UNIVERSITÉ
D'ANGERS

Année : 2007
N° d'ordre : 831

NOUVELLES FONCTIONS D'ÉVALUATION POUR LES PROBLÈMES D'ÉTIQUETAGE DE GRAPHES BMP ET MINLA

THÈSE DE DOCTORAT

Spécialité : Informatique

ÉCOLE DOCTORALE D'ANGERS

Présentée et soutenue publiquement

Le 2 juillet 2007

À Angers

Par **Eduardo A. RODRÍGUEZ TELLO**

Devant le jury ci-dessous :

Présidente :	Catherine ROUCAIROL,	Professeur à l'Université de Versailles
Rapporteurs :	Alain HERTZ,	Professeur à l'École Polytechnique Montréal, Canada
	Marc SCHOENAUER,	Directeur de Recherche à l'INRIA
Examinateurs :	Frédéric SAUBION,	Professeur à l'Université d'Angers
	José TORRES JIMENEZ,	Professeur au CINVESTAV Tamaulipas, Mexique
Directeur de thèse :	Jin-Kao HAO,	Professeur à l'Université d'Angers

Laboratoire d'Étude et de Recherche en Informatique d'Angers
2 Bd. Lavoisier, 49045 Angers Cedex 01

Remerciements

Mes premiers remerciements s'adressent à Jin-Kao HAO, mon directeur de thèse, pour m'avoir accueilli, encadré et soutenu tout au long de ces quatre années de thèse. Je lui suis particulièrement reconnaissant de m'avoir laissé une grande liberté scientifique, ce qui m'a offert la possibilité d'explorer des pistes de recherche en complet accord avec mes centres d'intérêt. Je le remercie particulièrement pour le temps consacré au cours de la phase finale de la rédaction de ce manuscrit.

Je souhaite exprimer ma gratitude à José TORRES JIMENEZ, aujourd'hui membre du jury de cette thèse, pour son aide précieuse et le soutien qu'il a bien voulu m'apporter pendant la préparation de ma thèse. Merci pour tes conseils qui ont été très enrichissants et fructueux ainsi que pour ton amitié sincère.

J'adresse mes sincères remerciements à Alain HERTZ et Marc SCHOENAUER, les deux rapporteurs de ma thèse, pour avoir lu de manière approfondie mon manuscrit. Leurs avis éclairés m'ont permis d'en améliorer le fond et la forme. Je remercie également Catherine ROUCAIROL et Frédéric SAUBION qui m'ont fait l'honneur de faire partie du jury de cette thèse.

Mes remerciements vont aussi à tous les membres permanents du LERIA et du département informatique ; j'ai beaucoup apprécié l'ambiance conviviale régnant au laboratoire. Je remercie tout particulièrement Laurent GARCIA et Jean-Michel RICHER pour leur disponibilité, et leur participation à la tâche laborieuse que constitue la relecture du présent manuscrit.

Je ne peux qu'adresser un remerciement particulier et amical à mon collège de bureau Frédéric LARDEUX et Estelle sa femme, pour toute l'aide que vous m'avez apportée au moment de mon arrivée en France.

Rédiger une thèse dans une langue étrangère n'est jamais facile. Je suis reconnaissance envers mes amis doctorants : Olivier CANTIN, Vincent DERRIEN, Adrien GOËFFON, Tony LAMBERT, Thomas RAIMBAULT pour avoir bien voulu corriger mon français. Je remercie par la même occasion mes collèges thésards : Edmundo BONILLA-HUERTA, Marc-Olivier BUOB, Amandine DUFFOUX, Giglia GÓMEZ VILLOUTA, José Crispín HERNÁNDEZ HERNÁNDEZ, Sylvain LAMPRIER, Jorge MATURANA ORTIZ et Antoine ROBIN pour les bons moments de détente passés en leur compagnie. Merci à vous tous, car vous avez fait de mon séjour en France une expérience inoubliable.

Il me faudrait une liste énorme pour remercier tous mes amis mexicains et français qui m'ont encouragé tout au long de mes études. Ami, trouve dans cette ligne tous mes remerciements.

Je remercie très affectueusement mes parents Santa et Eduardo ainsi que ma sœur Elda Claudia pour leur soutien moral, leur présence et leur écoute. Qu'ils trouvent dans ce travail le témoignage de mon affection. Je remercie également ma belle-famille pour la gentillesse et la générosité dont ils font preuve à mon égard.

Mes remerciements ne seraient pas complets si je n'y associais pas Lisset, ma femme, que je remercie du fond du cœur pour avoir accepté de partager avec moi la grande aventure d'effectuer un doctorat à l'étranger avec touts les doutes et les satisfactions que cette expérience a pu nous procurer. Lisset, ton soutien sans faille, ton réconfort quotidien et le bonheur que tu m'apportes ont permis le déroulement de mon travail dans les meilleures conditions. Je te serai toujours très reconnaissant.

Finalement, je remercie le Conseil National de Science et Technologie du Mexique (CONACYT) pour avoir mis à ma disposition tous les moyens financiers pour le développement de cette thèse.

À Lisset le grand amour de ma vie.

Sommaire

Liste des figures

Liste des tables

Liste des algorithmes

Introduction générale

Contexte de travail

L'optimisation combinatoire est une branche des mathématiques appliquées qui se situe au carrefour de plusieurs domaines, dont l'informatique, la recherche opérationnelle et l'intelligence artificielle. Pour les problèmes d'optimisation combinatoire, cette discipline vise à concevoir des méthodes de résolution aussi efficaces que possible. Les problèmes de ce type consistent à trouver, parmi un ensemble fini de configurations potentielles, une solution vérifiant un critère particulier établi par une *fonction objectif*. D'un point de vue calculatoire, les problèmes d'optimisation combinatoire induisent bien souvent des complexités algorithmiques élevées puisque bon nombre d'entre eux relèvent de la classe des problèmes NP-difficiles [Garey et Johnson, 1979]. De tels problèmes sont étudiés dans le cadre de ce travail de thèse.

Les problèmes d'optimisation combinatoire apparaissent dans plusieurs domaines très variés des sciences et de l'ingénierie. La conception et la fabrication de circuits intégrés [Wong *et al.*, 1988], la planification du trafic aérien [Subramanian *et al.*, 1994], le modelage des écosystèmes [Duan *et al.*, 1992], la chimie informatique [Neumaier, 1997], et la robotique médicale [Webb, 1991] sont seulement quelques exemples d'applications réelles qui peuvent se modéliser sous la forme d'un problème d'optimisation combinatoire.

En raison de l'importance pratique et théorique des problèmes d'optimisation combinatoire, un grand nombre de méthodes ont été développées pour les résoudre. Elles peuvent être classées en deux grands groupes : les *méthodes exactes* et les *méthodes approchées*. Les méthodes exactes garantissent de trouver, pour chaque instance de taille finie d'un problème, une solution optimale si elle existe, et dans le cas contraire de déterminer avec certitude l'absence de solution, lorsqu'aucune contrainte de temps n'est donnée [Papadimitriou et Steiglitz, 1982]. En revanche, les méthodes approchées ou *heuristiques* sacrifient l'optimalité des solutions pour fournir, en un temps de calcul raisonnable, des solutions sous-optimales de la meilleure qualité possible. C'est le cas notamment des algorithmes de *Recherche Locale* (RL) comme le Recuit Simulé (RS) et la Recherche Tabou (RT) auxquels nous nous intéressons de manière particulière dans cette thèse.

Indépendamment de leur nature, tous les algorithmes de RL utilisent une *fonction d'évaluation* et une *relation de voisinage*. La fonction d'évaluation permet d'estimer la qualité (ou coût) de chaque solution visitée au cours de la recherche. La relation de voisi-

nage définit l'ensemble des solutions potentielles qui peuvent être atteintes à partir de la solution courante. En conséquence, la fonction d'évaluation et la relation de voisinage sont deux composantes essentielles des algorithmes de RL puisqu'elles agissent sur leurs comportements, et déterminent des caractéristiques importantes du *paysage de recherche* [Stadler, 1992]. Un mauvais choix de l'un de ces deux composants fondamentaux peut produire une baisse de performance de l'algorithme.

Par exemple, une fonction de voisinage mal choisie peut produire un ensemble de solutions voisines exponentiel par rapport à la taille du problème traité. Par conséquent, chercher à améliorer les solutions potentielles peut prendre un temps exponentiel dans le pire des cas, ce qui n'est pas envisageable. De la même manière, une fonction d'évaluation mal conçue peut générer des paysages de recherche peu favorables pour les algorithmes de recherche, contenant par exemple beaucoup d'optima locaux ou de solutions voisines qui ont la même performance (des plateaux) [Duvivier *et al.*, 1996; Hertz et Widmer, 2003; Hoos et Stützle, 2004].

Bien que la fonction d'évaluation et la relation de voisinage aient été identifiées comme cruciales pour l'application efficace des métaheuristiques à la résolution des problèmes d'optimisation combinatoire, elles n'ont pas été étudiées avec la même attention. En effet, on observe que la relation de voisinage a été largement étudiée [Lin et Kernighan, 1973; Glover, 1996; Mladenović et Hansen, 1997; Cavique *et al.*, 1999; Yagiura et Ibaraki, 1999; Yagiura et Ibaraki, 2001] dans différents contextes, alors que la fonction d'évaluation est plutôt négligée dans la littérature. En effet, une pratique très fréquente en optimisation combinatoire est d'évaluer la qualité des solutions potentielles en utilisant simplement la fonction objectif associée au problème traité, même si dans certains cas cette fonction ne fournit pas un guidage efficace aux algorithmes de recherche vers des solutions de bonne qualité. C'est pourquoi dans cette thèse nous nous intéressons à l'étude des fonctions d'évaluation et plus particulièrement à la conception des nouvelles fonctions d'évaluation comme un moyen d'améliorer l'efficacité des algorithmes de recherche.

Motivation et objectifs

Au début de ce travail de thèse deux problèmes d'optimisation combinatoire ont spécialement retenu notre attention : le problème de *Minimisation de Largeur de Bande des Graphes* (BMP) et le problème de l'*Arrangement Linéaire Minimum des Graphes* (MinLA). Tous les deux sont des problèmes d'étiquetage de graphes et relèvent de la classe NP-difficile [Papadimitriou, 1976; Garey et Johnson, 1979].

Le problème BMP réside dans le fait de trouver une manière d'étiqueter les sommets d'un graphe de sorte que la plus grande différence absolue, appelée *largeur de bande* β, entre les étiquettes des sommets adjacents soit minimum. Depuis son origine, dans les années cinquante, le problème BMP a suscité l'intérêt de nombreux chercheurs grâce à ses applications, notamment dans les domaines suivants : stockage de données, conception de circuits VLSI, analyse des réseaux, électromagnétisme industriel [Esposito *et al.*, 1998a], méthodes des éléments finis [Sourd et Schoenauer, 1998], systèmes de transport

d'énergie à grande échelle, cinétique chimique, géophysique numérique [Piñana *et al.*, 2004], dans la gestion de grandes collections d'hypertextes [Berry *et al.*, 1996], etc.

Le problème MinLA consiste à trouver une manière d'étiqueter les sommets d'un graphe (un arrangement) de sorte que la somme de toutes les différences absolues entre les étiquettes des sommets adjacents, appelée *largeur totale des arêtes* LA, soit minimum. Ce problème, énoncé pour la première fois dans les années soixante, présente de nombreuses applications pratiques, en particulier dans les domaines suivants : conception de circuits électroniques [Harper, 1964; Adolphson et Hu, 1973], modélisation d'activités nerveuses dans le cortex [Mitchison et Durbin, 1986], biologie moléculaire [Karp, 1993], ordonnancement d'atelier [Adolphson, 1977; Ravi *et al.*, 1991], dessin automatique des graphes [Shahrokhi *et al.*, 2001], arrangement automatique des diagrammes de flux [Lai et Williams, 1999], pour en mentionner seulement quelques uns.

De nombreux travaux de recherche ont été effectués afin de développer des algorithmes exacts et approchés pour résoudre plus efficacement ces deux problèmes de grande importance théorique et pratique. Dans le cas du problème BMP, la majorité des algorithmes existants évaluent la qualité d'une solution comme le changement de la fonction objectif β qui est *a priori* la fonction d'évaluation la plus évidente. Cependant, l'utilisation de β comme une fonction d'évaluation peut être problématique pour une recherche efficace. En fait, β attribue la même valeur de coût à de nombreuses solutions potentielles, même lorsque ces solutions n'ont pas la même opportunité d'être améliorées dans les itérations ultérieures. Il en va de même pour les algorithmes conçus pour résoudre le problème MinLA, la majeure partie d'entre eux utilisent la fonction objectif LA associée au problème pour évaluer la qualité des solutions potentielles, et ceci malgré la faible capacité de guidage que cette fonction offre. De plus, nous pouvons imaginer que les fonctions d'évaluation β et LA, étant donné qu'elles attribuent le même coût à de nombreuses solutions potentielles, produisent des paysages de recherche contenant beaucoup de plateaux. Ceci représente un grand inconvénient car les paysages de recherche de ce type sont très difficiles à exploiter en utilisant un algorithme de recherche.

Partant de ces constats, il est justifié de s'interroger sur l'efficacité des fonctions d'évaluation utilisées actuellement pour résoudre ces deux problèmes d'étiquetage de graphes. L'hypothèse ayant motivée nos travaux de recherche est donc la suivante :

> « *La performance des algorithmes existants pour les problèmes BMP et MinLA, et plus généralement des algorithmes heuristiques, peut être améliorée en introduisant des nouvelles fonctions d'évaluation plus pertinentes* ».

Afin de confirmer une telle hypothèse, notre travail de thèse comporte trois objectifs principaux. Le premier consiste à développer des nouvelles fonctions d'évaluation pour les problèmes BMP et MinLA capables de surmonter les inconvénients des fonctions d'évaluation classiques β et LA. Pour atteindre cet objectif, il est nécessaire d'effectuer une étude approfondie des particularités mathématiques de ces deux problèmes. Cette étude permettra de collecter des informations susceptibles d'être intégrées dans les nouvelles fonctions d'évaluation afin d'augmenter leur efficacité de guidage.

Une fois créées ces nouvelles fonctions d'évaluation pour les problèmes étudiés, elles doivent être confrontées aux fonctions d'évaluation classiques β et LA afin de conclure

sur leur performance relative. Le deuxième objectif de cette thèse consiste donc à effectuer une telle comparaison expérimentale.

Le troisième et dernier objectif de ce travail de recherche concerne l'implémentation d'algorithmes approchés qui, en tirant avantage des nouvelles fonctions d'évaluation proposées, soient capables d'être concurrentiels avec les heuristiques les plus efficaces rapportées dans la littérature pour les problèmes BMP et MinLA. Ce dernier objectif vise donc à évaluer l'utilité pratique des nouvelles fonctions d'évaluation proposées et à améliorer les meilleures résultats connus pour ces deux problèmes.

Organisation de la thèse

Ce manuscrit comporte six chapitres dont nous esquissons une brève description dans les lignes suivantes :

- **Chapitre 1.** Expose une brève introduction aux concepts de base de l'optimisation combinatoire et de la théorie de la complexité. Une classification des algorithmes développés pour résoudre les problèmes d'optimisation combinatoire est ensuite montrée (méthodes exactes et approchées). Enfin, les principes généraux des algorithmes approchés les plus représentatifs sont brièvement décrits.
- **Chapitre 2.** Offre une étude de synthèse portant sur les techniques rapportées dans la littérature pour améliorer l'efficacité de guidage des fonctions d'évaluation. Cette étude est également accompagnée d'une classification originale de ces techniques par rapport à leurs caractéristiques.
- **Chapitre 3.** Introduit formellement le problème BMP et fait une brève description des algorithmes pour le résoudre. Ensuite, certaines caractéristique de la fonction d'évaluation classique β sont analysées. De telles informations seront par la suite intégrées dans une nouvelle fonction d'évaluation spécifique pour BMP, notée δ. Contrairement à la fonction classique β, cette nouvelle fonction tient compte des informations induites par tous les sommets du graphe pour distinguer des solutions ayant la même largeur de bande.
- **Chapitre 4.** Compare expérimentalement les fonctions d'évaluation β et δ à l'aide de deux algorithmes, la Descente Stricte et le Recuit Simulé. Les résultats de cette comparaison sont ensuite présentés. Dans un seconde temps, la mis en œuvre d'un *Recuit Simulé Amélioré*, appelé RSA-δ, est décrit en détail. Finalement, RSA-δ est comparé aux heuristiques de l'état de l'art en utilisant des instances d'essai issues de la littérature.
- **Chapitre 5.** Définit formellement le problème MinLA ainsi que la fonction d'évaluation classique LA pour ce problème. Ensuite, une étude de certaines particularités de la fonction LA est réalisée. Les résultats de cette étude, liés à la fréquence d'apparition des différences absolues, sont utilisés ultérieurement afin de créer une nouvelle fonction d'évaluation pour MinLA, appelée Φ. Cette nouvelle fonction plus informative permet de surmonter les principaux inconvénients présentés par

la fonction d'évaluation LA.

– **Chapitre 6.** Montre des comparaisons expérimentales entre les fonctions d'évaluation LA et Φ en utilisant la Descente Stricte et un Algorithme Mémétique. Ensuite, un *Algorithme Mémétique Amélioré* nommé AMA-Φ est présenté et sa performance est analysée. Au vu des limitations de cette approche mémétique, en fin de chapitre l'implémentation d'un *Recuit Simulé en Deux Phases* est présentée. Cet algorithme, appelé RSDP-Φ, est ensuite comparé expérimentalement vis-à-vis de cinq autres heuristiques de l'état de l'art.

Cette thèse conclut sur une synthèse de nos différentes contributions, et ouvre sur quelques perspectives de recherche.

Chapitre 1

Optimisation combinatoire et métaheuristiques

L'OPTIMISATION combinatoire est une branche des mathématiques appliquées qui se situe au carrefour de plusieurs domaines, dont l'informatique, la recherche opérationnelle et l'intelligence artificielle. Cette discipline vise à concevoir des méthodes de résolution aussi efficaces que possible pour les problèmes d'optimisation combinatoire. Les problèmes de ce type consistent à trouver, parmi un ensemble fini de configurations potentielles, une solution vérifiant un critère particulier. Ils sont réputés pour être très difficiles à résoudre.

Dans ce première chapitre nous présentons d'abord de manière concise les concepts fondamentaux de cette discipline et sa relation avec la théorie de la complexité. Ensuite, nous décrivons brièvement certaines méthodes, parmi les plus représentatives, développées pour résoudre les problèmes d'optimisation combinatoire. Ce chapitre a pour but de situer le contexte de ce travail de thèse et non de faire une synthèse exhaustive sur le sujet.

Sommaire

1.1 Optimisation combinatoire

En mathématiques, l'analyse combinatoire, ou simplement *la combinatoire*, étudie les configurations (combinaisons) d'ensembles finis d'objets, et les dénombrements [Comtet, 1974]. En particulier, elle s'intéresse aux méthodes permettant de compter les éléments dans des ensembles finis, on parle de *combinatoire énumérative*, de trouver les structures algébriques que ces objets peuvent avoir, on dit *combinatoire algébrique*, et de chercher des optima dans les configurations ainsi qu'à leurs existences, *combinatoire extrémale* et *optimisation combinatoire*.

Un problème d'optimisation combinatoire (POC) consiste à trouver parmi un ensemble discret de solutions potentielles la ou les meilleures solutions vérifiant un critère particulier. En général, cet ensemble est d'une part fini mais de cardinalité très élevée, et d'autre part décrit de manière implicite. Nous présentons maintenant une formulation générale d'un POC.

Définition 1.1 (Problème d'optimisation combinatoire) *Un problème d'optimisation combinatoire* Π *est soit un* problème de minimisation *soit un* problème de maximisation*, et il est composé par un ensemble I des instances du problème.*

Définition 1.2 (Instance d'un problème d'optimisation combinatoire) *Une instance* π *d'un problème d'optimisation combinatoire* Π *est un couple* $(\mathscr{X}, f)$*.* $\mathscr{X}$ *est l'*espace de recherche*, qui représente l'ensemble des solutions possibles.* $f : \mathscr{X} \to \mathbb{R}$ *est la* fonction objectif*, qui à chaque solution potentielle* $x \in \mathscr{X}$*, appelée* configuration*, associe un coût réel* $f(x)$*. L'objectif d'un tel problème revient à déterminer la solution optimale* x^**, dite* optimum global*, tel que* $f(x^*) \leq f(x)\ \forall x \in \mathscr{X}$ *s'il s'agit d'un problème de minimisation.*

Cependant, comme maximiser une fonction f est équivalent à minimiser $-f$, nous pouvons supposer, sans perte de généralité, que nous traitons des problèmes de minimisation.

Les problèmes d'optimisation combinatoire apparaissent dans plusieurs domaines, aussi bien théoriques qu'appliqués de l'informatique et d'autres disciplines très diverses comme la gestion, les sciences sociales, l'ingénierie, les transports et les télécommunications. La conception et la fabrication de circuits intégrés [Wong *et al.*, 1988], la planification du trafic aérien [Subramanian *et al.*, 1994], le modelage des écosystèmes [Duan *et al.*, 1992], la chimie informatique [Neumaier, 1997], et la robotique médicale [Webb, 1991] sont quelques exemples d'applications réelles qui peuvent se modéliser sous la forme d'un POC.

Étant donné que l'ensemble de solutions potentielles $\mathscr{X}$ est discret et fini, il existe toujours en théorie un algorithme pour trouver une solution optimale. Cet algorithme, désigné sous le nom de la *recherche exhaustive*, évalue et compare simplement la valeur $f(x)$ pour tout $x \in \mathscr{X}$. Néanmoins, le temps de calcul nécessaire pour atteindre une solution optimale peut devenir vite prohibitif, car dans beaucoup de problèmes d'optimisation combinatoire l'espace de recherche croît exponentiellement avec la taille du problème. Pour une grande classe de ces problèmes aucun algorithme déterministe de résolution en temps polynomial n'est connu. Ce phénomène a mené les chercheurs au

développement de la théorie de la complexité [Garey et Johnson, 1979], et en particulier, à l'étude des problèmes NP-complets que nous définissons dans la section suivante.

1.2 La théorie de la complexité

La théorie de la complexité repose sur la définition de classes qui regroupent les problèmes en fonction de la complexité des algorithmes permettant de les résoudre. Elle se concentre sur les problèmes de décision qui posent la question de l'existence d'une solution comme :

Étant donné un entier L, existe-t-il une solution $x \in \mathscr{X}$ telle que $f(x) \leq L$?

Cette théorie nous permet de classifier les problèmes de décision en deux grandes classes : la *classe P* (Polynomial time) qui comprend les problèmes pour lesquels il existe un algorithme déterministe de résolution en temps polynomial, et la *classe NP* (Nondeterministic Polynomial time) à laquelle appartiennent les problèmes pour lesquels un tel algorithme n'a pas encore été trouvé. Les problèmes NP-complets sont définis comme suit [Garey et Johnson, 1979] :

Définition 1.3 (Problème NP-complet) *Un problème de décision* Π *est NP-complet s'il satisfait les deux conditions suivantes :* $\Pi \in$ *NP, et tout problème NP se réduit polynomialement à* Π.

L'un des problèmes ouverts les plus fondamentaux et intéressants en informatique théorique est probablement la question si « P=NP ? ». Ceci revient à savoir si on peut résoudre un problème NP-complet avec un algorithme polynomial. Trouver un tel algorithme, pour un seul problème appartenant à la classe NP-complet, signifierait que tous les problèmes de cette classe pourraient être résolus en temps polynomial (voir Définition 1.3) et en conséquence, que P=NP. Cependant, il est commun de penser que P≠NP, mais aucune preuve n'a encore été trouvée jusqu'à aujourd'hui.

Il est important de préciser que les problèmes d'optimisation ne peuvent pas être classifiés comme problèmes NP-complets, puisqu'ils ne sont pas des problèmes de décision, même si pour chaque problème d'optimisation on peut définir un problème de décision qui a une complexité équivalente.

Définition 1.4 (Problème NP-difficile) *Un problème* Π *quelconque (de décision ou non) est NP-difficile si et seulement s'il existe un problème NP-complet* Π' *qui est réductible à lui polynomialement.*

La définition d'un problème NP-difficile est donc moins restreinte que celle de la NP-complétude [Papadimitriou et Steiglitz, 1982]. De cette définition on peut observer que pour montrer qu'un problème d'optimisation est NP-difficile, il suffit de montrer que le problème de décision associé à lui est NP-complet. De cette façon un grand nombre de problèmes d'optimisation ont été prouvés NP-difficiles. C'est notamment le cas des problèmes du Voyageur de Commerce [Johnson et Papadimitriou, 1985], de Partitionnement de Graphes [Garey et Johnson, 1979] et d'Affectation Quadratique [Koopmans et Beckmann, 1957].

Dans le cadre de ce travail de thèse, nous nous intéressons particulièrement aux problèmes d'optimisation combinatoire NP-difficiles. Nous présentons par la suite une classification des algorithmes qui ont été développés pour les résoudre.

1.3 Méthodes de résolution exactes et approchées

En raison de l'importance pratique et théorique des problèmes d'optimisation combinatoire, un grand nombre de méthodes ont été développées pour essayer de les résoudre. Elles peuvent être classées en deux grands groupes : les *méthodes exactes* et les *méthodes approchées*.

Les méthodes exactes garantissent de trouver, pour chaque instance de taille finie d'un problème, une solution optimale si elle existe, et dans le cas contraire de déterminer la non existence avec certitude, lorsqu'aucune contrainte de temps n'est donnée [Papadimitriou et Steiglitz, 1982]. En revanche, les méthodes approchées ou *heuristiques* sacrifient l'optimalité des solutions pour fournir, en un temps de calcul raisonnable, des solutions sous-optimales de la meilleure qualité possible.

La méthode exacte la plus simple consiste à énumérer la totalité des solutions potentielles en évaluant leurs coûts à l'aide de la fonction objectif, puis à choisir la meilleure d'entre elles. Cette approche appelée *recherche exhaustive*, bien que largement applicable, est inutilisable dans la pratique en raison du grand nombre de solutions potentielles pour n'importe quel problème de taille raisonnable. Cependant, il y a d'autres méthodes pour trouver des solutions optimales garanties.

Tout au début de l'histoire de l'optimisation combinatoire, la plupart des efforts ont été concentrés sur la Programmation Linéaire (PL). Le problème traité était reformulé en employant des variables représentant des nombres entiers qui habituellement prenaient les valeurs 0 ou 1 pour produire une formulation de Programmation en Nombres Entiers (PNE). Un tel problème pouvait être ensuite résolu grâce aux méthodes fondées sur le principe de *Branch-and-Bound*, également appelé séparation et évaluation [Land et Doig, 1960]. Dans cette méthode, l'analyse des propriétés du problème permet de construire une arborescence représentant une énumération intelligente et implicite des solutions potentielles. Cette arborescence est composée uniquement des branches (solutions partielles) pour lesquelles on n'a pas de preuve qu'elles ne peuvent pas mener à une solution optimale, de sorte que le temps de résolution est réduit considérablement par rapport à la recherche exhaustive.

La complexité, dans le pire des cas, des algorithmes de PNE croît exponentiellement avec la taille du problème. En conséquence, il arrive souvent qu'ils ne peuvent pas être utilisés pour résoudre des cas réels, en raison de leurs grandes tailles. De plus, pour certains problèmes il est difficile de trouver une formulation de PNE, et même si l'on en trouve une, il en résulte souvent un grand nombre de variables et de contraintes. C'est justement dans ce type de situations que l'utilisation des méthodes approchées s'avère indispensable.

Initialement, les heuristiques ont été traitées avec scepticisme. Néanmoins, les heuristiques ont aujourd'hui gagné une position proéminente parmi les méthodes d'opti-

misation. Non seulement grâce aux développements théoriques dans la complexité qui témoignent de la difficulté inhérente des problèmes NP-difficiles, mais aussi pour les excellents résultats produits par les heuristiques modernes sur de nombreux problèmes qui en raison de leur taille auraient été impossible de résoudre en utilisant des méthodes exactes.

Parmi les méthodes approchées de base on distingue habituellement les *méthodes constructives* et les *méthodes de recherche locale*. Les algorithmes constructifs produisent des solutions en ajoutant progressivement des composants sur la base de l'efficacité locale, jusqu'à ce qu'une solution soit complète. Ils constituent généralement les méthodes approchées les plus rapides, pourtant ils retournent souvent des solutions de qualité inférieure par rapport aux algorithmes de recherche locale, dont nous introduisons ses concepts de base dans la section suivante.

1.4 Recherche locale

Dans le domaine de l'optimisation combinatoire, les algorithmes de *Recherche Locale* (RL) ont une longue histoire puisqu'ils sont à la fois intuitifs et très efficaces. C'est le cas notamment du premier algorithme de RL pour le problème du Voyageur de Commerce, qui a été proposé dans les années cinquante [Croes, 1958]. Notons que dans la littérature, le terme RL est de plus en plus employé pour désigner la classe des *méthodes à base de voisinages* (recuit simulé, recherche tabou,...). Par la suite, nous désignerons donc par « Recherche Locale », l'ensemble de ces méthodes.

Généralement, un algorithme de RL explore l'espace de recherche $\mathscr{X}$ à partir d'une solution initiale x, construite par exemple par une heuristique constructive, puis il essaie, à chaque itération, de remplacer la solution courante x avec une autre solution x' trouvée parmi les configurations voisines de x, ce changement de configuration est couramment appelé un *mouvement*.

La RL se caractérise principalement par trois concepts : une fonction de voisinage, une fonction d'évaluation et une stratégie de mouvement.

Définition 1.5 (Fonction de voisinage) *Soit $\mathscr{X}$ l'espace de recherche d'un problème, une fonction de voisinage est une application $\mathcal{N} : \mathscr{X} \rightarrow 2^{\mathscr{X}}$ qui détermine pour chaque $x \in \mathscr{X}$ un ensemble de solutions voisines $\mathcal{N}(x) \subseteq \mathscr{X}$. On appelle $\mathcal{N}(x)$ le voisinage de x.*

Définition 1.6 (Fonction d'évaluation) *La fonction d'évaluation est une application $g : \mathscr{X} \rightarrow \mathbb{R}$ qui permet d'estimer la* qualité *ou* coût *$g(x)$ de chaque solution potentielle $x \in \mathscr{X}$.*

Remarquons que la fonction objectif f associée à un problème d'optimisation combinatoire particulier est très souvent employée comme fonction d'évaluation g. Cependant, dans certains cas des fonctions d'évaluation différentes peuvent fournir un guidage plus efficace vers des solutions de bonne qualité, comme nous le verrons dans le chapitre suivant. Par la suite, nous ferons systématiquement la distinction entre la valeur d'une fonction objectif $f(x)$ et celle d'une fonction d'évaluation $g(x)$ afin d'éviter toute confusion entre la définition du problème à résoudre (dont la définition inclut une fonction objectif)

et la définition d'un algorithme pour résoudre ce problème (qui pourrait employer une fonction d'évaluation différente de la fonction objectif).

Définition 1.7 (Stratégie de mouvement) *Une stratégie de mouvement $\mathcal{S}$ dans un algorithme de RL décrit la procédure de passage d'une solution courante x à une de ses solutions voisines $x' \in \mathcal{N}(x)$.*

Les concepts que nous venons de présenter ci-dessus nous amènent maintenant à introduire la notion de minimum local.

Définition 1.8 (Minimum local) *Étant donné un voisinage $\mathcal{N}$ et une solution $x \in \mathscr{X}$, on dit que x est un minimum local par rapport à $\mathcal{N}$ si $\forall x' \in \mathcal{N}(x),\ g(x) \leq g(x')$.*

L'approche de RL la plus connue est la méthode de *Descente*, appelé aussi *Hillclimbing*, qui grâce a sa simplicité d'implémentation a été largement utilisée en optimisation combinatoire. Nous présentons dans l'Algorithme 1.1 le pseudo-code de la méthode de *Descente Stricte* qui à chaque itération choisit un voisin s'il améliore strictement le coût de la configuration courante (calculé avec la fonction d'évaluation). Pour effectuer ce choix, les deux stratégies de mouvement les plus utilisées en pratique sont : sélectionner le premier voisin trouvé de coût inférieur (méthode de la première amélioration), et évaluer la totalité des voisins pour choisir le meilleur, uniquement s'il améliore la solution courante (méthode de la meilleure amélioration). L'Algorithme 1.1 illustre la stratégie de mouvement de la meilleure amélioration.

Algorithme 1.1 : Descente Stricte (hillclimbing)

Entrées : $\mathscr{X}, \mathcal{N}, g$
début
  générer une solution initiale $x \in \mathscr{X}$
  $x^* \leftarrow x$
  tant que *x n'est pas un optimum local* **faire**
    sélectionner $x' \in \mathcal{N}(x) : g(x') \leq g(x''), \forall\, x'' \in \mathcal{N}(x)$
    si $g(x') < g(x)$ **alors**
      $x \leftarrow x'$
      si $g(x) < g(x^*)$ **alors** $x^* \leftarrow x$
    fin
  fin
  retourner x^*
fin

1.5 Métaheuristiques

Bien que les méthodes de descente soient extrêmement rapides, leur principal inconvénient est que les solutions obtenues sont (par définition) seulement des optima locaux. Bien que ces solutions puissent être de bonne qualité, elles ne sont pas nécessairement

optimales. De plus, un fois que la méthode tombe dans un minimum local, elle n'a aucune manière évidente de continuer vers des solutions d'un meilleur coût. Pour remédier à cette situation, il existe plusieurs possibilités : la première, et la plus simple, est la *méthode de relance*. Elle consiste à générer une nouvelle solution de départ (de préférence éloignée de la précédente) et à recommencer une nouvelle phase de descente. La deuxième possibilité est celle du *chemin aléatoire*. Elle consiste à effectuer, de temps en temps, un mouvement aléatoire pour diversifier la recherche et ainsi essayer de se rapprocher un peu plus de l'optimum global.

Dans les vingt dernières années, un nouveau type de méthodes approchées a surgi. Essentiellement, ces méthodes essayent de combiner des méthodes heuristiques (comme les algorithmes constructifs et la méthode de descente) dans des cadres de haut niveau visant à explorer efficacement l'espace de recherche. Actuellement, ces nouvelles méthodes sont généralement appelées *métaheuristiques*. Le terme métaheuristique, utilisé pour la première fois en [Glover, 1986], est dérivé du suffixe *méta* qui dénote « au niveau supérieur, au-delà de » et du verbe grec *heuriskein* qui signifie « rechercher ». Avant l'adoption généralisée du mot métaheuristique, ces méthodes se sont souvent appelées *heuristiques modernes* [Reeves, 1993].

La suite de ce chapitre présente une brève description des métaheuristiques les plus connues, commençant par le Recuit Simulé. Nous invitons le lecteur intéressé par plus de détails sur les métaheuristiques à consulter les articles de synthèse et livres suivants [Rayward-Smith *et al.*, 1996; Osman et Kelly, 1996; Pirlot, 1996; Aarts et Lenstra, 1997; Hoos et Stützle, 2004], ainsi que les références suivantes [Corne *et al.*, 1999; Ibaraki, 1997] qui récapitulent l'approche hybride (par exemple l'hybridation de deux métaheuristiques, l'hybridation de méthodes exactes et approchées, ...).

1.6 Recuit simulé

Le Recuit Simulé (RS) a été introduit de manière indépendante par [Kirkpatrick *et al.*, 1983] et [Ĉerný, 1985]. Ses origines remontent aux expériences réalisées pour simuler l'évolution du processus de recuit physique en métallurgie [Metropolis *et al.*, 1953].

Le RS est considéré comme la plus ancienne métaheuristique et la première à en avoir intégré une stratégie explicite pour échapper aux minima locaux [Blum et Roli, 2003]. L'idée fondamentale de cette stratégie est d'autoriser de manière contrôlée des dégradations occasionnelles de la solution courante, c'est-à-dire des mouvements vers des configurations de moindre qualité. La probabilité d'exécuter un tel type de mouvement diminue à mesure que la recherche avance. L'Algorithme 1.2 présente le pseudo-code de cette métaheuristique.

Cet algorithme commence par générer une solution initiale $x \in \mathscr{X}$ (construite aléatoirement ou avec l'aide d'une heuristique), puis par initialiser un paramètre T, appelé *température*, à une valeur T_i. Chaque itération de cet algorithme consiste à choisir aléatoirement une solution voisine $x' \in \mathcal{N}(x)$ de la solution courante x, puis à décider si x' doit ou non remplacer x en utilisant le *critère de Metropolis*. Ce critère stochastique dépend de la différence du coût Δg entre la configuration voisine et la solution courante, ainsi que

de la valeur actuelle de la température T. Un mouvement de la solution x vers la solution x' est accepté dans l'une de deux conditions suivantes : a) il améliore la qualité de la solution courante, c'est-à-dire $\Delta g < 0$, ou b) il détériore la qualité de la solution courante ($\Delta g \geq 0$) et la probabilité d'accepter un tel mouvement est supérieure à une valeur aléatoire p choisi entre 0 et 1 ($e^{-\Delta g/T} > p$).

Algorithme 1.2 : Recuit Simulé

Entrées : $\mathscr{X}, \mathcal{N}, g, T_i, T_f, maxIter, \mathcal{Q}$
début
  générer une solution initiale $x \in \mathscr{X}$
  $x^* \leftarrow x$
  $T \leftarrow T_i$
  tant que $T > T_f$ **faire**
    $nbIter \leftarrow 0$
    tant que $nbIter < maxIter$ **faire**
      $nbIter \leftarrow nbIter + 1$
      sélectionner $x' \in \mathcal{N}(x)$
      $\Delta g \leftarrow g(x') - g(x)$
      tirer aléatoirement un réel $p \in [0, 1]$
      si $(\Delta g < 0)$ **ou** $(e^{-\Delta g/T} > p)$ **alors**
        $x \leftarrow x'$
        si $g(x') < g(x^*)$ **alors** $x^* \leftarrow x'$
      fin
    fin
    $T \leftarrow \mathcal{Q}(T)$
  fin
  retourner x^*
fin

Au début de l'algorithme quand la température est élevée, la probabilité d'accepter des dégradations est grande. Cette probabilité diminue graduellement à mesure que la recherche avance, contrôlée par la température T qui décroît selon un *schéma de refroidissement* prédéfini $\mathcal{Q}(T)$. En fin de recherche, lorsque la température atteint sa valeur minimale T_f (très proche de 0), plus aucune dégradation n'est autorisée et l'algorithme se comporte comme une descente stricte. C'est généralement à ce moment que l'algorithme se termine et retourne la meilleure solution trouvée.

Le choix d'un schéma de refroidissement approprié est crucial pour l'obtention d'une implémentation efficace du RS. Les paramètres qui définissent un schéma de refroidissement sont : une température initiale, une température finale ou un critère d'arrêt, le nombre maximum des solutions voisines qui peuvent être visitées à chaque température, et une règle pour décrémenter la température [Osman, 1995]. La littérature offre plusieurs schémas de refroidissement. Sans être exhaustif, on rencontre habituellement trois grandes classes de schémas :

– *Réduction par paliers.* La température reste constante pendant un certain nombre d'itérations avant d'être modifiée selon une règle de mise à jour. En général, la règle de mise à jour utilisée est une fonction de réduction géométrique $\mathcal{Q}(T_k) = \alpha(T_k)$, où

T_k représente la température courante et $\alpha \in [0, 1]$ le taux de refroidissement. C'est pourquoi ce type de refroidissement s'appelle souvent *refroidissement géométrique*. Le nombre d'itérations à chaque température est lié à la taille du voisinage mais peut également changer d'une température à l'autre [Kirkpatrick *et al.*, 1983].

– *Réduction continue.* La température est réduite après chaque itération. Cette réduction est très lente et est conduite selon la règle $\mathcal{Q}(T_k) = T_k/(1 + \rho T_k)$ où ρ est une petite valeur [Lundy et Mees, 1986].
– *Réduction non monotone.* La température est réduite après chaque itération, cependant des augmentations de température sont autorisées. [Connolly, 1990; Osman, 1993; Dowsland, 1993]

Pour d'autres schémas de refroidissement de température le lecteur peut se reporter aux articles suivants [Huang *et al.*, 1986; Collins *et al.*, 1988; Hajek, 1988; Abramson *et al.*, 1999; Varanelli et Cohoon, 1999; Poupaert et Deville, 2000].

De nombreux travaux de recherche sur les propriétés de convergence du RS ont été publiés. Les études employant la théorie des *chaînes de Markov* ont montré que si la température est diminuée assez lentement, l'algorithme convergera éventuellement vers un minimum global. Malheureusement, ces mêmes études prouvent que cette diminution lente de la température, exigera en général plus d'itérations que la recherche exhaustive. Pour une information plus détaillée sur les résultats de convergence de cette métaheuristique, le lecteur peut consulter ces deux excellents livres [Van Laarhoven et Aarts, 1988; Aarts et Korst, 1989]. De plus, [Johnson *et al.*, 1989; Johnson *et al.*, 1991] fournissent des très bons résultats expérimentaux concernant l'application du RS à la résolution de trois problèmes d'optimisation.

1.7 Recherche tabou

La *Recherche Tabou* (RT) est parmi les métaheuristiques les plus utilisées et référencées pour résoudre des POC. Les idées fondamentales de la RT ont été présentées pour la première fois par Glover [1986], elles sont basées sur les travaux précédents de l'auteur dans la résolution de problèmes de couverture non linéaires [Glover, 1977]. Les deux articles de référence qui ont contribué de manière importante à la popularité de cette méthode sont [Glover, 1989; Glover, 1990].

Une caractéristique essentielle de cette méthode est l'utilisation de structures de mémoire spécifiques. Contrairement à la méthode de RS qui choisit aléatoirement une configuration voisine, la recherche tabou examine à chaque itération un sous-ensemble (voir la totalité) du voisinage de la solution courante x, pour ne conserver que la configuration x' possédant la meilleure valeur $g(x')$ même si elle est dégrade x.

Cette stratégie permet d'échapper aux minima locaux, cependant elle peut hélas occasionner facilement des cycles. Pour éviter de boucler ainsi, la méthode utilise une structure de *mémoire à court terme*, appelée la *liste tabou* $\mathcal{T}$. Cette mémoire évite de revenir à des solutions déjà explorées, dites *solutions tabou*. Le voisinage de la solution courante $\mathcal{N}(x)$ est limité aux solutions qui n'appartiennent pas à la liste tabou, pour former le nouveaux voisinage, dit réduit $\mathcal{N}^*(x) = \{x'' \in \mathcal{N}(x) \setminus \mathcal{T}\}$. Une manière de gérer la liste

tabou consiste à mémoriser les $\mathcal{L}$ dernières configurations visitées pour en interdire l'accès. De cette manière, on empêche le processus de recherche d'être bloqué sur une suite de configurations de longueur inférieure ou égale à $\mathcal{L}$.

Cependant l'implémentation de la mémoire à court terme avec une liste des configurations entières n'est pas aisée. Sa gestion serait inefficace et trop coûteuse en temps de calcul, mais aussi en place mémoire. Par conséquent, il est courant de mémoriser dans la liste tabou des *caractéristiques* des configurations au lieu de configurations entières. Toutefois, garder des caractéristiques au lieu des solutions entières cause une perte d'information, car l'interdiction d'une caractéristique peut avoir des répercutions sur plusieurs solutions en les rendant tabou. De cette manière, il est alors possible qu'une solution de bien meilleure qualité soit exclue de l'ensemble $\mathcal{N}^*(x)$. Pour surmonter ce problème, un mécanisme particulier, appelé *critère d'aspiration* $\mathcal{A}$, est mis en place. Ce mécanisme consiste à révoquer le statut tabou d'une configuration selon certaines conditions. Le critère d'aspiration le plus utilisé consiste à enlever le statut tabou d'une configuration si celle-ci est meilleure que la meilleure solution trouvée jusqu'alors x^*. La méthode de RT, décrite ci-dessus est présentée sous la forme de pseudo-code dans l'Algorithme 1.3.

Algorithme 1.3 : Recherche Tabou

Entrées : $\mathscr{X}, \mathcal{N}, g, maxIter, \mathcal{L}, \mathcal{A}$
début
  générer une solution initiale $x \in \mathscr{X}$
  $x^* \leftarrow x$, $\mathcal{T} \leftarrow \emptyset$
  $nbIter \leftarrow 1$
  tant que $nbIter < maxIter$ **faire**
    construire $\mathcal{N}^*(x) = \{x'' \in \mathcal{N}(x) \setminus \mathcal{T}$ ou $\mathcal{A}(x'')\}$
    sélectionner $x' \in \mathcal{N}^*(x) : g(x') \leq g(x''), \forall\, x'' \in \mathcal{N}^*(x)$
    remplacer le plus ancien élément de $\mathcal{T}$ avec x
    si $g(x') < g(x^*)$ **alors** $x^* \leftarrow x'$
    $nbIter \leftarrow nbIter + 1$
  fin
  retourner x^*
fin

Un des paramètres les plus importants pour cette métaheuristique est la *longueur de la liste tabou* $\mathcal{L}$ qui contrôle la mémoire du processus de recherche. Avec une valeur petite de $\mathcal{L}$ la recherche se concentrera sur des petites zones de l'espace de recherche, *intensification*. En revanche, une valeur grande de $\mathcal{L}$ force le processus de recherche à visiter des régions plus vastes, *diversification*, parce qu'elle interdit de visiter un plus grand nombre de solutions. La longueur de la liste tabou peut aussi être modifiée à mesure que la recherche avance, menant à des algorithmes plus robustes. C'est notamment le cas de [Taillard, 1991] où la valeur de $\mathcal{L}$ est périodiquement réinitialisée au hasard dans l'intervalle $[\mathcal{L}_{min}, \mathcal{L}_{max}]$, ou de l'approche plus sophistiquée présentée dans [Battiti et Tecchiolli, 1994] qui augmente $\mathcal{L}$ en fonction du nombre de solutions revisitées, alors que sa valeur est diminuée s'il n'y a aucune amélioration. D'autres schémas plus avancés pour modifier dynamiquement la longueur de la liste tabou sont décrits dans [Glover, 1990].

Au cours des dernières années, la méthode de RT a évolué et a incorporé beaucoup de nouveaux éléments qui augmentent sa performance globale. Le lecteur intéressé trouvera une description plus détaillée des éléments présentés dans cette section en se reportant aux papiers de synthèse suivants [Glover *et al.*, 1993; Glover et Laguna, 1993; Glover, 1994; Glover, 1997]. D'autres techniques plus sophistiquées sont considérées dans le livre très complet de Glover et Laguna [1997]. De plus, la preuve de convergence de la méthode est proposée dans l'article de Glover et Hanafi [2002].

1.8 Algorithmes évolutionnaires

Les Algorithmes Évolutionnaires (AE) se sont essentiellement inspirés du modèle de l'évolution darwinienne des populations biologiques : les individus les plus adaptés survivent et se reproduisent [Darwin, 1859]. On peut distinguer quatre grandes classes d'AE : les Stratégies d'Évolution [Rechenberg, 1973; Schwefel, 1981], la Programmation Évolutionnaire [Fogel *et al.*, 1966; Fogel, 1995], les Algorithmes Génétiques [Holland, 1975; Goldberg, 1989] et la Programmation Génétique [Koza, 1992].

Ces méthodes se différencient essentiellement par leur manière de représenter l'information et par leur technique pour gérer l'évolution de la population d'une génération à l'autre. Un AE est typiquement composé de trois éléments fondamentaux. Une *population* composée de plusieurs individus chacun représentant une solution potentielle du problème donnée. Un *mécanisme d'évaluation* des individus permettant de mesurer leur adaptation à l'environnement et un *mécanisme d'évolution* de la population permettant, grâce à des opérateurs prédéfinis d'éliminer certains individus et d'en créer de nouveaux.

Nous invitons le lecteur intéressé par plus de détails sur les AE à consulter les références bibliographiques suivantes [Michalewicz, 1996; Bäck *et al.*, 1997a; Hertz et Kobler, 2000; Eiben et Schoenauer, 2002; Eiben et Smith, 2003].

Dans la suite de cette section, nous présentons à titre d'exemple les détails des Algorithmes Génétiques qui sont un des plus connus des AE.

1.8.1 Algorithmes génétiques

Les premiers travaux sur les Algorithmes Génétiques (AG) ont commencé dans les années soixante-dix lorsque Holland [1975] a essayé d'implanter artificiellement des systèmes évolutifs. Ensuite, l'ouvrage de Goldberg [1989] décrit l'utilisation des AG dans le cadre de résolution de problèmes d'optimisation concrets. Cet ouvrage a permis de mieux faire connaître les AG et a marqué le début d'un nouvel intérêt pour ces techniques.

Le principe de base des AG classiques introduits par Holland consiste à simuler le processus d'évolution naturelle des êtres vivants dans un environnement hostile. De manière générale, on dispose d'une *population* P de solutions appelées *individus*, dont l'adaptation à leur environnement est mesurée grâce à une *fonction d'aptitude* g qui retourne une valeur réelle, appelée *fitness* (ceci peut être la valeur de la fonction objectif ou un autre genre de mesure de la qualité). Ces individus sont codés sous forme de chaînes binaires de longueur fixe. À chaque itération la population courante est soumise à un *mécanisme*

d'évolution décrit à l'aide d'*opérateurs génétiques*. Ces derniers agissent aléatoirement sur un ou deux individus sans aucune connaissance du problème traité, afin de produire une nouvelle population P' (une nouvelle génération).

Les AG standards utilisent des opérateurs appelés *croisement* pour recombiner deux individus et ainsi produire de nouveaux *fils* ou descendants. Ils emploient également des opérateurs de *mutation* ou modification qui causent une auto-adaptation des individus. Un autre opérateur important et indispensable est celui de *sélection*, chargé de choisir des individus en fonction de leur valeur d'aptitude (fitness). Les individus avec la meilleure aptitude ont une probabilité plus élevée d'être choisis comme membres de la population de la prochaine itération (ou comme parents pour la génération de nouveaux individus). Ceci correspond au principe de « la survie du plus apte » dans l'évolution naturelle. Le mécanisme itératif d'évolution décrit ci-dessus continue jusqu'à ce qu'une condition d'arrêt soit vérifiée. Par exemple, quand un certain nombre de générations, fixé a priori, est complété ou lorsque la population cesse d'évoluer (une population homogène). À la fin de l'AG le meilleur individu trouvé x^* est retourné.

Bien que les AG soient actuellement considérés comme une méthode d'optimisation, l'objectif initial consistait à concevoir des systèmes d'apprentissage généraux, robustes et adaptatifs, applicables à une large classe de problèmes. Ce principe de généralité a permis à Holland d'analyser théoriquement les AG, de proposer la *théorie des schémas*, ainsi que de caractériser le rôle et l'importance du croisement [Holland, 1975].

Néanmoins, à cause de cette généralité, les AG classiques sont en pratique peu efficaces car ils utilisent uniquement des opérateurs « aveugles » (aléatoires). En effet, les résultats produits pour un tel algorithme sont rarement comparables à ceux d'une métaheuristique à base de voisinage [Beasley *et al.*, 1993; Tomassini, 1995]. Pour cette raison, il est désormais bien connu qu'il est essentiel d'incorporer dans les AG une certaine forme de connaissance du problème pour arriver à produire des algorithmes de recherche efficaces [Grefenstette, 1987; Davis, 1991; Bäck *et al.*, 1997b; Hoos et Stützle, 2004]. Ceci peut être réalisé de différentes manières, notamment en employant, à la place des opérateurs aléatoires de croisement et de mutation, des opérateurs génétiques qui sont conçus en se basant sur des connaissances spécifiques du problème [Hao *et al.*, 1999; Nagata, 2006; Nagata, 2007], ou en utilisant des codages spécialisés. Une autre approche consiste à *hybrider* les AG avec des méthodes de RL. Par la suite, nous présentons les détails de ce type d'hybridation.

1.8.2 Algorithmes mémétiques

L'idée essentielle d'hybrider les AG avec des méthodes de RL est de profiter au maximum de la capacité d'exploitation des méthodes à base de voisinages (la recherche se concentrera sur des petites zones), et de l'habilité d'exploration des AG (la recherche visite des régions plus vastes). Les *Algorithmes Mémétiques* (AM) suivent cette approche [Moscato, 1989; Moscato, 1999]. Sous différents contextes et situations, les AM sont également connus sous le nom de recherche locale génétique ou algorithmes génétiques hybrides [Corne *et al.*, 1999].

En général, dans un AM l'opérateur de RL remplace ou succède à la mutation, et le

croisement doit impérativement être adapté au problème traité. Les principales caractéristiques fonctionnelles d'un AM sont décrites dans l'Algorithme 1.4, qui reçoit comme paramètres : la taille de la population n, la probabilité de croisement P_c et de mutation P_m, ainsi que le nombre maximal de générations $maxIter$.

Algorithme 1.4 : Algorithme Mémétique

Entrées : $\mathscr{X}, g, n, maxIter, P_c, P_m$
début
  générer$(P,\ n)$ ▷ $P = \{x_1, x_2, ..., x_n\},\ P \subseteq \mathscr{X}$
  rechercheLocale(P)
  $x^* \leftarrow$ **évaluation**$(P,\ g)$ ▷ `retourne le meilleur individu`
  $nbIter \leftarrow 1$
  tant que $nbIter < maxIter$ **faire**
    $P' \leftarrow$ **croisement**$(P,\ P_c)$
    $P'' \leftarrow$ **mutation**$(P',\ P_m)$
    rechercheLocale(P'')
    $x \leftarrow$ **évaluation**$(P'',\ g)$
    $P \leftarrow$ **sélection**$(P'' \cup P)$
    si $g(x) < g(x^*)$ **alors** $x^* \leftarrow x$
    $nbIter \leftarrow nbIter + 1$
  fin
  retourner x^*
fin

De nombreuses applications ont prouvé à quel point les algorithmes mémétiques pouvaient être performants, sur des problèmes bien connus comme celui du Voyageur de Commerce [Freisleben et Merz, 1996b; Freisleben et Merz, 1996a], du Sac à Dos [Falkenauer, 1996], d'Affectation Quadratique [Merz et Freisleben, 1997], de Coloration de Graphes [Galinier et Hao, 1999], de Satisfiabilité [Lardeux *et al.*, 2006] et Maximum de Parcimonie [Goëffon *et al.*, 2006].

Le lecteur intéressé pourra se reporter aux références suivantes pour une description plus détaillée des éléments présentés dans cette section : [Ibaraki, 1997; Merz et Freisleben, 1999; Hart *et al.*, 2004].

1.9 Synthèse du chapitre

Nous avons présenté dans ce chapitre une brève introduction aux concepts de base de l'optimisation combinatoire et de la théorie de la complexité, tout en soulignant l'importance théorique et pratique des problèmes issus de cette branche des mathématiques, et plus particulièrement des problèmes NP-difficiles. Une classification des algorithmes développés pour les résoudre a ensuite été montrée (méthodes exactes et approchées). Enfin, nous avons brièvement décrit les principes généraux des algorithmes approchés, notamment de la Descente Stricte et des métaheuristiques suivantes : Recuit Simulé, Recherche Tabou et Algorithmes Génétiques.

L'univers des méthodes approchées est vaste, et beaucoup de méthodes n'ont pas été

présentées ici. Par exemple, la Recherche à Voisinage Variable [Mladenović et Hansen, 1997], la méthode GRASP [Féo et Resende, 1995], la Recherche Dispersée [Glover, 1999; Laguna et Martí, 2003], les Algorithmes de Colonies de Fourmis [Dorigo et Colorni, 1996; Dorigo et Stützle, 2004], l'Optimisation par Essaims de Particules [Bonabeau *et al.*, 1999], etc. Il est cependant essentiel d'attirer l'attention sur le fait que toutes ces méthodes utilisent une fonction d'évaluation g pour guider le processus de recherche afin d'explorer partiellement et sélectivement l'ensemble de configurations potentielles $\mathscr{X}$. Cette fonction g est donc un élément qui influence de manière très importante la performance des algorithmes de recherche.

Dans la mesure où il n'y a aucune étude systématique sur l'importance de la fonction d'évaluation dans l'optimisation combinatoire, nous fournissons dans le chapitre suivant un examen global de plusieurs approches, présentées dans la littérature, pour développer des fonctions d'évaluation efficaces.

Chapitre 2

Fonction d'évaluation

LA FONCTION d'évaluation est un composant essentiel pour tout algorithme de recherche, dans la mesure où elle est utilisée pour estimer la qualité de chaque solution potentielle dans l'espace de recherche. Elle guide donc les algorithmes de recherche vers des solutions de la meilleure qualité possible.

Plusieurs approches visant à améliorer l'efficacité de guidage des fonctions d'évaluation ont été proposées dans la littérature. Dans ce chapitre, nous présentons une étude de synthèse portant sur ces techniques, et mettons en avant l'importance de la fonction d'évaluation dans l'optimisation combinatoire. Nous accompagnons cet état de l'art par une classification des techniques analysées par rapport à leurs caractéristiques.

Sommaire

2.1 Importance de la fonction d'évaluation

En optimisation combinatoire, la fonction d'évaluation et la relation de voisinage déterminent des caractéristiques importantes du *paysage de recherche* et agissent sur le comportement de tout algorithme basé sur ces dernières [Stadler, 1992]. Un mauvais choix de l'un de ces deux éléments fondamentaux produira généralement une baisse de performance de l'algorithme de recherche.

Par exemple, une fonction de voisinage mal choisie peut produire un ensemble de solutions voisines exponentiel par rapport à la taille du problème traité. Par conséquent, chercher à améliorer les solutions potentielles peut prendre un temps exponentiel dans le pire des cas, ce qui n'est pas envisageable. De la même manière, une fonction d'évaluation mal conçue peut générer des paysages de recherche peu favorables pour les algorithmes de recherche, contenant par exemple beaucoup d'optima locaux ou de solutions voisines qui ont la même performance (des plateaux) [Duvivier *et al.*, 1996; Hertz et Widmer, 2003; Hoos et Stützle, 2004].

Bien que ces deux éléments fondamentaux aient été identifiés comme cruciaux pour l'application efficace des métaheuristiques à la résolution des POC, ils n'ont pas été étudiés avec la même attention. En effet, on observe que la relation de voisinage a été largement étudiée [Lin et Kernighan, 1973; Glover, 1996; Mladenović et Hansen, 1997; Cavique *et al.*, 1999; Yagiura et Ibaraki, 1999; Yagiura et Ibaraki, 2001] dans différents contextes, alors que la fonction d'évaluation est plutôt négligée dans la littérature.

Nous venons de constater dans le chapitre précédent que toutes les méthodes développées pour résoudre les POC, qu'elles soient exactes ou approchées, utilisent une fonction d'évaluation pour estimer la qualité (ou coût) de chaque solution visitée au cours de la recherche. Le rôle de la fonction d'évaluation est donc de guider les algorithmes vers des solutions de bonne qualité dans un espace de recherche combinatoire. C'est le cas notamment de la fonction d'évaluation dans un algorithme génétique, où elle mesure l'aptitude (fitness) d'un individu et elle en détermine considérablement sa survie dans les générations futures [Beasley *et al.*, 1993]. De la même manière, dans les méthodes de RL (Descente, Recuit Simulé, Recherche Tabou,...), c'est la fonction d'évaluation qui influence directement le choix des solutions voisines et en conséquence la trajectoire du processus de recherche. Plus cette fonction d'évaluation est informative, plus le processus de recherche sera efficace.

On remarque aussi qu'en intelligence artificielle, la fonction d'évaluation est identifiée comme une structure de données fondamentale pour la prise de décisions dans les grands espaces de recherche (espaces de configurations). Par exemple, le choix de la fonction d'évaluation détermine, de façon très importante, les résultats de la recherche dans les algorithmes de planification et de contrôle comme A^* ou encore de la théorie des jeux comme $\alpha\beta$ [Nilsson, 1986]. Dans le domaine du jeu d'échecs, une fonction d'évaluation efficace a été obtenue en additionnant l'avantage des pièces pondérée par 1 pour les pions, 3 pour les fous et les cavaliers, 5 pour les tours, et 9 pour les dames [Hsu *et al.*, 1990]. Cette fonction d'évaluation a été un des facteurs déterminants dans le triomphe du système « Deep Blue » sur Garry Kasparov, le champion mondial de la spécialité, en 1997 [Campbell *et al.*, 2002].

Nous venons de discuter de l'importance du choix d'une fonction d'évaluation et d'une relation de voisinage. Les fonctions d'évaluation ayant été beaucoup moins analysées que les relations de voisinage, il nous a semblé pertinent de réaliser une étude portant sur l'état de l'art des techniques employées pour rendre plus précise l'évaluation des solutions potentielles. Cette étude de synthèse nous a permis d'établir une classification des techniques analysées par rapport à leurs caractéristiques :

- Fonctions d'évaluation avec des pénalités
- Fonctions d'évaluation dynamiques
- Fonctions d'évaluation par apprentissage
- Fonctions d'évaluation hiérarchiques
- Fonctions d'évaluation spécifiques au problème traité

Il est évident que cette classification n'est pas la seule possible, mais elle nous paraît être la plus appropriée pour présenter de manière structurée l'ensemble des méthodes décrites par la suite.

2.2 Fonctions d'évaluation avec des pénalités

Pour certains problèmes d'optimisation combinatoire, l'espace de recherche inclut des solutions non réalisables (ou non faisables). Dans ce cas, la fonction d'évaluation g se compose habituellement par la somme pondérée de la fonction objectif f et de termes de pénalité qui évaluent le degré de non faisabilité d'une solution.

Un exemple parlant de ce genre de fonction d'évaluation serait celui présenté dans [Johnson *et al.*, 1989]. Les auteurs ont proposé un algorithme de RS pour résoudre le problème de Partitionnement de Graphes (PPG). Étant donné un graphe $G(V, E)$, où le nombre de sommets $|V|$ est pair et E est l'ensemble d'arêtes, le PPG consiste à trouver une partition de l'ensemble de sommets, $V = V_1 \cup V_2$ en sous-ensembles d'égale taille, qui réduit au minimum le nombre d'arêtes dont les extrémités appartiennent à différents ensembles. Le coût d'une partition (V_1, V_2) est calculé avec la fonction :

$$g(V_1, V_2) = |\{\{u, v\} \in E : u \in V_1,\ v \in V_2\}| + \alpha(|V_1| - |V_2|)^2 , \tag{2.1}$$

où α est un terme de pénalité fixé à l'avance, appelé facteur de déséquilibre. Même si ce mécanisme autorise les partitions non faisables, il les pénalise selon le carré du déséquilibre.

Pour ce type de fonction d'évaluation, il est recommandable d'employer de petits poids de pénalité afin d'améliorer la performance des algorithmes de recherche. De cette manière, les visites dans les régions non faisables sont plus fréquemment encouragées.

La technique des pénalités a été appliquée pendant longtemps à un grand nombre de problèmes combinatoires. C'est le cas notamment des problèmes de Satisfaction de Contraintes (Constraint Satisfaction Problems, CSP). Par exemple, dans [Tsang et Wang, 1992; Davenport *et al.*, 1994] les auteurs rapportent un système appelé GENET pour le problème d'Ordonnancement de Véhicules. Il est basé sur un réseau de neurones et utilise une fonction d'évaluation avec des pénalités modifiées de manière adaptative. Un autre exemple est montré dans l'algorithme de RT présenté par Galinier et Hao [1998]

pour le problème d'Affectation d'Équipages (Progressive Party Problem PPP) [Brailsford *et al.*, 1996]. Dans ce cas, les auteurs ont déterminé empiriquement un poids spécifique de pénalité pour chacun des trois types de contraintes que comporte le problème. Pour plus de détails sur cette approche appliquée aux problèmes CSP, le lecteur intéressé peut consulter les références suivantes : [Olsen, 1994; Hao *et al.*, 1999; Schoofs et Naudts, 2000; Runarsson et Yao, 2002].

D'ailleurs, la technique des pénalités est aussi très employée pour résoudre des problèmes de satisfiabilité (SAT). En effet, l'approche connue sous le nom de *pondération de clauses* (clause weighting) emploie des poids de pénalité associés aux clauses de la formule logique du problème traité. C'est le cas de l'algorithme GSAT avec pondération de clauses présenté par Selman et Kautz [1993] et des versions de cet algorithme étudiées dans [Cha et Iwama, 1995; Frank, 1996; Frank *et al.*, 1997]. Dans le cas particulier du problème MAX-SAT (la variante d'optimisation du problème SAT), la méthode de pondération de clauses est aussi largement utilisée et s'inspire des techniques de relaxation Lagrangienne [Wu et Wah, 2000; Hutter *et al.*, 2002].

Il est important de noter que l'on ne trouvera pas de solutions réalisables avec ces approches si les poids de pénalité fixés sont trop petits. Pour éviter ce problème, plusieurs techniques ont été proposées pour contrôler autrement les pénalités. C'est notamment le cas des méthodes de pénalités dynamiques, adaptatives, basées sur le principe du RS, ...

D'autres algorithmes utilisant une fonction d'évaluation avec des pénalités, pour résoudre autres problèmes d'optimisation combinatoire, sont présentés dans [Gendreau *et al.*, 1994; S. Szykman et Weisser, 1998; Yagiura *et al.*, 1999; Gribkovskaia *et al.*, 2007]. De plus, dans [Michalewicz et Schoenauer, 1996] et [Coello Coello, 2002] les auteurs dressent un panorama très complet de nombreuses techniques de pénalités.

Notons enfin que certaines méthodes présentées dans la section suivante sont en réalité fondées sur le principe de pénalité.

2.3 Fonctions d'évaluation dynamiques

Parfois, il est également efficace de changer dynamiquement la fonction d'évaluation pendant la recherche comme dans les méthodes suivantes : bruitage, perturbation, lissage de l'espace de recherche, recherche locale guidée et évasion (breakout). Ces méthodes seront décrites tout au long de cette section.

2.3.1 Méthode de bruitage

La méthode de *Bruitage* (Noising Method) proposée par Charon et Hudry [1993] tire son origine des expériences réalisées par les auteurs en essayant d'améliorer un algorithme génétique pour résoudre des problèmes d'optimisation combinatoire avec des contraintes structurales fortes. Ils pensaient que leur algorithme génétique pourrait produire de meilleurs résultats si l'environnement, mesuré d'une certaine manière par la fonction à optimiser f (fonction objectif), fluctuait comme cela se passe dans la génétique réelle. Ainsi, ils ont décidé de perturber les valeurs de f avec des bruits aléatoires. Les

valeurs de ces bruits sont diminuées graduellement à mesure que la recherche avance, de sorte que la fonction d'évolution bruitée $\tilde{g}$ converge vers f à la fin.

Pour comparer la performance de leur nouvel algorithme génétique, ils ont appliqué également cette fonction d'évolution bruitée $\tilde{g}$ à un algorithme de *Recherche Locale Itérative* (RLI). Étonnamment, les expériences ont prouvé que l'algorithme de RLI employant le schéma du bruit fournissait de meilleurs résultats que l'algorithme de RLI classique et même que l'algorithme génétique. Partant de ces résultats, les auteurs ont étudié plusieurs variantes de leur méthode et l'ont appliquée avec succès à différents problèmes d'optimisation (voir l'article de synthèse [Charon et Hudry, 2002]).

Les principes de base de la méthode de bruitage sont récapitulés dans l'Algorithme 2.1. De nombreuses manières de perturber les valeurs de la fonction objectif f ont été présentées dans la littérature [Charon et Hudry, 1999; Charon et Hudry, 2001]. Cependant, l'une des plus utilisées est l'ajout de bruit à la variation Δf.

Algorithme 2.1 : Méthode de Bruitage

```
   Entrées : r_max, maxIter, nbV, nbB
 1 début
 2     générer une solution initiale x ∈ 𝒳
 3     x* ← x
 4     r ← r_max
 5     nbIter ← 0
 6     tant que nbIter < totalIter faire
 7         nbIter ← nbIter + 1
 8         sélectionner x' ∈ 𝒩(x)
 9         générer une valeur aléatoire ρ ∈ [−r, r]
10         calculer Δf(x, x') et Δg̃(x, x') = Δf(x, x') + ρ
11         si Δg̃(x, x') < 0 alors
12             x ← x'
13             si f(x) < f(x*) alors x ← x
14         fin
15         si nbIter ≡ 0 (mod nbV) alors diminuer r
16         si nbIter ≡ 0 (mod nbB) alors
17             rechercheLocale(x)                    ▷ obtient une nouvelle x
18             si f(x) < f(x*) alors x ← x
19         fin
20     fin
21     retourner x*
22 fin
```

Cette perturbation $\Delta\tilde{g}$ est calculée à chaque fois qu'un voisin x' de la solution courante x est produit en employant la relation suivante : $\Delta\tilde{g}(x, x') = \Delta f(x, x') + \rho$. Le bruit ρ additionné à la vraie variation Δf est une valeur réelle générée aléatoirement dans l'intervalle $[-r, r]$. Le paramètre r, appelé *taux de bruit*, est diminué après avoir exploré un nombre prédéfini de voisins nbV, afin que l'algorithme converge vers f. La méthode de bruitage alterne généralement un certain nombre d'itérations bruitées nbB avec des descentes non bruitées (voir lignes 16-19) afin d'explorer un bon nombre de minima locaux (par rapport à f). Le processus entier s'arrête quand un nombre d'itérations totales

$maxIter$, fixé à l'avance, est atteint. Enfin, la meilleure solution x^* visitée au cours de la recherche est retournée.

Codenotti *et al.* [1996] ont présenté un algorithme légèrement différent appelé méthode de *perturbation*. Les différences par rapport à la méthode de bruitage sont : la quantité de perturbation appliquée à f est fixée (ou dépend de la qualité de la solution courante) et la solution initiale pour la prochaine itération de RL avec $\tilde{g}$ est toujours la meilleure solution trouvée. De plus, cette méthode a seulement été employée pour résoudre le problème du Voyageur de Commerce (PVC).

2.3.2 Méthode de lissage de l'espace de recherche

Quand le paysage de recherche comporte un grand nombre d'optima locaux, les algorithmes de RL ont tendance à rester fréquemment bloqués dans ce type de solutions ce qui produit une baisse de leurs performances. La méthode de *Lissage de l'Espace de Recherche* (LER) [Gu, 1990; Gu, 1994] a été développée comme une alternative pour surmonter cet inconvénient.

La méthode de LER transforme de manière déterministe la fonction objectif f en une fonction d'évaluation $g^{(\alpha)}$ en employant le *facteur de lissage* α ($\alpha \geq 1$), qui contrôle la force d'une opération de lissage. L'idée essentielle est celle de « réduire » le nombre de minima locaux après chaque opération de lissage. Ainsi, la possibilité de stagner dans l'un d'entre eux est diminuée, et par conséquent, la probabilité de trouver l'optimum global augmente. Quand $\alpha = 1$, aucune opération de lissage n'est appliquée, ainsi le paysage de recherche est le même que celui du problème original ($g^{(1)} = f$). Si $\alpha > 1$, le paysage de recherche résultant est plus plat que l'original ; et si $\alpha \gg 1$, l'opération de lissage a un effet plus fort, ayant pour résultat un paysage de recherche presque plat vérifiant $g^{(\infty)}(x) = c,\ \forall x \in \mathscr{X}$, où c est une constante.

Par exemple, Gu et Huang [1994] ont défini $g^{(\alpha)}$ pour le PVC comme suit. Soit n le nombre de villes dans le problème, d_{ij} la distance entre deux villes i et j ($0 \leq d_{ij} \leq 1\ \forall i, j \in \{1..n\}$), et $\bar{d} = \sum_{i \neq j} d_{ij}/(n(n-1))$ la distance moyenne entre toutes villes. Ainsi, une instance simplifiée du PVC peut être définie en transformant la distance entre deux villes i et j avec un facteur de lissage spécifique α en utilisant l'Équation 2.2.

$$d_{ij}(\alpha) = \begin{cases} \bar{d} + |d_{ij} - \bar{d}|^{\alpha}, & d_{ij} \geq \bar{d} \\ \bar{d} - |d_{ij} - \bar{d}|^{\alpha}, & d_{ij} < \bar{d} \end{cases} \tag{2.2}$$

Ensuite, $g^{(\alpha)}$ est définie comme la somme des $d_{ij}(\alpha)$ pour toute arête $\{i, j\}$. Au début, la valeur de α est élevée, et par la suite, elle diminue graduellement jusqu'à atteindre la valeur de 1 en fin d'exécution. Par cette règle, les structures d'une solution qui affectent considérablement le coût sont capturées dans l'étape initiale, et après, les structures détaillées sont graduellement considérées dans les pas ultérieurs. Ces idées sont récapitulées dans l'Algorithme 2.2, qui commence par générer une solution initiale $x \in \mathscr{X}$ et calculer son coût $f(x)$ avec la fonction objectif. Ensuite, une valeur initiale α_0 pour le facteur de lissage est choisie (les auteurs ont employé $\alpha_0 = 10$). À chaque itération de l'algorithme, une instance simplifiée du PVC est générée. Elle comporte un espace de recherche plus plat qui a été produit en calculant les distances entre les villes de l'instance

avec l'Équation 2.2 (voir ligne 7). Ensuite, un algorithme de RL quelconque est employé pour minimiser la fonction objectif f et après, la valeur de α est réduite d'une unité. L'itération suivante travaille avec une instance du PVC plus complexe (espace de recherche avec plus de rugosité) et prend comme configuration initiale la solution produite dans l'itération précédente. Quand α atteint la valeur de 1, la solution rapportée par l'algorithme est ainsi une solution au problème original.

Algorithme 2.2 : Méthode de Lissage de l'Espace de Recherche pour le PVC

```
   Entrées : α_0
 1 début
 2     générer une solution initiale x ∈ 𝒳
 3     α ← α_0
 4     tant que α ≥ 1 faire
 5         pour chaque i ≤ n faire
 6             pour chaque j ≤ n faire
 7                 d[i][j] ← d_ij(α)                ▷ génère un problème simplifié
 8             fin
 9         fin
10         x ← rechercheLocale(d, x)                ▷ retourne la meilleure solution x
11         α ← α − 1
12     fin
13     retourner x
14 fin
```

La méthode de LER a prouvé son efficacité dans les expérimentations menées par Gu et Huang [1994]. Les résultats présentés par les auteurs montrent que la méthode de LER produit des solutions qui sont entre 7% et 31% meilleures que celles fournies par l'algorithme de RL *2-opt* [Lin et Kernighan, 1973]. Partant de ces résultats, differents chercheurs ont contribué à améliorer la méthode de LER en examinant la performance d'autres fonctions de lissage ou d'autre méthodes de RL [Schneider *et al.*, 1997; Coy *et al.*, 2000; Dong *et al.*, 2003].

2.3.3 Méthode de recherche locale guidée

Plus tard, une autre alternative pour aider la RL à sortir des optima locaux et pour distribuer l'effort de recherche a été proposé par Voudouris et Tsang [1995]. Cette métaheuristique, appelée *Recherche Locale Guidée* (RLG), a été construite sur l'expérience des auteurs dans le domaine de la satisfaction de contraintes [Wang et Tsang, 1991; Tsang et Wang, 1992; Davenport *et al.*, 1994], mais aussi à partir du concept de pénalité utilisé dans la recherche opérationnelle [Koopman, 1957; Stone, 1983]. Le principe de base de la RLG est d'augmenter dynamiquement la fonction objectif f avec des pénalités calculées pour certaines caractéristiques de la solution potentielle. Cette nouvelle fonction d'évaluation g (fonction objectif augmentée) est manipulée par la RLG chaque fois que l'algorithme de RL trouve un optimum local. De cette façon le processus de recherche est guidé loin de ces optima locaux.

Afin d'appliquer la RLG, il faut tout d'abord définir des caractéristiques dans les solutions potentielles. Par exemple, Voudouris et Tsang [1999] ont défini comme une *caractéristique* pour le problème du Voyageur de Commerce, l'existence d'une arête l_i dans la solution potentielle de la ville W à la ville Y. Ces caractéristiques ont un coût et une valeur de pénalité associés. Dans ce cas, les auteurs ont défini le coût de la caractéristique l_i comme la distance entre les villes W et Y. Les pénalités sont initialisées à 0 et sont augmentées uniquement quand l'algorithme de RL atteint un optimum local. La fonction d'évaluation g, qui associe à chaque solution potentielle $x \in \mathscr{X}$ une valeur numérique, est définie par l'Équation 2.3 comme suit :

$$g(x) = f(x) + \lambda \sum_{i=1}^{L} (p_i \; I_i(x)) \;\; , \tag{2.3}$$

où L est le nombre de caractéristiques définies pour les solutions, p_i est la pénalité pour la caractéristique l_i, $I_i(x)$ prend la valeur de 1 si x possède la caractéristique i et de 0 autrement, et λ est un paramètre pour l'algorithme de RLG qui contrôle l'importance relative des pénalités par rapport au coût d'une solution. Cette fonction d'évaluation g est manipulée quand l'algorithme de RL rencontre un minimum local particulier x_*. Ceci est fait en incrémentant de 1 les pénalités p_i de toutes les caractéristiques l_i qui maximisent l'expression d'utilité suivante :

$$util(x_*, l_i) = I_i(x_*) \left(\frac{c_i}{1 + p_i} \right) \;\; , \tag{2.4}$$

où c_i est le coût de la caractéristique l_i. Observez que par rapport à cette formule, l'utilité de pénaliser une caractéristique l_i, qui n'est pas présente dans le minimum local, est égale à 0. Grâce à l'Équation 2.4, les caractéristiques sont pénalisées avec une fréquence proportionnelle à leur coût. Autrement dit, plus le coût d'une caractéristique (c_i) est élevé, plus l'utilité de la pénaliser est grande. D'ailleurs, plus une caractéristique a été pénalisée, plus l'utilité de la pénaliser à nouveau est faible.

L'effort de recherche est mieux distribué en employant le coût ainsi que la valeur courante de pénalité pour choisir la caractéristique qui sera pénalisée. Plus acun effort de recherche ne sera donné aux solutions potentielles qui possèdent les caractéristiques avec un coût faible (bonnes caractéristiques). Les pénalités empêchent que tout l'effort de recherche soit dirigé vers les meilleures caractéristiques. L'algorithme de RLG décrit ci-dessus est présenté dans l'Algorithme 2.3.

La méthode de RLG a été appliquée avec succès sur des problèmes bien connus comme celui d'Affectation Quadratique [Mills *et al.*, 2003], de Satisfiabilité [Mills et Tsang, 2000], d'Affectation du Personnel [Tsang et Voudouris, 1997], ou encore d'Affectation de Fréquences Radio [Voudouris, 1997].

2.3.4 Méthode d'évasion

La méthode d'*Évasion* (Breakout Method) développée par Morris [1993] est un algorithme de descente pour résoudre des problèmes de Satisfaction de Contraintes (CSP).

Algorithme 2.3 : Méthode de recherche locale Guidée pour le PVC

Entrées : $\lambda,\ L$
début
  générer une solution initiale $x_0 \in \mathscr{X}$
  $x^* \leftarrow x_0$
  pour chaque $i \leq L$ **faire** $p_i \leftarrow 0$ ▷ `initialise les pénalités à 0`
  $k \leftarrow 1$
  tant que *le critère d'arrêt n'est pas satisfait* **faire**
    $x_{k+1} \leftarrow$ **rechercheLocale**$(x_k,\ g,\ \lambda)$ ▷ `applique RL par rapport à` g `(Éq. 2.3)`
    si $f(x_{k+1}) < f(x^*)$ **alors** $x^* \leftarrow x_{k+1}$
    pour chaque $i \leq L$ **faire**
      $util(x_{k+1}, l_i) = I_i(x_{k+1}) * (c_i/1 + p_i)$
    fin
    pour chaque i *telle que* $util(x_{k+1}, l_i)$ *est maximum* **faire**
      $p_i \leftarrow p_i + 1$
    fin
    $k \leftarrow k + 1$
  fin
  retourner x
fin

Un CSP est généralement exprimé en termes de contraintes satisfaites, toutefois la méthode d'évasion se focalise dans l'ensemble $I = \{c_1, ..., c_m\}$ des contraintes violées par la solution courante. Dans la méthode d'évasion, chaque contrainte violée $c_l \in I$ a un poids associé $W(c_l)$. Tous les poids sont des entiers positifs et leurs valeurs son initialisées à 1. Le coût d'une solution potentielle x est défini par la fonction d'évaluation suivante :

$$g(x) = \sum_{c_l \in I} W(c_l)\ ; \tag{2.5}$$

La méthode d'évasion contient deux pas essentiels : déterminer le changement local qui minimise les conflits, appelé *minimisation de conflits*, et augmenter le poids de chaque contrainte violée, dit *évasion*. La minimisation de conflits consiste à choisir une variable et une nouvelle valeur qui réduit autant que possible les conflits dans la configuration courante. S'il y a une telle combinaison, la combinaison variable/valeur avec la meilleure amélioration est donc choisie comme mouvement local. Si aucun progrès n'est possible, l'algorithme est dans un minimum local. Dans ce cas, l'algorithme incrémente le poids de chaque contrainte violée de 1, et recense une nouvelle fois les améliorations éventuelles. Étant donné que les violations courantes incrémenteront plus leurs poids, par la suite une amélioration de la valeur de conflit sera possible (breakout). Ce processus continue jusqu'à ce qu'un maximum d'itérations prédéfini soit atteint, ou qu'un certain nombre maximum des pas d'évasion soient exécutés. En fin d'exécution, l'algorithme retourne la meilleure solution trouvée.

Cette méthode a été appliquée avec succès à la résolution d'une classe de problèmes SAT et de coloration de graphes [Morris, 1993].

Algorithme 2.4 : Méthode d'Évasion

Entrées : CSP, $maxIter$, $maxEva$
début
  générer une solution initiale $x \in \mathscr{X}$
  initialiser un vecteur de poids $W \leftarrow 1$
  $nbIter \leftarrow 1$
  $nbEva \leftarrow 1$
  tant que *x n'est pas une solution* **et** $nbIter \leq maxIter$ **et** $nbEva \leq maxEva$ **faire**
    si *x n'est pas un minimum local* **alors**
      effectuer des changements locaux pour minimiser les conflits
      $nbIter \leftarrow nbIter + 1$
    sinon
      incrémenter les poids pour toutes les contraintes violées actuellement
      $nbEva \leftarrow nbEva + 1$
    fin
  fin
  retourner x
fin

2.4 Fonctions d'évaluation par apprentissage

Afin d'essayer d'améliorer l'exécution des algorithmes de RL pour l'optimisation combinatoire, des méthodes d'*Apprentissage par Renforcement* (AR) peuvent être utilisées pour construire des fonctions d'évaluation plus efficaces.

Dans la littérature, trois approches différentes employant l'AR pour améliorer l'optimisation combinatoire ont été décrites. La première consiste à collecter des données à partir de nombreuses trajectoires de recherche d'une seule instance d'un problème. Ces données sont ensuite utilisées comme des paramètres d'entrée pour un algorithme d'apprentissage afin de construire une nouvelle fonction d'évaluation. L'algorithme de Boyan et Moore [2000], appelé STAGE, entre dans cette catégorie.

Dans la deuxième approche, une fonction d'évaluation est construite par apprentissage sur des données d'entraînement lors d'une première phase, pour ensuite s'en servir au moment de résoudre de nouvelles instances du même problème. Ce schéma est utilisé dans une application d'AR sur un problème d'ordonnancement de tâches pour les lancements spatiales à la NASA [Zhang et Dietterich, 1995]. Dans les deux approches précédentes, un *état* du problème d'AR est une solution complète (par exemple, un circuit Hamiltonien dans le PVC).

Dans une troisième approche, décrite par Bertsekas et Tsitsiklis [1996], au lieu d'employer la nouvelle fonction d'évaluation pour se déplacer parmi les solutions potentielles, cette fonction est utilisée pour guider la construction de nouvelles solutions potentielles.

Par la suite, nous présentons à titre d'exemple les détails de l'algorithme STAGE.

2.4.1 Algorithme STAGE

L'algorithme STAGE, proposé par Boyan et Moore [2000], comme d'autres méthodes d'AR, suppose que le problème traité peut être formalisé comme un *Processus Décisionnel de Markov* (PDM), pour lequel il existe une fonction d'évaluation spéciale et idéale $V^{\pi}(x)$ connue sous le nom de *fonction de valeur optimale*. Cette fonction retourne la meilleure valeur de coût qui peut être atteinte par une méthode de RL π lors d'une trajectoire de recherche commençant par l'état x. Ceci est valable lorsque toutes les décisions ont été prises de manière optimale pour π.

Dans un PDM possédant un espace de recherche de grande taille, comme c'est le cas habituellement en optimisation combinatoire, l'approche pour calculer la fonction de valeur optimale $V^{\pi}(x)$ consiste à l'estimer en employant une *fonction d'approximation* telle qu'un réseau de neurones [Bertsekas et Tsitsiklis, 1996] ou la régression linéaire [Boyan et Moore, 1994]. Ce genre de méthodes, combinant AR et des fonctions d'approximation, s'est montré être efficace pour résoudre plusieurs problèmes tels que l'ordonnancement de tâches [Zhang et Dietterich, 1995], le jeu de jacquet [Tesauro, 1995], et le contrôle automatique d'ascenseurs [Crites et Barto, 1998].

Dans l'algorithme STAGE, la fonction de valeur optimale V^{π} est estimée (apprise) par un modèle de régression polynomial supervisé qui utilise plusieurs trajectoires de recherche prélevées au fur et à mesure. Dans ce modèle, les états sont codés comme des vecteurs contenant D composantes et sont représentés par des valeurs réelles à l'aide de la fonction de correspondance $F : \mathscr{X} \rightarrow \mathbb{R}^D$. La valeur originale de la fonction objectif $f(x)$ ainsi que toutes les propriétés utiles et spécifiques du problème peuvent être représentées par ces vecteurs. L'approximation de $V^{\pi}(x)$ est notée $\tilde{V}^{\pi}(F(x))$ par Boyan et Moore [2000].

L'algorithme STAGE commence par produire un état initial aléatoire x_0. Ensuite, la méthode de RL π est lancée à partir de x_0 pour optimiser la fonction objectif f. La trajectoire de recherche $\{x_0, x_1, ..., x_T\}$ produite par cette étape est employée en tant que nouvelles données d'apprentissage pour estimer $\tilde{V}^{\pi}$. Ensuite, une optimisation de la fonction $\tilde{V}^{\pi}(F(x))$ est exécutée avec une méthode de RL stochastique qui démarre de l'état x_T. Cette étape produit un nouvel état initial plus prometteur que x_0 pour π. Ce processus itératif se poursuit jusqu'à ce qu'un nombre prédéfini d'états évalués soit atteint. En fin d'exécution, l'algorithme STAGE retourne le meilleur état trouvé.

Cette méthode a été appliquée avec succès à plusieurs domaines d'optimisation : le bin packing (remplissage de conteneurs), le routage dans la conception des circuits VLSI, la recherche de structure de réseaux Bayésiens, la planification de traitements de radiothérapie, la conception cartographique et la satisfiabilité [Boyan et Moore, 2000].

2.5 Fonctions d'évaluation hiérarchiques

Une fonction d'évaluation hiérarchique se compose typiquement de plusieurs sous-fonctions $g = \langle g_1, g_2, ..., g_k \rangle$ qui respectent un ordre de priorité décroissant $g_1 \succ g_2 \succ ... \succ g_k$. Chaque sous-fonction g_i correspond soit à une contrainte relaxée, soit à un objectif du

problème initial. Étant données deux configurations x et y, elles sont comparées avec la relation suivante, où $\succ$ signifie « meilleur que » :

$$g(x) \succ g(y) \equiv \begin{cases} & g_1(x) \succ g_1(y) \\ \text{ou} & g_1(x) = g_1(y) \text{ et } g_2(x) \succ g_2(y) \\ \vdots & \\ \text{ou} & g_1(x) = g_1(y)..., \ g_{k-1}(x) = g_{k-1}(y) \text{ et } g_k(x) \succ g_k(y) \end{cases} \tag{2.6}$$

Par la suite, nous montrons deux exemples qui utilisent ce type de fonctions d'évaluation.

2.5.1 Problème du sac à dos multidimensionnel en variables 0-1

Dans [Vasquez et Hao, 2001b], les auteurs ont utilisé une fonction d'évaluation formée par deux sous-fonctions dans leur algorithme de RT pour le problème du Sac à Dos Multidimensionnel en Variables 0-1 (ou MKP01). De manière informelle, le MKP01 consiste à choisir un sous-ensemble parmi n objets de manière à maximiser le profit total des objets choisis, tout en respectant les contraintes de limitation de chacune des m ressources ($m > 1$). Le MKP01 est formellement énoncé comme suit :

$$\text{MKP01} = \begin{cases} \text{maximiser} & c.x \\ \text{tel que} & A.x \leq b \quad x \in \{0,1\}^n \end{cases} \tag{2.7}$$

avec $c \in \mathbb{N}^{*n}$, $A \in \mathbb{N}^{m \times n}$ et $b \in \mathbb{N}^m$. Les éléments binaires x_j de x sont des variables de décision : $x_j = 1$ si l'objet j est choisi, $x_j = 0$ autrement. c_j est le profit associé à l'objet j. Chacune des m contraintes s'appelle « contrainte de type sac à dos ».

L'algorithme de RT présenté par Vasquez et Hao travaille avec des configurations non réalisables, c'est-à-dire qui violent certaines contraintes de type sac à dos. La qualité d'une configuration x est évaluée en utilisant deux sous-fonctions. La première mesure son degré de non-faisabilité, et la deuxième son profit. Pour évaluer le degré de non-faisabilité de x, la mesure $v_b(x)$ a été introduite :

$$v_b(x) = \sum_{i \,|a_i.x > b_i} (a_i.x - b_i), \tag{2.8}$$

Par conséquent, $v_b(x) = 0$ indique que x est une configuration faisable, tandis qu'une grande valeur de $v_b(x)$ signifie une forte violation des contraintes de sac à dos. Pour confronter deux configurations x et y, on compare d'abord leur degré respectif de non-faisabilité : $v_b(x)$ et $v_b(y)$. Si $v_b(x) = v_b(y)$, alors leurs valeurs de profit $c.x$ et $c.y$ sont employés pour distinguer x et y. Remarquez qu'une priorité plus élevée est naturellement accordée à la sous-fonction de contrainte $v_b()$ afin de favoriser la satisfaction de la contrainte relaxée car sinon, cette contrainte ne pourra jamais être satisfaite.

2.5.2 Problème de positionnement d'antenne

Un deuxième exemple de fonction d'évaluation hiérarchique est présenté dans [Vasquez et Hao, 2001a], où un problème d'optimisation multiobjectifs sous contraintes est résolu en utilisant un algorithme de RT. Le problème traité est tiré du domaine de la planification de réseaux cellulaires, et possède trois contraintes impératives et quatre objectifs d'optimisation (appelons les g_i pour $1 \leq i \leq 4$).

Dans leur implémentation de RT les auteurs ont décidé de relaxer une des trois contraintes (désignons-la h) qui est satisfaite au cours du processus d'optimisation. La raison essentielle de relaxer cette contrainte est qu'il n'existe aucune manière évidente de la satisfaire. Pour évaluer une configuration x, une fonction hiérarchique qui se compose de cinq sous-fonctions $g = \langle h, g_1, g_2, g_3, g_4 \rangle$ est employée, où la priorité la plus élevée est accordée à la sous-fonction correspondant à la contrainte h.

D'autres exemples de ce type de fonctions peuvent être trouvés dans [Russell *et al.*, 2006; Lacomme *et al.*, 2006].

2.6 Fonctions d'évaluation spécifiques au problème traité

Nous avons décrit dans les sections précédentes différentes techniques générales visant à évaluer plus précisément la qualité des solutions potentielles afin de mieux guider la recherche. En effet, ces techniques étant très généralistes, elles peuvent être adaptées de manière relativement simple à une grande classe de problèmes d'optimisation, même si l'on ne dispose pas d'une connaissance approfondie de leurs propriétés mathématiques. Il reste cependant des problèmes pour lesquels une simple adaptation de ces techniques ne garantit pas la qualité des solutions fournies.

Pour de tels problèmes, l'alternative consiste à développer de nouvelles fonctions d'évaluation plus informatives qui sont conçues pour un problème particulier. En général, une certaine forme de connaissance du problème traité est incorporée dans la fonction d'évaluation pour la rendre plus efficace comme dans [Khanna *et al.*, 1994; Torres-Jimenez et Rodriguez-Tello, 2000; Singh et Gupta, 2005]. Il est évident que cette approche demande une étude plus approfondie, du point de vue mathématique, du problème traité. Cette étude qui permet de découvrir les particularités du problème et de les intégrer dans la fonction d'évaluation, n'est pas forcement facile à réaliser. Toutefois, cette approche s'avère extrêmement profitable en fournissant de très bons résultats, comme le montrent les exemples présentés dans la suite de cette section.

2.6.1 Problème de coloration de graphes

Le problème de Coloration de Graphes (PCG) consiste à colorer les sommets d'un graphe donné avec un nombre minimal de couleurs, appelé *nombre chromatique*, selon la contrainte que deux sommets adjacents doivent recevoir deux couleurs différentes. De cette manière, l'espace de recherche est défini par l'ensemble de toutes les affectations de couleurs aux sommets. Pour ce problème, Johnson *et al.* [1991] ont proposé de minimiser la fonction d'évaluation g représentée par l'Équation 2.9. Dans cette équation, le nombre

de couleurs k est variable, V_i représente l'ensemble des sommets colorés par i dans la solution courante x, et E_i ($1 \leq i \leq k$) est l'ensemble des arêtes dont les deux extrémités (sommets) sont dans V_i.

$$g(x) = -\sum_{i=1}^{k} |V_i|^2 + \sum_{i=1}^{k} 2|V_i||E_i| \ . \tag{2.9}$$

Notons que par le premier terme de l'Équation 2.9, un ensemble V_i (sommets colorés avec i) de grande taille tend à obtenir plus de sommets que les ensembles plus petits. Ceci a comme effet secondaire de voir le nombre de couleurs k être minimisé. Le second terme est utilisé pour pénaliser les arêtes dont les deux extrémités sont colorées avec la même couleur. Une autre caractéristique importante est que tous les optima locaux d'après g correspondent aux colorations légales (c'est-a-dire $E_i = \emptyset \ \forall i$). L'avantage principal de la fonction d'évaluation g comparée à la fonction objectif f (c'est-a-dire la somme pondérée de k et de $\sum_{i=1}^{k} |E_i|$) est que g peut évaluer le gain d'un mouvement pour lequel le nombre de couleurs k reste inchangé, mais dont la taille d'un petit ensemble V_i est diminuée. Dans leur article, Johnson *et al.* ont démontré que l'utilisation de la fonction d'évaluation g dans un algorithme de RS mène à de bons résultats, principalement du fait que g est plus informative que f et que g aide à améliorer le guidage de la recherche.

2.6.2 Problème de remplissage de conteneurs (bin packing)

Un autre exemple de fonction d'évaluation conçue pour un problème spécifique est présenté par Falkenauer et Delchambre [1992] pour le problème de *Remplissage de Conteneurs* (PRC). Dans le PRC, il y a une capacité donnée Q pour les conteneurs et une liste de n objets $A = (a_1, a_2, ..., a_n)$, chacun d'une taille $s(a_i) > 0$. L'objectif de ce problème est de ranger les objets en utilisant le moins possible de conteneurs, c'est-à-dire les partitionner dans un nombre minimum m des sous-ensembles $B_1, B_2, ..., B_m$ tels que pour chaque B_j, $\sum_{a_i \in B_j} s(a_i) \leq Q$. Une fonction d'évaluation à priori évidente serait simplement de minimiser le nombre de conteneurs employés pour ranger tous les objets. En pratique, elle est inutilisable car elle n'a aucune capacité pour guider un algorithme dans le processus de recherche. Elle attribue la même valeur de coût à de nombreuses solutions potentielles, même quand ces solutions n'ont pas la même probabilité d'amélioration par la suite.

En fait, elle mène à un paysage de recherche comportant très peu de points optima perdus parmi un nombre exponentiel de solutions potentielles (points), lorsque le coût de la solution courante est juste d'une unité au-dessus de l'optimum [Falkenauer, 1996]. Pour surmonter ces inconvénients, Falkenauer et Delchambre ont proposé de maximiser la fonction d'évaluation suivante pour résoudre le PRC :

$$g(x) = \frac{\sum_{i=1}^{m} (F_i/Q)^k}{m} \ , \tag{2.10}$$

où m est le nombre de conteneurs utilisés dans la solution x, F_i est la somme des tailles des objets à ranger dans le conteneur i et k est une constante ($k > 1$) qui exprime l'im-

portance donnée aux conteneurs les plus remplis par rapport aux moins remplis. Plus la valeur de k est grande, plus de conteneurs bien remplis seront produits.

La fonction d'évaluation proposée (g) maximise l'efficacité moyenne de tous les conteneurs car elle mesure l'exploitation de leurs capacités de rangement. Ainsi, elle encourage le rendement de chaque conteneur, plutôt que la performance de tous les conteneurs ensemble. De plus, g affecte des valeurs semblables (mais pas égales) aux solutions similaires, tout en conservant les mêmes optima que la fonction objectif f, ce qui a pour conséquence des paysages de recherche moins rugueux.

La supériorité de l'AG présenté par Falkenauer et Delchambre [1992] sur d'autres techniques a été confirmée par une expérimentation approfondie. Cette supériorité est en grande partie due à la fonction d'évaluation g.

2.6.3 Problème de routage dans la conception des circuits VLSI

Le *Routage* (channel routing) est un problème de base dans la conception des circuits VLSI [Deutsch, 1976]. Une instance du problème consiste en une grille rectangulaire, appelée *canal*, où tous les points des extrémités sont occupés par des *terminaux* étiquetés. Ces derniers sont séparés en paires appelés « liaisons » (nets). Les terminaux de chaque liaison appartiennent à deux extrémités différentes et doivent être reliés entre eux en plaçant des segments de fil dans les lignes et les colonnes de la grille appelées *pistes*. Les segments peuvent se croiser mais pas se recouvrir. L'objectif de ce problème est de trouver un routage x pour réduire au minimum la *largeur du canal* $w(x)$, autrement dit pour réduire au minimum le nombre de lignes utilisées du canal. Le routage est connu pour être un problème NP-complet [Szymanski, 1985].

La fonction objectif à minimiser est bien sûr la largeur du canal $w(x)$. Cependant, Wong *et al.* [1988] ont remarqué que $w(x)$ était une mesure trop brute de la qualité des solutions potentielles. Au lieu de cela, ils ont proposé pour un routage valide x la fonction d'évaluation suivante :

$$g(x) = w(x)^2 + \lambda_p \cdot p(x)^2 + \lambda_U \cdot U(x) \tag{2.11}$$

où $p(x)$ est la distance du plus long chemin de G_x (un graphe induit par le routage x), λ_p et λ_U sont deux constantes, et $U(x) = \sum_{i=1}^{w(x)} u_i(x)^2$, où $u_i(x)$ est la fraction inoccupée de la piste i.

Wong *et al.* ont utilisé cette fonction d'évaluation, conçue spécialement pour évaluer plus précisément la qualité des solutions potentielles, dans un algorithme de RS en utilisant les valeurs $\lambda_p = 0.5$ et $\lambda_U = 10$. Les résultats obtenus ont montré que la qualité des solutions atteinte avec la fonction $g(x)$, était supérieure à celle obtenue par le même algorithme utilisant uniquement la largeur du canal $w(x)$ comme fonction d'évaluation.

2.7 Synthèse du chapitre

Dans ce deuxième chapitre nous avons observé l'importance de la fonction d'évaluation, non seulement dans l'optimisation combinatoire, mais aussi en l'intelligence ar-

tificielle. En effet, il est indispensable pour tout algorithme de recherche d'utiliser une fonction d'évaluation suffisamment informative, car le choix de la fonction d'évaluation détermine fortement les résultats de la recherche [Nilsson, 1986; Hsu *et al.*, 1990; Beasley *et al.*, 1993; Hoos et Stützle, 2004].

Nous avons également présenté dans ce chapitre une étude systématique de plusieurs techniques proposées dans la littérature, visant à améliorer l'efficacité de guidage des fonctions d'évaluation. Nous avons classé ces fonctions d'évaluation en cinq grands types : avec des pénalités, dynamiques, par apprentissage, hiérarchiques et spécifiques au problème traité.

Au cours de cette étude de synthèse, nous avons distingué deux types de techniques pour la production de fonctions d'évaluation, par rapport à leur adaptabilité au problème traité : les *techniques généralistes* et les *techniques spécifiques*. Les techniques généralistes peuvent être adaptées de manière relativement simple à une grande classe de problèmes d'optimisation. En revanche, les techniques spécifiques nécessitent d'une étude approfondie, du point de vue mathématique, du problème traité. Cette étude permet la découverte de certaines particularités du problème, qui seront ensuite intégrées dans la fonction d'évaluation afin de la rendre plus efficace. Les techniques spécifiques se sont montrées très efficaces pour certains problèmes, cependant elles sont moins nombreuses dans la littérature que les techniques généralistes.

Les chapitres suivants de cette thèse introduisent deux nouvelles fonctions d'évaluation qui ont été spécifiquement développées pour deux problèmes d'étiquetage de graphes NP-difficiles : la *Minimisation de Largeur de Bande* (BMP) et l'*Arrangement Linéaire Minimum* (MinLA). Chaque nouvelle fonction a été créée en intégrant des informations collectées au cours d'une analyse détaillée du problème traité. Les deux nouvelles fonctions sont ensuite comparées expérimentalement aux fonctions utilisées généralement pour résoudre ces problèmes, afin de vérifier l'utilité pratique des fonctions proposées.

Comme nous le verrons ces chapitres illustrent le fait qu'une fonction d'évaluation soigneusement conçue peut augmenter considérablement l'efficacité de recherche des métaheuristiques.

Chapitre 3

Problème de Minimisation de Largeur de Bande des Graphes

LE PROBLÈME de *Minimisation de Largeur de Bande des Graphes* (Bandwidth Minimization Problem, BMP) consiste à trouver une manière d'étiqueter les sommets d'un graphe de sorte que la plus grande différence absolue entre les étiquettes des sommets adjacents soit minimum. Depuis son origine, dans les années cinquante, le problème BMP a suscité l'intérêt de nombreux chercheurs grâce à ses applications dans plusieurs domaines très variés de l'ingénierie et des sciences.

Dans ce chapitre nous présentons formellement le problème BMP, ainsi qu'une brève révision des algorithmes les plus représentatifs pour le résoudre. Nous réalisons ensuite une étude sur la fonction d'évaluation classique β à l'aide d'une analyse effectuée sur l'ensemble de tous les graphes étiquetés à n sommets. Les résultats de cette analyse nous serviront à développer ultérieurement une nouvelle fonction d'évaluation spécifique pour BMP, notée δ, qui prend en compte non seulement la valeur β, mais également d'autres informations supplémentaires liées à chaque étiquetage. Ainsi, cette nouvelle fonction d'évaluation nous permet de distinguer des permutations possédant la même largeur de bande β.

Sommaire

3.1 Définition du problème BMP

Historiquement, le problème de *Minimisation de Largeur de Bande des Matrices* (Matrix Bandwidth Minimization Problem, MBMP) trouve son origine dans les années cinquante. Certains ingénieurs des structures analysaient alors pour la première fois des cadres en acier en manipulant par ordinateur leurs matrices structurales [Lotkin et Remage, 1952; Kosko, 1957; Livesley, 1960]. Les matrices utilisées dans ces applications étaient des matrices 0-1 creuses de grande taille (voir Figure 3.1(a)). Les opérations appliquées sur elles consistaient principalement en des inversions et des calculs de déterminants.

De nombreux efforts ont été réalisés afin de trouver une matrice équivalente dans laquelle toutes les entrées différentes de zéro soient positionnées dans une bande étroite proche de la diagonale principale, d'où le terme de *largeur de bande* [Chinn *et al.*, 1982] (voir Figure 3.1(b)). Les avantages principaux de ce type d'arrangement sont une réduction de l'espace de stockage et une diminution du temps nécessaire pour manipuler une telle matrice. Le premier algorithme pour résoudre le problème MBMP a été proposé par Alway et Martin [1965], puis plusieurs autres ont été publiés [Akyuz et Utku, 1968; Rosen, 1968; Cuthill et McKee, 1969; Cheng, 1973; Collins, 1973].

(a) Avant **(b)** Après

Figure 3.1 – Exemple d'une matrice 0-1 creuse avant et après avoir minimisé la largeur de bande.

Dans les années soixante, Harper [1964] et Harary [1967] ont proposé indépendamment le problème de *Minimisation de Largeur de Bande des Graphes* (BMP). Il est facile d'observer l'équivalence entre les problèmes MBMP et BMP, étant donné qu'une matrice 0-1 creuse $A = \{a_{ij}\}_{n \times n}$ peut être transformée dans un graphe $G(V, E)$ en l'interprétant comme la matrice d'incidence du graphe. De cette manière, le graphe G aura une arête reliant les sommets i et j si et seulement si a_{ij} est différent de zéro.

Avant d'introduire formellement le problème BMP, nous présentons certains concepts qui sont nécessaires à la définition du problème auquel nous nous attachons dans ce chapitre.

Définition 3.1 (Étiquetage pour un graphe) *Un étiquetage pour un graphe $G = (V, E)$ est une fonction bijective de la forme $\varphi : V \rightarrow \{1, 2, ..., n\}$, qui à chaque sommet du graphe associe un nombre entier distinct entre 1 et n. Un étiquetage est appelé aussi un* arrangement linéaire *des sommets du graphe. Pour désigner l'étiquette d'un sommet u nous utilisons la notation $\varphi(u)$.*

La largeur de bande est un paramètre fondamental qui intervient dans la formulation de nombreux problèmes modélisés par des graphes. Elle est définie comme suit :

Définition 3.2 (Largeur de bande) *Soit $G = (V, E)$ un graphe non orienté d'ordre n, et $\varphi : V \rightarrow \{1, 2, ..., n\}$ un étiquetage pour G. La* largeur de bande *β de G pour φ est définie selon l'équation suivante :*

$$\beta(G, \varphi) = \max\{|\varphi(i) - \varphi(j)| : \{i, j\} \in E\} \tag{3.1}$$

Partant de ce concept, nous pouvons à présent définir le problème BMP :

Définition 3.3 (Problème BMP) *Soit $G = (V, E)$ un graphe non orienté d'ordre n. Le problème BMP consiste à trouver un étiquetage φ^* pour lequel la* largeur de bande *$\beta(G, \varphi^*)$, ou coût, est minimum :*

$$\beta(G, \varphi^*) = \min\{\beta(G, \varphi) : \varphi \in \mathscr{L}\}\ , \tag{3.2}$$

avec $\mathscr{L}$ l'ensemble de tous les étiquetages possibles.

Nous pouvons observer qu'un étiquetage peut également être vu comme une *permutation*. Ainsi, il est facile de vérifier qu'un étiquetage φ peut être transformé en un autre étiquetage φ' en lui appliquant au maximum $n - 1$ opérations d'échange entre deux étiquettes.

Puisqu'il y a $n!$ étiquetages possibles pour un graphe avec n sommets, le problème BMP est un problème fortement combinatoire. Papadimitriou a démontré que trouver la largeur de bande minimum d'un graphe est un problème NP-complet [Papadimitriou, 1976]. Ceci signifie qu'il est très peu probable qu'il existe un algorithme capable de trouver systématiquement la largeur de bande minimum en temps polynomial par rapport à la taille du graphe.

Postérieurement, il a été démontré que le problème BMP est NP-complet même pour des arbres avec un degré maximum de trois [Garey *et al.*, 1978]. Il est possible de trouver l'étiquetage optimal en temps polynomial seulement dans certains cas très particuliers (voir par exemple [Garey *et al.*, 1978; Smithline, 1995; Kratsch, 1987; Diaz *et al.*, 2002]).

Le problème BMP est très utile dans un grand nombre d'applications, notamment dans les domaines suivants : stockage de données, conception de circuits VLSI, analyse des réseaux, électromagnétisme industriel [Esposito *et al.*, 1998a], méthodes des éléments finis, systèmes de transport d'énergie à grande échelle, cinétique chimique, géophysique numérique [Piñana *et al.*, 2004], gestion de grandes collections d'hypertextes [Berry *et al.*, 1996], etc.

3.2 Principales approches de résolution pour le problème BMP

En raison de l'importance théorique et pratique du problème BMP, beaucoup de recherche a été effectuée pour développer des algorithmes pour le résoudre. Ces algorithmes peuvent être classés en deux grands groupes : les *méthodes exactes* et les *méthodes*

approchées. Les méthodes exactes garantissent de trouver toujours la largeur de bande optimale. Un algorithme de ce type est présenté par Gurari et Sudborough [1984] ; il résout le problème BMP en $O(n^\beta)$ itérations, où β est la largeur de bande pour un graphe d'ordre n. Deux autres exemples d'algorithmes exacts ont été proposés par del Corso et Manzini [1999]. Cependant, les trois méthodes mentionnées ci-dessus sont très limitées car elles parviennent à résoudre seulement des graphes de moins de cent sommets. En conséquence, il existe une nécessité d'utiliser des algorithmes approchés pour résoudre de grandes instances de ce problème dans des temps de calcul raisonnables.

Les méthodes approchées ou heuristiques sacrifient l'optimalité des solutions pour fournir, en un temps de calcul raisonnable, des solutions sous-optimales de la meilleure qualité possible ; citons à titre d'exemples la méthode GPS [Gibbs *et al.*, 1976] et l'algorithme de Cuthill-McKee [Cuthill et McKee, 1969]. Plus récemment, certaines métaheuristiques ont été appliquées avec succès au problème BMP, comme RS [Dueck et Jeffs, 1995], RT [Martí *et al.*, 2001], une version améliorée de l'algorithme de Cuthill-McKee [Esposito *et al.*, 1998b], AG [Sourd et Schoenauer, 1998; Lim *et al.*, 2006], et GRASP [Piñana *et al.*, 2004].

L'objectif pour le reste de cette section est de faire une brève révision des quatre algorithmes approchés les plus représentatifs pour résoudre le problème BMP, et qui sont ensuite employés pour nos comparaisons expérimentales. Pour une description plus détaillée de ces algorithmes nous invitons le lecteur à se reporter aux articles correspondants.

3.2.1 Méthode GPS

En 1976, Gibbs, Poole et Stockmeyer ont développé une heuristique constructive, appelée *GPS*, pour le problème BMP [Gibbs *et al.*, 1976]. Cette méthode est basée sur une structure de niveaux $L = \{L_1, L_2, ..., L_k\}$, produite en utilisant un algorithme de *Recherche en Profondeur* (Depth First Search). La méthode GPS est composée de trois phases :

- Elle commence par calculer le diamètre du graphe $diam(G)$, c'est-à-dire la plus longue distance entre deux sommets (u, v), puis construit deux structures de niveaux enracinées respectivement aux sommets u et v. Ces structures de niveaux sont créées de la manière suivante : le sommet racine est inséré dans le niveau L_1 et le reste des sommets sont incorporés de telle manière à ce que tous les sommets adjacents soient situés dans le même niveau ou dans deux niveaux contigus. De plus, la cardinalité maximale de tous les niveaux, appelée la largeur de la structure, doit être minimisée. En employant les deux extrémités u et v du diamètre $diam(G)$ comme racines, les structures de niveau résultantes auront une faible largeur, et en conséquence une *profondeur* maximale (le nombre de niveaux dans la structure).
- Les deux structures de niveaux créées précédemment sont combinées dans une nouvelle structure, dont la largeur est habituellement plus petite que celles des structures originales.
- Enfin, la méthode GPS assigne des nombres entiers consécutifs aux sommets du graphe, guidée par la structure de niveaux résultante. Chaque niveau commence par le sommet de plus petit degré. De cette façon, le sommet du niveau L_1 qui a le

plus petit degré aura comme étiquette 1 et le sommet avec le degré le plus élevé du dernier niveau (L_k) aura l'étiquette n.

Dans [Gibbs *et al.*, 1976], des expériences ont été réalisées sur 19 matrices symétriques composées de 68 à 918 colonnes. Les auteurs ont comparé la méthode GPS contre une autre heuristique constructive, l'algorithme *Inversé de Cuthill-McKee* (ICM) [Cuthill et McKee, 1969]. Ces expériences ont montré que la méthode GPS donne en moyenne une largeur de bande légèrement inférieure (donc meilleure) que celle fournie par l'algorithme ICM. Bien que la méthode GPS soit en réalité plus efficace que l'algorithme ICM seulement pour 9 matrices parmi les 19 étudiées, elle est beaucoup plus rapide.

3.2.2 Recuit Simulé

En 1995, Dueck et Jeffs ont implémenté le premier algorithme de RS (RS-DJ) pour le problème BMP [Dueck et Jeffs, 1995]. Cet algorithme représente l'étiquetage d'un graphe par une permutation des sommets $\{1, 2, ..., n\}$. L'algorithme RS-DJ débute par un étiquetage initial et effectue une série d'itérations pour parcourir l'espace de recherche selon un voisinage. À chaque itération, un étiquetage voisin φ' est produit en échangeant aléatoirement deux valeurs dans la permutation courante φ. Le coût de φ' est alors calculé directement en employant l'Équation 3.3.

$$\Delta C = \beta(G, \varphi') - \beta(G, \varphi) \tag{3.3}$$

Si ΔC est négative ou égale à zéro, l'étiquetage voisin φ' est accepté. Dans le cas contraire, on l'accepte avec une probabilité $P(\Delta C) = e^{-\Delta C/T_k}$, où T_k est la température courante déterminée par un schéma de refroidissement.

L'algorithme RS-DJ emploie un schéma de refroidissement géométrique $\mathcal{Q}(T_k) = 0.95(T_k)$ avec une température initiale fixée à $T_i = 1.0$. À chaque palier de température, un nombre maximum $maxMov = 4|E|$ d'étiquetages voisins sont acceptés et un nombre maximum $maxEss = 80(maxMov)$ d'étiquetages voisins peuvent être produits. L'algorithme s'arrête si l'une des deux conditions suivantes est vérifiée : si la température courante atteint une limite ($T_f = 0.0001$), ou si le nombre de configurations acceptées à chaque température tombe au-dessous de la limite de $minAcc = 50$. Les auteurs justifient leur choix des valeurs pour ces paramètres par des ajustements expérimentaux.

Dueck et Jeffs ont examiné la performance de leur algorithme en employant un ensemble de 18 graphes de différents types qui comportent entre 13 et 255 sommets. Les résultats des expériences menées prouvent que l'algorithme RS-DJ est inférieur à la méthode GPS en terme de qualité de solution pour les graphes structurés (grilles, chemins, cercles, etc.). Cependant, RS-DJ surpasse la méthode GPS en terme de qualité de solution sur les arbres ternaires et les graphes aléatoires. En effet, l'algorithme RS-DJ est capable de trouver de meilleurs étiquetages que la méthode GPS pour 11 graphes, mais en des temps de calcul supérieurs. Les auteurs argumentent que la complexité de leur algorithme RS-DJ limite son utilisation, mais il peut être utilisé comme une référence pour comparer la performance d'autres procédures.

3.2.3 Recherche Tabou

Plus récemment, un autre algorithme pour résoudre le problème BMP a été proposé par Martí *et al.* [2001], il s'agit d'un algorithme de RT. Cet algorithme implémente des opérations de mouvement ($mouvement(u, v)$) qui échangent les étiquettes d'une paire de sommets critiques afin de réduire la valeur courante de la largeur de bande $\beta(G, \varphi)$. La liste des sommets critiques et « presque critiques » $C(\varphi)$ est définie par l'Équation 3.4.

$$C(\varphi) = \{v : \max\{|\varphi(v) - \varphi(u)| : u \in A(v)\} \geq \alpha\beta(G, \varphi)\} \ , \tag{3.4}$$

où $A(v)$ est l'ensemble des sommets adjacents à v et $1 > \alpha > 0$.

Dans le but de créer un ensemble de sommets appropriés pour une permutation $S(v)$, les auteurs ont introduit les trois mesures suivantes pour un sommet v et un étiquetage φ.

$$max(v) = \max\{\varphi(u) : u \in A(v)\} \tag{3.5}$$

$$min(v) = \min\{\varphi(u) : u \in A(v)\} \tag{3.6}$$

$$mid(v) = \left\lfloor \frac{max(v) + min(v)}{2} \right\rfloor \tag{3.7}$$

De cette manière, la meilleure étiquette pour v dans l'étiquetage courant φ est $mid(v)$; par conséquent, $S(v)$ peut être défini comme suit :

$$S(v) = \{u : |mid(v) - \varphi(u)| < |mid(v) - \varphi(v)|\} \tag{3.8}$$

La liste de mouvements candidats associée au sommet $v \in C(\varphi)$ est déterminée par l'équation suivante :

$$CL(v) = \{mouvement(u, v) : u \in S(v)\} \tag{3.9}$$

La valeur d'un mouvement $mouvement(u, v)$ est alors calculée comme le nombre de sommets adjacents à v et u (y compris u) pour lesquels la largeur de bande, $B(v) = \max\{\varphi(u) : u \in A(v)\}$, augmente à cause de ce mouvement. L'implémentation de RT de Martí *et al.* utilise une mémoire à court terme afin d'éviter de revenir à des solutions déjà explorées. Plus précisément, une fois que l'opérateur $mouvement(u, v)$ a été appliqué, les étiquettes des sommets v et u sont fixées pendant un certain nombre d'itérations $\mathcal{L}$. Une stratégie de diversification à long terme basée sur la fréquence d'affectation des étiquettes est ajoutée. De plus, l'algorithme incorpore une structure de mémoire pour répertorier l'information concernant les solutions trouvées. Cette information est alors employée pour *relancer* la recherche à partir d'un nouvel étiquetage, qui est construit en considérant non seulement la diversité, mais aussi la qualité de la solution produite.

Les auteurs ont présenté une comparaison expérimentale entre la méthode GPS, l'algorithme RS-DJ et deux versions de leur algorithme de RT (avec et sans le mécanisme de relance). Pour cette comparaison, ils ont employé une série de tests composée de 113 matrices standard tirées de la *Harwell-Boeing Sparse Matrix Collection*[1]. Cette série de tests inclut des matrices symétriques composées de 30 à 1000 colonnes provenant d'une grande variété de disciplines scientifiques et technologiques.

[1] http://math.nist.gov/MatrixMarket/data/Harwell-Boeing

Les expérimentations ont confirmé que l'algorithme de RT fournit une qualité de solution meilleure que l'algorithme RS-DJ en un temps de calcul moindre. La version de base de l'algorithme de RT s'est montrée supérieure à la méthode GPS en terme de qualité de solution et compétitive en termes de temps d'exécution sur les instances de petite taille. La version de RT utilisant le mécanisme de relance est robuste en terme de qualité de solution, avec une déviation moyenne des meilleures solutions connues de 5%.

3.2.4 Méthode GRASP-PR

En 2004, les résultats obtenus en appliquant la méthode GRASP (Greedy Randomized Adaptive Search Procedure) agrémentée d'une stratégie de *Path Relinking* (PR) au problème BMP sont présentés par Piñana *et al.* [2004] (appelons-la GRASP-PR).

La méthode GRASP-PR commence par la création d'un ensemble initial de solutions dites d'élite. En effet, la méthode GRASP est employée pour construire un grand ensemble d'étiquetages à partir desquels les *nbÉlite* meilleurs sont stockés dans l'*ensemble d'élite*. La procédure de PR explore alors toutes les paires d'éléments dans l'ensemble d'élite en utilisant un ordre lexicographique ; à chaque appel de la procédure PR, le plus mauvais élément dans l'ensemble d'élite est remplacé par la solution améliorée qui vient d'être produite. La méthode GRASP-PR s'arrête lorsqu'aucune amélioration ne peut être faite à l'ensemble d'élite.

La méthode GRASP est composée de deux étapes principales : la phase de construction et la phase d'amélioration. Pendant la *phase de construction*, une structure de niveau semblable à celle employée par la méthode GPS, est créée. La *phase d'amélioration* est quant à elle constituée typiquement d'un algorithme de RL. Dans cette implémentation en particulier il s'agit de l'algorithme de RT proposé dans [Martí *et al.*, 2001]. Les auteurs considèrent un ensemble des sommets critiques $C(\varphi)$, l'opérateur $mouvement(u, v)$ et une liste de mouvements candidats $CL(v)$ associée au sommet $v \in C(\varphi)$.

Les principales différences avec l'algorithme de RT proposé par Martí *et al.* sont les suivantes : les sommets « presque critiques » dans l'ensemble $C(\varphi)$ ne sont pas considérés et la valeur d'un mouvement est définie comme la différence entre le nombre de sommets critiques avant et après ce mouvement, c'est-à-dire *coûtMouvement*$(u, v) = C(\varphi) - C(\varphi')$, où φ' est l'étiquetage obtenu en en appliquant $mouvement(u, v)$ à la solution courante φ. De cette manière, une valeur positive de l'expression *coûtMouvement*(u, v) indique que la solution s'améliore car le nombre de sommets critiques diminue, même si la valeur de $\beta(G, \varphi)$ ne peut pas être réduite.

La procédure de PR est utilisée pour produire des nouvelles solutions en explorant les trajectoires qui relient les solutions de haute qualité. La procédure PR est lancée à partir d'une solution de bonne qualité, appelée *solution de départ*, et produit un chemin dans l'espace de voisinage qui mène vers les autres solutions, appelées *solutions de guidage*. Ceci est accompli en sélectionnant des mouvements qui introduisent des attributs contenus dans les solutions de guidage.

Les auteurs présentent une comparaison approfondie entre la méthode GPS, l'algorithme de RT de Martí *et al.* et deux versions de leur méthode GRASP (avec et sans le mécanisme de PR). Pour cette comparaison ils ont employé la même série de tests de

113 matrices utilisée dans [Martí *et al.*, 2001]. Les expériences ont montré que la méthode GRASP-PR est capable d'améliorer les meilleures solutions connues pour 81 des 113 instances, et ainsi surpasser l'algorithme de RT qui a trouvé seulement 41 des meilleures solutions connues.

3.3 Étude de caractéristiques de la fonction d'évaluation β

La majorité des algorithmes que nous venons de présenter évaluent la qualité d'une solution en considérant la variation de la fonction objectif $\beta(G, \varphi)$ (appelons-la seulement β par simplicité). Cependant, employer β comme fonction d'évaluation dans un algorithme de recherche représente un éventuel inconvénient. En effet, β fournit un guidage peu efficace pour la recherche parce que plusieurs solutions potentielles peuvent avoir la même valeur de β, même si elles n'ont pas la même opportunité d'être améliorées dans les itérations ultérieures. Cette remarque sera clarifiée ci-dessous.

Deux exceptions sont rapportées dans la littérature concernant la fonction d'évaluation pour le problème BMP. La première est présentée dans [Dueck et Jeffs, 1995], où les auteurs tiennent compte des cinq plus grandes différences absolues pour évaluer la qualité d'une solution. La deuxième est une fonction d'évaluation, appelée γ, qui considère toutes les arêtes du graphe, mais en raison de sa nature elle ne peut être utilisée que pour des graphes contenant moins de cent-cinquante sommets [Torres-Jimenez et Rodriguez-Tello, 2000]. Il est donc intéressant d'étudier d'autres fonctions d'évaluation plus pertinentes pour résoudre ce problème.

Dans la suite de cette section, nous présentons une analyse détaillée de certaines caractéristiques de la fonction d'évaluation classique β visant à mettre en évidence les inconvénients qu'elle présente. Cette analyse nous fournit également des informations qui nous guident vers le développement d'une nouvelle fonction d'évaluation plus efficace.

Avant de commencer cette analyse, nous introduisons deux définitions importantes qui sont utilisées dans l'étude des graphes étiquetés.

Définition 3.4 (Différence absolue) *Soit φ un étiquetage pour un graphe $G = (V, E)$ d'ordre n. La différence absolue entre les étiquettes de deux sommets adjacents $\{u, v\} \in E$ est définie comme suit : $|\varphi(u) - \varphi(v)| = k$, pour $0 \leq k \leq n - 1$.*

Définition 3.5 (Fréquence d'apparition) *Soit φ un étiquetage pour un graphe $G = (V, E)$ d'ordre n. La fréquence d'apparition d_k d'une différence absolue de valeur k produite par φ est définie par l'équation suivante :*

$$d_k = \sum_{\{u,v\} \in E} l_{uv} \ , \tag{3.10}$$

où l_{uv} est égale à 1 si $|\varphi(u) - \varphi(v)| = k$, et l_{uv} est égale à 0 autrement.

Soit $G = (V, E)$ un graphe non orienté d'ordre n et φ un étiquetage. Nous pouvons observer que ce graphe G peut avoir potentiellement n arêtes réflexives et $n(n-1)/2$ arêtes non réflexives, en conséquence G peut totaliser jusqu'à $n(n+1)/2$ arêtes. Étant

donné que chacune de ces arêtes peut être présente ou absente dans le graphe (le cas d'un graphe vide est également envisagé), le nombre total de graphes étiquetés possibles est exprimé par la Formule 3.11, où $\mathscr{G}$ représente l'ensemble de ces graphes.

$$|\mathscr{G}| = 2^{\frac{n(n+1)}{2}} \tag{3.11}$$

Supposons un graphe étiqueté d'ordre n. Il est facile d'observer que dans ce graphe le nombre maximum de différences absolues est distribué comme suit : 1 de valeur $(n-1)$, 2 de valeur $(n-2)$, et en général k différences absolues de valeur $(n-k)$ pour tout k dans l'intervalle $[1, n]$. Nous pouvons ensuite observer que β peut être également exprimée en fonction des fréquences d'apparition d_k du graphe comme suit.

$$\beta(G, \varphi) = \max\{k : d_k \neq 0, 0 \leq k \leq n-1\} \tag{3.12}$$

Il est donc clair que la fonction d'évaluation β peut prendre n valeurs différentes ($0 \leq \beta \leq n-1$) ; $\beta = 0$ implique que G est soit un graphe vide, soit un graphe composé seulement d'arêtes réflexives. Dans notre analyse, nous considérons même le cas où $\beta = 0$; en effet nous pensons qu'il est important de prendre en compte toutes les arêtes du graphe pour évaluer plus précisément la qualité d'un étiquetage.

Soit $\mathcal{R}_\beta \subseteq \mathscr{G} \times \mathscr{G}$ une *relation d'équivalence* [Rotman, 2003] sur l'ensemble de tous les graphes étiquetés et $x, y \in \mathscr{G}$, alors x est en relation avec y ($x \ \mathcal{R}_\beta \ y$) si et seulement si x et y ont la même valeur β (c'est-à-dire la même différence absolue maximale)[2]. Nous pouvons alors définir une *classe d'équivalence* $[x] = \{y \in \mathscr{G} : x \ \mathcal{R}_\beta \ y\} \subseteq \mathscr{G}$ pour chaque valeur possible de β. Par conséquent, le nombre total de classes d'équivalence $\mathcal{E}_\beta$ sous β, parmi lesquelles l'ensemble $\mathscr{G}$ de tous les graphes étiquetés d'ordre n est divisé, est :

$$\mathcal{E}_\beta = n \tag{3.13}$$

Soit $\omega_\beta(i)$ la cardinalité de la classe d'équivalence sous $\beta = i$. Il est alors simple d'observer que $\omega_\beta(0) = 2^n$, $\omega_\beta(1) = 2^n(2^{(n-1)} - 1)$ et qu'en général :

$$\omega_\beta(i) = (2^{(n-i)} - 1) \prod_{j=0}^{i-1} 2^{(n-j)} \tag{3.14}$$

Afin de vérifier que tous les graphes étiquetés possibles sont pris en compte dans notre analyse, nous pouvons effectuer la somme de toutes les classes d'équivalence et la comparer avec le nombre total de graphes étiquetés $|\mathscr{G}|$. Ceci est réalisé dans l'Équation 3.15 et elle est bien égale à la valeur de $|\mathscr{G}|$ présentée dans l'Équation 3.11.

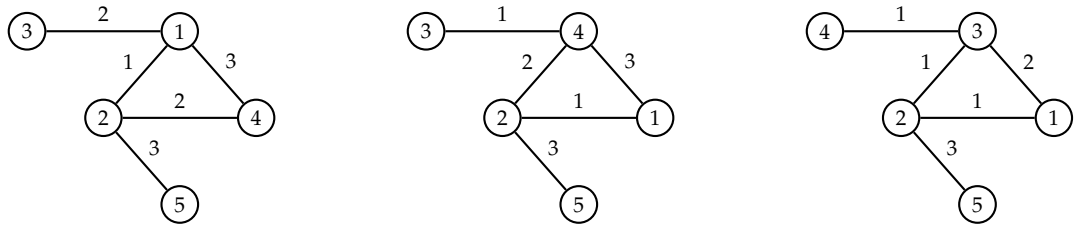
$$\omega_\beta(0) + \sum_{i=1}^{n-1} \omega_\beta(i) = 2^n + \sum_{i=1}^{n-1} \left((2^{(n-i)} - 1) \prod_{j=1}^{i-1} 2^{(n-j)} \right) = 2^{\frac{n(n+1)}{2}} \tag{3.15}$$

[2]Observez qu'il peut s'agir d'un même graphe ou de deux graphes différents.

Il est très important de remarquer que la fonction d'évaluation β ne tient pas compte de toutes les différences absolues du graphe, mais seulement de la plus grande (Équation 3.12). À cause de cette particularité, il n'y a aucune possibilité de faire la distinction entre les $\omega_\beta(i)$ graphes étiquetés potentiels qui appartiennent à la même classe d'équivalence $\beta = i$.

Par exemple, les trois étiquetages (permutations) pour le graphe représenté sur la Figure 3.2 relèvent de la même classe d'équivalence ($\beta = 3$). Cependant, une analyse plus fine de toutes les arêtes du graphe nous permet de confirmer que la permutation φ'' sur la Figure 3.2(c) est peut-être plus intéressante que celles sur les Figures 3.2(a) et 3.2(b) car elle possède seulement une différence absolue de valeur trois, donc pourrait être améliorée plus facilement.

En réponse à cette problématique, nous proposons dans la section suivante une nouvelle fonction d'évaluation pour le problème BMP qui est plus informative. En effet, pour évaluer la qualité d'un étiquetage elle tient compte de trois caractéristiques : la largeur de bande β, chaque différence absolue k du graphe, ainsi que leurs fréquences d'apparition.



(a) Permutation φ **(b)** Permutation φ' **(c)** Permutation φ''

Figure 3.2 – Exemple de trois permutations avec la même valeur de $\beta = 3$.

3.4 Nouvelle fonction d'évaluation δ

L'analyse conduite dans la section précédente nous a permis d'identifier certains inconvénients présentés par la fonction d'évaluation β et en même temps de découvrir des informations sur les particularités du problème traité concernant les fréquences d'apparition d_k.

Nous présentons par la suite une nouvelle fonction d'évaluation spécifique au problème BMP, appelée δ, qui intègre ce type d'information afin de distinguer plus finement les étiquetages analysés [Rodriguez-Tello *et al.*, 2004]. La fonction d'évaluation proposée est définie par l'Équation 3.16, où d_k est la fréquence d'apparition des différences absolues de valeur k et β la largeur de bande pour l'étiquetage (permutation) φ.

$$\delta(G, \varphi) = \beta + \sum_{k=0}^{\beta} \left(\frac{d_k}{\frac{(n+\beta-k+1)!}{n!}} \right) \tag{3.16}$$

Pour illustrer le calcul de cette nouvelle fonction d'évaluation, considérons la permutation φ pour le graphe d'ordre $n = 5$ présenté dans la Figure 3.2(a). Pour cette permutation particulière $\beta = 3$ et les fréquences d'apparition d_k des différences absolues sont : $d_0 = 0$, $d_1 = 1$, $d_2 = 2$, $d_3 = 2$, $d_4 = 0$. En faisant la substitution de ces valeurs dans la Formule 3.16 et en simplifiant, nous obtenons que :

$$\delta(\varphi) = 3 + \frac{1}{336} + \frac{2}{42} + \frac{2}{6} = 3.3839 \tag{3.17}$$

En revanche, si δ est calculée pour la permutation φ' montrée dans la Figure 3.2(b) dont les fréquences d'apparition d_k sont $d_0 = 0$, $d_1 = 2$, $d_2 = 1$, $d_3 = 2$ et $d_4 = 0$ nous obtenons une valeur plus petite :

$$\delta(\varphi') = 3 + \frac{2}{336} + \frac{1}{42} + \frac{2}{6} = 3.3631 \tag{3.18}$$

L'idée principale de la nouvelle fonction d'évaluation δ est de pénaliser de manière plus importante les différences absolues proches de β afin de favoriser les solutions contenant aussi peu de grandes différences que possible. Ceci peut être clairement observé dans l'exemple montré ci-dessus (Équations 3.17 et 3.18), où la fonction δ pénalise les différences absolues de valeur 3 plus que celles de valeur 2 en les multipliant respectivement par un facteur de $\frac{1}{6}$ et $\frac{1}{42}$. De cette manière δ attribuera toujours un coût inférieur à un étiquetage comportant un nombre inférieur de différences absolues de grande valeur comme celui montré dans la Figure 3.2(c) car intuitivement cet étiquetage a une probabilité plus forte d'être amélioré ultérieurement.

En effet, si δ est calculée pour cette permutation φ'' possédant les fréquences d'apparition suivantes : $d_0 = 0$, $d_1 = 3$, $d_2 = 1$, $d_3 = 1$ et $d_4 = 0$, la valeur résultante est encore plus petite :

$$\delta(\varphi'') = 3 + \frac{3}{336} + \frac{1}{42} + \frac{1}{6} = 3.1994 \tag{3.19}$$

Grâce à cette discrimination plus fine de chaque différence absolue produite par un étiquetage, δ est capable de distinguer des permutations ayant la même largeur de bande β.

3.5 Étude de caractéristiques de la nouvelle fonction d'évaluation δ

Nous présentons dans cette section une étude de certaines caractéristiques de la nouvelle fonction d'évaluation δ. Cette étude est similaire à celle présentée dans la Section 3.3 pour la fonction d'évaluation classique β.

Soit $\mathcal{R}_\delta \subseteq \mathscr{G} \times \mathscr{G}$ une *relation d'équivalence* sur l'ensemble de tous les graphes étiquetés et $x, y \in \mathscr{G}$, alors x est en relation avec y ($x \ \mathcal{R}_\delta \ y$) si et seulement si x et y ont le même ensemble de fréquences d'apparition $D = \{d_0, d_1, ..., d_{n-1}\}$ (la même valeur δ). Nous pouvons alors définir une *classe d'équivalence* $[x] = \{y \in \mathscr{G} : x \ \mathcal{R}_\delta \ y\} \subseteq \mathscr{G}$ pour chaque valeur possible de δ.

Rappelons que dans un graphe étiqueté d'ordre n, il y a en général au maximum k différences absolues de valeur $(n-k)$ pour $k \in [1, n]$. En conséquence, les fréquences d'apparition d_k prennent des valeurs comprises entre 0 et $(n-k)$. Ainsi, le nombre total $\mathcal{E}_\delta$ de classes d'équivalence sous δ, parmi lesquelles l'ensemble $\mathscr{G}$ de tous les graphes étiquetés à n sommets est divisé, est :

$$\mathcal{E}_\delta = \prod_{k=0}^{n-1} (n-k+1) = (n+1)! \tag{3.20}$$

Afin de connaître la cardinalité de chaque classe d'équivalence produite par δ, il est nécessaire de calculer le nombre total d'étiquetages possédant le même ensemble de fréquences d'apparition $D = \{d_0, d_1, ..., d_{n-1}\}$. Nous pouvons donc calculer la cardinalité $\omega_\delta(i, D)$ pour la classe d'équivalence $\delta = i$ en utilisant l'équation suivante :

$$\omega_\delta(i, D) = \binom{1}{d_{n-1}}\binom{2}{d_{n-2}}...\binom{n-1}{d_1}\binom{n}{d_0} = \prod_{k=1}^{n}\binom{k}{d_{n-k}} \tag{3.21}$$

Pour démontrer que la somme des cardinalités de toutes les classes d'équivalence est égale au nombre total de graphes étiquetés $|\mathscr{G}|$, il est nécessaire d'utiliser la Formule 3.21 en affectant toutes les valeurs possibles des fréquences d'apparition d_k et d'en faire la somme. Ceci peut être exprimé dans la formule suivante :

$$\prod_{k=1}^{n}\left(\sum_{d_{n-k}}^{k}\binom{k}{d_{n-k}}\right) = \prod_{k=1}^{n} 2^k = 2^{\left(\sum_{k=1}^{n} x\right)} = 2^{\frac{n(n+1)}{2}} \tag{3.22}$$

Étant donné le résultat de l'Équation 3.22, nous pouvons conclure que les $(n+1)!$ classes d'équivalence produites par δ englobent tous les $|\mathscr{G}|$ graphes étiquetés possibles.

L'étude des classes d'équivalence produites par δ, que nous venons de présenter, permet de tirer certaines conclusions importantes. En particulier, nous avons observé le fait que δ divise chaque classe d'équivalence produite par la fonction β en sous-ensembles (classes d'équivalence) de plus petite taille qui regroupent les étiquetages partageant le même ensemble de fréquences d'apparition d_k. C'est grâce à cette manière rationnelle d'incrémenter le nombre de classes d'équivalence que δ permet de distinguer des permutations ayant la même valeur de β. De plus, δ est cohérente par rapport à l'objectif du problème BMP qui consiste à minimiser β : pour deux étiquetages φ et φ' si $\delta(\varphi) < \delta(\varphi')$, alors $\beta(\varphi) \leq \beta(\varphi')$ (voir Équations 3.1 et 3.16).

3.6 Exemple d'application de β et δ

Dans les sections précédentes, nous avons présenté et analysé les deux fonctions d'évaluation β et δ dans le cas général de tous les graphes étiquetés à n sommets. Nous allons maintenant montrer sur un exemple concret ($n = 3$) les différentes classes d'équivalences produites par chacune de ces deux fonctions. Cela nous permettra d'observer

d'une part de quelle manière chaque classe d'équivalence fournie par β est divisée en sous-classes d'équivalence par la fonction δ, et d'autre part comment la fonction δ affecte des valeurs différentes aux permutations de même coût β.

Rappelons que le nombre total de graphes étiquetés à 3 sommets est égal à 64 (Équation 3.11). En utilisant la fonction d'évaluation β ces 64 permutations sont regroupées en $\mathcal{E}_\beta = 3$ classes d'équivalence dont les cardinalités sont 8, 24 et 32 (Table 3.1, colonnes β et $\omega_\beta(i)$). En revanche, la nouvelle fonction δ partitionne chacune de ces 3 classes d'équivalence produites par β en sous-classes d'équivalence de plus petite taille pour totaliser $\mathcal{E}_\delta = 24$. Ce partitionnement est réalisé en fonction des fréquences d'apparition d_k.

Par exemple, la classe d'équivalence sous $\beta = 2$ contient 32 permutations distinctes, alors que sous δ ces permutations sont divisées en 12 classes d'équivalence, dont les cardinalités vont de 1 à 6 (Table 3.1, colonnes δ et $\omega_\delta(i, D)$). De plus, δ affecte une valeur d'évaluation différente pour chacune de ces 12 classes d'équivalence. Ainsi, même si ces 32 permutations ont la même valeur $\beta = 2$, δ juge que certaines permutations sont plus prometteuses que d'autres car elles possèdent un nombre inférieur de différences absolues de grande valeur (voir l'exemple en fin de Section 3.4).

	D						
j	d_2	d_1	d_0	$\omega_\beta(i)$	β	$\omega_\delta(i, D)$	δ
1	0	0	0	8	0	1	0.000
2	0	0	1			3	0.250
3	0	0	2			3	0.500
4	0	0	3			1	0.750
5	0	1	0	24	1	2	1.250
6	0	1	1			6	1.300
7	0	1	2			6	1.350
8	0	1	3			2	1.400
9	0	2	0			1	1.500
10	0	2	1			3	1.550
11	0	2	2			3	1.600
12	0	2	3			1	1.650
13	1	0	0	32	2	1	2.250
14	1	0	1			3	2.258
15	1	0	2			3	2.267
16	1	0	3			1	2.275
17	1	1	0			2	2.300
18	1	1	1			6	2.308
19	1	1	2			6	2.317
20	1	1	3			2	2.325
21	1	2	0			1	2.350
22	1	2	1			3	2.358
23	1	2	2			3	2.367
24	1	2	3			1	2.375

Table 3.1 – Exemple des classes d'équivalence produites par les fonctions d'évaluation β et δ pour les graphes étiquetés d'ordre $n = 3$.

Une question qui se pose naturellement est la suivante : la partition d'une classe d'équivalence de β faite en appliquant la fonction d'évaluation δ est-elle pertinente lors d'une recherche dans l'espace de permutations d'un graphe particulier ?

Dans l'état actuel, nous ne disposons pas de réponse théorique. Néanmoins, intuitivement δ permet de créer un escalier de plusieurs marches de taille réduite à partir d'une grande marche (plateau), ce qui devrait être très utile pour la procédure de recherche. Dans le chapitre suivant, nous montrerons expérimentalement l'utilité pratique de cette nouvelle fonction d'évaluation à l'aide de deux algorithmes de recherche locale et analyserons son influence sur la répartition des solutions au sein d'un voisinage donné.

3.7 Comparaison de complexité entre β et δ

Pour calculer le coût d'une permutation φ en utilisant la fonction d'évaluation classique β, la totalité des arêtes du graphe $G = (V, E)$ doit être analysée (voir 3.1) ; en conséquence, $O(|E|)$ opérations sont requises.

Afin de calculer de manière plus efficace la fonction d'évaluation δ, nous pouvons pré-évaluer chaque terme $(n + \beta - k + 1)!/n!$ de l'Équation 3.16 et le stocker dans une matrice $M = \{m_{ij}\}_{n \times n}$. Ceci nécessite d'exécuter $2|V|$ opérations, c'est-à-dire $O(|V|)$. Ainsi, à chaque fois que la valeur de δ doit être déterminée à nouveau, la somme $\beta + \sum_{k=0}^{\beta}(d_k/m_{\beta k})$ est recalculée, ce qui implique la même complexité algorithmique que celle requise pour calculer β. De plus, la fonction δ permet l'évaluation incrémentale des solutions voisines[3]. Supposons que les étiquettes de deux sommets distincts (u, v) sont permutées, alors nous devrions seulement recalculer les $|A(u)| + |A(v)|$ différences absolues qui changent, où $|A(u)|$ et $|A(v)|$ représentent le nombre de sommets adjacents à u et à v respectivement. Comme nous pouvons l'observer, ceci est plus rapide que les $O(|E|)$ opérations requises originellement.

3.8 Synthèse du chapitre

Au début de ce chapitre, nous avons présenté formellement un important problème d'étiquetage de graphes connu sous le nom de Minimisation de Largeur de Bande (BMP). Nous avons alors étudié les principaux inconvénients présentés par la fonction d'évaluation classique β à l'aide d'une analyse des classes d'équivalence produites par cette fonction.

Une caractéristique importante de cette analyse est qu'elle se fait à un niveau très général du problème traité (l'ensemble de tous les graphes étiquetés à n sommets) et pas seulement sur une instance particulière (un seul graphe). Ceci représente un avantage important car les observations faites à l'aide de cette analyse sont applicables à toutes les instances du problème BMP.

C'est grâce à cette analyse du problème BMP que nous avons observé l'importance de considérer de manière individuelle chaque fréquence d'apparition d_k afin de faire une meilleure distinction des étiquetages évalués. Partant de ces observations, nous avons créé une nouvelle fonction d'évaluation, notée δ, qui est capable de distinguer des permutations de même largeur de bande β.

[3]Notons que β peut être aussi calculée de manière incrémentale.

Dans le chapitre suivant, nous présentons des comparaisons expérimentales entre les fonctions d'évaluation β et δ qui permettent de vérifier l'utilité pratique de la fonction d'évaluation proposée. Un algorithme approché très performant qui tire pleinement profit de la fonction d'évaluation δ est également montré.

Chapitre 4

Comparaisons expérimentales entre les fonctions d'évaluation β et δ

DANS CE chapitre, nous nous sommes attachés à comparer expérimentalement la nouvelle fonction d'évaluation δ (introduite au Chapitre 3) et la fonction classique β pour le problème BMP. Cette étude comparative a deux objectifs essentiels : analyser la distribution de voisins améliorants produits par la nouvelle fonction d'évaluation δ quand elle est couplée à une relation de voisinage donnée, et comparer la performance de δ par rapport à celle de la fonction classique β. Pour atteindre ces objectifs, β et δ ont été implémentées dans deux algorithmes de recherche : la Descente Stricte (DS) et le Recuit Simulé (RS). Les résultats expérimentaux montrent que δ améliore considérablement les performances de ces algorithmes.

Encouragés par ces bons résultats, nous avons décidé de concevoir, dans un second temps, un algorithme de RS plus compétitif pour le problème BMP. Cet algorithme, appelé RSA-δ, tire avantage de la nouvelle fonction d'évaluation δ, mais aussi d'une représentation interne des solutions originale et d'une fonction de voisinage basée sur des mouvements de rotation. Les comparaisons entre RSA-δ et les heuristiques de l'état de l'art, effectuées sur 125 instances de test issues de la littérature, mettent en évidence que RSA-δ est très compétitif. En effet, il permet d'améliorer les meilleurs résultats connus pour de nombreuses instances.

Une partie des travaux présentés dans ce chapitre ont fait l'objet de deux publications [Rodriguez-Tello *et al.*, 2004; Rodriguez-Tello *et al.*, 2006a].

Sommaire

4.1 Instances de test et critères de comparaison

Toutes les expérimentations présentées dans ce chapitre ont été réalisées sur un large panel d'instances d'essai (ou *benchmarks*), décomposé en deux séries de tests différentes.

La première série contient 12 *graphes aléatoires structurés* qui ont été générés selon le modèle proposé par Dueck et Jeffs [1995]. Les graphes sont de tailles diverses et appartiennent à six types différents : chaînes, grilles, cycles, arbres binaires, arbres ternaires et arbres quaternaires. Si nous avons choisi d'utiliser ces instances d'essai dans nos expérimentations, c'est principalement parce qu'elles possèdent des solutions optimales connues [Diaz *et al.*, 2002].

La seconde série comporte 113 matrices de *problèmes réels* provenant d'une grande variété de disciplines scientifiques et technologiques qui ont été réunies par Martí *et al.* [2001] à partir de la collection *Harwell-Boeing*[1]. Les instances dans cette série de test sont divisées en deux sous-ensembles. Le premier comporte 33 *petites instances* contenant entre 30 et 199 sommets. Le deuxième est constitué de 80 *grandes instances* dont les tailles varient entre 200 et 1000 sommets. Nous avons sélectionné ces instances issues de problèmes réels car elles sont aussi utilisées par d'autres chercheurs pour des expérimentations [Piñana *et al.*, 2004; Lim *et al.*, 2003; Lim *et al.*, 2006].

Pour évaluer la performance des fonctions d'évaluation nous montrons des résultats comparatifs sur les différentes instances de test. Le critère de comparaison qui est retenu pour ces comparaisons est celui utilisé généralement dans la littérature : la *largeur de bande minimum* atteinte. Le temps de calcul est également donné à titre indicatif.

4.2 Comparaison avec un algorithme de descente stricte

Dans cette section nous effectuons une première comparaison expérimentale entre la nouvelle fonction d'évaluation δ et la fonction d'évaluation conventionnelle β. À cette fin, nous avons développé un algorithme de Descente Stricte (DS).

4.2.1 Descente stricte

Le choix de la DS pour effectuer cette première comparaison expérimentale est entièrement justifié par le fait que cet algorithme n'utilise pas de paramètres. Par conséquent, il permet une comparaison directe des deux fonctions d'évaluation sans biais. Rappelons que la DS, utilisant une stratégie de mouvement de la meilleure amélioration, a été présentée sous forme de pseudo-code dans l'Algorithme 1.1. Les détails de son implémentation sont présentés par la suite de cette section.

Représentation et fonction d'évaluation

Supposons un graphe $G = (V, E)$ d'ordre n, l'espace de recherche $\mathscr{L}$ est composé des $n!$ permutations possibles (étiquetages). Dans notre algorithme de DS une permutation

[1] http://math.nist.gov/MatrixMarket/data/Harwell-Boeing

φ est représentée par un vecteur l contenant les n premiers nombres entiers. Ce vecteur est indexé par les sommets du graphe, de cette manière la k-ème valeur $l[k]$ exprime l'étiquette affectée au sommet k. Le coût d'une permutation φ est estimé en employant soit la fonction d'évaluation β (Équation 3.1), soit la fonction d'évaluation δ (Équation 3.16).

Solution initiale

La solution initiale pour notre algorithme de DS est générée aléatoirement.

Fonction de voisinage

Soit $swap(\varphi, u, v)$ une fonction permettant d'échanger les étiquettes de deux sommets u et v dans une permutation φ. Le voisinage $\mathcal{N}_1(\varphi)$ d'une permutation φ peut être alors défini comme suit :

$$\mathcal{N}_1(\varphi) = \left\{\varphi' \in \mathscr{L} : swap(\varphi, u, v) = \varphi',\ u, v \in V,\ u \neq v\right\} \tag{4.1}$$

Ce voisinage présente l'avantage de permettre l'évaluation incrémentale[2] des solutions voisines en mettant à jour uniquement les différences absolues dk des sommets concernés par le mouvement ($A(u)$ et $A(v)$).

Condition d'arrêt

L'algorithme commence à partir d'une solution initiale $\varphi \in \mathscr{L}$, puis à chaque itération il évalue la totalité des voisins $\mathcal{N}_1(\varphi)$ pour choisir le meilleur, uniquement s'il améliore la solution courante. Dans le cas contraire l'algorithme s'arrête.

4.2.2 Conditions expérimentales

Le but de cette expérience est d'analyser la distribution des voisins améliorants produits par notre nouvelle fonction d'évaluation δ quand elle est couplée à une relation de voisinage donnée. En effet, nous considérons que le nombre de voisins améliorants produits par une fonction d'évaluation peut être un indicateur intéressant de sa capacité à explorer efficacement l'espace de recherche.

Bien évidement d'autres études plus élaborées sur la topologie de l'espace de recherche, comme celles réalisées pour d'autres problèmes d'optimisation comme celui du Voyageur de Commerce [Kirkpatrick et Toulouse, 1985; Stadler, 1992; Fonlupt *et al.*, 1998], de Coloration de Graphes [Hertz *et al.*, 1994] et de Satisfiabilité [Frank *et al.*, 1997], pourraient être effectuées pour le problème BMP. Cependant, pour comparer les différentes fonctions d'évaluation, l'analyse de la distribution des voisins améliorants proposée dans cette section nous a semblé suffisante.

Pour effectuer cette étude expérimentale, l'algorithme de DS présenté dans la Section 4.2.1 a été implémenté en C, appelons-le DS-β ou DS-δ selon la fonction d'évaluation qui

[2]Sans parcourir toutes les arêtes du graphe.

est employée. L'algorithme, compilé avec *gcc* utilisant l'option d'optimisation *-O3*, est exécuté séquentiellement sur un cluster de dix nœuds. Chaque nœud est composé d'un processeur Xeon cadencé à 2 GHz disposant de 1 Go de mémoire vive sous Linux.

La méthodologie utilisée tout au long de cette première expérimentation est la suivante. Dix solutions aléatoires (étiquetages) ont été produites pour chacun des douze graphes aléatoires structurés appartenant à la première série de test. Ces étiquetages ont été ensuite utilisés comme solutions initiales pour les vingt exécutions effectuées pour comparer les deux fonctions d'évaluation. Au cours de ces vingt exécutions des informations statistiques sur la distribution des solutions voisines produites par chaque fonction comparée ont été collectées.

4.2.3 Résultats expérimentaux

Les résultats moyens produits par les vingt exécutions de cette comparaison sont récapitulés dans la Table 4.1. Les trois premières colonnes de cette table indiquent le nom du graphe, son nombre de sommets et d'arêtes. Les colonnes quatre à neuf correspondent au nombre total d'itérations I, au meilleur coût atteint en terme de largeur de bande C et au temps de calcul moyen T en secondes pour les algorithmes DS-β et DS-δ. La colonne dix présente le nombre moyen de voisins améliorants $\mathcal{N}_A$ trouvés par DS-δ, à la même itération lorsque le nombre moyen de voisins améliorants fournis par DS-β est nul (c'est-à-dire quand DS-β s'arrête). La dernière colonne Δ_C montre le gain en terme de pourcentage du coût moyen atteint par DS-δ par rapport à celui produit par DS-β ($100 * (1 -$ coût de DS-δ/coût de DS-β)).

			DS-β			DS-δ				%
Graphe	$\|V\|$	$\|E\|$	I	C	T	I	C	T	$\mathcal{N}_A$	Δ_C
Path100	100	99	8.250	64.750	0.004	268.300	11.388	1.573	157.600	82.413
Path150	150	149	10.600	98.200	0.021	593.700	15.457	11.015	143.550	84.259
Cycle100	100	99	5.600	70.500	0.000	259.650	12.389	1.585	75.300	82.428
Cycle150	150	149	10.800	96.000	0.021	588.000	16.062	11.245	101.700	83.269
TreeB63	63	62	7.050	38.800	0.000	135.100	8.268	0.225	33.850	78.691
TreeB127	127	126	10.700	80.800	0.015	443.450	14.975	6.121	63.200	81.466
TreeT40	40	39	4.700	24.300	0.000	53.100	7.398	0.031	33.700	69.557
TreeT121	121	120	8.000	79.550	0.008	358.100	16.875	4.264	132.200	78.788
TreeQ85	85	84	6.400	54.800	0.000	165.350	14.969	0.657	72.500	72.684
TreeQ205	205	204	7.100	154.600	0.027	494.050	30.813	26.839	229.200	80.069
Grid100	100	180	7.400	74.000	0.007	203.950	21.069	1.547	114.100	71.529
Grid225	225	420	7.450	188.100	0.054	631.200	41.542	54.073	262.050	77.915
Moyenne				85.367			17.600			78.589

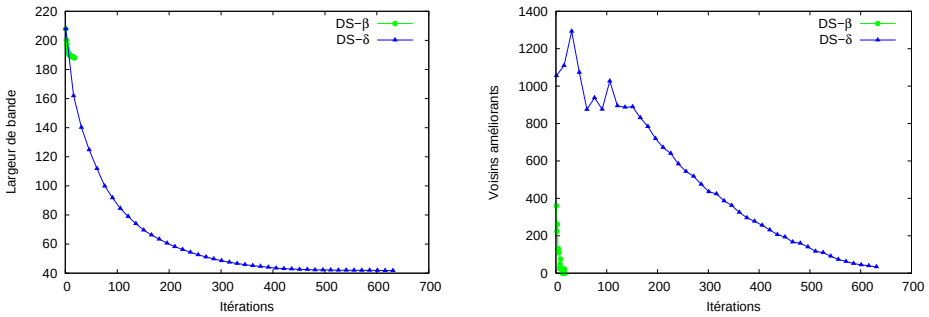
Table 4.1 – Comparaison de performance entre les algorithmes DS-β et DS-δ.

L'information présentée dans la Table 4.1 montre que l'algorithme DS-δ retourne, de manière consistante, de meilleurs résultats que l'algorithme DS-β. En effet, l'utilisation de δ comme fonction d'évaluation permet d'obtenir une amélioration moyenne de 78.589%, ce qui mène à une réduction significative de la largeur de bande pour certains graphes. Nous remarquons que DS-β arrête le processus de recherche dans tous les cas avant l'algorithme DS-δ (en comparant les colonnes quatre et sept). Ce comportement est dû à

l'impossibilité de β de distinguer des solutions voisines de même coût, et par conséquent de guider la recherche vers des voisins améliorants (voir la colonne dix).

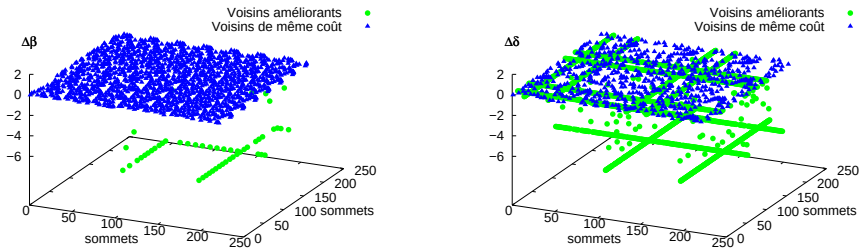
Concernant le temps d'exécution de DS-δ, on observe qu'il est en moyenne 9 fois plus long que celui de DS-β. C'est le prix à payer pour les améliorations en terme de largeur de bande obtenues par DS-δ.

La domination de δ est mieux illustrée sur la Figure 4.1, où nous présentons le comportement des fonctions d'évaluation étudiées sur l'instance $Grid225$ (le reste des instances étudiées fournissent des résultats semblables). Dans la Figure 4.1(a) l'axe des abscisses représente le nombre d'itérations, alors que l'axe des ordonnées indique la qualité moyenne de solution. De plus, la Figure 4.1(b) illustre l'évolution du nombre moyen de voisins améliorants (l'axe des ordonnées) par rapport au nombre d'itérations. Nous pouvons observer à partir de ces deux figures que DS-δ fournit de meilleurs résultats que DS-β en effectuant le même nombre d'itérations.



(a) Qualité moyenne de solution **(b)** Nombre moyen de voisins améliorants

Figure 4.1 – Comparaison de performance entre DS-β et DS-δ sur l'instance $Grid225$.

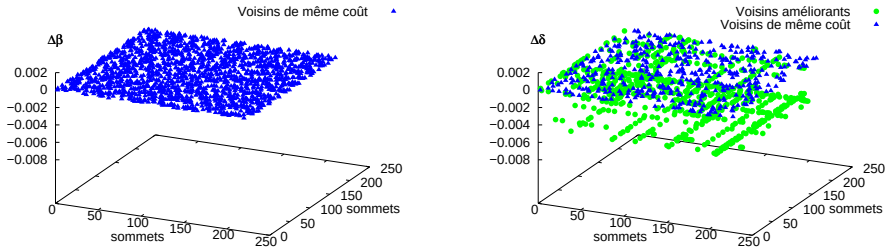


(a) Algorithme DS-β **(b)** Algorithme DS-δ

Figure 4.2 – Comparaison entre les voisins produits par DS-β et DS-δ sur l'instance $Grid225$ après 5 itérations.

Pour compléter cette illustration, nous montrons sur les Figures 4.2(a) et 4.2(b) l'ensemble de voisins de même coût et de voisins améliorants produits après 5 itérations des algorithmes DS-β et DS-δ sur l'instance $Grid225$. Nous constatons une différence très

nette, DS-β affecte la même valeur de coût à un grand nombre de voisins, alors que DS-δ fait une distinction plus fine de ces solutions. Cet inconvénient de la fonction d'évaluation β est beaucoup plus marqué à mesure que la recherche avance. Par exemple, les Figures 4.3(a) et 4.3(b) montrent l'ensemble de voisins produits par les deux algorithmes lors de la dernière itération qui précède l'arrêt de DS-β. Notez comment δ discrimine toutes ces solutions voisines de même coût, qui sont impossibles à distinguer en utilisant la fonction β.



(a) Algorithme DS-β **(b)** Algorithme DS-δ

Figure 4.3 – Comparaison entre les voisins produits par DS-β et DS-δ sur l'instance $Grid225$ lors de la dernière itération qui précède l'arrêt de DS-β.

4.3 Comparaison avec un algorithme de recuit simulé

Après avoir étudié les caractéristiques de la fonction d'évaluation δ, en utilisant un simple algorithme de DS, nous avons choisi de mesurer sa performance dans un algorithme de recherche un peu plus avancé. Nous avons donc programmé un Recuit Simulé (RS) à cette fin, dont les détails d'implémentation sont montrés dans cette section.

4.3.1 Recuit simulé

Nous avons décidé de conserver cette implémentation de l'algorithme de RS aussi simple que possible. De cette manière, nous sommes en mesure d'obtenir une idée plus précise de l'efficacité des fonctions d'évaluation comparées. En effet, nous utilisons une représentation interne standard, un voisinage simple, et une génération aléatoire de la solution initiale. En outre, un schéma de refroidissement géométrique a été utilisé, afin de réduire la forte influence d'autres schémas plus sophistiqués.

Représentation, fonction d'évaluation et de voisinage

La représentation ainsi que les fonctions d'évaluation et de voisinage sont les mêmes que celles utilisées par l'algorithme de DS présenté dans la Section 4.2.1.

Schéma de refroidissement

Nous employons dans cette implémentation un schéma de refroidissement géométrique $\mathcal{Q}(T_k) = \alpha(T_k)$, où α représente le taux de refroidissement. À chaque palier de température, un nombre maximum $maxMov = \mu|E|$ d'étiquetages voisins sont acceptés et un nombre maximum $maxEss = \mu(maxMov)$ d'étiquetages voisins peuvent être visités (mouvements). Ces deux paramètres dépendent directement de la quantité d'arêtes du graphe (E) et d'une constante μ que nous appelons *facteur de mouvements*, puisqu'un nombre important de mouvements sont nécessaires pour des graphes denses.

Condition d'arrêt

Notre algorithme de RS s'arrête si l'une des deux conditions suivantes est vérifiée : si la température courante atteint sa valeur minimale T_f, ou si le nombre de configurations acceptées à chaque température est inférieur à $minAcc$, appelé *seuil minimum d'acceptation*.

4.3.2 Conditions expérimentales

Pour cette deuxième étude de comparaison expérimentale, l'algorithme de RS présenté ci-dessus a été codé en langage *C*, compilé et optimisé (option $-O3$) avec *gcc* sur la machine mentionnée en Section 4.2.2. Nous appelons l'algorithme RS-β ou RS-δ selon la fonction d'évaluation qu'il utilise.

Les valeurs des paramètres du RS ont été fixées empiriquement en considérant les travaux présentés dans [Kirkpatrick *et al.*, 1983; Dueck et Jeffs, 1995; Torres-Jimenez et Rodriguez-Tello, 2000], et de manière à ce que l'algorithme soit performant, robuste et avec des temps de calcul raisonnables. Les valeurs choisies sont :

- température initiale $T_i = 1.0$E-03
- température finale $T_f = 1.0$E-09
- taux de refroidissement $\alpha = 0.92$
- facteur de mouvements $\mu = 15$
- seuil minimum d'acceptation $minAcc = 25$

Chaque algorithme ainsi paramétré a été exécuté vingt fois sur les douze instances aléatoires de la première série d'essai (voir Section 4.1). Les résultats obtenus au cours de ces vingt exécutions ont ensuite été utilisés pour le calcul de certaines mesures statistiques.

4.3.3 Résultats expérimentaux

La Table 4.2 donne, pour chaque instance utilisée et chaque algorithme comparé (RS-β et RS-δ), le meilleur coût trouvé en terme de largeur de bande C, le coût moyen $moy.$, son écart type $e.t.$, ainsi que le temps de calcul moyen d'une exécution en secondes T. Cette table précise également d'une part la solution optimale C^* de chaque instance [Diaz *et al.*, 2002], et d'autre part le gain Δ_C en terme de pourcentage du meilleur coût atteint par RS-δ par rapport à celui produit par RS-β ($100 * (1 -$ coût de RS-δ/coût de RS-β)).

Les résultats fournis dans la Table 4.2 mettent clairement en évidence les bénéfices apportés par l'utilisation de la fonction d'évaluation δ. Nous remarquons que l'algorithme RS-δ obtient de meilleurs résultats que l'algorithme RS-β dans toutes les instances d'essai (voir colonne Δ_C). En effet, l'algorithme RS-δ obtient des solutions optimales pour toutes les instances, alors que RS-β est capable de produire la solution optimale C^* seulement pour l'instance $TreeT40$. En outre, remarquons que l'emploi de δ comme fonction d'évaluation n'augmente pas le temps de calcul de l'algorithme pour ces instances.

				RS-β				RS-δ				%
Graphe	$\|V\|$	$\|E\|$	C^*	C	$moy.$	$e.t.$	T	C	$moy.$	$e.t.$	T	Δ_C
Path100	100	99	1	10	10.100	0.308	13.760	1	1.200	0.410	11.432	90.000
Path150	150	149	1	15	15.600	0.503	28.402	1	1.400	0.503	23.694	93.333
Cycle100	100	99	2	10	10.600	0.503	13.842	2	2.200	0.410	11.378	80.000
Cycle150	150	149	2	15	15.800	0.768	28.156	2	2.600	0.821	23.974	86.667
TreeB63	63	62	7	8	8.100	0.308	6.736	7	7.000	0.000	5.714	12.500
TreeB127	127	126	11	15	15.600	0.503	20.042	11	11.050	0.224	18.652	26.667
TreeT40	40	39	7	7	7.150	0.366	2.812	7	7.000	0.000	3.434	0.000
TreeT121	121	120	17	17	17.800	0.410	15.574	15	15.000	0.000	18.456	11.765
TreeQ85	85	84	14	15	15.050	0.224	7.104	14	14.000	0.000	8.592	6.667
TreeQ205	205	204	26	30	30.600	0.503	31.986	26	26.000	0.000	39.452	13.333
Grid100	100	180	10	15	15.800	0.410	23.640	10	10.000	0.000	21.218	33.333
Grid225	225	420	15	30	31.200	1.196	44.876	15	15.000	0.000	42.944	50.000
Moyenne				16	16.117		19.744	9	9.371		19.078	42.022

Table 4.2 – Comparaison de performance entre les algorithmes RS-β et RS-δ.

Le comportement des algorithmes RS-β et RS-δ, en terme de convergence, est illustré sur la Figure 4.4. Elle montre l'évolution de la largeur de bande des algorithmes sur l'instance $Grid100$ par rapport au nombre de mouvements exécutés (l'axe des abscisses). Nous pouvons remarquer sur cette figure que RS-δ réduit la largeur de bande presque sans interruption, tandis que RS-β reste plus longtemps bloqué. De plus, la largeur de bande finale atteinte par RS-δ est sensiblement plus petite que celle retournée par RS-β. Ce comportement est aussi vérifié pour les autres instances de cette série d'essai.

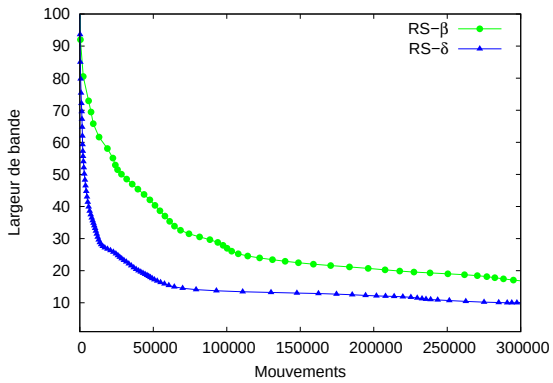


Figure 4.4 – Comparaison de performance entre les algorithmes RS-β et RS-δ sur l'instance $Grid100$.

4.4 Recuit simulé amélioré pour le problème BMP

Encouragés par les bons résultats obtenus dans les expérimentations des Sections 4.2.3 et 4.3.3, nous avons décidé de développer un algorithme de RS amélioré pour le problème BMP. L'objectif est de produire un algorithme capable d'être concurrentiel avec les heuristiques les plus efficaces rapportées dans la littérature.

4.4.1 Détails d'implémentation

L'algorithme de RS amélioré pour le problème BMP que nous proposons a le mérite de perfectionner trois caractéristiques clés qui impactent de manière très importante sa puissance de recherche. En effet, notre algorithme tire avantage non seulement de la nouvelle fonction d'évaluation δ, mais aussi d'une représentation interne des solutions originale et d'une fonction de voisinage basée sur des mouvements de rotation. Par la suite nous mettons en avant les détails les plus importants de cette implémentation que nous appelons RSA-δ.

Représentation interne

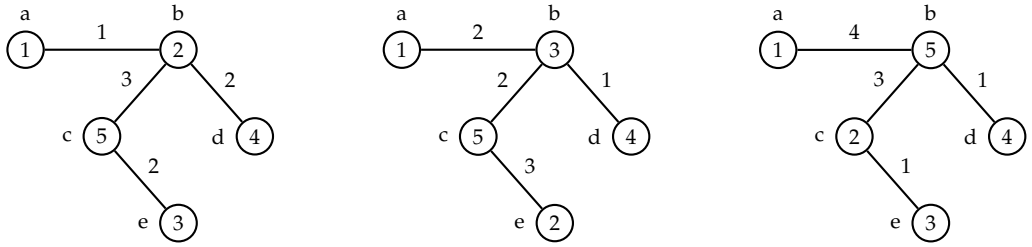
Soit $G = (V, E)$ un graphe d'ordre n et $\varphi : V \rightarrow \{1, 2, ..., n\}$ sa fonction d'étiquetage. Un étiquetage φ (une permutation) peut alors être représenté par un vecteur l contenant n nombres entiers, et indexé par les étiquettes affectées aux sommets du graphe, de telle sorte que la k-ème valeur $l[k]$ exprime le sommet contenant l'étiquette k (voir Figure 4.6(a)). Cette représentation est différente de celle utilisée dans les algorithmes présentés dans les Sections 4.2.1 et 4.3.1, mais aussi des autres méthodes rapportées [Dueck et Jeffs, 1995; Martí *et al.*, 2001; Lim *et al.*, 2003; Piñana *et al.*, 2004].

La nouvelle représentation que nous proposons possède une importante caractéristique : étant donné que les positions contiguës dans le vecteur représentent les étiquettes du graphe avec une différence absolue unitaire, un échange de deux éléments adjacents produit une légère modification de la permutation, parce que la contribution de ces étiquettes à la largeur de bande changera au maximum d'une unité. De plus, comme nous le verrons plus tard, cette représentation utilisée simultanément avec la fonction de voisinage permet d'accélérer l'évaluation des solutions voisines.

Ce concept sera mieux compris avec l'exemple suivant. Considérons le graphe montré sur la Figure 4.5(a). Il se compose de cinq sommets identifiés par des lettres. L'étiquetage courant φ est représenté comme un nombre à l'intérieur de chaque sommet. Nous avons également indiqué sur chaque arête la différence absolue respective. Supposez que nous prenons les étiquettes contiguës 2 et 3 (c'est-à-dire le deuxième et le troisième élément dans la nouvelle représentation) pour les échanger. Alors, l'étiquette 2 sera affectée au sommet e et l'étiquette 3 sera attribuée au sommet b. L'étiquetage résultant φ' est illustré sur la Figure 4.5(b). Il est important de noter que les changements dans les différences absolues ont été au maximum d'une unité (voir par exemple les arêtes (c, b) et (c, e)).

En revanche, si la représentation classique du problème BMP est utilisée, et si nous échangeons les étiquettes de deux sommets contigus b et c (aussi le deuxième et troisième

élément dans cette représentation), alors nous obtenons l'étiquetage φ'' montré sur la Figure 4.5(c). Observez que les valeurs des différences absolues ont été fortement changées. Cet échange a même augmenté la largeur de bande du graphe (voir l'arête (a, b)).



(a) Permutation φ **(b)** Permutation φ' **(c)** Permutation φ''

Figure 4.5 – Trois permutations différentes pour un graphe d'ordre cinq avec une largeur de bande de : (a) $\beta = 3$, (b) $\beta = 3$, et (c) $\beta = 4$.

Fonction de voisinage

Soit $swap(\varphi, u, v)$ une fonction permettant d'échanger les étiquettes de deux sommets u et v dans une permutation $\varphi \in \mathscr{L}$, $L = \{1, 2, ..., n\}$ l'ensemble des étiquettes d'un graphe $G = (V, E)$ d'ordre n, et φ^{-1} la correspondance réciproque d'un étiquetage, autrement dit, une fonction associant à chaque nombre d'étiquette k le sommet du graphe qui la contient ($l[k]$ dans notre représentation). Alors, le voisinage $\mathcal{N}_2(\varphi)$ d'une permutation φ peut être défini comme suit :

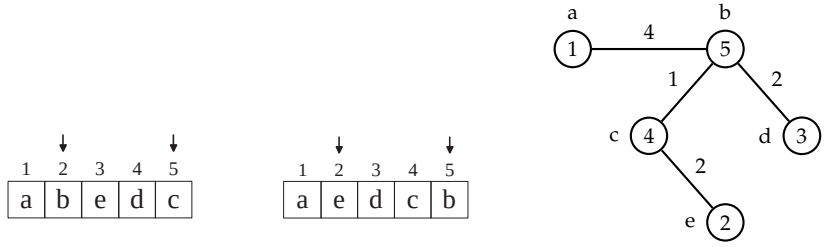
$$\mathcal{N}_2(\varphi) = \left\{\varphi' \in \mathscr{L} : rotation(\varphi, i, j) = \varphi',\ i, j \in L,\ i < j\right\} \tag{4.2}$$

où $rotation(\varphi, i, j)$ est un mouvement de rotation entre les étiquettes i et j qui peut être exprimé comme le produit de $(j - i)$ opérations de $swap$ comme suit.

$$\begin{aligned} rotation(\varphi, i, j) \quad = \quad & swap(\varphi, \varphi^{-1}(i), \varphi^{-1}(j)) * swap(\varphi, \varphi^{-1}(i), \varphi^{-1}(j-1)) * \\ & swap(\varphi, \varphi^{-1}(i), \varphi^{-1}(j-2)) * ... * \\ & swap(\varphi, \varphi^{-1}(i), \varphi^{-1}(i+1)) \end{aligned} \tag{4.3}$$

Nous allons présenter un exemple graphique de rotation pour clarifier ce concept. Dans la Figure 4.6(a) l'étiquetage φ pour le graphe de la Figure 4.5(a) contenant cinq sommets est présenté en utilisant la représentation interne proposée ci-dessus. Supposez que l'on sélectionne les étiquettes 2 et 5 comme les extrémités d'une rotation. Alors, en appliquant la définition montrée dans l'Équation 4.3, chaque sommet $\varphi^{-1}(k)$ pour $3 \leq k < 5$ est déplacé d'une position à gauche dans le vecteur. Ensuite, le sommet qui occupait l'emplacement 2 est situé dans la position 5. Nous montrons l'étiquetage voisin résultant φ' dans la Figure 4.6(b) et le graphe correspondant dans la Figure 4.6(c).

On constate dans cet exemple que grâce à la représentation interne proposée en Section 4.4.1, l'échange entre deux sommets qui possédaient des étiquettes $\varphi^{-1}(k)$ pour $3 \leq k < 5$ produit une légère modification de la permutation (au maximum d'une unité). Cependant, on reconnaît que ce changement est maximum dans le sommet qui été situé originalement à l'extrême gauche de la rotation (sommet b). Observez le cas des arêtes (b, a) et (b, c), où les différences absolues passent respectivement de 1 à 4 et de 3 à 1.



(a) Étiquetage φ **(b)** Étiquetage φ' **(c)** Étiquetage φ'

Figure 4.6 – Voisinage basé sur des mouvements de rotation. (a) étiquetage original φ. (b) étiquetage voisin $\varphi' = rotation(\varphi, 2, 5)$. (c) graphe voisin.

De cette manière, une rotation peut être vue comme une combinaison de deux types de mouvements : le premier qui produit des modifications légères à une permutation, et le deuxième qui entraîne des changements plus importants. Nous avons décidé d'utiliser ce type de fonction de voisinage, parce qu'il est désormais connu que les mouvements composés peuvent mener à de meilleures méthodes de RL que celles qui emploient seulement des mouvements simples [Lin et Kernighan, 1973; Glover, 1996; Cavique *et al.*, 1999]. Les résultats expérimentaux présentés en Section 4.4.4 nous ont permis de confirmer la meilleure performance du mouvement $rotation$ par rapport au mouvement conventionnel $swap$.

Fonction d'évaluation

Pour cette implémentation nous avons choisi d'employer la nouvelle fonction d'évaluation δ, car, comme nous venons de le constater, elle permet de capturer même la plus petite amélioration qui oriente la recherche vers des zones prometteuses de l'espace de recherche. De plus, elle peut être calculée de manière incrémentale, ce qui réduit considérablement les temps de calcul utilisés.

Solution initiale

La solution initiale pour notre algorithme RSA-δ est générée aléatoirement.

Schéma de refroidissement

Dans la littérature on trouve de nombreux schémas de refroidissement, par exemple [Aarts et Van Laarhoven, 1985; Hajek, 1988; Huang *et al.*, 1986; Varanelli et Cohoon, 1999].

Pour l'algorithme RSA-δ nous avons choisi d'utiliser la même approche de réduction de la température par paliers décrite en Section 4.3.1, c'est-à-dire un schéma de refroidissement géométrique. Cette décision est principalement basée sur la simplicité de ce schéma de refroidissement. Nous allons montrer, plus tard dans la Section 4.4.4, que grâce aux trois caractéristiques clés présentées ci-dessus, notre algorithme RSA-δ produit des résultats remarquables, ceci malgré la simplicité du schéma de refroidissement utilisé.

Condition d'arrêt

L'algorithme RSA-δ s'arrête si la température courante atteint sa valeur minimale T_f, ou si le nombre de configurations acceptées à chaque température est inférieur au seuil minimum d'acceptation $minAcc$.

4.4.2 Paramétrage

Il est bien connu que les performances d'un algorithme de RS sont en général très sensibles au paramétrage utilisé. Par la suite nous présentons donc la méthode que nous avons employée pour mettre au point de tels paramètres.

D'abord un sous-ensemble de dix instances représentatives a été sélectionné de la série de tests présentée dans [Martí *et al.*, 2001]. Toutes les expérimentations présentées dans cette section ont été accomplies sur ces dix instances.

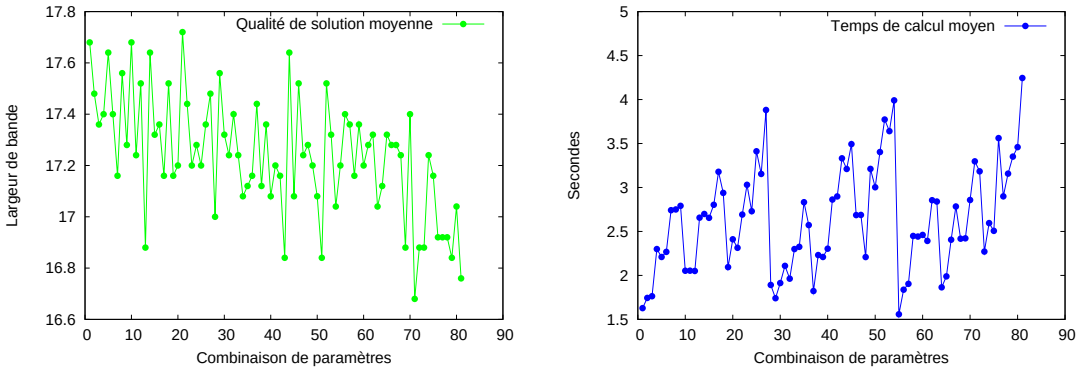
En prenant en compte des expériences préliminaires, nous avons déterminé un intervalle raisonnable pour les valeurs de chaque paramètre de notre algorithme. Vingt exécutions ont été effectuées avec chaque combinaison de valeurs et chaque instance d'essai. Enfin, les données ainsi générées sont analysées afin d'obtenir la combinaison de valeurs (pour ces paramètres) qui rapporte les meilleures performances pour l'algorithme RSA-δ. Les paramètres utilisés dans cette étude expérimentale sont présentés par la suite. Pour chacun d'entre eux nous indiquons également l'intervalle de leurs valeurs et l'incrément utilisé S.

- Température initiale $4.0\text{E-}04 \leq T_i \leq 1.2\text{E-}02$, $S = 2.0\text{E-}03$
- Température finale $8.0\text{E-}10 \leq T_f \leq 1.6\text{E-}09$, $S = 2.0\text{E-}10$
- Taux de refroidissement $0.88 \leq \alpha \leq 0.96$, $S = 0.02$
- Facteur de mouvements $10 \leq \mu \leq 14$, $S = 1$
- Seuil minimum d'acceptation $10 \leq minAcc \leq 30$, $S = 5$

Ces intervalles avec ces incréments produisent un total de 3125 combinaisons de valeurs que nous avons testées. Dans la Figure 4.7(a) nous représentons la largeur de bande moyenne atteinte sur les dix instances d'essai avec les 81 combinaisons de valeurs les plus prometteuses, tandis que la Figure 4.7(b) montre le temps de calcul moyen.

Les graphes présentés ci-dessus nous ont permis de repérer plus facilement les dix meilleures combinaisons de paramètres. La largeur de bande et le temps de calcul moyens atteints pour ces dix combinaisons sont récapitulés dans la Table 4.3. Nous remarquons de cette table que la combinaison de valeurs qui donne la meilleure qualité de solution dans un temps de calcul raisonnable est la suivante :

- température initiale $T_i = 1.0\text{E-}02$



(a) **(b)**

Figure 4.7 – Résultats des expérimentations pour paramétrer l'algorithme RSA-δ.

- température finale $T_f = 1.0\text{E-}09$
- taux de refroidissement $\alpha = 0.92$
- facteur de mouvements $\mu = 12$
- seuil minimum d'acceptation $minAcc = 25$

Ces valeurs seront donc utilisées pour les expérimentations présentées dans la section suivante.

T_i	T_f	α	μ	$minAcc$	C	T
4.0E-03	8.0E-10	0.94	13	15	17.72	2.314
4.0E-03	8.0E-10	0.94	13	20	17.28	2.730
8.0E-03	1.4E-09	0.90	11	20	17.24	2.109
8.0E-03	1.2E-09	0.94	12	15	17.36	2.210
8.0E-03	1.0E-09	0.90	12	25	16.84	3.332
8.0E-03	8.0E-10	0.92	12	25	17.64	3.211
1.0E-02	1.2E-09	0.90	11	15	17.20	1.558
1.0E-02	**1.0E-09**	**0.92**	**12**	**25**	**16.68**	**3.297**
1.0E-02	8.0E-10	0.90	13	20	16.92	3.561
1.0E-02	1.4E-09	0.94	13	25	16.76	4.245

Table 4.3 – Résultats des dix meilleures combinaisons de valeurs pour paramétrer l'algorithme RSA-δ.

4.4.3 Conditions expérimentales

L'objectif principal de cette expérience est d'évaluer l'efficacité de notre algorithme RSA-δ par rapport aux heuristiques les plus performantes pour le problème BMP. Pour atteindre le but de cette troisième phase d'expérimentation, l'algorithme RSA-δ implémenté en langage *C*, a été compilé sur Linux avec *gcc* en utilisant un niveau d'optimisation *-O3*. Cette démarche ainsi que les exécutions de test ont été réalisées sur le même cluster de dix nœuds que nous avons utilisé précédemment (voir Section 4.2.2).

L'algorithme ainsi implémenté a été paramétré selon les valeurs obtenues dans l'étude effectuée en Section 4.4.2. En raison de sa nature incomplète et non déterministe, vingt

exécutions indépendantes de l'algorithme RSA-δ ont été effectuées sur les 113 instances de la deuxième série de test (voir Section 4.1). Par la suite, nous présentons les résultats obtenus au cours de ces vingt exécutions et nous les comparons à ceux produits par d'autres méthodes de la littérature.

4.4.4 Résultats expérimentaux

Les Tables 4.4 et 4.5 présentent une comparaison instance par instance entre notre algorithme RSA-δ et deux métaheuristiques fortement efficaces : RT [Martí *et al.*, 2001] et GRASP-PR [Piñana *et al.*, 2004]. En effet, RT et GRASP-PR produisaient les meilleurs résultats sur les 113 instances de test utilisées pour cette expérience avant l'introduction de RSA-δ. Dans les deux tables, les deux premières colonnes indiquent le nom du graphe et son nombre de sommets. Les colonnes trois à six montrent le coût en terme de largeur de bande C et le temps d'exécution en secondes T rapportés pour les algorithmes RT et GRASP-PR. Ces résultats ont été repris de [Piñana *et al.*, 2004; Martí, 2004] et correspondent à une seule exécution des algorithmes sur chaque instance, en utilisant un processeur Athlon K-7 cadencé à 1.2 GHz. Afin de faire une comparaison la plus équitable possible, nous présentons dans les colonnes sept à onze le meilleur C_m, le pire C_p, la moyenne $moy.$ et l'écart type $e.t.$ du coût trouvé par RSA-δ en terme de largeur de bande sur vingt exécutions, ainsi que le temps moyen d'exécution en secondes T. Les deux dernières colonnes mettent en avant la meilleure solution connue C^*, sans tenir compte des résultats fournis par nos algorithmes et le gain Δ_C en terme de pourcentage du meilleur coût atteint par RSA-δ par rapport à celui produit par RSA-β ($100 * (1 -$ coût de RSA-δ/coût de RSA-β)).

De la Table 4.4 nous pouvons constater que globalement les résultats de RT et GRASP-PR, obtenus en une seule exécution, sont légèrement meilleurs que le pire coût C_p atteint par RSA-δ sur vingt exécutions (23.33 et 22.52 contre 23.36). Cette différence est réduite quand le coût moyen trouvé par RSA-δ (colonne $moy.$) est comparé avec celui de GRASP-PR. Elle est même transformée en une amélioration de 3.05% par rapport à l'algorithme RT. Enfin, la meilleure solution C_m trouvée par RSA-δ dépasse la meilleure solution connue C^* pour 11 des 33 instances (voir colonne Δ_C).

Les données présentées dans la Table 4.5 nous permettent d'observer que la pire solution C_p trouvée par RSA-δ (97.4) est meilleure que celle atteinte par RT et GRASP-PR (100.8 et 99.4), ce qui représente une amélioration moyenne de 3.31% et 2.00% respectivement. Ces améliorations augmentent à 5.04% et 3.75% quand le coût moyen $moy.$ de RSA-δ (95.7) est consideré. Pour conclure, nous observons que pour 46 des 80 instances d'essai la meilleure solution connue C^* est améliorée par RSA-δ.

En ce qui concerne les temps d'exécution des algorithmes comparés, nous voulons remarquer qu'ils ne peuvent pas être confrontés directement car les machines utilisées pour les expérimentations sont différentes. Cependant, pour avoir une idée de l'importance de ces différences, nous avons obtenu du site internet « Tom's hardware guide »[3] une comparaison de processeurs réalisée sur 3300 instances de test.

[3] `http://www.tomshardware.fr`

		RT		GRASP-PR		RSA-δ						%
Graphe	$\|V\|$	C	T	C	T	C_m	C_p	$moy.$	$e.t.$	T	C^*	Δ_C
pores_1	30	7	0.3	7	0.3	7	9	8.0	0.9	3.1	7	0.0
ibm32	32	12	0.2	11	0.3	11	13	11.8	0.8	0.3	11	0.0
bcspwr01	39	5	0.1	5	0.1	5	6	5.8	0.4	0.4	5	0.0
bcsstk01	48	16	0.9	16	1.0	16	20	18.6	1.4	0.6	16	0.0
bcspwr02	49	7	0.2	7	0.6	7	8	7.8	0.4	0.2	7	0.0
curtis54	54	10	0.7	10	0.7	10	10	10.0	0.0	0.5	10	0.0
will57	57	7	0.4	7	0.4	6	6	6.0	0.0	1.1	7	14.3
impcol_b	59	21	1.3	21	1.2	20	23	21.0	1.3	1.2	21	4.8
steam3	80	7	0.7	7	1.0	7	7	7.0	0.0	8.9	7	0.0
ash85	85	10	0.7	9	0.4	9	9	9.0	0.0	1.1	9	0.0
nos4	100	10	1.1	10	1.4	10	10	10.0	0.0	0.9	10	0.0
gent113	104	27	6.3	27	1.0	27	27	27.0	0.0	3.9	27	0.0
bcsstk22	110	11	1.1	10	1.6	11	11	11.0	0.0	5.5	10	-10.0
gre_115	115	25	2.4	24	3.2	23	24	23.6	0.5	1.6	24	4.2
dwt_234	117	11	1.2	11	1.9	11	13	12.0	0.9	1.1	11	0.0
bcspwr03	118	11	1.7	11	0.9	10	11	10.6	0.5	1.2	11	9.1
lns_131	123	21	3.4	22	2.6	20	21	20.6	0.5	1.8	21	4.8
arc130	130	63	4.8	63	1.9	63	64	63.8	0.4	23.2	63	0.0
bcsstk04	132	38	16.7	37	5.4	37	42	37.6	1.2	79.2	37	0.0
west0132	132	36	3.4	35	8.5	33	36	34.0	1.1	5.4	35	5.7
impcol_c	137	33	3.5	31	4.5	30	31	30.6	0.5	3.1	31	3.2
can_144	144	14	1.7	14	3.1	13	13	13.0	0.0	17.6	14	7.1
lund_a	147	25	8.8	23	5.4	23	23	23.0	0.0	40.6	23	0.0
lund_b	147	26	5.3	23	5.5	23	23	23.0	0.0	40.8	23	0.0
bcsstk05	153	23	4.7	20	7.1	20	22	20.8	1.0	10.6	20	0.0
west0156	156	39	7.4	37	8.4	36	37	36.8	0.4	2.8	37	2.7
nos1	158	7	1.3	3	2.6	3	3	3.0	0.0	1.1	3	0.0
can_161	161	20	4.0	18	0.7	18	19	18.8	0.4	3.0	18	0.0
west0167	167	35	5.8	35	5.6	34	35	34.4	0.5	4.8	35	2.9
mcca	168	38	23.9	37	10.8	37	38	37.4	0.5	81.8	37	0.0
fs_183_1	183	63	32.4	61	11.8	61	66	62.6	2.1	8.7	61	0.0
gre_185	185	22	6.2	22	6.1	22	24	22.2	0.5	6.8	22	0.0
will199	199	70	12.0	69	26.9	65	67	65.7	0.8	6.1	69	5.8
Moyenne		23.33	4.99	22.52	4.03	22.06	23.36	22.62	0.52	11.18	22.48	1.65

Table 4.4 – Comparaison des performances sur 33 instances de petite taille entre RT, GRASP-PR et RSA-δ.

Cette information nous a permis d'observer que la machine utilisée pour exécuter les algorithmes RT et GRASP-PR est approximativement 27% plus lente que la nôtre. En utilisant ce facteur d'équivalence sur le temps de calcul de notre algorithme nous avons la possibilité de le comparer avec ceux des algorithmes RT et GRASP-PR.

En particulier, nous observons que pour les 33 instances de petite taille un temps moyen d'exécution de 4.99 et 4.03 secondes est utilisé par les algorithmes RT et GRASP-PR, alors que le temps corrigé avec le facteur d'équivalence de RSA-δ est de 14.19 secondes. En ce qui concerne les 80 instances de grande taille les temps moyens de calcul sont de 264.0 et 340.0 secondes pour RT et GRASP-PR. En revanche, le temps corrigé de RSA-δ est de 253.0 secondes. En résumé, les temps d'exécution de notre algorithme RSA-δ restent compétitifs.

Dans la littérature du problème BMP, les performances des algorithmes sont souvent comparées sous la forme de meilleurs résultats globaux sur tout un ensemble d'instances de test. La Table 4.6 met en avant ce type de comparaison non seulement de RSA-δ, RT

		RT		GRASP-PR		RSA-δ						%
Graphe	$\|V\|$	C	T	C	T	C_m	C_p	$moy.$	$e.t.$	T	C^*	Δ_C
impcol_a	206	35	5.5	34	3.4	32	33	32.6	0.5	5.4	34	5.9
dwt_209	209	25	10.8	24	1.3	23	27	23.8	0.9	29.2	24	4.2
gre_216a	216	22	7.2	21	9.5	21	21	21.0	0.0	3.5	21	0.0
dwt_221	221	15	5.0	13	7.0	13	13	13.0	0.0	23.4	13	0.0
impcol_e	225	44	10.7	42	4.0	42	43	42.2	0.4	49.6	42	0.0
saylr1	238	16	4.3	15	8.5	14	14	14.0	0.0	2.3	14	0.0
steam1	240	50	16.9	46	12.7	44	47	45.2	1.2	79.1	46	4.3
dwt_245	245	25	7.5	26	14.4	24	25	24.2	0.4	9.3	25	4.0
nnc261	261	26	8.6	25	22.4	25	25	25.0	0.0	8.7	25	0.0
bcspwr04	274	26	9.6	26	5.2	25	25	25.0	0.0	28.0	26	3.8
ash292	292	24	7.9	22	8.7	21	21	21.0	0.0	34.4	22	4.5
can_292	292	41	19.0	42	7.9	41	46	42.5	1.8	61.5	41	0.0
dwt_310	310	13	15.2	12	8.6	12	13	12.2	0.4	13.0	12	0.0
gre_343	343	29	16.6	29	21.2	28	28	28.0	0.0	7.4	28	0.0
dwt_361	361	16	11.8	15	0.8	14	14	14.0	0.0	8.3	14	0.0
plat362	362	38	24.8	36	3.4	36	39	36.5	1.1	189.4	36	0.0
plskz362	362	19	27.4	20	7.1	19	24	21.2	2.4	20.9	19	0.0
str_0	363	124	120.8	124	119.1	119	121	120.0	0.9	39.9	124	4.0
str_200	363	136	90.9	135	114.1	128	135	132.0	2.7	47.3	135	5.2
str_600	363	143	180.3	144	92.0	132	134	132.8	0.8	55.9	143	7.7
west0381	381	164	113.0	159	185.4	153	157	154.4	1.4	38.1	159	3.8
dwt_419	419	32	23.7	29	28.0	26	26	26.0	0.0	59.9	29	10.3
bcsstk06	420	47	47.6	50	50.6	45	45	45.0	0.0	247.9	47	4.3
bcsstm07	420	50	40.3	48	113.7	48	50	48.4	0.8	215.7	48	0.0
impcol_d	425	44	21.9	42	30.5	42	49	43.6	2.8	77.5	42	0.0
hor_131	434	60	26.0	64	27.7	55	56	55.4	0.5	154.1	60	8.3
bcspwr05	443	29	24.7	35	16.3	30	32	31.4	0.8	26.8	29	-3.4
can_445	445	56	64.5	58	47.2	56	63	57.8	2.7	114.0	56	0.0
pores_3	456	17	16.8	13	3.2	13	15	14.2	1.0	13.0	13	0.0
bcsstk20	467	21	27.8	19	11.0	14	18	14.9	1.1	44.6	19	26.3
nos5	468	67	60.8	69	103.1	64	64	64.0	0.0	121.9	67	4.5
west0479	479	130	81.2	127	163.0	123	125	123.6	0.8	40.5	127	3.1
mbeacxc	487	270	3409.1	272	5464.5	262	264	263.0	0.9	1774.0	270	3.0
mbeaflw	487	270	3409.4	272	5467.8	261	263	261.8	0.8	1744.9	270	3.3
mbeause	492	260	2637.6	269	5494.3	255	260	256.6	1.9	1289.5	260	1.9
494_bus	494	32	29.2	35	13.5	32	37	33.8	1.8	24.4	32	0.0
west0497	497	90	78.3	92	88.7	87	89	88.2	0.8	137.9	90	3.3
dwt_503	503	45	99.3	45	7.9	44	47	44.7	0.9	163.5	45	2.2
lns_511	503	48	74.2	49	58.3	44	46	44.8	0.8	123.8	48	8.3
gre_512	512	37	77.1	36	92.7	36	36	36.0	0.0	14.7	36	0.0
fs_541_1	541	270	54.4	270	26.5	270	270	270.0	0.0	82.4	270	0.0
sherman4	546	29	33.0	27	4.1	27	27	27.0	0.0	11.5	27	0.0
dwt_592	592	33	52.9	33	106.8	32	33	32.4	0.5	123.7	33	3.0
steam2	600	80	242.2	65	182.5	65	67	66.0	0.9	687.4	65	0.0
nos2	638	9	43.7	3	15.7	3	5	3.8	0.8	114.5	3	0.0
west0655	655	171	150.1	167	245.8	161	169	162.1	2.0	80.5	167	3.6
662_bus	662	42	113.8	44	32.9	43	44	43.2	0.4	70.4	42	-2.4
shl_0	663	235	153.1	241	110.2	229	231	230.2	0.8	211.5	235	2.6
shl_200	663	245	161.3	247	98.4	235	241	238.4	2.0	213.5	245	4.1
shl_400	663	243	188.3	242	121.2	235	239	237.0	1.5	221.4	242	2.9
nnc666	666	45	138.5	45	55.8	42	43	42.2	0.4	108.5	45	6.7
nos6	675	21	70.9	16	42.8	16	16	16.0	0.0	12.5	16	0.0
fs_680_1	680	19	43.4	17	33.6	17	19	18.4	0.8	39.8	17	0.0
saylr3	681	53	107.2	52	77.8	53	57	54.4	1.5	15.6	52	-1.9

suite dans la page suivante...

... suite de la page précédente

		RT		GRASP-PR		RSA-δ						%
Graphe	$\|V\|$	C	T	C	T	C_m	C_p	$moy.$	$e.t.$	T	C^*	Δ_C
sherman1	681	53	107.2	52	78.0	52	57	53.4	1.9	15.6	52	0.0
685_bus	685	38	90.4	46	12.7	36	38	37.0	1.0	62.0	38	5.3
can_715	715	74	183.1	78	14.2	74	78	75.1	1.4	223.0	74	0.0
nos7	729	73	74.5	66	89.9	65	65	65.0	0.0	23.9	65	0.0
mcfe	731	135	800.2	130	247.2	126	127	126.2	0.4	1868.9	130	3.1
fs_760_1	760	40	101.1	39	97.9	38	38	38.0	0.0	95.6	39	2.6
bcsstk19	817	20	174.3	16	86.6	15	16	15.2	0.4	222.9	16	6.3
bp_0	822	252	386.8	258	483.0	240	251	241.7	2.7	140.7	252	4.8
bp_1000	822	313	886.7	297	886.2	291	295	292.2	1.4	216.2	297	2.0
bp_1200	822	320	674.8	303	897.1	296	299	297.2	1.2	218.2	303	2.3
bp_1400	822	327	521.0	313	741.2	300	306	302.8	2.5	217.4	313	4.2
bp_1600	822	322	546.8	317	783.7	299	308	300.6	2.8	231.8	317	5.7
bp_200	822	281	315.2	271	550.3	267	272	268.7	1.8	168.4	271	1.5
bp_400	822	288	355.7	285	560.6	276	282	277.8	2.3	180.0	285	3.2
bp_600	822	299	480.9	297	556.8	279	287	281.8	2.9	193.3	297	6.1
bp_800	822	305	520.9	307	636.9	286	289	286.9	1.2	219.9	305	6.2
can_838	838	91	158.9	88	37.4	88	89	88.2	0.4	384.2	88	0.0
dwt_878	878	31	195.0	35	99.6	26	26	26.0	0.0	106.7	27	3.7
orsirr_2	886	95	203.0	91	42.5	88	89	88.4	0.5	164.5	91	3.3
gr_30_30	900	44	282.2	58	85.4	45	49	46.6	1.7	370.1	44	-2.3
dwt_918	918	37	290.8	36	12.6	33	36	34.4	1.0	243.0	36	8.3
jagmesh1	936	31	150.6	27	107.8	27	27	27.0	0.0	33.8	27	0.0
nos3	960	76	143.8	79	209.1	62	64	63.0	0.6	811.6	75	17.3
jpwh_991	983	94	312.6	96	65.1	94	103	96.6	3.5	104.3	94	0.0
west0989	989	228	372.7	217	416.9	215	217	216.0	0.9	172.1	217	0.9
dwt_992	992	64	272.5	49	306.4	35	36	35.6	0.5	116.4	35	0.0
Moyenne		100.8	264.0	99.4	340.0	94.8	97.44	95.70	0.97	199.25	97.98	2.82

Table 4.5 – Comparaison des performances sur 80 instances de grande taille entre RT, GRASP-PR et RSA-δ.

et GRASP-PR, mais aussi de l'algorithme GPS [Gibbs *et al.*, 1976] et RS-DJ [Dueck et Jeffs, 1995]. En effet, cette comparaison tient compte du meilleur C_m, du pire C_p, et de la moyenne $moy.$ du coût trouvé par RSA-δ en terme de largeur de bande sur vingt exécutions. Ces résultats sont confrontés à ceux atteints par RT, GRASP-PR, GPS et RS-DJ. Pour chaque algorithme il est indiqué la largeur de bande moyenne sur chacun des sous-ensembles d'instances de test, le temps de calcul moyen en secondes, ainsi que l'écart type par rapport à la meilleure solution trouvée par toutes les heuristiques (y compris RSA-δ). Les temps de calcul pour les algorithmes GPS, RT et GRASP-PR ont été tirés de [Piñana *et al.*, 2004] où un processeur Athlon K-7 à 1.2 GHz a été utilisé (environ 27% plus lent que le nôtre). Les résultats pour RS-DJ ont été produits en exécutant le code original sur notre machine. Les temps de calcul affichés pour RS-DJ et notre algorithme ont été corrigés en appliquant le facteur d'équivalence correspondant.

Sur cette table, nous observons d'abord que RS-DJ est très lent. Notre algorithme RSA-δ emploie un ensemble de paramètres, obtenus en Section 4.4.2, qui accélèrent la convergence et permettent d'atteindre de meilleures solutions dans un temps de calcul inférieur. Nous remarquons aussi que l'exécution de l'algorithme classique GPS, bien que très rapide, donne des résultats inférieurs comparés à ceux des autres heuristiques. En particulier, il affiche un écart type moyen beaucoup plus grand que celui obtenu avec

	33 instances avec $30 \leq \|V\| \leq 199$						
					RSA-δ		
	GPS	RS-DJ	RT	GRASP-PR	C_m	C_p	$moy.$
Moyenne β	31.424	29.364	23.333	22.515	22.061	23.364	22.621
Écart type β	35.20%	56.20%	9.43%	2.27%	0.30%	7.29%	3.83%
Temps	0.003	2414.704	4.985	4.025	14.194	14.194	14.194
	80 instances avec $206 \leq \|V\| \leq 992$						
					RSA-δ		
	GPS	RS-DJ	RT	GRASP-PR	C_m	C_p	$moy.$
Moyenne β	156.375	164.588	100.775	99.425	94.800	97.438	95.699
Écart type β	46.52%	221.71%	11.44%	6.28%	0.13%	5.29%	1.97%
Temps	0.111	30945.131	263.969	339.969	253.050	253.050	253.050

Table 4.6 – Comparaison globale des performances par rapport à la taille des instances.

le pire résultat C_p trouvé par RSA-δ.

Pour le sous-ensemble de petites instances l'écart type moyen de GRASP-PR (2.27%) est plus petit que celui produit par la solution moyenne $moy.$ fournie par RSA-δ (3.83%), mais considérablement plus grand que celui produit par notre meilleure solution C_m (0.30%). D'autre part, pour les grandes instances on observe une amélioration claire de la largeur de bande moyenne obtenue avec notre algorithme (c'est-à-dire 94.800 contre 99.425 pour GRASP-PR). Cette comparaison met en évidence que RSA-δ est réellement compétitif, puisqu'il permet d'améliorer les anciens meilleurs résultats obtenus par les algorithmes de l'état de l'art RT et GRASP-PR, notamment, sur les grandes instances où l'amélioration est d'un peu plus de quatre unités. De plus, le temps d'exécution de RSA-δ reste très compétitif.

4.4.5 Discussion

Il est bien connu que le paramétrage et certaines caractéristiques clés ont un impact très important sur la performance globale du RS. C'est pourquoi nous avons effectué quelques expériences pour analyser ces influences de manière à mieux les comprendre.

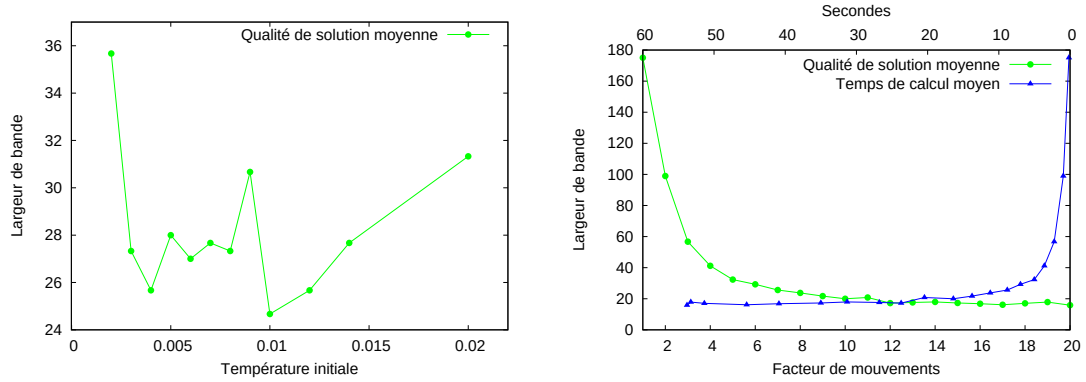
La série de tests formée par douze instances aléatoires décrite en Section 4.1 a été utilisée uniformément sur toutes les expériences présentées dans la suite de cette section. Des résultats semblables ont été obtenus avec toutes ces instances, donc nous avons décidé de montrer le produit de ces expériences avec quelques graphiques représentatifs. Ils ont été produits à partir des données générées au cours de vingt exécutions indépendantes.

Influence du schéma de refroidissement

Dans un algorithme de RS, le schéma de refroidissement détermine le degré de mouvements dégradant la solution courante qui est autorisé au cours de la recherche. Il est donc un composant critique pour les performances du RS. Au cours de cette section nous analysons deux paramètres importants pour le schéma de refroidissement.

Le premier paramètre est la température initiale T_i. La Figure 4.8(a) illustre l'influence de ce paramètre sur la qualité de solution produite par l'algorithme RSA-δ. Ce graphique

représente la qualité moyenne de solution atteinte (l'axe des ordonnées) à différentes températures initiales (l'axe des abscisses) sur vingt exécutions. Nous remarquons de cette courbe que les meilleurs résultats sont enregistrés lorsque la valeur de la température initiale est comprise dans l'intervalle 4.0E-03 $\leq T_i \leq$ 1.0E-02. Cette observation est valide sur les différentes instances d'essai testées.



(a) Influence de la température initiale

(b) Influence du facteur de mouvements

Figure 4.8 – Influence du schéma de refroidissement sur la performance de RSA-δ.

Le deuxième paramètre important pour le schéma de refroidissement est le nombre maximum des solutions voisines qui peuvent être visitées à chaque palier de température $maxEss$. Dans notre algorithme, $maxEss$ dépend directement du nombre d'arêtes du graphe E et du facteur de mouvements μ. Dans la Figure 4.8(b) l'influence du facteur de mouvements μ sur la qualité de solution atteinte par RSA-δ est montrée à l'aide de deux courbes. La première représente la qualité moyenne de solution obtenue sur vingt exécutions en utilisant différentes valeurs de μ. La seconde indique le temps de calcul nécessaire pour atteindre ces solutions. En observant ces deux courbes nous pouvons conclure que le meilleur compromis entre qualité de solution et temps d'exécution se trouve lorsque la valeur du facteur de mouvements μ est égale à 12.

Interaction entre la relation de voisinage et la fonction d'évaluation

Il est bien connu que la fonction d'évaluation et la relation de voisinage sont deux éléments cruciaux pour l'application efficace des métaheuristiques à la résolution des problèmes d'optimisation combinatoire [Stadler, 1992; Duvivier *et al.*, 1996; Hertz et Widmer, 2003; Hoos et Stützle, 2004]. Dans cette section, nous nous sommes attachés à étudier l'interaction entre ces deux éléments importants qui agissent ensemble sur le comportement de tout algorithme de recherche basé sur eux. Pour cela, nous considérons les fonctions d'évaluation β et δ, les voisinages $\mathcal{N}_1(\varphi)$ et $\mathcal{N}_2(\varphi)$ décrits dans les Sections 4.2.1 et 4.4.1, ainsi qu'un troisième exprimé dans l'équation suivante :

$$\mathcal{N}_3(\varphi) = \left\{\varphi' \in \mathscr{L} : swap(\varphi, u, v) = \varphi',\ u, v \in V,\ v \in A(u)\right\} \tag{4.4}$$

avec A(u) l'ensemble de sommets adjacents de u.

Les résultats des expériences effectuées pour cette étude sont montrés sur la Figure 4.9 qui représente la qualité moyenne de solution obtenue sur vingt exécutions de RSA-β et RSA-δ en utilisant les différents voisinage étudiés.

De ce graphique nous avons fait les observations suivantes. Tout d'abord, lorsque les algorithmes utilisent la fonction d'évaluation δ, ils obtiennent de meilleures résultats que les algorithmes utilisant β pour évaluer la qualité des solutions visitées ; ce qui montre que δ est une meilleure fonction d'évaluation que la fonction classique β. La deuxième observation concerne les voisinages étudiés, nous remarquons une claire domination du voisinage $\mathcal{N}_2(\varphi)$ fondée sur des mouvements de rotation sur les deux autres voisinages. Par rapport aux six combinaisons entre les fonctions d'évaluation et les voisinages, nous observons que la meilleure combinaison est $(\delta, \mathcal{N}_2(\varphi))$ car elle permet à l'algorithme RSA d'obtenir une largeur de bande de 11.2. Les autres combinaisons produisent les résultats suivants : $(\delta, \mathcal{N}_1(\varphi))$ 16.3, $(\delta, \mathcal{N}_3(\varphi))$ 17.5, $(\beta, \mathcal{N}_1(\varphi))$ 28.4, $(\beta, \mathcal{N}_2(\varphi))$ 22.5 et $(\beta, \mathcal{N}_1(\varphi))$ 31.1. Ceci justifie notre décision d'utiliser la combinaison $(\delta, \mathcal{N}_2(\varphi))$ dans l'implémentation de l'algorithme RSA.

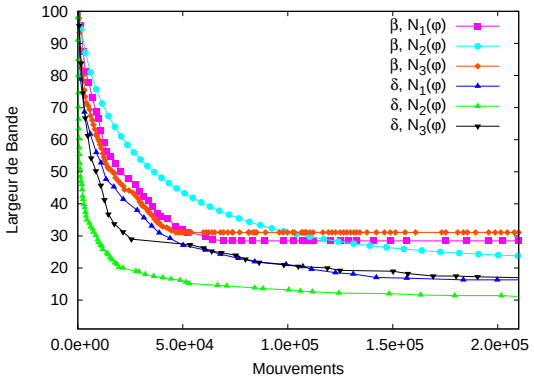


Figure 4.9 – Interaction entre le voisinage et la fonction d'évaluation sur la performance de l'algorithme RSA.

4.5 Synthèse du chapitre

Nous avons présenté dans ce chapitre des comparaisons expérimentales entre la fonction d'évaluation classique β du problème BMP, et la nouvelle fonction d'évaluation δ (introduite en Section 3.4). Ces comparaisons ont été effectuées à l'aide de deux algorithmes de recherche différents, la Descente Stricte et le Recuit Simulé. Les résultats obtenus permettent de confirmer que δ est plus pertinente comme fonction d'évaluation que β car notre nouvelle fonction améliore de manière importante les performances des deux algorithmes employés.

Les résultats de cette étude comparative nous ont permis également d'orienter nos efforts de recherche afin de concevoir, dans un second temps, un algorithme plus compétitif pour résoudre le problème BMP. Cet algorithme, appelé RSA-δ, a le mérite de per-

fectionner trois caractéristiques clés qui impactent de manière très importante sa capacité de recherche. Il tire avantage non seulement de la nouvelle fonction d'évaluation δ, mais aussi d'une représentation interne des solutions originale et d'une fonction de voisinage basée sur des mouvements de rotation. Les influences de ces caractéristiques clés ainsi que du paramétrage ont été analysées à travers de nombreuses expérimentations, afin de mieux les comprendre pour rendre notre algorithme plus performant.

Ensuite, RSA-δ a été comparé aux meilleures heuristiques connues en utilisant un large panel d'instances d'essai issues de la littérature. Cette comparaison a mis en évidence que RSA-δ est très compétitif, puisqu'il a amélioré les meilleurs résultats connus pour 57 des 113 instances d'essai.

Dans le chapitre suivant nous présentons un deuxième problème d'étiquetage de graphes, appelé le problème de l'*Arrangement Linéaire Minimum des Graphes* (MinLA), pour lequel nous proposons également une nouvelle fonction d'évaluation.

Chapitre 5

Problème de l'Arrangement Linéaire Minimum des Graphes

LE PROBLÈME de l'*Arrangement Linéaire Minimum des Graphes* (Minimum Linear Arrangement Problem, MinLA) consiste à trouver une manière d'étiqueter les sommets d'un graphe (un arrangement) de sorte que la somme de toutes les différences absolues entre les étiquettes des sommets adjacents soit minimum. Ce problème, énoncé pour la première fois dans les années soixante, apparaît dans plusieurs domaines aussi bien théoriques qu'appliqués de l'informatique et d'autres disciplines diverses comme les télécommunications, la gestion, l'ingénierie, la bioinformatique et l'électronique.

Dans ce chapitre, nous présentons formellement cet important problème d'étiquetage de graphes. Ensuite, une étude sur la fonction d'évaluation classique LA pour MinLA est effectuée en considérant l'ensemble de tous les graphes étiquetés simples à n sommets.

Les résultats de cette étude, liés à la fréquence d'apparition des différences absolues, nous ont conduit à créer une nouvelle fonction d'évaluation pour MinLA, appelée Φ. Cette nouvelle fonction évalue la qualité d'un étiquetage en considérant non seulement la somme de toutes leurs différences absolues, mais également certaines informations additionnelles induites par elles. Grâce à ces informations, Φ permet de surmonter les principaux inconvénients présentés par la fonction d'évaluation conventionnelle LA.

Une partie des travaux présentés dans ce chapitre a fait l'objet d'une publication dans la conférence internationale *MICAI'06* [Rodriguez-Tello *et al.*, 2006c].

Sommaire

5.1 Définition du problème MinLA

Le problème de l'*Arrangement Linéaire Minimum des Graphes* (Minimum Linear Arrangement Problem, MinLA), connu aussi sous le nom du problème de l'Arrangement Linéaire Optimum, a été énoncé pour la première fois dans les années soixante par Harper [1964]. Son objectif était de concevoir un code correcteur d'erreurs systématique, produisant des erreurs absolues moyennes minimales, pour certaines classes des graphes.

Plus tard, dans les années soixante-dix, le problème MinLA a été employé comme un modèle abstrait dans la phase de placement de la conception de circuits VLSI. Dans ce modèle, les sommets du graphe représentaient des modules et les arêtes des interconnexions entre ces modules. Dans ce cas, le coût de l'étiquetage mesurait la longueur totale du fil [Adolphson et Hu, 1973]. MinLA a aussi été utilisé plus récemment par [Mitchison et Durbin, 1986] pour modéliser, d'une manière très simplifiée, certaines activités nerveuses dans le cortex.

Le problème MinLA trouve aussi d'autres applications pratiques, notamment dans les domaines suivants : bioinformatique [Karp, 1993], ordonnancement d'atelier [Adolphson, 1977; Ravi *et al.*, 1991], dessin automatique des graphes [Shahrokhi *et al.*, 2001], arrangement automatique des diagrammes de flux [Lai et Williams, 1999], pour en mentionner seulement quelques uns.

Avant d'introduire formellement le problème MinLA, nous présentons le concept de largeur totale des arêtes d'un graphe qui est nécessaire à la définition du problème auquel nous nous attachons dans ce chapitre.

Définition 5.1 (Largeur totale des arêtes) *Soit $G = (V, E)$ un graphe non orienté d'ordre n, et $\varphi : V \rightarrow \{1, 2, ..., n\}$ un* arrangement linéaire, *ou étiquetage. La* largeur totale des arêtes LA *de G pour φ est définie selon l'équation suivante :*

$$\text{LA}(G, \varphi) = \sum_{\{u,v\} \in E} |\varphi(u) - \varphi(v)| \tag{5.1}$$

Définition 5.2 (Problème MinLA) *Soit $G = (V, E)$ un graphe non orienté d'ordre n. Le problème MinLA consiste à trouver un étiquetage φ^* pour lequel la* largeur totale des arêtes $\text{LA}(G, \varphi^*)$, *ou coût, est minimum :*

$$\text{LA}(G, \varphi^*) = \min\{\text{LA}(G, \varphi) : \varphi \in \mathscr{L}\} \, , \tag{5.2}$$

avec $\mathscr{L}$ l'ensemble de tous les étiquetages possibles.

Il est simple d'observer que l'ensemble $\mathscr{L}$ se compose de $n!$ étiquetages possibles (arrangements linéaires) pour un graphe d'ordre n.[1] C'est pourquoi le problème MinLA est considéré comme un problème fortement combinatoire. Comme dans la grande majorité des problèmes d'étiquetage des graphes, trouver l'arrangement minimum pour un

[1] Notez que chacun des $n!$ étiquetages peut être inversé pour obtenir le même coût.

graphe en général est un problème NP-difficile [Garey et Johnson, 1979]. Il a été démontré par Even et Shiloah [1975] que le problème de décision associé à MinLA reste NP-complet, même pour des graphes bipartis. Seulement dans certains cas très spécifiques, il est possible de trouver l'arrangement linéaire optimum en temps polynomial.

5.2 Principales approches de résolution pour le problème MinLA

Comme nous venons de le mentionner, il existe des *algorithmes exacts* de résolution en temps polynomial pour certaines instances spéciales du problème MinLA. C'est notamment le cas pour les arbres, les arbres enracinés, les hypercubes, les grilles, les graphes planaires extérieurs, les graphes de De Bruijn, etc. Pour une information plus détaillée sur les classes de graphes qui peuvent être résolus pour MinLA en temps polynomial, le lecteur peut consulter l'article de synthèse suivant [Diaz *et al.*, 2002].

Cependant, étant donnée la complexité du problème MinLA, il y a une nécessité d'utiliser des *algorithmes approchés* pour résoudre de grandes instances de ce problème dans des temps de calcul raisonnables. Parmi les algorithmes de ce type rapportés dans la littérature on trouve : a) des heuristiques spécialement développées pour le problème MinLA, comme l'heuristique de Minimisation Frontale Améliorée (MFA) [McAllister, 1999], l'heuristique de l'Arbre Binaire de Décomposition (ABD) [Bar-Yehuda *et al.*, 2001], l'algorithme Multi-Couche (MC) [Koren et Harel, 2002], et le schéma Algébrique Multi-Grille (AMG) [Safro *et al.*, 2006] ; b) des métaheuristiques comme le Recuit Simulé (RS) [Petit, 2003b], ainsi que les Algorithmes Génétiques (AG) [Poranen, 2005] ; et c) des combinaisons de ces méthodes.

Par la suite, nous présentons une brève révision des cinq procédures les plus représentatives pour la résolution du problème MinLA.

5.2.1 Ordonnancement Spectral et Recuit Simulé

En raison de l'importance pratique et théorique du problème de MinLA, beaucoup de recherches ont été effectuées en développant des heuristiques efficaces pour le résoudre. C'est notamment le cas de l'heuristique OS+RS proposée par Petit [2003a]. Cet algorithme consiste à obtenir une première solution en utilisant la méthode d'*Ordonnancement Spectrale* (OS) [Juvan et Mohar, 1992]. Ensuite, l'étiquetage résultant est amélioré localement par l'algorithme du *Recuit Simule* (RS) rapporté dans [Petit, 2003b]. Cet algorithme de RS, qui met en application un schéma de refroidissement géométrique, est basé sur une distribution spéciale de voisinage qui cherche à favoriser des mouvements avec une probabilité élevée d'être acceptés. L'auteur a fait des comparaisons expérimentales de la méthode OS, de l'algorithme de RS, et de la combinaison de ces dernières sur un ensemble de vingt-et-une instances d'essai collectées par lui-même. Il conclut que l'heuristique OS+RS améliore toujours les solutions produites par la méthode OS et seulement pour deux graphes ($c5y$ et $gd96a$) elle ne peut pas améliorer la solution fournie par le RS. De plus, les temps d'exécution de OS+RS sont habituellement inférieurs à ceux de l'algorithme de RS.

5.2.2 Heuristique de l'Arbre Binaire de Décomposition

En plus du travail de Petit, Bar-Yehuda *et al.* [2001] ont présenté une approche de solution pour le problème MinLA inspirée par le principe de « diviser pour régner ». Cet algorithme, qui s'exécute en temps polynomial (de complexité $O(|V|^{2.2})$), calcule l'étiquetage induit par un *Arbre Binaire de Décomposition* (ABD).

Pour évaluer leur approche, les auteurs ont utilisé la même série d'essais de vingt-et-une instances proposées par Petit [2003b]. Bar-Yehuda *et al.* ont appliqué leur algorithme de manière itérative, en commençant chaque itération avec le résultat de la précédente. Après quelques dizaines d'itérations, l'algorithme ABD retourne habituellement des résultats entre cinq et dix pour cent moins bons que ceux du RS de Petit [2003b], mais en utilisant seulement une petite fraction du temps de calcul dépensé par ce dernier.

Dans un second temps, Bar-Yehuda *et al.* ont employé les étiquetages produits avec leur heuristique ABD, comme des solutions initiales pour le RS de Petit [2003b], obtenant des résultats légèrement meilleurs que ceux du RS seul.

5.2.3 Heuristique de Minimisation Frontale Améliorée

En 1999, une heuristique s'exécutant en temps linéaire (de complexité $O(|E|)$) et basée sur la *Minimisation Frontale Améliorée* (MFA) a été développée par McAllister [1999]. Dans ce travail, l'auteur a comparé son heuristique MFA avec trois méthodes développées originalement pour le problème BMP et de réduction du profil des matrices (RP) : l'algorithme Inversé de Cuthill-McKee (ICM) [Cuthill et McKee, 1969], la méthode GPS de Gibbs *et al.* [1976], et la méthode GK Gibbs [1976]. La comparaison inclut également un quatrième algorithme conçu pour MinLA qui se base sur le calcul des valeurs propres [Liu et Vannelli, 1995].

Pour ses expérimentations, McAllister a rassemblé trente-quatre instances de test, divisées en deux séries d'essais. La première composée de vingt graphes dérivés des diagrammes de flux, et la seconde formée de quatorze matrices structurales[2] pris de la collection Rutherford-Boeing de matrices creuses. Les résultats expérimentaux de l'auteur montrent que l'algorithme MFA fournit le meilleur étiquetage pour dix-sept instances de la première série de tests. Pour la seconde série de tests, bien que MFA produise de meilleurs résultats que les algorithmes ICM, GPS et GK (pour quatorze instances), il est moins efficace que l'approche basée sur le calcul des valeurs propres.

5.2.4 Algorithme Multi-Couche

Koren et Harel ont présenté en 2002 un autre algorithme s'exécutant en temps linéaire (de complexité $O(|E|)$) pour le problème MinLA. Il est basé sur la combinaison des méthodes spectrales et du paradigme *Multi-Couche* (MC) [Koren et Harel, 2002].

Les techniques de MC transforment un problème hautement dimensionnel, de manière itérative, en des sous-problèmes de dimensions de plus en plus petites, en employant un processus appelé *coarsening* ou déraffinage. À l'échelle la plus grossière, le

[2] Utilisées pour le calcul de la résistance de cadres en acier.

problème est résolu exactement. La solution trouvée est alors raffinée et progressivement projetée de retour dans des dimensions de plus en plus élevées. Ce processus de raffinage itératif s'arrête lorsque le problème original est reproduit et résolu. Cet ensemble de pas est appelé un *V-cycle*.

Pour leurs expérimentations, les auteurs ont également utilisé les instances de test proposées dans [Petit, 2003b]. Ils ont exécuté leur algorithme MC en utilisant tout d'abord un seul V-cycle, puis avec dix. Les comparaisons expérimentales menées par Koren et Harel ont mis en évidence que leur algorithme MC, paramétré avec dix V-cycles, produit des résultats comparables à ceux du RS de Petit [2003b], mais en dépensant un temps de calcul légèrement inférieur.

5.2.5 Schéma Algébrique Multi-Grille

En 2004, un *Schéma Algébrique Multi-Grille* (AMG) pour la résolution du problème MinLA a été proposé par Safro *et al.* [2006]. Cette approche peut être considérée comme une amélioration à l'algorithme MC de Koren et Harel.

La principale différence entre ces deux méthodes se trouve dans la technique de déraffinage (coarsening). MC emploie l'agrégation *stricte*, alors que AMG utilise l'agrégation *pondérée*. Dans la procédure d'agrégation stricte, les sommets du graphe sont bloqués dans de petits sous-ensembles disjoints nommés *agrégats*. En revanche, dans l'agrégation pondérée, chaque sommet peut être divisé en plusieurs fractions, et ces fractions peuvent appartenir à différents agrégats.

Safro *et al.* ont montré expérimentalement que leur heuristique AMG pour MinLA, est capable d'obtenir des résultats de très bonne qualité en utilisant un temps de calcul linéaire. AMG peut être considéré actuellement comme un des meilleurs algorithmes pour résoudre ce problème.

5.3 Étude de caractéristiques de la fonction d'évaluation LA

Il est important de signaler que la totalité des algorithmes de résolution du problème MinLA que nous venons de présenter évaluent la qualité d'une solution en considérant la variation de la fonction objectif $\text{LA}(G, \varphi)$ associée au problème (appelons-la seulement LA pour simplifier). Cependant, nous avons observé que cette fonction d'évaluation classique n'est pas forcément le meilleur choix car LA ne permet pas de distinguer des étiquetages avec la même largeur totale des arêtes. Ce point sera abordé en détail plus tard.

Dans la suite de cette section, nous présentons une analyse détaillée de certaines caractéristiques de la fonction d'évaluation classique LA. L'objectif essentiel de cette étude est de mettre en évidence les inconvénients présentés par la fonction LA et en même temps d'obtenir des informations qui nous guident vers la conception d'une nouvelle fonction d'évaluation plus efficace.

Soit $G = (V, E)$ un graphe non orienté d'ordre n et φ un étiquetage (ou permutation). Nous constatons que ce graphe G peut avoir potentiellement $n(n-1)/2$ arêtes non

réflexives. Étant donné que chacune de ces arêtes peut être présente ou absente dans le graphe, le nombre total de graphes étiquetés simples (sans boucle) est exprimé par l'Équation 3.11, où $\mathscr{G}_s$ représente l'ensemble contenant ces graphes.

$$|\mathscr{G}_s| = 2^{\frac{n(n-1)}{2}} = 2^{\binom{n}{2}} \tag{5.3}$$

Dans l'analyse suivante, nous ne considérons pas les graphes avec boucle car aucune contribution n'est faite par ces arêtes réflexives au coût d'un étiquetage. Cependant, nous tenons compte de toutes les valeurs possibles de LA, même $\text{LA} = 0$.

Supposons un graphe étiqueté sans boucle d'ordre n. Dans ce graphe le nombre maximum de différences absolues est distribué comme suit : 1 de valeur $(n-1)$, 2 de valeur $(n-2)$, et en général k différences absolues de valeur $(n-k)$ pour tout k dans l'intervalle $[1, n-1]$. Nous pouvons donc exprimer la fonction LA en terme des fréquences d'apparition d_k du graphe comme suit.

$$\text{LA}(G, \varphi) = \sum_{k=1}^{n-1} k d_k \tag{5.4}$$

Il est donc clair que la fonction d'évaluation LA peut valoir entre 0 (un graphe vide ou composé seulement d'arêtes réflexives) et la valeur exprimée par l'Équation 5.5.

$$\sum_{k=1}^{n-1} k(n-k) = \frac{n(n^2-1)}{6} \tag{5.5}$$

Soit $\mathcal{R}_{\text{LA}} \subseteq \mathscr{G}_s \times \mathscr{G}_s$ une *relation d'équivalence* [Rotman, 2003] sur l'ensemble de tous les graphes étiquetés simples et $x, y \in \mathscr{G}_s$, alors x est en relation avec y ($x \ \mathcal{R}_{\text{LA}} \ y$) si et seulement si x et y ont un ensemble de fréquences d'apparition $D = \{d_1, d_2, ..., d_{n-1}\}$ produisant la même largeur totale des arêtes LA.[3] Une *classe d'équivalence* $[x] = \{y \in \mathscr{G}_s : x \ \mathcal{R}_{\text{LA}} \ y\} \subseteq \mathscr{G}_s$ peut être alors définie pour chaque valeur possible de LA. Par conséquent, le nombre total de classes d'équivalence $\mathcal{E}_{\text{LA}}$ sous LA, parmi lesquelles l'ensemble $\mathscr{G}_s$ de tous les graphes étiquetés simples d'ordre n est divisé, est :

$$\mathcal{E}_{\text{LA}} = 1 + \frac{n(n^2-1)}{6} \tag{5.6}$$

Nous observons qu'il peut exister différents étiquetages résultant dans le même ensemble de fréquences d'apparition $D = \{d_1, d_2, ..., d_{n-1}\}$ dont le nombre exact z peut être calculé avec la Formule suivante :

$$z(D) = \prod_{k=1}^{n-1} \binom{k}{d_{n-k}} \tag{5.7}$$

[3] Observez qu'il peut s'agir d'un même graphe ou de deux graphes différents.

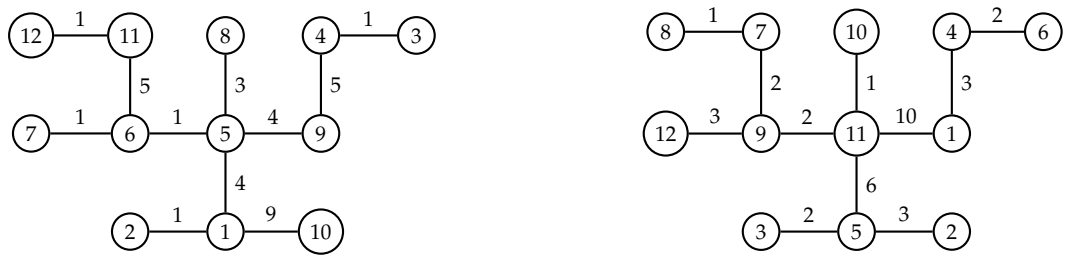
Néanmoins, on remarque qu'il peut exister p différents ensembles de fréquences d'apparition produisant la même valeur LA, c'est-à-dire $P = \{D_1, D_2, ..., D_p\}$. De cette manière, la cardinalité $\omega_{\mathrm{LA}}(i, P)$ pour la classe d'équivalence sous $\mathrm{LA} = i$ peut être calculée à l'aide de l'Équation 5.8.

$$\omega_{\mathrm{LA}}(i, P) = \sum_{j=1}^{p} z(D_j) \tag{5.8}$$

D'après ce que nous venons d'observer, la fonction d'évaluation LA ne fait pas de distinction entre les différences absolues de grande valeur et celles de petite valeur. Par exemple, pour la fonction LA il est équivalent d'avoir une différence absolue de valeur 25 plutôt que 25 différences de valeur 1 (Équation 5.1). En conséquence, il n'y a aucune possibilité de faire la distinction entre les $\omega_{\mathrm{LA}}(i, P)$ graphes étiquetés simples qui appartiennent à la même classe d'équivalence $\mathrm{LA} = i$.

Par exemple, la fonction LA associe la même valeur de coût $\mathrm{LA} = 35$ aux deux étiquetages représentés sur la Figure 5.1, pour un graphe simple d'ordre 12. Cependant, une analyse plus fine de chaque fréquence d'apparition d_k nous permet de confirmer que la permutation φ' sur la Figure 5.1(b) est peut-être plus intéressante que celle de la Figure 5.1(a) car il est plus simple de minimiser la valeur de LA en réduisant une différence absolue de valeur 10 ($d_{10} = 1$) de φ' que les cinq différences absolues de valeur 1 ($d_1 = 5$) de φ. En effet, si l'on réduit une différence absolue de valeur 10, LA pourrait diminuer de 10 unités ; alors que si nous arrivons à éliminer cinq différence absolues de valeur 1 (ce qui est plus difficile), LA pourrait seulement baisser sa valeur de 5 unités.

Partant de cette observation, nous proposons dans la section suivante une nouvelle fonction d'évaluation pour le problème MinLA qui est plus informative. En effet, elle juge de manière individualisée chaque fréquence d'apparition produite par un étiquetage afin d'évaluer sa qualité.



(a) Permutation φ **(b)** Permutation φ'

Figure 5.1 – Exemple de deux étiquetages avec la même valeur de $\mathrm{LA} = 35$.

5.4 Nouvelle fonction d'évaluation Φ

L'étude de la fonction LA, que nous venons de présenter dans la section précédente, a mis en évidence les limitations de cette fonction d'évaluation. De plus, cette étude nous

a permis d'observer l'importance de considérer de manière différente chaque fréquence d'apparition d_k au moment d'évaluer le coût d'un étiquetage.

Nous poursuivons cette section en introduisant une nouvelle fonction d'évaluation, appelée Φ, qui est fondée sur ce principe. Cette fonction évalue un étiquetage en considérant non seulement sa largeur totale des arêtes (LA), mais également des informations additionnelles induites par les fréquences d'apparition d_k. De plus, elle maintient le fait que $\lfloor \Phi \rfloor$ fournit la même valeur entière produite par les Équations 5.1 et 5.4.

Dans le calcul de la nouvelle fonction d'évaluation Φ, chaque fréquence d'apparition d_k doit apporter une contribution différente au coût d'un étiquetage. Cette contribution est calculée en utilisant l'Équation 5.9.

$$k + \frac{1}{\prod_{j=1}^{k}(n+j)} = k + \frac{n!}{(n+k)!} \tag{5.9}$$

En appliquant ces contributions dans la Formule 5.4, l'expression suivante est atteinte :

$$\sum_{k=1}^{n-1} \left(k + \frac{n!}{(n+k)!} \right) d_k \tag{5.10}$$

En simplifiant cette formule, nous obtenons la nouvelle fonction d'évaluation Φ dont le premier terme est la fonction LA (Équation 5.4), et le deuxième terme (une valeur fractionnelle) est utilisé pour distinguer les étiquetages ayant la même largeur totale des arêtes LA.

$$\Phi(G, \varphi) = \sum_{k=1}^{n-1} k d_k + \sum_{k=1}^{n-1} \frac{n! d_k}{(n+k)!} \tag{5.11}$$

À titre d'exemple, considérons l'étiquetage (permutation) φ pour le graphe d'ordre $n = 12$ présenté sur la Figure 5.1(a). Pour cette permutation en particulier, LA $= 35$ et les fréquences d'apparition d_k des différences absolues sont : $d_1 = 5$, $d_3 = 1$, $d_4 = 2$, $d_5 = 2$ et $d_9 = 1$. En faisant la substitution de ces valeurs dans la Formule 5.11 nous obtenons :

$$\Phi(G, \varphi) = 35 + \frac{5}{\text{6.2E+09}} + \frac{1}{\text{1.3E+12}} + \frac{2}{\text{2.0E+13}} + \frac{2}{\text{3.5E+14}} + \frac{1}{\text{5.1E+19}} = 35.385 \tag{5.12}$$

En revanche, si Φ est calculée pour la permutation φ' représentée sur la Figure 5.1(b), nous observons que leurs fréquences d'apparition d_k sont : $d_1 = 2$, $d_2 = 4$, $d_3 = 3$, $d_6 = 1$ et $d_{10} = 1$, ce qui fournit une valeur plus petite :

$$\Phi(G, \varphi') = 35 + \frac{2}{\text{6.2E+09}} + \frac{4}{\text{8.7E+10}} + \frac{3}{\text{1.3E+12}} + \frac{1}{\text{6.4E+15}} + \frac{1}{\text{1.1E+21}} = 35.177 \tag{5.13}$$

L'idée principale de la nouvelle fonction d'évaluation Φ est de pénaliser les différences absolues comportant des petites valeurs, et de favoriser celles ayant des valeurs proches de β. Ceci peut clairement être observé dans l'exemple ci-dessus (Équation 5.12), où la fonction Φ pénalise les différences absolues de valeur 1 ($d1 = 5$) plus que celles de

valeur 3 ($d3 = 1$) en les multipliant respectivement par un facteur de $\frac{1}{6.2\text{E+}09}$ et $\frac{1}{1.3\text{E+}12}$. De cette manière, Φ attribuera toujours un coût inférieur à un étiquetage comportant plus de différences absolues de grande valeur comme celui de la Figure 5.1(b), car intuitivement cet étiquetage a une probabilité plus forte d'être amélioré ultérieurement.

En effet, rappelons que dans un graphe étiqueté sans boucle d'ordre n, il y a en général un nombre maximum $(n-k)$ de différences absolues de valeur k ($k \in [1, n-1]$). En conséquence, plus la valeur d'une différence absolue k est grande, plus la fréquence d'apparition d_k est faible. Par exemple, dans un graphe étiqueté sans boucle d'ordre $n = 50$ il existe au plus 49 différences absolues de valeur 1, alors que il y a au maximum une différence absolue de valeur 49. Il est donc plus simple de réduire la largeur totale des arêtes d'un étiquetage en modifiant les étiquettes dans les extrémités d'une seule arête, que de changer les étiquettes de tous les sommets liés par les 49 arêtes qui produisent une différence absolue de 1. Partant de ce constat, nous avons conçu la fonction Φ de sorte qu'elle puisse favoriser les étiquetages qui comportent des différences absolues de grande valeur.

5.5 Étude de caractéristiques de la nouvelle fonction d'évaluation Φ

Nous présentons dans cette section une étude de certaines caractéristiques de la nouvelle fonction d'évaluation Φ. Il s'agit d'une étude similaire à celle présentée dans la Section 5.3 pour la fonction d'évaluation classique LA.

Soit $\mathcal{R}_\Phi \subseteq \mathscr{G}_s \times \mathscr{G}_s$ une *relation d'équivalence* sur l'ensemble de tous les graphes étiquetés simples et $x, y \in \mathscr{G}_s$, alors x est en relation avec y ($x \ \mathcal{R}_\Phi \ y$) si et seulement si x et y ont le même ensemble de fréquences d'apparition $D = \{d_1, d_2, ..., d_{n-1}\}$ (la même valeur Φ). Une *classe d'équivalence* $[x] = \{y \in \mathscr{G}_s : x \ \mathcal{R}_\Phi \ y\} \subseteq \mathscr{G}_s$ peut être alors définie pour chaque valeur possible de Φ.

Rappelons que dans un graphe étiqueté d'ordre n, il y a en général k différences absolues de valeur $(n-k)$ pour $k \in [1, n-1]$. En conséquence, les fréquences d'apparition d_k prennent des valeurs comprises entre 0 et $(n-k)$. Ainsi, le nombre total $\mathcal{E}_\Phi$ de classes d'équivalence sous Φ, parmi lesquelles l'ensemble $\mathscr{G}_s$ de tous les graphes étiquetés simples à n sommets est divisé, est :

$$\mathcal{E}_\Phi = \prod_{k=0}^{n-1} (n-k) = n! \tag{5.14}$$

Étant donné que deux étiquetages appartiennent à la même classe d'équivalence sous Φ s'ils ont le même ensemble de fréquences d'apparition $D = \{d_1, d_2, ..., d_{n-1}\}$, nous pouvons calculer la cardinalité $\omega_\Phi(i, D)$ pour la classe d'équivalence $\Phi = i$ en utilisant l'équation suivante :

$$\omega_\Phi(i, D) = \prod_{k=1}^{n-1} \binom{k}{d_{n-k}} \tag{5.15}$$

L'étude des classes d'équivalence produites par Φ, que nous venons de présenter, permet de tirer certaines conclusions importantes. En particulier, nous avons observé le fait que Φ divise chaque classe d'équivalence produite par la fonction LA en sous-ensembles (classes d'équivalence) de plus petite taille qui regroupent les étiquetages partageant le même ensemble de fréquences d'apparition d_k. C'est grâce à cette manière rationnelle d'incrémenter le nombre de classes d'équivalence que Φ permet de distinguer des permutations ayant la même valeur de LA. De plus, Φ est cohérente par rapport à l'objectif du problème MinLA qui consiste à minimiser LA : pour deux étiquetages φ et φ' si $\Phi(\varphi) < \Phi(\varphi')$, alors $\mathrm{LA}(\varphi) \leq \mathrm{LA}(\varphi')$ (voir Équations 5.4 et 5.11).

5.6 Exemple d'application de LA et Φ

Dans les sections précédentes, nous avons présenté et analysé les deux fonctions d'évaluation LA et Φ dans le cas général de tous les graphes étiquetés à n sommets. Nous allons montrer à présent sur un exemple concret ($n = 4$) les différentes classes d'équivalences produites par chacune de ces deux fonctions. Cela nous permettra d'observer d'une part de quelle manière chaque classe d'équivalence fournie par LA est divisée en sous-classes d'équivalence par la fonction Φ, et d'autre part comment la fonction Φ affecte des valeurs différentes aux étiquetages ayant la même largeur totale des arêtes LA.

	D							
j	d_3	d_2	d_1	$z(D)$	$\omega_{\mathrm{LA}}(i,P)$	LA	$\omega_{\Phi}(i,D)$	Φ
1	0	0	0	1	1	0	1	0.0000
2	0	0	1	3	3	1	3	1.2000
3	0	1	0	2	5	2	2	2.0333
4	0	0	2	3			3	2.4000
5	1	0	0	1			1	3.0048
6	0	1	1	6	8	3	6	3.2333
7	0	0	3	1			1	3.6000
8	0	2	0	1			1	4.0667
9	1	0	1	3	10	4	3	4.2048
10	0	1	2	6			6	4.4333
11	1	1	0	2			2	5.0381
12	0	2	1	3	10	5	3	5.2667
13	1	0	2	3			3	5.4048
14	0	1	3	2			2	5.6333
15	1	1	1	6			6	6.2381
16	0	2	2	3	10	6	3	6.4667
17	1	0	3	1			1	6.6048
18	1	2	0	1			1	7.0714
19	1	1	2	6	8	7	6	7.4381
20	0	2	3	1			1	7.6667
21	1	2	1	3	5	8	3	8.2714
22	1	1	3	2			2	8.6381
23	1	2	2	3	3	9	3	9.4714
24	1	2	3	1	1	10	1	10.6714

Table 5.1 – Comparaison entre les classes d'équivalence produites par les fonctions d'évaluation LA et Φ pour les graphes étiquetés d'ordre $n = 4$

Rappelons que le nombre total de graphes étiquetés simples à quatre sommets est

égal à 64 (Équation 5.3). En utilisant la fonction d'évaluation LA ces 64 étiquetages sont regroupés en $\mathcal{E}_{\text{LA}} = 11$ classes d'équivalence dont les cardinalités sont montrées dans la colonne $\omega_{\text{LA}}(i, P)$ de la Table 5.1. En revanche, la nouvelle fonction Φ partitionne chacune de ces 11 classes d'équivalence produites par LA en sous-classes d'équivalence de plus petite taille pour en totaliser $\mathcal{E}_{\Phi} = 24$.

Par exemple, la classe d'équivalence sous $\text{LA} = 5$ contient dix étiquetages distincts, alors que sous Φ ces étiquetages sont divisés en quatre classes d'équivalence différentes dont les cardinalités sont respectivement 2, 3, 3 et 2 (Table 5.1, colonnes Φ et $\omega_{\Phi}(i, D)$). De plus, Φ affecte une valeur d'évaluation différente pour chacune de ces 12 classes d'équivalence. Ainsi, même si ces dix étiquetages ont la même valeur $\text{LA} = 5$, Φ juge que certains d'entre eux sont plus prometteurs que d'autres car ils possèdent un nombre supérieur de différences absolues de grande valeur (voir l'exemple en fin de Section 5.4).

5.7 Comparaison de complexité entre LA et Φ

Pour calculer le coût d'une permutation φ en utilisant la fonction d'évaluation classique LA, la totalité des arêtes du graphe $G = (V, E)$ doit être analysée (voir 5.1) ; en conséquence, $O(|E|)$ opérations sont requises.

Afin de calculer plus efficacement la fonction d'évaluation Φ, nous pouvons pré-évaluer chaque terme $k + (n!/(n + k)!)$ de l'Équation 5.11 et le stocker dans une matrice $M = \{m_{ij}\}_{n \times n}$. Ceci nécessite d'exécuter $2|V|$ opérations, c'est-à-dire $O(|V|)$. Ainsi, à chaque fois que la valeur de Φ doit être déterminée à nouveau, la somme $\sum_{(u,v) \in E} m_{uv}$ est recalculée, ce qui implique la même complexité algorithmique que celle requise pour calculer LA. De plus, la fonction Φ permet l'évaluation incrémentale des solutions voisines[4]. En effet, supposons que les étiquettes de deux sommets distincts (u, v) soient permutées, alors nous devrions seulement recalculer les $|A(u)| + |A(v)|$ différences absolues modifiées, où $|A(u)|$ et $|A(v)|$ représentent respectivement le nombre de sommets adjacents à u et à v. Comme nous pouvons l'observer, ceci est plus rapide que les $O(|E|)$ opérations requises originellement.

5.8 Synthèse du chapitre

Au début de ce chapitre, nous avons présenté de manière formelle le problème de l'Arrangement Linéaire Minimum des Graphes, ainsi que les principales approches de résolution rapportées dans la littérature pour cet important problème d'étiquetage de graphes. Ensuite, nous avons effectué une étude à un niveau très général de certaines caractéristiques de la fonction d'évaluation classique LA pour MinLA en considérant l'ensemble de graphes étiquetés simples à n sommets.

Cette étude a mis en évidence les principaux inconvénients présentés par la fonction LA et nous a permis d'observer l'importance de considérer individuellement chaque fréquence d'apparition d_k afin de faire une distinction plus fine des étiquetages évalués.

[4]Notons que LA peut être aussi calculée de manière incrémentale.

Cette observation nous a amené à créer une nouvelle fonction d'évaluation, appelée Φ, fondée sur ce principe. Elle évalue la qualité d'un étiquetage en tenant compte non seulement de la largeur totale des arêtes, mais également d'autres informations induites par les fréquences d'apparition d_k. Ainsi, Φ est capable de distinguer des étiquetages ayant la même valeur LA.

Dans le chapitre suivant, nous présentons une comparaison expérimentale entre les fonctions d'évaluation LA et Φ afin d'établir la pertinence de la fonction d'évaluation proposée dans ce chapitre. Nous montrons également deux algorithmes approchés très performants qui tirent pleinement profit de la nouvelle fonction d'évaluation Φ.

Chapitre 6

Comparaisons expérimentales entre les fonctions d'évaluation LA et Φ

NOUS VENONS d'introduire dans le chapitre précédent une nouvelle fonction d'évaluation pour le problème MinLA, appelée Φ. Dans ce chapitre nous présentons une série de comparaisons expérimentales entre cette fonction et la fonction d'évaluation classique LA. Ces comparaisons visent, d'une part, à analyser la distribution des voisins améliorants produits par Φ quand elle est couplée à une relation de voisinage donnée et, d'autre part, à évaluer l'utilité pratique de cette nouvelle fonction. Pour ce faire, deux algorithmes de recherche ont été implémentés : la *Descente Stricte* et un *Algorithme Mémétique*. Les résultats expérimentaux obtenus montrent que Φ permet d'améliorer considérablement les performances de ces deux algorithmes.

La connaissance acquise au cours de cette étude expérimentale nous a permis de concevoir dans un second temps, un *Algorithme Mémétique Amélioré* pour le problème MinLA [Rodriguez-Tello *et al.*, 2006b]. Cet algorithme, appelé AMA-Φ, tire avantage de la nouvelle fonction d'évaluation Φ, mais aussi d'une procédure efficace d'initialisation de la population, d'un opérateur de croisement spécialisé et d'un opérateur de RL avancé [Rodriguez-Tello *et al.*, 2005]. Les comparaisons entre AMA-Φ et les heuristiques de l'état de l'art, effectuées sur 21 instances de test issues de la littérature, mettent en évidence que AMA-Φ est très compétitif en terme de qualité de solution atteinte [Rodriguez-Tello *et al.*, 2006c]. En effet, il permet d'améliorer les meilleurs résultats connus pour 7 instances et parvient à les égaler pour 8 autres. Cependant, il présente certaines limites comme la taille des instances qui peuvent être traitées ou le temps de calcul consommé qui peut s'avérer assez long.

Au vu des limitations que comporte AMA-Φ, nous nous sommes tournés vers un algorithme de Recuit Simulé. Après avoir étudié différents types de schémas de refroidissement et de fonctions de voisinage nous avons mis au point un algorithme de *Recuit Simulé en Deux Phases* hautement efficace. Cet algorithme, que nous appelons RSDP-Φ, base ses performances sur la fonction d'évaluation Φ, un schéma de refroidissement statistique et une fonction de voisinage combinée. Afin d'évaluer l'efficacité pratique de cet algorithme, nous avons effectué des expérimentations approfondies sur un ensemble de

30 instances d'essai standard de la littérature. Dans ces expériences l'algorithme RSDP-Φ a été soigneusement comparé à cinq autres heuristiques de l'état de l'art. Les résultats obtenus témoignent d'un important gain en efficacité de RSDP-Φ par rapport aux heuristiques existantes, notamment sur les instances de très grande taille. En effet, RSDP-Φ permet d'améliorer 17 des 30 meilleures solutions connues, en terme de qualité de solution [Rodriguez-Tello *et al.*, 2007].

Sommaire

6.1 Instances de test et critères de comparaison

Toutes les expérimentations présentées dans ce chapitre ont été effectuées sur un panel d'instances d'essai comprenant notamment des *benchmarks* issus de la littérature. Ces instances se regroupent par rapport à leur taille en deux séries de tests.

La première série consiste en 21 instances d'essai[1] proposées originalement par [Petit, 2003b] et utilisées ensuite par de nombreux chercheurs [Bar-Yehuda *et al.*, 2001; Poranen, 2005; Rodriguez-Tello *et al.*, 2006b]. Cette série inclut des graphes appartenant à six familles différentes : aléatoires uniformes, aléatoires géométriques, avec solution optimale connue, discrétisations provenant de méthodes des éléments finis, conception de circuits VLSI et compétitions de dessin automatique des graphes. La taille de ces instances varie entre 62 et 9800 sommets.

La deuxième série se compose de 9 graphes de très grande taille formés à partir de discrétisations provenant de méthodes des éléments finis. Ces instances sont issues des collections de George Karypis[2] et François Pellegrini.[3] Cette série de tests a été utilisée pour la première fois par Koren et Harel [2002], puis par Safro *et al.* [2006]. Les instances qui la composent comportent entre 78136 et 1017253 sommets.

Pour évaluer la performance des fonctions d'évaluation nous montrons des résultats comparatifs sur les différentes instances de test. Le critère de comparaison qui est retenu pour ces comparaisons est celui utilisé régulièrement dans la littérature : la *largeur totale des arêtes minimum* atteinte. Le temps de calcul est également donné à titre indicatif.

6.2 Comparaison avec un algorithme de descente stricte

Au cours de cette section, une première comparaison expérimentale entre la nouvelle fonction d'évaluation Φ et la fonction d'évaluation classique LA est effectuée à l'aide d'un algorithme de Descente Stricte (DS) similaire à celui décrit dans la Section 4.2.1. Cet algorithme de DS utilise aussi une stratégie de mouvement de la meilleure amélioration.

6.2.1 Descente stricte

La DS a été choisie pour effectuer cette comparaison expérimentale pour deux raisons : 1) le fait que la DS n'utilise pas de paramètres permet une comparaison plus équitable et sans biais entre les fonctions étudiées, 2) la stratégie de mouvement utilisée dans la DS permet d'étudier certaines caractéristiques de l'espace de recherche auxquelles nous nous intéressons telles que la distribution du nombre de voisins améliorants. Les détails d'implémentation de cet algorithme de DS sont présentés dans cette section.

[1] http://www.lsi.upc.es/~jpetit/MinLA/Experiments
[2] ftp://ftp.cs.umn.edu/users/kumar/Graphs
[3] http://www.labri.u-bordeaux.fr/Equipe/PARADIS/Member/pelegrin/graph

Représentation et fonction d'évaluation

Soit un graphe $G = (V, E)$ d'ordre n, l'espace de recherche $\mathscr{L}$ est composé des $n!/2$ étiquetages possibles (permutations). Dans notre algorithme de DS une permutation φ est représentée par un vecteur l contenant les n premiers nombres entiers. Ce vecteur est indexé par les sommets du graphe, de cette manière la k-ème valeur $l[k]$ exprime l'étiquette affectée au sommet k. Le coût d'une permutation φ est estimé en employant soit la fonction d'évaluation LA (Équation 5.1), soit la fonction d'évaluation Φ (Équation 5.11).

Solution initiale

La solution initiale pour notre algorithme de DS est générée aléatoirement.

Fonction de voisinage

Soit $swap(\varphi, u, v)$ une fonction permettant d'échanger les étiquettes de deux sommets u et v dans une permutation φ. Le voisinage $\mathcal{N}_1(\varphi)$ d'une permutation φ peut être alors défini comme suit :

$$\mathcal{N}_1(\varphi) = \left\{\varphi' \in \mathscr{L} : swap(\varphi, u, v) = \varphi',\ u, v \in V,\ u \neq v\right\} \tag{6.1}$$

Ce voisinage présente l'avantage de permettre l'évaluation des solutions voisines sans parcourir toutes les arêtes du graphe, c'est-à-dire en mettant à jour uniquement les différences absolues dk des sommets concernés par le mouvement ($A(u)$ et $A(v)$).

Condition d'arrêt

L'algorithme commence à partir d'une solution initiale $\varphi \in \mathscr{L}$, puis à chaque itération il visite la totalité des voisins $\mathcal{N}_1(\varphi)$ pour choisir le meilleur, uniquement s'il améliore la solution courante. Dans le cas contraire l'algorithme s'arrête.

6.2.2 Conditions expérimentales

Deux objectifs sont visés par les expériences présentées dans cette section. Le premier consiste à analyser la distribution des voisins améliorants produits par notre nouvelle fonction d'évaluation Φ quand elle est couplée à une relation de voisinage donnée. Le deuxième objectif est d'approfondir nos connaissances sur le comportement de la fonction d'évaluation Φ au cours de la recherche.

Afin d'effectuer cette première comparaison expérimentale entre les fonctions d'évaluation LA et Φ, l'algorithme de DS présenté ci-dessus a été implémenté en C. Nous l'appelons DS-LA ou DS-Φ selon la fonction d'évaluation qui est employée. L'algorithme, compilé avec *gcc* utilisant l'option d'optimisation *-O3*, est exécuté séquentiellement sur un cluster de dix nœuds. Chaque nœud est composé d'un processeur Xeon cadencé à 2 GHz disposant de 1 Go de mémoire vive sous Linux.

Pour cette comparaison, nous avons tout d'abord produit dix solutions aléatoires (étiquetages) pour chacun des vingt-et-un graphes appartenant à la première série de tests. Ensuite, ces étiquetages ont été utilisés comme solutions initiales pour les vingt exécutions effectuées pour comparer les deux fonctions d'évaluation.

6.2.3 Résultats expérimentaux

Les résultats moyens des vingt exécutions effectuées pour cette étude comparative sont résumés dans la Table 6.1. Les trois premières colonnes de cette table indiquent le nom du graphe, son nombre de sommets et d'arêtes. Les colonnes quatre à neuf correspondent au nombre total d'itérations I, au meilleur coût atteint en terme de largeur totale des arêtes C et au temps de calcul moyen T en secondes pour les algorithmes DS-LA et DS-Φ. La colonne dix montre le nombre moyen de solutions voisines améliorantes $\mathcal{N}_A$ trouvées par DS-Φ à la même itération lorsque le nombre moyen de voisins améliorants fournis par DS-LA est nul (c'est-à-dire quand DS-LA s'arrête). La dernière colonne Δ_C montre le gain en terme de pourcentage du coût moyen atteint par DS-Φ par rapport à celui produit par DS-LA ($100 * (1 -$ coût de DS-Φ/coût de DS-LA)).

			DS-LA			DS-Φ				%
Graphe	$\|V\|$	$\|E\|$	I	C	T	I	C	T	$\mathcal{N}_A$	Δ_C
randomA1	1000	4974	2116.7	946033.1	41.24	3135.2	941677.7	57.14	234.7	0.460
randomA2	1000	24738	2352.6	6678051.6	312.49	3115.0	6673334.8	462.01	289.2	0.071
randomA3	1000	49820	2461.1	14397879.8	1190.35	2888.3	14396580.9	1351.68	327.4	0.009
randomA4	1000	8177	2198.0	1810393.5	60.00	3041.1	1807026.5	84.39	250.8	0.186
randomG4	1000	8173	2223.1	377994.3	54.40	2366.2	381003.1	57.73	255.8	-0.796
bintree10	1023	1022	1234.6	51716.8	16.24	1407.9	51548.8	18.63	58.2	0.325
hc10	1024	5120	1738.9	648728.1	46.88	1959.9	650515.8	36.25	361.4	-0.276
mesh33x33	1089	2112	2995.4	130751.3	45.67	8144.9	112171.7	123.31	305.6	14.210
3elt	4720	13722	27119.4	2630144.6	7343.34	52062.6	2392981.3	14561.29	1150.8	9.017
airfoil1	4253	12289	23567.4	2184693.3	5622.94	45910.1	1958983.4	10387.07	862.7	10.331
whitaker3	9800	28989	71049.9	11473102.7	88118.78	143499.9	10377562.1	163397.05	1788.1	9.549
c1y	828	1749	1737.9	122959.2	29.79	2502.2	120812.3	32.65	147.7	1.746
c2y	980	2102	2141.9	169434.7	33.99	3079.9	167127.9	52.50	153.4	1.361
c3y	1327	2844	3335.1	282818.5	186.30	5098.4	275529.3	234.98	247.2	2.577
c4y	1366	2915	3410.7	287198.8	88.93	5421.9	280551.9	156.31	249.1	2.314
c5y	1202	2557	2811.0	233286.1	61.07	4104.1	228226.2	95.66	160.5	2.169
gd95c	62	144	70.2	716.3	0.01	82.3	697.4	0.02	14.8	2.637
gd96a	1096	1676	2216.7	148158.3	36.94	3284.5	144751.7	58.02	190.8	2.299
gd96b	111	193	94.3	1857.3	0.03	109.1	1783.9	0.04	20.9	3.954
gd96c	65	125	75.1	692.9	0.01	96.2	657.0	0.01	17.2	5.171
gd96d	180	228	186.2	4166.3	0.11	257.3	4057.4	0.15	26.4	2.614
Moyenne				2027656.1			1950837.2			3.330

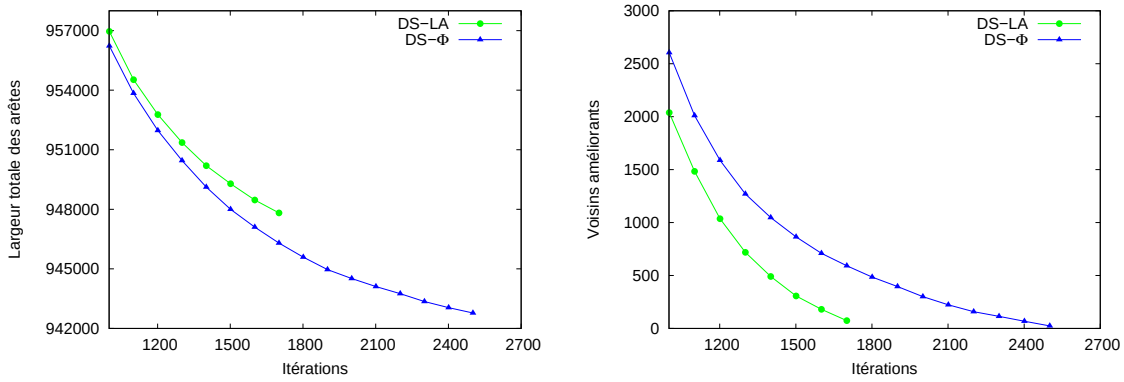
Table 6.1 – Comparaison de performance entre les algorithmes DS-LA et DS-Φ.

Nous pouvons observer clairement dans la Table 6.1 que l'algorithme DS-Φ retourne pour 19 des 21 instances de meilleurs résultats que l'algorithme DS-LA, ce qui mène à une amélioration moyenne Δ_C de 3.330% lorsque Φ est utilisée comme fonction d'évaluation. Nous remarquons aussi que DS-LA arrête le processus de recherche dans tous les cas avant l'algorithme DS-Φ (comparez les colonnes quatre et sept). Ce comportement est dû à l'impossibilité de LA de distinguer des solutions voisines de même coût, et par

conséquent de guider la recherche vers des solutions voisines de meilleure qualité (voir la colonne dix).

En outre, il est important de noter que ces résultats sont fournis sans augmenter le temps de calcul consommé. Le temps de calcul moyen par itération (I/T) de DS-LA est 0.1261 secondes, alors que il est de 0.1203 secondes pour DS-Φ.

La supériorité de Φ est clairement illustrée sur la Figure 6.1, où nous présentons le comportement des fonctions d'évaluation étudiées sur l'instance $randomA1$ (le reste des graphes étudiés fournissent des résultats semblables). Dans la Figure 6.1(a) l'axe des abscisses représente le nombre d'itérations alors que l'axe des ordonnées indique la qualité de la solution moyenne obtenue. La Figure 6.1(b) illustre l'évolution du nombre moyen de voisins améliorants (l'axe des ordonnées) par rapport au nombre d'itérations. Ces deux figures mettent en avant que DS-Φ fournit de meilleurs résultats que DS-LA avec le même nombre d'itérations.

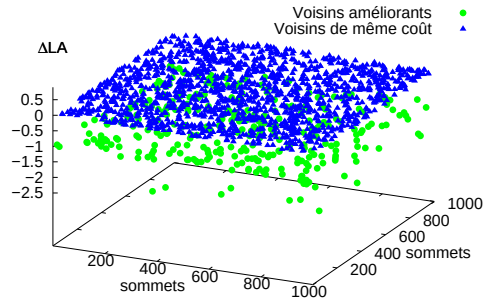


(a) Qualité moyenne de solution

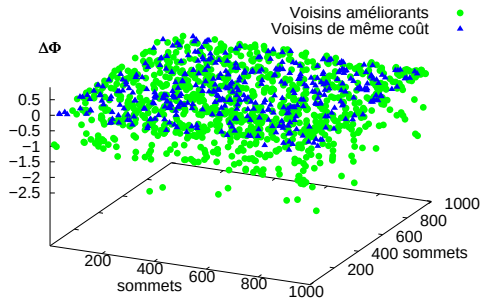
(b) Nombre moyen de voisins améliorants

Figure 6.1 – Comparaison de performance entre les algorithmes DS-LA et DS-Φ sur l'instance $randomA1$.

Ce fait est mis en exergue encore plus clairement sur les Figures 6.2(a) et 6.2(b), où nous illustrons l'ensemble de voisins de même coût et de voisins améliorants produits après 1500 itérations des algorithmes DS-LA et DS-Φ sur l'instance $randomA1$. Nous constatons une différence très nette ; DS-LA affecte la même valeur de coût à un grand nombre de voisins alors que DS-Φ fait une distinction plus fine des ces solutions. Cette différence entre les fonctions d'évaluation comparées s'accentue encore plus à mesure que la recherche avance. Par exemple, observons les Figures 6.3(a) et 6.3(b) qui montrent également l'ensemble de voisins de même coût et de voisins améliorants générés par les algorithmes comparés à l'itération lorsque DS-LA s'arrête. Notez comment Φ discrimine toutes ces solutions voisines de même coût qui sont impossibles à distinguer en utilisant la fonction classique LA.

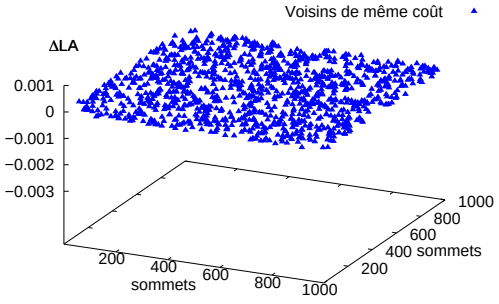


(a) Algorithme DS-LA

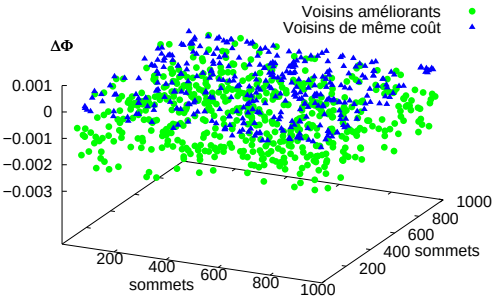


(b) Algorithme DS-Φ

Figure 6.2 – Comparaison entre les voisins produits par DS-LA et DS-Φ sur l'instance $randomA1$ après 1500 itérations.



(a) Algorithme DS-LA



(b) Algorithme DS-Φ

Figure 6.3 – Comparaison entre les voisins produits par DS-LA et DS-Φ sur l'instance $randomA1$ à l'itération lorsque DS-LA s'arrête.

6.3 Comparaison avec un algorithme mémétique

Pour cette deuxième comparaison expérimentale entre les fonctions d'évaluation LA et Φ, nous avons choisi d'implémenter un algorithme de recherche plus efficace que la simple DS utilisée dans la section précédente. Nous avons donc programmé un Algorithme Mémétique (AM), dont les détails sont expliqués ci-dessous.

6.3.1 Algorithme mémétique

Nous avons décidé de conserver notre implémentation de l'AM simple, afin de pouvoir obtenir une idée plus exacte du comportement des fonctions d'évaluation comparées. En effet, l'opérateur de croisement employé ne considère aucune connaissance spé-

cifique du problème[4] et la population est aléatoirement initialisée. De plus, un simple algorithme de DS est employé, afin de réduire la forte influence d'autres algorithmes de RL plus efficaces et sophistiqués.

Représentation et fonction d'évaluation

La représentation ainsi que la fonction d'évaluation (fonction d'aptitude) sont les mêmes que celles utilisées par l'algorithme de DS que nous avons présentées dans la Section 6.2.1.

Initialisation

La population P est initialisée avec $|P|$ étiquetages (configurations) générés aléatoirement.

Sélection

Dans cet AM, les deux parents utilisés lors du croisement sont élus grâce à la sélection par tournoi. Cette technique utilise la sélection proportionnelle sur des paires d'individus, puis choisit pour ces paires l'individu qui a la meilleure valeur d'adaptation. La nouvelle génération est ensuite créée en choisissant les meilleurs individus de l'ensemble formé par les parents et les fils, en faisant attention que chaque individu existe seulement une fois dans la nouvelle population (schéma de sélection $(\mu + \lambda)$).

Opérateur de croisement

Pour cette implémentation nous avons utilisé l'opérateur de croisement appelé PMX (Partially Matched Crossover) introduit par Goldberg et Lingle [1985]. Cet opérateur a été conçu pour préserver les positions absolues dans les chromosomes des deux parents (permutations).

PMX commence par choisir deux points de croisement dans le premier parent, les éléments situés entre ces deux points sont ensuite copiés dans le premier fils. Ce transfert définit également un ensemble de correspondances entre les éléments qui ont déjà été copiés et ceux dans les positions respectives dans le deuxième parent. Ensuite, le reste des éléments sont copiés dans les positions où ils se trouvent dans le deuxième parent. Si une position est déjà occupée par un élément copié du premier parent, l'élément fourni par l'ensemble de correspondances est alors considéré. Ce processus est répété jusqu'à ce que le conflit soit résolu. Cette procédure est également utilisée pour produire le deuxième fils.

[4] Il s'agit d'un opérateur « aveugle » (aléatoire).

Opérateur de recherche locale

Le but de cet opérateur est d'améliorer les individus produits par l'opérateur de croisement. Dans cet AM nous avons décidé d'utiliser une version modifiée de l'algorithme de DS présenté dans la Section 6.2.1. La principale différence est la stratégie de mouvement utilisée. Au lieu de prendre le meilleur voisin, nous choisissons le premier voisin améliorant. Notons que cet algorithme de DS est moins efficace que la version qui utilise la meilleure amélioration, mais il est beaucoup plus rapide. L'opérateur de RL s'arrête donc quand il ne trouve aucun voisin améliorant ou qu'un nombre prédéfini maximal d'itérations est exécuté.

Procédure générale

Notre AM commence par construire une population initiale P. Ensuite, à chaque génération un nombre prédéfini de fils (*nbFils*) sont produits. Pour chaque croisement, deux individus sont choisis de la population courante grâce à une sélection par tournoi, ensuite ils sont recombinés pour produire deux descendants ou fils. L'opérateur de RL est appliqué pour les améliorer pendant un certain nombre L d'itérations. Enfin, la population est mise à jour en sélectionnant les meilleurs individus entre l'ensemble de parents et de fils. Cette procédure itérative continue jusqu'à ce qu'un nombre maximal de générations (*maxGénérations*) soit atteint.

6.3.2 Conditions expérimentales

Pour cette deuxième comparaison empirique, l'AM présenté ci-dessus a été codé en langage *C*, compilé et optimisé (option $-O3$) avec *gcc* sur la machine mentionnée en Section 6.2.2. Nous appelons l'algorithme AM-LA ou AM-Φ selon la fonction d'évaluation qui est utilisée. Les valeurs des paramètres de l'AM ont été fixées empiriquement en considérant les travaux présentés dans [Merz et Freisleben, 2000], et de manière à ce que l'algorithme soit performant, robuste et avec des temps de calcul raisonnables. Les valeurs choisies pour AM-LA et AM-Φ sont :

- taille de la population $|P| = 50$
- nombre de fils par génération *nbFils*$= 5$
- nombre maximal d'itérations de RL $L = 0.20 * |V|$
- nombre maximal de générations *maxGénérations*$= 1000$.

Chaque algorithme ainsi paramétré a été exécuté vingt fois sur les vingt-et-un graphes de la première série d'essai (voir Section 6.1). Les résultats obtenus au cours de ces vingt exécutions ont ensuite été utilisés pour le calcul de certaines mesures statistiques.

6.3.3 Résultats expérimentaux

La Table 6.2 présente les résultats moyens sur vingt exécutions indépendantes sur toutes les instances de la première série de tests. Dans cette table, les trois premières colonnes indiquent le nom du graphe, son nombre de sommets et la solution optimale

C^*, lorsque celle-ci est connue [Diaz *et al.*, 2002]. Pour chaque algorithme comparé (AM-LA et AM-Φ), les colonnes quatre à onze listent le meilleur coût trouvé en terme de largeur totale des arêtes C, le coût moyen $moy.$, son écart type $e.t.$ et le temps d'exécution moyen T en secondes. Enfin, la colonne douze montre le gain Δ_C en terme de pourcentage du meilleur coût atteint par AM-Φ par rapport à celui fourni par AM-LA ($100 * (1 -$ coût de AM-Φ/coût de AM-LA$)$).

			AM-LA				AM-Φ				%
Graphe	$\|V\|$	C^*	C	$moy.$	$e.t.$	T	C	$moy.$	$e.t.$	T	Δ_C
randomA1	1000		874872	882305.9	3833.1	937.7	870174	877033.1	4371.4	930.2	0.537
randomA2	1000		6541549	6558089.1	14804.7	4618.5	6527991	6553267.7	10028.7	4839.8	0.207
randomA3	1000		14230306	14243888.8	11272.2	7843.6	14216058	14235917.8	12236.3	9030.9	0.100
randomA4	1000		1727892	1736126.9	5348.5	1451.6	1724138	1736094.1	5588.9	1531.5	0.217
randomG4	1000		153287	166537.0	9151.0	1191.6	151626	164636.0	5538.5	1375.7	1.084
bintree10	1023	3696	8060	13869.0	5274.3	171.2	6120	13422.3	4119.3	176.7	24.069
hc10	1024	523776	523776	523776.0	0.0	173.5	523776	523776.0	0.0	178.8	0.000
mesh33x33	1089	31680	34932	35151.8	359.3	65.7	34382	35083.4	90.0	129.9	1.574
3elt	4720		445956	458695.8	7230.1	10968.2	441342	453149.4	5176.7	11822.6	1.035
airfoil1	4253		375639	382781.0	2701.0	8855.5	366978	380404.6	4464.6	9515.3	2.306
whitaker3	9800		1306767	1307092.4	138.4	51907.0	1306730	1306867.5	108.0	57196.4	0.003
c1y	828		63767	64531.5	473.4	231.6	63606	64451.5	634.9	227.0	0.252
c2y	980		84170	85240.5	733.6	370.4	82467	84201.9	723.2	405.8	2.023
c3y	1327		135791	137983.9	1027.1	686.7	134341	137281.7	1746.9	705.9	1.068
c4y	1366		121253	122943.4	942.4	747.6	120341	122666.7	551.8	843.6	0.752
c5y	1202		105768	106359.7	563.0	487.9	103497	105733.8	596.5	527.5	2.147
gd95c	62		506	506.2	0.6	0.4	506	506.1	0.4	0.5	0.000
gd96a	1096		100038	102827.8	1759.9	319.9	99349	102285.3	2052.9	348.1	0.689
gd96b	111		1416	1422.0	2.7	1.5	1416	1420.2	2.8	1.7	0.000
gd96c	65		519	520.6	1.0	0.6	519	520.5	1.1	0.7	0.000
gd96d	180		2411	2445.4	18.5	3.8	2407	2429.6	18.3	4.2	0.166
Moyenne			1278032.1	1282528.3		4335.0	1275132	1281007.1		4752.0	1.820

Table 6.2 – Comparaison de performance entre les algorithmes AM-LA et AM-Φ.

De cette table, nous observons que AM-Φ est capable d'améliorer les résultats produits par AM-LA pour 17 des 21 instances d'essai, alors que pour les 4 autre graphes les deux algorithmes trouvent la même solution. Une amélioration moyenne de 1.820% en terme de calité de solution est atteinte grâce à l'utilisation de la nouvelle fonction d'évaluation Φ. Concernant le temps de calcul moyen, nous notons que la durée des exécutions de AM-Φ sont en général 9.62% plus longues que celles de AM-LA.

Enfin, cette expérimentation confirme encore une fois que Φ permet d'améliorer la performance des algorithmes de recherche.

6.4 Algorithme mémétique amélioré pour le problème MinLA

Étant donnés les bons résultats obtenus dans les expérimentations de la Section 6.3.3, nous avons décidé de développer un *Algorithme Mémétique Amélioré* pour MinLA. L'objectif est de mettre en place un algorithme capable de rivaliser avec les heuristiques de l'état de l'art pour ce problème.

6.4.1 Détails d'implémentation

Pour cette implémentation améliorée, que nous appelons AMA-Φ, nous avons pris comme base l'AM décrit dans la Section 6.3.1 qui utilise la fonction d'évaluation Φ. Ensuite, nous avons modifié la procédure d'initialisation de la population, l'opérateur de croisement et l'opérateur de RL. Nous continuons cette section en montrant les détails de ces trois composants modifiés.

Initialisation

La population P est initialisée avec $|P|$ étiquetages. Pour générer chaque étiquetage (configuration), nous utilisons l'Heuristique de Minimisation Frontale Améliorée (MFA) proposée par McAllister [1999], légèrement modifiée pour travailler de manière randomisée. Grâce au comportement aléatoire de notre procédure d'initialisation, les configurations dans la population initiale sont tout à fait différentes. Ce point est très important pour les AM en général parce qu'une population homogène a peu de chance d'évoluer efficacement.

Opérateur de croisement

Le rôle principal de l'opérateur de croisement est de produire des individus diversifiés et potentiellement prometteurs. Pour atteindre cet objectif, un bon opérateur de croisement devrait prendre en compte, autant que possible, des connaissances spécifiques du problème traité [Grefenstette, 1987; Davis, 1991; Bäck *et al.*, 1997b; Moscato, 1999].

Il existe plusieurs opérateurs de croisement rapportés dans la littérature qui peuvent être employés pour des problèmes de permutations [Davis, 1985; Goldberg et Lingle, 1985; Oliver *et al.*, 1987; Freisleben et Merz, 1996b; Whitley *et al.*, 1989]. Cependant, ils présentent un important inconvénient, ils ne sont pas adaptés pour gérer certaines particularités du problème MinLA.

Dans cette section, nous proposons un nouvel opérateur de croisement, appelé *Croisement par Trajectoire* (TX), conçu spécialement pour MinLA. Notre nouveau croisement (TX) est inspiré de la méthode de *Path Relinking* (PR) présentée par Glover et Laguna comme une alternative pour intégrer des stratégies d'intensification et de diversification dans le contexte de la Recherche Tabou [Glover et Laguna, 1997].

L'opérateur TX produit de nouveaux descendants en explorant les *trajectoires* qui relient deux parents (A et B). Il commence d'un parent, appelé *solution initiale*, et génère une trajectoire dans le voisinage de cette solution qui mène vers l'autre parent, appelé *solution guide*. Ce processus est accompli par le choix des mouvements qui présentent des attributs contenus dans la solution guide. Notons que chaque nouvelle solution dans la trajectoire correspond à un nouvel individu. Une caractéristique de cet opérateur de croisement est que les fils produits héritent des éléments communs aux deux parents.

Nous illustrons le fonctionnement de l'opérateur TX avec un exemple montré dans la Figure 6.4. Soient deux parents A et B, et une position de départ aléatoirement générée (marqué avec la flèche), pour cet exemple nous prenons la troisième position. Dans cette

position, nous trouvons les éléments 7 et 5 des parents A et B respectivement. Il y a deux manières possibles de rapprocher ces deux parents entre eux. En permutant les éléments 5 et 7 dans le parent A, ou bien en échangeant les mêmes éléments dans le parent B (Figure 6.4(a)). Les valeurs d'aptitude de ces deux solutions sont calculées, supposons que le fils X retourne le meilleur coût. En conséquence, le fils W est éliminé et la position suivante (quatrième) est considérée dans les deux solutions résultantes (A et X). Dans la Figure 6.4(b), les fils obtenus en échangeant les éléments 1 et 8 dans le parent A et dans le fils X sont représentés. Ensuite, leurs valeurs d'aptitude sont calculées et le fils ayant le plus grand coût est éliminé. Ce processus continue en examinant les positions dans les deux parents de gauche à droite et de manière cyclique jusqu'à ce que toutes aient été considérées.

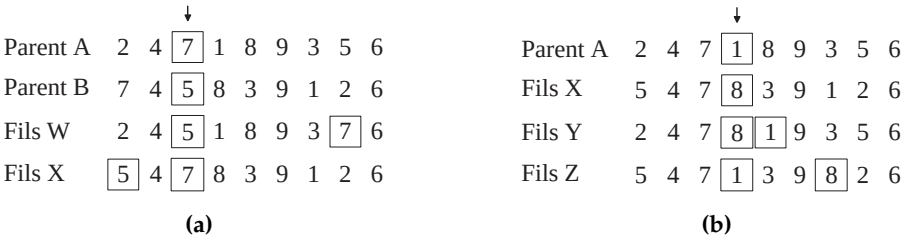


Figure 6.4 – Exemple de l'opérateur de Croisement par Trajectoire (TX).

Opérateur de recherche locale

Dans un AM, toutes sortes de métaheuristiques peuvent être utilisées comme des opérateurs de RL. Dans le cas de notre implémentation, nous avons décidé d'employer un algorithme de Recuit Simulé (RS).

Pour notre opérateur de RL, nous avons implémenté une version, adaptée pour le problème MinLA, de l'algorithme de RS décrit dans la Section 4.3.1 qui utilise le même voisinage $\mathcal{N}_1(\varphi)$ introduit dans la Section 6.2.1. Les principales modifications apportées à cet algorithme sont : le nombre maximum d'étiquetages voisins visités, la condition d'arrêt et le paramétrage.

Nous avons programmé notre opérateur de RL pour visiter un nombre fixe d'étiquetages voisins $maxEss$ à chaque température. Cet opérateur s'arrête si l'une de deux conditions suivantes est vérifiée : si la température courante atteint sa valeur minimale T_f, ou si le nombre total d'itérations de RL L est atteint.

Les valeurs des paramètres de notre opérateur de RL ont été fixées empiriquement de manière à ce que l'algorithme soit rapide et performant. Les valeurs choisies sont :

- température initiale $T_i = 10$
- température finale $T_f = 0.001$
- taux de refroidissement $\alpha = 0.955$
- nombre maximum d'étiquetages voisins visités $maxEss = 1000$.

6.4.2 Conditions expérimentales

L'objectif principal de cette expérience est d'évaluer l'efficacité de notre algorithme mémétique amélioré AMA-Φ par rapport aux heuristiques les plus performantes pour le problème MinLA. Pour atteindre le but de cette troisième phase d'expérimentation, l'algorithme AMA-Φ implémenté en langage *C*, a été compilé sur Linux avec *gcc* en utilisant un niveau d'optimisation *-O3*. Cette démarche ainsi que les exécutions de test ont été réalisées sur le même cluster de dix nœuds que nous avons utilisé précédemment (voir Section 6.2.2).

L'algorithme ainsi implémenté a été paramétré selon les valeurs suivantes :

- taille de la population $|P| = 50$
- nombre de fils par génération *nbFils*$= 5$
- nombre maximal d'itérations de RL $L = 150000$
- nombre maximal de générations *maxGénérations*$= 1000$.

En raison de la nature incomplète et non déterministe d'AMA-Φ, vingt exécutions indépendantes de cet algorithme ont été effectuées sur les 21 instances de la première série de tests (voir Section 6.1). Par la suite, nous présentons les résultats obtenus au cours de ces vingt exécutions et nous les comparons à ceux produits par d'autres méthodes de la littérature.

6.4.3 Résultats expérimentaux

Dans cette expérience nous effectuons une comparaison de performance entre notre algorithme AMA-Φ et les quatre méthodes les plus efficaces rapportées dans la littérature : OS+RS [Petit, 2003a; Petit, 2003b], ABD+RS [Bar-Yehuda *et al.*, 2001], MC [Koren et Harel, 2002] et AMG [Safro *et al.*, 2006].

Étant donné que chacun de ces algorithmes a été exécuté dans une machine distincte, les temps de calcul des algorithmes étudiés ne peuvent pas être directement con-frontés. Cependant, nous avons utilisé les données de la comparaison de processeurs[5] mentionnée dans la Section 4.4.4 afin d'obtenir un facteur d'équivalence pour chacun des ces quatre ordinateurs (Table 6.3). De tells facteurs d'équivalence ont été ensuite utilisés pour corriger les temps de calcul consommés pour ces algorithmes.

Algorithme	Processeur	Facteur d'équivalence
OS+RS	K6 III à 600 MHz	4.96
ABD+RS	Pentium III à 600 MHz	3.02
MC	Pentium III à 700 MHz	2.59
AMG	Pentium IV à 1.7 GHz	1.22
AMA-Φ	Xeon à 2.0 GHz	1.00

Table 6.3 – Facteurs d'équivalence pour corriger les temps de calcul consommés par les algorithmes comparés : OS+RS, ABD+RS, MC, AMG et AMA-Φ.

Nous résumons dans la Table 6.4 les meilleurs résultats fournis par les algorithmes

[5] `http://www.tomshardware.fr`

OS+RS, ABD+RS, MC et AMG ; ils ont été tirés des articles correspondants. Pour chaque algorithme, cette table présente le meilleur coût C, le temps de calcul T en secondes consommé pour son obtention, ainsi que le temps de calcul corrigé T_c.

	OS+RS			ABD+RS			MC			AMG		
Graphe	C	T	T_c	C	T	T_c	C	T	T_c	C	T	T_c
randomA1	884261	275.4	55.5	884261	2328.0	776.0	909126	22.9	8.8	888381	105.4	86.4
randomA2	6528780	577.8	116.4	6576912	191645.0	63881.7	6606174	63.9	24.6	6596081	321.6	263.6
randomA3	14310861	682.2	137.4	14289214	58771.0	19590.3	14457452	109.2	42.0	14303980	431.9	354.0
randomA4	1721670	400.2	80.6	1747143	10896.0	3632.0	1765217	30.0	11.6	1747822	189.0	154.9
randomG4	146996	267.0	53.8	146996	8376.0	2792.0	149513	19.0	7.3	140211	31.2	25.6
bintree10	4069	150.6	30.3	3762	60.0	20.0	3950	8.0	3.1	3696	8.5	7.0
hc10	548352	152.4	30.7	523776	2545.0	848.3	523776	18.5	7.1	523776	85.4	70.0
mesh33x33	34515	537.0	108.2	33531	278.0	92.7	31729	9.9	3.8	31729	19.3	15.8
3elt	363204	9216.0	1856.8	363204	5757.0	1919.0	373464	61.2	23.6	357329	111.7	91.6
airfoil1	277412	7416.0	1494.1	289217	4552.0	1517.3	291794	61.0	23.5	272931	95.4	78.2
whitaker3	1151064	41148.0	8290.1	1200374	29937.0	9979.0	1205919	127.8	49.2	1144476	280.3	229.7
c1y	62936	108.0	21.8	62333	205.0	68.3	64934	8.9	3.4	62262	10.5	8.6
c2y	79429	183.0	36.9	79571	299.0	99.7	80148	10.8	4.2	78822	12.9	10.6
c3y	123548	436.0	87.8	127065	553.0	184.3	127315	15.3	5.9	123514	18.2	14.9
c4y	115222	524.0	105.6	115222	580.0	193.3	118437	15.6	6.0	115131	19.6	16.1
c5y	96965	325.0	65.5	96956	446.0	148.7	104076	13.3	5.1	96899	16.5	13.5
gd95c	509	0.4	0.1	506	2.0	0.7	509	0.6	0.2	506	1.7	1.4
gd96a	96342	250.8	50.5	99944	268.0	89.3	106668	11.9	4.6	96249	15.3	12.5
gd96b	1422	0.8	0.2	1422	3.0	1.0	1434	1.0	0.4	1416	1.9	1.5
gd96c	519	0.4	0.1	519	1.0	0.3	519	0.6	0.2	519	1.8	1.5
gd96d	2409	2.6	0.5	2409	4.0	1.3	2420	1.5	0.6	2391	2.4	1.9
Moyenne	1.2643E+6	2983.50	601.09	1.2688E+6	15119.33	5039.78	1.2821E+6	29.10	11.19	1.2661E+6	84.78	69.49

Table 6.4 – Meilleurs résultats produits par les quatre algorithmes les plus efficaces pour MinLA, considérant des temps de calcul corrigés.

Les résultats produits au cours de la comparaison expérimentale entre notre algorithme AMA-Φ et les algorithme OS+RS, ABD+RS, MC et AMG sont listés en détail dans la Table 6.5. Les deux premières colonnes de cette table indiquent le nom du graphe et son nombre de sommets. La colonne trois montre la meilleure solution connue C^* rapportée dans la littérature en terme de largeur totale des arêtes, tandis que la colonne quatre présente la référence où ce résultat a été trouvé. Les colonnes cinq à neuf listent les résultats moyens obtenus par AMA-Φ sur vingt exécutions, dans l'ordre suivant : la meilleure solution trouvée C_m, le pire coût en terme de largeur totale des arêtes C_p, le coût moyen $moy.$, son écart type $e.t.$ et le temps moyen d'exécution en secondes T. La dernière colonne liste le gain Δ_C en terme de pourcentage du meilleur coût C_m atteint par AMA-Φ par rapport à la meilleure solution connue C^* ($100 * (1 - C_m/C^*)$).

Nous observons dans cette table que notre approche est très compétitive. En effet, AMA-Φ permet d'améliorer les résultats de 7 meilleures solutions connues, et de les égaler pour 8 autres. Pour les 6 graphes restants de cette série de tests, AMA-Φ n'atteint pas la meilleure solution connue. De cette manière, l'algorithme AMA-Φ permet d'obtenir une amélioration moyenne de 0.106% par rapport aux meilleures solutions connues C^* (1.2577E+6 contre 1.2608E+6). Nous remarquons que la solution moyenne $moy.$ fournie par AMA-Φ est également compétitive comparée avec les meilleures solutions connues

				AMA-Φ					%
Graphe	$\|V\|$	C^*	Référence	C_m	C_p	$moy.$	$e.t.$	T	Δ_C
randomA1	1000	884261	1,2	867214	868779	867581.7	634.2	909.0	1.928
randomA2	1000	6528780	1	6532341	6537824	6534770.7	2616.2	3486.3	-0.055
randomA3	1000	14289214	2	14238712	14241628	14239766.4	1213.0	5175.7	0.353
randomA4	1000	1721670	1	1718746	1722685	1720260.5	1479.7	1904.1	0.170
randomG4	1000	140211	4	140211	141090	140420.7	354.1	2077.2	0.000
bintree10	1023	3696	4	3721	3797	3749.7	36.3	978.7	-0.676
hc10	1024	523776	2,3,4	523776	523778	523776.2	0.6	1140.8	0.000
mesh33x33	1089	31729	3,4	31789	31982	31847.5	68.8	1123.2	-0.189
3elt	4720	357329	4	357329	364096	361174.7	2198.0	5609.9	0.000
airfoil1	4253	272931	4	273090	291844	278259.8	7063.0	5443.1	-0.058
whitaker3	9800	1144476	4	1148652	1169481	1160117.7	9207.0	15299.7	-0.365
c1y	828	62262	4	62262	62478	62302.7	65.0	643.5	0.000
c2y	980	78822	4	78929	79017	78964.2	45.4	654.8	-0.136
c3y	1327	123514	4	123376	123624	123458.5	92.3	728.1	0.112
c4y	1366	115131	4	115051	115646	115129.1	185.7	733.3	0.069
c5y	1202	96899	4	96878	97989	97080.5	391.9	715.3	0.022
gd95c	62	506	2,4	506	507	506.1	0.3	1.6	0.000
gd96a	1096	96249	4	95242	96780	96019.4	559.9	636.8	1.046
gd96b	111	1416	4	1416	1417	1416.1	0.3	3.7	0.000
gd96c	65	519	1,2,3,4	519	523	519.7	1.3	1.4	0.000
gd96d	180	2391	4	2391	2394	2391.5	1.1	7.7	0.000
Moyenne		1.2608E+6		1.2577E+6	1.2608E+6	1.2590E+6		2251.12	0.106

1. OS+RS [Petit, 2003a; Petit, 2003b] 2. ABD+RS [Bar-Yehuda *et al.*, 2001] 3. MC [Koren et Harel, 2002]
4. AMG [Safro *et al.*, 2006]

Table 6.5 – Comparaison des performances entre AMA-Φ et les algorithmes les plus performants pour MinLA.

C^* (1.2590E+6 contre 1.2608E+6).

En ce qui concerne les temps d'exécution des algorithmes comparés, nous avons remarqué que l'algorithme AMA-Φ, étant donné qu'il est un algorithme mémétique, consomme considérablement plus de temps de calcul que les autre heuristiques, notamment ABD+RS [Bar-Yehuda *et al.*, 2001], MC [Koren et Harel, 2002] et AMG [Safro *et al.*, 2006].

6.4.4 Discussion

Il est désormais bien connu qu'il est essentiel d'incorporer dans les AM une certaine forme de connaissance du problème pour améliorer leurs performances [Davis, 1991; Bäck *et al.*, 1997b; Hoos et Stützle, 2004]. Nous avons choisi de le faire en employant, à la place des opérateurs aléatoires de croisement et de mutation, des opérateurs génétiques spécialement conçus pour tirer avantage de ces informations. Par la suite, nous analysons les influences des opérateurs de croisement et de recherche locale.

Influence de l'opérateur de croisement

Dans cette section nous comparons la performance de notre nouvel opérateur de croisement TX par rapport à quatre autres opérateurs de croisement existants : Ordonné (Order Crossover OX) [Davis, 1985], Partiellement Couplé (Partially Matched Crossover PMX) [Goldberg et Lingle, 1985], par Cycles (Cycle Crossover CX) [Oliver *et al.*, 1987] et Préservateur de Distance (Distance Preserving Crossover DPX) [Freisleben et Merz, 1996b]. Cette étude comparative a pour objectif de mieux comprendre les influences du nouvel opérateur de croisement TX sur la performance générale de notre AM. C'est pourquoi elle tient compte de deux aspects : la capacité à produire de nouveaux individus potentiellement prometteurs et la capacité à préserver la diversité de la population. Les deux caractéristiques sont très importantes dans le processus de recherche parce qu'elles représentent un compromis entre la capacité d'exploration et d'exploitation de l'algorithme. Pour le premier critère, nous avons utilisé l'aptitude moyenne de la population, alors que pour le deuxième nous avons calculé la diversité de la population avec la mesure d'entropie proposée par Grefenstette [1987]. Cette mesure est montrée dans l'Équation 6.2, où n_{ij} représente le nombre de fois où la variable i prend la valeur j dans la population P. Cette fonction retourne des valeurs dans l'intervalle $[0, 1]$. Une entropie égale à zéro indique que tous les individus de la population sont identiques.

$$\mathscr{E}(P) = \frac{-\sum_{i=1}^{n}\sum_{j=1}^{n}(\frac{n_{ij}}{|P|})\log(\frac{n_{ij}}{|P|})}{n\log n} \tag{6.2}$$

Afin de permettre une comparaison équitable, les opérateurs de croisement ont été testés dans les mêmes conditions sur trois instances représentatives de la première série de tests : $randomA1$, $c2y$ et $mesh33x33$. Notez que le paramétrage utilisé pour l'algorithme mémétique dans cette expérimentation est très différent de celui employé dans la Section 6.4.3. La raison est que nous cherchons à observer le comportement des opérateurs de croisement comparés et non à produire les meilleurs résultats. Les paramètres utilisées sont :

- taille de la population $|P| = 100$
- nombre de fils par génération *nbFils*$= 50$
- nombre maximal d'itérations de RL $L = 500$
- nombre maximal de générations *maxGénérations*$= 1000$

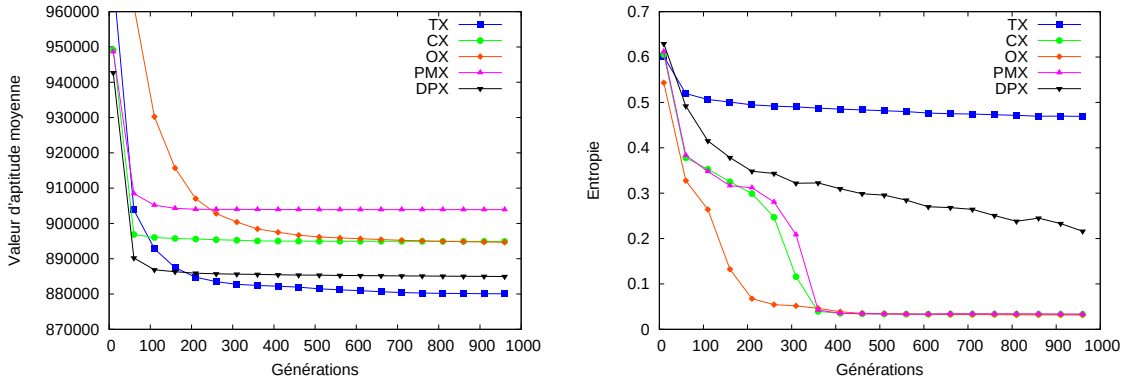
Un nombre relativement petit d'itérations de RL est employé afin de réduire le plus possible la forte influence de la RL dans les résultats.

En raison de la nature non déterministe de l'algorithme, vingt exécutions indépendantes ont été effectuées pour chaque paire instance/opérateur. Les résultats de ces exécutions sont récapitulés dans la Table 6.6. Pour chaque paire instance/opérateur nous présentons la valeur d'aptitude moyenne de population C et son entropie moyenne $\mathscr{E}$ après 1000 générations. Cette table montre clairement que l'opérateur de croisement TX permet d'obtenir de meilleurs résultats pour les trois graphes étudiés, tout en conservant la diversité de la population. Il est important de noter que dans nos expérimentations,

	randomA1		mesh33x33		c2y	
Opérateur	C	$\mathscr{E}$	C	$\mathscr{E}$	C	$\mathscr{E}$
OX	894603.0	0.031	35054.6	0.031	89088.3	0.033
PMX	916023.3	0.334	35952.3	0.218	89193.0	0.223
DPX	899313.6	0.208	34975.3	0.258	84651.3	0.296
CX	891294.0	0.331	35041.6	0.032	84673.3	0.035
TX	881023.0	0.469	34827.0	0.305	83865.0	0.277

Table 6.6 – Comparaison entre différents opérateurs de croisement.

nous avons obtenu des résultats similaires pour les autres dix-huit instances de la série de tests.



(a) Aptitude moyenne **(b)** Entropie moyenne

Figure 6.5 – Comparaison de performance entre cinq différents opérateurs de croisement sur l'instance $randomA1$.

Nous illustrons également les résultats de cette étude comparative sur la Figure 6.5, où le comportement des opérateurs de croisement étudiés sur l'instance $randomA1$ est présenté. Dans la Figure 6.5(a) l'axe des abscisses représente le nombre de générations, alors que l'axe des ordonnées indique la valeur d'aptitude moyenne de la population. La Figure 6.5(b) illustre l'évolution de l'entropie (l'axe des ordonnées) par rapport au nombre de générations. Ces deux figures montrent que le meilleur compromis entre la capacité d'exploration et d'exploitation de l'algorithme est fourni par notre opérateur de croisement TX.

Interaction entre le voisinage et la fonction d'évaluation dans l'opérateur de recherche locale

Nous avons observé dans la Section 2.1 que la relation de voisinage et la fonction d'évaluation sont deux éléments essentiels qui ensemble déterminent le comportement de tout algorithme de recherche basé sur eux [Stadler, 1992; Duvivier *et al.*, 1996; Hertz et Widmer, 2003]. L'objectif premier de cette expérience est d'évaluer l'interaction de ces deux éléments et son influence sur la performance de l'opérateur de RL (basé sur un

algorithme de Recuit Simulé) que nous utilisons dans notre AM.

Afin d'atteindre cet objectif, nous considérons les fonctions d'évaluation LA et Φ ainsi que les voisinages $\mathcal{N}_1(\varphi)$ (Section 6.2.1) et $\mathcal{N}_2(\varphi)$. Ce dernier est le voisinage basé sur de mouvements de rotation introduit en Section 4.4.1 qui s'est montré très efficace pour le problème BMP.

Pour réaliser cette expérience nous avons procédé comme suit. Dix exécutions indépendantes de l'opérateur de RL ont été effectuées pour chaque paire formée entre la fonction d'évaluation et la relation de voisinage, sur les vingt-et-une instances de la première série de tests (voir Section 6.1). Les résultats de ces exécutions sont récapitulés dans la Figure 6.6 qui montre la qualité moyenne de solution obtenue sur l'instance $randomA1$ (l'axe des ordonnées) par rapport aux nombre de mouvements effectués pour chaque paire fonction d'évaluation/voisinage étudiée (les vingt autres instances étudiées fournissent des résultats semblables).

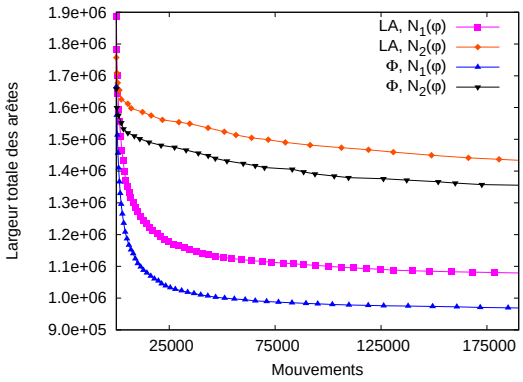


Figure 6.6 – Interaction entre le voisinage et la fonction d'évaluation sur la performance de l'opérateur de RL (utilisé dans l'algorithme AMA) appliqué à l'instance $randomA1$.

De ce graphique nous observons tout d'abord que l'utilisation de la fonction d'évaluation Φ permet à l'opérateur de RL de produire de meilleurs résultats ; ce qui montre que Φ est une meilleure fonction d'évaluation que la fonction classique LA.

Nous avons remarqué également que le voisinage $\mathcal{N}_2(\varphi)$ qui utilise des mouvements de rotation produit toujours des résultats de moindre qualité que ceux fournis par $\mathcal{N}_1(\varphi)$. Bien que le voisinage $\mathcal{N}_2(\varphi)$ s'est montré le plus efficace pour le problème BMP, il ne semble pas approprié pour le problème MinLA. En effet, les mouvements utilisés par ce type de voisinage occasionnent trop de changements dans les étiquetages produisant en général des solutions voisines de moindre qualité. Ceci n'est pas surprenant car même si les deux problèmes se ressemblent, ils ne poursuivent pas le même objectif.

Concernant les quatre paires entre les fonctions d'évaluation et les voisinages, nous observons que la meilleure est $(\Phi, \mathcal{N}_1(\varphi))$ car elle permet à l'opérateur de RL d'obtenir une qualité de solution de 968979.12 en terme de largeur totale des arêtes. Les trois autres paires produisent les résultats suivants : $(\Phi, \mathcal{N}_2(\varphi))$ 1356107.56, $(\text{LA}, \mathcal{N}_1(\varphi))$ 1079163.45 et $(\text{LA}, \mathcal{N}_2(\varphi))$ 1433319.57.

Ces observations nous ont permis de choisir la meilleure paire fonction d'évaluation/voisinage pour notre opérateur de RL, constituée par la fonction d'évaluation Φ et le voisinage $\mathcal{N}_1(\varphi)$).

6.5 Recuit simulé en deux phases pour le problème MinLA

Nous avons constaté que l'algorithme AMA-Φ, décrit en Section 6.4, comporte certaines limites comme la taille des instances qui peuvent être traitées ou le temps de calcul consommé qui peut s'avérer assez long. C'est pour cette raison que nous avons décidé de diriger nos efforts de recherche vers le développement d'un algorithme de *Recuit Simulé en Deux Phases* pour le problème MinLA.

6.5.1 Détails d'implémentation

Le RS est un algorithme stochastique d'optimisation qui s'est avéré très efficace pour résoudre de nombreux problèmes NP-difficiles. Cependant, il présente en général un grand inconvénient : il nécessite un temps de calcul considérable. En conséquence, la réduction du temps de calcul consommé par le RS a été un domaine de recherche très actif depuis son introduction [Kirkpatrick *et al.*, 1983]. La plupart des études se sont concentrées sur le développement de :

- schémas de refroidissement plus rapides [Aarts et Laarhoven, 1985; Lam et Delosme, 1988; Huang *et al.*, 1986] ;
- types de mouvements alternatifs [Grover, 1986] ;
- stratégies d'acceptation [Greene et Supowit, 1988] ;
- schémas de changement de la température en temps fini [Boese et Kahng, 1994; Hajek, 1988; Strenski et Kirkpatrick, 1991].

Une autre approche est le *Recuit Simulé en Deux Phases* (RSDP) [Grover, 1987; Johnson *et al.*, 1989; Rose *et al.*, 1990; Varanelli et Cohoon, 1999].

Dans le RSDP, une heuristique rapide est employée pour remplacer les actions effectuées par un RS traditionnel à températures élevées. Cette heuristique est suivie d'un processus d'amélioration basé sur un algorithme de RS conventionnel qui est initialisé à une température plus basse que la normale. La détermination de cette température initiale représente un des éléments les plus importants dans la conception de ce type d'algorithme.

Nous poursuivons cette section en présentant l'implémentation que nous avons faite d'un algorithme de RSDP pour résoudre le problème de MinLA. Cet algorithme, que nous appelons RSDP-Φ, tire avantage de la nouvelle fonction d'évaluation Φ. De plus, il améliore certaines caractéristiques qui déterminent de manière considérable son efficacité : une heuristique pour produire rapidement des solutions initiales de bonne qualité, une fonction de voisinage spécialisée et un schéma de refroidissement avancé.

Représentation interne

Dans cette implémentation nous avons utilisé simultanément deux représentations. La première correspond à celle présentée dans la Section 4.4.1, où un étiquetage est décrit par un vecteur contenant n nombres entiers, et indexé par les étiquettes affectées aux sommets du graphe. La deuxième représentation interne, est la même employée par l'algorithme de DS présenté dans la Section 6.2.1, c'est-à-dire un vecteur de taille n qui est indexé par les sommets du graphe. L'objectif d'utiliser les deux types de représentations dans le même algorithme est d'accélérer l'évaluation des solutions voisines.

Fonction de voisinage

Définissons la contribution partielle d'un sommet u à la largeur totale des arêtes d'un étiquetage φ comme suit : $L(u,\varphi) = \sum_{v \in A(u)} |\varphi(u) - \varphi(v)|$, où $A(u)$ est l'ensemble de sommets adjacents de u. Soit $swap(\varphi, u, v)$ une fonction permettant d'échanger les étiquettes de deux sommets u et v dans une permutation φ.

Nous avons observé que la meilleure manière d'étiqueter un sommet u, afin de réduire la valeur courante de $L(u,\varphi)$, peut être trouvée en employant la notion de *médiane statistique* exprimée dans la Formule 6.3. Dans cette formule, d représente le degré du sommet u, c'est-à-dire $A(u) = \{v_1, v_2, ..., v_d\}$, et $s_1 < s_2 < ... < s_d$ le résultat de leurs étiquettes triées. On appelle s_k le k-ème ordre statistique de cette série de données [Hogg et Craig, 1970].

$$\text{médiane}(u) = \begin{cases} s_{((d+1)/2)} & \text{si } d \text{ est impair} \\ \frac{1}{2}\left(s_{(d/2)} + s_{(1+d/2)}\right) & \text{si } d \text{ est pair} \end{cases} \tag{6.3}$$

En s'appuyant sur cette observation, un ensemble $M(u)$ de sommets dont les étiquettes courantes sont proches de la valeur médiane(u) peut être construit. Parmi les étiquettes des sommets dans l'ensemble $M(u)$, nous pouvons trouver le choix le plus approprié pour la nouvelle étiquette de u par rapport à leur sommets adjacents. $M(u)$ peut être défini formellement comme suit :

$$M(u) = \{v : \text{médiane}(u) - 2 \leq \varphi(v) \leq \text{médiane}(u) + 2\} \tag{6.4}$$

À titre d'exemple de ce concept, considérons la Figure 6.7 qui représente un sous-graphe issu d'un graphe contenant trente sommets. Sur la figure nous observons le sommet u et ses 5 sommets adjacents. L'étiquetage courant est représenté comme un nombre à l'intérieur de chaque sommet. Il est facile d'observer que la contribution partielle du sommet u à la largeur totale des arêtes est $L(u,\varphi) = 70$. Ensuite, en utilisant l'Équation 6.3 nous obtenons la valeur médiane$(u) = 26$, ainsi $M(u)$ contient tous les sommets qui possèdent actuellement une étiquette entre 24 et 28 (certains d'entre eux ne sont pas montrés dans la Figure 6.7). En examinant chacune des cinq étiquettes possibles pour u, nous pouvons observer que le meilleur choix correspond aux sommets avec les étiquettes 25 et 27. Tous les deux ramènent la contribution partielle de u à $L(u,\varphi) = 51$. Par exemple, si l'étiquette 1 est affectée au sommet u, la valeur maximale de $L(u,\varphi) = 87$ est obtenue

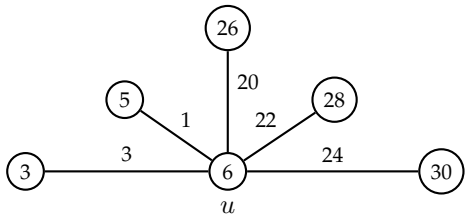


Figure 6.7 – Sous-graphe issu d'un graphe étiqueté contenant trente sommets.

car cette étiquette produit la différence absolue la plus grande par rapport à la valeur médiane(u) dans le graphe, c'est-à-dire 25.

Une fois ce point clarifié, nous pouvons maintenant définir le voisinage $\mathcal{N}_3(\varphi)$ d'une permutation φ dans notre implémentation comme suit :

$$\mathcal{N}_3(\varphi) = \left\{\varphi' \in \mathscr{L} : swap(\varphi, u, v) = \varphi',\ u, v \in V,\ v \in M(u)\right\} \tag{6.5}$$

Nous avons observé dans les expériences présentées dans la Section 6.5.4 que la fonction de voisinage $\mathcal{N}_3(\varphi)$ permet de réduire très rapidement la largeur totale des arêtes du graphe. Cependant, ceci représente un éventuel inconvénient car l'algorithme RSDP-Φ peut rester longtemps bloqué dans un minimum local à cause de la faible capacité d'exploration de $\mathcal{N}_3(\varphi)$. C'est pour cette raison que nous avons décidé de combiner $\mathcal{N}_3(\varphi)$ avec la fonction de voisinage $\mathcal{N}_1(\varphi)$ (présentée en Section 6.2.1) afin d'obtenir un meilleur compromis entre l'exploration et l'exploitation.

Au cours de la recherche nous utilisons donc une combinaison des voisinages $\mathcal{N}_3(\varphi)$ et $\mathcal{N}_1(\varphi)$. La fonction de voisinage $\mathcal{N}_3(\varphi)$ est appliquée avec une probabilité p, alors que $\mathcal{N}_1(\varphi)$ est utilisé le reste du temps. La fonction de voisinage combinée $\mathcal{N}_4(\varphi)$ est exprimé dans la Formule 6.6, où x est un nombre aléatoire dans l'intervalle $[0, 1]$.

$$\mathcal{N}_4(\varphi, x) = \begin{cases} \mathcal{N}_3(\varphi) & \text{si } x \leq p \\ \mathcal{N}_1(\varphi) & \text{si } x > p \end{cases} \tag{6.6}$$

Solution initiale

Après avoir comparé plusieurs heuristiques pour MinLA, tant en qualité de solution qu'en temps de calcul utilisé, nous avons retenu la méthode MFA proposée par McAllister [1999]. Ce choix est entièrement justifié par le fait que cette heuristique atteint une qualité de solution élevée en utilisant un temps d'exécution très raisonnable (linéaire par rapport au nombre d'arêtes dans le graphe).

Détermination de la température initiale

Nous avons opté pour initialiser la température de l'algorithme RSDP-Φ en utilisant la méthode proposée par Varanelli et Cohoon [1999]. Cette méthode est basée sur les études numériques à grande échelle effectuées par différents auteurs [Aarts et Korst,

1989; Hajek, 1988; Otten et Van Ginneken, 1988; White, 1984] pour examiner les densités des qualités de solutions à différentes températures du RS. Ces investigations ont permis de déterminer le comportement typique du coût espéré C_k et de son écart type σ_k par rapport à la température T_k du RS, étant donné un schéma de refroidissement homogène. De plus, ces études ont démontré indépendamment que la distribution de probabilité des valeurs de coût peut être approximée par une distribution normale qui présente les comportements suivants :

$$C_k \approx C_\infty - (\sigma_\infty^2 / T_k) \tag{6.7}$$

$$\sigma_k \approx \sigma_\infty \tag{6.8}$$

où C_∞ et σ_∞ représentent respectivement le coût espéré et son écart type sur l'espace de recherche.

Partant de ces informations Varanelli et Cohoon ont proposé une équation qui permet d'approximer la température $T_k(i)$ à laquelle une solution i de coût $c(i)$ serait trouvée comme étant la meilleure solution courante :

$$T_k(i) \approx \frac{\sigma_\infty^2}{C_\infty - c(i) - \gamma_\infty \sigma_\infty} \tag{6.9}$$

Dans l'Équation 6.9, le paramètre γ_∞ représente le décalage entre le coût espéré C_k et la meilleure solution courante trouvée $c(i)$ à la température T_k. Ce décalage peut être calculé de manière probabiliste en employant la formule suivante, où r est le nombre de mouvements générés à chaque température :

$$P[C_\infty - \gamma_\infty \sigma_\infty < X < C_\infty + \gamma_\infty \sigma_\infty] \approx 1 - |r|^{-1} \tag{6.10}$$

Pour une explication plus détaillée sur la dérivation de ces équations et leurs démonstrations le lecteur peut se rapporter à l'article suivant [Varanelli et Cohoon, 1999].

Dans notre implémentation, nous avons procédé comme suit : d'abord 10^3 solutions aléatoires indépendantes ont été générées, tout en calculant la moyenne et l'écart type de ces données. Ces valeurs sont utilisées comme les approximations du coût espéré C_∞ et de son écart type σ_∞ sur l'espace de recherche. Une solution heuristique avec un coût $c(i)$ est ensuite créée en utilisant la méthode MFA [McAllister, 1999]. Le décalage γ_∞ entre le coût espéré C_k et la meilleure solution trouvée courante $c(i)$ est alors calculé à l'aide de la Formule 6.10. Enfin, les valeurs C_∞, σ_∞, $c(i)$ et γ_∞ sont utilisées dans l'Équation 6.9 afin d'obtenir l'approximation de la température initiale $T_k(i)$ qui sera utilisée dans la deuxième phase de l'algorithme RSDP-Φ.

Schéma de refroidissement

Pour cette implémentation du RSDP nous avons choisi d'utiliser un schéma de refroidissement statistique [Aarts et Laarhoven, 1985] car cette manière de faire varier la température a montré qu'elle peut être très efficace pour résoudre plusieurs problèmes d'optimisation combinatoire [Aarts et Korst, 1989; Poupaert et Deville, 2000].

Dans l'algorithme RSDP-Φ le schéma de refroidissement commence à la température initiale estimée T_i qui est calculée avec la Formule 6.9. Ensuite, à la fin de chaque palier, la température courante est décrémentée en utilisant la relation suivante :

$$T_k = T_{k-1}\left(1 + \frac{\ln(1+\vartheta)T_{k-1}}{3\sigma_{T_{k-1}}}\right)^{-1} \tag{6.11}$$

où $\sigma_{T_{k-1}}$ est l'écart type des valeurs de coût visitées à la température courante et ϑ est appelé le *paramètre de distance*. Une petite valeur de ϑ produit de petites diminutions de la température. Pour chaque température, le nombre maximal d'étiquetages voisins générés est r, il dépend directement du nombre des arêtes du graphe ($|E|$), puisqu'un nombre plus important de mouvements est nécessaire pour des graphes denses.

Condition d'arrêt

L'algorithme RSDP-Φ s'arrête quand la valeur moyenne du coût montre seulement de très petites variations. Dans la pratique, ceci est implémenté suivant la technique proposée par Aarts et Korst [1989], en calculant la dérivée de la valeur moyenne atténuée de la fonction d'évaluation $\bar{C}_k$. Ainsi, l'algorithme se termine si à une certaine température T_k la condition suivante est vérifiée : $\bar{C}_k < \epsilon$, où ϵ est une petite valeur positive appelée le *facteur d'arrêt*.

6.5.2 Conditions expérimentales

Tout comme dans la Section 6.4, nous cherchons dans cette série d'expérimentations à évaluer les performances de l'algorithme RSDP-Φ par rapport aux heuristiques de l'état de l'art pour le problème MinLA. Afin d'accomplir ces expériences, l'algorithme RSDP-Φ implémenté en langage C a été compilé et optimisé sur la machine décrite dans la Section 6.2.2.

Dans cette expérimentation nous avons utilisé les deux séries de tests présentées dans la Section 6.1. Pour chaque instance d'essai, nous avons effectué dix exécutions indépendantes. En conséquence, lorsque des résultats moyens sont présentés ils sont basés sur ces dix exécutions. Dans toutes ces expériences l'algorithme RSDP-Φ a été paramétré de la manière suivante :

- Température initiale T_i calculée avec la procédure décrite dans la Section 6.5.1.
- Paramètre de distance du schéma de refroidissement $\vartheta = 0.10$.
- Nombre maximal d'étiquetages voisins générés par température r :

$$r = \begin{cases} \text{5.0E+05} & 1 < |E| \leq 500 \\ \text{2.0E+06} & 501 < |E| \leq 50000 \\ \text{3.5E+06} & 50001 < |E| \leq \text{1.1E+06} \\ \text{7.0E+06} & |E| > \text{1.1E+06} \end{cases} \tag{6.12}$$

- Facteur d'arrêt $\epsilon =$ 1.0E+10.
- Fonction de voisinage $\mathcal{N}_4(\varphi, x)$ appliquée avec une probabilité $p = 0.90$.

6.5.3 Résultats expérimentaux

Cette comparaison expérimentale a été réalisée en deux parties. La première confronte notre algorithme RSDP-Φ aux heuristiques de l'état de l'art en utilisant la série de tests présentée dans [Petit, 2003b]. La seconde compare TSSA-Φ à deux algorithmes rapportés dans la littérature spécialisés dans les instances d'essai de très grande taille.

Comparaison entre RSDP-Φ et les heuristiques de l'état de l'art

Dans la première partie de cette comparaison expérimentale, les performances de notre algorithme RSDP-Φ sont confrontées à celles produites par les méthodes suivantes : OS+RS [Petit, 2003a; Petit, 2003b], ABD+RS [Bar-Yehuda *et al.*, 2001], MC [Koren et Harel, 2002], AMG [Safro *et al.*, 2006] et AMA-Φ [Rodriguez-Tello *et al.* 2006c]. Pour un résumé détaillé des meilleurs résultats fournis par ces algorithmes nous invitons le lecteur à se reporter à la Table 6.4 qui indique également les temps de calcul corrigés (voir Table 6.3) par rapport à l'ordinateur utilisé pour réaliser nos expériences.

Les résultats produits au cours de la comparaison des performances entre notre algorithme RSDP-Φ et les algorithmes OS+RS, ABD+RS, MC, AMG et AMA-Φ sont montrés dans la Table 6.7. La colonne trois de cette table met en avant le coût C_i des solutions initiales utilisées par notre algorithme RSDP-Φ, ainsi que le temps T en secondes utilisé pour les obtenir. Ces solutions initiales ont été produites avec l'heuristique MFA [McAllister, 1999] (voir Section 6.5.1). La colonne cinq liste les meilleures solutions connues C^* alors que la colonne six indique la référence bibliographique où ces résultats ont été tirés. Ensuite, les cinq colonnes suivantes présentent le meilleur coût en terme de largeur totale des arêtes C_m, la pire solution C_p, le coût moyen $moy.$, son écart type $e.t.$ et le temps de calcul moyen T en secondes qui ont été obtenus sur dix exécutions de l'algorithme RSDP-Φ. Les quantités sur la colonne onze incluent le temps dépensé par le calcul de la solution initiale. Enfin, la colonne douze montre le gain Δ_C en terme de pourcentage du meilleur coût C_m atteint par RSDP-Φ par rapport à la meilleure solution connue C^* $(100 * (1 - C_m/C^*))$.

Dans la Table 6.7 nous observons que l'algorithme RSDP-Φ est très compétitif en terme de qualité de solution car il obtient un coût moyen $moy.$ inférieur (1.2552E+6) à celui des meilleures solutions connues (1.2573E+6) ; cette différence favorable à RSDP-Φ est plus importante en comparant la moyenne de notre meilleure solution C_m avec celle de la meilleure solution connue (1.2550E+6 contre 1.2573E+6). En effet, RSDP-Φ est capable d'améliorer 10 meilleures solutions connues et d'égaler ces résultats pour 6 autres instances d'essai. Pour certains graphes, l'amélioration atteinte mène à une réduction du coût Δ_C atteignant jusqu'à 0.310%. La détérioration du coût produite par RSDP-Φ pour les 5 instances d'essai restantes peut aller jusqu'à -1.264%. D'ailleurs, les algorithmes OS+RS, ABD+RS, MC, AMG et AMA-Φ arrivent seulement à égaler les meilleures solutions connues respectivement pour 2, 3, 3, 13 et 15 instances de test. Pour les autres graphes, ces cinq algorithmes ont trouvé des étiquetages de moindre qualité que les meilleures solutions connues.

En comparant les temps de calcul corrigés des algorithmes étudiés (Table 6.4 et 6.5)

		MFA				RSDP-Φ					%
Graphe	$\|V\|$	C_i	T	C^*	Réf.	C_m	C_p	$moy.$	$e.t.$	T	Δ_C
randomA1	1000	972583	0.03	867214	5	866968	866987	866975.4	8.2	86.5	0.028
randomA2	1000	6724470	0.11	6528780	1	6522206	6522344	6522221.6	43.0	181.0	0.101
randomA3	1000	14493215	0.22	14238712	5	14194583	14194588	14194585.5	2.6	279.1	0.310
randomA4	1000	1845977	0.04	1718746	5	1717176	1717182	1717179.6	3.1	90.0	0.091
randomG4	1000	280961	0.04	140211	4,5	140596	140598	140597.0	1.1	76.5	-0.275
bintree10	1023	71517	0.01	3696	4	3696	3701	3697.1	1.6	38.8	0.000
hc10	1024	523776	0.03	523776	2,3,4,5	523776	523776	523776.0	0.0	1.2	0.000
mesh33x33	1089	44430	0.01	31729	3,4	31856	32000	31904.2	62.5	89.9	-0.400
3elt	4720	481815	0.07	357329	4,5	359151	359201	359176.0	26.4	1030.8	-0.510
airfoil1	4253	384013	0.06	272931	4	276381	277352	276866.5	511.8	982.1	-1.264
whitaker3	9800	1231912	0.18	1144476	4	1143645	1146016	1145304.7	1145.3	3330.1	0.073
c1y	828	70922	0.01	62262	4,5	62230	62238	62234.4	3.8	32.8	0.051
c2y	980	90347	0.01	78822	4	78757	79264	78810.8	159.3	46.7	0.082
c3y	1327	151622	0.02	123376	5	123145	123158	123151.1	4.8	93.3	0.187
c4y	1366	131106	0.02	115051	5	114936	115238	114971.6	93.8	88.1	0.100
c5y	1202	118541	0.02	96878	5	96850	97054	96877.2	62.4	69.2	0.029
gd95c	62	525	0.01	506	2,4,5	506	507	506.1	0.3	2.1	0.000
gd96a	1096	146407	0.01	95242	5	95263	95310	95277.9	19.6	61.0	-0.022
gd96b	111	1497	0.01	1416	4,5	1416	1422	1417.8	2.9	2.7	0.000
gd96c	65	537	0.01	519	1,2,3,4,5	519	520	519.1	0.3	2.8	0.000
gd96d	180	2937	0.01	2391	4,5	2391	2407	2394.2	6.7	5.7	0.000
Moyenne		1.3223E+6	0.01	1.2573E+6		1.2550E+6	1.2553E+6	1.2552E+6		313.8	0.066

1. OS+RS [Petit, 2003a; Petit, 2003b] 2. ABD+RS [Bar-Yehuda *et al.*, 2001] 3. MC [Koren et Harel, 2002]
4. AMG [Safro *et al.*, 2006] 5. AMA-Φ [Rodriguez-Tello *et al.*, 2006c]

Table 6.7 – Comparaison des performances entre RSDP-Φ et cinq heuristiques de l'état de l'art pour MinLA sur vingt-et-une instances d'essai standard de taille moyenne.

nous observons que le temps d'exécution moyen de RSDP-Φ est très compétitif (313.8 secondes). En effet, il est moins important que les temps consommés par les algorithmes OS+RS, ABD+RS et AMA-Φ qui dépensent respectivement 601.09, 5039.78 et 2251.12 secondes. En comparant RSDP-Φ et les deux autres algorithmes utilisant un RS (OS+RS, ABD+RS) nous pouvons remarquer que notre approche consomme en moyenne moins de temps. Cependant, RSDP-Φ ne peut pas concurrencer avec les heuristiques MC et AMG en terme de temps de calcul (11.19 et 69.49 secondes).

Comparaison de RSDP-Φ sur des instances de très grande taille

Il existent seulement deux algorithmes pour MinLA rapportés dans la littérature qui présentent des expériences sur les instances de très grande taille proposées par Koren et Harel : MC et AMG. Par la suite nous montrons une étude comparative entre les résultats obtenus par ces algorithmes et ceux atteints par RSDP-Φ sur de telles instances d'essai de très grande taille.

Les résultats présentés dans la Section 6.5.3 montrent que RSDP-Φ consomme plus de temps de calcul que les heuristiques MC et AMG. C'est pour cette raison que nous avons décidé de diviser cette étude comparative en deux parties, afin de mener une comparaison juste entre les trois algorithmes. La première en arrêtant l'algorithme RSDP-Φ

immédiatement après qu'il commence à améliorer la meilleure solution connue. La seconde en utilisant la condition d'arrêt présentée dans la Section 6.5.1.

Les données obtenues de ces deux comparaisons sont collectées respectivement dans les Tables 6.8 et 6.9. Toutes les deux présentent : dans la première colonne le nom du graphe étudié et dans la pénultième colonne le gain Δ_C en terme de pourcentage du meilleur coût C_m atteint par RSDP-Φ par rapport à la meilleure solution connue C^* $(100 * (1 - C_m/C^*))$.[6] La dernière colonne indique la différence en pourcentage Δ_T entre le temps de calcul consommé par notre algorithme et celui dépensé par l'heuristique qui a trouvé la meilleure solution connue. Les colonnes intitulées C et T (dans les deux tables) montrent le meilleur coût atteint par chacun des algorithmes (MFA, MC, AMG et RSDP-Φ), ainsi que le temps d'exécution en minutes consommés pour les obtenir. Pour les algorithmes MC et AMG nous avons corrigé le temps (colonne T_c) d'exécution à l'aide du facteur d'équivalence (1.22) montré dans la Table 6.3 car les deux algorithmes ont été exécutées sur un processeur Pentium IV cadencé à 1.7 GHz. Enfin, et à titre d'illustration, la pire solution C_p, le coût moyen $moy.$ obtenu sur dix exécutions de l'algorithme RSDP-Φ et son écart type $e.t.$ sont listés dans les colonnes cinq à sept de la Table 6.9. Tous les résultats présentés pour les heuristiques de MC et AMG sont tirés des articles correspondants. Les données sur les colonnes dix à treize de la Table 6.8 ont été omises, pour les instances de test *m14b* et *auto* parce que RSDP-Φ n'a pas amélioré la meilleure solution connue pour ces deux graphes.

			MC			AMG			RSDP-Φ		%	%
Graphe	$\|V\|$	$\|E\|$	C	T	T_c	C^*	T	T_c	C_m	T	Δ_C	Δ_T
tooth	78136	452591	255465042	10.5	8.6	227639682	27.0	22.1	227634482	52.7	2.28E-3	-1.38E+2
ocean	143437	409593	141732687	13.5	11.1	118882522	72.0	59.0	118880582	147.8	1.63E-3	-1.50E+2
mrngA	257000	505048	348448986	23.5	19.3	305560971	90.0	73.8	305503135	290.8	1.89E-2	-2.94E+2
rotor	99617	662431	247583742	16.5	13.5	221832991	42.0	34.4	221832985	573.1	2.70E-6	-1.56E+3
598	110971	741934	340886008	19.0	15.6	281033967	57.0	46.7	281032048	392.3	6.83E-4	-7.40E+2
144	144649	1074393	772846779	28.5	23.4	745701842	84.0	68.9	745682093	407.1	2.65E-3	-4.91E+2
m14b	214765	1679018	1004606217	40.0	32.8	857743008	130.0	106.6	-	-	-	-
mrngB	1017253	2015714	3558254373	98.0	80.3	3254023540	520.0	426.2	3253970040	1529.6	1.64E-3	-2.59E+2
auto	448695	3314611	4501150138	100.0	82.0	3871472605	340.0	278.7	-	-	-	-
Moyenne			1.2412E+9	38.83	31.83	1.0982E+9	151.33	124.04			3.975E-3	-5.196E+2

Table 6.8 – Comparaison des performances entre RSDP-Φ, MC et AMG sur 9 instances de grande taille. RSDP-Φ s'arrête dès qu'il dépasse la meilleure solution connue.

L'analyse des résultats présentés dans les Tables 6.8 et 6.9 nous amène aux observations suivantes. Tout d'abord, la qualité de solution atteinte par l'algorithme RSDP-Φ, qui s'arrête dès qu'il améliore la meilleure solution connue, est très compétitive par rapport à celle produite par les algorithmes MC et AMG. En fait, RSDP-Φ améliore la meilleure solution connue pour 7 des 9 instances de test, en réalisant une amélioration moyenne de 3.975E-3% (voir colonne Δ_C dans la Table 6.8). Pour certaines instances d'essai cette réduction atteint jusqu'à 1.89E-2%. Néanmoins, RSDP-Φ trouve des solutions de moindre qualité que l'heuristique AMG pour deux des instances étudiées (*m14b* et *auto*). La raison de ces résultats est, à notre avis, la densité très élevée de ces deux graphes.

[6] Pour les neuf instances les meilleures solutions connues ont été trouvées par l'algorithme AMG.

	MFA		RSDP-Φ					%	%
Graphe	C	T	C_m	C_p	$moy.$	$e.t.$	T	Δ_C	Δ_T
tooth	253104451	0.09	214678297	215775888	214884320.5	326043.8	361.6	5.69E+0	-1.53E+3
ocean	210694459	0.10	114722301	115339139	114908560.6	219666.8	553.5	3.50E+0	-8.38E+2
mrngA	697873953	0.19	294042454	296004334	294819720.3	798840.4	639.4	3.77E+0	-7.67E+2
rotor	286752751	0.14	221696343	223282736	222045040.2	496501.4	952.4	6.16E-2	-2.67E+3
598	351481792	0.20	280075154	281907655	280567592.1	721882.8	743.6	3.41E-1	-1.49E+3
144	915001621	0.26	736442970	737645850	736844481.8	449505.6	770.3	1.24E+0	-1.02E+3
m14b	1013429382	0.39	859059510	861192875	859676014.1	854704.9	4183.3	-1.53E-1	-3.83E+3
mrngB	8103013010	0.82	3178145258	3180193230	3178689877.2	806108.0	3254.4	2.33E+0	-6.64E+2
auto	5750501188	0.95	4049967166	4052270846	4050372915.5	775540.9	4842.1	-4.61E+0	-1.64E+3
Moyenne	1.9535E+9	0.35	1.1054E+9	1.1071E+9	1.1059E+9		1811.17	2.42E+0	-1.605E+3

Table 6.9 – Comparaison des performances entre RSDP-Φ, MC et AMG sur 9 instances de grande taille. RSDP-Φ s'arrête quand la valeur moyenne du coût cesse de décrémenter.

Le temps de calcul moyen utilisé par notre algorithme pour produire ces résultats est supérieur à celui consommé par les heuristiques MC et AMG (en moyenne 5.196E+2% plus grand). Cependant, comme RSDP-Φ surpasse les deux autres méthodes comparées en terme de coût, nous pensons que le temps de calcul supplémentaire consommé est entièrement justifié. En outre, on peut observer dans la deuxième partie de la comparaison (Table 6.9) que RSDP-Φ continue à améliorer la qualité de solution quand la condition d'arrêt décrite dans la Section 6.5.1 est employée (c'est-à-dire quand il est exécuté pendant un peu plus de temps). Ceci mène à accomplir des réductions plus importantes du coût comme dans le cas du graphe *mrngB* où la réduction obtenue est $\Delta_C = 2.33\text{E+}0\%$. En revanche, il a été reconnu par les auteurs respectifs que les heuristiques MC et AMG ne tirent pas profit des exécutions plus longues [Koren et Harel, 2002; Safro *et al.*, 2006].

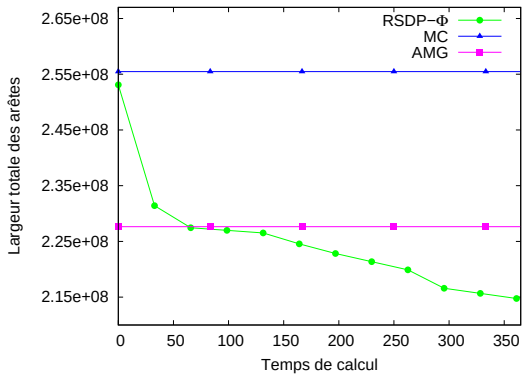


Figure 6.8 – Processus de convergence de l'algorithme RSDP-Φ sur l'instance de test $tooth$. La condition d'arrêt utilisée est celle décrite dans la Section 6.5.1.

Le comportement favorable de l'algorithme RSDP-Φ est illustré sur la Figure 6.8, où le processus de convergence sur l'instance $tooth$ est présenté. Le graphique représente la qualité de solution moyenne obtenue sur dix exécutions (l'axe des ordonnées) par rapport au temps de calcul consommé (l'axe des abscisses). Les meilleures solutions trouvées

par MC et AMG sont également incluses à titre comparatif. Sur cette figure, nous pouvons observer que RSDP-Φ continue à réduire le coût presque sans interruption, après avoir surpassé les deux autres heuristiques comparées. Par exemple, il peut atteindre un coût de 223988122.2 en dépensant 5.34 fois plus de temps de calcul que l'heuristique AMG (c'est-à-dire 171.10 minutes), ce qui représente une amélioration de $\Delta_C = 1.604\text{E+0}$. L'amélioration maximale en coût (214884320.5 contre 227639682 pour AMG) est obtenue en utilisant 361.56 minutes.

6.5.4 Discussion

L'objectif principal des expérimentations présentées dans cette section est de mieux comprendre les influences de certaines caractéristiques importantes pour tout algorithme basé sur le RS. Dans certaines de ces expériences nous avons employé une version légèrement modifiée de l'algorithme RSDP-Φ présenté dans la Section 6.5.1. Cet algorithme modifié, que nous appelons RSUP (Recuit Simulé en Une Phase), nous a permis d'apprécier plus clairement les influences étudiées. Les principales modifications sont que RSUP commence la recherche à partir d'une solution initiale aléatoire et avec une température initiale garantissant 70% de mouvements admis dans le premier palier de température.

Les trente instances appartenant aux deux séries de tests décrites dans la Section 6.1 ont été employées de manière uniforme lors des expérimentations présentes dans les sections suivantes. Des résultats similaires ont été obtenus pour les trente instances, cependant, dans le but de synthétiser ces informations, nous avons décidé de les présenter en utilisant seulement quelques graphes représentatifs. Les résultats présentés dans la suite de cette section correspondent à la moyenne sur dix exécutions indépendantes.

Interaction entre la relation de voisinage et la fonction d'évaluation

Dans cette section, nous nous intéressons à l'étude de l'interaction entre le voisinage et la fonction d'évaluation afin mieux comprendre l'influence de ces deux éléments sur la performance de notre algorithme. Pour cette étude nous considérons les fonctions d'évaluation LA et Φ, les fonctions de voisinage $\mathcal{N}_1(\varphi)$ et $\mathcal{N}_3(\varphi)$ mentionnées dans la Section 6.5.1, mais également deux autres relations de voisinage exprimées dans les Équations 6.13 et 6.14 :

$$\mathcal{N}_5(\varphi) = \left\{\varphi' \in \mathscr{L} : swap(\varphi, u, v) = \varphi',\ u, v \in V,\ v \in A(u)\right\} \tag{6.13}$$

$$\mathcal{N}_6(\varphi) = \left\{\varphi' \in \mathscr{L} : swap(\varphi, u, v, w) = \varphi',\ (u, v, w) \in V,\ u \neq v \neq w\right\} \tag{6.14}$$

où $A(u)$ représente l'ensemble de sommets adjacents à u, et $swap(\varphi, u, v, w)$ est le résultat de l'application successive des opérations $swap(\varphi, u, v)$ et $swap(\varphi, v, w)$.

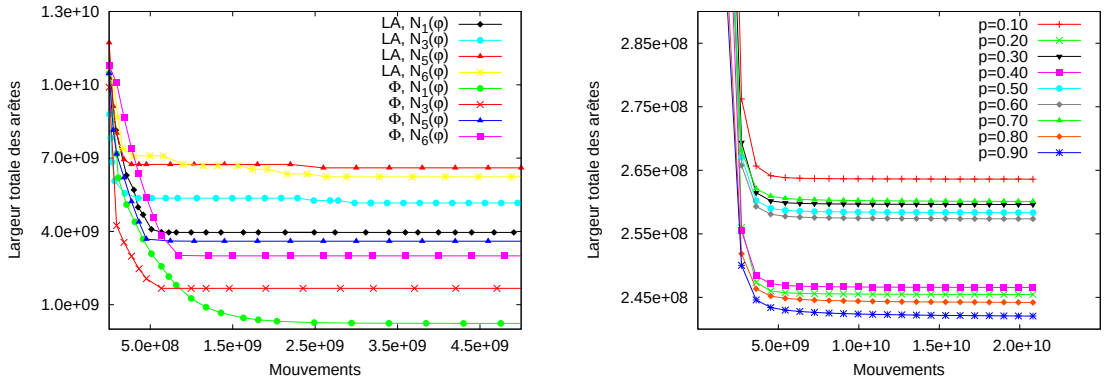
De nombreuses expérimentations ont été effectuées en utilisant l'algorithme RSUP décrit ci-dessus, afin de comparer les performances de chacune des huit combinaisons formées entre les deux fonctions d'évaluation et les quatre voisinages analysés. Le graphique de la Figure 6.9(a) montre le processus de convergence, en terme de qualité

moyenne de solution atteinte par RSUP, quand chacune des huit combinaisons étudiées est utilisée pour résoudre l'instance de test *tooth*.

Les observations suivantes ont été faites à partir de ce graphique. La fonction d'évaluation Φ permet dans tous les cas d'améliorer les résultats fournis par l'algorithme RSUP, ce qui met en évidence que Φ est plus efficace que LA.

En ce qui concerne les voisinages étudiés, nous remarquons sur ce graphique que la meilleure performance est atteinte par RSUP lorsque la fonction de voisinage $\mathcal{N}_1(\varphi)$ est utilisée. Les voisinages $\mathcal{N}_5(\varphi)$ et $\mathcal{N}_6(\varphi)$ produisent tous les deux des solutions de moindre qualité que $\mathcal{N}_1(\varphi)$ et $\mathcal{N}_3(\varphi)$.

Enfin, si l'on considère les huit combinaisons analysées entre les fonctions d'évaluation et les voisinages, la meilleure d'entre elles est lorsque Φ et $\mathcal{N}_1(\varphi)$ sont utilisés simultanément (atteignant un coût moyen de $\text{LA} = 235301805.89$). Les sept autres combinaisons restantes fournissent des solutions de moindre qualité. Il est important d'observer que la combinaison $(\Phi, \mathcal{N}_3(\varphi))$ permet de réduire la largeur totale des arêtes plus rapidement que la paire formée par $(\Phi, \mathcal{N}_1(\varphi))$. Cependant, l'algorithme RSUP peut rester longtemps bloqué dans un minimum local à cause de la faible capacité d'exploration de $\mathcal{N}_3(\varphi)$.



(a) Comparaison entre quatre voisinages et deux fonctions d'évaluation.

(b) Influence de la valeur de p sur $\mathcal{N}_4(\varphi, x)$.

Figure 6.9 – Interaction entre le voisinage et la fonction d'évaluation sur la performance de l'algorithme RSDP.

En partant de ces observations, nous avons décidé de combiner $\mathcal{N}_3(\varphi)$ avec la fonction de voisinage $\mathcal{N}_1(\varphi)$ afin d'obtenir un meilleur compromis entre l'exploration et l'exploitation. Ce voisinage combiné, appelé $\mathcal{N}_4(\varphi)$, a été présenté dans la Formule 6.6 et nous l'utilisons couplé à la fonction d'évaluation Φ. La fonction de voisinage $\mathcal{N}_4(\varphi)$ utilise un paramètre x qui représente la probabilité d'appliquer le voisinage $\mathcal{N}_3(\varphi)$ (le reste du temps, $\mathcal{N}_1(\varphi)$ est employé).

Afin de trouver la valeur la plus appropriée pour la probabilité p utilisée dans $\mathcal{N}_4(\varphi)$, nous avons procédé comme suit : pour chacune des 9 valeurs de p, dans l'intervalle $[0.10, 0.90]$ avec un incrément de 0.10, 10 exécutions de l'algorithme RSUP (utilisant la fonction Φ) sur le graphe *tooth* ont été effectuées (des résultats semblables ont été obtenus avec d'autres instances). Les résultats moyens de ces exécutions sont illustrés sur la

figure 6.9(b). Il est facilement observable sur ce graphique que la meilleure qualité de solution moyenne est obtenue par RSUP lorsqu'une probabilité $p = 0.90$ est utilisée par la fonction de voisinage $\mathcal{N}_4(\varphi, x)$ (LA $= 225943530.37$). Cette probabilité permet à notre fonction de voisinage combinée $\mathcal{N}_4(\varphi, x)$ d'obtenir de bien meilleurs résultats que $\mathcal{N}_3(\varphi)$. C'est pour cette raison que nous avons décidé d'utiliser le voisinage combiné $\mathcal{N}_4(\varphi, x)$ couplé à la fonction d'évaluation Φ dans notre implémentation du RSDP.

Influence du schéma de refroidissement

Cette expérimentation a pour objectif d'analyser l'influence du schéma de refroidissement sur la performance de notre algorithme RSDP-Φ. Pour cette étude nous avons choisi deux schémas de refroidissement représentatifs de la littérature : géométrique [Kirkpatrick *et al.*, 1983] et statistique [Aarts et Laarhoven, 1985]. Ils ont été implémentés dans l'algorithme RSUP décrit au début de cette section. Nous les avons appelés RSUP-*Géométrique* et RSUP-*Statistique* afin de les distinguer.

Les deux algorithmes ainsi implémentés utilisent la méthode décrite dans [Aarts *et al.*, 1985] pour calculer leur température initiale. Cette méthode permet d'assurer un taux initial d'acceptation choisi. Les deux algorithmes commencent la recherche à partir d'une solution initiale aléatoirement générée et ils produisent r mouvements à chaque palier de température. Dans l'algorithme RSUP-*Géométrique*, la mise à jour de la température courante T_k est effectuée en utilisant la relation suivante : $\mathcal{Q}(T_k) = 0.96(T_k)$. En revanche, RSUP-*Statistique* décrémente la température en appliquant l'Équation 6.11. Ce processus continue pour les deux algorithmes jusqu'à ce que la condition d'arrêt détaillée en Section 6.5.1 soit vérifiée.

Les résultats produits par cette comparaison expérimentale mettent en avant l'avantage d'employer un schéma de refroidissement statistique. Cette tendance est visible sur la Figure 6.10 qui montre le comportement typique de RSUP-*Statistique* par rapport à celui de RSUP-*Géométrique*. Dans ce graphique, l'axe des abscisses représente le nombre de mouvements alors que l'axe des ordonnées correspond à la qualité de la solution obtenue. Ce graphique inclut également une troisième courbe représentant l'évolution de l'algorithme RSDP-Φ afin d'établir des comparaisons. Rappelons que RSDP-Φ emploie également le schéma de refroidissement statistique, mais qu'il commence à partir d'une solution de bonne qualité et à une température appropriée déterminée par la méthode proposée dans [Varanelli et Cohoon, 1999].

Nous observons sur cette figure que l'algorithme RSUP employant le schéma de refroidissement statistique produit de meilleurs résultats que ceux obtenus par l'algorithme RSUP-*Géométrique* et en effectuant moins de mouvements. En fait, ceci est possible parce que RSUP-*Statistique* ajuste la température courante en employant certaines informations extraites des solutions potentielles visitées. C'est grâce à ces informations que RSUP-*Statistique* explore l'espace de recherche de manière plus efficace que l'algorithme RSUP-*Géométrique*. La troisième courbe de la Figure 6.10 nous permet d'observer l'important gain de temps produit par l'algorithme RSDP-Φ par rapport aux deux autres algorithmes comparés. Les expériences effectuées, sur les deux séries de tests, montrent que RSDP-Φ converge en moyenne 24.9% plus rapidement que l'algorithme RSUP-*Statistique* car il né-

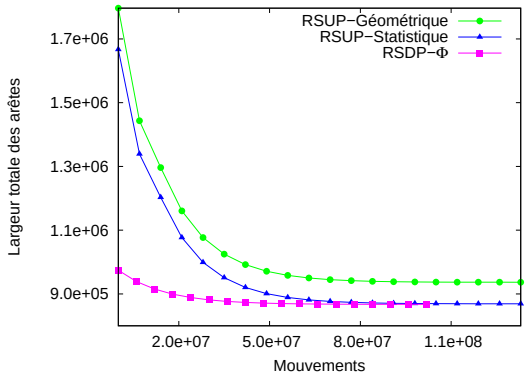


Figure 6.10 – Influence du schéma de refroidissement sur l'algorithme RSDP-Φ.

cessite un nombre inférieur de mouvements, et par conséquent moins de temps de calcul pour retourner une solution de bonne qualité.

6.6 Synthèse du chapitre

Nous avons présenté dans ce chapitre une série de comparaisons expérimentales entre la fonction d'évaluation classique LA du problème MinLA, et la nouvelle fonction d'évaluation Φ (introduite en Section 5.4). Ces comparaisons ont été effectuées à l'aide de deux algorithmes de recherche différents, la *Descente Stricte* et un *Algorithme Mémétique*. Les résultats obtenus de ces expérimentations montrent que Φ est une meilleure fonction d'évaluation que la fonction classique LA car elle permet d'améliorer de manière très importante les performances des deux algorithmes en terme de qualité de solution.

Nous avons présenté dans un second temps, un *Algorithme Mémétique Amélioré* pour le problème MinLA. Cet algorithme, appelé AMA-Φ, a le mérite de perfectionner quatre caractéristiques clés qui impactent de manière très importante sa capacité de recherche. Il tire avantage non seulement de la nouvelle fonction d'évaluation Φ, mais aussi d'une procédure efficace d'initialisation de la population, d'un opérateur de croisement inspiré de la méthode de *Path Relinking* et d'un opérateur de RL avancé. Les influences de ces caractéristiques clés sur la performance globale de l'algorithme AMA-Φ ont été analysées à travers de nombreuses expérimentations. Les comparaisons entre AMA-Φ et les heuristiques de l'état de l'art, effectuées sur 21 instances de test issues de la littérature, mettent en évidence que AMA-Φ produit un gain important en efficacité par rapport aux heuristiques existantes. En effet, il améliore pour 7 instances les meilleures solutions connues et parvient à les égaler pour 8 autres instances. Néanmoins, AMA-Φ présente certaines limites, notamment par rapport à la taille des instances qu'il peut traiter et au temps de calcul qu'il consomme.

Compte tenu de ces limitations, nous avons orienté nos efforts de recherche afin de concevoir un algorithme de *Recuit Simulé en Deux Phases*. Cet algorithme, que nous appelons RSDP-Φ, utilise la fonction d'évaluation Φ, un schéma de refroidissement statistique

et une fonction de voisinage combinée.

Dans la première phase de l'algorithme RSDP-Φ une heuristique rapide est employée pour remplacer les actions effectuées par un RS traditionnel à températures élevées. Dans la seconde phase, la solution heuristique obtenue est améliorée avec un RS conventionnel qui est initialisé à une température plus basse que la normale.

Afin d'évaluer l'efficacité pratique de cet algorithme, nous avons effectué des expérimentations approfondies sur un large panel d'instances d'essai issues de la littérature. Dans ces expériences, l'algorithme RSDP-Φ a été comparé à cinq autres heuristiques de l'état de l'art. Les résultats obtenus mettent en évidence que RSDP-Φ est plus efficace que les heuristiques existantes, notamment sur les instances de très grande taille. En effet, il permet d'améliorer 17 des 30 meilleures solutions connues en terme de qualité de solution en utilisant des temps de calcul très raisonnables.

Conclusion générale

Au début de ce manuscrit, nous avons avancé l'hypothèse que *la performance des algorithmes existants pour les problèmes BMP et MinLA, et plus généralement des algorithmes heuristiques, peut être améliorée en introduisant des nouvelles fonctions d'évaluation plus pertinentes*. Les différents résultats présentés dans les chapitres précédents nous ont permis de confirmer cette hypothèse en illustrant le fait qu'une fonction d'évaluation soigneusement conçue peut améliorer considérablement l'efficacité de recherche de ce type d'algorithmes.

Dans ce dernier chapitre, nous présentons une synthèse des différents apports de cette thèse ainsi qu'une discussion ouvrant sur quelques perspectives de recherche.

Principales contributions

Les différents travaux réalisés au cours de cette thèse ont donné lieu à plusieurs contributions de nature différente.

Notre *premier apport* est une étude systématique de plusieurs techniques proposées dans la littérature, visant à améliorer l'efficacité de guidage des fonctions d'évaluation (Chapitre 2). Il s'agit, à notre connaissance, de la première revue détaillée consacrée à ce sujet. Nous avons accompagné cette étude de synthèse par une classification des fonctions d'évaluation en cinq grands types : avec des pénalités, dynamiques, par apprentissage, hiérarchiques et spécifiques au problème traité.

La *seconde contribution* de cette thèse est la création de deux nouvelles fonctions d'évaluation pour les problèmes d'étiquetage de graphes NP-difficiles BMP (Section 3.4) et MinLA (Section 5.4). Ces deux nouvelles fonctions appelées respectivement δ et Φ ont été soigneusement conçues afin d'évaluer la qualité des solutions potentielles en considérant non seulement l'information fournie par la fonction objectif, mais également certaines connaissances additionnelles du problème traité.

Notre fonction d'évaluation δ pour BMP, contrairement à la fonction classique β, tient compte des informations induites par toutes les arêtes du graphe. Ces informations sont ensuite utilisées afin de favoriser les solutions contenant aussi peu de grandes différences absolues que possible. Grâce à ces informations supplémentaires, δ est capable de distinguer des solutions ayant la même largeur de bande β.

La nouvelle fonction d'évaluation Φ pour MinLA évalue la qualité d'une solution en

considérant trois caractéristiques : la largeur totale des arêtes LA, chaque différence absolue k du graphe, ainsi que leurs fréquences d'apparition. En utilisant ces informations, Φ favorise les solutions contenant des différences absolues de grande valeur car intuitivement ce type d'étiquetages ont une probabilité plus forte d'être amélioré ultérieurement. En effet, il est plus facile de réduire le coût d'une solution en modifiant les étiquettes dans les extrémités d'une seule arête qui pourrait produire une diminution importante de la largeur totale des arêtes. En outre, Φ permet de distinguer des solutions qui pour LA ont le même coût.

Les études expérimentales réalisées nous ont permis de conclure que les deux nouvelles fonctions proposées sont plus efficaces que les fonctions d'évaluation classiques car elles augmentent considérablement l'efficacité de recherche des métaheuristiques étudiées.

Le *troisième apport* de cette thèse correspond aux différents algorithmes que nous avons développés dans le cadre des études menées. C'est le cas notamment de l'algorithme RSA-δ, un *Recuit Simulé Amélioré* que nous avons mis en œuvre afin de résoudre efficacement le problème BMP (Section 4.4). Notre algorithme RSA-δ perfectionne trois caractéristiques clés qui impactent de manière très importante sa capacité de recherche. Il tire avantage non seulement de la nouvelle fonction d'évaluation δ, mais aussi d'une représentation interne des solutions originale et d'une fonction de voisinage basée sur des mouvements de rotation. Les influences de ces caractéristiques clés ainsi que du paramétrage ont été analysées à travers de nombreuses expérimentations, ce qui nous a permis de mieux les comprendre et de les utiliser ensuite plus efficacement. Les comparaisons expérimentales entre RSA-δ et les heuristiques de l'état de l'art, effectuées sur 125 instances de test issues de la littérature, mettent en évidence que RSA-δ est très compétitif. Il permet l'amélioration des anciens meilleurs résultats pour 57 des 113 instances d'essai, surpassant ainsi les meilleures heuristiques connues.

Dans un second temps, nous avons développé l'algorithme AMA-Φ pour le problème MinLA. Il s'agit d'un *Algorithme Mémétique Amélioré* capable de rivaliser avec les meilleures heuristiques rapportées dans la littérature pour ce problème (Section 6.4). Notre algorithme AMA-Φ est construit autour de la nouvelle fonction d'évaluation Φ et de l'amélioration de trois autres composants importants. Le premier est une procédure efficace d'initialisation de la population basée sur l'heuristique de minimisation frontale améliorée (MFA) de McAllister [1999]. Cette procédure permet de construire une population initiale avec des individus de très bonne qualité, tout en utilisant un temps d'exécution raisonnable. Le deuxième composant est un nouvel opérateur de croisement appelé *TX* qui est inspiré de la méthode de *Path Relinking*. C'est à notre connaissance le premier opérateur de croisement conçu pour tenir compte des spécificités du problème MinLA. Le troisième et dernier composant perfectionné par AMA-Φ est un opérateur de Recherche Locale. Cet opérateur est fondé sur un algorithme de RS qui a été soigneusement paramétré pour être performant, robuste et avec des temps de calcul très courts.

Les influences de ces trois composants clés pour la performance globale de l'algorithme AMA-Φ ont été analysées en réalisant de nombreuses expérimentations. Cette analyse nous a permis de trouver la meilleur manière de les utiliser afin de rendre notre

algorithme plus efficace. Les comparaisons entre AMA-Φ et les heuristiques de référence, effectuées sur 21 instances de test issues de la littérature, montrent que AMA-Φ est capable d'améliorer les meilleures solutions connues pour 7 instances et de les égaler pour 8 autres. Néanmoins, notre algorithme mémétique présente certaines limites comme la taille des instances qui peuvent être traitées ou le temps de calcul consommé qui peut s'avérer assez long.

Compte tenu de ces limitations, nous avons dirigé nos efforts de recherche vers le développement d'un algorithme de *Recuit Simulé en Deux Phases* pour le problème MinLA. Cet algorithme que nous appelons RSDP-Φ est le résultat de nombreuses études sur les schémas de refroidissement et les fonctions de voisinage effectuées afin de trouver les plus adaptés. Il intègre également la nouvelle fonction d'évaluation Φ (Section 6.5).

Notre algorithme RSDP-Φ se compose de deux phases. Dans la première, l'heuristique MFA de McAllister [1999] est employée pour remplacer les actions effectuées par un RS traditionnel à températures élevées. Dans la seconde phase, la solution heuristique obtenue est améliorée avec un algorithme de RS qui est initialisé à une température plus basse que la normale déterminée avec la méthode proposée par Varanelli et Cohoon [1999].

Afin d'évaluer l'efficacité pratique de l'algorithme RSDP-Φ, nous avons effectué des expérimentations approfondies sur un ensemble de 30 instances d'essai standard de la littérature. Dans ces expériences, l'algorithme RSDP-Φ a été soigneusement comparé à cinq autres heuristiques de l'état de l'art. Les résultats obtenus témoignent de la supériorité de RSDP-Φ par rapport aux heuristiques existantes, notamment sur les instances de très grande taille. En effet, RSDP-Φ permet d'améliorer 17 des 30 meilleures solutions connues, en terme de qualité de solution, en utilisant des temps de calcul très compétitifs.

Perspectives de recherche

Bien sûr, les travaux réalisés au cours de cette thèse comportent quelques limites. C'est le cas notamment de l'analyse effectuée sur les caractéristiques des fonctions d'évaluation classiques (Chapitres 3 et 5). Bien qu'elle nous a permis d'obtenir des informations qui nous ont guidé vers la conception de deux nouvelles fonctions d'évaluation plus efficaces, nous sommes conscients que cette analyse pourrait être approfondie afin de considérer d'autres propriétés des fonctions d'évaluation liées à la topologie de l'espace de recherche. Par exemple, il serait très intéressant d'analyser la corrélation entre les classes d'équivalence produites par une fonction d'évaluation et les réseaux de neutralité de l'espace de recherche [Kauffman et Levin, 1987; Reidys *et al.*, 1997].

De même, différentes améliorations peuvent encore être apportées aux divers algorithmes que nous avons développés afin de dépasser leurs limites. Par exemple, dans le cas de notre algorithme RSA-δ (*Recuit Simulé Amélioré*) pour le problème BMP, nous pensons que d'autres schémas de refroidissement peuvent être envisagés afin de rendre l'algorithme plus performant en terme de qualité de solution, tout en réduisant la durée de calcul.

Un autre exemple est l'algorithme AMA-Φ (*Algorithme Mémétique Amélioré*) pour le

problème MinLA dont la principale limite est la taille des instances qui peuvent être traitées à cause du temps de calcul consommé. Nous pensons qu'une solution envisageable serait de paralléliser notre algorithme afin de pouvoir résoudre efficacement des instance de plus grande taille. De plus, cette ligne de recherche future serait très intéressante car à notre connaissance il n'existe encore aucun algorithme mémétique de ce type rapporté dans la littérature pour le problème MinLA.

L'algorithme RSDP-Φ (*Recuit Simulé en Deux Phases*) pour MinLA nous a permis d'obtenir des résultats de très bonne qualité. Cependant, il existe des instances pour lesquelles RSDP-Φ trouve des solutions de moindre qualité que les meilleures solutions connues. Nous avons remarqué que cette limite est probablement liée à la densité très élevée de ces graphes. Il s'avère donc nécessaire d'étudier plus en détail ce comportement afin d'améliorer la performance globale de notre algorithme RSDP-Φ.

En ce qui concerne les perspectives de recherche, l'état actuel de nos travaux laisse entrevoir de nombreuses possibilités. La première consiste à effectuer une étude détaillée qui porte sur l'espace de recherche des problèmes BMP et MinLA afin de considérer des informations comme : le nombre et la distribution des plateaux [Schuster, 1997; Verel *et al.*, 2006], le nombre de minima locaux [Garnier et Kallel, 2002] et la taille des bassins d'attraction [Garnier et Kallel, 2000]. Ce type d'études ont été déjà réalisées pour d'autres problèmes d'optimisation comme celui du Voyageur de Commerce [Kirkpatrick et Toulouse, 1985; Stadler, 1992; Fonlupt *et al.*, 1998], de Coloration de Graphes [Hertz *et al.*, 1994] et de Satisfiabilité [Frank *et al.*, 1997]. Elles ont permis de mieux expliquer le comportement des algorithmes de rechercher et ainsi proposer des nouvelles métaheuristiques adaptées à ce type de problèmes.

La deuxième direction pour de futurs travaux de recherche porte sur la création des nouvelles fonctions d'évaluation pour d'autre problèmes combinatoires. Nous pensons à plusieurs problèmes dont les algorithmes approchés pour les résoudre utilisent la fonction objectif comme fonction d'évaluation, perdant ainsi la possibilité d'atteindre une meilleure performance. C'est le cas par exemple du problème des Arrangements de Couverture (Covering Arrays) ou des arrangements orthogonaux [Cohen, 2004; Nurmela, 2004], où la fonction d'évaluation prend en compte seulement l'objectif à minimiser.

Enfin, nous envisageons aussi d'orienter nos efforts de recherche vers une approche complètement différente à celle présentée dans cette thèse pour produire de nouvelles fonctions d'évaluation plus efficaces. Cette approche consisterait à utiliser la programmation génétique proposée par Koza [1992] pour faire évoluer une population constituée de fonctions d'évaluation créées aléatoirement à partir d'un ensemble d'opérateurs et de constantes.

Annexe A

Rappels de théorie des graphes

Nous présentons dans cette annexe une série de définitions et notations générales concernant la théorie des graphes que nous utilisons dans ce travail de thèse.

L'histoire de la *Théorie des Graphes* (TG) débute probablement au XVIII^{e} siècle, quand Leonhard Euler démontra qu'il était impossible de traverser chacun des sept ponts de la ville russe de Königsberg (aujourd'hui Kaliningrad) une fois exactement et de revenir au point de départ. Les ponts enjambent les bras de la Pregel qui coulent de part et d'autre de l'île de Kneiphof [Euler, 1741]. Ce problème est représenté dans la Figure A.1 où les rives son étiquetées avec des lettres.

(a) **(b)**

Figure A.1 – Les sept ponts de Königsberg.

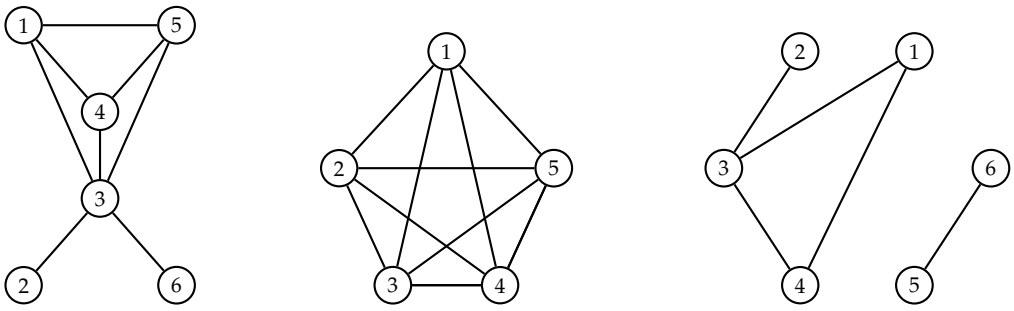
La TG constitue un domaine important des mathématiques qui, historiquement, s'est aussi développé au sein d'autres disciplines telles que la biologie (génome), la chimie (modélisation de structures), les sciences sociales (modélisation des relations) ou en vue d'applications industrielles (conception des circuits VLSI). Elle constitue l'un des instruments les plus courants et les plus efficaces pour résoudre des problèmes discrets posés en recherche opérationnelle.

De manière générale, un *graphe* permet de modéliser de nombreuses situations concrètes où interviennent des objets en interaction. Par exemple, les composants d'un circuit électronique, les interconnexions routières, ferroviaires ou aériennes entre différentes agglomérations, les arbres généalogiques, les diagrammes de succession de tâches pour la gestion des projets, ...

Les graphes permettent de manipuler plus facilement des objets et leurs relations grâce à sa représentation intuitive. L'ensemble des techniques et outils mathématiques mis au point en TG permettent de démontrer efficacement des propriétés et d'en déduire des méthodes de résolution. En plus d'être un concept purement mathématique, le graphe est aussi une structure de données très utilisée en informatique.

Nous continuons cette annexe en présentant les définitions et notations fondamentales des graphes que nous utilisons dans les deux cas d'étude exposés dans cette thèse.

Définition 1 (Graphe non orienté) *Un graphe non orienté $G = (V, E)$ est une structure composée d'un ensemble fini V d'éléments appelés* sommets*, et d'un ensemble de paires non ordonnées de sommets $E \subseteq V \times V = \{\{u, v\} : u, v \in V\}$ nommées* arêtes*.*



(a) Graphe simple **(b)** Graphe complet **(c)** Graphe non connexe

Figure A.2 – Exemples des graphes.

Définition 2 (Ordre d'un graphe) *On appelle ordre d'un graphe $G = (V, E)$ le nombre de sommets $|V|$ de ce graphe.*

Définition 3 (Sommets adjacents) *Deux sommets u et v sont dits adjacents ou voisins lorsqu'ils sont reliés par une arête $e = \{u, v\} \in E$. u et v sont appelés les* extrémités *de e.*

Définition 4 (Arête incidente) *Une arête est incidente à un sommet u si u est l'une de ses extrémités.*

Définition 5 (Arête multiple) *Une arête est dite multiple s'il existe au moins une autre arête avec les mêmes sommets, sa* multiplicité *étant le nombre total d'arêtes ayant ces sommets.*

Définition 6 (Boucle) *Une arête e est appelée boucle si ses deux sommets sont identiques ($e = \{v, v\} \in E$), c'est-à-dire une arête joignant un sommet à lui-même. On l'appelle aussi* arête réflexive*.*

Définition 7 (Graphe simple) *Un graphe simple $G = (V, E)$ est un graphe sans boucle ni arête multiple (voir Figure 6.2(a)).*

Définition 8 (Degré d'un sommet) *Soit $G = (V, E)$ un graphe non orienté simple, et v un sommet de ce graphe. Le degré de v, noté $d(v)$, est le nombre d'arêtes de E incidentes à v, c'est-à-dire contenant v. Lorsque $d(v) = 0$, on dit que le sommet v est* isolé.

Définition 9 (Degré d'un graphe) *Le degré d'un graphe $G = (V, E)$ est le degré maximum de tous les sommets de ce graphe $\Delta(G) = \max\{d(v) : v \in E\}$.*

Définition 10 (Graphe vide) *Un graphe vide $(V, \emptyset)$ est un graphe comportant n sommets isolés, c'est-à-dire un graphe sans arête.*

Définition 11 (Graphe nul) *Un graphe nul $(\emptyset, \emptyset)$ ou simplement $\emptyset$ est un graphe d'ordre zéro.*

Définition 12 (Graphe complet) *Un graphe $G = (V, E)$ est dit complet si, pour toute paire de sommets $\{u, v\}$, il existe au moins une arête $e = \{u, v\}$ (voir Figure 6.2(b)).*

Définition 13 (Chaîne) *Une chaîne entre deux sommets a et b d'un graphe $G = (V, E)$ est une suite $\{\{v_1, v_2\}, \{v_2, v_3\}, ..., \{v_{k-1}, v_k\}\}$ d'arêtes de E telle que $a = v_1$ et $b = v_k$. Le nombre d'arêtes d'une chaîne est appelé* longueur *de la chaîne.*

Définition 14 (Distance entre sommets) *La distance entre deux sommets u et v d'un graphe $G = (V, E)$ est la longueur de la plus courte chaîne les reliant.*

Définition 15 (Graphe connexe) *Un graphe $G = (V, E)$ est dit connexe s'il existe une chaîne entre toute paire de sommets du graphe (Figure 6.2(a)). Un graphe* non connexe *se divise en composantes connexes (Figure 6.2(c)).*

Définition 16 (Diamètre d'un graphe) *On appelle diamètre d'un graphe $G = (V, E)$, noté $diam(G)$, la plus longue des distances entre deux sommets.*

Nous invitons le lecteur intéressé par plus de détails sur la TG à consulter les références suivantes [Gross et Yellen, 2005; Diestel, 2005].

Rappels de théorie des ensembles

Un concept fondamental dans la *théorie des ensembles* est celui d'une *relation binaire*. Une relation binaire est une association arbitraire des éléments d'un ensemble avec les éléments d'un autre ensemble (éventuellement le même).

Un exemple est la relation « divise » entre l'ensemble de nombres premiers $\mathbb{P}$ et l'ensemble de nombres entiers $\mathbb{N}$, dans laquelle chaque nombre premier p est associé à chaque nombre entier z qui est un multiple de p. C'est ainsi que dans cette relation, le nombre premier 7 est associé aux nombres qui incluent 0, 14, 21, 28, mais pas 1 ou 9.

Les relations binaires sont employées dans beaucoup de branches des mathématiques pour formaliser des concepts comme « est supérieur à », « est égal à », et « divise à » dans l'arithmétique, « est congruent à » dans la géométrie, « est adjacent à » dans la théorie

des graphes, et beaucoup d'autres. Les relations binaires sont également très utilisées en informatique, particulièrement dans le modèle relationnel pour les bases de données.

Après cette brève introduction informelle, nous présentons un ensemble de définitions, habituellement utilisées, concernant les relations binaires, et plus particulièrement le concept de relation d'équivalence que nous utilisons pour analyser le comportement d'une fonction d'évaluation.

Définition 17 (Relation binaire) *Une relation binaire $\mathcal{R}$ est un triplet ordonné $(X, Y, \mathcal{G})$, où X et Y sont deux ensembles arbitraires (ou classes), et $\mathcal{G}$ est un sous-ensemble du produit cartésien $X \times Y$. On appelle respectivement X et Y l'*ensemble de départ *et l'*ensemble d'arrivée *de la relation, et $\mathcal{G}$ son* graphe*.*

L'énoncé $(x, y) \in \mathcal{G}$ signifie que x est en relation avec y et on le note $x\mathcal{R}y$. Dans le cas particulier où $X = Y$ on dit que $\mathcal{R}$ est une relation sur X*.*

Une relation binaire peut être considérée comme une fonction de $X \times Y$ à valeur dans l'ensemble $\{Vrai, Faux\}$, et qui à un couple (x, y) associe $Vrai$ si x est en relation avec y et $Faux$ sinon, autrement dit une fonction indiquant si le couple (x, y) est un élément du graphe de la relation ou non.

Définition 18 (Relation réflexive) *Une relation $\mathcal{R}$ sur X est réflexive si $\forall x \in X,\ x\mathcal{R}x$.*

Définition 19 (Relation symétrique) *Une relation $\mathcal{R}$ sur X est symétrique si $\forall x, y \in X \times X$, dénoté aussi X^2, $x\mathcal{R}y \Rightarrow y\mathcal{R}x$.*

Définition 20 (Relation transitive) *Une relation $\mathcal{R}$ sur X est transitive si $\forall x, y, z \in X^3$, $(x\mathcal{R}y \wedge y\mathcal{R}z) \Rightarrow x\mathcal{R}z$.*

Définition 21 (Relation d'équivalence) *Une relation d'équivalence, dénotée souvent $\sim$, sur un ensemble X est une relation binaire qui est à la fois réflexive, symétrique et transitive.*

Définition 22 (Classe d'équivalence) *Soit $\sim$ une relation d'équivalence sur un ensemble X. Si $x \in X$, la classe d'équivalence contenant x, dénotée par $[x]$, est définie comme un sous-ensemble de la forme $[x] = \{y \in X : x \sim y\} \subseteq X$.*

Considérons un ensemble X muni d'une relation d'équivalence $\sim$ et un élément $x \in X$; la classe d'équivalence $[x]$ à laquelle il appartient présente les propriétés suivantes :

Propriété 6.1 *La classe d'équivalence $[x]$ n'est jamais vide, car elle contient toujours au moins x lui-même (puisque $\sim$, est réflexive).*

Propriété 6.2 *Tout élément de X appartient à au moins une classe d'équivalence : la sienne.*

Propriété 6.3 $([y] = [x]) \Rightarrow (y \in [x])$.

Propriété 6.4 *Si $y \in X$ et $y \notin [x]$, alors $[y] \cap [x] = \emptyset$.*

On déduit de ce qui précède que l'ensemble des classes d'équivalence de X forme une partition de X. Inversement, toute partition d'un ensemble y définit une relation d'équivalence. On peut établir une *bijection canonique* entre les partitions d'un ensemble et les relations d'équivalence dans cet ensemble.

Définition 23 (Ensemble des parties) *Si X désigne un ensemble, l'ensemble des parties de X, noté $\mathcal{P}(X)$, est l'ensemble de tous les sous-ensembles de X.*

Définition 24 (Ensemble quotient) *L'ensemble de toutes les classes d'équivalence possibles de X suivant $\sim$, noté $X/\sim = \{[x] : x \in X\}$, est appelé l'ensemble quotient de X suivant $\sim$.*

L'ensemble quotient est donc un nouvel ensemble construit à partir de X et de $\sim$, tel que $X/\sim \subseteq \mathcal{P}(X)$.

Le lecteur souhaitant une référence plus complète sur le sujet pourra consulter [Hungerford, 2003; Rotman, 2003].

Index

Références bibliographiques

[Aarts et Korst, 1989] cité p. 15, 109, 110, 111
E. H. L. Aarts and J. H. M. Korst. *Simulated Annealing and Boltzmann Machines: A Stochastic Approach to Combinatorial Optimization and Neural Computing*. John Wiley & Sons, 1989.

[Aarts et Laarhoven, 1985] cité p. 107, 110, 118
E. H. L. Aarts and P. J. M. Van Laarhoven. Statistical cooling: A general approach to combinatorial optimization problems. *Philips Journal of Research*, 40:193–226, 1985.

[Aarts et Lenstra, 1997] cité p. 13
E. H. L. Aarts and J. K. Lenstra, editors. *Local Search in Combinatorial Optimization*. John Wiley & Sons, 1997.

[Aarts et Van Laarhoven, 1985] cité p. 64
E. H. L. Aarts and P. J. M. Van Laarhoven. A new polynomial-time cooling schedule. In *Proceedings of the IEEE International Conference on Computer Aided Design*, pages 206–208, Santa Clara, CA, USA, 1985. IEEE Press.

[Abramson *et al.*, 1999] cité p. 15
D. Abramson, M. Krishnamoorthy, and H. Dang. Simulated annealing cooling schedules for the school timetabling problem. *Asia-Pacific Journal of Operational Research*, 16(1):1–22, 1999.

[Adolphson et Hu, 1973] cité p. 3, 77
D. L. Adolphson and T. C. Hu. Optimal linear ordering. *SIAM Journal on Applied Mathematics*, 25(3):403–423, 1973.

[Adolphson, 1977] cité p. 3, 77
D. L. Adolphson. Single machine job sequencing with precedence constraints. *SIAM Journal on Computing*, 6(1):40–54, 1977.

[Akyuz et Utku, 1968] cité p. 39
F. A. Akyuz and S. Utku. An automatic relabeling scheme for bandwidth minimization of stiffness matrices. *Journal of the American Institute of Aeronautics and Astronautics*, 6:728–730, 1968.

[Alway et Martin, 1965] cité p. 39
G. G. Alway and D. W. Martin. An algorithm for reducing the bandwidth of a matrix of symmetrical configuration. *The Computer Journal*, 8(3):264–272, 1965.

[Bar-Yehuda *et al.*, 2001] cité p. 78, 91, 102, 112
R. Bar-Yehuda, G. Even, J. Feldman, and J. Naor. Computing an optimal orientation of a balanced decomposition tree for linear arrangement problems. *Journal of Graph Algorithms and Applications*, 5(4):1–27, 2001.

[Battiti et Tecchiolli, 1994] cité p. 16
R. Battiti and G. Tecchiolli. The reactive tabu search. *ORSA Journal on Computing*, 6(2):126–140, 1994.

[Bäck *et al.*, 1997a] cité p. 17
T. Bäck, D. B. Fogel, and Z. Michalewicz, editors. *Handbook of Evolutionary Computation*. IOP Publishing Ltd., 1997.

[Bäck *et al.*, 1997b] cité p. 18, 99, 103
T. Bäck, U. Hammel, and H. P. Schwefel. Evolutionary computation: Comments on the history and current state. *IEEE Transactions on Evolutionary Computation*, 1(1):3–17, 1997.

[Beasley *et al.*, 1993] cité p. 18, 22, 35
D. Beasley, D. R. Bull, and R. R. Martin. An overview of genetic algorithms: Part 1, fundamentals. *University Computing*, 15(2):58–69, 1993.

[Berry *et al.*, 1996] cité p. 2, 40
M. Berry, B. Hendrickson, and P. Raghavan. Sparse matrix reordering schemes for browsing hypertext. *Lectures in Applied Mathematics*, 32:99–123, 1996.

[Bertsekas et Tsitsiklis, 1996] cité p. 31
D. Bertsekas and J. N. Tsitsiklis. *Neuro-Dynamic Programming*. Athena Scientific, Belmont, MA, USA, 1996.

[Blum et Roli, 2003] cité p. 13
C. Blum and A. Roli. Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM Computing Surveys*, 35(3):268–308, 2003.

[Boese et Kahng, 1994] cité p. 107
K. D. Boese and A. B. Kahng. Best-so-far vs. where-you-are: Implications for optimal finite-time annealing. *Systems and Control Letters*, 22(1):71–78, 1994.

[Bonabeau *et al.*, 1999] cité p. 19
E. Bonabeau, M. Dorigo, and G. Theraulaz. *Swarm Intelligence, From Natural to Artificial Systems*. Oxford University Press, New York, NY, USA, 1999.

[Boyan et Moore, 1994] cité p. 31
J. A. Boyan and A. W. Moore. Generalization in reinforcement learning: Safely approximating the value function. In G. Tesauro, D. S. Touretzky, and T. K. Leen, editors, *Proceedings of the 7th Conference on Advances in Neural Information Processing Systems*, pages 369–376, Denver, CO, USA, 1994. MIT Press.

[Boyan et Moore, 2000] cité p. 30, 31
J. A. Boyan and A. W. Moore. Learning evaluation functions to improve optimization by local search. *Journal of Machine Learning Research*, 1:77–112, 2000.

[Brailsford *et al.*, 1996] cité p. 23
S. C. Brailsford, P. M. Hubbard, and B. M. Smith. Organizing a social event: A difficult problem of combinatorial optimization. *Computers & Operations Research*, 23(9):845–856, 1996.

[Campbell *et al.*, 2002] cité p. 22
M. Campbell, A. J. Hoane Jr., and F. H. Hsu. Deep blue. *Artificial Intelligence*, 134:57–83, 2002.

[Cavique *et al.*, 1999] cité p. 2, 22, 64
L. Cavique, C. Rego, and I. Themido. Subgraph ejection chains and tabu search for the crew scheduling problem. *Journal of the Operational Research Society*, 50(6):608–616, 1999.

[Ĉerný, 1985] cité p. 13
V. Ĉerný. A thermodynamical approach to the traveling salesman problem: An efficient simulated algorithm. *Journal of Optimization Theory and Applications*, 45(1):41–51, 1985.

[Cha et Iwama, 1995] cité p. 24
B. Cha and K. Iwama. Performance test of local search algorithms using new types of random CNF formulas. In *Proceedings of the 14th International Joint Conference on Arificial Intelligence*, pages 304–310, Montréal, Canada, 1995. Morgan Kaufmann Publishers.

[Charon et Hudry, 1993] cité p. 24
I. Charon and O. Hudry. The noising method: A new method for combinatorial optimization. *Operations Research Letters*, 14(3):133–137, 1993.

[Charon et Hudry, 1999] cité p. 25
I. Charon and O. Hudry. Variations on the noising schemes for a clustering problem. In *Proceedings of the 3rd Metaheuristics International Conference*, pages 147–150, 1999.

[Charon et Hudry, 2001] cité p. 25
I. Charon and O. Hudry. The noising methods: A generalization of some metaheuristics. *European Journal of Operational Research*, 135(1):86–101, 2001.

[Charon et Hudry, 2002] cité p. 25
I. Charon and O. Hudry. *Essays and Surveys in Metaheuristics*, chapter The Noising Methods: A Survey, pages 245–261. Kluwer Academic Publishers, 2002.

[Cheng, 1973] cité p. 39
K. Y. Cheng. Note on minimizing the bandwidth of sparse symmetric matrices. *Computing*, 11(1):27–30, 1973.

[Chinn *et al.*, 1982] cité p. 39
P. Z. Chinn, J. Chvátalová, A. K. Dewdney, and N. E. Gibbs. The bandwidth problem for graphs and matrices - a survey. *Journal of Graph Theory*, 6(3):223–254, 1982.

[Codenotti *et al.*, 1996] cité p. 26
B. Codenotti, G. Manzini, and L. Margara. Perturbation: An efficient technique for the solution of very large instances of the euclidean TSP. *INFORMS Journal on Computing*, 8(2):125–133, 1996.

[Coello Coello, 2002] cité p. 24
C. A. Coello Coello. Theoretical and numerical constraint-handling techniques used with evolutionary algorithms: A survey of the state of the art. *Computer Methods in Applied Mechanics and Engineering*, 191(11-12):1245–1287, 2002.

[Cohen, 2004] cité p. 124
M. B. Cohen. *Designing Test Suites for Software Interaction Testing*. PhD thesis, University of Auckland, 2004.

[Collins *et al.*, 1988] cité p. 15
N. E. Collins, R. W. Eglese, and B. L. Golden. Simulated annealing: An annotated bibliography. *American Journal of Mathematical and Management Sciences*, 8(3-4):209–307, 1988.

[Collins, 1973] cité p. 39
R. J. Collins. Bandwidth reduction by automatic renumbering. *International Journal for Numerical Methods in Engineering*, 6(3):345–356, 1973.

[Comtet, 1974] cité p. 8
L. Comtet. *Advanced Combinatorics: The Art of Finite and Infinite Expansions*. D. Reidel Publishing Company, 1974.

[Connolly, 1990] cité p. 15
D. T. Connolly. An improved annealing scheme for the QAP. *European Journal of Operational Research*, 46(1):93–100, 1990.

[Corne *et al.*, 1999] cité p. 13, 18
D. Corne, M. Dorigo, and F. Glover, editors. *New Ideas in Optimization*, chapter Memetic Algorithms. McGraw-Hill, 1999.

[Coy *et al.*, 2000] cité p. 27
S. P. Coy, B. L. Golden, and E. A. Wasil. A computational study of smoothing heuristics for the traveling salesman problem. *European Journal of Operational Research*, 124(1):15–27, 2000.

[Crites et Barto, 1998] cité p. 31
R. H. Crites and A. G. Barto. Elevator group control using multiple reinforcement learning agents. *Machine Learning*, 33(2):235–262, 1998.

[Croes, 1958] cité p. 11
G. A. Croes. A method for solving traveling-salesman problems. *Operations Research*, 6(6):791–812, 1958.

[Cuthill et McKee, 1969] cité p. 39, 41, 42, 79
E. Cuthill and J. McKee. Reducing the bandwidth of sparse symmetric matrices. In *Proceedings of the 24th ACM National Conference*, pages 157–172, New York, NY, USA, 1969. ACM Press.

[Darwin, 1859] cité p. 17
C. Darwin. *On the Origin of Species by Means of Natural Selection, or the Preservation of Favoured Races in the Struggle for Life*. John Murray, London, UK, 1859.

[Davenport *et al.*, 1994] cité p. 23, 27
A. Davenport, E. Tsang, C. Wang, and K. Zhu. GENET: A connectionist architecture for solving constraint satisfaction problems by iterative improvement. In *Proceedings of the 12th National Conference on Artificial Intelligence*, pages 325–330, Seattle, WA, USA, 1994. AAAI Press/MIT Press.

[Davis, 1985] cité p. 99, 103
L. Davis. Applying adaptive algorithms to epistatic domains. In *Proceedings of the 9th International Joint Conference on Arificial Intelligence*, pages 162–164, Los Angeles, CA, USA, 1985. Morgan Kaufmann Publishers.

[Davis, 1991] cité p. 18, 99, 103
L. Davis. *Handbook of Genetic Algorithms*. Van Nostrad, New York, 1991.

[del Corso et Manzini, 1999] cité p. 40
G. M. del Corso and G. Manzini. Finding exact solutions to the bandwidth minimization problem. *Computing*, 62(3):189–203, 1999.

[Deutsch, 1976] cité p. 35
D. N. Deutsch. A "dogleg" channel router. In *Proceedings of the 13th ACM/IEEE Design Automation Conference*, pages 425–433, San Francisco, CA, USA, 1976. ACM Press.

[Diaz *et al.*, 2002] cité p. 40, 55, 60, 78, 97
J. Diaz, J. Petit, and M. Serna. A survey of graph layout problems. *ACM Computing Surveys*, 34(3):313–356, 2002.

[Diestel, 2005] cité p. 127
R. Diestel. *Graph Theory*. Springer, 3rd. edition edition, 2005.

[Dong *et al.*, 2003] cité p. 27
S. Dong, X. Hong, X. Qi, R. Wang, S. Chen, and J. Gu. VLSI module placement with pre-placed modules and considering congestion using solution space smoothing. In *Proceedings of the Asia and South Pacific Design Automation Conference*, pages 741–744, Kitakyushu, Japan, 2003. ACM Press.

[Dorigo et Colorni, 1996] cité p. 19
M. Dorigo and A. Colorni. Ant system: Optimization by a colony of cooperating agents. *IEEE Transactions on Systems, Man, and Cybernetics, Part B*, 26(1):29–41, 1996.

[Dorigo et Stützle, 2004] cité p. 19
M. Dorigo and T. Stützle. *Ant Colony Optimization*. MIT Press, Boston, MA, USA, 2004.

[Dowsland, 1993] cité p. 15
K. A. Dowsland. *Modern Heuristic Techniques for Combinatorial Problems*, chapter Simulated Annealing, pages 20–69. John Wiley & Sons, Oxford, UK, 1993.

[Duan *et al.*, 1992] cité p. 1, 8
Q. Duan, S. Sorooshian, and V. Gupta. Effective and efficient global optimization of conceptual rainfall-runoff models. *Water resources research*, 28(4):1015–1031, 1992.

[Dueck et Jeffs, 1995] cité p. 41, 42, 45, 55, 60, 62, 68
G. W. Dueck and J. Jeffs. A heuristic bandwidth reduction algorithm. *Journal of Combinatorial Mathematics and Combinatorial Computing*, 18:97–108, 1995.

[Duvivier *et al.*, 1996] cité p. 2, 22, 72, 105
D. Duvivier, P. Preux, and E. G. Talbi. Climbing-up NP-Hard hills. *Lecture Notes in Computer Science*, 1141:574–583, 1996.

[Eiben et Schoenauer, 2002] cité p. 17
A.E. Eiben and M. Schoenauer. Evolutionary computation. *Information Processing Letters*, 82(1):1–6, 2002.

[Eiben et Smith, 2003] cité p. 17
A. E. Eiben and J. E. Smith. *Introduction to Evolutionary Computing*. Springer Verlag, Berlin, Germany, 2003.

[Esposito *et al.*, 1998a] cité p. 2, 40
A. Esposito, M. S. Fiorenzo Catallano, F. Malucelli, and L. Tarricone. *Advances in the Theory of Computation and Computational Mathematics, Volume 2, High Performance Algorithms for Structured Matrix Problems*, chapter Sparse Matrix Bandwith Reduction: Algorithms, Applications and Real Industrial Cases in Electromagnetics, pages 27–45. Nova Biomedical Press, 1998.

[Esposito *et al.*, 1998b] cité p. 41
A. Esposito, M. S. Fiorenzo Catallano, F. Malucelli, and L. Tarricone. A new matrix bandwidth reduction algorithm. *Operations Research Letters*, 23(3):99–107, 1998.

[Euler, 1741] cité p. 125
L. Euler. Solutio problematis ad geometriam situs pertinentis (the solution of a problem relating to the geometry of position). *Commentarii Academiae Scientiarum Petropolitanae*, 8:128–140, 1741.

[Even et Shiloah, 1975] cité p. 77
S. Even and Y. Shiloah. NP-completeness of several arrangement problems. Technical Report CS0043, Computer Science Department, Technion, Israel Institute of Technology, Haifa, Israel, January 1975.

[Falkenauer et Delchambre, 1992] cité p. 34
E. Falkenauer and A. Delchambre. A genetic algorithm for bin packing and line balancing. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 1186–1192, Los Alamitos, CA, USA, 1992. IEEE Press.

[Falkenauer, 1996] cité p. 19, 34
E. Falkenauer. A hybrid grouping genetic algorithm for bin packing. *Journal of Heuristics*, 2:5–30, 1996.

[Féo et Resende, 1995] cité p. 19
T. A. Féo and M. G. C. Resende. Greedy randomized adaptative search procedures. *Journal of Global Optimization*, 6:109–133, 1995.

[Fogel *et al.*, 1966] cité p. 17
L.J. Fogel, A.J. Owens, and M.J. Walsh. *Artificial Intelligence through Simulated Evolution*. John Wiley, New York, NY, USA, 1966.

[Fogel, 1995] cité p. 17
D.B. Fogel. *Evolutionary Computation, Toward a New Philosophy of Machine Intelligence*. IEEE Press, Piscataway, NJ, USA, 1995.

[Fonlupt *et al.*, 1998] cité p. 56, 124
C. Fonlupt, D. Robilliard, and E. G. Talbi P. Preux. *Meta-heuristics: Advances and Trends in Local Search Paradigms for Optimization*, chapter Fitness Landscape and Performance of Meta-Heuristics, pages 255–266. Kluwer Academic Publishers, 1998.

[Frank *et al.*, 1997] cité p. 24, 56, 124
J. Frank, P. Cheeseman, and J. Stutz. When gravity fails: Local search topology. *Journal of Artificial Intelligence Research*, 7:249–281, 1997.

[Frank, 1996] cité p. 24
J. Frank. Weighting for godot: Learning heuristics for GSAT. In *Proceedings of the 13th National Conference on Artificial Intelligence*, pages 338–343, Portland, OR, USA, 1996. MIT Press.

[Freisleben et Merz, 1996a] cité p. 19
B. Freisleben and P. Merz. New genetic local search operators for the traveling salesman problem. *Lecture Notes in Computer Science*, 1141:890–900, 1996.

[Freisleben et Merz, 1996b] cité p. 19, 99, 103
B. Freisleben and Peter Merz. A genetic local search algorithm for solving symmetric and asymmetric traveling salesman problems. In *Proceedings of the IEEE International Conference on Evolutionary Computation*, pages 616–621. IEEE Press, 1996.

[Galinier et Hao, 1998] cité p. 23
P. Galinier and J. K. Hao. *Meta-heuristics: Advances and Trends in Local Search Paradigms for Optimization*, chapter Solving the Progressive Party Problem by Local Search, pages 418–432. Kluwer Academic Publishers, 1998.

[Galinier et Hao, 1999] cité p. 19
P. Galinier and J. K. Hao. Hybrid evolutionary algorithms for graph coloring. *Journal of Combinatorial Optimization*, 3(4):379–397, 1999.

[Garey *et al.*, 1978] cité p. 40
M. R. Garey, R. L. Graham, D. S. Johnson, and D. E. Knuth. Complexity results for bandwidth minimization. *SIAM Journal on Applied Mathematics*, 34(3):477–495, 1978.

[Garey et Johnson, 1979] cité p. 1, 2, 9, 77
M. R. Garey and D. S. Johnson. *Computers and Intractability: A guide to the Theory of NP-Completeness*. W.H. Freeman and Company, New York, 1979.

[Garnier et Kallel, 2000] cité p. 124
J. Garnier and L. Kallel. Statistical distribution of the convergence time of evolutionary algorithms for long-path problems. *IEEE Transactions on Evolutionary Computation*, 4(1):16–30, 2000.

[Garnier et Kallel, 2002] cité p. 124
J. Garnier and L. Kallel. Efficiency of local search with multiple local optima. *SIAM Journal on Discrete Mathematics*, 15(1):122–141, 2002.

[Gendreau *et al.*, 1994] cité p. 24
M. Gendreau, A. Hertz, and G. Laporte. A tabu search heuristic for the vehicle routing problem. *Management Science*, 40:1276–1290, 1994.

[Gibbs *et al.*, 1976] cité p. 41, 68, 79
N. E. Gibbs, W. G. Poole, and P. K. Stockmeyer. An algorithm for reducing the bandwidth and profile of a sparse matrix. *SIAM Journal on Numerical Analysis*, 13(2):236–250, 1976.

[Gibbs, 1976] cité p. 79
N. E. Gibbs. Algorithm 509: A hybrid profile reduction algorithm. *ACM Transactions on Mathematical Software*, 2(4):378–387, 1976.

[Glover *et al.*, 1993] cité p. 16
F. Glover, E. Taillard, and D. de Werra. A user's guide to tabu search. *Annals of Operations Research*, 41(1):3–28, 1993.

[Glover et Hanafi, 2002] cité p. 16
F. Glover and S. Hanafi. Tabu search and finite convergence. *Discrete Applied Mathematics*, 119(1-2):3–36, 2002.

[Glover et Laguna, 1993] cité p. 16
F. Glover and M. Laguna. *Modern Heuristic Techniques for Combinatorial Problems*, chapter Tabu Search, pages 71–141. John Wiley & Sons, Oxford, UK, 1993.

[Glover et Laguna, 1997] cité p. 16, 99
F. Glover and M. Laguna. *Tabu Search*. Kluwer Academic Publishers, 1997.

[Glover, 1977] cité p. 15
F. Glover. Heuristics for integer programming using surrogate constraints. *Decision Sciences*, 8:156–166, 1977.

[Glover, 1986] cité p. 13, 15
F. Glover. Future paths for integer programming and links to artificial intelligence. *Computers & Operations Research*, 13(5):533–549, 1986.

[Glover, 1989] cité p. 15
F. Glover. Tabu Search - Part I. *ORSA Journal on Computing*, 1(3):190–206, 1989.

[Glover, 1990] cité p. 15, 16
F. Glover. Tabu Search - Part II. *ORSA Journal on Computing*, 2(1):4–32, 1990.

[Glover, 1994] cité p. 16
F. Glover. *Mathematical Programming: State of the Art*, chapter Tabu Search: Improved Solution Alternatives for Real World Problems, pages 64–92. University of Michigan, 1994.

[Glover, 1996] cité p. 2, 22, 64
F. Glover. Ejection chains, reference structures and alternating path methods for traveling salesman problems. *Discrete Applied Mathematics*, 65(1-3):223–253, 1996.

[Glover, 1997] cité p. 16
F. Glover. *Interfaces in Computer Science and Operations Research*, chapter Tabu Search and Adaptive Memory Programming - Advances, Applications and Challenges, pages 1–75. Kluwer Academic Publishers, 1997.

[Glover, 1999] cité p. 19
F. Glover. *New Ideas in Optimization*, chapter Scatter Search and Path Relinking. McGraw-Hill, 1999.

[Goëffon *et al.*, 2006] cité p. 19
A. Goëffon, J. M. Richer, and J. K. Hao. A distance-based information preservation tree crossover for the maximum parsimony problem. *Lecture Notes in Computer Science*, 4193:761–770, 2006.

[Goldberg et Lingle, 1985] cité p. 96, 99, 103
D. E. Goldberg and R. Lingle. Alleles, loci, and the travelling salesman problem. In *Proceedings of the International Conference on Genetic Algorithms*, pages 154–159. Carnegie Mellon publishers, 1985.

[Goldberg, 1989] cité p. 17
D. E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley Publishing Company, Inc., 1989.

[Greene et Supowit, 1988] cité p. 107
J. W. Greene and K. J. Supowit. Simulated annealing without rejected moves. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 5(1):221–228, 1988.

[Grefenstette, 1987] cité p. 18, 99, 103
J. J. Grefenstette. *Genetic Algorithms and Simulated Annealing*, chapter Incorporating Problem Specific Knowledge Into Genetic Algorithms, pages 42–60. Morgan Kaufmann Publishers, 1987.

[Gribkovskaia *et al.*, 2007] cité p. 24
I. Gribkovskaia, Ø. Halskau-sr, G. Laporte, and M. Vlĉek. General solutions to the single vehicle routing problem with pickups and deliveries. *European Journal of Operational Research*, 180(2):568–584, 2007.

[Gross et Yellen, 2005] cité p. 127
J. L. Gross and J. Yellen. *Graph Theory and its Applications*. Chapman & Hall, 2nd. edition edition, 2005.

[Grover, 1986] cité p. 107
L. K. Grover. A new simulated annealing algorithm for standard cell placement. In *Proceedings of the IEEE International Conference on Computer Aided Design*, pages 378–380, Santa Clara, CA, USA, 1986. IEEE Press.

[Grover, 1987] cité p. 107
L. K. Grover. Standard cell placement using simulated sintering. In *Proceedings of the 24th ACM/IEEE Design Automation Conference*, pages 56–59, Miami Beach, FL, USA, 1987. IEEE Press.

[Gu et Huang, 1994] cité p. 26
J. Gu and X. Huang. Efficient local search with search space smoothing: A case study of the traveling salesman problem (TSP). *IEEE Transactions on Systems, Man and Cybernetics*, 24(5):728–735, 1994.

[Gu, 1990] cité p. 26
J. Gu. Optimization by multispace search. Technical Report UCECE-TR-90-001, Department of Electrical and Computer Engineering, University of Calgary, 1990.

[Gu, 1994] cité p. 26
J. Gu. Multispace search: A new optimization approach. In *Proceedings of the 5th International Symposium on Algorithms and Computation*, pages 252–260, London, UK, 1994. Springer-Verlag.

[Gurari et Sudborough, 1984] cité p. 40
E. M. Gurari and I. H. Sudborough. Improved dynamic programming algorithms for bandwidth minimization and the min-cut linear arrangement problem. *Journal of Algorithms*, 5(4):531–546, 1984.

[Hajek, 1988] cité p. 15, 64, 107, 109
B. Hajek. Cooling schedules for optimal annealing. *Mathematics of Operations Research*, 13(2):311–329, 1988.

[Hao *et al.*, 1999] cité p. 18, 23
J. K. Hao, P. Galinier, and M. Habib. Metaheuristiques pour l'optimisation combinatoire et l'affectation sous contraintes. *Revue d'Intelligence Artificielle*, 13(2):283–324, 1999.

[Harary, 1967] cité p. 39
F. Harary. *Theory of Graphs and its Applications*. Czechoslovak Academy of Science, Prague, 1967.

[Harper, 1964] cité p. 3, 39, 77
L. H. Harper. Optimal assignment of numbers to vertices. *SIAM Journal on Applied Mathematics*, 12(1):131–135, 1964.

[Hart *et al.*, 2004] cité p. 19
W. E. Hart, N. Krasnogor, and J. E. Smith, editors. *Recent Advances in Memetic Algorithms and Related Search Technologies*. Springer-Verlag, 2004.

[Hertz *et al.*, 1994] cité p. 56, 124
A. Hertz, B. Jaumard, and M. Poggi de Aragão. Local optima topology for the k-coloring problem. *Discrete Applied Mathematics*, 49(1-3):257–280, 1994.

[Hertz et Kobler, 2000] cité p. 17
A. Hertz and D. Kobler. A framework for the description of evolutionary algorithms. *European Journal of Operational Research*, 126(1):1–12, 2000.

[Hertz et Widmer, 2003] cité p. 2, 22, 72, 105
A. Hertz and M. Widmer. Guidelines for the use of meta-heuristics in combinatorial optimization. *European Journal of Operational Research*, 151(2):247–252, 2003.

[Hogg et Craig, 1970] cité p. 108
R. V. Hogg and A. T. Craig. *Introduction to Mathematical Statistics*. Macmillan, New York, 1970.

[Holland, 1975] cité p. 17, 18
J. H. Holland. *Adaptation in natural and artificial systems*. The University of Michigan Press, Ann Arbor, MI, USA, 1975.

[Hoos et Stützle, 2004] cité p. 2, 13, 18, 22, 35, 72, 103
H. H. Hoos and T. Stützle. *Stochastic Local Search: Foundations and Applications*. Morgan Kaufmann, 2004.

[Hsu *et al.*, 1990] cité p. 22, 35
F. H. Hsu, T. S. Anantharaman, M. Campbell, and A. Nowatzyk. A grandmaster chess machine. *Scientific American*, 263(4):44–50, 1990.

[Huang *et al.*, 1986] cité p. 15, 64, 107
M. D. Huang, F. Romeo, and A. L. Sangiovanni-Vincentelli. An efficient general cooling schedule for simulated annealing. In *Proceedings of the IEEE International Conference on Computer Aided Design*, pages 381–384, Santa Clara, CA, USA, 1986. IEEE Press.

[Hungerford, 2003] cité p. 129
T. W. Hungerford. *Graduate Texts in Mathematics: Algebra*. Springer, 2003.

[Hutter *et al.*, 2002] cité p. 24
F. Hutter, D. A. D. Tompkins, and H. H. Hoos. Scaling and probabilistic smoothing: Efficient dynamic local search for SAT. *Lecture Notes in Computer Science*, 2470:233–248, 2002.

[Ibaraki, 1997] cité p. 13, 19
T. Ibaraki. *Handbook of Evolutionary Computation*, chapter D3, Combinations with other optimization methods. IOP Publishing Ltd., 1997.

[Johnson *et al.*, 1989] cité p. 15, 23, 107
D. S. Johnson, C. R. Aragon, L. A. McGeoch, and C. Schevon. Optimization by simulated annealing: An experimental evaluation; part I, graph partitioning. *Operations Research*, 37(6):865–892, 1989.

[Johnson *et al.*, 1991] cité p. 15, 33
D. S. Johnson, C. R. Aragon, L. A. McGeoch, and C. Schevon. Optimization by simulated annealing: An experimental evaluation; part II, graph coloring and number partitioning. *Operations Research*, 39(3):378–406, 1991.

[Johnson et Papadimitriou, 1985] cité p. 9
D. S. Johnson and C. H. Papadimitriou. *The Traveling Salesman Problem: A Guided Tour of Combinatorial Optimization*, chapter 3, Computational Complexity, pages 37–85. John Wiley & Sons, 1985.

[Juvan et Mohar, 1992] cité p. 78
M. Juvan and B. Mohar. Optimal linear labelings and eigenvalues of graphs. *Discrete Applied Mathematics*, 36(2):153–168, 1992.

[Karp, 1993] cité p. 3, 77
R. M. Karp. Mapping the genome: some combinatorial problems arising in molecular biology. In *Proceedings of the 25th annual ACM symposium on Theory of computing*, pages 278–285, San Diego, CA, USA, 1993. ACM Press.

[Kauffman et Levin, 1987] cité p. 123
S. Kauffman and S. Levin. Towards a general theory of adaptive walks on rugged landscapes. *Journal of Theoretical Biology*, 128(1):11–45, 1987.

[Khanna *et al.*, 1994] cité p. 33
S. Khanna, R. Motwani, M. Sudan, and U. Vazirani. On syntactic versus computational views of approximability. In *Proceedings of the 35th IEEE Symposium on Foundations of Computer Science*, pages 819–830. IEEE Press, 1994.

[Kirkpatrick *et al.*, 1983] cité p. 13, 14, 60, 107, 118
S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220:671–680, 1983.

[Kirkpatrick et Toulouse, 1985] cité p. 56, 124
S. Kirkpatrick and G. Toulouse. Configuration space analysis of travelling salesman problems. *Physique*, 46:1277–1292, 1985.

[Koopman, 1957] cité p. 27
B. O. Koopman. The theory of search, part III, the optimum distribution of searching effort. *Operations Research*, 5:613–626, 1957.

[Koopmans et Beckmann, 1957] cité p. 9
T. C. Koopmans and M. J. Beckmann. Assignment problems and the location of economic activities. *Econometrica*, 25:53–76, 1957.

[Koren et Harel, 2002] cité p. 78, 79, 91, 102, 112, 114
Y. Koren and D. Harel. A multi-scale algorithm for the linear arrangement problem. *Lecture Notes in Computer Science*, 2573:293–306, 2002.

[Kosko, 1957] cité p. 39
E. Kosko. Matrix inversion by partitioning. *The Aeronautical Quarterly*, 8:157–184, 1957.

[Koza, 1992] cité p. 17, 124
J. R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, 1992.

[Kratsch, 1987] cité p. 40
D. Kratsch. Finding the minimum bandwidth of an interval graph. *Information and Computation*, 74(2):140–158, 1987.

[Lacomme *et al.*, 2006] cité p. 33
P. Lacomme, C. Prins, and M. Sevaux. A genetic algorithm for a bi-objective capacitated arc routing problem. *European Journal of Operational Research*, 33(12):3473–3493, 2006.

[Laguna et Martí, 2003] cité p. 19
M. Laguna and R. Martí. *Scatter Search: Methodology and Implementations in C*. Kluwer Academic Publishers, Boston, MA, USA, 2003.

[Lai et Williams, 1999] cité p. 3, 77
Y. L. Lai and K. Williams. A survey of solved problems and applications on bandwidth, edge-sum, and profile of graphs. *Graph Theory*, 31:75–94, 1999.

[Lam et Delosme, 1988] cité p. 107
J. Lam and J. M. Delosme. Performance of a new annealing schedule. In *Proceedings of the 25th ACM/IEEE Design Automation Conference*, pages 306–311, Atlantic City, NJ, USA, 1988. IEEE Press.

[Land et Doig, 1960] cité p. 10
A. H. Land and A. G. Doig. An automatic method of solving discrete programming problems. *Econometrica*, 28(3):497–520, 1960.

[Lardeux *et al.*, 2006] cité p. 19
F. Lardeux, F. Saubion, and J. K. Hao. Gasat: A genetic local search algorithm for the satisfiability problem. *Evolutionary Computation*, 14(2):223–253, 2006.

[Lim *et al.*, 2003] cité p. 55, 62
A. Lim, B. Rodrigues, and F. Xiao. Integrated genetic algorithm with hill climbing for bandwidth minimization problem. *Lecture Notes in Computer Science*, 2724:1594–1595, 2003.

[Lim *et al.*, 2006] cité p. 41, 55
A. Lim, B. Rodrigues, and F. Xiao. Heuristics for matrix bandwidth reduction. *European Journal of Operational Research*, 174:69–91, 2006.

[Lin et Kernighan, 1973] cité p. 2, 22, 27, 64
S. Lin and B. W. Kernighan. An effective heuristic algorithm for the traveling salesman problem. *Operations Research*, 21(2):498–516, 1973.

[Liu et Vannelli, 1995] cité p. 79
W. Liu and A. Vannelli. Generating lower bounds for the linear arrangement problem. *Discrete Applied Mathematics*, 59(2):137–151, 1995.

[Livesley, 1960] cité p. 39
R. K. Livesley. The analysis of large structural systems. *The Computer Journal*, 3(1):34–39, 1960.

[Lotkin et Remage, 1952] cité p. 39
M. Lotkin and R. Remage. Matrix inversion by partitioning. In *Proceedings of the 1952 ACM National Meeting (Toronto)*, pages 36–41, Toronto, Ontario, Canada, 1952. ACM Press.

[Lundy et Mees, 1986] cité p. 15
M. Lundy and A. Mees. Convergence of an annealing algorithm. *Mathematical Programming*, 34(1):111–124, 1986.

[Martí *et al.*, 2001] cité p. 41, 42, 44, 55, 62, 65, 67
R. Martí, M. Laguna, F. Glover, and V. Campos. Reducing the bandwidth of a sparse matrix with tabu search. *European Journal of Operational Research*, 135(2):211–220, 2001.

[Martí, 2004] cité p. 67
Rafael Martí. Personal communication. September, 2004.

[McAllister, 1999] cité p. 78, 79, 99, 109, 110, 112, 122
A. J. McAllister. A new heuristic algorithm for the linear arrangement problem. Technical Report TR-99-126a, Faculty of Computer Science, University of New Brunswick, 1999.

[Merz et Freisleben, 1997] cité p. 19
P. Merz and B. Freisleben. A genetic local search approach to the quadratic assignment problem. In *Proceedings of the 7th International Conference on Genetic Algorithms*, pages 465–472, East Lansing, MI, USA, 1997. Morgan Kaufmann Publishers Inc.

[Merz et Freisleben, 1999] cité p. 19
P. Merz and B. Freisleben. A comparison of memetic algorithms, tabu search, and ant colonies for the quadratic assignment problem. In *Proceedings of the IEEE International Congress on Evolutionary Computation*, pages 2063–2070, Washington DC, USA, 1999. IEEE Press.

[Merz et Freisleben, 2000] cité p. 97
P. Merz and B. Freisleben. Fitness landscapes, memetic algorithms and greedy operators for graph bi-partitioning. *Evolutionary Computation*, 8(1):61–91, 2000.

[Metropolis *et al.*, 1953] cité p. 13
N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller, and E. Teller. Equation of state calculations by fast computing machines. *The Journal of Chemical Physics*, 21(6):1087–1092, 1953.

[Michalewicz et Schoenauer, 1996] cité p. 24
Z. Michalewicz and M. Schoenauer. Evolutionary algorithms for constrained parameter optimization problems. *Evolutionary Computation*, 4(1):1–32, 1996.

[Michalewicz, 1996] cité p. 17
Z. Michalewicz. *Genetic Algorithms + Data Structures = Evolution Programs*. Springer-Verlag, Berlin, Germany, 3rd edition, 1996.

[Mills *et al.*, 2003] cité p. 28
P. Mills, E. Tsang, and J. Ford. Applying an extended guided local search on the quadratic assignment problem. *Annals of Operations Research*, 118:121–135, 2003.

[Mills et Tsang, 2000] cité p. 28
P. Mills and E. Tsang. Guided local search for solving SAT and weighted MAX-SAT problems. *Journal of Automated Reasoning, Special Issue on Satisfiability Problems*, 24:205–223, 2000.

[Mitchison et Durbin, 1986] cité p. 3, 77
G. Mitchison and R. Durbin. Optimal numbering of an nxn array. *SIAM Journal on Matrix Analysis and Applications*, 7(4):571–582, 1986.

[Mladenović et Hansen, 1997] cité p. 2, 19, 22
N. Mladenović and P. Hansen. Variable neighborhood search. *Computers & Operations Research*, 24(11):1097–1100, 1997.

[Morris, 1993] cité p. 28, 29
P. Morris. The breakout method for escaping local minima. In *Proceedings of the 11th National Conference on Artificial Intelligence*, pages 40–45, Washington, DC, USA, 1993. AAAI Press/MIT Press.

[Moscato, 1989] cité p. 18
P. Moscato. On evolution, search, optimization, genetic algorithms and martial arts: Towards memetic algorithms. Technical Report C3P Report 826, Caltech Concurrent Computation Program, Pasadena, CA, 1989.

[Moscato, 1999] cité p. 18, 99
P. Moscato. *New Ideas in Optimization*, chapter Memetic Algorithms: A Short Introduction. McGraw-Hill, 1999.

[Nagata, 2006] cité p. 18
Y. Nagata. New EAX crossover for large TSP instances. *Lecture Notes in Computer Science*, 4193:372–381, 2006.

[Nagata, 2007] cité p. 18
Y. Nagata. Edge assembly crossover for the capacitated vehicle routing problem. *Lecture Notes in Computer Science*, 4446:142–153, 2007.

[Neumaier, 1997] cité p. 1, 8
A. Neumaier. Molecular modeling of proteins and mathematical prediction of protein structure. *SIAM Review*, 39(3):407–460, 1997.

[Nilsson, 1986] cité p. 22, 35
N. J. Nilsson. *Principles of Artificial Intelligence*. Morgan Kaufmann, 1986.

[Nurmela, 2004] cité p. 124
K. J. Nurmela. Upper bounds for covering arrays by tabu search. *Discrete Applied Mathematics*, 138(1-2):143–152, 2004.

[Oliver *et al.*, 1987] cité p. 99, 103
I. M. Oliver, D. J. Smith, and J. R. C. Holland. A study of permutation crossover operators on the travel salesman problem. In J. J. Grefenstette, editor, *Proceedings of the 2nd International Conference on Genetic Algorithms*, pages 224–230, Mahwah, NJ, USA, 1987. Lawrence Erlbaum Associates, Inc.

[Olsen, 1994] cité p. 23
A. L. Olsen. Penalty functions for the knapsack problem. In *Proceedings of the 1st IEEE Conference on Evolutionary Computation*, pages 554–558. IEEE Press, 1994.

[Osman et Kelly, 1996] cité p. 13
I. H. Osman and J. P. Kelly, editors. *Meta-Heuristics: Theory & Applications*. Kluwer Academic Publishers, Norwell, MA, USA, 1996.

[Osman, 1993] cité p. 15
I. H. Osman. Metastrategy simulated annealing and tabu search algorithms for the vehicle routing problems. *Annals of Operations Research*, 41(4):421–451, 1993.

[Osman, 1995] cité p. 14
I. H. Osman. *Operational Research Tutorial Papers*, chapter An Introduction to Meta-Heuristics, pages 92–122. Operational Research Society Press, Birmingham, UK., 1995.

[Otten et Van Ginneken, 1988] cité p. 109
R. H. J. M. Otten and L. P. P. P. Van Ginneken. Stop criterion in simulated annealing. In *Proceedings of the IEEE International Conference on Computer Design: VLSI in Computers and Processors*, pages 549–552, Rye Brook, NY, USA, 1988. IEEE Press.

[Papadimitriou et Steiglitz, 1982] cité p. 1, 9, 10
C. H. Papadimitriou and K. Steiglitz. *Combinatorial Optimization Algorithms and Complexity*. Prentice Hall, 1982.

[Papadimitriou, 1976] cité p. 2, 40
C. H. Papadimitriou. The NP-Completeness of the bandwidth minimization problem. *Computing*, 16(3):263–270, 1976.

[Petit, 2003a] cité p. 78, 102, 112
J. Petit. Combining spectral sequencing and parallel simulated annealing for the MinLA problem. *Parallel Processing Letters*, 13(1):71–91, 2003.

[Petit, 2003b] cité p. 78, 79, 80, 91, 102, 111, 112
J. Petit. Experiments on the minimum linear arrangement problem. *The ACM Journal of Experimental Algorithmics*, 8, 2003.

[Piñana *et al.*, 2004] cité p. 2, 41, 44, 55, 62, 67, 68
E. Piñana, I. Plana, V. Campos, and R. Martí. GRASP and path relinking for the matrix bandwidth minimization. *European Journal of Operational Research*, 153(1):200–210, 2004.

[Pirlot, 1996] cité p. 13
M. Pirlot. General local search methods. *European Journal of Operational Research*, 92(3):493–511, 1996.

[Poranen, 2005] cité p. 78, 91
T. Poranen. A genetic hillclimbing algorithm for the optimal linear arrangement problem. *Fundamenta Informaticae*, 68(4):333–356, 2005.

[Poupaert et Deville, 2000] cité p. 15, 110
E. Poupaert and Y. Deville. Simulated annealing with estimated temperature. *AI Communications*, 13(1):19–26, 2000.

[Ravi *et al.*, 1991] cité p. 3, 77
R. Ravi, A. Agrawal, and P. N. Klein. Ordering problems approximated: Single-processor scheduling and interval graph completion. *Lecture Notes in Computer Science*, 510:751–762, 1991.

[Rayward-Smith *et al.*, 1996] cité p. 13
V. J. Rayward-Smith, I. H. Osman, C. R. Reeves, and G. D. Smith, editors. *Modern Heuristic Search Methods*. John Wiley & Sons, 1996.

[Rechenberg, 1973] cité p. 17
I. Rechenberg. *Evolutionstrategie: Optimierung Technisher Systeme nach Prinzipien des Biologischen Evolution*. Fromman-Hozlboog Verlag, Stuttgart, Germany, 1973.

[Reeves, 1993] cité p. 13
C. R. Reeves, editor. *Modern Heuristic Techniques for Combinatorial Problems*. John Wiley & Sons, Oxford, UK, 1993.

[Reidys *et al.*, 1997] cité p. 123
C. Reidys, P. F. Stadler, and P. Schuster. Generic properties of combinatory maps: Neutral networks of rna secondary structures. *Bulletin of Mathematical Biology*, 59(2):339–397, 1997.

[Rodriguez-Tello *et al.*, 2004] cité p. 47, 53
E. Rodriguez-Tello, J. K. Hao, and J. Torres-Jimenez. An improved evaluation function for the bandwidth minimization problem. *Lecture Notes in Computer Science*, 3242:650–659, 2004.

[Rodriguez-Tello *et al.*, 2005] cité p. 89
E. Rodriguez-Tello, J. K. Hao, and J. Torres-Jimenez. A comparison of memetic recombination operators for the MinLA problem. *Lecture Notes in Computer Science*, 3789:613–622, 2005.

[Rodriguez-Tello *et al.*, 2006a] cité p. 53
E. Rodriguez-Tello, J. K. Hao, and J. Torres-Jimenez. An improved simulated annealing algorithm for bandwidth minimization. *European Journal of Operational Research*, In Press: Elsevier, 2006.

[Rodriguez-Tello *et al.*, 2006b] cité p. 89, 91
E. Rodriguez-Tello, J. K. Hao, and J. Torres-Jimenez. Memetic algorithms for the MinLA problem. *Lecture Notes in Computer Science*, 3871:73–84, 2006.

[Rodriguez-Tello *et al.*, 2006c] cité p. 75, 89, 112
E. Rodriguez-Tello, J. K. Hao, and J. Torres-Jimenez. A refined evaluation function for the MinLA problem. *Lecture Notes in Computer Science*, 4293:392–403, 2006.

[Rodriguez-Tello *et al.*, 2007] cité p. 89
E. Rodriguez-Tello, J. K. Hao, and J. Torres-Jimenez. An effective two-stage simulated annealing algorithm for the minimum linear arrangement problem. *Computers & Operations Research*, In Press: Elsevier, 2007.

[Rose *et al.*, 1990] cité p. 107
J. Rose, W. Klebsch, and J. Wolf. Temperature measurement and equilibrium dynamics of simulated annealing placements. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 9(3):253–259, 1990.

[Rosen, 1968] cité p. 39
R. Rosen. Matrix bandwidth minimization. In *Proceedings of the 23rd ACM National Conference*, pages 585–595, New York, NY, USA, 1968. ACM Press.

[Rotman, 2003] cité p. 46, 81, 129
J. J. Rotman. *Advanced Modern Algebra*. Prentice Hall, 2003.

[Runarsson et Yao, 2002] cité p. 23
T. P. Runarsson and X. Yao. *Evolutionary Optimization*, chapter Constrained Evolutionary Optimization: The Penalty Function Approach, pages 87–113. Kluwer Academic Publishers, 2002.

[Russell *et al.*, 2006] cité p. 33
R. Russell, W. C. Chiang, and D. Zepeda. Integrating multi-product production and distribution in newspaper logistics. *Computers & Operations Research*, In Press:Elsevier, 2006.

[S. Szykman et Weisser, 1998] cité p. 24
J. Cagan S. Szykman and P. Weisser. An integrated approach to optimal three dimensional layout and routing. *ASME Journal of Mechanical Design*, 120(3):510–512, 1998.

[Safro *et al.*, 2006] cité p. 78, 80, 91, 102, 112, 114
I. Safro, D. Ron, and A. Brandt. Graph minimum linear arrangement by multilevel weighted edge contractions. *Journal of Algorithms*, 60(1):24–41, 2006.

[Schneider *et al.*, 1997] cité p. 27
J. Schneider, M. Dankesreiter, W. Fettes, I. Morgenstern, M. Schmid, and J. M. Singer. Search-space smoothing for combinatorial optimization problems. *Physica A: Statistical and Theoretical Physics*, 243(1-2):77–112, 1997.

[Schoofs et Naudts, 2000] cité p. 23
L. Schoofs and B. Naudts. Solving csp instances beyond the phase transition using stochastic search algorithms. *Lecture Notes in Computer Science*, 1917:549–558, 2000.

[Schuster, 1997] cité p. 124
P. Schuster. Landscapes and molecular evolution. *Physica D*, 107:351–365, 1997.

[Schwefel, 1981] cité p. 17
H. P. Schwefel. *Numerical Optimization of Computer Models*. John Wiley & Sons, New York, NY, USA, 2nd. edn., 1995 edition, 1981.

[Selman et Kautz, 1993] cité p. 24
B. Selman and H. A. Kautz. Domain-independent extensions to GSAT: Solving large structured satisfiability problems. In *Proceedings of the 13th International Joint Conference on Arificial Intelligence*, pages 290–295, Chambéry, France, 1993. Morgan Kaufmann Publishers.

[Shahrokhi *et al.*, 2001] cité p. 3, 77
F. Shahrokhi, O. Sykora, L. A. Szekely, and I. Vrto. On bipartite drawings and the linear arrangement problem. *SIAM Journal on Computing*, 30(6):1773–1789, 2001.

[Singh et Gupta, 2005] cité p. 33
A. Singh and A. K. Gupta. Two new heuristics for the equal piles problem. In B. Prasad, editor, *Proceedings of the 2nd Indian International Conference on Artificial Intelligence*, pages 3408–3423, Pune, India, 2005. IICAI.

[Smithline, 1995] cité p. 40
L. Smithline. Bandwidth of the complete k-ary tree. *Discrete Mathematics*, 142(1-3):203–212, 1995.

[Sourd et Schoenauer, 1998] cité p. 2, 41
F. Sourd and M. Schoenauer. Evolutionary mesh numbering: Preliminary results. In *Proceedings of the ACDM'98*, pages 137–150. Springer Verlag, 1998.

[Stadler, 1992] cité p. 1, 22, 56, 72, 105, 124
P. F. Stadler. Correlation in landscapes of combinatorial optimization problems. *Europhysics Letters*, 20:479–482, 1992.

[Stone, 1983] cité p. 27
L. D. Stone. The process of search planning: Current approaches and continuing problems. *Operations Research*, 31:207–233, 1983.

[Strenski et Kirkpatrick, 1991] cité p. 107
P. N. Strenski and S. Kirkpatrick. Analysis of finite length annealing schedules. *Algorithmica*, 6(1):346–366, 1991.

[Subramanian *et al.*, 1994] cité p. 1, 8
R. Subramanian, R. P. Sheff, J. D. Quillinan, D. S. Wiper, and R. E. Marsten. Coldstart: Fleet assignment at delta air lines. *Interfaces*, 24(1):104–120, 1994.

[Szymanski, 1985] cité p. 35
T. G. Szymanski. Dogleg channel routing is NP-Complete. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 4(1):31–41, 1985.

[Taillard, 1991] cité p. 16
E. Taillard. Robust taboo search for the quadratic assignment problem. *Parallel Computing*, 17(4-5):443–455, 1991.

[Tesauro, 1995] cité p. 31
G. Tesauro. Temporal difference learning and TD-gammon. *Communications of the ACM*, 38(3):58–68, 1995.

[Tomassini, 1995] cité p. 18
M. Tomassini. A survey of genetic algorithms. *Annual Reviews of Computational Physics*, III:87–118, 1995.

[Torres-Jimenez et Rodriguez-Tello, 2000] cité p. 33, 45, 60
J. Torres-Jimenez and E. Rodriguez-Tello. A new measure for the bandwidth minimization problem. *Lecture Notes in Artificial Intelligence*, 1952:477–486, 2000.

[Tsang et Voudouris, 1997] cité p. 28
E. Tsang and C. Voudouris. Fast local search and guided local search and their application to british telecom's workforce scheduling problem. *Operations Research Letters*, 20(3):119–127, 1997.

[Tsang et Wang, 1992] cité p. 23, 27
E. Tsang and C. Wang. *Neural Network Applications*, chapter Generic Neural Network Approach for Constraint Satisfaction Problems, pages 12–22. Springer-Verlag, 1992.

[Van Laarhoven et Aarts, 1988] cité p. 15
P. J. M. Van Laarhoven and E. H. L. Aarts. *Simulated Annealing: Theory and Applications*. Kluwer Academic Publishers, 1988.

[Varanelli et Cohoon, 1999] cité p. 15, 64, 107, 109, 110, 118, 123
J. M. Varanelli and J. P. Cohoon. A fast method for generalized starting temperature determination in homogeneous two-stage simulated annealing systems. *Computers & Operations Research*, 26(5):481–503, 1999.

[Vasquez et Hao, 2001a] cité p. 32
M. Vasquez and J. K. Hao. A heuristic approach for antenna positioning in cellular networks. *Journal of Heuristics*, 7(5):443–472, 2001.

[Vasquez et Hao, 2001b] cité p. 32
M. Vasquez and J. K. Hao. A hybrid approach for the multidimensional 0-1 knapsack problem. In *Proceedings of the 17th International Joint Conference on Arificial Intelligence*, pages 328–333, Seattle, WA, USA, 2001. Morgan Kaufmann Publishers.

[Verel *et al.*, 2006] cité p. 124
S. Verel, P. Collard, and M. Clergue. Neutralité dans les paysages de fitness. *Technique et Science Informatique*, 25(8-9):1023–1048, 2006.

[Voudouris et Tsang, 1995] cité p. 27
C. Voudouris and E. Tsang. Guided local search. Technical Report CSM-247, Department of Computer Science, University of Essex, 1995.

[Voudouris et Tsang, 1999] cité p. 27
C. Voudouris and E. Tsang. Guided local search and its application to the traveling salesman problem. *European Journal of Operational Research*, 113(2):469–499, 1999.

[Voudouris, 1997] cité p. 28
C. Voudouris. *Guided Local Search for Combinatorial Optimization Problems*. PhD thesis, Department of Computer Science, University of Essex, Colchester, UK, July 1997.

[Wang et Tsang, 1991] cité p. 27
C. Wang and E. Tsang. Solving constraint satisfaction problems using neural-networks. In *Proceedings of the 2nd IEEE International Conference on Artificial Neural Networks*, pages 295–299, Bournemouth, UK, 1991. IEEE Press.

[Webb, 1991] cité p. 1, 8
S. Webb. Optimization by simulated annealing of three-dimensional conformal treatment planning for radiation fields defined by a multilead collimator. *Physics in Medicine and Biology*, 36(9):1201–1226, 1991.

[White, 1984] cité p. 109
S. R. White. Concepts of scale in simulated annealing. In *Proceedings of the IEEE International Conference on Computer Design*, pages 646–651, Port Chester, NY, USA, 1984. IEEE Press.

[Whitley *et al.*, 1989] cité p. 99
D. Whitley, T. Starkweather, and D. Fuquay. Scheduling problems and traveling salesman: The genetic edge recombination operator. In *Proceedings of the 3rd International Conference on Genetic Algorithms*, pages 133–140, San Francisco, CA, USA, 1989. Morgan Kaufmann Publishers Inc.

[Wong *et al.*, 1988] cité p. 1, 8, 35
M. D. F. Wong, H. W. Leong, and C. L. Liu. *Simulated Annealing for VLSI Design*. Kluwer Academic Publishers, 1988.

[Wu et Wah, 2000] cité p. 24
Z. Wu and B. W. Wah. An efficient global-search strategy in discrete lagrangian methods for solving hard satisfiability problems. In *Proceedings of the 17th National Conference on Artificial Intelligence*, pages 310–315, Austin, TX, USA, 2000. MIT Press.

[Yagiura *et al.*, 1999] cité p. 24
M. Yagiura, T. Ibaraki, and F. Glover. An ejection chain approach for the generalized assignment problem. Technical Report 99013, Department of Applied Mathematics and Physics, Graduate School of Informatics, Kyoto University, 1999.

[Yagiura et Ibaraki, 1999] cité p. 2, 22
M. Yagiura and T. Ibaraki. Analyses on the 2 and 3-flip neighborhoods for the MAX-SAT. *Journal of Combinatorial Optimization*, 3(1):95–114, 1999.

[Yagiura et Ibaraki, 2001] cité p. 2, 22
M. Yagiura and T. Ibaraki. On metaheuristic algorithms for combinatorial optimization problems. *Systems and Computers in Japan*, 32(3):33–55, 2001.

[Zhang et Dietterich, 1995] cité p. 30
W. Zhang and T. G. Dietterich. A reinforcement learning approach to job-shop scheduling.

In *Proceedings of the 14th International Joint Conference on Artificial Intelligence*, pages 1114–1120, Montreal, Canada, 1995. Morgan-Kaufmann.

Liste des publications personnelles

Revues internationales

1. E. Rodriguez-Tello, J. K. Hao, and J. Torres-Jimenez. An effective two-stage simulated annealing algorithm for the minimum linear arrangement problem. *Computers & Operations Research*, Elsevier, accepted 8 janvier 2007, doi: 10.1016/j.cor.2007.03.-001.
2. E. Rodriguez-Tello, J. K. Hao, and J. Torres-Jimenez. An improved simulated annealing algorithm for bandwidth minimization. *European Journal of Operational Research*, Elsevier, accepted 8 décembre 2005, doi: 10.1016/j.ejor.2005.12.052.

Conférences internationales avec comité de sélection

1. E. Rodriguez-Tello, J. K. Hao, and J. Torres-Jimenez. A refined evaluation function for the MinLA problem. *Proceedings of the 5th Mexican International Conference on Artificial Intelligence (MICAI'06)*, volume 4293 of *Lecture Notes in Computer Science*, pages 392–403, Apizaco, México, November 2006. Springer. Third place, best paper award.
2. E. Rodriguez-Tello, J. K. Hao, and J. Torres-Jimenez. A comparison of memetic recombination operators for the MinLA problem. *Proceedings of the 4th Mexican International Conference on Artificial Intelligence (MICAI'05)*, volume 3789 of *Lecture Notes in Computer Science*, pages 613–622, Monterrey, México, November 2005. Springer.
3. E. Rodriguez-Tello, J. K. Hao, and J. Torres-Jimenez. Memetic algorithms for the MinLA problem. *Proceedings of the 7th International Conference on Artificial Evolution (AE'05)*, volume 3871 of *Lecture Notes in Computer Science*, pages 73–84, Lille, France, October 2005. Springer.
4. E. Rodriguez-Tello, J. K. Hao, and J. Torres-Jimenez. An improved evaluation function for the bandwidth minimization problem. *Proceedings of the 8th International Conference on Parallel Problem Solving from Nature (PPSN'04)*, volume 3242 of *Lecture Notes in Computer Science*, pages 650–659, Birmingham, UK, September 2004. Springer.
5. E. Rodriguez-Tello and J. Torres-Jimenez. Improving the performance of a genetic algorithm using a variable-reordering algorithm. *Proceedings of the GECCO 2004*,

volume 3103 of *Lecture Notes in Computer Science*, pages 102–113, Seattle, WA, USA, June 2004. Springer.

Conférences internationales avec actes de résumés étendus

1. E. Rodriguez-Tello, J. K. Hao, and J. Torres-Jimenez. A new evaluation function for the MinLA problem. *Proceedings of the 6th Metaheuristics International Conference (MIC'05)*, pages 796–801, Vienna, Austria, August 2005.

Conférences francophones avec actes de résumés étendus

1. E. Rodriguez-Tello, and J. K. Hao. Recherche tabou réactive pour le problème de l'arrangement linéaire minimum. *Actes du Sixième Congrès de la Société Française de Recherche Opérationnelle et d'Aide à la Décision (ROADEF'05)*, pages 314–315, Tours, France, Février 2005.
2. E. Rodriguez-Tello, J. K. Hao, and J. Torres-Jimenez. Une nouvelle mesure pour le problème de minimisation de largeur de bande. *Actes des Quatrièmes Journées Francophones de Recherche Opérationnelle (FRANCORO-IV)*, pages 82–83, Fribourg, Suisse, Août 2004.

Nouvelles Fonctions d'Évaluation pour les Problèmes d'Étiquetage de Graphes BMP et MinLA

Résumé

Cette thèse porte sur la conception de nouvelles fonctions d'évaluation ayant pour but d'améliorer la performance des algorithmes approchés conçus pour résoudre des problèmes d'optimisation combinatoire. En particulier, nous nous intéressons à l'amélioration de la résolution de deux problèmes d'étiquetage de graphes NP-difficiles : la *Minimisation de Largeur de Bande* (BMP) et l'*Arrangement Linéaire Minimum* (MinLA). Pour ce faire, deux nouvelles fonctions d'évaluation, notées respectivement δ et Φ, sont introduites. Contrairement aux fonctions classiques, elles incorporent des informations sur les particularités du problème afin d'améliorer leurs capacités de guidage. Des comparaisons expérimentales ont été effectuées pour évaluer l'efficacité des fonctions δ et Φ en utilisant divers algorithmes. Les résultats confirment que nos fonctions permettent d'améliorer considérablement les performances des algorithmes étudiés. Finalement, l'implémentation de deux algorithmes de Recuit Simulé, appelés RSA-δ et RSDP-Φ, a permis de tirer avantage des nouvelles fonctions proposées, mais aussi d'autres composants avancés. Les comparaisons expérimentales entre nos algorithmes et les heuristiques de référence, effectuées sur des instances d'essai issues de la littérature, montrent que RSA-δ et RSDP-Φ sont très compétitifs. En effet, ils permettent d'améliorer significativement les meilleurs résultats connus pour de nombreuses instances.

Mots-clés : fonction d'évaluation, algorithmes approchés, optimisation combinatoire, problèmes d'étiquetage de graphes.

New Evaluation Functions for the BMP and MinLA Graph Labeling Problems

Abstract

This thesis deals with the development of new evaluation functions aiming at improving the performance of heuristic algorithms developed for solving combinatorial optimisation problems. In particular, we are interested in the improvement of the results obtained for two NP-hard graph labeling problems: the *Bandwidth Minimization* (BMP) and the *Minimum Linear Arrangement* (MinLA). To accomplish it, two evaluation functions, noted respectively δ and Φ, are introduced. Contrary to the classical functions, they incorporate information on the problem's characteristics in order to improve their guidance capacities. Experimental comparisons were carried out by using different algorithms to assess the practical usefulness of the δ and Φ functions. The results confirm that our functions allow to considerably improve the performances of the studied algorithms. Finally, the implementation of two Simulated Annealing algorithms, called RSA-δ and RSDP-Φ, has made possible to take advantage of the new functions proposed, but also of other advanced components. The experimental comparisons between our algorithms and the state-of-the-art heuristics, carried out on benchmark instances from the literature, show that RSA-δ and RSDP-Φ are very competitive. Indeed, they allow us to significantly improve the best-known results for many benchmarks.

Keywords: evaluation function, heuristic algorithms, combinatorial optimisation, graph labeling problems.