excessif. La deuxième approche, consistant à optimiser les solutions Pareto région par région, a abouti à des résultats beaucoup plus intéressants, et a permis d'améliorer les meilleures solutions que nous avions obtenues jusque-là.

Au fil des différentes coopérations réalisées, nous avons montré l'apport de chaque mécanisme de coopération présenté. Cependant, il serait prétentieux d'affirmer que chaque mécanisme est toujours utile, quelle que soit l'instance traitée, ou quel que soit le problème traité. En particulier, l'efficacité des méthodes d'optimisation et des coopérations présentées dans cette thèse dépend fortement de la taille du problème traité. Ainsi, en observant les expérimentations réalisées sur les instances de Taillard étendues au cas bi-objectif, on remarque par exemple que pour l'instance la plus facile ($ta\_20\_5\_01$), la méthode exacte TPM reste la méthode la plus efficace pour trouver le front optimal rapidement. Pour les autres instances à 20 et 50 jobs, les résultats les plus probants sont obtenus par les algorithmes génétiques coopératifs $AGMA$ et $PAGMA$, la coopération méta/exacte avec TPM étant trop lente et peu efficace. Par contre, sur les plus grandes instances, la coopération méta/exacte $SPPOBB$ est efficace. D'une manière générale, plus les instances traitées sont grandes, plus les approches coopératives sont efficaces au regard des méthodes plus simples.

Différentes perspectives s'ouvrent sur ce travail. En premier lieu, les méthodes d'optimisation présentées dans cette thèse peuvent être améliorées, indépendamment des stratégies de coopération. Nous avons défini des mécanismes adaptatifs pour mettre au point AGA. Il semble intéressant de mettre au point d'autres mécanismes du même type pour d'autres fonctionnalités des AGs, comme les opérateurs de croisement, de sélection,... Il peut également être utile de mettre au point une recherche locale PLS partielle, qui permettrait de pouvoir réaliser des recherches locales Pareto sur des problèmes à plus de 200 jobs, ce qui n'est pas possible pour le moment, étant donné la taille de l'espace à explorer. Enfin, nous avons présenté des mécanismes de base pour réaliser un algorithme de *path relinking* multi-objectif. Ces mécanismes ont permis de tester l'efficacité de l'algorithme de *path relinking* $PRMO$ dans le cadre d'une coopération avec PLS et AGA. On peut imaginer des mécanismes plus complexes d'exploration des chemins dans le cadre multi-objectif, en

explorant par exemple un nombre plus grand de chemins possibles et non pas un seul.

D'autres perspectives s'ouvrent au niveau des mécanismes de coopération. Dans un premier temps, on peut imaginer utiliser des mécanismes plus évolués pour établir un algorithme de *path relinking* multi-objectif coopératif encore plus performant, en s'inspirant du mécanisme de coopération établi dans $AGMA$. Dans le cadre des coopérations méta/exacte, il semble intéressant de combiner les deux approches coopératives méta/exacte heuristiques proposées, afin de combiner la recherche sur voisinage large avec le partitionnement. De plus, il serait intéressant d'évaluer les coopérations de type *low-level*, en remplaçant par exemple un opérateur génétique d'une métaheuristique par une méthode exacte en s'inspirant des algorithmes mimétiques. Enfin, les coopérations méta/exacte présentées peuvent très facilement faire l'objet de coopérations parallèles.

Afin de valider les différents mécanismes présentés dans cette thèse, il serait révélateur de les tester sur d'autres problèmes que le $BOFSP$. En effet, ces mécanismes ont été généralement définis de manière générique, de telle sorte qu'ils puissent être facilement appliqués à d'autres problèmes. Il serait donc intéressant de tester aussi bien les mécanismes adaptatifs de l'algorithme génétique AGA que les mécanismes mis en oeuvre pour réaliser $PRMO$ sur d'autres problèmes multi-objectif. Les différentes coopérations réalisées pourraient être testées sur d'autres problèmes multi-objectif, voire des problèmes mono-objectif complexes. Pour ces expérimentations, nous préconisons de privilégier deux types de problèmes :

- les problèmes à plus de deux objectifs, afin d'évaluer la qualité des mécanismes multi-objectif présentés dans cette thèse. En effet, nous nous sommes restreints pour le moment au seul cas bi-objectif. Des adaptations seront sûrement nécessaires pour traiter convenablement les cas à plus de deux objectifs ;
- les problèmes multi-objectif réels, permettant de traiter un problème de la vie courante. Ceci permettrait de valider l'apport pour des problèmes cruciaux du monde réél, souvent très difficiles, car il sont souvent très grands et soumis à de nombreuses contraintes.

Une perspective à long terme est de pouvoir définir un ensemble de méthodes d'optimisation coopérant entre elles, et de pouvoir définir quelles coopérations réaliser, en fonction de la taille de l'instance, ou en fonction de l'évolution de l'algorithme. Ce type de coopération est déjà en partie réalisé dans l'algorithme génétique coopératif $AGMA$, mais pourrait être défini de manière plus générale.

L'optimisation approchée, du point de vue des entreprises concernées par les problèmes multi-objectif, consiste en plusieurs buts. En premier lieu, les temps de calcul doivent être adaptés aux ressources disponibles pour la réaliser. Les solutions proposées doivent être de bonne qualité en terme de convergence, et offrir de bons compromis entre les différents objectifs. Ces objectifs de temps de calcul et de qualité des résultats sont en partie obtenus par l'utilisation d'approches heuristiques, utilisées de manière coopérative pour allier intensification et diversification de la recherche. Mais, d'une manière générale, il est demandé au concepteur que l'algorithme proposé soit robuste, de telle sorte qu'il garantisse d'obtenir à chaque fois des résultats satisfaisants, malgré l'utilisation de paramètres qui font fluctuer ces résultats. Cette fluctuation doit être la plus réduite possible. Nous avons envisagé dans

cette thèse des coopérations parallèles de type *modèle en îles*, qui ont montré une grande faculté à rendre les algorithmes robustes. Ce type de coopération semble extrêmement prometteur, surtout avec l'avènement du calcul massivement parallèle. Il semble qu'il y ait de belles perspectives dans ce domaine, où de nombreux schémas de coopération sont encore à explorer.

Annexe 1

Notations des méthodes présentées dans cette thèse

Dans cette annexe, nous rappelons les différentes métaheuristiques et métaheuristiques coopératives mises en oeuvre dans cette thèse (ou méthodes exactes). Pour chaque méthode, nous rappelons son nom complet, la section y faisant référence et dans le cas d'approches coopératives, le type de la coopération, selon la classification proposée dans le chapitre 2.

- Méthodes de base :
  - **AGA** : Algorithme Génétique Adaptatif (*Adaptive Genetic Algorithm* - voir chapitre 4).
  - **PLS** : Recherche Locale Pareto (*Pareto Local Search* - voir section 5.2.1).
  - **TPM** : Méthode Deux-Phases (*Two Phases Method* - voir section 7.2).
- Méthodes coopératives :
  - **AMA** : Algorithme Mimétique Adaptatif (*Adaptive Memetic Algorithm*). Coopération méta/méta de type **LTH(AGA(PLS))** (Section 5.2.2).
  - **AGA+AMA** : Algorithme Génétique Adaptatif + Algorithme Mimétique Adaptatif (*Adaptive Genetic Algorithm + Adaptive Memetic Algorithm*). Coopération méta/méta de type **HTH(AGA+AMA)** (Section 5.2.3).
  - **AGMA** : Algorithme Génétique/Mimétique Adaptatif (*Adaptive Genetic/Memetic Algorithm*). Coopération méta/méta de type **HTH(AGA+AMA)** (Section 5.2.3).
  - **PAGMA** : Algorithme Génétique/Mimétique Adaptatif Parallèle (*Parallel Adaptive Genetic/Memetic Algorithm*). Coopération méta/méta de type **HTH(AGMA +AGMA)*** (Section 5.3).
  - **MOPR** : Path Relinking Multi-Objectif (*MultiObjective Path Relinking*). Coopération méta/méta de type **LTH(PR(PLS))** (Section 6.2).
  - **AGA+MOPR** : Algorithme Génétique Adaptatif + Path Relinking Multi-Objectif (*Adaptive Genetic Algorithm + MultiObjective Path Relinking*). Coopération méta/méta de type **HTH(AGA,MOPR)** (Section 6.3).
  - **SPPOBB** : Méthode de Post Optimisation par *Branch and Bound* sur Partitions (*Simple Partitionning Post Optimization Branch and Bound*). Coopération méta/exacte de type **HTH(PAGMA,TPM)** (Section 7.3.3).

Annexe 2

Meilleurs ensembles Pareto trouvés par les différentes méthodes d'optimisation présentées dans cette thèse

Dans cette annexe, nous avons répertorié les meilleurs ensembles Pareto (éventuellement optimaux), trouvés par les différentes méthodes d'optimisation présentées dans cette thèse. Pour chaque instance, il est précisé quelle approche a permis de trouver cet ensemble. Si plusieurs approches ont permis de trouver le même ensemble Pareto, l'approche la plus rapide en moyenne a été retenue. Ces résultats sont disponibles et remis à jour régulièrement sur le web à l'adresse : `http ://www.lifl.fr/ basseur/res05.html`.

$ta\_20\_5\_01$ (TPM)

$\bar{T}$	C_{max}	$\bar{T}$	C_{max}
554	1278	453	1297
515	1296	452	1339

$ta\_20\_5\_02$ $(AGMA)$

$\bar{T}$	C_{max}	$\bar{T}$	C_{max}	$\bar{T}$	C_{max}
896	1359	754	1363	603	1367
892	1361	631	1366	469	1368

$ta\_20\_10\_01$ $(AGMA)$

$\bar{T}$	C_{max}	$\bar{T}$	C_{max}	$\bar{T}$	C_{max}
3613	1582	2255	1637	1612	1702
3065	1583	2213	1645	1607	1704
3064	1589	2147	1646	1595	1705
2820	1590	2143	1649	1546	1706
2814	1592	2081	1650	1509	1715
2765	1593	2068	1654	1501	1716
2670	1594	2024	1659	1415	1722
2626	1606	1903	1667	1395	1726
2614	1610	1749	1674	1391	1731
2460	1611	1745	1675	1323	1735
2351	1617	1744	1690	1238	1752
2341	1621	1683	1693	1229	1755
2327	1630	1667	1698	1226	1761
2258	1636	1651	1700	1224	1773

$ta\_20\_10\_02$ $(AGMA)$

$\bar{T}$	C_{max}	$\bar{T}$	C_{max}	$\bar{T}$	C_{max}
3438	1659	2016	1694	1564	1743
2699	1660	1850	1696	1557	1744
2694	1668	1842	1700	1528	1745
2500	1669	1771	1708	1488	1750
2477	1671	1765	1710	1453	1756
2348	1672	1717	1719	1413	1760
2200	1677	1656	1721	1399	1763
2174	1683	1635	1723	1360	1776
2160	1684	1613	1724	1341	1778
2072	1685	1597	1725	1275	1795
2037	1687				

$ta\_20\_20\_01$ $(AGMA)$

$\bar{T}$	C_{max}	$\bar{T}$	C_{max}	$\bar{T}$	C_{max}
3762	2297	2202	2336	1563	2408
3598	2298	2172	2355	1548	2411
3561	2299	2136	2358	1507	2418
3089	2300	2132	2359	1495	2419
3063	2307	2071	2362	1441	2420
3019	2308	2052	2363	1438	2425
2924	2310	2007	2364	1346	2427
2918	2311	1922	2367	1273	2450
2807	2312	1829	2377	1243	2454
2618	2315	1824	2381	1158	2468
2449	2316	1682	2383	1062	2476
2422	2330	1618	2393	1057	2497
2415	2334	1607	2407	1031	2509
2349	2335				

$ta\_50\_5\_01$ $(AGMA)$

$\bar{T}$	C_{max}	$\bar{T}$	C_{max}	$\bar{T}$	C_{max}
4461	2724	3496	2745	3275	2752
3801	2729	3455	2746	3238	2807
3783	2743	3438	2751	3231	2840
3779	2744				

$ta\_50\_10\_01$ $(PAGMA + SPPOBB)$

$\bar{T}$	C_{max}	$\bar{T}$	C_{max}	$\bar{T}$	C_{max}
7999	3025	5929	3065	4964	3131
7792	3028	5903	3066	4920	3132
7576	3030	5800	3067	4918	3135
7569	3032	5746	3070	4818	3140
7301	3033	5737	3074	4779	3145
7058	3034	5604	3075	4778	3146
7017	3039	5475	3079	4688	3147
6977	3040	5459	3084	4687	3153
6808	3042	5436	3086	4662	3159
6569	3045	5362	3088	4593	3163
6489	3046	5301	3097	4533	3164
6396	3047	5282	3098	4524	3167
6276	3050	5256	3099	4504	3168
6132	3051	5190	3100	4475	3176
6126	3055	5164	3101	4469	3178
6112	3057	5050	3116	4422	3191
6036	3061	5035	3124	4420	3209
6014	3062	5023	3125	4392	3210
5961	3063	5019	3126	4359	3245
5956	3064	5004	3127		

$ta\_50\_20\_01$ $(PAGMA + SPPOBB)$

$\bar{T}$	C_{max}	$\bar{T}$	C_{max}	$\bar{T}$	C_{max}
13720	3899	10349	3966	7963	4077
13514	3900	10209	3968	7915	4084
13455	3901	10122	3971	7853	4089
13142	3902	10103	3972	7852	4093
12550	3903	10059	3973	7846	4096
12339	3906	10008	3975	7777	4097
12250	3909	9792	3976	7737	4102
12021	3911	9624	3978	7687	4105
11906	3913	9410	3984	7604	4109
11794	3915	9370	3994	7575	4113
11780	3920	9220	3995	7530	4116
11748	3921	9029	3998	7506	4123
11733	3922	8947	4001	7482	4127
11641	3926	8944	4002	7470	4136
11562	3928	8917	4009	7461	4137
11438	3930	8904	4012	7436	4142
11054	3932	8886	4014	7410	4145
11010	3937	8802	4017	7402	4149
10995	3940	8788	4020	7390	4165
10947	3941	8617	4022	7389	4166
10942	3942	8478	4025	7378	4169
10902	3947	8393	4038	7334	4173
10663	3948	8355	4048	7307	4182
10658	3953	8330	4050	7275	4186
10632	3954	8316	4051	7265	4199
10569	3955	8235	4054	7244	4200
10565	3957	8198	4058	7213	4213
10481	3958	8046	4063	7211	4245
10431	3962	8045	4064	7210	4248
10356	3963	8028	4067	7204	4268
10353	3964	8018	4068		
10351	3965	7984	4072		

$ta\_100\_5\_01$ $(PAGMA)$

$\bar{T}$	C_{max}	$\bar{T}$	C_{max}	$\bar{T}$	C_{max}
2747	5493	253	5523	11	5541
2513	5495	204	5526	0	5544
1828	5522	63	5527		

$ta\_100\_10\_01$ $(PAGMA + SPPOBB)$

$\bar{T}$	C_{max}	$\bar{T}$	C_{max}	$\bar{T}$	C_{max}
13649	5781	9499	5842	5316	5931
13512	5783	9477	5845	5310	5932
13462	5784	9468	5846	5305	5933
12585	5785	7704	5853	5274	5934
12314	5786	7682	5854	5268	5935
11693	5788	7675	5857	5240	5939
11682	5789	7620	5863	5237	5947
11678	5790	7608	5865	5226	5948
11670	5792	7490	5870	5223	5949
11650	5793	7446	5871	5220	5951
11482	5797	7417	5875	5191	5953
11469	5800	7412	5876	5181	5955
11445	5801	7050	5878	5171	5957
11073	5802	7028	5880	5166	5958
10902	5803	6861	5881	5108	5973
10804	5804	6714	5883	5102	5974
10756	5806	6502	5886	5097	5976
10611	5807	6333	5887	5067	5978
10495	5808	6264	5892	5060	5980
10431	5809	6218	5893	5018	5982
10376	5810	6132	5896	4960	5983
10308	5811	6062	5897	4937	5985
10307	5812	6038	5900	4815	5987
10231	5813	6019	5903	4810	5991
10186	5814	5984	5904	4763	5993
10159	5815	5935	5907	4758	6000
10076	5816	5864	5910	4756	6003
10008	5818	5791	5913	4748	6011
9998	5819	5752	5914	4742	6012
9973	5822	5747	5915	4725	6016
9837	5823	5706	5918	4718	6017
9726	5825	5688	5919	4716	6050
9717	5831	5537	5920	4705	6071
9611	5832	5525	5921	4665	6086
9605	5835	5486	5923	4658	6145
9596	5836	5422	5926	4633	6160
9508	5837	5416	5929		
9505	5838	5380	5930		

$ta\_100\_20\_01$ $(PAGMA + SPPOBB)$

$\bar{T}$	C_{max}	$\bar{T}$	C_{max}	$\bar{T}$	C_{max}	$\bar{T}$	C_{max}
43163	6349	27809	6444	20246	6564	15788	6680
43021	6350	27596	6445	19459	6565	15609	6683
42470	6351	27582	6446	19442	6567	15608	6684
42445	6352	27534	6448	19388	6573	15494	6690
42411	6354	27328	6449	19375	6575	15442	6692
40523	6358	27312	6454	19362	6577	15389	6697
40470	6360	27271	6457	19317	6578	15344	6699
40424	6363	27175	6458	19134	6580	15313	6706
40047	6364	26624	6460	19130	6582	15144	6707
40038	6365	26230	6465	19102	6584	14908	6713
39913	6366	25949	6470	19063	6585	14470	6716
39270	6367	25880	6471	19032	6591	14469	6717
38479	6368	25859	6474	18697	6592	14458	6728
37743	6372	25742	6476	18559	6593	14420	6729
37671	6374	25596	6477	18537	6596	14041	6731
37057	6377	25491	6478	18259	6598	14029	6733
35057	6379	25327	6481	18204	6600	13866	6738
34575	6382	24833	6483	17716	6603	13856	6758
34302	6385	24565	6486	17633	6605	13843	6759
34155	6390	24337	6493	17626	6606	13808	6772
33197	6393	24331	6494	17557	6607	13773	6777
32830	6395	23915	6495	17459	6608	13700	6778
32815	6396	23174	6501	17326	6609	13696	6779
32486	6398	22809	6502	17322	6610	13588	6785
32484	6400	22719	6503	17221	6619	13567	6787
32347	6401	22663	6504	17211	6622	13560	6795
32165	6402	22583	6505	17149	6626	13493	6798
31792	6405	22572	6506	17126	6628	13483	6800
31774	6407	21995	6507	16825	6632	13408	6801
31716	6408	21886	6508	16763	6635	13370	6804
31525	6409	21883	6524	16668	6636	13357	6811
31469	6411	21761	6530	16635	6638	13332	6812
31370	6412	21741	6531	16578	6649	13289	6820
31166	6414	21573	6532	16569	6653	13275	6835
31122	6416	21365	6534	16448	6654	13253	6845
31031	6418	21225	6535	16384	6657	13224	6854
30652	6419	21115	6538	16366	6662	13203	6867
30602	6423	21084	6542	16341	6663	13196	6913
30417	6424	21035	6543	16289	6664	13184	6919
29520	6425	20791	6546	16248	6665	13160	6929
29457	6426	20727	6548	16098	6666	13150	6946
28730	6429	20601	6549	16076	6668	13129	6959
28642	6432	20519	6553	16070	6671	13119	6988
28570	6434	20474	6554	16067	6673	13113	6998
28493	6436	20403	6556	16062	6675		
27929	6439	20340	6561	16059	6676		

$ta\_200\_10\_01$ $(PAGMA + SPPOBB)$

$\bar{T}$	C_{max}	$\bar{T}$	C_{max}	$\bar{T}$	C_{max}
54278	10872	46901	10915	40448	10963
53892	10873	45637	10918	40413	10970
53521	10874	45213	10919	40370	10974
53329	10877	43884	10923	39503	10992
52732	10878	43848	10927	39426	10997
52631	10882	43355	10928	39423	10999
52594	10883	43346	10932	39369	11000
51999	10885	43318	10933	39309	11004
51591	10886	43265	10934	39263	11008
51587	10890	43262	10935	39087	11016
49982	10892	43092	10937	39082	11018
49960	10893	43033	10939	38981	11019
49580	10896	42787	10941	38953	11020
47949	10897	42740	10944	38918	11021
47619	10899	42647	10945	38897	11023
47508	10900	42605	10948	38835	11028
47456	10902	42014	10949	38793	11030
47385	10905	41498	10950	37744	11041
47376	10906	41280	10951	37726	11051
47370	10910	40816	10957	37628	11204
46994	10911	40760	10960		
46964	10913	40525	10962		

Références bibliographiques

[AAC95] F. Abbatista, N. Abbatista, and L. Caponetti. An evolutionary and cooperative agent model for optimization. In *IEEE International Conference on Evolutionary Computation ICEC'95*, pages 668–671, Perth, Australia, December 1995.

[ABB+98] P. Augerat, J. M. Belenguer, E. Benavent, A. Corber'n, and D. Naddef. Separating capacity constraints in the CVRP using tabu search. *European Journal of Operational Research*, 106(2-3) :546–557, April 1998.

[ABCC99] D. Applegate, R. Bixby, V. Chvátal, and W. Cook. Finding tours in the TSP. Technical Report 99885, Forschungsinstitut für Diskrete Mathematik, Universität Bonn, Germany, 1999.

[ABR03] R. M. Aiex, S. Binato, and M. G. C. Resende. Parallel GRASP with path-relinking for job shop scheduling. *Parallel Computing*, 29 :393–430, 2003.

[AC04] V. A. Armentano and J. E. Claudio. An application of a multi-objective tabu search algorithm to a bicriteria flowshop problem. *Journal of Heuristics*, 10(5) :463–481, September 2004.

[ACK+02] D. P. Anderson, J. Cobb, E. Korpela, M. Lepofsky, and D. Werthimer. SETI@home : An Experiment in Public-Resource Computing. *Communications of the ACM*, 45(11) :56–61, November 2002.

[AEOP02] R. K. Ahuja, O. Ergun, J. B. Orlin, and A. P. Punnen. A survey of very large-scale neighborhood search techniques. *Discrete Discrete Applied Mathematics*, 123(1-3) :75–102, 2002.

[All92] R. Allenson. Genetic algorithms with gender for multifunction optimisation. Technical Report EPCC-SS92-01, Edimburg Parallel Computing Center, Edinburg, Scotland, 1992.

[AN79] Y. P. Aneja and K. P. K. Nair. Bicriteria transportation problem. *Management Science*, 25 :73–78, 1979.

[ANS96] A. Atamturk, G.L. Nemhauser, and M.W.P. Savelsbergh. A combined lagrangian, linear programming and implication heuristic for large-scale set partitioning problems. *Journal of Heuristics*, 1 :247–259, 1996.

[AOBH96] T. Arslan, E. Ozdemir, M. S. Bright, and D. H. Horrock. Genetic synthesis techniques for low-power digital signal processing circuits. In *IEEE colloquium on digital synthesis*, pages 71–75, London, UK, 1996. IEEE service center.

[Bac99] V. Bachelet. *Métaheuristiques Parallèles Hybrides : Application au Problème d'Affectation Quadratique*. PhD thesis, University of Lille, France, December 1999.

[BCK01] E.K. Burke, P.I. Cowling, and R. Keuthen. Effective local and guided variable neighbourhood search methods for the asymmetric travelling salesman problem. In E. J. W. Boers, S. Cagnoni, J. Gottlieb, E. Hart, P. L. Lanzi, G. Raidl, R. E. Smith, and H. Tijink, editors, *Applications of Evolutionary Computing. EvoWorkshops2001 : EvoCOP, EvoFlight, EvoIASP, EvoLearn, and EvoSTIM. Proceedings*, volume 2037 of *Lecture Note in Computer Science*, pages 203–212, Como, Italy, April 2001. Springer-Verlag.

[BD01] J.A. Bennell and K.A. Dowsland. Hybridising tabu search with optimisation techniques for irregular stock cutting. *Management Science*, 47(8) :1160–1172, 2001.

[BGS00] K. Büdenbender, T. Grünert, and H-J. Sebastian. A hybrid tabu search/branch-and-bound algorithm for the direct flight network design problem. *Transportation Science*, 34(4) :364–380, November 2000.

[BH03] V. Barichard and J. K. Hao. A population and interval constraint propagation algorithm. In C. A. Coello Coello, A. H. Aguirre, and E. Zitzler, editors, *Evolutionary Multi-Criterion Optimization, EMO'2003*, volume 2632 of *Lecture Notes in Computer Science*, pages 88–101, Faro, Portugal, 2003. Springer-Verlag.

[BHS89] D.E. Brown, C.L. Huntley, and A.R. Spillane. A parallel genetic heuristic for the quadratic assignment problem. In *Third International Conference on Genetic Algorithms ICGA'89*, pages 406–415. Morgan Kauffmann, San Mateo, USA, July 1989.

[BJLR78] J. K. Lenstra B. J. Lageweg and A. H. G. Rinnooy Kan. A general bounding scheme for the permutation flow-shop problem. *Operations Research*, 26(1) :53–67, 1978.

[BJN+98] C. Barnhart, E. L. Johnson, G. L. Nemhauser, M. W. P. Savelsbergh, and P. H. Vance. Branch-and-price : column generation for solving huge integer programs. *Operations Research*, 46 :316–329, 1998.

[BJT05] M. Basseur, L. Jourdan, and E-G. Talbi. A taxonomy of cooperative approaches between exact and metaheuristics optimization methods. *ACM Computing Surveys*, 2005. submitted.

[BKY02] J.F. Bard, G. Kontoravdis, and G. Yu. A branch-and-cut procedure for the vehicle routing problem with time windows. *Transportation Science*, 36 :250–269, 2002.

[BLDT04] M. Basseur, J. Lemesre, C. Dhaenens, and E-G. Talbi. Cooperation between branch and bound and evolutionary approaches to solve a biobjective flow shop problem. In *Workshop on Efficient and Experimental Algorithms (WEA'04)*, volume 3059, pages 72–86. Springer-Verlag, 2004.

[BS00] E.K. Burke and A.J. Smith. Hybrid evolutionary techniques for the maintenance scheduling problem. *IEEE Transactions on Power System*, 15(1) :122–128, 2000.

[BST02] M. Basseur, F. Seynhaeve, and E-G. Talbi. Design of Multi-objective Evolutionary Algorithms : Application to the Flow-shop Scheduling Problem. In *Congress on Evolutionary Computation CEC'02*, pages 1151–1156, Honolulu, Hawaii, USA, May 2002.

[BST03] M. Basseur, F. Seynhaeve, and E-G. Talbi. Adaptive mechanisms for multi-objective evolutionary algorithms. In *Congress on Engineering in System Application CESA'03*, pages 72–86, Lille, France, July 2003.

[BST05a] M. Basseur, F. Seynhaeve, and E-G. Talbi. An adaptive genetic/memetic algorithm for multiobjective problems. *IEEE Transaction on Evolutionary Computation*, 2005. submitted.

[BST05b] M. Basseur, F. Seynhaeve, and E-G. Talbi. A hybrid multiobjective path relinking algorithm. *Journal of Heuristics*, 2005. submitted.

[BST05c] M. Basseur, F. Seynhaeve, and E-G. Talbi. Path relinking in pareto multi-objective genetic algorithms. In C. A. Coello Coello, A. H. Aguirre, and E. Zitzler, editors, *Evolutionary Multi-Criterion Optimization, EMO'2005*, volume 3410 of *Lecture Notes in Computer Science*, pages 120–134, Guanajuato, Mexico, March 2005. Springer-Verlag.

[BST05d] M. Basseur, F. Seynhaeve, and E-G. Talbi. *Real-world Multi-objective System Engineering*, chapter 6 : A Cooperative Metaheuristic Applied to Multi-Objective Flow-Shop Scheduling Problem. Nova Science, 2005.

[BST05e] M. Basseur, F. Seynhaeve, and E-G. Talbi. Solving a biobjective flow-shop problem with cooperative methods. *SIAM Journal on Optimization*, 2005. submitted.

[BV04] R. Bent and P. Van Hentenryck. A two-stage hybrid local search for the vehicle routing problem with time windows. *Transportation Science*, 38(4) :515–530, November 2004.

[BW97] P. J. Bentley and J. P. Wakefield. Finding acceptable solutions in the pareto-optimal range using multiobjective genetic algorithms. *Soft Computing in Engineering Design and Manufacturing*, 5 :231–240, 1997.

[CANT95] C. Cotta, J. F. Aldana, A. J. Nebro, and J.M. Troya. Hybridizing genetic algorithms with branch and bound techniques for the resolution of the TSP. In D.W. Pearson, N.C. Steele, and R.F. Albrecht, editors, *Artificial Neural Nets and Genetic Algorithms 2*, pages 277–280. Springer-Verlag, 1995.

[CCJ+97] D. Clements, J. Crawford, D. Joslin, G. Nemhauser, M. Puttlitz, and M. Savelsbergh. Heuristic Optimization : A hybrid AI/OR approach. In *Workshop on Industrial Constraint-Directed Scheduling*, Schloss Hagenberg, Austria, 1997.

[CDG99] D. Corne, M. Dorigo, and F. Glover, editors. *New Ideas in Optimization*. McGraw-Hill, 1999.

[CDP02] A. Chabrier, E. Danna, and C. Le Pape. Coopération entre génération de colonnes sans cycle et recherche locale appliquée au routage de véhicules. In *Journées nationales sur la résolution pratique de problèmes NP-Complets (JNPC'2001)*, 2002.

[CHMR87] J. Cohoon, S. Hedge, W. Martin, and D. Richards. Punctuated equilibria : A parallel genetic algorithm. In *Second International Conference on Genetic Algorithms ICGA'87*, pages 148–154, Cambridge, 1987. MIT.

[CJ98] P. Czyzak and A. Jaszkiewicz. Pareto simulated annealing - a metaheuristic technique for multiple-objective combinatorial optimization. *Journal of Multi-Criteria Decision Analysis*, 7, 1998.

[CJKO01] D. W. Corne, N. R. Jerram, J. D. Knowles, and M. J. Oates. PESA-II : Region-based selection in evolutionary multiobjective optimization. In L. Spector, E. D. Goodman, A. Wu, W. Langdon, H.-M. Voigt, M. Gen, S. Sen, M. Dorigo, S. Pezeshk, M. H. Garzon, and E. Burke, editors, *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO'2001)*, pages 283–290. Morgan Kaufmann Publisher, 2001.

[CJP83] H. Crowder, E. L. Johnson, and M. W. Padberg. Solving large-scale zero-one linear programming problems. *Operations Research*, 31 :803–834, 1983.

[CKO00] D. W. Corne, J. D. Knowles, and M. J. Oates. The pareto envelope-based selection algorithm for multiobjective optimization. In M. Schoenauer, K. Deb, G. Rudolph, X. Yao, E. Lutton, J. J. Merelo, and H.-P. Schwefel, editors, *Proceedings of Parallel Problem Solving from Nature VI*, volume 1917 of *Lecture Notes in Computer Science*, pages 839–848. Springer-Verlag, 2000.

[CLR90] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to algorithms*, chapter 15, pages 350–355. The MIT Press, Cambridge, Massachuchetts, 1990.

[CMM67] R. Conway, W. Maxwell, and L. Miller. *Theory of scheduling*. Addison-Wesley, 1967.

[CMMT97] V-D. Cung, T. Mautor, P. Michelon, and A. Tavares. A scatter search based approach for the quadratic assignment problem. In *IEEE International Conference on Evolutionary Computation ICEC'97*, pages 165–170, Indianapolis, USA, April 1997.

[CN00] J. Carlier and E. Neron. An exact method for solving the multi-processor Flow-Shop. *RAIRO - Recherche Opérationnelle*, 34(1) :1–25, 2000.

[Coe98] C. A. Coello Coello. Using a min-max method to solve multiobjective optimization problems with genetic algorithms. In *IBERAMIA'98*, volume 1993 of *LNCS*, pages 303–314. Springer-Verlag, 1998.

[CPv02] R. K. Congram, C. N. Potts, and 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.

[CS03] Y. Collette and P. Siarry. *Multiobjective optimization*. Springer, August 2003.

[CS05] R. Chelouah and P. Siarry. A hybrid method combining continuous tabu search and nelder-mead simplex algorithms for the global optimization of multiminima functions. *European Journal of Operational Research (EJOR)*, 161 :636–654, 2005.

[CT01] C. A. Coello Coello and G. Toscano Pulido. A micro-genetic algorithm for multiobjective optimization. In E. Zitzler, K. Deb, L. Thiele, C. A. Coello

Coello, and D. Corne, editors, *Proceedings of the First International Conference on Evolutionary Multi-criterion Optimization*, volume 1993 of *Lecture Notes in Computer Science*, pages 126–140. Springer-Verlag, 2001.

[CT03] C. Cotta and J. M. Troya. Embedding branch and bound within evolutionary algorithms. *Applied Intelligence*, 18 :137–153, 2003.

[CTG97] T. G. Crainic, M. Toulouse, and M. Gendreau. Toward a taxonomy of parallel tabu search heuristics. *INFORMS Journal on Computing*, 9(1) :61–72, 1997.

[CVL02] C. A. Coello Coello, D. A. Van Veldhuizen, and G. B. Lamont. *Evolutionary algorithms for solving Multi-Objective Problems*. Kluwer, New York, 2002.

[DAPM00] K. Deb, S. Agrawal, A. Pratab, and T. Meyarivan. A fast elitist Non-dominated Sorting Genetic Algorithm for Multi-objective Optimization : NSGA-II. In *Proceedings of the Parallel Problem Solving from Nature VI Conference*, pages 849–858, 2000.

[Dar59] C. Darwin. *On the Origin of Species*. John Murray, London, 1859.

[Dav85] L. Davis. Job-shop scheduling with genetic algorithms. In *International Conference on Genetic Algorithms and their Applications*, pages 136–140. Lawrence Erlbaum Associates, 1985.

[Deb01] K. Deb. *Multi-objective optimization using evolutionary algorithms*. Wiley, Chichester, UK, 2001.

[Del01] R. P. Beausoleil Delgado. Multiple criteria scatter search. In J. Pinho de Sousa, editor, 4^{th} *Metaheuristics International Conference (MIC'2001)*, pages 539–543, Porto, Portugal, July 2001.

[DEO03] A. Döyen, O. Engin, and C. Ozkan. A new artificial immune system approach to solve permutation flow-shop scheduling problem. In *Turkish Symposium on Artificial Intelligence and Neural Networks - TAINN'03*, volume 1, 2003.

[DFK+00] B. De Backer, V. Furnon, P. Kilby, P. Prosser, and P. Shaw. Solving vehicle routing problems using constraint programming and metaheuristics. *Journal of Heuristics*, 6(4) :501–524, 2000.

[DMC91] M. Dorigo, V. Maniezzo, and A. Colorni. Ant system : an autocatalytic optimizing process. Technical Report 91-016, Politecnico di Milano, Italy, 1991.

[DMC96] M. Dorigo, V. Manizzo, and A. Colorni. The ant system : optimization by a colony of cooperating agents. *IEEE Transaction on Systems, Man, and Cybernetics-Part B*, 26(1) :1–13, 1996.

[Dor92] M. Dorigo. *Learning and natural algorithmes*. PhD thesis, DEI, Politecnico di Milano, Italy, 1992.

[DPST03] J. Dréo, A. Pétrowski, P. Siarry, and E. Taillard. *Métaheuristiques pour l'optimisation difficile*. Eyrolles, 2003.

[DQ01] C. Duhamel and A. Quilliot. Coupling a metaheuristic with an exact method for the local access network design problem. In *Optimization*, Aveiro, Portugal, July 2001.

[DS03] I. Dumitrescu and T. Stützle. Combinations of local search and exact algorithms. In S. Cagnoni et al., editors, *EvoWorkshops 2003*, volume 2611 of

Lecture Notes in Computer Science, pages 211–223, Essex, UK, 2003. Springer-Verlag.

|Edg81| F. Y. Edgeworth. *Mathematical physics*. P. Keagan, London, England, 1881.

|EE99| N. H. Eklund and M. J. Embrecht. GA-based multi-objective optimization of visible spectra for lamp design. In C. H. Dagli, A. L. Buczak, J. Ghosh, M. J. Embrecht, and O. Ersoy, editors, *Smart engineering system design : neural networks, fuzzy logic, evolutionary programming, data mining and complex systems*, pages 451–456, New York, NY, USA, 1999. ASME Press.

|EMH01| M. Erickson, A. Mayer, and J. Horn. The niched pareto genetic algorithm 2 applied to the design of groundwater remediation systems. In E. Zitzler, K. Deb, L. Thiele, C. A. Coello Coello, and D. Corne, editors, *First International Conference on Evolutionary Multi-criterion Optimization*, volume 1993 of *Lecture Notes in Computer Science*, pages 681–695. Springer-Verlag, 2001.

|Esp98| M. L. Espinouse. *Flowshop et extensions : chevauchement des tâches, indisponibilité des machines et système de transport*. PhD thesis, University of Joseph Fourier-Grenoble 1, December 1998.

|FBT95| I. De Falco, R. Del Balio, and E. Tarentino. An analysis of parallel heuristics for task allocation in multicomputers. *Computing*, 3(59), 1995.

|FBTV94| I. De Falco, R. Del Balio, E. Tarantino, and R. Vaccaro. Improving search by incorporating evolution principles in parallel tabu search. In *International Conference on Machine Learning*, pages 823–828, 1994.

|FF93| C. M. Fonseca and P. J. Flemming. Genetic algorithms for multiobjective optimization : Formulation, discussion and generalization. In *Fifth International Conference on Genetic Algorithms (ICGA'93)*, pages 416–423, San Mateo, USA, 1993.

|FF94| C. Fleurent and A. Ferland. Genetic hybrids for the quadratic assignment problem. *DIMACS Series in discrete Mathematics and Theoretical Computer Science*, 16 :173–188, 1994.

|FF98| C. M. Fonseca and P. J. Fleming. Multiobjective optimization and multiple constraint handling with evolutionary algorithms-part I : A unified formulation. *EEE Transactions on Systems, Man., and Cybernetics, Part A : Systems and Humans*, 28(1) :26–37, 1998.

|FGNC01| G. Fedak, C. Germain, V. Neri, and F. Cappello. XtremWeb : building an experimental platform for Global Computing. 1^{st} *Workshop on Global Computing on Personal Devices (CCGRID2001), IEEE Press*, 1971, May 2001.

|Fou85| M. P. Fourman. Compaction of symbolic layout using genetic algorithms. In *Proceedings of the first international conference on genetic algorithms (ICGA)*, pages 141–153, 1985.

|FR04| H. Feltl and G. R. Raidl. An improved hybrid genetic algorithm for the generalized assignment problem. In *Symposium on Applied Computing, SAC'04*, pages 990–995, March 2004.

|FRW01| A. P. French, A.C. Robinson, and J.M. Wilson. Using a hybrid genetic-algorithm/branch and bound approach to solve feasibility and optimization

integer programming problems. *Journal of Heuristics*, 7(6) :551–564, November 2001.

[FVB91] T. A. Feo, K. Venkatraman, and J. F. Bard. A GRASP for a difficult single machine scheduling problem. *Computers and Operations Research*, 18 :635–643, 1991.

[GHGL99] C. A. Garrett, J. Huang, M. N. Goltz, and G. B. Lamont. Parallel real-valued genetic algorithms for bioremediation of TCE-contaminated groundwater. In *Congress on Evolutionary Computation (CEC'99)*, pages 2183–2189, Washinton, D. C. USA, 1999. IEEE service center.

[GLLK79] R. L. Graham, E. L. Lawler, J. K. Lenstra, and A. H. G. Rinnooy Kan. Optimization and approximation in deterministic sequencing and scheduling : a survey. In *Annals of Discrete Mathematics*, volume 5, pages 287–326. 1979.

[Glo77] F. Glover. Heuristic for integer programming using surrogate constraints. *Decision Sciences*, 8 :156–166, 1977.

[Glo86] F. Glover. Future paths for integer programming and links to artificial intelligence. *Computers and Operations Research*, 13(5) :533–549, 1986.

[Glo96] F. Glover. Tabu search and adaptive memory programing advances, applications and challenges. In R.S. Barr, R.V. Helgason, and J.L. Kennington, editors, *Interfaces in Computer Science and Operations Research*, pages 1–75. Kluwer Academic Publishers, Boston, 1996.

[GMR02] J. F. Gonçalves, J. J. M. Mendes, and M. G. C. Resende. A hybrid algorithm for the job shop scheduling problem. Technical Report TD-5EAL6J, AT&T Labs Research, September 2002.

[Gol89] D. E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley Publishing Company, Reading, Massachusetts, 1989.

[Gom63] R. E. Gomory. An algorithm for integer solutions of linear programs. *Recent Advances in Mathematical Programming*, pages 269–302, 1963.

[GP02] J. Grabowski and 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.

[GR87] D. E. Goldberg and J. Richardson. Genetic algorithms with sharing for multimodal function optimisation. In *Genetic algorithms and their applications : Proceedings of the Second International Conference on Genetic Algorithms*, pages 41–49, 1987.

[Gre87] J.J. Grefenstette. Incorporating problem specific knowledge into genetic algorithms. In L. Davis, editor, *Genetic algorithms and Simulated annealing*, Research Notes in Artificial Intelligence, pages 42–60, San Mateo, CA, USA, 1987. Morgan Kaufmann.

[Han86] P. Hansen. The steepest ascent mildest descent heuristic for combinatorial programming. In *Numerical Methods in Combinatorial Optimization*, Capri, Italy, 1986.

[Hen85] M. I. Henig. The shortest path problem with two objective functions. *European Journal of the Operational Research*, 25 :281–291, 1985.

[HJ95] M. J. M. Heijligers and J. A. G. Jess. High-level synthesis scheduling and allocation using genetic algorithms based on constructive topological scheduling techniques. In *2nd IEEE International Conference on Evolutionary Computation ICEC'95*, pages 61–66, 1995.

[HL03] M. Haouari and T. Ladhari. A branch and bound based local search for the flowshop problem. *Journal of the Operational Research Society*, 54(10) :1076–1084, 2003.

[HN93] J. Horn and N. Nafpliotis. Multiobjective optimization using the niched pareto genetic algorithm. IlliGAL Report 93005, Illinois Genetic Algorithm Laboratory, Univerty of Illinois at Urbana-Champaign, Illinois, USA, 1993.

[HNG94] J. Horn, N. Nafpliotis, and D. E. Goldberg. A niched Pareto Genetic Algorithm for Multiobjective Optimization. In *Proceedings of the first IEEE Conference on Evolutionary Computation, IEEE World Congress on Computational Intelligence*, pages 82–87, Toronto, Canada, 1994. IEEE Service Center.

[Hol75] J. H. Holland. *Adaptation in natural and artificial systems.* The University of Michigan Press, Ann Arbor, MI, USA, 1975.

[HR94] H. W. Hamacher and G. Ruhe. On spanning tree problems with multiple objectives. *Annals of Operations Research*, 52 :209–230, 1994.

[HW00] T.-P. Hong and T.-T. Wang. Fuzzy flexible flow shops at two machine centers for continuous fuzzy domains. *Information Sciences*, 129 :227–237, 2000.

[HWC00] T.-P. Hong, H.-S. Wang, and W.-C. Chen. Simultaneous applying multiple mutation operators in genetic algorithm. *Journal of Heuristics*, 6(4) :439–455, September 2000.

[IM98] H. Ishibuchi and T. Murata. A multi-objective genetic local search algorithm and its application to flowshop scheduling. *IEEE Transactions on Systems, Man, and Cybernetics - Part C : Applications and reviews*, 28 :392–403, 1998.

[IYM03] H. Ishibuchi, T. Yoshida, and 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.

[Jac56] J. R. Jackson. An extension of Johnson's results on job-lot scheduling. *Naval Research Logistics Quarterly*, 3, 1956.

[Jah02] C. A. R. Jahuira. Hybrid genetic algorithm with exact techniques applied to TSP. In *Second international workshop on Intelligent systems design and application*, pages 119–124. Dynamic Publishers, 2002.

[JCV03] C. A. R. Jahuira and E. Cuadros-Vargas. Solving the TSP by mixing GAs with Minimal Spanning Tree. In Ernesto Cuadros-Vargas, Eduardo Tejada-Gamero, and Adenilso da Silva Simão, editors, *1st International Conference of the Peruvian Computer Society*, pages 123–123, Lima-Perú, January 2003. Peruvian Computer Society.

[Joh54] S. M. Johnson. Optimal two- and three-stage production schedules with setup times included. *Naval Research Logistics Quarterly*, 1 :61–68, 1954.

[Joz04] N. Jozefowiez. *Modélisation et résolution approchée de problèmes de tournées multi-objectif*. PhD thesis, University of Lille, Lille, France, December 2004.

[KA95] J. Kosa and D. Andre. Parallel genetic programming on a network of transputers. Technical Report CS-TR-95-1542, Standford University, 1995.

[KC00] J. D. Knowles and D. W. Corne. Approximating the nondominated front using the pareto archived evolution strategy. *Evolutionary Computation*, 8 :149–172, 2000.

[KC02] J. D. Knowles and D. W. Corne. On metrics for comparing non-dominated sets. In IEEE Service Center, editor, *Congress on Evolutionary Computation (CEC'2002)*, volume 1, pages 711–716, Piscataway, New Jersey, May 2002.

[KE95] J. Kennedy and R.C. Eberhart. Particle swarm optimization. In IEEE Service Center, editor, *IEEE International Conference on Neural Networks*, pages 1942–1948, November 1995.

[KF04] K. Kostikas and C. Fragakis. Genetic programming applied to mixed integer programming. In *EuroGP 2004*, volume 3003 of *Lecture Notes in Computer Science*, pages 113–124, 2004.

[KGV83] S. Kirkpatrick, C.D. Gelatt, and M.P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598) :671–680, May 1983.

[Kim95] Y-D. Kim. Minimizing total tardiness in permutation flowshops. *European Journal of Operational Research*, 33 :541–551, 1995.

[KLM+04] G. W. Klau, I. Ljubic, A. Moser, P. Mutzel, P. Neuner, U. Pferschy, G. Raidl, and R. Weiskircher. Combining a memetic algorithm with integer programming to solve the prize-collecting Steiner tree problem. In K. Deb, R. Poli, W. Banzhaf, H.-G. Beyer, E. Burke, P. Darwen, D. Dasgupta, D. Floreano, J. Foster, M. Harman, O. Holland, P. Lanzi, L. Spector, A. Tettamanzi, D. Thierens, and A. Tyrrell, editors, *Proceedings of the 2004 Genetic and Evolutionary Computation Conference*, volume 3102, pages 1304–1315, Seattle, Washington USA, June 2004. Lecture Note in Computer Science.

[Koz92] J. R. Koza. *Genetic Programming : On the Programming of Computers by Means of Natural Selection*. Cambridge, MA : The MIT Press, 1992.

[KPF03] J. L. Klepeis, M. J. Pieja, and C. A. Floudas. Hybrid global optimization algorithms for protein structure prediction : Alternating hybrids. *Biophysical Journal*, 4(84) :869–882, February 2003.

[Kra97] L. V. Kragelund. Solving a timetabling problem using hybrid genetic algorithms. *Software Practice and Experience*, 27(10) :1121–1134, 1997.

[Kru93] M. Krueger. *Méthodes d'analyse d'algorithmes d'optimisation stochastiques à l'aide d'algorithmes génétiques*. PhD thesis, Ecole Nationale Supérieure des Télécommunications de Paris, 1993.

[KS75] W. H. Kohler and K. Steiglitz. Exact, approximate, and guaranteed accuracy algorithms for the flow-shop problem $n/2/F/\bar{F}$. *Journal ACM*, 22(1) :106–114, 1975.

[LDT05] J. Lemesre, C. Dhaenens, and E.-G. Talbi. An exact parallel method for a bi-objective permutation flowshop problem. *European Journal of Operational Research*, 2005. to appear.

[Lee97] D. Lee. Multiobjective design of a marine vehicle with aid of design knowledge. *International Journal for Numerical methods in Engineering*, 40 :2665–2677, 1997.

[LKH91] F. Lin, C. Kao, and C. Hsu. Incorporating genetic algorithms into simulated annealing. In *Fourth International Symposium on AI*, pages 290–297, 1991.

[LLP97] C-Y. Lee, L. Lei, and M. Pinedo. Current trends in deterministic scheduling. *Annals of Operations Research*, 70 :1–41, 1997.

[Man99] V. Maniezzo. Exact and approximate nondeterministic tree-search pro- cedures for the quadratic assignment problem. *INFORM Journal on Computing*, 11(4) :358–369, 1999.

[MG95] Samir W. Mahfoud and D. E. Goldberg. Parallel recombinative simulated annealing : a genetic algorithm. *Parallel Computing*, 21(1) :1–28, 1995.

[Mit59] L. G. Mitten. Sequencing n jobs on two machines with arbitrary time lags. *Management Science*, 5(3), 1959.

[MIT96] T. Murata, H. Ishibuchi, and H. Tanaka. Multi-objective genetic algorithm and its applications to flowshop scheduling. *Computers and Industrial Engineering*, 30 :957–968, September 1996.

[MM97] T. Mautor and P. Michelon. Mimausa : a new hybrid method combining exact solution and local search. In *Second International Conference on Meta-Heuristics (MIC 2)*, page 15, Sophia Antipolis, France, 1997.

[MOF92] O. Martin, S. Otto, and E. Felten. Large-step markov chains for the TSP : Incorporating local search heuristics. *Operation Research Letters*, 11(4) :219–224, 1992.

[Mos89] P. Moscato. On evolution, search, optimization, genetic algorithms and martial arts : Towards memetic algorithms. Technical Report 826, California Institute of Technology, Pasadena, California, USA, 1989.

[MPT00] S. Mardle, S. Pascoes, and M. Tamiz. An investigation of genetic algorithm for the optimization of multi-objective fisheries bioeconomic models. *International Transaction of Operations Research*, 7 :33–49, 2000.

[MS95] I. I. Melamed and I. K. Sigal. An investigation of linear convolution of criteria in multicriteria discrete programming. *Computational Mathematics and Mathematical Physics*, 35(8) :1009–1017, 1995.

[MS96] Z. Michalewicz and M. Schoenauer. Evolutionary algorithms for constrained parameter optimization problems. *Evolutionary Computation*, 4(1) :1–32, 1996.

[MTR00] H. Meunier, E. G. Talbi, and P. Reininger. A multiobjective genetic algorithm for radio network optimisation. In *CEC*, volume 1, pages 317–324, Piscataway, New Jersey, July 2000. IEEE Service Center.

[Nis94] V. Nissen. Solving the quadratic assignment problem with clues from nature. *IEEE Transactions on Neural Networks*, 5(1) :66–72, January 1994.

[NS96] E. Nowicki and C. Smutnicki. A fast tabu search algorithm for the flow shop problem. *European Journal of Operational Research*, 91 :160–175, 1996.

[NS03] E. Nowicki and C. Smutnicki. Some aspects of scatter search in the flow shop problem. Technical Report 45, Institute of Engineering Cybernetics, University of Technology, Wroclaw, Poland, 2003.

[OO95] U-M. O'Reilly and F. Oppacher. Hybridized crossover-based techniques for program discovery. In *IEEE International Conference on Evolutionary Computation ICEC'95*, pages 573–578, Perth, Australia, December 1995.

[PAM04] M. Palpant, C. Artigues, and P. Michelon. LSSPER : Solving the resource-constrained project scheduling problem with large neighbourhood search. *Annals of Operations Research*, 31(1) :237–257, 2004.

[Pap76] C.H. Papadimitriou. *The complexity of combinatorial optimization problems*. PhD thesis, Princeton University, New Jersey, USA, 1976.

[Par96] V. Pareto. *Cours D'Economie Politique*, volume 1–2. Rouge, Lausanne, Switzerland, 1896.

[PG99] G. Pesant and P. Gendreau. A constraint programming framework for local search methods. *Journal of Heuristics*, 5(3) :255–279, 1999.

[Pin95] M. Pinedo. *Scheduling : Theory, Algorithms, and Systems*. Prentice-Hall, 1995.

[PR91] M. Padberg and G. Rinaldi. A branch-and-cut algorithm for the resolution of large-scale symmetric traveling salesman problems. *SIAM Review*, 33(1) :60–100, 1991.

[PR98] S. Parthasarathy and 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.

[PRRL97] P.M. Pardalos, K.G. Ramakrishnan, M.G.C. Resende, and Y. Li. Implementation of a variance reduction based lower bound in a branch and bound algorithm for the quadratic assignment problem. *SIAM Journal on Optimization*, 7 :280–294, 1997.

[PS82] C. H. Papadimitriou and K. Steiglitz. *Combinatorial Optimization : Algorithms and Complexity*. Prentice-Hall, 1982.

[PVDD98] M. C. Portmann, A. Vignier, D. Dardihac, and D. Dezalay. Branch and bound crossed with GA to solve hybrid flowshops. *European Journal of Operational Research*, 107(2) :389–400, 1998.

[REB97] A. J. Ruiz-Torres, E. E. Enscore, and R. R. Barton. Simulated annealing heuristics for the average flow-time and the number of tardy jobs bi-criteria identical parallel machine problem. *Computers and Industrial Engineering*, 33 :257–260, 1997.

[RMB98] R. Z. Rios-Mercado and J. F. Bard. Computational experience with a branch-and-cut algorithm for flowshop scheduling with setups. *Computers and Operations Research*, 25(5) :351–366, 1998.

[RMB99] R. Z. Rios-Mercado and J. F. Bard. A branch-and-bound algorithm for permutation flow shops with sequence-dependent setup times. *IIE Transactions*, 31(8) :721–731, 1999.

[RR97] C. Rosing and C. ReVelle. Heuristic concentration : Two stage solution construction. *European Journal of Operational Research*, 97 :75–86, 1997.

[RY98] C.R. Reeves and T. Yamada. Genetic algorithms, path relinking and the flowshop sequencing problem. *Evolutionary Computation*, 6(1) :230–234, 1998.

[Sch85] J. D. Schaffer. Multiple objective optimisation with Vector Evaluated Genetic Algorithms. In Grefenstette, editor, *Genetic Algorithm and Their Applications : Proceedings of the First International Conference on Genetic Algorithms (ICGA'93)*, page 93, 1985.

[Sch95] J. R. Schott. *Fault tolerant design using single and multicriteria genetic algorithm optimization.* PhD thesis, Department of Aeronautics and Astronautics, Massachusetts Institute of Technologie, Cambridge, Massachusetts, USA, 1995.

[SD94] N. Srinivas and K. Deb. Multiobjective optimization using nondominated sorting in genetic algorithms. *Evolutionary Computation*, 2(3) :221–248, 1994.

[SG87] J. Suh and D. Van Gucht. Incorporating heuristic information into genetic search. In *2nd International Conference on Genetic Algorithms*, pages 100–107. Lawrence Erlbaum Associates, 1987.

[Sha98] P. Shaw. Using constraint programming and local search methods to solve vehicle routing problems. In *Principles and Practice of Constraint Programming - CP98, 4th International Conference*, volume 1520 of *Lecture Notes in Computer Science*, pages 417–431, Pisa, Italy, 1998. Springer-Verlag.

[Sil86] B. W. Silverman. *Density estimation for statistics and data analysis.* Chapman and Hall, London, England, 1986.

[SK99] S. Sayin and S. Karabati. A bicriteria approach to the two-machine flow shop scheduling problem. *European Journal of Operational Research*, (113) :435–449, 1999.

[SM90] K. Shahookar and P. Mazumber. A genetic approach to standard cell placement using meta-genetic parameter optimization. *IEEE Transactions on Computer-Aided Design*, 9(5) :500–511, May 1990.

[Stü98a] T. Stützle. An ant approach for the flow shop problem. In *Proceedings of the 6th European Congress on Intelligent & Soft Computing (EUFIT'98)*, volume 3, pages 1560–1564. Verlag-Mainz, 1998.

[Stü98b] T. Stützle. Applying iterated local search to the permutation flow shop problem. Technical report, Darmstadt University of Technology, Computer Science Department, April 1998.

[SU98] F. Sivrikaya-Serifoglu and G. Ulusoy. A bicriteria two-machine permutation flowshop problem. *European Journal of Operational Research*, 107(2) :414–430, 1998.

[SW03] H. Shi and L. Wang. A mixed branch-and-bound and neural network approach for the broadcast scheduling problem. pages 42–49, 2003.

[Tai93a] E. Taillard. Benchmarks for basic scheduling problems. *European Journal of Operations Research*, 64 :278–285, 1993.

[Tai93b] E. Taillard. Parallel iterative search methods for vehicle routing problem. *Networks*, 23 :661–673, 1993.

[Tal02] E-G. Talbi. A taxonomy of hybrid metaheuristics. *Journal of Heuristics*, 8 :541–564, 2002.

[Tan87] R. Tanese. Parallel genetic algorithms for a hypercube. In *Second International Conference on Genetic Algorithms ICGA'87*, pages 177–183, Cambridge, July 1987. MIT.

[TB02] V. T'kindt and J. C. Billaut. *Multicriteria Scheduling - Theory, Models and Algorithms*. Springer-Verlag, 2002.

[Ten03] D. Tenfelde-Pohehl. A recursive algorithm for multiobjective combinatorial optimization problems with Q criteria. unpublished, 2003.

[TG97] E. D. Taillard and L.M. Gambardella. An ant approach for structured quadratic assignment problems. In C. Roucairol, I. Osman, S. Martello, and S. Voss, editors, 2^{nd} *Metaheuristics International Conference MIC'97*, pages 217–222, Sophia Antipolis, France, July 1997.

[TK98] T. Tagami and T. Kawabe. Genetic Algorithm with a Pareto Partitioning Method for Multi-objective Flowshop Scheduling. In *Proceedings of the 1998 International Symposium of Nonlinear Theory and its Applications (NOLTA'98)*, pages 1069–1072, Crans-Montana, 1998.

[TMS94] E-G. Talbi, T. Muntean, and I. Samarandache. Hybridation des algorithmes génétiques avec la recherche tabou. In *Evolution Artificielle EA'94*, Toulouse, France, September 1994.

[TMTL02] V. T'kindt, N. Monmarché, F. Tercinet, and 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.

[TRMD01] E. G. Talbi, M. Rahoual, M. H. Mabed, and C. Dhaenens. A hybrid evolutionary approach for multicriteria optimization problems : Application to the flow shop. In E. Zitzler, K. Deb, and L. Thiele, editors, *Evolutionary Multi-Criterion Optimization*, volume 1993 of *Lecture Notes in Computer Science*, pages 416–428. Springer-Verlag, 2001.

[TSS94] *Scheduling Theory - Multi-stage Systems*. Kluwer edition, 1994.

[UT95] E. L. Ulungu and J. Teghem. The two phases method : An efficient procedure to solve bi-objective combinatorial optimization problems. *Foundations of Computing and Decision Sciences*, 20(2) :149–165, 1995.

[UYI04] S. Umetani, M. Yagiura, and T. Ibaraki. One-dimensional cutting stock problem with a given number of setups : a hybrid approach of metaheuristics and linear programming. In *Hybrid Metaheuristics HM'04, at workshop European Conference on Artificial Intelligence ECAI'04*, pages 101–114, Valencia, Spain, August 2004.

[Vae95] R. J. M. Vaessens. Permutation flow shop bound found by branch and bound technique. Personal communication, 1995.

[Vae96] R. J. M. Vaessens. Permutation flow shop bound found by branch and bound technique with initial solution of e. nowicki and c. smutnicki results. Personal communication, 1996.

[VBSK90] H-M. Voigt, J. Born, and I. Santibanez-Koref. Modelling and simulation of distributed evolutionary search processes for function optimization. In H-P. Schwefel and R. Manner, editors, *Parallel Problem Solving from Nature*, volume 496, pages 373–380, Dortmund, Germany, October 1990. Lecture Note in Computer Science, Springer-Verlag.

[VH01] M. Vasquez and J-K. Hao. A hybrid approach for the 0/1 multidimensional knapsack problem. In *IJCAI*, pages 328–333, 2001.

[VL00a] D. A. Van Veldhuizen and G. B. Lamont. Multiobjective optimization with messy genetic algorithms. In *Proceedings of the 2000 ACM Symposium on Applied Computing*, pages 470–476. ACM, 2000.

[VL00b] D. A. Van Veldhuizen and G. B. Lamont. On measuring multiobjective evolutionary algorithm performance. In *In 2000 Congress on Evolutionary Computation, Piscataway, New Jersey*, volume 1, pages 204–211, July 2000.

[WLK92] P. B. Wienke, C. Lucasius, and G. Kateman. Multicriterion target optimization of analytical procedures using a genetic algorithm. *Analitical Chimica Acta*, 265 :211–225, 1992.

[YA04] W-C. Yeh and 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, may 2004.

[Yam02] T. Yamada. A pruning pattern list approach to the permutation flowshop scheduling problem. *Essays and Surveys in Metaheuristics*, 12(1) :641–651, 2002.

[Yeh99] W.-C. Yeh. A new branch-and-bound approach for the n/2/flowshop/$\alpha F + \beta C_{max}$ flowshop scheduling problem. *Computers and Operations Research*, 26(13) :1293–1310, 1999.

[YL01] H. Yu and W. Liang. Neural network and genetic algorithm-based hybrid approach to expanded job-shop scheduling. *Computers and Industrial Engineering*, 39(3-4) :337–356, 2001.

[YL04] K-C. Ying and C-J. Liao. An ant colony system for permutation flow-shop sequencing. *Computers and Operational Research*, 31(5) :791–801, 2004.

[YR95] T. Yamada and C. R. Reeves. Solving the C_{sum} permutation flowshop scheduling problem by genetic local search. In *2nd IEEE International Conference on Evolutionary Computation ICEC'95*, pages 230–234, 1995.

[YS01] Z. Yong and 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.

[Zit99] E. Zitzler. Evolutionary algorithms for multiobjective optimization : Methods and applications. Master's thesis, Swiss federal Institute of technology (ETH), Zurich, Switzerland, November 1999.

[ZLB04] E. Zitzler, M. Laumanns, and S. Bleuer. A tutorial on evolutionary multiobjective optimisation. *Workshop on Multiple Objective Metaheuristics (MOMH 2002)*, 2004.

[ZP02] M. Zdansky and J. Pozivil. Combination genetic/tabu search algorithm for hybrid flowshops optimization. In *Conference on Scientific Computing Algorithmy'02*, pages 230–236, 2002.

[ZT99] E. Zitzler and L. Thiele. Multiobjective evolutionary algorithms : A comparative case study and the Strength Pareto Approach. *IEEE Transactions on Evolutionary Computation*, 3 :257–271, 1999.

[ZTL01] E. Zitzler, L. Thiele, and M. Laumanns. Scalable test problems for evolutionary multi-objective optimization. Technical Report 112, Computer Engineering and communication Networks Lab (TIK), Swiss Federal Institute of Technology, July 2001.

[ZVL01] J. B. Zydallis, D. A. Van Veldhuizen, and G. B. Lamont. A statistical comparison of multiobjective evolutionary algorithm including the MOMGA-II. In E. Zitzler, K. Deb, L. Thiele, C. A. Coello Coello, and D. Corne, editors, *Proceedings of the First International Conference on Evolutionary Multi-criterion Optimization*, volume 1993 of *Lecture Notes in Computer Science*, pages 226–240. Springer-Verlag, 2001.