Contrôle autonome d'opérateurs pour la recherche locale

Nadarajen Veerapen

▶ **To cite this version:**

Nadarajen Veerapen. Contrôle autonome d'opérateurs pour la recherche locale. Modélisation et simulation. Université d'Angers, 2012. Français. NNT : . tel-00995607

HAL Id: tel-00995607
https://theses.hal.science/tel-00995607

Submitted on 23 May 2014

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

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

l'université nantes angers le mans
PÔLE DE RECHERCHE ET D'ENSEIGNEMENT SUPÉRIEUR

université angers

Thèse de Doctorat

Nadarajen VEERAPEN

Mémoire présenté en vue de l'obtention du
grade de Docteur de l'Université d'Angers
sous le label de l'Université de Nantes Angers Le Mans

Discipline : Informatique
Spécialité : Informatique
Laboratoire : Laboratoire d'Étude et de Recherche en Informatique d'Angers (LERIA)

Soutenue le 29 novembre 2012

École doctorale : 503 (STIM)
Thèse n° : 1275

Contrôle autonome d'opérateurs pour la recherche locale

JURY

Président :	**M. Laurent GRANVILLIERS**, Professeur, Université de Nantes
Rapporteurs :	**M^me^ Clarisse DHAENENS**, Professeur, Université Lille 1 **M. Bertrand MAZURE**, Professeur, Université d'Artois
Examinatrice :	**M^me^ Gabriela OCHOA**, Senior Researcher, University of Sterling
Directeur de thèse :	**M. Frédéric SAUBION**, Professeur, Université d'Angers
Co-encadrants :	**M. Youssef HAMADI**, Senior Researcher, Microsoft Research Cambridge **M. Lucas BORDEAUX**, Research Engineer, Microsoft Research Cambridge

Remerciements

La thèse est bien évidemment une aventure scientifique. C'est également un parcours rempli de rencontres professionnelles et personnelles. Je souhaite commencer ce document en remerciant les personnes qui m'ont accompagné au fil de ces trois années.

Tout d'abord je souhaite remercier Frédéric Saubion, mon directeur de thèse, pour son soutien, son encadrement constructif, la confiance qu'il m'a accordée et, si je peux m'exprimer ainsi, sa force tranquille. Mes remerciements vont aussi à Youssef Hamadi et Lucas Bordeaux qui ont suivi ce travail de thèse avec bienveillance. Je remercie par la même occasion Microsoft Research qui a financé cette thèse.

Je remercie mes rapporteurs, Clarisse Dhaenens et Bertrand Mazure, pour l'intérêt qu'ils ont porté à mon travail et pour leurs commentaires qui m'ont permis d'améliorer ce document. Je remercie également Gabriela Ochoa et Laurent Granvilliers d'avoir accepté de participer au jury en tant qu'examinateurs.

Je tiens à remercier Jorge Maturana, avec qui j'ai eu l'occasion de travailler pendant un mois au Chili, ainsi que son épouse, Massiel, pour leur accueil et leur amitié.

Je souhaite remercier l'ensemble des membres du LERIA et du département d'informatique de l'université d'Angers. Merci aux doctorants et jeunes docteurs, en commençant par Fabien Chhel pour sa bonne humeur ainsi que pour la diversité et la qualité des conversations que j'ai pu avoir avec lui, Qinghua Wu et Yang Wang grâce à qui j'ai commencé à apprendre le mandarin,谢谢你们, Giacomo di Tollo, Sami Laroum, Daniel Porumbel, Benoit Da Mota, Lionel Chauvin, Aymeric Le Dorze, Caner Candan, Yan Jin, Una Benlic, Rongqiang Zeng, Stéphane Ngoma, Wassim Ayadi, Adila Bouabdallah, et une pensée spéciale pour Julien Robet. Merci aux enseignants-chercheurs et notamment Frédéric Lardeux, Adrien Goëffon, Matthieu Basseur et Claire Lefèvre pour les discussions que nous avons pu avoir. Merci aux secrétaires, Catherine Pawlonski, Christine Bardaine et Marie-Paule Gervaud, qui ont toujours su trouver le temps pour m'aider, avec le sourire. Merci aux ingénieurs et techniciens, Stéphane Vincendeau, Éric Girardeau, Frantz de Germain et Thierry Forest pour leur expertise et leur réactivité.

Merci à mes amis de Nantes et d'ailleurs, Thomas Vincent, Aurélia Mahé, Marie Pelleau et Aurelien Lejeune.

Merci aux personnes avec qui j'ai eu l'occasion de découvrir les coulisses du monde universitaire en tant que représentant des doctorants, notamment Loïc Chaumont et François Chapeau-Blondeau à l'école doctorale, Anne Brouard au CIES Centre, Émilie Bomal et Catherine Lefebvre au collège doctoral. Je tiens également à remercier Christian Bertheuil qui, dans le cadre du Nouveau Chapitre de Thèse, m'a amené à analyser la thèse en tant qu'expérience valorisable au delà de la recherche.

Je souhaite aussi remercier mes étudiants anonymes à qui j'ai beaucoup aimé enseigner et qui me surprennent chaque année.

Pour achever cette liste déjà longue, je veux remercier ma famille pour leur soutien : ma mère, Geneviève, mon père, Vijyand, et ma sœur, Ruby. Je pense également à mes grands-parents, Renée et Georges Guilloteau.

Table des matières

Introduction **1**

I État de l'art **5**

1 Optimisation combinatoire et recherche locale **7**
1.1 Introduction . . . 8
1.2 Problèmes d'optimisation et de satisfaction . . . 9
1.3 Complexité . . . 11
1.4 Heuristiques et métaheuristiques . . . 12
1.5 Recherche locale . . . 14
1.5.1 Voisinages de base . . . 15
1.5.2 Heuristiques de sélection . . . 16
1.5.3 Algorithmes de recherche locale . . . 17
1.6 Méthodes à population de solutions . . . 20
1.7 Conclusion . . . 20

2 Recherche autonome **23**
2.1 Introduction . . . 24
2.1.1 No Free Lunch . . . 24
2.1.2 Solveur . . . 25
2.1.3 Taxonomie et caractérisation . . . 26
2.2 Approches hors ligne . . . 27
2.2.1 REVAC . . . 28
2.2.2 Racing . . . 28
2.2.3 ParamILS . . . 29
2.3 Approches en ligne . . . 31
2.3.1 Recherche locale réactive . . . 32
2.3.2 Hyperheuristiques . . . 33
2.3.3 Sélection adaptative d'opérateurs . . . 35
2.4 Conclusion . . . 41

3 Recherche locale basée sur les contraintes **43**
3.1 Introduction . . . 44
3.2 Panorama des systèmes proposés . . . 44
3.2.1 Algorithmes . . . 44
3.2.2 Langages et bibliothèques . . . 44
3.2.3 Solveurs boîtes noires . . . 45
3.3 Comet . . . 45
3.3.1 Langage et concepts . . . 45
3.3.2 Contraintes globales . . . 48
3.3.3 Contraintes et violations . . . 49
3.3.4 Discussion . . . 51

3.3.5 Contributions et feedback 51
3.4 Conclusion 52

4 Problèmes **53**
4.1 One Max 54
4.2 Problème d'affectation quadratique 54
4.2.1 Représentations et structures de données 55
4.2.2 Jeux de données 55
4.2.3 Méthodes de résolution 56
4.3 Problème du voyageur de commerce 57
4.3.1 Représentations et structures de données 57
4.3.2 Mouvements pour la recherche locale 58
4.3.3 Jeux de données 59
4.3.4 Méthodes de résolution 59
4.4 Progressive party problem 60
4.4.1 Problème d'optimisation 60
4.4.2 Problème de satisfaction de contraintes 61
4.4.3 Méthodes de résolution 62
4.5 Conclusion 62

II Contributions **63**

5 Sélection d'opérateurs pour la LS **69**
5.1 Introduction 70
5.2 Cadre pour la recherche locale 71
5.2.1 Définitions de base 71
5.2.2 Structure opérationnelle 71
5.3 Encodage et mouvements 72
5.3.1 Voisinages binaires 72
5.3.2 Permutations 73
5.4 Contrôle pour la recherche locale 74
5.4.1 Métriques 74
5.4.2 Utilité et sélection 75
5.4.3 Mise en œuvre 77
5.5 Validation initiale 78
5.5.1 One Max 78
5.5.2 Discussion 78
5.6 Expérimentations sur les problèmes de permutations 79
5.6.1 Protocole expérimental 81
5.6.2 Résultats et discussion 81
5.7 Conclusion 87

6 Stratégies de sélection adaptative **89**
6.1 Introduction 90
6.2 Mesure de distance entre opérateurs 90
6.3 Compromis statique : expériences 91
6.3.1 Protocole expérimental 91
6.3.2 Résultats et discussion 93
6.4 Stratégie de contrôle du compromis 93
6.5 Compromis dynamique : expériences 95
6.5.1 Protocole expérimental 95
6.5.2 Résultats et discussion 96

6.6 Conclusion 100

7 Résolution de CSP **101**
7.1 Introduction 102
7.2 Contraintes 102
7.3 Application 104
7.3.1 Modèle PPP 104
7.3.2 Recherche tabou en COMET 105
7.3.3 Opérateurs de CSP 107
7.3.4 Protocole expérimental 108
7.3.5 Résultats et discussion pour la sélection proportionnelle . . . 109
7.3.6 Résultats et discussion pour les stratégies adaptatives 114
7.4 Conclusion 114

Conclusion générale **117**

Publications personnelles **119**

Références bibliographiques **121**

Introduction

Motivation et contexte

Cette thèse se situe dans le contexte de la recherche autonome qui vise à développer des méthodes d'optimisation demandant une intervention minime de la part de l'utilisateur. Cette thématique est étudiée au sein de l'équipe Métaheuristiques, Optimisation et Applications (MOA) du Laboratoire d'Étude et de Recherche en Informatique d'Angers (LERIA).

Au cours de ces dernières dizaines d'années, des progrès majeurs ont été accomplis dans la résolution de problèmes complexes d'optimisation combinatoire, souvent issus d'applications du monde réel, impliquant toujours plus de données et plus de contraintes. Afin de gérer des problèmes aux instances de grande taille et aux structures complexes, des techniques sophistiquées de résolution ont été développées et combinées pour créer des solveurs efficaces.

Les problèmes combinatoires sont souvent modélisés comme des problèmes de satisfaction de contraintes ou des problèmes d'optimisation. Ils consistent en un ensemble de variables, un ensemble de valeurs possibles pour ces variables et un ensemble de contraintes à satisfaire. L'objectif est alors de trouver une solution qui satisfasse toutes les contraintes ou une solution minimisant ou maximisant une fonction objectif donnée.

Les techniques de résolution complètes, issues du domaine de l'intelligence artificielle, basées sur des heuristiques d'exploration d'arbres, ont contribué au succès de la programmation par contraintes. Sur le plan des méthodes approchées, les métaheuristiques, qui sont des stratégies générales de recherche permettant d'explorer l'ensemble des configurations possibles d'un problème, ont été largement utilisées dans le champs de la recherche opérationnelle. L'intérêt des méthodes approchées est qu'elles permettent de traiter des problèmes de plus grande taille que les méthodes complètes. Ainsi les méthodes approchées n'explorent pas toutes les configurations possibles. Il est donc important de gérer un compromis entre exploration et exploitation dans l'échantillonage de l'ensemble des configurations.

Parmi ces métaheuristiques, les stratégies de recherche locale sont très populaires, notamment pour résoudre des problèmes contraints. Elles sont même intégrées et combinées à des techniques complètes de résolution. Plus récemment, des cadres génériques de programmation ont été proposés afin de faciliter la conception d'algorithmes de recherche locale basés sur les contraintes [Van Hentenryck et Michel, 2005].

La recherche locale repose sur le concept de voisinage. Étant donné un problème, l'espace de recherche, c'est-à-dire l'ensemble des configurations/solutions possibles du problème, est structuré par une relation de voisinage. Celle-ci associe à chaque élément de l'espace de recherche un ensemble d'autres éléments appelés voisins. Une fonction objectif est utilisée afin d'évaluer la qualité des voisins. Le but de la recherche est alors d'atteindre une bonne solution et, au mieux, un optimum

global. En partant d'une configuration initiale, un algorithme de recherche locale essaie d'atteindre cet optimum en se déplaçant localement d'une configuration à l'un de ses voisins en fonction de son évaluation. D'un point de vue plus abstrait, on peut aussi dire qu'une recherche locale applique donc un opérateur de mouvement à chaque itération.

Dans ce contexte, il est nécessaire de considérer plusieurs problèmes cruciaux. Une stratégie d'amélioration naïve sera rapidement prisonnière d'un optimum local. Des techniques spécifiques ont donc été développées afin d'échapper à de tels optima (redémarrages, recuit simulé, liste tabou, ...). La performance d'un algorithme de recherche locale est fortement corrélée à sa propension à correctement explorer et exploiter le paysage de recherche. Ce dernier est induit par la relation de voisinage et la fonction objectif. Par exemple, face à un paysage très rugueux, il faut être capable d'échapper aux nombreux optima locaux. Ces paysages ont été étudiés mais il est généralement difficile de prédire leur forme pour des problèmes quelconques.

Si l'on souhaite concevoir un algorithme de recherche locale sans ajouter de nombreux paramètres et sans combiner d'heuristiques très complexes, deux possibilités s'offrent à nous : améliorer la fonction objectif afin de modifier favorablement le paysage ou modifier la relation de voisinage dans la même optique.

En ce qui concerne les voisinages, de nombreux travaux ont exploré des extensions de voisinages classiques. Par exemple, de très grands voisinages peuvent être utilisés conjointement à des techniques efficaces d'exploration afin d'atteindre rapidement de bonnes configurations. Toutefois l'exploration de grands voisinages peut aussi avoir un impact négatif sur le processus de recherche et d'autres techniques prennent plutôt le parti de restreindre le voisinage. Par ailleurs, des algorithmes à voisinages variables ont été développés afin de tirer parti des propriétés de différents voisinages.

Toutefois, le gestion de voisinages multiples introduit des paramètres et des heuristiques spécifiques afin d'obtenir des résultats raisonnables. Ceci rend ardue l'utilisation de tels algorithmes par des non spécialistes sur des ensembles de problèmes variés. La difficulté principale vient du fait que cela demande souvent une connaissance approfondie du problème, de la spécificité des instances du problème et de la paramétrisation de l'algorithme. Le réglage des paramètres d'un algorithme se fait avant son exécution. Si ces paramètres sont modifiés en cours de l'exécution, on parle alors de contrôle.

La spécialisation et le réglage d'algorithmes de recherche locale est effectivement un problème plus général. Des méthodologies ont été proposées afin de faciliter la gestion ou le contrôle d'heuristiques de recherche, notamment les hyperheuristiques et la sélection adaptative d'opérateurs. [1] Cette thèse s'inscrit dans cette deuxième approche, et plus largement, dans le contexte de la recherche autonome.

L'objectif de cette thèse est de concevoir, au sein d'un solveur pour la recherche locale, un processus autonome de gestion des voisinages permettant de les contrôler au cours de la recherche. Cet aspect a déjà été exploré dans le cadre des algorithmes évolutionnaires.

Contributions

Cette thèse s'intéresse à l'étude de méthodes précédemment proposées pour la sélection d'opérateurs dans le contexte des algorithmes évolutionnaires et à leur adaptation à la recherche locale. Par la même occasion nous appliquons des notions plus souvent rencontrées dans le domaine évolutionnaire à la recherche locale. Ainsi le concept de population est mis en parallèle avec les solutions d'un chemin de recherche

1. Dans sa forme la plus simple, un opérateur est un voisinage couplé à un mécanisme de sélection d'un voisin dans ce voisinage.

locale. Ceci nous permet de proposer et de tester deux mesures de diversité. Un point clé est l'utilisation d'un compromis entre les performances en terme de qualité et de diversité afin de choisir les opérateurs de mouvement à appliquer à chaque itération.

Un aspect intéressant est que nous essayons de conserver un nombre restreint de paramètres. Nous montrons, par ailleurs, que lorsque nous introduisons des mécanismes plus complexes demandant davantage de paramètres, une paramétrisation déterminée pour un problème peut être transposée à un autre en conservant de bonnes performances. Nos travaux mettent aussi en évidence que, face à un ensemble d'opérateurs raisonnablement corrects, une sélection uniforme peut produire des résultats corrects.

Enfin nous démontrons qu'il est possible d'utiliser la sélection d'opérateurs pour résoudre avec succès un problème de satisfaction de contraintes en obtenant des performances similaires aux meilleurs résultats connus.

Plus généralement, notre approche s'intègre dans un solveur s'appuyant sur le langage Comet. Nous espérons ainsi qu'il sera, à terme, facilement adaptable et utilisable. Ce solveur est constitué de différents modules distincts. Ces différentes briques peuvent être modifiées indépendamment si l'utilisateur le souhaite, les opérateurs pouvant être générés idéalement automatiquement à partir du modèle du problème.

Organisation

Cette thèse est constituée de deux parties. La première présente le contexte et les concepts de base qui seront nécessaires par la suite.

Le Chapitre 1 s'attache à la présentation de l'optimisation combinatoire à travers ses définitions de bases, ses concepts et enjeux. Différentes méthodes de résolution sont également rappelées. Ce chapitre fixe le cadre général de la thèse en dressant un portrait assez large du domaine de l'optimisation. Nous nous concentrons par ailleurs sur les métaheuristiques basées sur la recherche locale et sur les concepts de cette dernière. En effet ces derniers constituent le socle sur lequel se baserons nos contributions. En présentant ces informations de façon synthétique, notre objectif est d'offrir une présentation directe et accessible à des lecteurs qui ne seraient pas des spécialistes du domaine. Ainsi nous rappelons le fonctionnement de diverses méthaheuristiques classiques. Celles-ci reposent sur des mécanismes souvent intuitifs, incluant de nombreux paramètres et heuristiques, et dont le comportement et le paramétrage peuvent être difficiles à appréhender. C'est ce qui motivera les approches évoquées au second chapitre.

Au Chapitre 2, nous nous employons à décrire la recherche autonome : son pourquoi et son comment. Nous dressons ainsi une présentation des taxonomies existantes et nous décrivons plusieurs méthodes représentatives. Ce chapitre présente des approches qui s'inscrivent dans une philosophie de création de techniques visant à améliorer et faciliter l'utilisation de méthodes d'optimisation. Les approches hors-ligne abordent ce problème en essayant d'optimiser le paramétrage initial d'un algorithme à partir d'expérimentations et, comme nous le montrons, ont rencontré un certains succès. Les approches en ligne, dont fait partie ce travail de thèse, tentent de modifier les paramètres internes d'un algorithme pendant son exécution. Dans le cas présent, lorsque l'on parle de paramètre, il ne faut pas avoir seulement en tête une valeur numérique mais également la configuration et l'ordre d'application des composants internes de l'algorithme, tels que des opérateurs de mouvement.

La recherche locale basée sur les contraintes est présentée au Chapitre 3. Nous y rappelons les méthodes disponibles et décrivons plus amplement l'environnement Comet que nous utilisons dans cette thèse. L'hybridation introduite par ce genre

de système nous rapproche de la programmation par contraintes et de son principale avantage à nos yeux : une modélisation déclarative du problème confié par la suite à un solveur fonctionnant sur le principe d'une boîte noire. Ayant été évoquée très brièvement dans le premier chapitre, la notion de fonction de violation est développée puisqu'elle est un des concepts fondamentaux de la résolution de problèmes contraints par des méthodes d'optimisation. Ces différentes notions seront importantes pour le dernier chapitres de nos contributions.

Pour terminer la première partie, le Chapitre 4 présente les différents problèmes que nous utiliserons pour tester et valider notre approche. Nous avons choisi des problèmes variés aussi bien dans leur modélisation que dans leur représentation et leur sémantique afin de pouvoir mettre en avant la généricité de notre approche. Parmi les quatre problèmes utilisés, un seul est un problème « jouet » mais nous l'utilisons pour ses propriétés théoriques connues. Les autres dérivent d'applications du monde réel.

La deuxième partie est consacrée aux contributions de cette thèse. En début de cette partie, le lecteur trouvera une introduction générale de la partie afin de lui permettre de mieux resituer et de mieux appréhender le lien entre l'état de l'art et les contributions ainsi que les motivations derrière ces dernières.

Le Chapitre 5 présente la sélection d'opérateurs, que nous avons introduite au Chapitre 2, dans le contexte de la recherche locale en considérant simultanément deux mesures de performance : une pour la qualité et une pour la diversité. Afin de démarrer sur des bases solides ce chapitre présente une formalisation des concepts nécessaires afin de pouvoir décrire et construire des opérateurs. L'emphase est mise sur la création d'un algorithme simple avec peu de paramètres mais pouvant fonctionner sur différents problèmes. Seuls des problèmes d'optimisation sont considérés. Les résultats obtenus montrent que notre approche générique fonctionne bien.

Le Chapitre 6 s'appuie sur le chapitre précédent afin de proposer un nouveau compromis entre les mesures de performance. Ce compromis s'adapte au fil de la recherche à travers une stratégie spécifique. Ici de nouveaux paramètres sont introduits mais l'objectif est que leur paramétrisation soit suffisamment robuste pour pouvoir continuer à fonctionner sur des problèmes aux sémantiques différentes. À nouveau nous ne considérons ici que des problèmes d'optimisation. Par contre, l'ensemble d'opérateurs utilisés est bien moins favorable que dans le chapitre précédent ce qui nous permet de mieux mettre en valeur, grâce aux résultats présentés, l'intérêt de notre approche de contrôle.

Le Chapitre 7 s'intéresse aux problèmes de satisfaction de contraintes et à l'application des méthodes développées aux chapitres précédents dans ce contexte. La transcription d'un problème de satisfaction de contraintes en problème d'optimisation demande de considérer les pénalités associées aux violations des contraintes en cherchant à les minimiser. Pour une meilleure compréhension du contexte de ce chapitre, le lecteur peut se référer au Chapitre 3 qui traite du paradigme de programmation dans lequel nous nous plaçons afin de résoudre ces problématiques. Ici nos résultats montrent que nous parvenons à atteindre les performances d'un algorithme dédié en terme de taux de résolution.

Enfin la conclusion clôt cette thèse en résumant les différents aspects évoqués dans le document et propose différentes perspectives de recherche.

Première partie

État de l'art

Chapitre 1

Optimisation combinatoire et recherche locale

Sommaire

1.1 Introduction **8**
1.2 Problèmes d'optimisation et de satisfaction **9**
1.3 Complexité **11**
1.4 Heuristiques et métaheuristiques **12**
1.5 Recherche locale **14**
1.5.1 Voisinages de base 15
1.5.2 Heuristiques de sélection 16
1.5.3 Algorithmes de recherche locale 17
1.6 Méthodes à population de solutions **20**
1.7 Conclusion **20**

1.1 Introduction

Les problèmes combinatoires sont présents dans un grand nombre de domaines dont la finance (par exemple la gestion de portefeuilles), la logistique (la planification de tournées de véhicules) ou même le domaine sportif (planification de rencontres).

À la frontière de l'informatique et des mathématiques appliquées, l'*optimisation combinatoire* [Papadimitriou et Steiglitz, 1982] consiste à résoudre des problèmes dont les variables sont discrètes. Il s'agit de trouver la meilleure ou les meilleures solutions parmi un ensemble fini, voire infini dénombrable, d'objets – un objet étant, par exemple, un entier, un ensemble, une permutation ou un graphe. La notion d'optimalité d'une solution est définie par une fonction objectif.

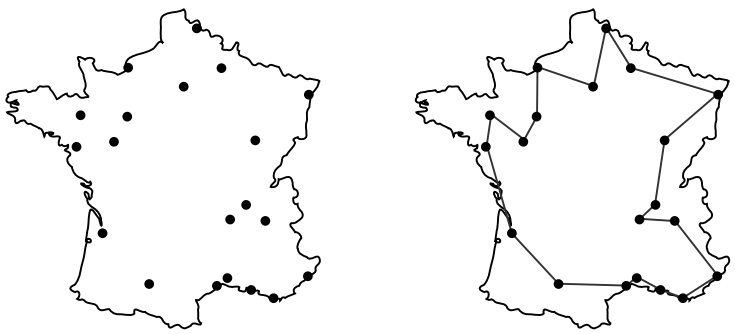
Un problème d'optimisation comprend généralement des contraintes qui rendent certaines solutions non réalisables, on parle alors d'optimisation sous contraintes.

Les problèmes de satisfaction de contraintes (*Constraint Satisfaction Problem*, CSP) ne demandent pas une solution optimale mais simplement une solution ne violant aucune contrainte. Dans la pratique il est possible de définir une fonction de violation des contraintes du problème qui pourra être utilisée comme fonction objectif afin de se ramener à un problème d'optimisation en cherchant à minimiser cette fonction (Chapitre 3).

Bien que l'espace de recherche de ces problèmes soit généralement fini, l'énumération exhaustive des solutions du problème est souvent impossible en un temps raisonnable. Bien entendu, juger de ce qu'est un temps d'exécution raisonnable dépend du contexte. Ainsi la durée accordée pour la planification de l'implantation de lieux de production pourra être de l'ordre du jour ou de la semaine, alors que la durée pour définir chaque matin le parcours d'un véhicule de livraison en fonction de demandes évoluant quotidiennement sera plutôt de l'ordre de la minute.

Évidemment, l'énumération exhaustive est l'approche la plus simple et la plus naïve mais elle n'est pas utilisée en pratique. En effet, et ce même sur des problèmes de taille très modeste, cette évaluation peut vite prendre un temps considérable. À titre d'exemple (Figure 1.1), considérons le problème du *voyageur de commerce* (Section 4.3) qui consiste à trouver le plus petit cycle hamiltonien dans un graphe non-orienté à n sommets (chaque sommet du graphe est présent une seule fois dans le cycle) : il y a $(n-1)!/2$ cycles possibles. En supposant une fréquence d'énumération de 1 GHz, énumérer tous les cycles d'un graphe ayant 16 sommets prendrait environ 10 minutes et pour 26 sommets plus de 245 millions d'années.

Certains problèmes combinatoires peuvent être résolus de manière exacte en temps polynomial, c'est-à-dire que le temps de calcul est borné par un polynôme de la taille des données. Toutefois, pour un grand nombre de cas, ces problèmes font



(a) Les points représentent les plus grandes villes de France.

(b) Le tracé reliant les villes correspond au circuit le plus court.

FIGURE 1.1 – Illustration du problème du voyageur de commerce.

partie de la classe des problèmes NP-difficiles. Ceci revient à dire que le problème est au moins aussi dur que tous les problèmes dans NP, [1] une classe de problèmes d'optimisation dont on conjecture qu'il n'existe pas d'algorithme déterministe en temps polynomial pour les résoudre (Section 1.3).

Il existe un certain nombre de méthodes exactes permettant d'explorer efficacement l'espace de recherche en éliminant de l'exploration des pans entiers de cet espace. Ces méthodes exactes emploient une approche arborescente en explorant les différents points de choix. Néanmoins, pour de nombreux problèmes, il est nécessaire de recourir à des méthodes approchées lorsqu'aucune méthode exacte ne peut résoudre le problème en un temps raisonnable. Comme leur nom l'indique, les méthodes approchées n'offrent pas la garantie de trouver la ou les solutions optimales.

Il est intéressant de noter qu'il est tout-à-fait possible de combiner une méthode approchée à une méthode exacte afin de trouver la ou les solutions optimales. Cette hybridation peut se faire de deux façons. La première consiste à intégrer une méthode exacte à une méthode approchée afin d'améliorer l'efficacité, en terme de qualité des solutions trouvées, de l'algorithme hybride [Blum et Mastrolilli, 2007; Dumitrescu et Stützle, 2010]. Dans ce cas la méthode approchée peut faire appel à la méthode exacte pour résoudre un sous-problème traitable en un temps raisonnable. Toutefois les résultats obtenus par l'algorithme hybride ne seront pas exacts mais approchés.

La seconde approche agit inversement, la méthode approchée est intégrée la méthode exacte [Puchinger *et al.*, 2010]. Le méthode approchée est utilisée pour trouver des solutions suffisamment bonnes permettant de borner le nombre de configurations de solutions à évaluer et ainsi améliorer la performance en temps et en espace de l'algorithme hybride par rapport à l'algorithme exact seul. Les solutions obtenues sont garanties optimales.

Notons que l'hybridation peut aussi se faire entre différentes méthodes approchées [Raidl, 2006].

1.2 Problèmes d'optimisation et de satisfaction

Nous présentons ici quelques définitions de base afin de formaliser le contexte dans lequel se place ce travail.

Définition 1.2.1. Un *problème de satisfaction de contraintes* (*Constraint Satisfaction Problem*, CSP) est défini par un triplet $(X, \mathcal{D}, \mathcal{C})$ où $X = \{x_1, \ldots, x_m\}$ est un ensemble fini de variables, $\mathcal{D} = \{\mathcal{D}_{x_1}, \ldots, \mathcal{D}_{x_m}\}$ l'ensemble de domaines associés et $\mathcal{C} = \{\mathcal{C}_1, \ldots, \mathcal{C}_p\}$ un ensemble fini de p contraintes.

Définition 1.2.2. Chaque *contrainte* $\mathcal{C}_i$ est une paire (s_i, R_i) où $s_i = (x_{i_1}, \ldots, x_{i_n})$ est un n-uplet de variables et R_i est une relation n-aire sur $\mathcal{D}$ c'est-à-dire le sous-ensemble de valeurs possibles des variables représentant les combinaisons simultanées de valeurs des variables de s_i.

Définition 1.2.3. Une *affectation* est une fonction des variables vers les domaines, $v : X \to \mathcal{D}$. Une affectation *satisfait* une contrainte (s_i, R_i) si $(v(x_{i_1}), \ldots, v(x_{i_n})) \in R_i$. Dans un CSP une *solution* est une affectation satisfaisant toutes les contraintes.

Définition 1.2.4. Un *problème d'optimisation* $P = (\mathcal{S}, f)$ est défini par un ensemble $\mathcal{S}$ et une *fonction objectif* $f : \mathcal{S} \to \mathbb{R}$.

Définition 1.2.5. Tout élément $s \in \mathcal{S}$ est appelée une *solution*.

1. La classe NP correspondant aux problèmes dont on peut tester la validité d'une solution en temps polynomial.

Définition 1.2.6. L'ensemble $\mathcal{S}$ est appelé *ensemble des solutions réalisables*, *ensemble des solutions candidates*, *espace de recherche* ou *espace des solutions*. Il peut être lui-même défini par un triplet $(X, \mathcal{D}, \mathcal{C})$ comme pour un CSP.

Notons ici qu'un CSP est en fait un problème d'optimisation sans fonction objectif. En général, il est possible de quantifier, plus ou moins finement, le degré de violations des contraintes du problème par l'intermédiaire de fonctions de violations. Ainsi l'agrégation de ces fonctions permet d'obtenir une fonction que l'on pourra chercher à minimiser. Ceci permet de ramener un problème de satisfaction de contraintes à un problème d'optimisation.

Notons, par ailleurs, que nous considérons ici un problème de minimisation sans perte de généralité car un problème de maximisation revient à minimiser $-f$.

Définition 1.2.7. *Résoudre* un problème d'optimisation revient alors à trouver une solution optimale $s^* \in \mathcal{S}$ pour laquelle la fonction objectif aura une valeur minimale, c'est-à-dire $f(s^*) \leq f(s), \forall s \in \mathcal{S}$. La solution s^* est appelée un *optimum global* de P.

Si les domaines des variables sont discrets (resp. continus), on parlera d'optimisation combinatoire ou discrète (resp. continue).

Considérons le petit problème d'optimisation suivant (Figure 1.2) :

$$\min f(x) = \sin(x/3) + \cos(x/\sqrt{3}) + 2, \forall x \in \mathbb{Z}, 0 \leq x \leq 30 \tag{1.1}$$

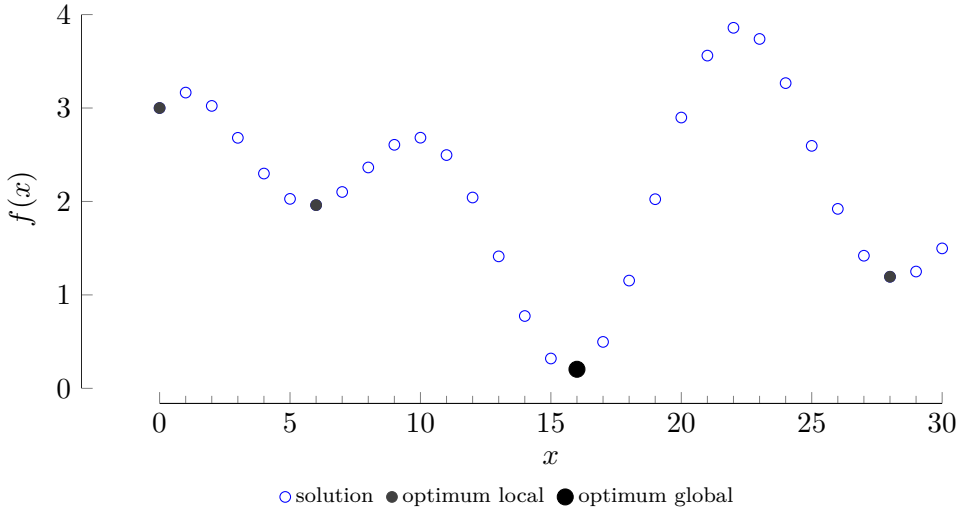


FIGURE 1.2 – Ensemble des solutions d'un problème d'optimisation (minimisation) simple illustrant les optima locaux et l'optimum global.

Définition 1.2.8. Une *relation de voisinage* est une relation binaire irréflexive $\mathcal{N} \subseteq \mathcal{S}^2$ sur l'espace de recherche. Dans la majorité des cas, la relation est aussi symétrique. Cette relation représente l'adjacence d'une solution par rapport à une autre, autrement dit, s'il est possible à partir de la première d'obtenir la seconde après application d'une opération donnée.

Définition 1.2.9. Le *voisinage*, $\mathcal{V} : \mathcal{S} \to 2^{\mathcal{S}}$, [2] d'une solution est alors donné par $\mathcal{V}(s) = \{s' | s' \in \mathcal{S}, \mathcal{N}(s, s')\}$. La taille d'un voisinage correspond au nombre de solutions qu'il contient, $|\mathcal{V}(s)|$.

2. $2^{\mathcal{S}}$ représente l'ensemble des parties de $\mathcal{S}$.

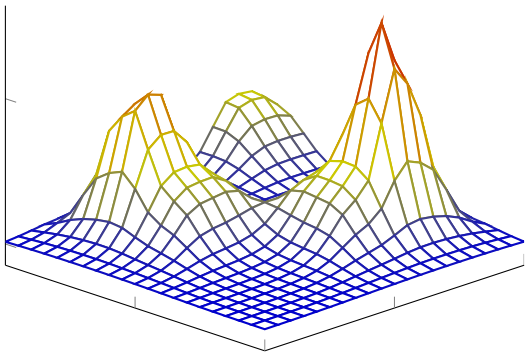


FIGURE 1.3 – Exemple d'un paysage de recherche pour une fonction $f : \mathbb{Z}^2 \to \mathbb{R}$ permettant de visualiser la nature géométrique du paysage. Les sommets sont des maximums locaux, des optima locaux si l'on considère un problème de maximisation et le plus haut sommet un optimum global. Les surfaces plates, notamment à l'avant plan, sont des plateaux.

Définition 1.2.10. Un *optimum local* est une solution pour laquelle le voisinage ne contient aucune solution candidate dont la valeur de la fonction objectif est meilleure. Dans un contexte de minimisation, une solution $s^* \in \mathcal{S}$ est un minimum local si $\forall s \in \mathcal{V}(s^*), f(s^*) \leq f(s)$. Si plusieurs solutions voisines ont la même valeur de fonction objectif, cet ensemble de solutions est appelé un *plateau*.

Définition 1.2.11. Un *paysage de recherche* est un triplet $(\mathcal{S}, \mathcal{V}, f)$.

Le paysage de recherche correspond à la structure dans laquelle s'effectue la résolution du problème. On peut faire le parallèle avec une structure topographique, ce qui amène à facilement employer des expressions liées aux déplacements que l'on peut y effectuer.

1.3 Complexité

Avant d'aborder les méthodes de résolution, intéressons nous à la complexité des algorithmes et aux classes de complexité des problèmes.

En général, un bon algorithme est rapide et trouve la meilleure solution. Le temps d'exécution d'un algorithme, et en l'occurrence d'un programme, dépendra bien sûr de la machine sur laquelle il est exécuté. Toutefois si l'on compte le nombre d'étapes, ou de pas, d'un algorithme, il est possible de faire abstraction de telles contraintes environnementales. La fonction de complexité en temps, $T(n)$, d'un algorithme exprime ainsi le temps maximum, au sens du nombre de pas, nécessaire pour résoudre une instance de taille n d'un certain problème.

Notons que la complexité d'un algorithme peut aussi faire référence à l'espace en mémoire nécessaire lors de l'exécution de l'algorithme. Ceci dit, dans cette section nous n'aborderons que la complexité en temps : nous nous permettons donc un abus de langage en utilisant le terme complexité au lieu de complexité en temps.

Définition 1.3.1. Un algorithme de complexité *polynomiale* aura une fonction de complexité $T(n)$ pouvant être exprimée par un polynôme p. Un algorithme dont la fonction $T(n)$ ne peut être exprimée par un polynôme est dite de complexité *exponentielle*.

Beaucoup d'algorithmes de complexité exponentielle sont « naïfs » et se basent sur l'énumération exhaustive des solutions potentielles. Une connaissance plus en

profondeur du problème peut alors permettre de trouver un algorithme de complexité polynomiale. Toutefois certains problèmes sont si compliqués qu'aucun algorithme de complexité polynomiale ne peut les résoudre. Un tel problème est dit *difficilement soluble* ou *insoluble*.

En théorie de la complexité [Garey et Johnson, 1990], la classe de problèmes P comprend tous les problèmes de décision solubles en temps polynomial sur une machine déterministe.

Définition 1.3.2. Un problème est dit de *décision* si sa solution est « oui » ou « non ».

Par exemple, « existe-t-il un chemin entre deux points A et B ? » est un problème de décision.

La classe de problèmes NP (Non déterministes Polynomiaux) regroupe les problèmes de décision solubles en temps polynomial sur une machine non déterministe.

Cela revient à dire que c'est la classe de problèmes pour lesquels il existe un algorithme polynomial permettant de tester la validité d'une solution du problème. Notons que N $\subseteq$ NP. La réciproque N $\supseteq$ NP n'est pas démontrée, et donc N = NP, reste un grand problème ouvert.

Définition 1.3.3. Un problème de décision C est *NP-complet* si

1. C est dans NP et
2. si tous les problèmes dans NP se réduisent polynomialement à C.

Ainsi il est possible de vérifier rapidement une solution d'un problème NP-complet, mais la caractéristique notable d'un problème NP-complet est qu'on ne sait pas trouver de solution efficacement. Ceci vaut cependant pour le pire des cas. Certaines sous-classes des instances d'un problème peuvent être plus faciles. Des algorithmes exacts, notamment de *séparation et d'évaluation* (*branch and bound*) peuvent donner des résultats en un temps raisonnable pour des instances de taille conséquente. Toutefois, dans un grand nombre de cas, il est nécessaire de faire appel à des méthodes approchées telles que les heuristiques ou les métaheuristiques (Section 1.4).

Jusqu'à présent nous n'avons abordé que les problèmes de décisions or, en optimisation, l'on cherche générale à trouver une solution qui soit meilleure que les autres. On parle alors de problèmes *difficiles*.

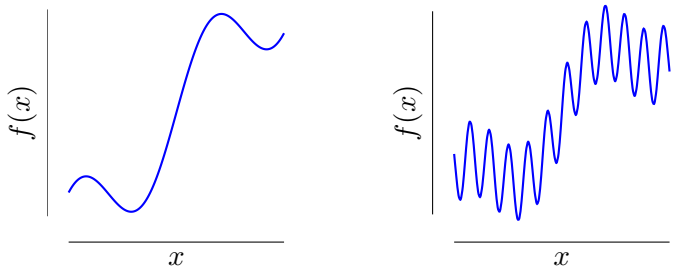
Définition 1.3.4. Un problème est *NP-difficile* s'il satisfait la condition deux de la NP-complétude (ci-dessus) ou, informellement, s'il est au moins aussi difficile que tous les problèmes dans NP.

Les problèmes NP-difficiles peuvent être de n'importe quel type (d'optimisation ou de décision). À un problème d'optimisation NP-difficile est associé un problème NP-complet, « existe-t-il une solution de coût inférieur à une borne B ? » dans le cas d'un problème de minimisation.

Maintenant que la notion de difficulté des problèmes a été présentée, nous pouvons nous intéresser aux méthodes de résolutions qui, en pratique, permettent de résoudre ce genre de problème. Dans la suite nous abordons les métaheuristiques qui sont des méthodes approchées.

1.4 Heuristiques et métaheuristiques

Une *heuristique* est une méthode de résolution, spécifique à un problème particulier, sans garantie formelle de qualité mais généralement beaucoup plus rapide qu'une méthode exacte. Pour citer [Papadimitriou et Steiglitz, 1982] : « However



(a) Un paysage lisse. (b) Un paysage rugueux.

FIGURE 1.4 – Deux paysages ayant une rugosité différente dans un espace de recherche d'une dimension (une variable).

unsatisfying mathematically, such approaches are certainly valid in practical situations. »

Parmi les heuristiques, notons entre autres les *heuristiques de construction* qui permettent de générer des solutions réalisables. Par exemple, revenons au problème du voyageur de commerce évoqué dans l'introduction de cette section. Une heuristique de construction gloutonne partira d'un sommet aléatoirement choisi, puis suivra itérativement l'arête de poids minimum vers un sommet non encore visité. Lorsqu'il ne reste plus de sommets, l'arête entre le premier et le dernier sommet est ajoutée au chemin pour faire un cycle.

Une *métaheuristique* [Glover, 1986; Blum et Roli, 2003] est un algorithme d'optimisation qui se veut moins spécifique au problème et à sa structure qu'une simple heuristique et qui vise à explorer plus ou moins efficacement l'espace de recherche. Les métaheuristiques peuvent être classées en deux grandes familles : les méthodes de recherche locale et celles à population de solutions. Les méthodes de recherche locale [Hoos et Stützle, 2004] sont basées sur la notion parcours ou de trajectoire, où une unique solution est modifiée itérativement (par exemple, le recuit simulé, Section 1.5.3). Les méthodes à population de solutions sont basées, comme leur nom l'indique, sur la notion de population où un ensemble de solutions est manipulé à chaque itération (par exemple, les algorithmes évolutionnaires ou de colonie de fourmis, Section 1.6).

En tant que méthode approchée, une bonne métaheuristique se doit de conjuguer astucieusement deux objectifs généralement divergents lors de l'exploration de l'espace de recherche : l'intensification (la faculté d'obtenir des solutions de qualité croissante) et la diversification (la capacité à échantillonner/explorer efficacement l'espace de recherche afin de visiter des solutions variées). Un algorithme uniquement basé sur l'intensification (comme la descente, Section 1.5.3) convergera vers un optimum local ; la diversification est importante afin de trouver de multiples optima locaux. L'un d'entre eux peut se révéler être, au mieux, l'optimum global. Au pire, celui trouvé initialement demeure le meilleur.

L'analogie topographique du concept de paysage de recherche amène à utiliser le terme *rugosité* [Weinberger, 1990] pour définir la nature de l'espace de recherche lorsque l'on considère la position de chaque solution par rapport à sa valeur d'évaluation. S'il n'y a pas de changements brusques entre des solutions proches (corrélation distance-qualité), on parle de paysage lisse. Si, au contraire, le paysage est très bruité, on parle de paysage rugueux (Figure 1.4). Selon la nature du paysage, le niveau et la nature de la diversification ne sont pas les mêmes pour s'éloigner d'un optimum local déjà rencontré.

Dans la section suivante, nous nous intéressons aux méthodes permettant d'explorer un espace de recherche en passant d'une solution à l'autre.

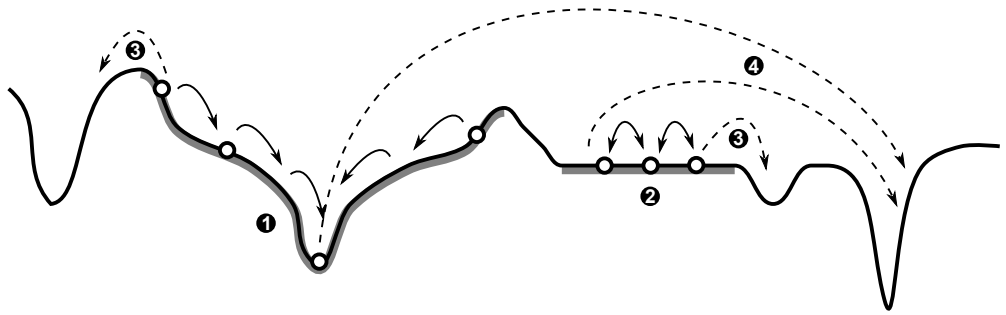


FIGURE 1.5 – Cette illustration montre un exemple de paysage de recherche dans lequel pourrait se déplacer un algorithme de recherche locale. La partie (1) est un bassin d'attraction. Appliquer une recherche locale strictement améliorante conduit à chaque fois vers le même minimum local. En (2) nous avons un exemple de plateau. Il est possible de « tourner en rond » et de revenir à des solutions précédemment visitées puisque les solutions du plateau ont la même valeur. Les mouvements (3) sont de petits mouvements de perturbations permettant d'échapper à un bassin d'attraction ou à un plateau. Les mouvements (4) permettent de vraiment changer de région, par exemple lorsqu'une nouvelle solution aléatoire est utilisée.

1.5 Recherche locale

La *recherche locale* (*local search*, LS) est un concept qui se fonde sur la notion de voisinage (l'ensemble des solutions atteignables à partir d'une solution en lui appliquant une opération que l'on appellera mouvement ou opérateur). Comme les algorithmes de parcours ne considèrent qu'une solution, la notion de diversification est encore plus critique que pour les algorithmes qui considèrent des populations de solutions. En général, elle est réalisée soit par des mécanismes de prohibition (comme dans la recherche tabou, Section 1.5.3), soit en ponctuant les phases d'intensification par des phases de perturbation ou de réinitialisation complète de la solution déterminées de façon déterministe ou stochastique (par exemple la recherche locale itérée, Section 1.5.3).

L'objectif de la diversification est de permettre à la recherche de sortir d'un bassin d'attraction. Ce dernier correspond à une région du paysage de recherche dans laquelle tout effort d'amélioration d'une solution ramènera vers le même optimum local. Suivant la stratégie de la recherche, il peut être préférable que la nouvelle solution examinée ne diffère pas trop de la meilleure solution obtenue si l'on suppose que les différents optima sont proches (paysage rugueux) ou partages certaines caractéristiques. Toutefois si les optima sont très différents ou si la recherche est sur un plateau (paysage lisse), il est en général intéressant d'avoir une nouvelle solution très différente, voire complètement nouvelle. La Figure 1.5 tente d'illustrer quelques-uns de ces concepts.

Les algorithmes de recherche locales partagent un squelette commun. C'est un processus itératif dans lequel on part d'une solution dont on explore le voisinage afin de trouver une nouvelle solution. Celle-ci est traitée d'une façon spécifique à l'algorithme considéré puis le processus recommence. La Figure 1.6 présente ce schéma conceptuel.

Dans le reste de cette section, la fonction `recherchelocale(`$\mathcal{V}$`,`f`,`c`,`x`)` retournera une solution sélectionnée selon le critère c, dépendant éventuellement de la fonction objectif f, parmi les voisins de x dans le voisinage $\mathcal{V}$. Cette fonction correspond au composant de recherche locale du schéma conceptuel.

Dans ce qui suit nous donnons quelques exemples de voisinages et d'heuristiques de sélection. Nous présentons ensuite un panorama d'algorithmes de recherche locale

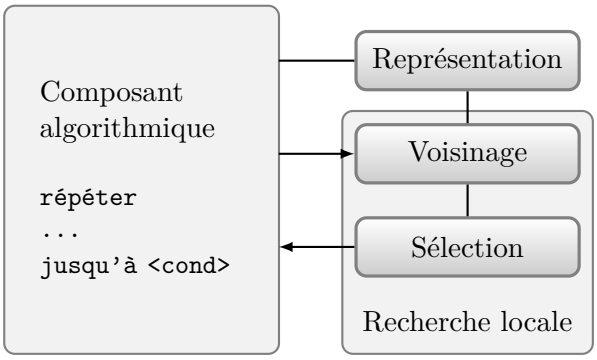
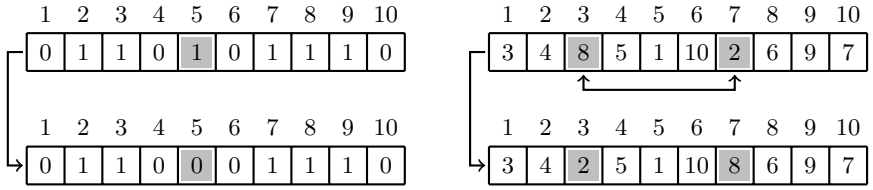


FIGURE 1.6 – Schéma conceptuel d'un algorithme de recherche locale – le composant algorithmique représente le comportement spécifique de la méthode de recherche locale, le composant recherche locale est commun à toutes les méthodes de recherche locale et la représentation est spécifique au problème traité. Le composant algorithmique comprend une boucle dans laquelle un appel à la recherche locale est effectué. Cet appel donne une solution à la recherche locale afin d'en explorer le voisinage. La sélection choisit une solution du voisinage qu'elle retourne au composant algorithmique.



(a) Flip (b) 2-échange

FIGURE 1.7 – Examples de mouvements simples pour la recherche locale.

afin d'illustrer les différentes techniques employées.

1.5.1 Voisinages de base

En recherche locale, les voisinages les plus simples sont généralement induits par des mouvements ou opérations qui modifient une ou deux variables (Figure 1.7). Les représentations choisies influencent bien sûr les voisinages utilisables, ne serait-ce que par les types de variables utilisées. Les représentations les plus communes sont des tableaux de variables binaires ou entières. Selon les problèmes traités, des structures plus compliquées à mettre en œuvres peuvent être conçues, en général par soucis de performance. Nous ne considérons ici que des mouvements simples qui peuvent éventuellement être combinés afin de créer des mouvements plus complexes.

Flip

Le mouvement *flip* inverse la valeur d'une variable booléenne. Ainsi 0 devient 1 et, alternativement, 1 devient 0.

Nouvelle valeur

Ce mouvement remplace la valeur d'une variable par une autre valeur du domaine de la variable. Dans sa forme la plus simple, dans un domaine binaire, il correspond au *flip*.

2-échange

Le mouvement *2-échange* échange les valeurs de deux variables. Ce mouvement fonctionne bien avec les permutations puisqu'il permet de conserver toutes les valeurs distinctes de la permutation. Naturellement, il est possible d'envisager des échanges entre un nombre plus important de variables, ce qui entraîne des voisinages de plus grandes tailles.

1.5.2 Heuristiques de sélection

Les heuristiques, que nous qualifierons *de sélection*, se concentrent sur le choix d'un voisin parmi le voisinage. Remarquons que lorsque la sélection intègre un aspect stochastique, cela contribue à plus de diversité.

Meilleur voisin

L'heuristique du meilleur voisin choisit toujours la solution du voisinage dont la valeur est la meilleure.

Premier voisin améliorant

Cette variante de l'heuristique précédente retourne le premier voisin qui améliore la solution courante. Cette heuristique demande d'avoir un ordre parmi les voisins.

Amélioration aléatoire

Cette heuristique choisi un voisin aléatoirement et l'accepte s'il est meilleur que la solution courante.

Heuristique de Metropolis

Variante de l'amélioration aléatoire, l'heuristique de Metropolis autorise les mouvements qui dégradent la valeur objectif de la solution. Le choix du voisin x se fait de façon aléatoire et il est accepté s'il améliore la solution courante x^*. Toutefois, s'il dégrade la solution, le voisin est accepté avec une faible probabilité

$$exp\left(\frac{-f(x) - f(x^*)}{t}\right)$$

où t est un paramètre appelé *température*. L'idée originale d'accepter une valeur avec une certaine probabilité vient de [Metropolis *et al.*, 1953] dans le contexte de la physique statistique.

Tabou

La sélection tabou (voir la recherche tabou, Section 1.3) combine une sélection de voisin qui améliore la solution actuelle et une mémoire des solutions déjà rencontrées ou, en général, des mouvements effectués récemment pour des raisons de stockage mémoire. La présence d'un élément en mémoire le rend tabou, interdit. En général une exception est faite s'il permet d'atteindre une solution meilleure que toutes celles trouvées précédemment : c'est *l'aspiration*. Bien entendu l'aspiration ne s'applique qu'aux mouvements.

K-variantes

Il peut être intéressant d'introduire un peu d'aléatoire dans un choix déterministe, comme pour le meilleur voisin. Il s'agit de déterminer les K premiers voisins correspondant au critère de sélection, dans le cas du meilleur voisin on choisira les K meilleurs voisins, et de sélectionner aléatoirement un voisin parmi ces K éléments.

1.5.3 Algorithmes de recherche locale

Nous présentons ici quelques algorithmes de recherche locale qui comptent parmi les plus connus. La première, la descente simple, peut être considérée comme une heuristique. Les autres, intégrant des mécanismes de diversification, sont des métaheuristiques.

Descente

Le schéma de base de la recherche locale est la *descente simple* (Algorithme 1.1). L'heuristique s'arrête une fois qu'un optimum local est atteint. L'appellation *descente* provient du fait que l'on se place implicitement dans un contexte de minimisation d'une fonction objectif. On cherche donc la plus petite valeur de cette fonction. Ceci rend naturel le parallèle avec un déplacement vers le bas. On notera que le terme anglophone *hill-climbing* se réfère quant à lui à un contexte de maximisation.

Algorithme 1.1: Algorithme d'une descente simple.

Entrées : $\mathcal{V}$ un voisinage, f une fonction objectif, x une solution initiale
$x^* \leftarrow x$;
répéter
  $x \leftarrow$ `recherchelocale(`$\mathcal{V}$`,`f`,selection,`x`)`;
  si $f(x) < f(x^*)$ **alors** $x^* \leftarrow x$
jusqu'à $f(x) \geq f(x^*)$;

Ce schéma peut être amélioré de façon simple en utilisant la *descente multiple* qui consiste à répéter l'opération de descente un certain nombre de fois en changeant de solution initiale à chaque fois et en conservant la meilleure solution obtenue.

Recuit simulé

Le recuit simulé [Kirkpatrick *et al.*, 1983] a été inspiré par un procédé métallurgique qui implique de chauffer et de refroidir un matériau afin d'augmenter la taille de ses cristaux et d'en réduire les défauts. Par analogie, la métaheuristique, par ailleurs basée sur l'heuristique de Metropolis, accepte une nouvelle solution avec une certaine probabilité fonction d'une température t, laquelle diminue graduellement au cours de l'exécution (Algorithme 1.2).

Recherche tabou

La recherche tabou a été initialement proposée par [Glover et McMillan, 1986]. Cette métaheuristique cherche à éviter de stagner dans des optima locaux en utilisant une mémoire à court terme. L'objectif de cette mémoire est d'empêcher la recherche de revenir sur ses pas. Pour ce faire, les mouvements produisant l'effet inverse des mouvements récents sont rendus tabous, c'est-à-dire qu'ils ne sont pas autorisés pendant un certain laps de temps. La version la plus simple de la recherche tabou est présentée dans l'Algorithme 1.3. La longueur de la liste tabou est un paramètre

Algorithme 1.2: Algorithme de recuit simulé.

Entrées : $\mathcal{V}$ un voisinage, f une fonction objectif, x une solution initiale
Soit t_1 la température initiale;
$k \leftarrow 1$; $x^* \leftarrow x$;
répéter
  $x \leftarrow$ `recherchelocale(`$\mathcal{V}$`,`f`,metropolis(`t_k`),`x`);`
  si $f(x) < f(x^*)$ **alors** $x^* \leftarrow x$;
  $t_{k+1} \leftarrow$`mise-a-jour-temperature(`x`,`t_k`);`
  $k \leftarrow k + 1$;
jusqu'à *la condition de fin*;

crucial. Si elle est trop petite, la recherche risque de « tourner en rond ». Si elle est trop grande, le risque est de ne plus avoir suffisamment de mouvements autorisés.

Algorithme 1.3: Algorithme d'une recherche tabou simple.

Entrées : $\mathcal{V}$ un voisinage, f une fonction objectif, x une solution initiale
Soit T la liste tabou ;
$T \leftarrow \emptyset$; $x^* \leftarrow x$;
répéter
  $x \leftarrow$ `recherchelocale(`$\mathcal{V}$`,`f`,selection,`x`);`
  si $f(x) < f(x^*)$ **alors** $x^* \leftarrow x$;
  Ajouter le mouvement à T et supprimer le plus ancien si nécessaire;
jusqu'à *la condition de fin*;

Recherche locale itérée

La recherche locale itérée [Stützle, 1998] applique, comme son nom l'indique, une succession d'appels à une recherche locale. La solution de départ de chaque appel est une solution plus ou moins proche de la solution obtenue à l'appel précédent (Algorithme 1.4). Ce mécanisme de perturbation est censé permettre à l'algorithme d'échapper aux bassins d'attractions. Il convient ainsi que choisir judicieusement le mécanisme et le voisinage permettant d'obtenir la perturbation. Si cette perturbation n'est pas assez grande, la recherche retombera dans le bassin qu'elle avait quitté. Si elle est trop importante alors cela revient à faire un redémarrage aléatoire.

Algorithme 1.4: Algorithme d'une recherche locale itérée.

Entrées : $\mathcal{V}_d$ et $\mathcal{V}_i$ deux voisinages éventuellement identiques, f une fonction objectif, x une solution initiale
$x \leftarrow$ `recherchelocale(`$\mathcal{V}_i$`,`f`,selection,`x`);`
$x^* \leftarrow x$;
répéter
  $x' \leftarrow$ `perturbation(`$\mathcal{V}_d, x$`);`
  $x \leftarrow$ `recherchelocale(`$\mathcal{V}_i$`,`f`,selection,`x'`);`
  si $f(x) < f(x^*)$ **alors** $x^* \leftarrow x$;
jusqu'à *la condition de fin*;

Recherche à voisinage variable

La *recherche à voisinage variable* [Mladenović et Hansen, 1997] est une métaheuristique basée sur le changement systématique de voisinage au cours de la recherche. Des voisinages $\mathcal{V}_k$ ($k \in \{1, 2, \ldots, k_{max}\}$) sont ordonnés selon un ordre défini par k. Cet ordre peut correspondre à une imbrication de voisinages, $\mathcal{V}_1 \subset \mathcal{V}_2 \subset \ldots \subset \mathcal{V}_{k_{max}}$, ou simplement correspondre à la taille des voisinages. $\mathcal{V}_1$ est alors le plus petit et $\mathcal{V}_{k_{max}}$ le plus grand, $\mathcal{V}_1 < \mathcal{V}_2 < \ldots < \mathcal{V}_{k_{max}}$. Une bonne connaissance du problème et du type d'instance à résoudre peut aussi amener l'utilisateur à définir un ordre *ad hoc*. Les voisinages sont explorés un à un, le passage au voisinage d'indice supérieur se faisant si une solution améliorante n'a pas été trouvée dans le voisinage courant (Algorithme 1.5).

Algorithme 1.5: Algorithme de recherche à voisinage variable.

Entrées : $\mathcal{V}_k$ un ensemble de voisinages ($k \in \{1, 2, \ldots, k_{max}\}$), f une fonction objectif, x une solution initiale

$x^* \leftarrow x$;
répéter
  $k \leftarrow 1$;
  répéter
    $x' \leftarrow$ `choixaléatoire(`$\mathcal{V}_k(x)$`)`;
    $x \leftarrow$ `recherchelocale(`$\mathcal{V}_k$`,`f`,selection,`x'`)`;
    si $f(x) < f(x^*)$ **alors**
      $x^* \leftarrow x$;
      $k \leftarrow 1$;
    sinon $k \leftarrow k + 1$
  jusqu'à $k > k_{max}$;
jusqu'à *la condition de fin*;

Recherche locale guidée

Une métaheuristique doit avoir un mécanisme pour échapper aux optima locaux et une solution est un optimum local en fonction d'une certaine fonction objectif f, pas forcément pour une autre fonction f'. La *recherche locale guidée* [Voudouris, 1997] se base sur cette observation : utiliser f' à la place de f peut permettre de s'échapper d'un optimum local de f. En pratique cette métaheuristique utilise plusieurs fonctions ou des variantes de la même fonction. L'algorithme 1.6 illustre le comportement de la métaheuristique. Bien entendu, l'aspect critique consiste à bien choisir la fonction. La nouvelle fonction pourra, par exemple, pénaliser les variables de la fonction objectif qui contribuent le plus à sa valeur.

Algorithme 1.6: Algorithme d'une recherche locale guidée.

Entrées : $\mathcal{V}$ un voisinage, f une fonction objectif, x une solution initiale

$f_1 \leftarrow f$; $k \leftarrow 1$; $x^* \leftarrow x$;
répéter
  $x \leftarrow$ `recherchelocale(`$\mathcal{V}$`,`f_k`,selection,`x`)`;
  si $f(x) < f(x^*)$ **alors** $x^* \leftarrow x$;
  $f_{k+1} \leftarrow$`mise-a-jour-fonction(`x`,`f_k`)`;
  $k \leftarrow k + 1$;
jusqu'à *la condition de fin*;

1.6 Méthodes à population de solutions

Comme leur nom l'indique, les métaheuristiques à population de solutions considèrent simultanément un ensemble de solutions. Ces méthodes s'inspirent généralement de la nature, et notamment du règne animal.

Les méthodes à population de solutions sont souvent des algorithmes évolutionnaires [Back, 1996; De Jong, 2006]. Ceux-ci sont des métaheuristiques qui utilisent des mécanismes de reproduction, de mutation [3] et de sélection, inspirés de la théorie de l'évolution, afin de faire évoluer une population de solutions. De nouvelles solutions sont produites en combinant deux solutions « parentes » ou plus (reproduction). Lors de la création d'une nouvelle solution certaines caractéristiques transmises par les parents peuvent changer aléatoirement avec une certaine probabilité (mutation). Des solutions sont éliminées au fur et à mesure de la recherche, en général pour conserver une population de taille constante (sélection).

Parmi les algorithmes évolutionnaires, la sous-classe d'algorithmes la plus populaire sont les algorithmes génétiques [Holland, 1975; Goldberg, 1989; Langdon et Poli, 2002]. Ces derniers utilisent initialement un encodage des solutions comme une chaîne de 0 et de 1, ce qui ressemble à l'encodage d'un brin d'ADN par les nucléotides A, G, C et T.

Les algorithmes mémétiques [Moscato, 1989] allient les algorithmes génétiques à la recherche locale. Celle-ci, remplaçant en général la mutation, permet d'améliorer la convergence vers de meilleures solutions.

Un autre pan des métaheuristiques à population de solutions cherche à émuler l'intelligence distribuée dont font preuve des groupes d'animaux ou d'organismes sans effort de concertation explicite. Un exemple majeur est l'optimisation par colonie de fourmis [Dorigo, 1992]. Cette technique visent à reproduire le comportement des fourmis qui arrivent, par l'utilisation de phéromones, à trouver le chemin le plus court entre deux points. On notera également qu'il existe, entre autres, des méthodes s'inspirant des essaims d'abeilles [Karaboga et Akay, 2009] ou des bancs de poissons [Filho *et al.*, 2009].

1.7 Conclusion

Dans ce chapitre nous avons présenté quelques notions relatives aux problèmes d'optimisation et de satisfaction de contraintes. Ces problèmes ne sont pas uniquement de nature académique et apparaissent dans le quotidien de tout un chacun : planification de tournées de véhicules, ordonnancement sur des chaînes de production ou encore génération d'emplois du temps.

Face à l'importance de ces problèmes, des efforts conséquents ont été déployés afin de trouver des solutions optimales. Nous avons vu que les problèmes que l'on souhaite traiter sont souvent difficiles. Trouver la solution optimale, ou plutôt que l'on pourra garantir comme étant optimale, n'est alors pas viable. On peut alors se satisfaire d'une solution de bonne qualité mais non nécessairement optimale. C'est ici que rentre en jeu les méthodes approchées, et notamment les métaheuristiques, qui permettent d'atteindre ce but.

Au sein des métaheuristiques, nous avons présenté le concept de la recherche locale, qui nous intéresse tout particulièrement dans cette thèse. Nous avons évoqué les différentes notions importantes de la recherche locale, notamment celles de mouvement, de voisinage, de paysage de recherche, de qualité et de diversité des solutions. Ce sont les éléments de bases sur lesquels nous construirons nos contributions.

3. Également appelés opérateurs de variance.

À travers la description d'un certain nombre de méthodes de recherche locale nous avons pu voir différentes stratégies de résolution se basant sur les notions évoquées précédemment. Toutefois elles partagent toutes l'exploration d'au moins un voisinage et intègrent, d'une façon ou d'une autre, un mécanisme de diversification. Ainsi ces méthodes utilisent au moins un mécanisme d'intensification et un mécanisme de diversification au sein d'un processus plus ou moins déterministe qui fixe l'ordre de leurs applications. Il est donc intéressant d'étudier des processus de sélection plus génériques, où l'intensification et la diversification peuvent avoir une importance plus ou moins grande selon l'état de la recherche.

Dans le chapitre suivant nous nous intéresserons à comment utiliser plusieurs voisinages, chacun au sein d'un opérateur. Parmi d'autres, nous formaliserons la notion d'opérateur de mouvement et nous examinerons comment les analyser et sélectionner ceux que l'on pourra considérer comme bons.

Chapitre 2

Recherche autonome

Sommaire

2.1 Introduction 24
2.1.1 No Free Lunch 24
2.1.2 Solveur 25
2.1.3 Taxonomie et caractérisation 26
2.2 Approches hors ligne 27
2.2.1 REVAC 28
2.2.2 Racing 28
2.2.3 ParamILS 29
2.3 Approches en ligne 31
2.3.1 Recherche locale réactive 32
2.3.2 Hyperheuristiques 33
2.3.3 Sélection adaptative d'opérateurs 35
2.4 Conclusion 41

2.1 Introduction

Ce travail de thèse s'inscrit dans le contexte de la recherche autonome qui s'intéresse à la gestion automatisée des algorithmes de résolution de problèmes, par exemple lorsqu'il s'agit de choisir les mouvements permettant de modifier les solutions ou encore les valeurs des paramètres. Ceci peut se faire grâce à des méthodes hors ligne, c'est-à-dire avant l'exécution de l'algorithme en lui-même, ou en ligne, c'est-à-dire pendant l'exécution de l'algorithme. Le terme *recherche autonome* [Hamadi *et al.*, 2012] est un terme générique qui vise à unifier un nombre de techniques existantes, qui elles-même se recouvrent parfois entre elles. Ainsi la recherche autonome regroupe les algorithmes adaptatifs, la recherche réactive, les hyperheuristiques ou encore les approches portfolio.

Il est utile de se rappeler que la création d'algorithmes de plus en plus sophistiqués a entraîné une inflation du nombre de paramètres. De plus les corrélations entre paramètres et l'influence des paramètres entre eux ne sont, dans nombres de cas, pas ou peu connues et étudiées. Ainsi les récentes avancées en paramétrage automatique ont permis de trouver de bien meilleurs paramétrages à des algorithmes qui avaient pourtant été longuement et assidûment paramétrés à la main [Hutter *et al.*, 2007].

Le problème de sélection d'un bon algorithme, ou de la meilleure configuration d'un algorithme, se présente dans de nombreuses situations. Il a été étudié relativement tôt dans l'histoire de l'informatique. Ainsi, [Rice, 1976] propose des modèles abstraits pour formaliser et clarifier les situations où se posent le problème de sélection d'algorithme. Une fois les caractéristiques du contexte isolées, il est nécessaire de trouver un algorithme approprié dans l'espace des algorithmes disponibles et d'en évaluer les performances par rapport à un ensemble de mesures. Plus récemment [Smith-Miles, 2008] propose un cadre unifié qui considère le problème de sélection d'algorithme comme un problème d'apprentissage et qui fédère les différentes méthodes développées dans différents champs disciplinaires.

2.1.1 No Free Lunch

Le recherche autonome s'intègre dans cette quête du choix et de la paramétrisation des algorithmes, généralement dans le contexte des problèmes d'optimisation et de satisfaction de contraintes. Cet objectif n'est toutefois pas anodin. En effet, l'intérêt croissant de la communauté scientifique et du monde de l'industrie pour des algorithmes d'optimisation génériques fonctionnant plus ou moins sur le principe d'une boite noire se heurte au théorème *No Free Lunch* (NFL) ou « aucun repas gratuit » [Wolpert et Macready, 1997]. Le NFL formalise l'intuition que tous les algorithmes boîte noire se comportent de façon identique si on les considère sur l'ensemble des *problèmes discrets*, et corollairement, que si un algorithme se comporte bien sur une certaine classe de problèmes alors, nécessairement, il en paie le prix, par une performance dégradée, sur l'ensemble des problèmes restants. En particulier, un algorithme ayant de meilleures performances qu'une recherche aléatoire sur une classe de problèmes aura des performances inférieures à une recherche aléatoire sur les problèmes restants. [Schumacher *et al.*, 2001] ayant montré que le NFL est vérifié si et seulement si l'on considère un ensemble de fonctions objectif – de problèmes – fermé par permutation, [1] la formalisation du NFL est donnée par :

$$\sum_{f \in \mathcal{F}} P(f, a_1) = \sum_{f \in \mathcal{F}} P(f, a_2) \tag{2.1}$$

1. Une permutation d'une fonction objectif correspond à un réarrangement des valeurs associées aux objets de l'espace de recherche. Alors, un ensemble de fonctions objectif est fermé par permutation si, pour chaque fonction f de cet ensemble, toutes les permutations possibles de f sont dans l'ensemble.

où a_1 et a_2 sont deux algorithmes et $\mathcal{F}$ est l'ensemble des fonctions objectif fermé par permutation. Alors sur toutes les fonctions $f \in \mathcal{F}$ les sommes des mesures de performance P sont égales.

Cependant, il n'est pas nécessaire d'avoir une lecture pessimiste du NFL. Dans l'absolu, il n'est pas grave de ne pas avoir un algorithme qui fonctionne bien sur tout, du moment que la classe de problèmes traités correctement est suffisamment large pour l'utilisation que l'on souhaite en faire. Le problème du NFL ne se pose alors pas vraiment.

En effet, [Poli et Graff, 2009] montre, grâce à la fermeture par permutation, que si l'on sait qu'un problème a au moins n valeurs distinctes $f_1, \ldots, f_n$ aux points $x_1, \ldots, x_n$ alors il est possible de permuter l'affectation des valeurs aux points x_i de, au moins, $n!$ façons différentes. Si l'ensemble de problèmes en contient moins de $n!$, alors il y a un *free lunch*.

Donc, en pratique, lorsque la recherche autonome est utilisée pour trouver un bon solveur, ou une bonne configuration d'un solveur, sur un ensemble de problèmes qui n'est pas trop grand alors il y a vraisemblablement un *free lunch*, un « repas gratuit ». Concevoir un solveur autonome n'en demeure pas moins une entreprise ambitieuse.

2.1.2 Solveur

Un mécanisme de recherche autonome est généralement intégré dans un solveur. Un *solveur* est constitué de différents composants internes qui correspondent aux différents algorithmes impliqués dans le processus de recherche. Un système de recherche ou un solveur autonome [Hamadi *et al.*, 2011] doit avoir la possibilité de modifier ses composants internes pour s'adapter à des événements externes. Ceux-ci peuvent être des informations collectées au cours de la recherche, plus particulièrement des informations sur le paysage de recherche. Ces événements peuvent également émaner de facteurs extérieurs tels que des règles ou des modèles prédictifs. L'objectif, pour le système, est alors d'essayer d'améliorer sa performance de recherche en adaptant sa stratégie de recherche au problème à traiter.

Le mécanisme de recherche a pour rôle de gérer les interactions entre différentes techniques de résolutions qui sont abstraites ici par la notion d'opérateurs. L'algorithme de recherche emploie alors une stratégie qui détermine la séquence d'application de ces opérateurs. L'algorithme de recherche est à différencier du solveur en lui-même, le premier faisant partie du second. Cet algorithme est conçu selon un modèle interne définissant l'espace de recherche et utilise une fonction afin d'évaluer les éléments de cet espace.

Tous ces composants peuvent être soumis à différents paramètres permettant de définir leur comportement. Une paramétrisation de ces composants est appelée une configuration du solveur. L'aspect « autonome » du solveur est alors incarné par une couche de contrôle permettant de gérer les composants et de modifier la configuration du solveur. La Figure 2.1 illustre l'architecture générale d'un solveur selon les concepts qui viennent d'être présentés. Les différents éléments du schéma sont brièvement décrits plus bas.

Modèle. Le codage du problème est considéré comme faisant partie intégrante du solveur. En effet, le solveur est conçu pour un encodage spécifique qui induit une représentation spécifique interne correspondant au modèle.

Évaluation. De façon générale une fonction d'évaluation est nécessaire afin d'évaluer les configurations possibles du problème en fonction de ses contraintes et de la valeur de ses variables. Elle peut permettre d'évaluer les violations des contraintes.

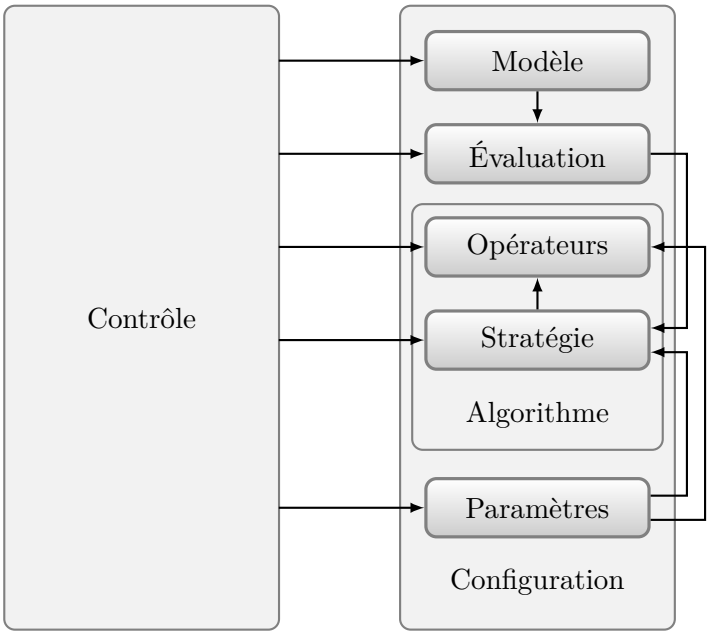


FIGURE 2.1 – Architecture générale d'un solveur.

Elle peut également être utilisée pour élaguer l'espace de recherche dans le cadre de problèmes d'optimisation.

Algorithme de résolution. Il est intéressant de distinguer la structure même de l'algorithme de ses composants configurables. Les opérateurs sont utilisés pour calculer les états de la recherche : instanciation de variables, propagation de contraintes, mouvement de recherche locale, sélection, ... La stratégie de recherche détermine l'ordre d'application des opérateurs. Dans le cas des métaheuristiques, cela correspondra par exemple à la gestion de la liste tabou.

Paramètres. Les paramètres servent à modifier le comportement des différents composants. Une configuration du solveur est alors une instance des paramètres et des composants.

Contrôle. Les solveurs modernes possèdent des mécanismes leur permettant de changer leur configuration (opérateurs, paramètres, modèle). Ces mécanismes sont souvent basés sur des techniques d'apprentissage.

2.1.3 Taxonomie et caractérisation

La gestion de paramètres est un problème majeur. Plusieurs auteurs ont tenté de fédérer et de catégoriser les différentes approches possibles. Dans le contexte des algorithmes évolutionnaire, [Eiben *et al.*, 1999; Eiben *et al.*, 2007] proposent une taxonomie générale de ces méthodes illustrée par la Figure 2.2. Cette classification peut toutefois être facilement transposée à des contextes autres que le champ évolutionnaire.

Cette taxonomie classe les méthodes selon le moment où elles tentent de déterminer la valeur des paramètres : avant l'exécution de l'algorithme de recherche, on parle alors de *réglage*, ou au cours de l'exécution, on parle alors de *contrôle*.

L'objectif du réglage est d'obtenir des valeurs de paramètres pouvant être utiles sur une large palette de problèmes. Ceci implique généralement un nombre consé-

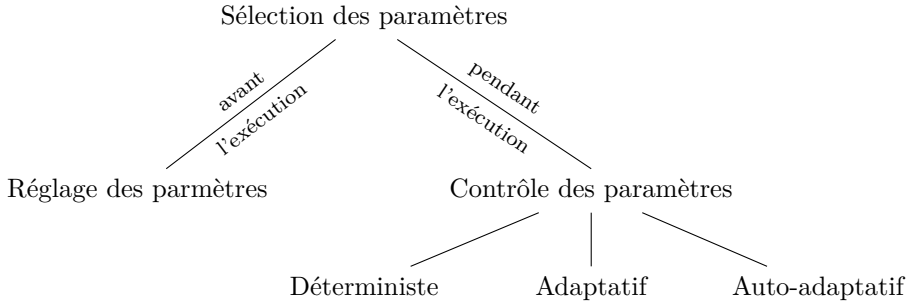


FIGURE 2.2 – Taxonomie du contrôle proposée dans [Eiben *et al.*, 1999; Eiben *et al.*, 2007].

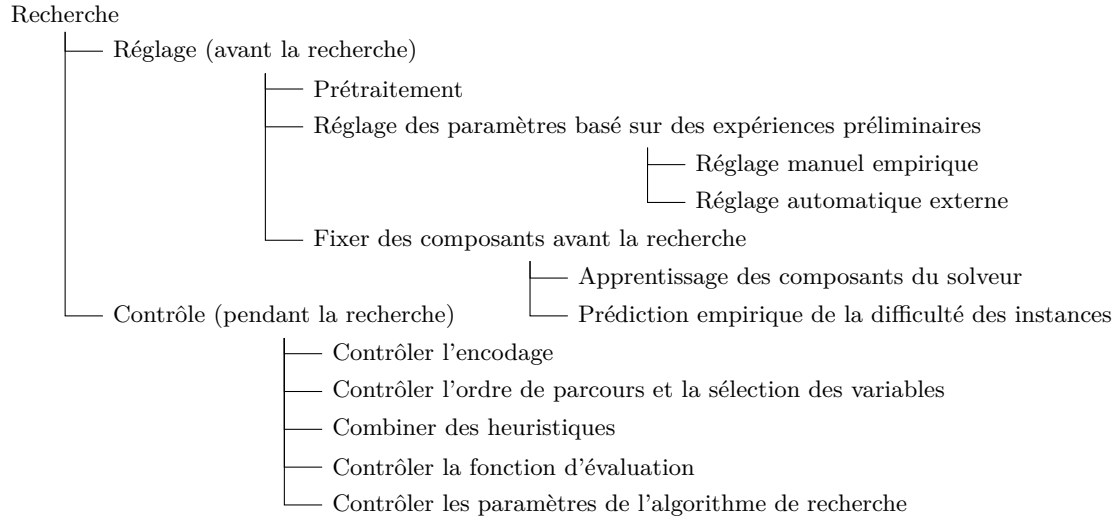


FIGURE 2.3 – Caractérisation générale des solveurs [Hamadi *et al.*, 2011].

quent d'expérimentations. Le contrôle quant à lui est décomposable en trois catégories. Il est *déterministe* lorsque les paramètres sont modifiés selon des règles préétablies. Il est *adaptatif* lorsque la modification des paramètres suit certaines règles qui tiennent compte de l'état de la recherche. Enfin, il est *auto-adaptatif* lorsque les paramètres sont encodés dans les individus – les solutions – afin d'évoluer conjointement avec les autres variables du problème.

Dans [Hamadi *et al.*, 2011], l'emphase étant mise sur les problèmes de satisfaction de contraintes, les auteurs proposent une description formelle du comportement des solveurs afin de pouvoir les caractériser rigoureusement. La représentation générale de ces caractéristiques est illustrée par la Figure 2.3.

Comme nous l'avons vu, le premier point de branchement des taxonomies proposées répartit les approches selon qu'elles soient hors ligne ou en ligne. Les deux prochaines sections abordent ces deux catégories.

2.2 Approches hors ligne

Les approches hors ligne de la recherche autonome, que l'on appellera également *méthodes de réglage*, visent à déterminer un bon paramétrage pour un algorithme donné. À cette fin, il est nécessaire d'explorer l'espace autorisé de recherche qui découle du produit cartésien de l'espace des paramètres, de l'espace des problèmes [2]

2. Un problème correspond ici à un ensemble d'instances possibles.

que l'on souhaite pouvoir résoudre (en général un seul problème) et de l'espace des instances d'apprentissage.

Un problème auquel se heurtent de telles approches est la taille de l'espace de recherche. À celui-ci s'ajoute celui de la nature chronophage de l'opération que l'on cherche à effectuer. En effet, l'évaluation de chaque triplet (vecteur de paramètres, problème, instance) demande l'exécution d'un programme ce qui requiert généralement un temps non négligeable. Enfin le paramétrage hors ligne se heurte à la même problématique que les autres problèmes d'apprentissage : le choix des instances d'apprentissage.

Notons que le problème de sélection d'opérateur auquel cette thèse s'intéresse peut naturellement être ramené à un problème de paramétrage.

Les méthodes de réglages peuvent être décomposées en trois groupes selon les approches qu'elles privilégient afin de réduire l'espace de recherche [Eiben et Smit, 2012] :

- utiliser un petit nombre de configurations de paramètres – par exemple REVAC utilise l'entropie de Shannon et l'entropie différentielle afin de trouver des paramètres qui ont un impact significatif sur la performance de l'algorithme à régler ;
- utiliser un petit nombre de tests – par exemple les méthodes de *racing* qui cherchent à éliminer rapidement les configurations les moins performantes ;
- utiliser un petit nombre de configurations de paramètres et de tests – par exemple ParamILS qui utilise la recherche locale itérée.

2.2.1 REVAC

REVAC (*Relevance Estimation and Value Calibration*) [Nannen et Eiben, 2006; Nannen et Eiben, 2007] est une méthode évolutionnaire. Au cours de son processus de recherche pour déterminer de bonnes valeurs de paramètres, REVAC crée une distribution de probabilité pour chaque paramètre. En fonction de leurs performances sur les tests antérieurs, les valeurs de paramètres ayant montrées qu'elles étaient utiles auront une plus forte probabilité que les autres. Initialement, toutes ces distributions représentent une variable aléatoire uniforme. Elles sont ensuite mises à jour après chaque test sur la base des informations qui ont été collectées. Au terme de l'exécution de REVAC, ces distributions peuvent être consultées et analysées, ceci afin d'obtenir non seulement les valeurs, ou les intervalles de valeurs, prometteuses, mais également des informations sur la pertinence de chaque paramètre.

En pratique REVAC part d'une distribution uniforme $\mathcal{C}_0$ sur l'espace des paramètres dans laquelle sont choisis m vecteurs de paramètres de façon uniforme. La performance de chaque vecteur est mesurée et les n meilleurs vecteurs sont sélectionnés. Ces derniers définissent une nouvelle distribution $\mathcal{C}_{i+1}$. Le vecteur de paramètre le plus ancien est remplacé par un vecteur choisi dans $\mathcal{C}_{i+1}$. Le processus se poursuit itérativement en mesurant la performance de ces m vecteurs et ainsi de suite.

On notera que contrairement aux méthodes de réglage évoquées par la suite, REVAC fonctionne sur des domaines continus de valeurs et ne nécessite pas leur discrétisation. Comme REVAC est une méthode évolutionnaire, les domaines continus se prêtent bien aux paramètres typiques de ces méthodes, notamment le taux de mutation.

2.2.2 Racing

Les procédures de *racing* – nous utiliserons ici le terme anglais qui nous parait plus approprié que la traduction littérale, course – évaluent itérativement des configurations de paramètres sur des instances de test et utilisent des tests d'hypothèses

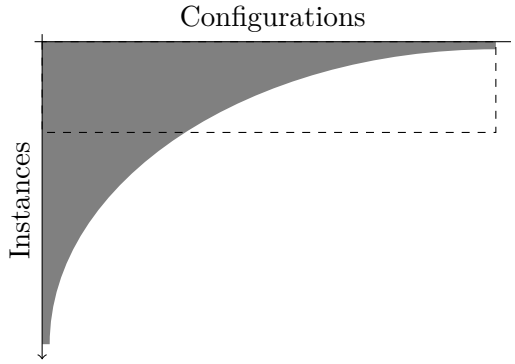


FIGURE 2.4 – Illustration de la différence de la stratégie de calcul d'une approche racing par rapport à la force brute. Le racing est représenté par la partie grisée. On observe que le nombre de configurations diminue à mesure que les configurations les moins performantes sont éliminées. Par contre la force brute teste toutes les combinaisons (instance, configuration). Le rectangle en pointillés représente, pour la force brute, un nombre de tests équivalent à celui utilisé par le racing. Cette illustration est reproduite à partir de [Birattari, 2004].

statistiques afin d'éliminer les configurations candidates qui sont statistiquement significativement surpassées par d'autres configurations. Cette méthode ne peut être utilisée qu'avec des algorithmes dont les paramètres peuvent être discrétisés, c'est-à-dire choisir un certain nombre de valeurs d'un domaine continu.

Les procédures de racing ont été introduites afin de résoudre les problèmes de sélection de modèle en apprentissage [Maron et Moore, 1994]. Dans le contexte de la sélection de configuration de paramètres, l'idée principale est d'évaluer séquentiellement les configurations candidates sur une série d'instances de test et d'éliminer les configurations qui restent à la traîne. De cette façon il est possible de se concentrer sur les configurations prometteuses. Cette approche a été tentée avec succès dans l'outil *F-Race* [Birattari, 2004] et plus récemment dans *Iterated F-Race* [Birattari *et al.*, 2010]. La Figure 2.4 illustre le gain en temps obtenu en utilisant une méthode de racing par rapport à l'utilisation de la force brute.

F-Race utilise le test statistique d'analyse de variance de rangs à deux critères de Friedman (*Friedman's two-way analysis of variance by ranks*) afin d'éliminer les configurations. Cette méthode sera utilisée dans la Section 6.5 pour régler certains algorithmes.

La variante Iterated F-Race est basée sur les modèles. En partant des mêmes valeurs de paramètres que pour F-Race, les configurations efficaces sont utilisées afin de mettre à jour un modèle probabiliste sur l'espace de recherche des configurations et ce à chaque itération. Ce modèle est ensuite utilisé afin de générer de nouvelles configurations candidates.

2.2.3 ParamILS

La métaheuristique de recherche locale itérée (Section 1.5.3) a été utilisée avec succès pour l'optimisation de paramètres dans le contexte de l'outil ParamILS [Hutter *et al.*, 2007]. Cet outil, encore une fois, demande à ce que les paramètres des algorithmes soient discrétisés.

La difficulté de paramétrer un algorithme dépend du nombre de ses paramètres. S'il y en a peu, le paramétrage est relativement facile : les domaines continus sont discrétisés et toutes les combinaisons de paramètres sont testées. S'il y a beaucoup de paramètres à prendre en compte, on ne peut évidemment pas tester exhaustivement les combinaisons : le problème est difficilement soluble.

L'optimisation manuelle commence généralement par une certaine combinaison de paramètres qui semble intuitivement cohérente. Les valeurs des paramètres sont ensuite modifiées une à une et acceptées si elles apportent une amélioration. Ce processus itératif est répété jusqu'à ce qu'aucune amélioration ne soit possible en changeant la valeur d'un paramètre à la fois. Cette technique correspond à une recherche locale exécutée manuellement dans l'espace des configurations de paramètres. ParamILS correspond ainsi à l'automatisation et à l'amélioration de cette procédure manuelle.

ParamILS (Algorithme 2.1) utilise une combinaison de réglages initiaux par défaut et également aléatoires. La première amélioration itérative est utilisée comme procédure de recherche locale secondaire. ParamILS utilise un nombre fixé s de mouvements aléatoires servant de perturbations, et accepte toujours les configurations de paramètres améliorantes ou équivalentes mais réinitialise la recherche aléatoirement avec une probabilité $p_{restart}$. Le voisinage utilisé est le 1-échange, c'est-à-dire que les voisins ne diffèrent de la solution courante que d'une seule valeur de paramètre.

Algorithme 2.1: Algorithme de ParamILS.

Entrées : Θ l'espace de configuration des paramètres, $\mathcal{N}$ une relation de voisinage, `meilleur` une fonction comparant $\theta, \theta' \in \Theta$
$\theta_0 \leftarrow$ configuration par défaut $\theta \in \Theta$;
pour $i \leftarrow 1 \dots R$ **faire**
  $\theta \leftarrow$ `random` $\theta \in \Theta$;
  si `meilleur(`θ`,`θ_0`)` **alors** $\theta_0 \leftarrow \theta$;
fin
$\theta_{ils} \leftarrow$ `PremiereAmeliorationIterative(`θ_0`,`$\mathcal{N}$`)`;
répéter
  $\theta \leftarrow \theta_{ils}$;
  pour $i \leftarrow 1 \dots s$ **faire** $\theta \leftarrow$ `random` $\theta' \in \mathcal{N}(\theta)$;
  $\theta \leftarrow$ `PremiereAmeliorationIterative(`θ`,`$\mathcal{N}$`)`;
  si `meilleur(`θ`,`θ_{ils}`)` **alors** $\theta_{ils} \leftarrow \theta$;
  avec probabilité p_{restart}, $\theta_{ils} \leftarrow$ `random` $\theta \in \Theta$;
jusqu'à *la condition de fin*;
retourner θ;
Procedure `PremiereAmeliorationIterative(`θ`,`$\mathcal{N}$`)`;
répéter
  $\theta' \leftarrow \theta$;
  pour chaque $\theta'' \in \mathcal{N}(\theta')$ *parcouru aléatoirement* **faire**
    si `meilleur(`θ''`,`θ'`)` **alors**
      $\theta \leftarrow \theta''$;
      break;
    fin
  fin
jusqu'à $\theta' = \theta$;
retourner θ;

ParamILS offre la possibilité d'utiliser des paramètres conditionnels, c'est-à-dire qu'un paramètre peut influer sur un autre. ParamILS gère les paramètres conditionnels en excluant toutes les configurations du voisinage qui ne diffèrent que de la valeur d'un paramètre conditionnel qui n'est pas pertinent pour ce voisinage.

ParamILS existe en deux variantes BasicILS et FocusedILS. Dans BasicILS, la procédure qui détermine si une configuration est meilleure qu'une autre, `meilleur`, utilise exactement N échantillons pour chacune des deux distributions de coûts cor-

respondants à l'exécution de l'algorithme avec les deux configurations. Pour chaque configuration, les échantillons sont collectés en exécutant l'algorithme sur les mêmes N instances. Le fait d'utiliser un nombre fixe d'échantillons, c'est-à-dire un ensemble de test de taille fixe, pose deux problèmes principaux.

Le premier problème est que l'estimation du coût de la meilleure configuration trouvée sur l'ensemble de test sous-estime le coût de cette configuration sur l'ensemble de validation : le fait d'avoir donné le coût minimum sur certaines instances ne garantit aucunement que cette configuration soit la meilleure pour une instance quelconque. Le second problème est le « sur-réglage » (*over-tuning*, [Birattari, 2004]), qui est analogue au sur-apprentissage dans le domaine de l'apprentissage automatique. Le résultat est alors que l'exécution de réglages supplémentaires pénalise la performance lors de la validation. Une solution à ce problème est d'utiliser un grand nombre d'échantillons mais cela est relativement coûteux, tout particulièrement lorsque l'on considère l'optimisation de paramètres.

La variante FocusedILS entend répondre à ces problèmes et atteint cet objectif en concentrant l'échantillonnage sur les configurations de paramètres prometteuses. Ici `meilleur` est un peu plus complexe que dans la variante initiale. Soit deux configurations θ_1 et θ_2 et $N(\theta)$ le nombre d'échantillons utilisé pour approximer une distribution de coût, alors θ_1 *domine* θ_2 si, et seulement si, $N(\theta_1) \geq N(\theta_2)$ et que la performance de la configuration θ_1 sur les $N(\theta_2)$ premiers échantillons est meilleure que celle de la configuration θ_2.

ParamILS a été utilisé avec succès pour trouver de bonnes paramétrisations dans de très grands espaces de paramètres. Dans [Khudabukhsh *et al.*, 2009], SATenstein-LS, un algorithme de recherche locale pour résoudre le problème SAT, est configuré dans un espace de taille 4.82×10^{12}. Des améliorations de l'ordre de trois ordres de grandeurs sont obtenues. CPLEX est un solveur commercial de programmation mixte en nombre entier. Il est configuré avec ParamILS dans [Hutter *et al.*, 2010] : 76 paramètres sont considérés et l'espace exploré est de taille 1.9×10^{47}. Des améliorations de l'ordre de 1 à 10 sont obtenues.

Comme pour toute métaheuristique, l'utilisation de ParamILS demande une certaine connaissance afin de le paramétrer correctement pour un problème *ad hoc*. Le site `www.prog-by-opt.net`, dédié au concept de programmation par optimisation [Hoos, 2012], offre quelques conseils sur l'utilisation de cet outil. La variante FocusedILS est recommandée ainsi que dix exécutions indépendantes, celle qui obtient la meilleure performance sur un ensemble d'instances de validation doit alors être sélectionnée. Lorsque ParamILS est utilisé pour optimiser le temps d'exécution, la recommandation est d'utiliser un ensemble de test l, un temps limite d'exécution t et un temps de configuration (pour chaque exécution indépendante) T tels que :

- pour au moins 75 % des instances de test l l'exécution de l'algorithme sur une de ces instances se termine en un temps t et
- le temps total budgété est au moins 200, voire 1 000, fois supérieur à t.

2.3 Approches en ligne

Les approches en ligne modifient dynamiquement les valeurs des paramètres et le comportement d'un algorithme au cours de son exécution. En effet, à mesure que la recherche progresse, les données disponibles sur le comportement de l'algorithme et ses paramètres permettent d'obtenir des informations. Celles-ci sont plus ou moins locales et sont influencées par le paysage de recherche dans lequel la recherche est en train de se déplacer. Il est raisonnable de penser que l'on puisse tirer partie de ces données accumulées. De plus, suivant la situation de la recherche, il peut être nécessaire de privilégier certaines stratégies par rapport à d'autres, et plus particu-

lièrement le compromis entre exploitation et exploration de l'espace de recherche.

Le problème qui nous intéresse est la sélection d'opérateurs. Un opérateur modélise une action permettant de produire une nouvelle solution. En recherche locale, dans sa forme la plus simple, un opérateur sera composé d'une relation de voisinage et d'une fonction de sélection permettant de choisir un voisin. Un opérateur peut également être beaucoup plus complexe : on pourrait très bien considérer une application d'une métaheuristique comme un opérateur.

La performance d'un opérateur dépend bien entendu des caractéristiques du problème à traiter. Comme il est généralement relativement difficile de prévoir le comportement d'un opérateur sur un problème *ad hoc*, il est possible de faire appel aux approches hors ligne mentionnées précédemment afin de déterminer une bonne, voire la meilleure, stratégie statique qui appliquera les opérateurs avec des probabilités fixes de sélection.

Toutefois, une stratégie statique ne tient pas compte des caractéristiques locales de la région du paysage de recherche en cours d'exploration. Ainsi un opérateur qui serait globalement mauvais si l'on considère le processus complet de recherche pourrait très bien être particulièrement performant dans certains cas très spécifiques.

Avant de voir la gestion et la sélection d'opérateur dans le contexte des hyperheuristiques (Section 2.3.2) ou de la sélection adaptative d'opérateurs (Section 2.3.3), intéressons nous aux techniques de recherche locale réactive.

2.3.1 Recherche locale réactive

Nous présentons ici quelques méthodes de recherche locale réactive [Battiti *et al.*, 2008]. La recherche locale réactive, ou adaptative, regroupe des méthodes de recherche locale basées sur des techniques d'apprentissage. Ces méthodes ajustent de façon automatique leurs paramètres internes pendant le processus de recherche.

Les méthodes de recherche locale, comme la recherche tabou, possèdent souvent des paramètres relativement sensibles. Les meilleures valeurs de ces paramètres ne sont pas les mêmes en fonction du problème à traiter, de l'instance du problème à résoudre, voire de la région du paysage de recherche.

La recherche tabou réactive est sans doute la plus connue des approches réactives. Nous avons également choisi de présenter trois autres méthodes moins connues mais que nous considérons intéressantes à mentionner. Elles emploient différentes propriétés dans leur processus adaptatif et se situent dans des contextes différents.

Recherche tabou réactive

Dans la recherche tabou classique (Section 1.5.3), lorsque qu'un mouvement est effectué, le mouvement inverse sera interdit pendant un temps t fixe (la longueur de la liste tabou). L'efficacité de la recherche tabou classique dépend de la valeur t. Si elle est trop petite, il ne sera pas possible d'échapper aux bassins d'attraction des optima locaux. Si elle est trop grande, la trajectoire de recherche sera fortement contrainte, ce qui augmente la probabilité de « passer à côté » de l'optimum global.

La recherche tabou réactive (*Reactive tabu*, ReTS, [Battiti et Tecchiolli, 1994]) introduit un temps t variable. Ce temps est augmenté en présence d'indices indiquant que la recherche est dans un bassin d'attraction, il est diminué dans le cas contraire. Ceci demande de conserver toutes les solutions trouvées pendant la recherche. Après l'exécution d'un mouvement, l'algorithme vérifie si la solution actuelle a déjà été trouvée et réagit en fonction.

Ainsi, s'il est vraiment difficile d'échapper à un bassin d'attraction et que l'algorithme retrouve plusieurs fois le même optimum local, le temps t augmentera jusqu'à ce qu'il n'y ait pas d'autre choix que de sortir du bassin.

Recherche à voisinage variable réactive

Contrairement aux autres métaheuristiques de recherche locale, la recherche à voisinage variable (VNS, Section 1.5.3) ne suit pas une trajectoire mais explore de nouveaux voisinages, généralement des voisinages de plus en plus grands. À l'intérieur de chaque voisinage, une procédure de recherche locale est utilisée afin de trouver un optimum local.

Dans la recherche à voisinage variable réactive [Bräysy, 2003] (RVNS), lorsque la recherche locale n'est plus en mesure d'améliorer la solution, la fonction objectif est modifiée afin d'échapper à l'optimum local. Cette modification peut être faite en remplaçant l'objectif courant par un nouvel objectif ou en considérant simultanément différents objectifs et en ajustant leurs poids. C'est l'approche employée dans la publication citée. Les poids sont modifiés en fonction des solutions déjà visitées, ce qui rappelle le mécanisme réactif de ReTS.

Recherche adaptative et neutralité

Lorsque l'on fait face à des plateaux très larges dans le paysage de recherche, il est important de pouvoir se diriger très rapidement vers de bonnes solutions. VEGAS, *Varying Evolvability-Guided Adaptive Search* [Marmion *et al.*, 2011], est une nouvelle méthode conçue pour tenir compte de la neutralité, c'est-à-dire les configurations ayant la même qualité. Elle est guidée par la solution connue la plus « évolvable ». [3] Plus une solution possède des voisins de meilleure qualité que les autres, plus elle est évolvable.

Les méthodes de recherche locales se basent sur la dernière solution obtenues afin de produire une nouvelle solution. Dans VEGAS, l'idée est de considérer toutes les solutions déjà visitées du plateau courant et pas uniquement la dernière. Le mécanisme de sélection utilise un bandit manchot multi-bras (Section 2.3.3).

Recherche à grands voisinages adaptative

La recherche à grands voisinages, *Large Neighborhood Search* [Shaw, 1998] (LNS), combine la recherche locale et la programmation par contraintes. À chaque étape de la recherche locale, un sous-ensemble des variables est relaxé et la programmation par contraintes est utilisée pour explorer le voisinage à la recherche d'une solution. La taille des sous-ensembles de variables et leur choix sont des paramètres cruciaux pour le bon fonctionnement de l'algorithme.

La recherche à grands voisinages adaptative [Mairy *et al.*, 2010; Mairy *et al.*, 2011] utilise l'impact antérieur des variables afin de sélectionner les sous-ensembles de variables à relaxer. Cet impact est calculé en fonction du résultat de la procédure de programmation par contraintes et des variables qui étaient en jeu. Plus spécifiquement, l'impact d'une variable x_i correspond au changement moyen de la borne de la fonction objectif lorsque x_i est modifiée. La borne en question est la borne inférieure ou supérieure selon que l'on se place dans un contexte de minimisation ou de maximisation.

2.3.2 Hyperheuristiques

Le terme *hyperheuristiques* a été proposé dans [Cowling *et al.*, 2001] pour décrire les approches « à un niveau d'abstraction au-dessus de celui d'une métaheuristique » et n'ayant « aucune connaissance du domaine d'application, autre que les connaissances intégrées dans des heuristiques simples, pauvres en connaissance

3. Le terme « évolvable » est un anglicisme. L'*évolvabilité* est un terme générique pour évoquer la capacité d'un système à s'adapter au cours de son évolution.

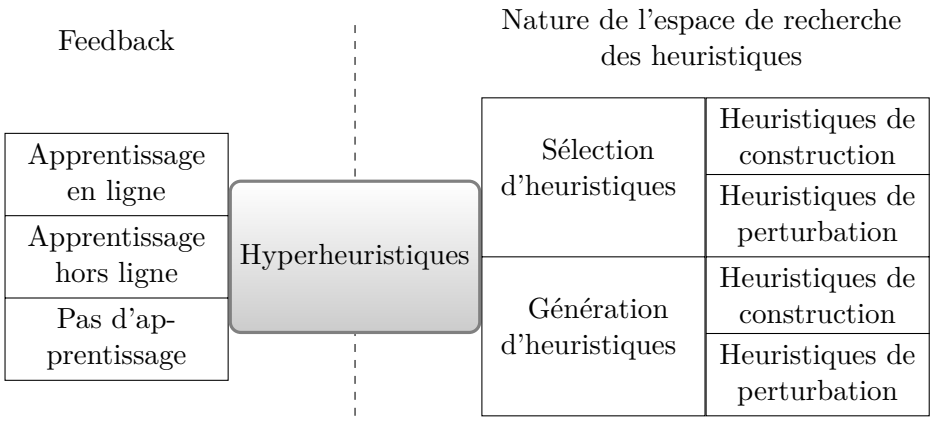


FIGURE 2.5 – Classification des approches hyperheuristiques selon deux dimensions : i) la nature de l'espace de recherche des heuristiques et ii) la source de feedback pendant l'apprentissage [Burke *et al.*, 2010].

du domaine ». Ceci était motivé par le fait que les métaheuristiques modernes ont tendance à être complexes : elles demandent des connaissances substantielles du domaine d'application et une profonde compréhension de la méthode de recherche elle-même. Bien que ces approches se soient montrées efficaces, leur complexité les rend coûteuses à mettre en œuvre et difficiles à adapter à un autre domaine. [Burke *et al.*, 2003] soutient que les utilisateurs finaux « sont plus souvent intéressés par des solutions à leurs problèmes qui seraient suffisamment correctes, suffisamment rapides et suffisamment peu coûteuses. »

Les hyperheuristiques peuvent être définies comme des heuristiques qui sélectionnent des heuristiques à partir d'une population d'heuristiques. Dès lors, une hyperheuristique n'opère plus dans l'espace des solutions, ceci est le privilège des heuristiques sélectionnées, mais elles opèrent dans l'espace des heuristiques. Il est important de noter que les heuristiques peuvent varier d'un simple mouvement à des métaheuristiques complexes. À chaque étape, le processus de sélection choisit l'heuristique la plus prometteuse. Idéalement, cette décision ne devrait ne demander aucune ou très peu de connaissance du fonctionnement intrinsèque des heuristiques, mais devrait se baser sur l'analyse de la ou des fonctions objectif et de certains indicateurs de performance.

La Figure 2.5 présente une classification [Burke *et al.*, 2010] des hyperheuristiques. L'espace de recherche des heuristiques peut être scindé en méthodes de sélection ou de génération d'heuristiques. Celles-ci peuvent être des heuristiques de construction ou de perturbation. Un hyperheuristique utilise généralement un processus de feedback qui peut être reçu en ligne ou hors ligne. Cela rejoint les taxonomies évoquées précédemment (Section 2.1.3).

Il existe plusieurs approches hyperheuristiques [Chakhlevitch et Cowling, 2008] :

- sélection aléatoire – ceci est l'approche la plus simple, où une heuristique choisie aléatoirement est appliquée à chaque point de décision, qu'il y ait une amélioration ou non. Cette méthode n'est en générale utilisée que comme point de comparaison par rapport à des hyperheuristiques plus complexes. Les variantes peuvent n'accepter que les solution améliorantes, procéder par descente (appliquer la même heuristique tant qu'elle donne des solutions améliorantes), ou accepter des solutions non améliorantes uniquement si elles ne dégradent pas trop la solution précédente afin d'échapper à un optimum local ;
- sélection gloutonne et « petit creux » (*peckish*) – l'approche gloutonne compare la performance de toutes les heuristiques à chaque point de choix et sélectionne

l'heuristique ayant produit la meilleure amélioration. Ce comportement rend les approches gloutonnes plus lentes que les autres hyperheuristiques et, de plus, elle n'explorent pas efficacement l'espace de recherche. Afin de contrecarrer ce dernier point, l'approche « petit creux » sélectionne aléatoirement une heuristique parmi une liste des meilleures heuristiques candidates ;

– sélection métaheuristique – ces méthodes se basent sur des métaheuristiques pour sélectionner des heuristiques en lieu et place des solutions pour lesquelles elles ont été traditionnellement conçues.

Les premières approches hyperheuristiques ressemblant à des métaheuristiques se sont basées sur les algorithmes génétiques. Par exemple [Fang *et al.*, 1994] utilisent un processus de « choix évolutif d'heuristiques ». Les hyperheuristiques basées sur les algorithmes génétiques utilisent un codage indirect. Ceci revient à dire qu'un chromosome ne représente pas une solution mais plutôt comment cette solution doit être construite. Cette description peut être sous la forme d'une séquence d'heuristiques à appliquer ou alors elle peut définir de façon déterministe l'heuristique à utiliser à un certain moment ou en présence d'une certaine configuration de solution.

Le recuit simulé, la recherche tabou, la recherche à voisinage variable et les algorithmes de colonies de fourmi ont également été utilisés pour diriger l'exploration dans l'espace des heuristiques. Par exemple, pour la recherche tabou, [Kendall et Hussin, 2005] proposent deux versions d'hyperheuristique tabou pour les problèmes d'emploi du temps. Dans la première, qui allie la recherche tabou et la descente, toutes les heuristiques non-taboues sont considérées et celle qui donne la meilleure amélioration est appliquée à plusieurs reprises jusqu'à ce qu'aucune amélioration ne soit produite, elle devient alors tabou. Dans la seconde, alliant la recherche tabou au grand déluge [Dueck, 1993] une métaheuristique basée sur le recuit simulé, une solution est acceptée avec une certaine probabilité.

Au lieu de comparer les heuristiques directement entre elles afin de sélectionner la meilleure à appliquer à un point donné, une autre technique est d'apprendre à partir de leur comportement et de leur performance passés. Ce genre de mécanisme se base sur l'apprentissage par renforcement [Kaelbling *et al.*, 1996] : récompenser à chaque itération les heuristiques améliorantes et punir celles réussissant moins bien. Un score sur lequel est basée la sélection est ainsi associé à chaque heuristique et il évolue au fil de la recherche. Le choix de l'heuristique à appliquer peut être déterminé par un mécanisme de roulette (choix proportionnel au score) ou simplement en choisissant l'heuristique avec le meilleur score [Nareyek, 2004].

Une autre approche est d'utiliser une fonction de choix. [Soubeiga, 2003] utilise la somme de trois fonctions. Deux d'entre elles décrivent le potentiel d'intensification de l'heuristique. La première fonction qualifie le comportement de l'heuristique lorsqu'elle est utilisée seule et la deuxième fonction qualifie le comportement de l'heuristique utilisée à la suite d'une autre. La troisième fonction décrit le potentiel de diversification (en l'occurrence, le nombre de secondes écoulées depuis le dernier appel de l'heuristique).

2.3.3 Sélection adaptative d'opérateurs

La sélection adaptative d'opérateurs [Fialho, 2010] entre dans le contexte des hyperheuristiques. Toutefois, comme le nom l'indique, l'emphase est mise sur la sélection des opérateurs de résolution de base alors que la communauté des hyperheuristiques semble plus souvent se concentrer sur l'utilisation et l'adaptation de métaheuristiques existantes.

À l'instar des travaux effectués en hyperheuristiques, les travaux en sélection d'opérateurs se sont initialement, par exemple [Davis, 1989], et, pour une grande partie, concentrés sur les algorithmes génétiques et évolutionnaires. En effet, ces

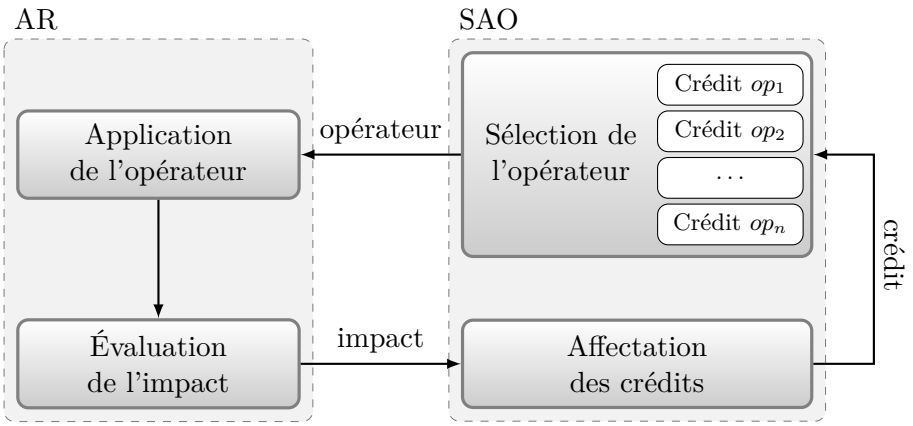


FIGURE 2.6 – Représentation générale de l'interaction entre un algorithme de recherche (AR) et un mécanisme de sélection adaptative d'opérateurs (SAO).

algorithmes utilisent en général plusieurs opérateurs de croisement et de mutation. Il est donc logique que des efforts aient été consentis afin d'étudier et d'améliorer la sélection de ces opérateurs.

La Figure 2.6 présente une vue générale de l'intégration de la sélection adaptative d'opérateurs (SAO) à un algorithme de recherche (AR), qu'il soit de recherche locale ou évolutionnaire. Ce processus peut être décrit comme suit :

1. à chaque étape de son exécution l'AR demande à la SAO de lui dire quel opérateur utiliser ;
2. la SAO sélectionne un opérateur en utilisant un mécanisme de sélection qui considère les performances antérieures de chaque opérateur.
3. l'AR applique l'opérateur sélectionné, ceci produit une ou plusieurs nouvelles solutions et influence ainsi la recherche, par exemple l'amélioration de la qualité ou augmentation de la diversité. Cette influence est appelée l'impact de l'opérateur ;
4. l'AR communique cet impact à la SAO, ce qui est typique d'un mécanisme de *feedback*. Cet impact est transformé en une valeur – appelée crédit ou récompense – calculée en fonction de la procédure d'affectation des crédits. Ce crédit est ensuite utilisé pour mettre à jour la performance (également appelée utilité) empirique estimée de chaque opérateur. Ce sont ces valeurs qui seront utilisées lors de la prochaine demande de sélection d'opérateur ;
5. le processus boucle ainsi de suite au cours de l'exécution de la recherche.

La sélection adaptative d'opérateurs peut être décomposée en deux parties distinctes. La première, l'affectation des crédits, est en amont du processus de sélection en lui-même. L'affectation des crédits est elle-même tributaire du calcul de l'impact par l'algorithme de recherche.

Impact et utilité

L'affectation des crédits revient à déterminer une valeur pour chaque opérateur permettant de juger de sa performance antérieure, et éventuellement d'autres critères, afin de pouvoir choisir un opérateur prometteur et le proposer à l'algorithme de recherche.

Trois aspects importants rentrent en jeu dans ce mécanisme :
- la façon dont est mesuré l'impact de l'application d'un opérateur ;
- la façon dont cet impact est utilisé afin de calculer le crédit ou l'utilité ;
- comment (à qui) est affecté le crédit (généralement l'opérateur lui-même).

Mesurer l'impact Généralement, l'impact de l'application d'un opérateur est directement la valeur de l'amélioration de la qualité. Cette qualité peut être celle de la solution produite, par exemple dans [Soubeiga, 2003]. Elle peut aussi être une valeur globale lorsque l'on considère une population de solutions, par exemple la qualité de la meilleure solution de la population [Davis, 1989].

Au lieu d'utiliser directement la qualité, il est possible d'utiliser des mesures binaires : si l'application de l'opérateur a produit une amélioration ou non [Niehaus et Banzhaf, 2001] ou encore si l'application de l'opérateur a produit une solution améliorant la meilleure solution trouvée jusqu'alors [Misir *et al.*, 2012].

En sus de la qualité, un autre critère qu'il est possible de considérer est la diversité de la population. Ceci est utile dans les méthodes à population afin de réduire le risque de convergence prématurée de la recherche. Dans [Maturana et Saubion, 2008a] la méthode *Compass* utilise une fonction de l'amélioration de la qualité et de la diversité. La *Pareto dominance* a également été utilisée pour l'agrégation de ces deux composantes [Maturana *et al.*, 2010].

Calculer le crédit Le crédit ou l'utilité d'un opérateur est calculé en fonction de l'impact obtenu afin d'avoir une estimation empirique sur laquelle la sélection pourra se baser. Le plus simple est d'utiliser la dernière valeur d'impact obtenue par l'application de l'opérateur. Toutefois cette valeur instantanée a tendance à être fortement variable étant donnée la nature même de la recherche qui fait appel à l'aléatoire. En général, plusieurs valeurs d'impact sont considérées, notamment en conservant les m dernières valeurs pour chaque opérateur dans une fenêtre glissante. Il est alors possible de définir le crédit comme la moyenne de ces valeurs [Maturana et Saubion, 2008a] ou comme la meilleure valeur de la fenêtre glissante [Fialho *et al.*, 2008; Maturana *et al.*, 2009]. Dans ces publications, la valeur extrême de la fenêtre permet d'obtenir de meilleures performances que la moyenne. Il est également possible d'utiliser une seule fenêtre glissante pour tous les opérateurs [Fialho, 2010] afin que des valeurs trop anciennes pour un opérateur peu sélectionné ne soient pas utilisées.

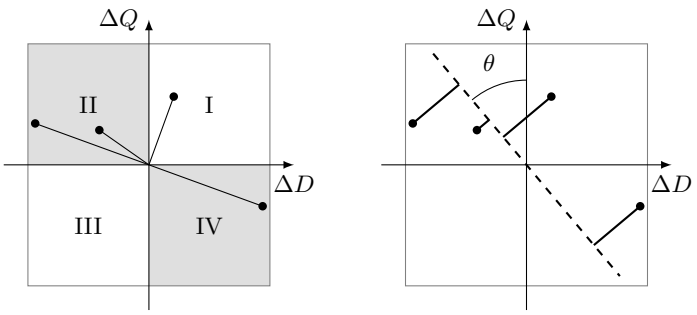
Afin que la sélection d'opérateurs soit indépendante du problème traité, il est nécessaire de s'affranchir des différents domaines de valeurs de l'impact communiqué par l'algorithme de recherche (ou du moins d'en mitiger les effets). Il convient ainsi de normaliser les valeurs obtenues en divisant le crédit d'un opérateur par le crédit le plus grand obtenu jusqu'à présent. Une alternative est de considérer les rangs que confèrent les crédits aux opérateurs [Fialho, 2010].

Affecter le crédit Bien évidemment, l'opérateur ayant eu un impact sur la recherche il est normal de lui affecter le crédit. Toutefois il existe de méthodes alternatives. Ainsi, [Julstrom, 1997] ne crédite pas uniquement l'opérateur ayant produit la solution mais également les opérateurs précédents, le crédit donné décroissant avec l'ancienneté de l'opérateur. Alternativement, au lieu de considérer l'opérateur seul, il est possible de considérer l'impact de n-uplets d'opérateurs, généralement des couples [Soubeiga, 2003; Misir *et al.*, 2012].

Considérons maintenant un exemple d'une méthode de calcul de l'utilité ayant été utilisée pour les algorithmes évolutionnaires.

Compass

La méthode *Compass* permet d'agréger différents indicateurs d'impact comme le changement de qualité, le changement de diversité ou le temps d'exécution afin



(a) Les vecteurs (b) Résultat

FIGURE 2.7 – Illustration de Compass.

d'obtenir une seule valeur qui peut être utilisée pour déterminer l'utilité d'un opérateur.

Considérons Compass avec deux indicateurs d'impact, en l'occurrence la variation de qualité et de diversité. En début de recherche, lorsqu'il n'y a aucune information sur les opérateurs, ceux-ci sont sélectionnés avec une probabilité uniforme. Une fois que les indicateurs d'impact des opérateurs sont connus – ils sont stockés dans des fenêtres glissantes – Compass a à sa disposition les valeurs q_i et d_i de la variation moyenne de la qualité et de la diversité. Ces deux valeurs définissent un vecteur $o_i = (d_i, q_i)$ qui caractérise l'opérateur i par rapport à son effet sur la variation de qualité et la diversité (axes ΔQ et ΔD dans la Figure 2.7). En général, comme la qualité et la diversité sont des critères conflictuels, les opérateurs se retrouveront dans les quadrants II et IV.

Afin de considérer simultanément les deux critères, une direction de recherche à privilégier est d'abord choisie. Elle est représentée par l'angle θ – par rapport à l'axe vertical – et définit le degré selon lequel le mécanisme doit privilégier la qualité ou la diversité. Compass calcule ensuite, pour chaque opérateur, la distance entre le point le représentant et la droite définie par l'angle θ (Figure 2.7b). La sélection d'un opérateur peut ensuite se faire en fonction de ces distances.

Sélection d'opérateurs

La sélection d'un opérateur se fait par rapport à son utilité. La méthode la plus simple et la plus largement utilisée est la sélection par roulette où chaque opérateur a une probabilité d'être sélectionné proportionnelle à son crédit. Cette probabilité proportionnelle peut être légèrement modifiée en tenant compte d'une probabilité minimum les opérateurs dont le crédit est nul. Une alternative à ce mécanisme simple de sélection est de favoriser l'opérateur ayant le plus grand score. Enfin, l'utilisation du concept de bandit manchot multi-bras de la théorie des jeux a été proposé récemment.

Probabilité proportionnelle

L'une des techniques de sélection les plus simples est la sélection avec une probabilité proportionnelle (PP) à l'utilité estimée des opérateurs. Ce processus est aussi appelé sélection par roulette, une référence à la roulette de casino. Cette analogie peut être appréhendée en imaginant une roulette sur laquelle l'encoche de chaque numéro (opérateur) est de taille proportionnelle à sa probabilité de sélection. Ce type de sélection est également appelé *probability matching* [Goldberg, 1990] en anglais. Étant très simple, elle est souvent utilisée dans la littérature sans qu'elle ne soit explicitement référencée.

Plus formellement, si l'on considère un ensemble de N opérateurs, la PP considère un vecteur $(p_{i,t}), i \in 1..N$ et une valeur d'utilité $u_{j,t}$ pour chaque opérateur j. À chaque étape t :

1. un opérateur j est sélectionné selon une probabilité $p_{j,t}$ grâce à une sélection par roulette ;
2. cet opérateur est appliqué et un crédit $r_{j,t}$ est calculé selon la méthode d'affection des crédits utilisée ;
3. l'utilité estimée $u_{j,t}$ est mise à jour. Ceci peut être fait de deux façons, selon que l'on utilise une fenêtre glissante pour conserver les résultats récents ou non :
   – sans fenêtre glissante, il convient d'utiliser une somme pondérée de l'utilité et du crédit. Le facteur $\alpha \in [0, 1]$ permet de régler l'importance de l'utilité antérieure par rapport au nouveau crédit. Ainsi, plus α est grand, plus l'importance donnée à la mémoire est faible :

$$u_{j,t} = (1 - \alpha)u_{j,t-1} + \alpha \cdot r_{j,t-1} \tag{2.2}$$

   – avec une fenêtre glissante des M derniers crédits et $S_{j,t}$ la somme de ces derniers pour l'opérateur j au temps t, si tous les crédits ont une importance identique alors :

$$u_{j,t} = S_{j,t} \tag{2.3}$$

   sinon, chacun des crédits a une importance quantifiée par un poids α_l, qui décroit vraisemblablement plus le crédit est ancien :

$$u_{j,t} = \sum_{l=1}^{M} \alpha_l \cdot r_{j,t-l} \tag{2.4}$$

4. enfin la probabilité de sélection de chaque opérateur est mise à jour :

$$p_{i,t} = \frac{q_{i,t}}{\sum_{l=1}^{N} q_{l,t}} \tag{2.5}$$

Il est important de noter que si un opérateur réalise de mauvaises performances pendant un certain temps au cours de la recherche, cela peut amener sa probabilité à être très faible, voire nulle. Or, si un opérateur est inefficace à un certain moment de la recherche, cela ne veut pas dire qu'il ne le sera jamais. Il est donc utile de spécifier une probabilité minimale obligatoire p_{min}.

L'intérêt d'une sélection par probabilité proportionnelle est que cette technique demeure très simple. C'est également ce qui peut être, selon le contexte, son inconvénient : tous les opérateurs ont une chance, même minime, d'être sélectionnés alors que cela n'a pas forcément lieu d'être. On pourrait aussi souhaiter exagérer les différences entre les opérateurs afin de produire des changements plus brusques.

Poursuite adaptative

La poursuite adaptative [Thierens, 2005] (AP) a été proposée afin de palier le problème introduit par l'utilisation de la probabilité minimum p_{min}. En effet, en plus de donner une chance aux opérateurs peu performants, ce paramètre a également tendance à favoriser les opérateurs moyens au détriment des meilleurs opérateurs : un opérateur avec une utilité nulle aura toujours une chance d'être sélectionné.

La poursuite adaptative est en grande partie similaire à la sélection proportionnelle. La différence se trouve dans la mise à jour des probabilités. Au lieu de la faire de façon proportionnelle, une stratégie de gagnant-emporte-tout (*winner-takes-all*)

est utilisée afin de favoriser le meilleur opérateur courant tout en pénalisant les autres opérateurs.

Soient $i_t^* = \arg\max_{i=1..N}\{u_{i,t}\}$ le meilleur opérateur estimé courant, [4] $\beta \in [0,1]$ le paramètre indiquant l'importance souhaité du meilleur opérateur estimé courant et $p_{max} = 1 - (N-1)p_{min}$ alors :

$$p_{i,t+1} = \begin{cases} p_{i,t} + \beta(p_{max} - p_{i,t}) & \text{si } i = i_t^* \\ p_{i,t} + \beta(p_{min} - p_{i,t}) & \text{sinon} \end{cases} \tag{2.6}$$

Dans [Thierens, 2005], AP se montre bien meilleur que PP. Toutefois, si l'on considère qu'un opérateur est définitivement le meilleur à un certain moment de la recherche, le paramètre p_{min} demeure handicapant puisqu'il favorise l'exploration d'autres opérateurs. De plus, le paramètre α est constant, ainsi l'algorithme accorde autant d'importance au crédit reçu par un opérateur qui n'a pas été utilisé depuis longtemps qu'au crédit d'un opérateur qui est constamment utilisé. Or l'opérateur qui n'a pas été utilisé depuis longtemps peut être un élément critique de la phase courante de la recherche. Il conviendrait donc de le favoriser ou, du moins de remettre rapidement les estimations d'utilité à jour. Cette réflexion a mené à la proposition de l'utilisation du bandit manchot multibras.

Bandit manchot multibras

Le concept du bandit manchot multibras [Robbins, 1952; Rodman, 1978; Lai et Robbins, 1985] (*multi-armed bandit*, MAB) provient du domaine de la théorie des jeux. Un problème MAB utilise N bras, chaque bras i étant caractérisé par une probabilité de récompense inconnue $p_i \in [0,1]$. À chaque étape t, le joueur sélectionne un bras j. Ce dernier reçoit une récompense $r_t = 1$ avec la probabilité p_j, sinon $r_t = 0$. À un instant T, la performance de la stratégie MAB est mesurée grâce à la somme des récompenses $\sum_{t=1}^{T} r_t$. Ainsi, on obtient la meilleure performance en jouant à chaque fois le meilleur bras (qui est inconnu), c'est-à-dire celui qui a la probabilité la plus élevée de récompense p^*. Le *regret* de la stratégie est la différence entre sa performance et la meilleure performance possible :

$$\mathcal{L}_T = T \cdot p^* - \sum_{t=1}^{T} r_t \tag{2.7}$$

Une des solutions au problème MAB, la borne de confiance supérieure (*Upper Confidence Bound*, UCB) [Auer *et al.*, 2002] atteint le regret optimal grâce à un critère de compromis entre exploration et exploitation. L'UCB est une approximation utilisée pour décider du bras à jouer et non une heuristique. Chaque bras i est associé à son estimation de qualité empirique u_i (la moyenne des crédits obtenus) et à un intervalle de confiance qui dépend du nombre de fois n_i que le bras i a été joué. À chaque étape, l'UCB sélectionne alors le bras i^* ayant la meilleure borne de confiance supérieure comme définie ci-après. Soient $n_{i,t+1} = n_{i,t} + 1$ le nombre d'utilisations du bras i et $u_{i,t} = \sum_{i=1}^{t} r_{i,j}/n_{i,t}$ alors

$$i_t^* = \underset{i=1..N}{\arg\max} \left(u_{i,t} + \sqrt{\frac{2\log \sum_k n_{k,t}}{n_{i,t}}} \right) \tag{2.8}$$

Le terme de gauche favorise la qualité (exploitation) et celui de droite favorise les tentatives avec les autres bras (exploration). Bien que chaque bras puisse théoriquement être sélectionné un nombre infini de fois, le laps de temps entre deux sélections

4. L'*argument maximum*, $\arg\max_x f(x)$, est la valeur de x pour laquelle $f(x)$ atteint la plus grande valeur.

d'un bras sous-optimal augmentera exponentiellement par rapport au nombre de pas.

Notons que l'optimalité mentionnée précédemment ne tient que pour des crédits binaires. Dans le contexte qui nous intéresse, celui de la sélection adaptative d'opérateurs, les crédits sont généralement des valeurs réelles. [Da Costa *et al.*, 2008] introduisent un facteur C afin de limiter l'exploration des bras, notamment lorsque le nombre de bras est conséquent.

Bandit manchot multibras dynamique

Le MAB est conçu afin de minimiser le regret et trouver le meilleur bras, l'hypothèse étant que la distribution des récompenses ne change pas. Or, dans le contexte de la sélection adaptative d'opérateurs, cette hypothèse est invalide : un opérateur n'aura pas forcément les mêmes performances suivant l'évolution de la recherche.

C'est ici que rentre en jeu le *bandit manchot multibras dynamique* (*Dynamic Multi-armed Bandit*, DMAB) [Da Costa *et al.*, 2008] qui marie le MAB à un test statistique permettant de détecter un changement dans la distribution des récompenses.

Le test statistique en question est le test de Page-Hinkley (TPH) [Page, 1954; Hartland *et al.*, 2006]. Le TPH est une technique d'analyse séquentielle permettant de détecter un changement abrupt de la moyenne d'un signal Gaussien. Ce test fonctionne grâce à une variable cumulative m_t qui correspond à la somme cumulée des différences entre les valeurs observées et leur moyenne du moment :

$$m_t = \sum_{l=1}^{t} (u_t - \overline{u}_t + \delta) \tag{2.9}$$

où $\overline{u}_t$ correspond à la moyenne des valeurs $(u_1, \ldots, u_t)$ et δ correspond à l'ampleur du changement autorisé, c'est-à-dire à la robustesse du test lorsque les changements se produisent lentement. Afin de détecter les changements, la valeur maximum $M_t = \max_{l=1,\ldots,t} u_l$ est calculée et la différence $PH_t = M_t - m_t$ est surveillée. Lorsque cette différence est supérieure à un seuil λ, qui défini le compromis entre les taux admissibles de faux positifs et de faux négatifs, l'hypothèse nulle est rejetée c'est-à-dire que le TPH est en mesure de conclure qu'un changement s'est produit.

Le DMAB fonctionne strictement de la même façon que le MAB mais lorsque, pour un opérateur *op*, $PH_{t,op} > \lambda$, le MAB ainsi que les variables du TPH sont réinitialisées.

2.4 Conclusion

Dans ce chapitre nous avons présenté la notion de recherche autonome qui englobe et fédère différentes approches pour résoudre des problèmes en minimisant l'intervention de l'utilisateur grâce à l'automatisation de certains processus. Un autre objectif est de concevoir des méthodes qui puissent être utilisées facilement sur des classes plus ou moins large de problèmes.

Les approches hors ligne, qui regroupent essentiellement les méthodes de réglage d'algorithmes, ont été présentées à travers trois exemples de méthodes employant différentes philosophies. Ces méthodes sont utiles pour optimiser la paramétrisation, avant l'exécution, d'algorithmes qui n'intègrent pas de stratégie adaptative et ainsi permettre de les utiliser plus facilement. Bien entendu les méthodes hors ligne peuvent également être utilisées pour régler des approches autonomes qui conservent elles-mêmes des paramètres. C'est d'ailleurs ce que nous ferons dans nos contributions puisque les méthodes que nous proposons et celles avec lesquelles nous nous comparons comportent des paramètres.

Le réglage implique d'explorer efficacement l'espace des paramètres. Toutefois, pour chaque paramétrisation il est nécessaire d'exécuter l'algorithme à optimiser et le temps d'exécution est incompressible. Même si le réglage réduit drastiquement le temps total d'optimisation des paramètres par rapport à une approche exhaustive, il n'en reste pas moins que cela demeure une initiative coûteuse.

Le cœur de ce chapitre a été dédié aux approches en ligne. Celles-ci essaient d'adapter le comportement de l'algorithme en fonction des caractéristiques du problème et de la région du paysage de recherche dans laquelle la recherche se déroule. Nous emploierons la même démarche pour nos contributions.

Différentes approches de recherche locale réactive, qui intègrent des mécanismes adaptatifs, ont été évoquées. Nous avons présenté les hyperheuristiques, des métaheuristiques pour gérer des (méta)heuristiques.

Les différents concepts de la sélection adaptative d'opérateurs, initialement apparue dans le contexte évolutionnaire, ont été présentés. Cette partie sur la sélection adaptative d'opérateurs est importante car il s'agit de notre objectif premier dans ce travail de thèse. Nous avons également décrit trois méthodes de sélection. Par la suite nous utiliserons la plus simple de ces méthodes, la sélection proportionnelle, dans nos contributions et nous verrons que malgré sa simplicité elle offre de bons résultats en recherche locale par rapport à des méthodes plus complexes basées sur la concept de « gagnant remporte tout » qui fonctionnent bien dans les algorithmes évolutionnaires.

Bien évidemment il existe de nombreuses méthodes de sélection d'opérateurs, qui d'ailleurs parfois ne disent pas leur nom. Nous avons choisi de n'en mentionner que quelques unes que nous considérons être représentatives et récentes. Pour plus d'informations [Fialho, 2010; Hamadi *et al.*, 2012] présentent un panorama plus détaillé de ces approches.

Chapitre 3

Recherche locale basée sur les contraintes

Sommaire

3.1 Introduction 44
3.2 Panorama des systèmes proposés 44
3.2.1 Algorithmes 44
3.2.2 Langages et bibliothèques 44
3.2.3 Solveurs boîtes noires 45
3.3 Comet 45
3.3.1 Langage et concepts 45
3.3.2 Contraintes globales 48
3.3.3 Contraintes et violations 49
3.3.4 Discussion 51
3.3.5 Contributions et feedback 51
3.4 Conclusion 52

3.1 Introduction

En programmation par contraintes, les contraintes servent à élaguer l'espace de recherche en réduisant le domaine de chaque variable. La recherche locale basée sur les contraintes (*Constraint-Based Local Search*, CBLS) allie la recherche locale aux contraintes en intégrant ces dernières à la fonction objectif. Il ne s'agit alors pas uniquement de considérer la valeur de vérité des contraintes mais également leur taux de violation. Ceci implique de définir des fonctions de pénalité ou de violation pour chaque contrainte.

Les systèmes plus avancés de CBLS apportent également des techniques inhérentes à la programmation par contraintes à la recherche locale telles que les variables incrémentales et la propagation. Les variables incrémentales permettent d'évaluer des expressions incrémentalement c'est-à-dire sans avoir à recalculer l'expression complète lorsque qu'un élément de l'expression change. De même, lorsque qu'une valeur d'une variable incrémentale change, la propagation permet de prendre en compte ce changement dans toutes les expressions qui utilisent la variable dont la valeur a été modifiée.

Dans ce chapitre nous présentons différents systèmes CBLS et nous mettons l'emphase sur COMET que nous utilisons pour cette thèse.

3.2 Panorama des systèmes proposés

Parmi les systèmes CBLS, du point de vue de l'utilisateur final, il est possible de distinguer trois grandes catégorie de solveurs :

1. des algorithmes relativement simples qui implémentent des techniques de résolutions ; à charge à l'utilisateur de mettre en œuvre ces techniques et de les adapter aux problèmes à traiter ;
2. des langages de programmation dédiés ou des bibliothèques pour des langages de programmation existants qui, comme les solveurs de la catégorie précédente, prennent en entrée la modélisation d'un problème mais permettent à l'utilisateur de construire son mécanisme de recherche ;
3. des solveurs permettant la modélisation d'un problème et capables de résoudre ce problème selon un processus de type « boîte noire ».

3.2.1 Algorithmes

Parmi les premières approches CBLS, et dans la première catégorie, on retrouve celles de Codognet et Diaz [Codognet et Diaz, 2001] et de Galinier et Hao [Galinier et Hao, 2004] qui se basent sur la recherche tabou avec l'objectif de résoudre des problèmes de satisfaction de contraintes. À chaque type de contrainte est associée une fonction de pénalité. Les contraintes du problème permettent alors d'obtenir une somme pondérée ou non des fonctions de pénalité. On cherche à minimiser cette somme qui est en fait la fonction objectif de la recherche tabou. [Galinier et Hao, 2004] définit la notion de variable critique. Une variable est critique si, lorsqu'elle est associée à une contrainte, un changement de valeur de cette variable peut diminuer la pénalité associée de la contrainte.

3.2.2 Langages et bibliothèques

Van Hentenryck et Michel ont d'abord proposé Localizer [Michel et Van Hentenryck, 2000], un langage de modélisation associé à une bibliothèque C++, avant de lancer COMET [Van Hentenryck et Michel, 2005], un langage de programmation

dédié à la recherche locale et à la programmation par contraintes. Contrairement aux approches évoquées précédemment, en plus des problèmes de satisfaction de contraintes, ces systèmes proposent aussi de résoudre des problèmes d'optimisation « quelconques ».

Kangaroo [Newton *et al.*, 2011] est une alternative à COMET qui a été proposée récemment. Contrairement à COMET, Kangaroo fonctionne sur le principe de la propagation paresseuse. Les résultats expérimentaux semblent montrer que ces choix de mise en œuvre apportent de réels gains en terme de temps d'exécution par rapport à COMET. Toutefois, malgré le souhait de ses concepteurs de publier ce logiciel sous licence libre, Kangaroo n'est pas encore disponible au public.

3.2.3 Solveurs boîtes noires

LocalSolver [Benoist *et al.*, 2011] est, à notre connaissance, le seul solveur commercial « clé en main » disponible à ce jour. En effet, l'utilisateur n'a qu'à fournir la modélisation de son problème au logiciel pour obtenir une solution. LocalSolver ne traite que les problèmes à variables booléennes bien que ses concepteurs souhaitent passer aux variables entières. En tant que logiciel commercial non libre, les détails de son fonctionnement ne sont pas clairement connus.

Par défaut, LocalSolver effectue une descente en utilisant différents mouvements de recherche locale. Un mécanisme de recuit simulé peut également être sélectionné via la ligne de commande. Il utilise des mouvements permettant de cibler des variables de décision dont la modification est susceptible d'améliorer la qualité de la solution. Plusieurs exécutions (au moins deux) sont faites en parallèle avec différentes graines pseudo-aléatoires. Les mouvements sont choisis aléatoirement sur la base d'une distribution non uniforme. Celle-ci évolue en fonction de leurs taux d'acceptation et d'amélioration.

Par ailleurs LocalSolver n'est pas conçu pour résoudre des problèmes fortement contraints. Il appartient à l'utilisateur de déterminer comment seront interprétées les violations des contraintes en les intégrant à la fonction objectif. Ceci rend donc la modélisation plus complexe et le paramétrage, à travers d'éventuelles pondérations, plus important.

3.3 Comet

Ce travail de thèse utilise principalement COMET comme environnement hôte. À ce titre, et puisque COMET n'est pas forcément connu de tous, nous présentons COMET dans cette section.

3.3.1 Langage et concepts

L'architecture de COMET s'articule autour du concept d'*invariants* sur les expressions algébriques et ensemblistes. Les invariants sont exprimés en termes de variables incrémentales. Ils spécifient une relation qui doit être conservée lorsque de nouvelles valeurs sont affectées à ses variables. L'utilisateur déclare un invariant simplement en spécifiant la relation à maintenir incrémentalement, et non comment la conserver à jour. Ceci est géré par COMET qui détermine un ordre topologique dans lequel mettre à jour les invariants. L'exemple de la Figure 3.1 définit un invariant `s` à valeur entière dans le contexte du solveur `m` qui permettra d'accéder à la somme des valeurs du tableau `a`.

```
int a[1..100] = 0;
var{int} s(m) <- sum(i in 1..100) a[i];
cout << "Somme␣:␣" << s << endl; // Somme : 0
a[50] = 5;
cout << "Somme␣:␣" << s << endl; // Somme : 5
```

FIGURE 3.1 – Exemple d'invariant en COMET.

```
interface Constraint {
    var{int}[] getVariables();
    var{bool} isTrue();
    var{int} violations();
    var{int} violations(var{int} x);
    int getAssignDelta(var{int} x, int v);
    int getSwapDelta(var{int} x1, var{int} x2);
    int getAssignDelta(var{int}[] x, int[] v);
    ...
}

interface Objective {
    var{int} evaluation();
    var{int} increase(var{int} x);
    var{int} decrease(var{int} x);
    var{int} flipDelta(var{bool} x);
    int getAssignDelta(var{int} x,int v);
    int getSwapDelta(var{int} x,var{int} y);
    void post();
    var{int}[] getVariables();
    ...
}
```

FIGURE 3.2 – Les méthodes principales des interfaces `Constraint` et `Objective`.

Le delta (δ) d'une fonction objectif est un concept important en recherche locale afin d'évaluer efficacement les mouvements. En effet, au lieu de recalculer toute une fonction lorsqu'une valeur d'une variable est modifiée, il est souvent préférable de ne recalculer que la différence (le δ). C'est ici que rentre en jeu le concept d'*objet différentiable* qui utilise les invariants afin de maintenir un certain nombre de propriétés.

Les objets différentiables se divisent en deux classes principales : les contraintes et les objectifs (Figure 3.2). Les propriétés maintenues pour les contraintes sont, par exemple, sa satisfiabilité ou ses violations. Il est possible de demander à un objet différentiable de type `Constraint` quel sera l'effet d'un mouvement sur telle ou telle propriété par l'intermédiaire des méthodes `getAssignDelta` et `getSwapDelta`. Les propriétés maintenues pour les objectifs sont la fonction objectif elle-même et éventuellement une fonction d'évaluation permettant de discriminer deux solutions ayant la même valeur. Notons que, contrairement à `Constraint`, `Objective` ne gère pas la méthode `getAssignDelta` pour les tableaux. Par ailleurs, l'utilisateur peut également créer de nouvelles contraintes et de nouveaux objectifs en se basant sur les interfaces appropriées.

Ces différents objets sont manipulables à l'aide des constructions et des abs-

```
1  int n = 8; range Size = 1..n;
2  LocalSolver m();
3  UniformDistribution distr(Size);
4  var{int} queen[Size](m,Size) := distr.get();

5  ConstraintSystem S(m);
6  S.post(alldifferent(queen));
7  S.post(alldifferent(all(i in Size) queen[i] + i));
8  S.post(alldifferent(all(i in Size) queen[i] - i));
9  m.close();

10 int it = 0;
11 while (S.violations() > 0 && it < 50 * n) {
12     selectMax(q in Size)(S.violations(queen[q]))
13         selectMin(v in Size)(S.getAssignDelta(queen[q],v))
14             queen[q] := v;
15     it = it + 1;
16 }
```

FIGURE 3.3 – Exemple de résolution des n-reines en COMET.

tractions de haut niveau que sont les *sélecteurs*, les *voisinages*, les *solutions* et les *points de reprise* (*checkpoints*). Ces derniers permettent d'écrire un programme de recherche locale qui se lit comme l'algorithme sur lequel il est basé.

Grâce à COMET il est possible, et conseillé, de dissocier la modélisation du problème de sa résolution, les deux pouvant être clairement séparés. Ceci facilite la modification de l'un sans affecter l'autre.

Prenons pour exemple le problème classique des n-reines. Ce problème implique de placer n reines sur un échiquier de taille $n \times n$ de sorte qu'il y ait une seule reine sur chaque rangée, colonne et diagonale. La Figure 3.3 présente le programme COMET. Celui-ci utilise une recherche locale qui cherche, à chaque itération, à minimiser les violations de la variable ayant le plus de violations. Les lignes 5-9 définissent le modèle. Les lignes 10-16 correspondent à la procédure de recherche. Un tableau de variables incrémentales (ligne 4) représente la solution. Puisque ce sont des variables incrémentales, l'affection à la ligne 14, sera automatiquement propagée et les violations des contraintes mises à jour.

Un exemple d'utilisation d'un sélecteur `MinNeighborSelector` et un exemple de la construction `neighbor(`δ`,N){M}` sont présentés à la Figure 3.4. Cette construction conserve les mouvements `M` dont la valeur δ respectent `N`. Dans l'exemple, le seul mouvement qui sera conservé sera celui ayant le plus petit δ puisque `N` sélectionne la plus petite valeur. Notons que l'opération `M`, généralement coûteuse, n'est exécutée qu'une seule fois lors de l'appel `call` alors que le calcul δ, théoriquement moins coûteux, se fait pour chaque voisin examiné.

Dans la sous-section suivante nous présentons quelques contraintes globales disponibles dans COMET et qui nous seront utiles pour décrire les problèmes du chapitre suivant.

```
1  MinNeighborSelector N();
2  while (S.violations() > 0) {
3      forall (i in X, j in X) {
4          d = S.getSwapDelta(x[i], x[j]);
5          neighbor(d,N) {
6              x[i] :=: x[j]; // échange des valeurs
7          }
8      }
9      if (N.hasMove())
10         call (N.getMove());
11     it++;
12 }
```

FIGURE 3.4 – Exemple d'utilisation d'un sélecteur et de la construction `neighbor`.

3.3.2 Contraintes globales

Alldifferent

La contrainte globale `alldifferent`(x) [Laurière, 1978; Régin, 1994] est vérifiée si toutes les valeurs des variables de la collection de m variables $x = [i_1, \ldots, i_m]$ sont distinctes. Cette contrainte correspond à l'union des inégalités entre tous les couples de variables de x.

`alldifferent`(x) ssi

$$\forall i \in \{1, \ldots, m\}, \forall j \neq i \in \{1, \ldots, m\}, x_i \neq x_j \tag{3.1}$$

Atmost

La contrainte globale `atmost`(n,x,v) est vérifiée s'il y a au plus n variables de la collection de m variables $x = [x_1, \ldots, x_m]$ dont la valeur est v.

`atmost`(n,x,v) ssi

$$\sum_{i=1}^{m} \natural(x_i = v) \leq n \tag{3.2}$$

où $\natural(b) = 1$ si b sinon 0.

Cycle

La contrainte globale `cycle`(n,v) [Beldiceanu et Contejean, 1994] est vérifiée s'il existe exactement n cycles dans la permutation donnée par le vecteur $v = \langle v_1, \ldots, v_n \rangle$.

`cycle`(n,v) ssi

$$\forall i \in \{1, \ldots, m\}, 1 \leq v_i \leq m \tag{3.3a}$$

$$\forall i \in \{1, \ldots, m\}, \forall j \neq i \in \{1, \ldots, m\}, v_i \neq v_j \tag{3.3b}$$

$\forall i \in \{1, \ldots, m\}$,

Soit c_i l'ensemble d'entiers défini de la façon suivante :

$i \in c_i$, si $j \in c_i$ alors $v_j \in c_i$

alors le schéma précédent défini exactement n ensembles distincts. (3.3c)

Multiknapsack

Dans la contrainte global `multiknapsack`(a,s,c) [Shaw, 2004], le vecteur $c = \langle c_1, \ldots, c_m \rangle$ représente les capacités de m sacs, le vecteur $s = \langle s_1, \ldots, s_n \rangle$ représente la taille de n items à placer dans les sacs et le vecteur $a = \langle a_1, \ldots, a_n \rangle$ représente, pour chaque item, l'indice du sac dans lequel il sera placé. La contrainte est vérifiée lorsque les items sont placés dans les sacs en respectant la capacité de chaque sac.

`multiknapsack`(a,s,c) ssi

$$\forall i \in \{1, \ldots, m\}, \sum_{\{j | j \in \{1,\ldots,n\} \wedge a_j = i\}} s_j \leq c_i \tag{3.4}$$

3.3.3 Contraintes et violations

Un des intérêts majeurs de COMET et d'un CBLS en général, est que les contraintes sont déjà associées à des fonctions de violations. Celles-ci sont utilisées afin de diriger la recherche locale.

Nous présentons ici les fonctions de violations associées aux contraintes dans COMET. Le Tableau 3.1 présente la notation utilisée pour décrire les contraintes et leurs fonctions de violations (Tableaux 3.2 et 3.3).

Notation	Définition
a	un tableau de variables incrémentales
c	une contrainte
v	un entier
V	un ensemble d'entiers
$Var(c)$	les variables de c
$range(a)$	les indices de a
$\tau_\alpha(c)$	la valeur de vérité de c pour l'affectation α
$v_\alpha(c)$	les violations de c pour l'affectation α
$\natural(b)$	si b alors 1 sinon 0
$dom(a)$	$\cup\{dom(a[i]) \mid i \in range(a)\}$ l'union des domaines des variables incrémentales de a
$\#_\alpha(v, a)$	$\#\{i \in range(a) \mid \alpha(a[i]) = v\}$ le nombre d'occurrences de v dans le tableau a pour l'affectation α
$\#_\alpha(V, a)$	$\#\{i \in range(a) \mid \alpha(a[i]) \in V\}$ le nombre d'occurrences d'un élément de l'ensemble V dans le tableau a pour l'affectation α
$\rho_\alpha(v, a)$	$\{i \in range(a) \mid \alpha(a[i]) = v\}$ l'ensemble des indices de tableau de a dont les variables sont égales à v pour l'affectation α

TABLEAU 3.1 – Notations associées à la description des contraintes et de leurs violations.

Contrainte	Violation
$l = r$	$\lvert l - r \rvert$
$l \neq r$	1
$l \leq r$	$\max(l - r, 0)$
$l < r$	$\max(l - r + 1, 0)$
$alldifferent(a)$	$\sum_{v \in dom(a)} \max(\#_\alpha(v, a) - 1, 0)$
$atmost(n, a)$	$\sum_{v \in range(a)} \max(\#_\alpha(v, a) - n[v], 0)$
$atleast(n, a)$	$\sum_{v \in range(a)} \max(n[v] - \#_\alpha(v, a), 0)$
$multiknapsack(a, w, n)$	$\sum_{v \in range(a)} \max\left(\sum_{i \in \rho_\alpha(v,a)} w[i] - n[v], 0\right)$
$sequence(a[1 \dots n], p, q, V)$	$\sum_{i \in 1 \dots n-q+1} \max(\#_\alpha(V, a[i \dots i+q-1]) - p, 0)$
$opposite(l, r)$	1
$nonopposite(l, r)$	1

TABLEAU 3.2 – Contraintes et leur fonction de violation associée. $opposite(l, r)$ est définie par $l = m \Rightarrow r = -m || l = r = 0$ et $nonopposite(l, r)$ par $l = m \Rightarrow r \neq -m || !(l = r = 0)$.

Combinateur	Violation
$c_1 \wedge c_2$	$v_\alpha(c_1) + v_\alpha(c_2)$
$c_1 \vee c_2$	$\min(v_\alpha(c_1), v_\alpha(c_2))$
$atmost(k, [c_1, \dots, c_n])$	$\max\left(\sum_{i=1}^{n} \natural(\tau_\alpha(c_i)) - k, 0\right)$
$atleast(k, [c_1, \dots, c_n])$	$\max\left(k - \sum_{i=1}^{n} \natural(\tau_\alpha(c_i)), 0\right)$
$exactly(k, [c_1, \dots, c_n])$	$\lvert k - \#\{c_i \mid 1 \leq i \leq n, \tau_\alpha(c_i)\} \rvert$
$k \times c$	$k \times v_\alpha(c)$
$satisfaction(c)$	$\natural(\tau_\alpha(c))$

TABLEAU 3.3 – Combinaisons de contraintes et leur fonction de violation associée.

3.3.4 Discussion

D'un point de vue pratique, la documentation de COMET est de bonne qualité mais plusieurs fonctions ne sont pas documentées et toutes les informations disponibles ne se retrouvent pas dans la documentation distribuée avec le langage. Il faut alors consulter les publications associées, notamment le livre [Van Hentenryck et Michel, 2005].

Utiliser COMET nous a amené à rencontrer certaines difficultés. Les sélecteurs sont des constructions et non des objets. Ceci rend impossible la création de nouveaux sélecteurs utilisables de la même façon que les sélecteurs d'origine. Sur les contraintes, il est possible de demander l'évaluation incrémentale pour des affectations sur plusieurs variables. L'utilisateur n'a ainsi pas à donner une fonction delta spécifique. Ceci n'est pas le cas pour les objectifs où l'évaluation incrémentale est limitée à l'affectation d'une seule variable. L'utilisateur doit alors forcément définir les fonctions delta s'il utilise des opérateurs légèrement complexes.

Une spécificité de COMET qui est peu commode pour son utilisation conjuguée avec d'autres outils d'optimisation, notamment les outils de configuration automatique, est la façon dont est géré le pseudo aléatoire. En effet, contrairement aux outils classiques qui prennent une graine en entrée, COMET ne fonctionne pas sur ce principe et génère ses propres graines sans contrôle de l'utilisateur. Il est possible de rendre une exécution d'un programme COMET déterministe ou de conserver les graines utilisées par COMET afin de dupliquer une exécution. Toutefois, dans ce cas COMET produit, pour une même exécution, un fichier contenant plusieurs nombres correspondant apparemment aux graines utilisées à chaque fois qu'une fonction pseudo aléatoire a été utilisée. Ce fichier ne peut être réutilisé que sur le même programme, on ne peut donc pas conserver les mêmes graines pseudo aléatoires sur plusieurs programmes différents. Pour parer à cette lacune nous avons donc codé de nouvelles fonctions pseudo aléatoires ce qui nous a amené à réécrire certaines fonctionnalités de COMET tels que les sélecteurs.

3.3.5 Contributions et feedback

COMET est un environnement et un langage relativement récent. De ce fait, nous avons eu l'occasion de participer au sein de la communauté des utilisateurs [1] à l'amélioration et au débogage de COMET.

Nous avons suggéré de pouvoir utiliser les types énumérés dans une instruction `switch` qui permet d'effectuer un branchement à partir de la valeur d'une variable. Cette suggestion a été intégrée dans la version 2.1.

Nous avons signalé que cette même version a introduit un bogue où les opérateurs d'incrémentation et de décrémentation préfixés ne fonctionnaient plus sur les tableaux (`++tab[0]`). Un autre dysfonctionnement mineur que nous avons trouvé est l'occurrence d'erreurs de segmentation lorsqu'une instruction `switch` est vide.

Un bogue plus sérieux trouvé est le fonctionnement inattendu d'un opérateur d'affection augmentée pour mettre à jour une valeur d'un tableau dont l'indice est une fonction. L'instruction `a[fct()] += 1` est en fait interprétée comme `a[appel 2 à fct()] = a[appel 1 à fct()] + 1`.

1. `http://forums.dynadec.com/`

3.4 Conclusion

Dans ce chapitre nous avons présenté différents systèmes de recherche locale fonctionnant avec des contraintes. Ces systèmes sont peu nombreux. Ils demandent en effet un important effort de mise en œuvre afin de gérer intelligemment les contraintes et la propagation des changements. C'est pour cette raison que nous avons pris le parti de ne pas créer le notre mais d'utiliser COMET qui est le système le plus avancé à ce jour. De plus, COMET s'inscrit dans un effort pour faciliter le développement d'algorithmes simples, clairs et transparents pour l'utilisateur. Nous partageons cet objectif. L'autre avantage de la recherche locale par rapport à la programmation par contraintes classique est qu'il est possible de traiter des problèmes de taille supérieure.

Nous avons présenté, au sein de COMET, certains concepts utiles pour l'utilisateur de CBLS et, plus généralement, le programmeur, en les illustrant à travers des morceaux de code. Ce chapitre énumère aussi les différentes contraintes dans COMET et leurs fonctions de violations. Nous avons également fait part des difficultés rencontrées dans l'utilisation du langage et que nous avons communiquées aux concepteurs de COMET afin de contribuer à son amélioration.

Chapitre 4

Problèmes

Sommaire

4.1 One Max 54
4.2 Problème d'affectation quadratique 54
4.2.1 Représentations et structures de données 55
4.2.2 Jeux de données 55
4.2.3 Méthodes de résolution 56
4.3 Problème du voyageur de commerce 57
4.3.1 Représentations et structures de données 57
4.3.2 Mouvements pour la recherche locale 58
4.3.3 Jeux de données 59
4.3.4 Méthodes de résolution 59
4.4 Progressive party problem 60
4.4.1 Problème d'optimisation 60
4.4.2 Problème de satisfaction de contraintes 61
4.4.3 Méthodes de résolution 62
4.5 Conclusion 62

Ce chapitre présente les problèmes qui seront utilisés pour tester les approches proposées aux Chapitres 5, 6 et 7.

Le *One Max* est un problème très simple qui permet une première validation des méthodes de sélection d'opérateurs.

Le problème d'affectation quadratique et le problème du voyageur de commerce (asymétrique) sont des problèmes classiques de permutation. Nous les avons choisis puisqu'ils étaient facile à représenter, par un vecteur, en ayant toutefois chacun une sémantique différente. Ceci nous permettra de tester l'opportunité d'employer les mêmes voisinages et les mêmes paramétrisations afin d'évaluer la robustesse des approches.

Enfin nous utiliserons le *progressive party problem* pour tester la résolution d'un problème de satisfaction de contraintes. Ce problème est souvent utilisé dans le contexte de la recherche locale avec des contraintes.

4.1 One Max

Le problème du *One Max* [Ackley, 1987] est parfois décrit comme la « drosophile » des algorithmes évolutionnaires. Le problème du *One Max* demande simplement de maximiser le nombre de 1 dans une chaîne de valeurs binaires.

Soit X_n l'espace de recherche de toutes les chaînes binaires de longueur n, c'est-à-dire l'ensemble de tous les $x = x_1, x_2, \ldots, x_n$, $x_i \in \{0, 1\}$, le problème du *One Max* est alors défini formellement par

$$max_{x \in X_n} \quad f(x) = \sum_{i=1}^{n} x_i \tag{4.1}$$

Bien qu'il soit trivial à résoudre, le problème du *One Max* peut être utilisé afin d'évaluer l'aptitude d'un mécanisme de sélection d'opérateur à faire de bons choix, et ce plus particulièrement puisque les résultats expérimentaux peuvent être comparés aux résultats théoriques attendus. Comme ce problème est simple, il est possible de calculer exactement la probabilité pour qu'une solution spécifique soit améliorée par un opérateur donné [Candan *et al.*, 2012].

Comme les variables du *One Max* sont binaires, les mouvements sont des *flips* – passage de 0 à 1 et inversement. Plusieurs opérateurs simples peuvent être créés en appliquant un nombre différent de *flips*. On parle alors de *k-flips* si k variables sont en jeu.

4.2 Problème d'affectation quadratique

Le *problème d'affectation quadratique* (*Quadratic Assignment Problem*, QAP) [Koopmans et Beckmann, 1957] modélise le problème qui consiste à trouver le coût minimum lorsque n unités de productions sont placées dans n localisations. Pour chaque paire de localisations, une distance est spécifiée entre elles et à chaque paire d'unités de production correspond un flot, c'est-à-dire la quantité de matériaux/-produits transportée entre les deux unités. Le coût est alors donné par la somme de tous les produits distance-flot possibles. Ce problème est NP-difficile et est l'un des problèmes fondamentaux de l'optimisation combinatoire [Garey et Johnson, 1990]. De ce fait, le QAP a été largement étudié [Loiola *et al.*, 2007].

Soient $A = (a_{i,j})$ et $B = (b_{i,j})$ deux matrices $n \times n$ à valeurs réelles, où $a_{i,j}$ (resp. $b_{i,j}$) est la distance (resp. le flot) entre les lieux (resp. les unités) i et j. Alors, la fonction objectif est donnée par

$$\min_{\pi \in \Pi_n} \quad f(\pi) = \sum_{i=1}^{n} \sum_{j=1}^{n} a_{\pi_i, \pi_j} b_{i,j} \tag{4.2}$$

4.2.1 Représentations et structures de données

Dans la pratique, il n'existe pas de structure de données « permutation » faisant respecter les contraintes inhérentes aux permutations. Il faut donc utiliser des structures de données plus classiques et intégrer des contraintes au problème ou choisir des opérateurs qui ne violent pas les contraintes d'une permutation. Nous présentons ici les deux représentations les plus fréquemment utilisées. D'autres représentations sont décrites dans [Loiola *et al.*, 2007].

Vecteur d'entiers

Une permutation peut être représentée par un vecteur v, toutefois il convient d'ajouter deux contraintes : une pour les domaines des variables et une pour que toutes les valeurs soient distinctes. Ainsi donc :

$$\min \quad f(v) = \sum_{i=1}^{n} \sum_{j=1}^{n} a_{v_i,v_j} b_{i,j} \tag{4.3a}$$

$$\text{sous contraintes} \quad \texttt{alldifferent}(v) \tag{4.3b}$$

$$v_i \in \{1, \ldots, n\} \qquad 1 \leq i \leq n \tag{4.3c}$$

La contrainte 4.3b spécifie que toutes les valeurs du vecteur v sont différentes. La contrainte `alldifferent` est explicitée dans la Section 3.3.2.

Notons que si les seuls opérateurs utilisés pour résoudre ce problème sont des opérateurs qui conservent les propriétés des permutations, par exemple les k-échanges, alors les contraintes du problème sont redondantes et il n'y a pas lieu de les vérifier, ce qui simplifie la recherche. C'est ce choix que nous avons fait pour nos expériences.

Matrice binaire

Une autre représentation souvent utilisée, notamment en programmation linéaire en nombre entier, est une matrice binaire $n \times n$, $X = (x_{i,j})$ où $x_{i,j} = 1$ si l'unité de production j est placée à la localisation i.

$$\min \quad f(X) = \sum_{i,j=1}^{n} \sum_{k,l=1}^{n} a_{i,j} b_{k,l} x_{i,k} x_{j,l} \tag{4.4a}$$

$$\text{sous contraintes} \quad \sum_{i=1}^{n} x_{i,j} = 1 \qquad 1 \leq j \leq n \tag{4.4b}$$

$$\sum_{j=1}^{n} x_{i,j} = 1 \qquad 1 \leq i \leq n \tag{4.4c}$$

$$x_{i,j} \in \{0, 1\} \qquad 1 \leq i, j \leq n \tag{4.4d}$$

4.2.2 Jeux de données

La principale source de jeux de données pour le QAP, qui regroupe des instances proposées par différents auteurs est la bibliothèque d'instances et de résultats QAPLIB [Burkard *et al.*, 1997] disponible en ligne. [1]

Les instances sont accompagnées de la solution optimum ainsi que de la valeur de la fonction objectif, si elles sont connues, ou de la meilleure solution connue et sa valeur (*Best Known Value*, BKV) si l'optimum est inconnu. Dans ce dernier cas, la métaheuristique utilisée pour déterminer la BKV est mentionnée.

1. `http://www.seas.upenn.edu/qaplib/`

Les métaheuristiques utilisées sont variées et regroupent des algorithmes datant de 1984 à 2008 au moment de l'écriture de ce document. Il est intéressant de noter que la métaheuristique ayant trouvé le plus de BKV est la méthode tabou robuste (RoTS) [Taillard, 1991] qui n'est pourtant pas la méthode la plus récente. Nous décrivons RoTS plus loin et nous l'utiliserons à des fins de comparaison dans les Chapitres 5 et 6.

4.2.3 Méthodes de résolution

Le QAP est un des problèmes les plus étudiés en optimisation combinatoire. Il est donc difficile d'être exhaustif et nous ne mentionnerons que quelques métaheuristiques utilisées afin de résoudre ce problème.

Dans [Loiola *et al.*, 2007], les auteurs de cet article de synthèse dressent un panorama exhaustif des différentes méthodes de résolution, et notamment des métaheuristiques. Ainsi le QAP a été résolu par recuit simulé [Misevičius, 2003], algorithme génétique [Drezner, 2005a], colonie de fourmis [Acan, 2005], recherche tabou [Taillard, 1991; Drezner, 2005b], recherche locale itérée [Stützle, 2006] ou encore par recherche tabou itérée [Misevicius, 2011].

L'article précédent présente de très bons résultats récents en parvenant à améliorer deux meilleures solutions connues sur des problèmes QAPLIB. Notons que cet article, même s'il a été écrit vingt ans plus tard, se compare toujours à [Taillard, 1991]. Ce dernier parvient à se comporter très honorablement par rapport aux algorithmes plus récents.

RoTS – Tabou robuste

Comme son nom l'indique la méthode *tabou robuste* (*Robust Taboo*, RoTS) [Taillard, 1991] est une variante de la recherche tabou (Section 1.5.3) spécifique au QAP.

RoTS utilise le mouvement 2-échange et une liste tabou de longueur variable. La liste tabou est construite de la façon suivante : pour chaque unité et chaque localisation, la dernière itération pendant laquelle une unité occupait une localisation est mémorisée. Un mouvement est tabou s'il affecte les unités échangées à des localisations qu'elles avaient occupées pendant les s plus récentes itérations.

Les détails de la mise en œuvre de cette métaheuristique correspondent à la version datée de 2006 distribuée par Taillard [2] qui diffère légèrement de la version proposée dans l'article cité, notamment pour la valeur des paramètres.

La recherche locale utilisant le 2-échange sélectionne le meilleur voisin. Le critère d'aspiration est employé dans sa version classique, c'est-à-dire qu'un voisin tabou est autorisé s'il améliore la meilleure solution trouvée jusqu'à présent. Toutefois, si une des deux affectations d'unité à une localisation n'a pas eu lieu pendant les $5n^2$ dernières itérations, le mouvement est aspiré.

La longueur s est choisie aléatoirement à chaque itération, $s = \lfloor 8nr^3 \rfloor$ où $r \in [0, 1]$ est une valeur pseudo-aléatoire.

2. `http://mistic.heig-vd.ch/taillard/`

4.3 Problème du voyageur de commerce

Tout comme le QAP, le *problème du voyageur de commerce* (*Traveling Salesman Problem*, TSP) [3] est un problème fondamental de l'optimisation combinatoire [Garey et Johnson, 1990]. Il est NP-difficile. Il s'agit de trouver un circuit de coût minimum permettant de visiter une, et une seule fois, chaque ville d'un ensemble de villes. En d'autres termes, il s'agit de trouver le circuit hamiltonien de coût minimum dans un graphe. Dans le problème le plus fréquemment considéré, que nous nommerons STSP, les coûts (généralement des distances) entre deux villes sont symétriques, c'est-à-dire que le coût de la ville A à la ville B est le même que le coût de la ville B à la ville A. Toutefois, en pratique, il peut arriver que ces coûts ne soient pas égaux : on peut penser à des rues en sens unique ; ainsi le trajet dans un sens n'est pas le même que dans l'autre et les distances sont différentes. De plus, il est possible de considérer autre chose que la distance, par exemple le coût financier, qui lui aussi n'est pas forcément égal d'un sens à l'autre. Cette variante du TSP est appelée *problème du voyageur de commerce asymétrique* (*Asymmetric Traveling Salesman Problem*, ATSP). Par convention, les distances sont exprimées par des entiers.

Soient un graphe orienté complet $G = (S, A)$ avec $S = \{1, \ldots, n\}$ l'ensemble des sommets de G, A ses arcs et $D = (d_{i,j})$ la matrice $n \times n$ à valeurs entières, où $d_{i,j}$ est le poids de l'arc (i, j). Alors la fonction objectif de l'ATSP est

$$\min_{\pi \in \Pi_n} \quad f(\pi) = \sum_{i=1}^{n-1} d_{\pi_i, \pi_{i+1}} + d_{\pi_n, \pi_1} \tag{4.5}$$

4.3.1 Représentations et structures de données

Comme pour le QAP, il existe plusieurs possibilités de représenter le TSP. Nous en présentons trois.

Chemin

Le circuit solution peut être représenté par un vecteur v d'indices de taille n représentant directement le cheminement à suivre, c'est-à-dire que v_i précède directement v_j, par exemple la solution $[4, 3, 1, 2]$ représente le circuit 4-3-1-2-4, alors la fonction objectif de le TSP est

$$\min \quad f(v) = \sum_{i=1}^{n-1} d_{v_i, v_{i+1}} + d_{v_n, v_1} \tag{4.6a}$$

$$\text{sous contrainte} \quad \texttt{alldifferent}(v) \tag{4.6b}$$

$$v_i \in \{1, \ldots, n\} \qquad 1 \leq i \leq n \tag{4.6c}$$

La même remarque que pour le QAP s'applique à cette représentation : si les opérateurs utilisés respecte les contraintes de permutations, alors celles-ci sont redondantes et il n'est pas nécessaire de les vérifier. C'est cette représentation que nous utiliserons.

Vecteur d'adjacence

Une seconde utilisation d'un vecteur, mais avec une sémantique différente, est la représentation du circuit solution par un vecteur d'adjacence v d'indices S, c'est-à-dire que si j est à la position i alors i précède directement j dans le circuit, par

3. L'origine du TSP est relativement floue [Schrijver, 2005]. Le problème, sous d'autres appellations, semble avoir été étudié par les mathématiciens dès la première moitié du XIX^{e} siècle. Toutefois la première référence au problème sous le nom que l'on lui connaît aujourd'hui semble dater de 1949 [Robinson, 1949].

exemple la solution $[4, 3, 1, 2]$ représente le circuit 1-4-2-3-1, alors la fonction objectif du TSP est

$$\min \quad f(v) = \sum_{i=1}^{n} d_{i,v_i} \tag{4.7}$$

$$\text{sous contrainte} \quad \texttt{cycle}(1, v) \tag{4.8}$$

La contrainte 4.8 spécifie qu'il n'existe qu'un seul cycle dans v. La contrainte `cycle` est explicitée dans la Section 3.3.2. Cette contrainte globale intègre la vérification des domaines et des valeurs distinctes.

Matrice binaire

Si l'on considère un modèle binaire, une solution est alors une matrice $n \times n$ à valeurs binaires, $X = (x_{i,j})$, où $x_{i,j} = 1$ si l'arête (i, j) fait partie du circuit et $x_{i,j} = 0$ sinon. La fonction objectif est alors donnée par

$$\min \quad f(X) = \sum_{i=1}^{n} \sum_{j=1}^{n} d_{i,j} x_{i,j} \tag{4.9a}$$

$$\text{sous contraintes} \quad \sum_{i=1}^{n} x_{i,j} = 1, \qquad j \in S \tag{4.9b}$$

$$\sum_{j=1}^{n} x_{i,j} = 1, \qquad i \in S \tag{4.9c}$$

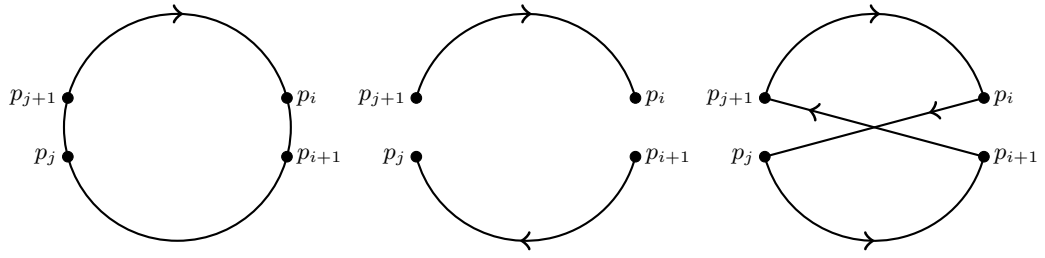
$$\sum_{i \in S'} \sum_{j \notin S'} x_{i,j} \geq 1 \qquad \forall S' \subset S, \quad S' \neq \emptyset \tag{4.9d}$$

Cette formulation permet de voir que le TSP est un cas particulier du QAP. En effet, la matrice de distances et la matrice binaire du TSP correspondent aux matrices de distances et de flots du QAP. Les matrices de distances sont les mêmes et la matrice de flots $B = (b_{i,j})$ est définie par :

$$b_{i,j} = \begin{cases} 1 & \text{si } j = i + 1, 1 \leq i \leq n - 1 \text{ ou } i = n, j = 1 \\ 0 & \text{sinon} \end{cases} \tag{4.10}$$

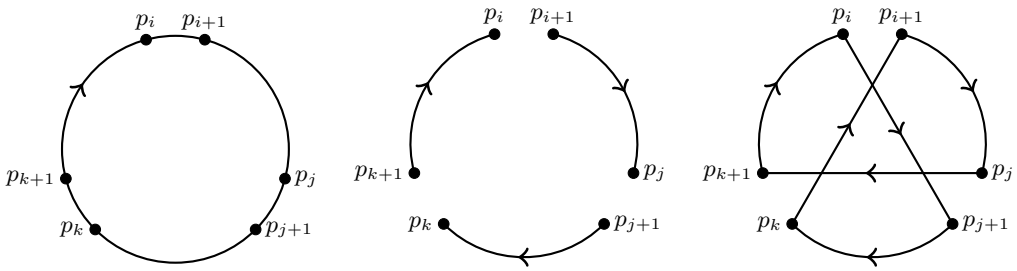
4.3.2 Mouvements pour la recherche locale

Les mouvements de base sur les permutations, comme les échanges, peuvent être utilisés dans une procédure de recherche locale pour résoudre les problèmes de la famille du TSP. Toutefois, le fait de considérer un circuit introduit une sémantique particulière qui permet d'appliquer d'autres mouvements qui prennent en compte cette caractéristique.



(a) Circuit initial (b) Deux arcs coupés (c) Nouveau circuit

FIGURE 4.1 – Lin-2-échange



(a) Circuit initial (b) Trois arcs coupés (c) Nouveau circuit

FIGURE 4.2 – Lin-3-échange

Les mouvements les plus simples sur les circuits considèrent les arcs et les sous-chemins du circuit et les déplacent [Stattenberger *et al.*, 2007], par exemple :

- l'insertion d'un nœud implique de déplacer un nœud v_i et de le placer entre deux nœuds consécutifs v_j et v_{j+1}. Les arcs (v_{i-1}, v_i), (v_i, v_{i+1}) et (v_j, v_{j+1}) sont supprimés et les arcs (v_{i-1}, v_{i+1}), (v_j, v_i) et (v_i, v_{j+1}) crés ;
- l'insertion d'un sous-chemin de longueur k sélectionne un sous-chemin commençant à un nœud v_i et long de k nœuds consécutifs et l'insère entre deux nœuds consécutifs v_j et v_{j+1}. Les arcs (v_{i-1}, v_i), (v_{i+k-1}, v_{i+k}) et (v_j, v_{j+1}) sont supprimés et les arcs (v_{i-1}, v_{i+1}), (v_j, v_i) et (v_{i+k-1}, v_{j+1}) créés ;
- le mouvement k-échange [Lin et Kernighan, 1973] (que nous appellerons Lin-k-échange pour le différencier du k-échange entre plusieurs variables) consiste à supprimer k arcs non-adjacents et à reconnecter les sous-chemins restants avec k nouveaux arcs. Par exemple le Lin-2-échange (Figure 4.1) supprime les arcs (v_i, v_{i+1}) et (v_j, v_{j+1}) et créé les arcs (v_i, v_j) et (v_{i+1}, v_{j+1}).

Dans le cas d'un Lin-2-échange, un des deux sous-chemins doit être parcouru à l'envers du sens initial. Ceci fonctionne bien pour le STSP mais est problématique pour le ATSP puisque le coût du sous-chemin inversé change. Ce calcul est coûteux [Johnson *et al.*, 2002]. Le Lin-3-échange est adapté à l'ATSP. En effet, il y a trois façons de reconnecter les sous-chemins dont une qui préserve leur sens de parcours (Figure 4.2).

Un optimum local obtenu par une méthode d'amélioration utilisant le voisinage du Lin-k-échange est dit k-optimal ou, pour faire court, k-opt. Ce terme est souvent utilisé de façon inappropriée dans la littérature pour parler sans distinction du Lin-k-échange et du k-opt.

4.3.3 Jeux de données

La TSPLIB [Reinelt, 1991] est une bibliothèque d'instances pour le TSP et des problèmes connexes disponible en ligne. [4] Les instances, de divers types, proviennent de sources variées. Tous les coûts sont exprimés par des entiers.

Des instances supplémentaires pour l'ATSP sont disponibles dans les ressources en ligne [5] accompagnant [Johnson *et al.*, 2002].

4.3.4 Méthodes de résolution

Dans [Roberti et Toth, 2012] les auteurs présentent un panorama exhaustif de méthodes exactes pour l'ATSP. Les plus anciennes sont des algorithmes de séparation et d'évaluation (*branch and bound*). Les algorithmes plus récents, comme [Fischetti *et*

4. `http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/`
5. `http://www2.research.att.com/~dsj/chtsp/atsp.html`

al., 2003], se basent sur le *branch and cut*, c'est-à-dire une combinaison de séparation et d'évaluation et de la méthode des plans sécants [Gomory, 1958].

Parmi les méthodes approchées, les algorithmes génétiques ont été utilisés avec succès. La méthode la plus récente, et vraisemblablement la plus performante, est [Nagata et Soler, 2012]. La population initiale est construite en utilisant une recherche locale basée sur le 3-opt. Les meilleurs résultats sont obtenus avec une population de 300 individus. L'algorithme parvient alors à retrouver la meilleure solution dans quasiment tous les cas.

[Rego *et al.*, 2011] est un article de synthèse sur les méthodes de recherche locale pour le TSP en général. Pour l'ATSP, les meilleurs méthodes sont une implémentation de l'algorithme de Kanellakis-Papadimitriou [Kanellakis et Papadimitriou, 1980] produite par Johnson et McGeoch et décrite dans [Cirasella *et al.*, 2001] et une variante de [Gamboa *et al.*, 2006] pour l'ATSP employée telle quelle ou en intégrant les mouvements de [Glover, 1996]. Elles emploient des structures de données spécifiques ainsi que différents voisinages. Ces derniers sont explorés à travers des règles de décision spécialisées afin de guider la recherche dans les régions prometteuses de l'espace de recherche. Les 3-, 4- et 5-opt sont notamment utilisés. Le calcul de ces mouvements est effectué à l'aide de la programmation dynamique ou en créant des circuits plus petits pour ensuite construire le vrai circuit. En termes de résultats, ces méthodes de recherche locale ne parviennent, en général, pas à trouver la solution optimale mais l'écart tourne autour de 1 %.

4.4 Progressive party problem

Le *Progressive Party Problem* (PPP) [Smith *et al.*, 1996] est, à l'origine, un problème d'optimisation combinatoire. Pour nos expériences nous nous intéresserons à sa version de problème de satisfaction de contraintes mais, dans un premier temps, abordons le problème d'optimisation.

Le problème en lui-même n'a pas forcément une importance pratique immédiate. Cependant il a l'avantage d'être un problème d'emploi du temps relativement bien étudié utilisant différentes contraintes.

4.4.1 Problème d'optimisation

D'apparence frivole, le contexte du PPP est l'organisation d'un événement mondain pour une régate de yachts. Les 39 yachts de cette régate étaient amarrés dans une marina de l'île de Wight sur la côte sud de l'Angleterre. La taille des équipages variait de 1 à 7. Afin de permettre à chaque participant de rencontrer un maximum d'autres participants, une soirée mondaine fut planifiée au cours de laquelle certains de ces bateaux seraient les hôtes. Les équipages des autres bateaux se relaieraient à tour de rôle pendant six périodes consécutives d'une demi-heure pendant la soirée. L'équipage d'un bateau hôte resterait à bord de leur bateau pour faire les hôtes. L'équipage d'un bateau invité resterait ensemble en tant qu'une seule et même unité pour toute la soirée. Un équipage invité ne pourrait pas revenir sur un bateau hôte et les équipages invités ne pourraient pas se rencontrer plus d'une fois. La taille des bateaux imposait des contraintes de capacités supplémentaires. Le problème auquel faisait face l'organisateur de la régate était de minimiser le nombre de bateaux hôtes, puisque chaque hôte devait être approvisionné en nourriture ainsi que d'autres prérequis à la soirée.

Un certain nombre de facteurs compliquaient la situation de ce problème concret. Par exemple, le bateau de l'organisateur de la régate se devait d'être un hôte. Deux autres bateaux avaient des équipages composés de parents d'adolescents ; ces bateaux devaient aussi être des hôtes ; les équipages furent séparés afin que les parents

Bateau	Cap.	Équip.	Bateau	Cap.	Équip.	Bateau	Cap.	Équip.
1	6	2	15	8	3	29	6	2
2	8	2	16	12	6	30	6	4
3	12	2	17	8	2	31	6	2
4	12	2	18	8	2	32	6	2
5	12	4	19	8	4	33	6	2
6	12	4	20	8	2	34	6	2
7	12	4	21	8	4	35	6	2
8	10	1	22	8	5	36	6	2
9	10	2	23	7	4	37	6	4
10	10	2	24	7	4	38	6	5
11	10	2	25	7	2	39	9	7
12	10	3	26	7	2	40	0	2
13	8	4	27	7	4	41	0	3
14	8	2	28	7	5	42	0	4

TABLEAU 4.1 – Capacité et équipage des bateaux du PPP.

Configuration	1	2	3	4	5	6
Hôtes	1–12,16	1–13	1,3–19	3–13,25,26	1-11,19,21	1–9,16–19

TABLEAU 4.2 – Configurations des bateaux représentant les numéros des bateaux sélectionnés pour être les hôtes.

restent sur le bateaux hôte et que les enfants deviennent un bateau « virtuel » de capacité nulle. Les enfants de l'organisateur de la régate constituèrent un troisième bateau virtuel. Il y avait ainsi 42 bateaux dont les données sont présentées dans le Tableau 4.1.

En classant les bateaux par ordre décroissant de capacité, après les bateaux 1 à 3 qui doivent obligatoirement être des hôtes, on obtient que 12 bateaux sont insuffisants alors que 13 bateaux suffisent pour embarquer tous les équipages. Ceci donne une borne inférieure pour laquelle, du reste, il existe une configuration réalisable pour 6 périodes [Smith *et al.*, 1996].

4.4.2 Problème de satisfaction de contraintes

La version de satisfaction du PPP part du fait que 13 bateaux soient nécessaires et essaie de trouver une configuration réalisable pour différentes configurations de ces bateaux et un nombre variables de périodes. Suivant la configuration des bateaux, le nombre de périodes ne peut dépasser 10, voire moins, du fait des contraintes « sociales ». Les configurations de 13 bateaux hôtes proposées dans [Walser, 1997] sont présentées dans le Tableau 4.2. La configuration 2 provient initialement de [Smith *et al.*, 1996] et les autres sont des variations aléatoires des hôtes.

Le problème de satisfaction peut être modélisé par $B = (b_{i,j})$ une matrice $I \times P$ à valeurs entières où I correspond au nombre d'équipages invités (il y en a 29), P le nombre de périodes, $b_{i,p}$ le bateau hôte sur lequel se trouve l'équipage du bateau i à la période p, le vecteur $E = \{e_1, \ldots, e_I\}$ représentant la taille de chaque équipage invité et le vecteur $C = \{c_1, \ldots, c_H\}$ où H est le nombre d'hôtes (soit 13) représentant la capacité d'accueil de chaque bateau. Dans une modélisation avec des contraintes

globales, les contraintes régissant le problème sont les suivantes :

$$\forall i \in \{1,\ldots,I\}, \quad \texttt{alldifferent}(\{b_{i,p}|p \in \{1,\ldots,P\}\}) \tag{4.11a}$$

$$\forall p \in \{1,\ldots,P\}, \quad \texttt{multiknapsack}(\{b_{i,p}|i \in \{1,\ldots,I\}, E, C) \tag{4.11b}$$

$$\forall i,j \in \{1,\ldots,I\}, j > i, \quad \texttt{atmost}(1, \{b_{i,p} = b_{j,p}|p \in \{1,\ldots,P\}\}) \tag{4.11c}$$

La contrainte $\texttt{alldifferent}(\{b_{i,p}|p \in \{1,\ldots,P\}\}$ s'assure qu'un équipage invité ne rend pas plus d'une fois visite à un bateau hôte. La contrainte $\texttt{multiknapsack}(\{b_{i,p}|i \in \{1,\ldots,I\}, E, C)$ s'assure que, pour chaque période, le nombre d'invités à bord ne dépasse pas la capacité du bateau. Enfin, la contrainte $\texttt{atmost}(1, \{b_{i,p} = b_{j,p}|p \in \{1,\ldots,P\}\})$ spécifie que deux équipages invités ne peuvent se rencontrer plus d'une fois. Ces contraintes sont explicitées dans la Section 3.3.2.

4.4.3 Méthodes de résolution

Dans [Smith *et al.*, 1996], les auteurs comparent la programmation linéaire en nombres entiers (ILP) avec la programmation par contraintes (PPC) uniquement sur la configuration 2 du problème avec 6 périodes. L'ILP s'avère infructueuse mais la PPC parvient à trouver une solution. Par la suite [Walser, 1997] propose de résoudre le problème exprimé avec des contraintes pseudo-booléennes en utilisant une recherche locale équipée d'une liste tabou et de redémarrages. Il parvient à trouver facilement des solutions pour les 6 configurations avec 6 périodes.

Une autre approche utilisant la recherche tabou est présentée dans [Galinier et Hao, 2004]. Cette fois la configuration 2 est testée avec un nombre de périodes allant de 6 à 10. L'ILP est testée mais ne parvient pas à trouver de solutions. La PPC échoue sur 10 périodes. La méthode proposée échoue également pour ce nombre de périodes mais prend beaucoup moins de temps pour les autres.

À notre connaissance [Van Hentenryck et Michel, 2005] présente les meilleurs résultats pour la recherche locale. Il s'agit à nouveau d'une recherche tabou. Les 6 configurations sont testées avec un nombre de périodes pouvant aller de 6 à 10. Des résultats équivalents sont obtenus dans [Ågren *et al.*, 2009] avec une autre recherche locale. Enfin, [Simonis, 2009] propose une nouvelle approche PPC avec une procédure de recherche spécifiquement conçue pour le problème afin d'obtenir des résultats équivalents aux meilleurs résultats de recherche locale.

4.5 Conclusion

Dans ce chapitre nous avons présenté les différents problèmes qui seront utilisés pour réaliser nos expériences dans la partie concertant nos contributions. Ces problèmes, au nombre de quatre, sont de nature variée aussi bien dans leur modélisation, leur représentation et leur sémantique. Nous espérons ainsi pouvoir démontrer que nos approches peuvent traiter des problèmes de nature différente.

Ce chapitre clôt cette première partie dédiée à l'état de l'art et plus généralement à situer le contexte de nos travaux. Ces derniers sont présentés dans la partie suivante.

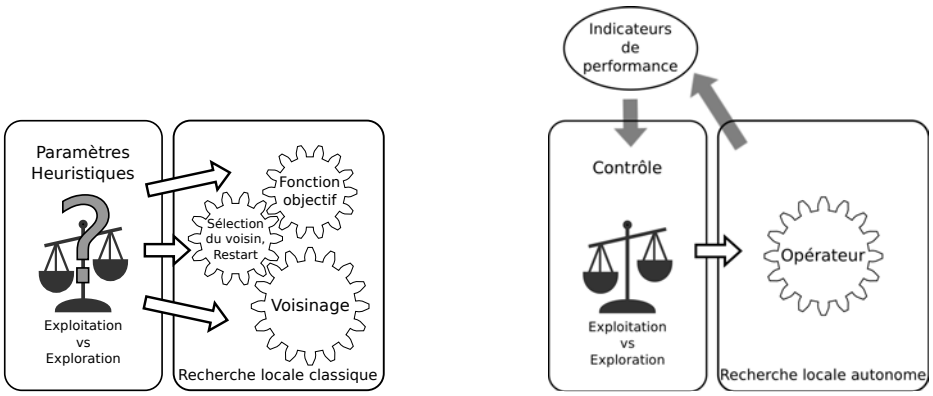
Deuxième partie

Contributions

Contributions : introduction générale

Dans la partie précédente nous avons présenté l'optimisation, ses enjeux et ses difficultés. Nous avons ensuite rappelé le fonctionnement des métaheuristiques classiques de recherche locale permettant de traiter de tels problèmes d'optimisation, mettant en avant leur structure générale, leurs différents composants et paramètres. Bien que reposant sur un principe de recherche de solution assez intuitif (gestion du compromis entre exploration et exploitation de l'espace de recherche), ces algorithmes s'avèrent souvent difficiles à concevoir et utiliser correctement : choix des opérateurs de base, valeur des paramètres... Par conséquent, nous avons décrit ensuite de nouvelles approches de recherche plus autonomes, qui visent à mieux gérer le comportement de ces algorithmes de résolution en automatisant le réglage de leurs paramètres, et notamment, la sélection adaptative d'opérateurs qui s'est initialement développée dans le domaine des algorithmes évolutionnaires. La recherche locale sera le cœur algorithmique de notre approche de résolution. Nous souhaitons étendre le champs de nos investigations à la programmation par contraintes. Nous avons donc proposé un rappel des différentes approches de recherche locale basée sur les contraintes qui permettent, entre autres, de traiter les problèmes de satisfaction de contraintes comme des problèmes d'optimisation. Enfin, afin de préparer les expérimentations que nous effectuerons pour valider nos contributions, nous avons conclu cette première partie par une description des différents problèmes qui sont utilisés dans cette thèse.

Cette nouvelle partie présente nos travaux qui s'inscrivent dans ce contexte de l'optimisation combinatoire, où nous cherchons, à partir de la définition d'un problème, à trouver la meilleure solution ou une solution de bonne qualité. Notre objectif principal est de mettre à la portée de l'utilisateur final, la puissance de résolution qu'offrent les algorithmes basés sur la recherche locale. En effet, parmi les différentes métaheuristiques, rappelées dans la Section 1.4, il est souvent difficile de déterminer quelle approche sera la plus efficace, ce choix s'effectuant souvent par habitude. Une fois le principe algorithmique général fixé, par exemple une recherche tabou, il reste à l'utilisateur à définir les composants principaux de son algorithme (Figure Aa) : fonction objectif, voisinage(s), stratégie(s) de mouvement parmi les voisin... Une fois cette architecture fixée, il lui faudra encore régler des paramètres plus « comportementaux » tels que la longueur de la liste tabou, le nombre d'itérations avant d'effectuer un redémarrage... Nous sommes alors bien loin de la modélisation déclarative d'un problème confié ensuite à un solveur de type boîte noire, tel que cela est envisagé dans le paradigme de la programmation par contraintes.



(a) Approche métaheuristique. (b) Notre approche.

FIGURE A – Schémas généraux.

Nous proposons ici de considérer un modèle plus abstrait de la recherche locale qui consiste à appliquer un opérateur (Figure Ab) sur une solution courante du problème à chaque itération jusqu'à l'obtention d'une solution satisfaisante ou d'un nombre maximal d'itérations. Cette notion d'opérateur peut être plus ou moins complexe. Notre objectif est alors de proposer de générer de manière automatique ou semi-automatique des opérateurs, à partir de briques de base, suffisamment génériques pour être utilisées sur des familles différentes de problèmes. Le cœur de nos contributions est ainsi de fournir un mécanisme de contrôle capable d'évaluer l'utilité (l'efficacité) de chaque opérateur au cours de la recherche et d'effectuer, de manière dynamique et à chaque itération de l'algorithme, le choix de l'opérateur le plus pertinent. Naturellement, ce choix sera guidé par les deux concepts importants qui définissent généralement le comportement des métaheuristiques : l'exploration de l'espace de recherche (diversification) et l'exploitation des zones prometteuses (intensification). Toutefois, ces concepts ne sont pas aussi simples à définir vis-à-vis des objectifs initiaux de la résolution, c'est-à-dire la construction itérative d'un chemin de recherche locale. Nous proposons donc différentes stratégies de haut niveau pour contrôler de manière globale le fonctionnement de l'algorithme et s'abstraire ainsi de l'ensemble de ses paramètres initiaux, dont les effets et les interactions sont souvent difficiles à comprendre et à prévoir. Dans un premier temps, nous abordons des problèmes d'optimisation dans lesquels la seule contrainte est que les solutions doivent avoir une structure de permutation. Mais, comme notre objectif ultime est d'offrir un véritable outil générique de résolution, nous abordons également, dans la dernière partie de cette thèse, la résolution de problèmes pouvant comporter des contraintes plus générales.

Nous détaillons à présent l'organisation et le cheminement de notre travail, ainsi que le contenu des différents chapitres de cette deuxième partie. La Figure B présente de façon schématique les contributions des différents chapitres par rapport au cadre général d'un système de recherche autonome.

En nous inspirant de la sélection adaptative d'opérateurs telle qu'elle a été utilisée dans les algorithmes évolutionnaires (Section 2.3.3), nous souhaitons utiliser et adapter ces mécanismes à la recherche locale. De plus, à travers les métaheuristiques de recherche locale qui ont été présentées, nous pouvons remarquer qu'elles partagent des caractéristiques communes : parcourir au moins un voisinage et sélectionner une solution dans ce voisinage. Les différences se trouvent principalement dans les mécanismes de diversification mis en œuvre. Dans l'absolu on pourrait donc considérer que chacune de ces métaheuristiques n'est qu'une instance bien spécifique d'une sélection adaptative d'opérateurs, notamment une alternance entre l'utilisation d'un voisinage d'intensification et d'un voisinage de diversification.

Dans le Chapitre 5, nous posons les bases de notre approche en formalisant la notion d'opérateur comme étant, au minimum, une combinaison d'un voisinage et d'une fonction de sélection. Des opérateurs plus complexes peuvent éventuellement être conçus en composant les opérateurs entre eux. Ceci nous amène à pouvoir, et à vouloir, manipuler plusieurs opérateurs en même temps avec pour objectif de choisir l'opérateur qui semble le plus approprié à l'état de recherche dans laquelle se trouve le processus de résolution. Bien entendu, comme dans les métaheuristiques de recherche locale classiques, nous prenons en compte la qualité des solutions trouvées. Toutefois, et ceci fait entre autres l'originalité de nos travaux, nous considérons également l'aspect de diversité que l'on retrouve plus souvent dans le contexte des algorithmes évolutionnaires. Pour ce faire, nous considérons la distance de la solution candidate par rapport à une fenêtre glissante du chemin parcouru : celle-ci pouvant être vue comme une population. Ces deux mesures – qualité et diversité – doivent

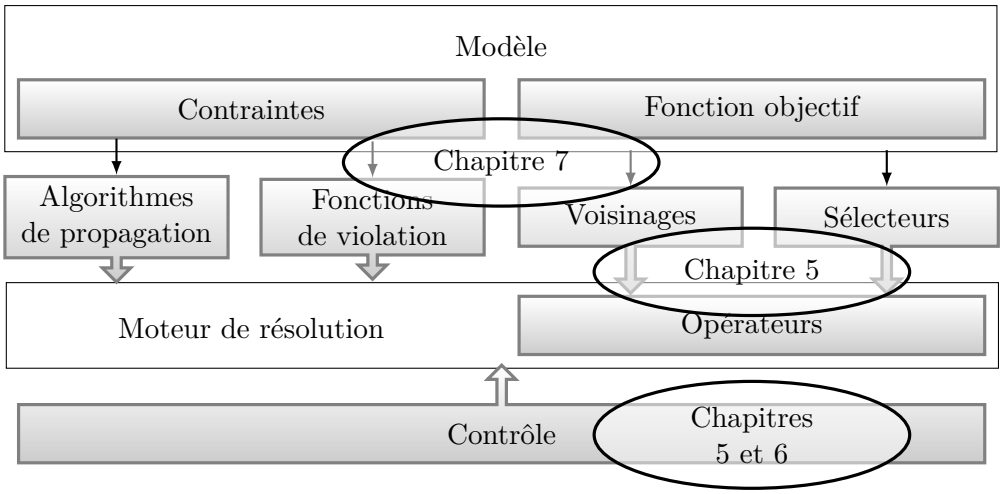


FIGURE B – Schéma synthétique d'un système pour la résolution de problèmes en recherche autonome indiquant les chapitres correspondant aux contributions dans ces différents aspects.

être agrégées afin d'obtenir une seule valeur nous permettant de faire une choix entre les opérateurs. À cette fin, nous utilisons le concept de Pareto dominance pour déterminer les opérateurs qui dominent d'autres opérateurs, le choix se faisant en fonction du nombre d'opérateurs dominés. Dans ce chapitre nous considérons les problèmes du One Max, de l'affectation quadratique et du voyageur de commerce présentés au Chapitre 4.

Dans la continuité du Chapitre 5, le Chapitre 6 propose des méthodes plus adaptatives. En particulier, le concept de Pareto dominance tel que nous l'utilisons ne nous permet pas de spécifier explicitement le compromis entre la qualité et la diversité. Bien entendu, cela peut être un avantage puisque cela nous évite d'avoir à faire un choix, et donc à régler un paramètre. Nous avons d'ailleurs vu au Chapitre 2 que la recherche autonome s'est développée, entre autres, suite à l'inflation du nombre de paramètres des algorithmes de résolution. Toutefois, il faut également reconnaître que spécifier ce compromis peut avoir un intérêt puisque cela nous permettrait de mieux contrôler la recherche, tout en restant un paramétrage de haut niveau plus facilement appréhendable par l'utilisateur. Dans cette optique nous introduisons une nouvelle façon de gérer ce compromis en considérant les distances entres opérateurs. Nous expérimentons également avec une stratégie adaptative de variation de ce compromis. Mais puisque tout ceci introduit de nouveaux paramètres, et afin d'assurer une expérimentation équitable, nous utilisons donc certaines techniques de paramétrage automatique décrites à la Section 2.2.

Le Chapitre 7 est une extension des deux chapitres précédents aux problèmes de satisfaction de contraintes, testée sur le Progressive Party Problem (Section 4.4). Les composants de algorithmes peuvent être extrait à partir du modèle, en particulier au travers d'un systèmes pénalités associé à chaque contrainte. La difficulté de concevoir une bonne recherche locale dans ce contexte consiste alors à considérer les violations des différentes contraintes afin d'obtenir une fonction objectif adéquate. Nous pouvons, dans ce contexte, générer des voisinages spécifiques aux problèmes de satisfaction de contraintes, notamment pour la sélection des variables qui tiennent compte de la violation des contraintes. Notre approche permet alors d'obtenir de bons résultats en évitant à l'utilisateur un paramétrage complexe de sa fonction objectif. L'utilisateur peut alors se contenter de fournir un modélisation du problème à partir de laquelle les composants de recherche locale seront inférés.

Chapitre 5

Sélection d'opérateurs pour la recherche locale

Sommaire

5.1 Introduction 70
5.2 Cadre pour la recherche locale 71
5.2.1 Définitions de base 71
5.2.2 Structure opérationnelle 71
5.3 Encodage et mouvements 72
5.3.1 Voisinages binaires 72
5.3.2 Permutations 73
5.4 Contrôle pour la recherche locale 74
5.4.1 Métriques 74
5.4.2 Utilité et sélection 75
5.4.3 Mise en œuvre 77
5.5 Validation initiale 78
5.5.1 One Max 78
5.5.2 Discussion 78
5.6 Expérimentations sur les problèmes de permutations . . 79
5.6.1 Protocole expérimental 81
5.6.2 Résultats et discussion 81
5.7 Conclusion 87

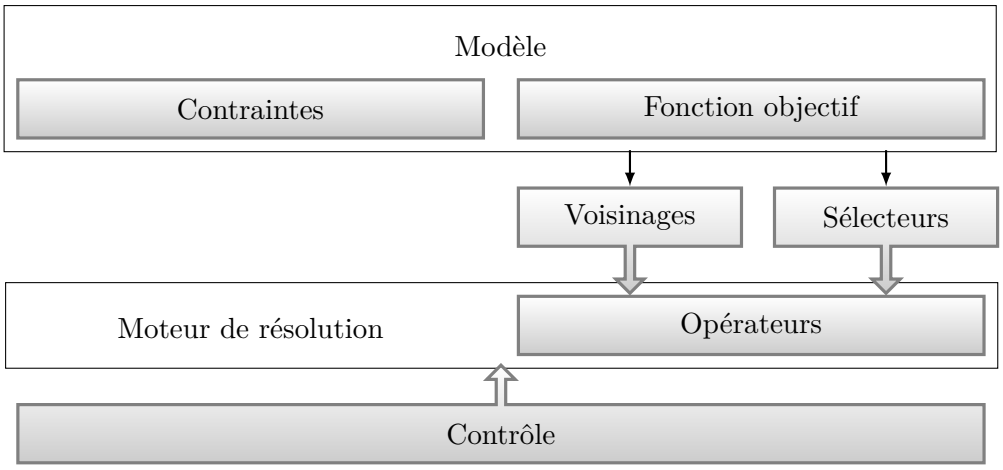


FIGURE 5.1 – Schéma synthétique d'un système pour la résolution de problèmes d'optimisation avec la recherche autonome.

5.1 Introduction

Notre objectif est de créer un solveur fonctionnant sur le principe de la recherche locale. Un solveur implique généralement un usage relativement simple pour l'utilisateur : donner une modélisation du problème et l'instance à tester puis lancer l'exécution sans autre forme d'intervention de sa part. C'est ce que nous tentons de faire avec COMET et que nous avons décrit avec plus de détails dans la Section 3.3.

Afin de pouvoir gérer la résolution d'un problème de façon autonome, il convient d'abord de formaliser les notions qui seront employées par la recherche locale : qu'est-ce qu'un opérateur, comment se déplace-t-il dans le paysage de recherche ou encore quel est le chemin induit par ces déplacements ? Il est également nécessaire de fixer des métriques permettant d'évaluer la qualité et la nature d'une solution par rapport aux autres. Ces métriques peuvent ainsi décrire le potentiel d'exploitation et d'exploration des opérateurs. À partir d'observations sur le comportement antérieur des opérateurs, il devient alors possible d'essayer de prédire quel sera le ou les opérateurs qui seront les mieux à même d'apporter un bénéfice au processus de recherche.

La Figure 5.1 présente de façon conceptuelle les différents composants à considérer. Le modèle du problème, à travers ses contraintes et sa fonction objectif influe sur le choix des voisinages et des sélecteurs. Ceux-ci permettent de créer des opérateurs qui seront utilisés au sein de l'algorithme de résolution. Le contrôle est un agent qui agit sur le processus de résolution en faisant des choix pour influencer la recherche.

Dans ce premier chapitre de la partie concernant nos contributions, nous examinons d'abord une méthode de sélection d'opérateurs basée sur un compromis entre exploitation et exploration. Ce compromis est basé sur la notion de Pareto dominance, un concept provenant de l'optimisation multiobjectif. À nos yeux, son intérêt est de ne pas introduire de paramètre supplémentaire afin de fixer le compromis entre l'exploitation et l'exploration. Un nombre minimum de paramètres implique un moindre effort de réglage de ceux-ci et donc un pas de plus vers une simplification de l'utilisation des techniques d'optimisation.

Ici, nous nous concentrerons principalement sur les problèmes de permutations. Ils sont faciles à modéliser et ne requièrent pas de structures de données complexes. Par ailleurs, il existe plusieurs opérateurs de recherche locale pouvant s'appliquer à ces problèmes et ils peuvent être assez aisément décrits formellement. Nous utiliserons également un problème binaire « jouet », le *One Max*, afin de pouvoir évaluer le comportement observé de la méthode proposée par rapport au choix optimal théorique qui peut être calculé dans ce contexte.

Une grande partie du contenu de ce chapitre a été publié dans [Veerapen et Saubion, 2011a; Veerapen et Saubion, 2011b].

5.2 Cadre pour la recherche locale

Cette section a pour objectif d'offrir une description simple et générique de ce que sont les voisinages et les opérateurs qui peuvent être utiles pour définir de nouveaux opérateurs et gérer leur application en fonction de leur impact sur le processus de recherche. Un deuxième objectif est de proposer un cadre de travail capable de comparer des voisinages et des sélecteurs dans le contexte d'une procédure de recherche multiopérateur.

5.2.1 Définitions de base

Afin de faciliter la lecture et la compréhension des définitions suivantes, rappelons-nous de la définition d'une relation de voisinage (Définition 1.2.8) :

Soit $\mathcal{S}$ l'espace des solutions candidates. Une *relation de voisinage* est une relation binaire irréflexive $\mathcal{N} \subseteq \mathcal{S}^2$ sur l'espace de recherche. Dans la majorité des cas, la relation est aussi symétrique.

Définition 5.2.1. Soit une relation de voisinage $\mathcal{N}$, nous définissons l'ensemble des *trajectoires de recherches* par $\mathcal{P}_{\mathcal{N}} = \{s_1 \cdots s_n \in \mathcal{S}^* | \forall i > 1, (s_{i-1}, s_i) \in \mathcal{N}\}$, où $\mathcal{S}^*$ représente classiquement l'ensemble des mots construits sur $\mathcal{S}$. Donc, toute paire (s, s') d'éléments de $\mathcal{S}$, telles que $(s, s') \in \mathcal{N}^+$, [1] définie une classe d'équivalence sur l'ensemble $\mathcal{P}_{\mathcal{N}}$, qui correspond à tous les chemins reliant s à s'. Nous notons cet ensemble $\mathcal{P}_{\mathcal{N}}/(s, s')$. Dans la plupart des cas, le voisinage est complet, c'est-à-dire $\forall s, s' \in S, \mathcal{P}_{\mathcal{N}}/(s, s') \neq \emptyset$.

Définition 5.2.2. En fait, la relation de voisinage définit la structure déclarative de l'espace de recherche. Nous pouvons ainsi définir la *distance* entre s et s' comme $d_{\mathcal{N}}(s, s') = min_{p \in \mathcal{P}_{\mathcal{N}}/(s,s')}|p|$, où $|p|$ est la longueur classique d'un mot. Par définition, nous imposons que $d_{\mathcal{N}}(s, s) = 0$. Notons qu'il est possible de requérir que $\mathcal{N}$ soit symétrique afin que d soit une distance.

Définition 5.2.3. Afin d'exprimer des structures de voisinages plus complexes, nous notons $\mathcal{N} \circ \mathcal{N}'$ la *composition* et $\mathcal{N} \cup \mathcal{N}'$ l'*union* qui sont les constructeurs de voisinage les plus communément utilisés. Un voisinage composé avec lui-même est noté $\mathcal{N}^2$ et $\mathcal{N}^{n+1} = \mathcal{N} \circ \mathcal{N}^n$.

5.2.2 Structure opérationnelle

Considérons maintenant le paysage de recherche, nous introduisons d'abord la relation d'ordre $<$ sur $\mathcal{S}$ correspondant à l'ordre induit par la fonction objectif du problème. Notons que nous considérons ici uniquement les problèmes de minimisation sans perte de généralité.

Il nous faut maintenant introduire la structure opérationnelle de la recherche locale afin d'explorer la relation de voisinage.

Sélecteur

Définition 5.2.4. Dans ce contexte, un *sélecteur* est une fonction qui procède à une sélection sur un voisinage, éventuellement guidé par la relation d'ordre $<$ et

1. $\mathcal{N}^+$ est la clôture transitive de $\mathcal{N}$.

défini par $\sigma : \mathcal{S} \times 2^{\mathcal{S}^2} \mapsto \mathcal{S}$ (ici la sélection retourne un unique voisin), tel que $(s, \sigma(s, \mathcal{N})) \in \mathcal{N}^=$ (la clôture réflexive de $\mathcal{S}$ afin d'inclure l'identité).

Nous pouvons maintenant proposer plusieurs fonctions classiques de sélection afin de construire des opérateurs.

Aléatoire : σ_R tel que $\sigma_R(s, \mathcal{N})$ est n'importe quel s' choisi aléatoirement tel que $(s, s') \in \mathcal{N}$.

Meilleure amélioration : σ_{BI} tel que $\sigma_{BI}(s, \mathcal{N})$ est un élément minimal s' selon l'ordre $<$, tel que $(s, s') \in \mathcal{N}$.

Meilleure amélioration k : σ_{BIk} tel que $\sigma_{BIk}(s, \mathcal{N})$ est un élément uniformément sélectionné $s' \in K$, K étant l'ensemble des k-meilleurs éléments selon l'ordre $<$, tel que $(s, s') \in \mathcal{N}$.

Amélioration : σ_I tel que $\sigma_I(s, \mathcal{N})$ est n'importe quel élément s' tel que $(s, s') \in \mathcal{N}$ et $s' < s$.

Tournois k : σ_{Tk} tel que $\sigma_{Tk}(s, \mathcal{N})$ est un élément s' tel que K est un sous-ensemble de k éléments en relation avec s dans $\mathcal{N}$ et s' est le meilleur de ces k éléments.

Opérateur de mouvement

Définition 5.2.5. Un *opérateur* est alors défini par une paire $(\mathcal{N}, \sigma)$.

Considérons à nouveau les chemins induits par un opérateur $\mathcal{P}_o = \bigcup_{n>1}\{s_1 \cdots s_n \in \mathcal{S}^* | o = (\mathcal{N}, \sigma), \forall i > 1, s_i = \sigma(s_{i-1}, \mathcal{N})\}$ Afin de simplifier la notation, nous utilisons $o(s) = \sigma(s, \mathcal{N})$ quand $o = (\mathcal{N}, \sigma)$ puisque o peut être considéré comme une fonction sur S. Nous notons $o \circ o'$ la composition entre opérateurs, o^2 la composition de o avec lui-même et $o^{n+1} = o \circ o^n$.

Notons, ici, que nous avons seulement l'inclusion $\mathcal{P}_o \subseteq \mathcal{P}_\mathcal{N}$, puisque certains chemins du voisinage peuvent ne pas forcément être construits par les opérateurs dès lors qu'ils contiennent un processus de sélection entre les voisins. De plus, même s'il existe un chemin dans $\mathcal{P}_o$ de s à s', il n'existe pas forcément un chemin de s' à s. Du fait de cet aspect non symétrique des opérateurs l'utilisation d'une distance simple sur les chemins créés par les opérateurs n'est pas forcément évidente.

Nous pouvons maintenant gérer la recherche locale à voisinages multiples en composant ou en réunissant des relations de voisinage. Nous souhaitons ainsi avoir la possibilité de générer automatiquement des opérateurs.

5.3 Encodage et mouvements

Les solutions des problèmes que nous considérerons dans ce chapitre sont facilement représentables par un vecteur – un tableau unidimensionnel – de valeurs binaires ou entières. À cette représentation vectorielle sont associées des opérateurs permettant d'en modifier les valeurs afin d'obtenir de nouvelles solutions.

5.3.1 Voisinages binaires

Nous considérons ici X l'espace de recherche de toutes les chaînes de bits de longueur n (Section 4.1), c'est-à-dire l'ensemble des $x = x_1, x_2, \ldots, x_n$, $x_i \in \{0, 1\}$.

Il existe quelques mouvements de base pour les variables binaires, par exemple le mouvement *bit-flip* inverse chaque bit avec une probabilité $1/n$. Un *k-flip* sélectionne k bits de façon uniforme et les inverse.

Les voisinages *k-flip*, $\mathcal{N}_F^k$, peuvent se formaliser facilement tel que $(s, s') \in \mathcal{N}_F^k$ si et seulement si $|h(s, s') = k|$ où h est la distance de Hamming classique.

Le voisinage *bit-flip* quant à lui, $\mathcal{N}_F^B$, est plus complexe à exprimer, sa taille change à chaque évaluation. Toutes les solutions sont atteignables à travers ce voisinage, on a donc $(s, s') \in \mathcal{N}_F^B$, $\forall s, s' \in \mathcal{S}$. Toutefois la probabilité d'inverser k bits n'est pas uniforme mais donnée par $P(k) = C_n^k (n-1)^{(n-k)}/n^n$.

Il est important de ne pas confondre le *1-flip* et le *bit-flip* puisque leurs noms peuvent éventuellement porter à confusion. Le premier ne change la valeur que d'un seul bit sélectionné de façon uniforme. Le second change la valeur de chaque bit avec une probabilité $1/n$.

5.3.2 Permutations

Concentrons-nous maintenant sur les permutations qui correspondent au principal encodage que nous utiliserons dans nos problèmes. Notre objectif est de proposer un panorama détaillé des opérateurs utilisables dans ce contexte.

Soit $\Pi(n)$ l'espace de recherche, c'est-à-dire l'ensemble de toutes les permutations de l'ensemble $\{1, \ldots, n\}$. Si $\pi \in \Pi(n)$ et $1 \leq i \leq n$, alors π_i dénote l'élément i dans π.

Comme décrit dans [Schiavinotto et Stützle, 2007] nous pouvons utiliser un ensemble de relations de base de voisinage induites par les permutations de base possibles.

- swap $\mathcal{N}_S$:
  $(s, s') \in \mathcal{N}_S$ ssi $s = (\pi_1, \ldots, \pi_i, \pi_{i+1}, \ldots, \pi_n)$ et
  $s' = (\pi_1, \ldots, \pi_{i+1}, \pi_i, \ldots, \pi_n)$ pour un i.
- échange $\mathcal{N}_E$:
  $(s, s') \in \mathcal{N}_E$ ssi $s =$
  $(\pi_1, \ldots, \pi_{i-1}, \pi_i, \pi_{i+1}, \ldots, \pi_{j-1}, \pi_j, \pi_{j+1}, \ldots, \pi_n)$ et
  $s' = (\pi_1, \ldots, \pi_{i-1}, \pi_j, \pi_{i+1}, \ldots, \pi_{j-1}, \pi_i, \pi_{j+1}, \ldots, \pi_n)$ pour un i et un j.
- insertion $\mathcal{N}_I$:
  $(s, s') \in \mathcal{N}_I$ ssi $s =$
  $(\pi_1, \ldots, \pi_{i-1}, \pi_i, \pi_{i+1}, \ldots, \pi_{j-1}, \pi_j, \pi_{j+1}, \ldots, \pi_n)$ et
  $s' = (\pi_1, \ldots, \pi_{i-1}, \pi_{i+1}, \ldots, \pi_{j-1}, \pi_i, \pi_j, \pi_{j+1}, \ldots, \pi_n)$ pour un i et un j.
- échange d'arcs $\mathcal{N}_{EE}$:
  $(s, s') \in \mathcal{N}_{EE}$ ssi $s =$
  $(\pi_1, \ldots, \pi_{i-1}, \pi_i, \pi_{i+1}, \ldots, \pi_{j-1}, \pi_j, \pi_{j+1}, \ldots, \pi_n)$ et
  $s' = (\pi_1, \ldots, \pi_i, \pi_j, \pi_{j-1}, \ldots, \pi_{i+1}, \pi_{j+1}, \ldots, \pi_n)$ pour $i + 1 < j$.

Ce dernier voisinage est approprié pour les permutations circulaires que l'on trouve par exemple dans le TSP.

Il apparaît clairement que le voisinage construit par $\mathcal{N}_S$ peut également être construit par $\mathcal{N}_E$ et $\mathcal{N}_I$. Des relations d'ordres peuvent donc être définies pour classifier les voisinages afin de mettre l'emphase sur les relations entre les distances qu'ils induisent [Schiavinotto et Stützle, 2007]. Dans cette publication, les auteurs considèrent formellement différents voisinages et définissent des distances associées à ces voisinages. Comme mentionné précédemment, puisque notre but est de gérer dynamiquement les opérateurs en fonction de leur comportement et de leurs propriétés, ces métriques nous intéressent tout particulièrement. Néanmoins, dans [Schiavinotto et Stützle, 2007], les auteurs ne considèrent que des méthodes utilisant un unique opérateur et les métriques pouvant être utilisées pour évaluer la diversité de la trajectoire de la recherche locale dépendent entièrement de l'opérateur. La question des métriques, entre autres, est abordée dans la section suivante.

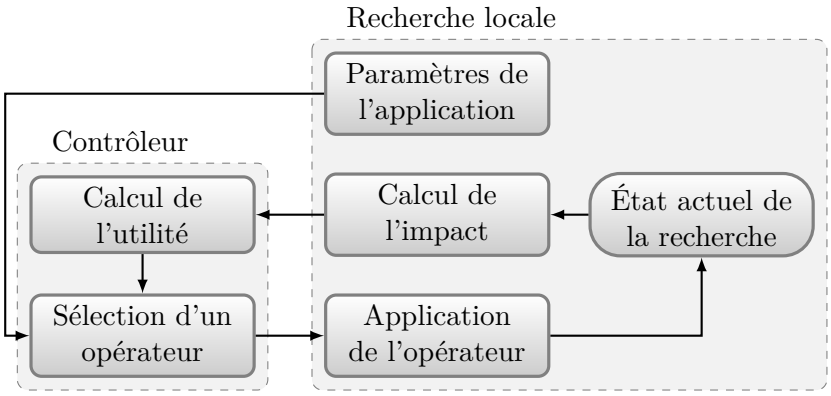


FIGURE 5.2 – Schéma général du contrôle pour la recherche locale. L'algorithme de recherche locale et le contrôleur sont deux entités distinctes qui communiquent à travers un nombre réduit d'informations. La recherche locale communique l'impact (la performance) des opérateurs et en retour le contrôleur lui indique quel opérateur utiliser.

5.4 Contrôle pour la recherche locale

L'objectif de notre approche est de sélectionner, à partir d'un ensemble donné d'opérateurs, un opérateur approprié à appliquer à chaque itération. Ceci requiert l'évaluation de l'efficacité des opérateurs basée sur leur comportement antérieur et la sélection d'un opérateur capable de faire progresser le processus de recherche, à des fins soit d'exploitation soit d'intensification.

La Figure 5.2 illustre cette approche. Nous considérons un algorithme de recherche locale auquel est associé un *contrôleur* dont l'objectif est de gérer les informations obtenues à partir des applications des opérateurs afin de sélectionner le prochain opérateur à appliquer.

5.4.1 Métriques

Comme mentionné dans l'introduction, un aspect important du contrôle est d'estimer l'équilibre exploitation-exploration en utilisant des métriques qui peuvent évaluer l'efficacité d'un opérateur par rapport à la trajectoire de recherche afin de choisir le prochain opérateur. Nous proposons de traiter simultanément deux critères, la qualité et la diversité, afin de gérer cet équilibre comme un compromis. Il est important de noter que la diversité dans le contexte de la recherche locale est plus difficile à exprimer que dans le contexte évolutionnaire.

Qualité

La qualité est mesurée directement grâce à la fonction objectif. Le changement relatif de qualité en utilisant un opérateur *op* sur une solution s est donné par

$$\Delta Q = eval(s) - eval(op(s))$$

Diversité

La diversité est un concept intuitif lorsque l'on considère des populations de solutions dans le contexte des algorithmes évolutionnaires. Elle l'est moins en recherche locale où une seule solution est produite à chaque itération. Cette notion a été explorée par exemple dans [Sidaner *et al.*, 2002] et dans [Linhares, 2004]. Nous

pourrions considérer la diversité du chemin de la recherche (la séquence de solutions déjà trouvées) ou une fenêtre glissante de celui-ci. À la place nous choisissons d'essayer de mesurer la différence entre le chemin et la solution candidate actuelle $c = op(s)$. Nous proposons deux perspectives différentes :

1. le degré de dissimilitude entre le chemin et c au niveau des variables elles-même ;
2. l'éloignement de c par rapport au chemin en termes de nombres d'opérations de mouvement les séparant.

Différence selon les variables La distance L_1 (de Manhattan) entre deux vecteurs p et q est définie par $d_1(p,q) = \sum_{i=1}^{n} |p_i - q_i|$. Nous utilisons une métrique simple mesurant la distance L_1 entre les représentations de la solution candidate et le barycentre du chemin.

Cette différence est calculée au niveau du couple variable-valeur : moins fréquentes sont les occurrences des couples variable-valeur de la solution candidate dans le chemin, plus grande est la distance est entre eux. L'équation suivante formalise cette notion. Soit $P_{i,j}$ le chemin de l'itération i jusqu'à j, $i \leq j$. Alors

$$d_1^P(c, P_{i,j}) = \frac{1}{n} \times \sum_{k_c=1}^{n} \left(1 - \frac{occ(P_{i,j}, (k_c, \pi_{k_c}))}{|P_{i,j}|}\right) \tag{5.1}$$

où $occ(P_{i,j}, (a,b))$ correspond au nombre de fois où le couple variable-valeur (a,b) apparaît dans $P_{i,j}$. Par exemple dans $(1,3,2)$, les couples sont $(1,1)$, $(2,3)$ et $(3,2)$. Pour des problèmes qui modélisent des cycles, (a,b) représente deux valeurs consécutives. Par exemple dans $(1,3,2)$ les couples sont $(1,3)$, $(3,2)$ et $(2,1)$.

Cette distance fonctionne avec des variables de domaines quelconques.

Éloignement d'une solution par rapport au chemin Ici l'idée est de comparer le chemin de recherche au chemin optimal qui aurait pu être obtenu en utilisant une relation de voisinage de référence.

Nous définissons ainsi la distance

$$d_{\mathcal{N}}^P(p_k, P_{i,j}) = \sum_{l=i}^{j} \frac{|P_{l,k}|}{d_{\mathcal{N}}(p_l, p_k)} \quad , i \leq j \leq k$$

En utilisant un opérateur suffisamment simple, $d_{\mathcal{N}}^P$ peut être utilisée pour évaluer les caractéristiques exploratoires d'opérateurs plus complexes. À cette fin, nous suggérons l'utilisation de l'opérateur Échange, $\mathcal{N}_E$. Nous utilisons un des algorithmes présentés dans [Schiavinotto et Stützle, 2007] pour calculer $d_{\mathcal{N}_E}$ (Algorithme 5.1) en temps linéaire par rapport à la taille de la solution.

Dans la Figure 5.3, nous présentons un exemple de l'utilisation que nous souhaitons faire de la distance $d_{\mathcal{N}}$. Le chemin a, b, c, d, e est celui qui a été créé par le processus itératif de recherche locale. On constate toutefois que a, f, e aurait été le chemin le plus cours. Nous souhaitons donc utiliser le rapport entre ces deux distances afin de caractériser l'exploration.

5.4.2 Utilité et sélection

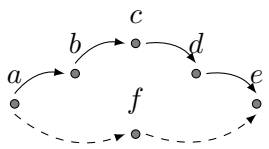
Nous présentons maintenant le processus de sélection utilisé pour choisir l'opérateur de mouvement à chaque étape.

Définition 5.4.1. Soient deux vecteurs u et v de même cardinalité p et considérant un problème de maximisation, u *domine* v si $u_k \geq v_k, \forall k \in \{1, \ldots, p\}$ avec au moins

Algorithme 5.1: Calcul de $d_{\mathcal{N}_E}(\pi, \pi')$.

Entrées : π et π' les deux permutations
$c \leftarrow 0$;
$\overline{\pi} \leftarrow \pi' \cdot \pi^{-1}$;
pour $i \leftarrow 1 \dots n$ **faire**
  si $\overline{\pi}_j$ *n'est pas marqué* **alors**
    $c \leftarrow c + 1$;
    $j \leftarrow i$;
    répéter
      marquer $\overline{\pi}_j$;
      $j \leftarrow \overline{\pi}_j$;
    jusqu'à j=i;
  fin
fin
Sorties : $n - c$

a	1	2	3	4	*5*	*6*
b	1	2	*3*	*4*	6	5
c	1	*2*	4	*3*	6	5
d	1	3	*4*	*2*	6	5
e	1	3	2	4	6	5



a	1	*2*	*3*	4	5	6
f	1	3	2	4	*5*	*6*
e	1	3	2	4	6	5

FIGURE 5.3 – Exemple illustratif de l'utilisation de la distance $d_{\mathcal{N}}$.

une inégalité stricte. Le terme *Pareto dominance* est fréquemment employé pour décrire cette relation.

L'utilité U_P de l'opérateur o dans l'ensemble O de n opérateurs est alors définie par

$$U_P(o) = |\{o' | o' \in O, o \succ o'\}| + \epsilon \tag{5.2}$$

où ϵ est une valeur positive afin d'obtenir une utilité non nulle et $o \succ o'$ signifie que o domine o'.

La Figure 5.4 illustre le concept de la Pareto dominance. Chaque point représente un opérateur par rapport à sa valeur correspondant à la qualité et la distance. Les opérateurs se trouvant dans le rectangle inférieur gauche d'un autre opérateur sont les opérateurs qui sont dominés par cet opérateur. En l'occurrence, ici l'opérateur a domine quatre autres opérateurs.

L'opérateur à utiliser à chaque itération de l'algorithme est sélectionné selon une probabilité proportionnelle à son utilité [Nareyek, 2004]. Nous définissons l'utilité d'un opérateur comme le nombre d'opérateurs que celui-ci domine augmenté d'un ϵ afin de conserver une utilité non nulle. Dans ce chapitre nous nous restreindrons intentionnellement à des mécanismes de sélection simples afin de minimiser

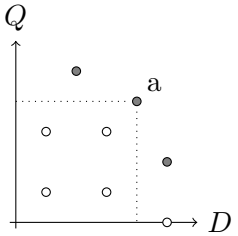


FIGURE 5.4 – Illustration du concept de Pareto dominance

le nombre de paramètres à prendre en compte. Des mécanismes plus sophistiqués seront considérés dans le chapitre suivant.

Notons qu'au début de cette thèse nous avons tenté d'utiliser la méthode Compass (Section 2.3.3) mais l'utilité basée sur la Pareto dominance s'est vite imposée comme plus robuste et conceptuellement plus simple.

5.4.3 Mise en œuvre

Notre objectif est d'avoir une approche aussi générique que possible pour résoudre les problèmes de permutations. À cette fin, notre solveur se décompose en quatre modules distincts :

- définition du problème ;
- gestionnaire de chemin ;
- gestionnaire d'opérateurs ;
- conteneur d'opérateurs.

Une vue globale de l'algorithme manipulant ces modules est donnée dans l'Algorithme 5.2.

Du point de vue de l'utilisateur, résoudre un nouveau problème nécessite seulement de définir une procédure de lecture (pour les données de l'instance) ainsi que la spécification de la fonction objectif et des contraintes. L'utilisateur peut, s'il le souhaite, ajouter de nouveaux opérateurs au Conteneur d'opérateurs. Il peut également définir une méthode d'initialisation de la solution initiale s'il ne veut pas utiliser une solution générée aléatoirement.

Le programme est écrit en Comet. Nous estimons que ce langage de programmation destiné à la recherche locale basée sur les contraintes nous offre des fonctionnalités intéressantes puisqu'il facilite la programmation d'algorithmes de recherche locale et qu'il fournit des mécanismes simples de manipulation de voisinages et de sélecteurs que nous avons décrit à la Section 3.3.

Notre programme se base sur ces fonctionnalités intuitives et peut être considéré comme une extension de Comet. Nous sommes convaincus que la généricité et la simplicité (modulo le minimum de connaissances nécessaires afin de définir le problème) alliées à la facilité intrinsèque d'utilisation de Comet basé sur la recherche locale, est un pas positif vers la démocratisation de l'utilisation de logiciels d'optimisation.

Algorithme 5.2: Description algorithmique globale

```
Définir le problème comme un Problème de permutation;
Ajouter les opérateurs au Conteneur d'opérateurs;
Initialiser le Gestionnaire de chemin;
Initialiser le Gestionnaire d'opérateurs;
s ← solution initiale;
s* ← s;
repeat
    op ← selectionner un opérateur;
    s ← op(s);
    Mise à jour du Gestionnaire de chemin avec s;
    Mise à jour du Gestionnaire d'opérateur avec les mesures de s;
    if s est meilleur que s* then s* ← s
until non condition de fin;
return s*
```

Notre approche se veut aussi générique que possible mais fait face à certaines limites. En pratique (pour des opérateurs plus complexes que de simples échanges),

l'utilisateur doit donner la fonction permettant de calculer le delta de l'évaluation d'une solution candidate puisqu'elle dépend de la fonction objectif.

5.5 Validation initiale

5.5.1 One Max

Afin d'évaluer la pertinence de l'utilisation de la Pareto dominance décrite précédemment, nous allons tout d'abord faire des tests sur le problème du One Max que nous avons décrit à la Section 4.1. Nous testerons aussi l'effet de la longueur de la fenêtre glissante.

Pour rappel, le problème du One Max demande de maximiser le nombre de 1 d'une chaîne de bits, ce qui est formalisé par

$$max_{x \in X_n} \sum_{i=1}^{n} x_i$$

où X_n est l'espace de recherche de toutes les chaînes de bits de longueur n, c'est-à-dire l'ensemble des $x = x_1, x_2, \ldots, x_n$, $x_i \in \{0, 1\}$.

Notre objectif ici est de comparer le comportement en pratique des méthodes que nous proposons avec les meilleurs choix possibles que pourrait faire un algorithme. En effet, la simplicité du problème One Max permet de calculer ces comportements théoriques optimaux.

Nous utilisons quatre voisinages, les mêmes que ceux définis à la Section 5.3.1 :
- le *bit-flip* qui inverse chaque bit avec une probabilité $1/n$;
- le 1-*flip* ;
- le 3-*flip* ;
- le 5-*flip*.

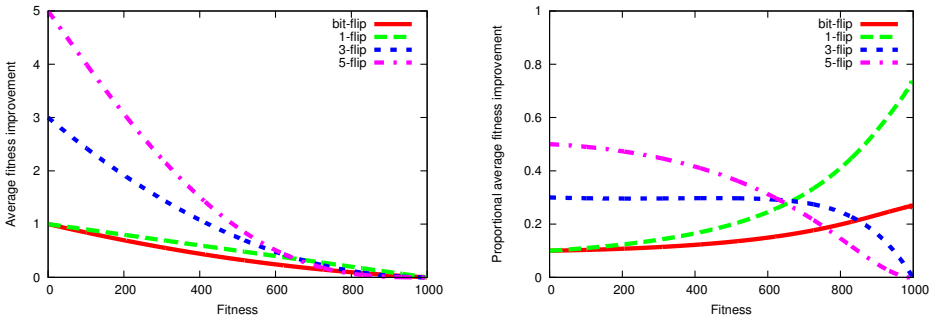
Ces quatre voisinages sont utilisés avec une fonction de sélection choisissant à chaque fois le voisin ayant le meilleur score (énumération exhaustive). La taille de la chaîne de bits est fixée à 1 000.

5.5.2 Discussion

Les Figures 5.5 représentent l'amélioration moyenne théorique pouvant être obtenue en utilisant les opérateurs sur des solutions ayant une valeur de fonction objectif allant de 0 à 1 000. Sur la Figure 5.5a, on peut constater que l'amélioration moyenne théorique se dégrade lentement à mesure que l'évaluation de la solution augmente. Sur la Figure 5.5b nous reprenons les mêmes données mais cette fois l'amélioration moyenne est représentée de façon proportionnelle afin de pouvoir observer la différence entre les quatre opérateurs pendant le stage final de recherche, quand l'évaluation des solutions est proche de 1 000. On peut noter, par exemple, que malgré le fait que l'amélioration moyenne du 1-*flip* soit marginale, il est néanmoins le meilleur opérateur lorsque l'évaluation des solutions est élevée.

Les Figures 5.6 montrent les résultats empiriques de la fréquence moyenne d'application de chaque opérateur sur 50 exécutions de la sélection proportionnelle avec la Pareto dominance comme utilité. Des fenêtres glissantes de longueurs 10 et 100 sont comparées, ainsi que des valeurs de ϵ de 0.1 et 0.01. Afin de mieux distinguer la forme des courbes résultats, celles-ci sont également représentées de façon lissée (*smoothed* dans la légende).

La forme générale de ces courbes se rapprochent de celles présentées sur le graphique normalisé (Figure 5.5b). Ceci montre que notre mécanisme de sélection est en mesure de déterminer que certains opérateurs fonctionnent mieux à certains moments que d'autres. Puisque l'on part sans rien savoir sur les opérateurs, la première



(a) Amélioration moyenne brute (b) Amélioration moyenne normalisée

FIGURE 5.5 – Amélioration moyenne théorique en fonction de la valeur de la fonction objectif du One Max.

partie du processus de recherche est utilisée afin d'apprendre leur comportement. C'est pour cette raison que la courbe de l'opérateur 5-*flip* augmente graduellement au début. Quand l'évaluation de la solution atteint approximativement une valeur de 700, la fréquence de sélection des opérateurs 3-*flip* et 5-*flip* est supérieure à l'amélioration théorique normalisée. C'est l'inverse pour les opérateurs *bit-flip* et 1-*flip*. Au-dessus de 700, les quatre opérateurs ont des fréquences de sélection relativement équivalentes. En effet, comme nous le montre l'amélioration théorique moyenne (non normalisée), il est plus difficile de discriminer entre les opérateurs.

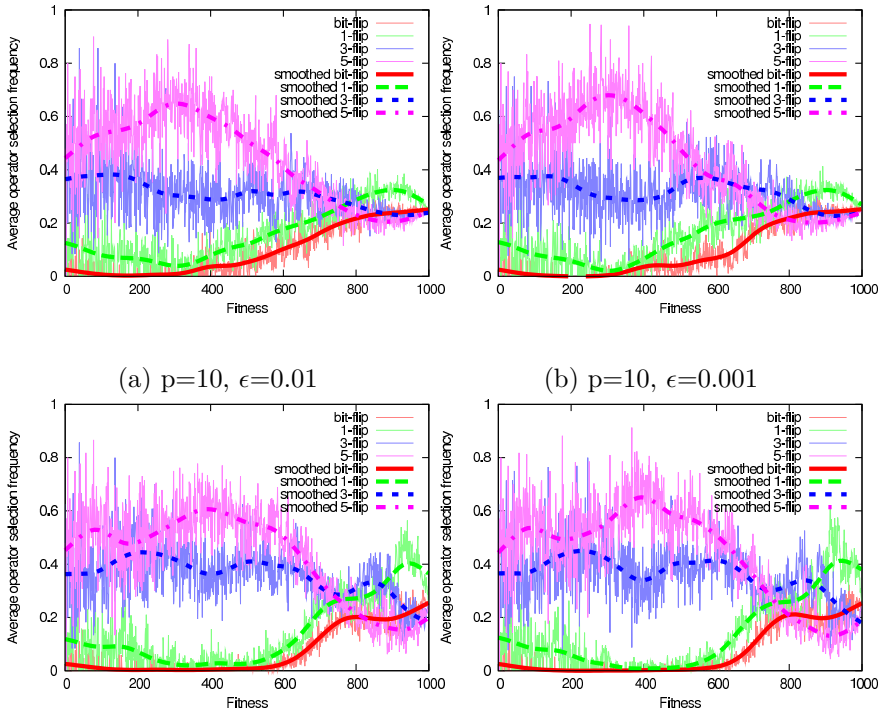
Remarquons que la fréquence de sélection de tous les opérateurs converge pour les plus grandes valeurs d'évaluation de la solution. Pour ces valeurs, aucun opérateur ne domine les autres sur plus de quelques itérations, voire aucun opérateur n'est dominé. Toutefois, alors que les courbes avec une longueur de fenêtre glissante de longueur 10 sont grossièrement équivalentes pour des valeurs élevées, les différences sont plus prononcées pour une longueur de fenêtre glissante fixée à 100. Ici, une fenêtre glissante de plus grande taille contient plus d'informations sur les performances passées. Il y a donc plus de chance d'avoir un mouvement améliorant dans la fenêtre, ce qui rend plus aisé le processus de discrimination entre opérateurs.

Nous supposons que le même phénomène est en jeu pour expliquer l'aspect moins lissé des courbes avec la plus grande longueur de fenêtre bien que cela soit un peu contre-intuitif. En effet, on pourrait s'attendre à ce qu'une plus petite longueur soit plus sensible aux changements. De même, notons que les fréquences d'utilisations des *bit-flips* et 1-*flips* sont beaucoup plus proches de 0 pour les valeurs entre 200 et 600 et que le changement autour de la valeur 700 est beaucoup plus prononcé pour une longue fenêtre glissante.

Notons toutefois que le problème One Max ne tire pas réellement parti du compromis exploitation-exploration puisque dans ce problème la diversification n'est pas utile. Il est également possible de noter que la fonction objectif du problème exprime déjà à la fois la qualité et la distance (de Hamming).

5.6 Expérimentations sur les problèmes de permutations

Nous testons notre méthode sur le *problème d'affectation quadratique* (QAP) et le *problème du voyageur de commerce asymétrique* (ATSP). Ces problèmes sont choisis car leurs solutions sont facilement représentables par un vecteur de variables. Une présentation détaillée de chacun de ces deux problèmes est donnée au Chapitre 4.



(a) p=10, ϵ=0.01 (b) p=10, ϵ=0.001

(c) p=100, ϵ=0.01 (d) p=100, ϵ=0.001

FIGURE 5.6 – Moyenne des fréquences de sélection en fonction de la valeur de la fonction objectif pour le One Max. Le paramètre p correspond à la longueur de la fenêtre et ϵ est le paramètre de l'utilité U_p.

La solution initiale pour le QAP est générée aléatoirement et l'heuristique de construction « voisin le plus proche » est utilisée pour le ATSP. Dans ces expériences nous utilisons dix opérateurs :

O1 $(\sigma_I, \mathcal{N}_E)$, première amélioration pour l'échange entre deux variables.
O2 $(\sigma_{BI}, \mathcal{N}_E)$, meilleure amélioration pour l'échange entre deux variables.
O3 $(\sigma_{BI5}, \mathcal{N}_E)$, choix aléatoire parmi les cinq meilleurs échanges entre deux variables.
O4 $(\sigma_{BI}, \mathcal{N}_E)^2$, deux meilleurs échanges consécutifs. Les variables échangées à l'étape une sont interdites à l'étape deux.
O5 $(\sigma_{BI}, \mathcal{N}_E)^3$, trois meilleurs échanges consécutifs. Les variables échangées aux étapes précédentes sont interdites à l'étape suivante.
O6 $(\sigma_T 3!, \mathcal{N}_E^2)$, meilleur échange entre 3 variables choisies aléatoirement.
O7 $(\sigma_T 4!, \mathcal{N}_E^3)$, meilleur échange entre 4 variables choisies aléatoirement.
O8 $(\sigma_T 5!, \mathcal{N}_E^4)$, meilleur échange entre 5 variables choisies aléatoirement.
O9 $(\sigma_T 6!, \mathcal{N}_E^5)$, meilleur échange entre 6 variables choisies aléatoirement.
O10 $(\sigma_R, \mathcal{N}_E^3)$, trois échanges aléatoires consécutifs entre deux variables.

Comme décrit plus loin, ces dix opérateurs ne donnent pas de bons résultats pour le ATSP. Nous ajoutons donc l'opérateur OA : le mouvement 3-opt.

OA $(\sigma_{BI}, \mathcal{N}_{EE}^2)$ sélectionne le meilleur voisin obtenu en cassant 3 arcs et en reconstruisant de nouveaux arcs de sorte qu'aucun sous-chemin soit inversé.

Dans ces expériences, nous mettons l'emphase sur des voisinages construits à partir de $\mathcal{N}_E$.D'autres voisinages pourraient également être utilisés (d'ailleurs ils peuvent également être exprimés en termes de $\mathcal{N}_E$).

5.6.1 Protocole expérimental

Chaque couple (algorithme, instance) est reproduit 30 fois. La taille de la fenêtre glissante est arbitrairement fixée à 100 et $\epsilon = 1$ pour le processus de sélection. Chaque exécution tourne au maximum pendant 40 000 itérations, voire moins si la meilleure valeur connue est atteinte.

Test statistique

Nous utilisons le test non paramétrique des rangs-signés de Wilcoxon [Sprent, 1992] comme suggéré dans [Chiarandini *et al.*, 2007] qui donne des indications quant à la méthodologie à adopter pour analyser les métaheuristiques. Soit deux algorithmes A et B, l'hypothèse nulle est la suivante : les médianes des distributions des solutions générées par A et B sont équivalentes. L'hypothèse est rejetée avec un seuil de confiance de 95%.

Une méthode statistique est dite non paramétrique si elle ne fait pas d'hypothèse sur la distribution de la population ou la taille de l'échantillon. La majorité des méthodes paramétriques font l'hypothèse que la population suit la loi normale. Dans le cas des métaheuristiques nous ne savons pas si cette hypothèse est valable. En effet peu d'études se sont penchées sur le problème. Toutefois [Ovacik *et al.*, 2000] semble indiquer que les distributions du coût des solutions sont bien approximées par la loi de Weibull. Cette dernière recouvre en fait une famille de lois. Elle est notamment interpolée avec la loi exponentielle et la loi de Rayleigh.

Instances

Les instances de test proviennent de QAPLIB [Burkard *et al.*, 1997] pour le QAP et de TSPLIB [Reinelt, 1991] pour l'ATSP. Les détails des instances utilisées sont présentés dans les Tableaux 5.1 et 5.2.

Les instances utilisées sont un sous-ensemble de chacune des deux bases d'instances. Dans les deux cas nous avons essayé de conserver la répartition initiale des deux bases en ce qui concerne les instances réelles et les instances générées automatiquement. Ainsi pour le QAP la majorité des instances sont générées alors que c'est l'inverse pour l'ATSP. Pour le QAP nous nous restreignons aux instances de taille moyenne.

5.6.2 Résultats et discussion

Le Tableau 5.3 présente les résultats pour le QAP et le Tableau 5.4 ceux du ATSP. Le pourcentage moyen d'écart entre la meilleure valeur connue (BKV) et le choix proportionnel aux valeurs d'utilité suivantes sont donnés : distribution uniforme, qualité, nombre de solutions Pareto-dominées en utilisant la distance d_1^P (ParDom d_1^P) et la distance $d_{\mathcal{N}_E}^P$ (ParDom $d_{\mathcal{N}_E}^P$). Les résultats pour la recherche tabou robuste (RoTS) pour le QAP sont aussi donnés à titre de comparaison. Notre objectif ici est simplement de donner un marqueur de référence, et de montrer que notre méthode, utilisant un ensemble non optimisé d'opérateurs est capable d'atteindre des résultats intéressants. Les meilleurs résultats pour chaque instance sont indiqués en gras. Les résultats de RoTS ne sont pas pris en compte car ils sont meilleurs ou équivalents sur toutes les instances sauf deux. Par ailleurs RoTS, tel qu'il a été conçu pour le QAP, ne peut être utilisé pour l'ATSP sans modification majeur du code source. Dans le Tableau 5.4, la colonne ParDom10 contient les résultats lorsque seuls les dix même opérateurs que pour le QAP sont utilisés avec la distance d_1^P.

Instance	Taille	BKV	Opt.	Commentaire
bur26a	26	5426670	Oui	Temps moyen de frappe d'un sténotypiste et fréquences des paires de lettre dans différentes langues.
chr25a	25	3796	Oui	Matrice d'adjacence d'un arbre pondéré et matrice d'adjacence d'un graphe complet.
kra30a	30	88900	Oui	[kra] Données utilisées pour planifier la
kra30b	30	91420	Oui	*Klinikum Regensburg* en Allemagne.
nug20	20	2570	Oui	[nug] Distances de Manhattan de grilles
nug30	30	6124	Oui	rectangulaires.
sko42	42	15812	RoTS	[sko] Distances rectangulaires et
sko49	49	23386	RoTS	nombres pseudoaléatoires.
tai30a	30	1818146	RoTS	[taixxa] Générées uniformément.
tai50a	50	4938796	ITS	
tai30b	30	637117113	Oui	[taixxb] Asymétriques générées
tai50b	50	458821517	RoTS	aléatoirement.
wil50	50	48816	SA	Distances rectangulaires.

TABLEAU 5.1 – Instances du QAP utilisées dans ce document. Ces instances proviennent de QAPLIB et sont décrites dans [Burkard *et al.*, 1997]. La colonne BKV correspond à la meilleure valeur connue. La colonne Opt. contient Oui si cette meilleure valeur est l'optimum. Dans le cas contraire, elle contient l'algorithme ayant été utilisé pour trouver cette valeur (RoTS – Robust Tabou, ITS – Recherche Tabou Itérée, SA – Recuit Simulé).

Instance	Taille	Opt	Commentaire
ry48p	48	14422	Distances euclidiennes symétriques perturbées afin
kro124p	101	36230	d'être asymétriques.
ft53	53	6905	[ft] Problème d'ordonnancement de tâches dans le
ft70	70	38673	processus de coloration dans la production de résine.
ftv38	39	1530	
ftv44	45	1613	
ftv47	48	1776	
ftv55	56	1608	
ftv64	65	1839	[ftv] Problème de livraison de médicaments à
ftv70	71	1950	Bologne. Les instances ftv90 à ftv140 utilisent un
ftv90	91	1579	sous ensemble des données de ftv160.
ftv100	100	1788	
ftv120	121	2166	
ftv140	141	2420	
ftv160	161	2683	
rbg323	323	1326	[rbg] Problème de desserte de rayonnages (*stacker*
rbg403	403	2465	*crane problem*)
rbg443	443	2720	

TABLEAU 5.2 – Instances du TSP asymétrique, provenant de TSPLIB utilisées dans ce document. Les descriptions sont celles rapportées dans [Cirasella *et al.*, 2001].

Instance	BKV	Uniforme	Qualité	ParDom d_1^P	ParDom $d_{\mathcal{N}_E}^P$	RoTS
bur26a	5426670	0.00	0.00	**0.00**	0.00	0.00
chr25a	3796	11.79	10.35	10.19	**9.38**	7.09
kra30a	88900	**0.47**	0.49	0.50	0.73	0.07
kra30b	91420	0.11	0.12	**0.06**	0.10	0.02
nug20	2570	**0.00**	**0.00**	**0.00**	**0.00**	0.00
nug30	6124	0.12	0.09	0.06	**0.05**	0.01
sko42	15812	0.16	0.15	**0.09**	0.12	0.03
sko49	23386	0.27	0.19	**0.19**	**0.19**	0.13
tai30a	1818146	1.13	1.18	0.80	**0.63**	0.51
tai50a	4941410	1.85	1.82	1.38	**1.37**	1.39
tai30b	637117113	0.15	0.11	**0.10**	0.13	0.03
tai50b	458821517	**0.17**	0.19	0.27	0.54	0.15
wil50	48816	0.08	**0.07**	0.08	0.09	0.05

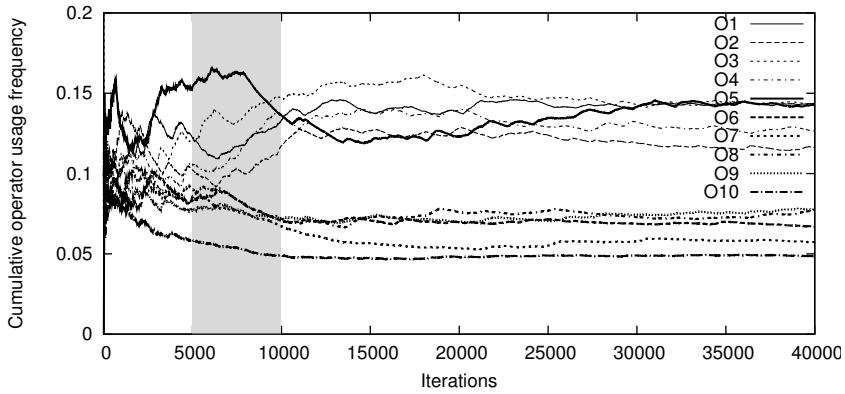
TABLEAU 5.3 – Résultats du QAP – Pourcentage moyen de différence entre la meilleure valeur connue (BKV) et les résultats obtenus par une sélection des opérateurs proportionnelle aux valeurs d'utilité suivantes : distribution uniforme, qualité, nombre de solutions Pareto-dominées en utilisant la distance d_1^P (ParDom d_1^P) et la distance $d_{\mathcal{N}_E}^P$ (ParDom $d_{\mathcal{N}_E}^P$). Les résultats pour le tabou robuste sont donnés à titre de comparaison.

Pour le QAP ParDom d_1^P et ParDom $d_{\mathcal{N}_E}^P$ partagent la majorité des résultats en gras. Cependant, selon le test de Wilcoxon, ParDom d_1^P semble être le meilleur algorithme lorsqu'il est comparé à la sélection uniforme et à celle basée sur la qualité. A contrario, les résultats de ParDom $d_{\mathcal{N}_E}^P$ ne sont pas statistiquement significatifs. Les résultats semblent donc montrer que ParDom d_1^P est meilleur que ParDom $d_{\mathcal{N}_E}^P$ (l'hypothèse nulle ne peut toutefois pas être rejetée si on les compare l'un à l'autre).

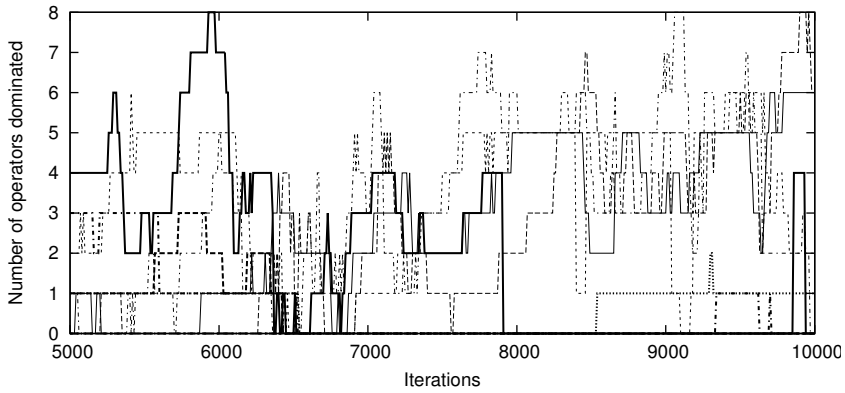
Pour le ATSP, l'utilisation des dix opérateurs utilisés pour le QAP n'est pas concluante. Cela s'explique aisément puisqu'aucun d'entre eux ne prend en compte la nature cyclique des solutions. L'ajout de l'opérateur 3-opt produit une amélioration notable. Avec un unique opérateur spécifique à l'ATSP, la population des opérateurs demeure très biaisée envers l'ATSP. Ceci résulte en une amélioration bien plus visible que dans le cas du QAP lorsque l'on compare les trois méthodes de sélection non triviale avec la sélection uniforme. Ici, ParDom d_1^P et ParDom $d_{\mathcal{N}_E}^P$ partagent les meilleurs résultats avec toutefois la majorité pour ParDom $d_{\mathcal{N}_E}^P$. En termes statistiques, les deux méthodes de sélection par Pareto dominance sur onze opérateurs sont meilleures que la sélection uniforme ainsi que celle basée sur la qualité. Lorsque l'on compare ParDom d_1^P à ParDom $d_{\mathcal{N}_E}^P$, l'hypothèse nulle peut être rejetée. Cela démontre clairement que ParDom $d_{\mathcal{N}_E}^P$ est la mieux adaptée à l'ATSP.

La Figure 5.7a représente la fréquence cumulative de l'application des opérateurs pour une exécution arbitraire du QAP et la Figure 5.8 représente la même chose pour l'ATSP. On peut observer que les opérateurs sont clairement séparés en deux groupes : un dont la fréquence est supérieure à la moyenne (0,1) et l'autre inférieure. Le détail nous révèle que le premier est le groupe qui améliore le plus la qualité alors que le second est celui qui perturbe le plus les solutions sans apporter de changement positif à la qualité.

L'opérateur qui est sélectionné le plus souvent est celui qui démontre sa constance dans l'amélioration de la solution tout en modifiant un certain nombre de variables au cours de ses cent dernières applications. Comme on peut le voir à la Figure 5.7a, l'opérateur O5 se comporte très bien au début de la recherche. Sa performance baisse



(a) Fréquences cumulées de sélection.



(b) Nombre d'opérateurs dominés correspondant aux itérations grisées en (a).

FIGURE 5.7 – Une exécution de l'instance tai50a (QAP).

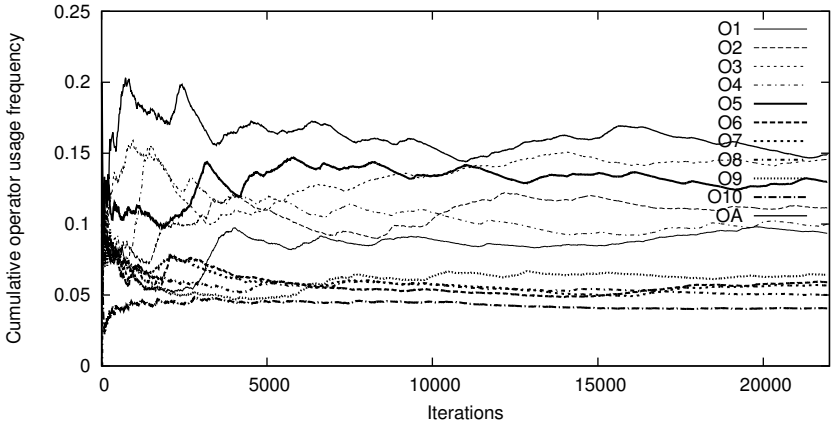


FIGURE 5.8 – Une exécution de l'instance ftv55 (ATSP).

Instances	BKV	ParDom10	Uniforme	Qualité	ParDom d_1^P	ParDom $d_{\mathcal{N}_E}^P$
ry48p	14422	4.31	0.66	0.35	0.20	**0.17**
ft53	6905	12.61	1.11	0.52	0.19	**0.17**
ft70	38673	5.89	0.75	0.46	0.08	**0.05**
ftv38	1530	6.14	0.72	0.43	0.31	**0.26**
ftv44	1613	9.07	0.73	0.38	**0.24**	0.25
ftv47	1776	11.01	0.48	0.19	0.14	**0.11**
ftv55	1608	14.20	0.97	0.21	0.14	**0.09**
ftv64	1839	17.13	1.39	0.78	**0.55**	0.58
ftv70	1950	17.04	1.54	0.92	**0.64**	0.64
ftv90	1579	25.43	2.18	1.25	0.98	**0.82**
ftv100	1788	24.57	2.84	1.50	1.17	**1.05**
ftv120	2166	25.39	3.46	2.37	2.14	**2.13**
ftv140	2420	32.84	4.72	3.29	3.00	**2.97**
ftv160	2683	35.24	6.00	3.47	3.47	**3.33**
kro124p	36230	18.52	2.33	1.36	1.15	**1.06**
rbg323	1326	7.99	0.07	0.01	**0.00**	**0.00**
rbg403	2465	1.15	**0.00**	**0.00**	**0.00**	**0.00**
rbg443	2720	1.46	**0.00**	**0.00**	**0.00**	**0.00**

TABLEAU 5.4 – Résultats expérimentaux pour l'ATSP – Pourcentage moyen de différence entre la meilleure valeur connue (BKV) et les résultats obtenus par une sélection des opérateurs proportionnelle aux valeurs d'utilité suivantes : nombre de solutions Pareto-dominées en utilisant la distance d_1^P (ParDom10) avec les dix opérateurs utilisés pour le QAP, et distribution uniforme, qualité, nombre de solutions Pareto-dominées en utilisant la distance d_1^P (ParDom d_1^P) et la distance $d_{\mathcal{N}_E}^P$ (ParDom $d_{\mathcal{N}_E}^P$) avec onze opérateurs pour le reste.

ensuite pour augmenter à nouveau jusqu'au niveau des autres « bons » opérateurs, la recherche stagnant à la fin. Ceci illustre le fait qu'un opérateur n'est pas toujours le meilleur tout au long de la recherche et qu'il est important pour le mécanisme de sélection d'être influencé par l'étape à laquelle se trouve la recherche. A contrario, dans la Figure 5.8, l'opérateur OA est constamment l'opérateur le plus souvent sélectionné. C'est un résultat auquel on pouvait s'attendre puisque c'est le seul opérateur spécifique à l'ATSP. Il se comporte donc forcément mieux que les autres opérateurs. Les opérateurs qui améliorent la qualité pour le QAP gardent toutefois un intérêt pour l'ATSP.

Afin d'expliquer pourquoi les opérateurs améliorant la qualité sont sélectionnés le plus souvent, on peut remarquer que la modification d'une solution pour la rendre meilleure requiert également la modification de ses variables et donc de la distance depuis la dernière solution. Les opérateurs améliorant la qualité sont donc plus enclin à dominer les opérateurs dont la seule action est de causer de perturbations dans la solution.

La Figure 5.7b montre le nombre d'opérateurs dominés par chaque opérateur sur une partie de la recherche de la Figure 5.7a (région grisée). On peut observer qu'à certains moments de la recherche un opérateur domine presque tous les autres, ayant ainsi la probabilité la plus grande d'être sélectionné. Cette probabilité accrue se reflète dans les courbes de fréquences cumulées. On peut observer que pendant les quelques centaines d'itérations avant l'itération 7 000, aucun opérateur n'est considéré comme bien meilleur que les autres. Ceci transforme le processus de sélection en

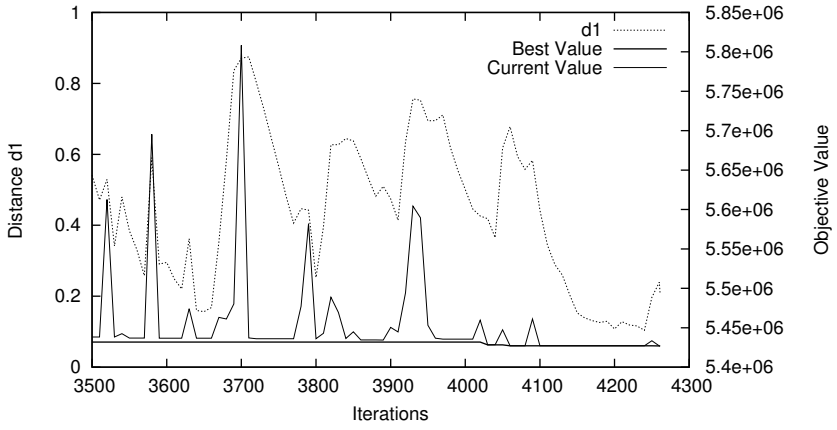


FIGURE 5.9 – Partie finale de la recherche pour une exécution de l'instance bur26a (QAP).

une simple sélection uniforme. Une sélection plus discriminante émerge ensuite, des opérateurs différents étant apparemment mieux adaptés à la suite de la recherche.

Une autre observation intéressante, plus particulièrement à la Figure 5.8, est que l'augmentation significative de la fréquence de sélection d'un opérateur implique souvent l'inverse pour un autre opérateur.

La Figure 5.9 est un instantané d'une partie de la recherche sur une instance de QAP. Elle représente la meilleure valeur obtenue, la valeur courante de la fonction objectif et la distance du chemin à la nouvelle solution. Elle met en valeur que le contrôle, gérant le compromis entre qualité et diversité, est capable d'échapper à des minima locaux et également d'atteindre de bonnes solutions. La corrélation entre notre mesure de distance et la qualité apparaît aussi clairement.

Complément expérimental

Dans les expériences précédentes nous avons fixé la longueur des fenêtres glissantes à 100 à lumière des expériences sur le One Max. Considérons à nouveau cette valeur. Le Tableau 5.5 présente les résultats pour le QAP de la sélection proportionnelle associée à l'utilité de Pareto dominance avec la distance d_1^P pour des fenêtres de taille 1 (instantanée), 10 et 100.

On remarque que la taille 100 a un léger avantage par rapport aux autres. Toutefois le test statistique ne nous permettant pas de conclure que les résultats pour la taille 100 diffèrent significativement des autres. Néanmoins nous conserverons cette valeur dans les chapitres suivants.

Comme nous l'avons vu la sélection uniforme se comporte relativement bien, notamment avec le QAP. Dans le Tableau 5.6 nous présentons les résultats pour la sélection uniforme et la sélection proportionnelle associée à l'utilité de Pareto dominance avec la distance d_1^P lorsque l'on utilise les dix opérateurs précédents auxquels nous ajoutons cinq opérateurs identité. Ces opérateurs ne font donc rien : ils retournent simplement la solution courante.

On peut observer que la différence entre la sélection uniforme et la sélection proportionnelle a augmentée. Ainsi cette seconde méthode se comporte mieux face à un ensemble d'opérateurs moins bien choisi.

Enfin nous avons également testé quelques sous ensembles de deux ou trois opérateurs en prenant ceux ayant les meilleures performances en intensification ou en diversification. Ces petits ensembles d'opérateurs font beaucoup moins bien que dix opérateurs. Par exemple, dans le Tableau 5.7 nous donnons les résultats pour la

Instances	Instantané	Fenêtre 10	Fenêtre 100
bur26a	**0,00**	**0,00**	**0,00**
chr25a	9,54	**9,04**	10,19
kra30a	0,59	0,59	**0,50**
kra30b	0,11	0,12	**0,06**
nug20	**0,00**	**0,00**	**0,00**
nug30	0,10	0,08	**0,06**
sko42	0,14	0,15	**0,09**
sko49	0,23	**0,17**	0,19
tai30a	1,01	1,19	**0,80**
tai50a	1,72	1,59	**1,38**
tai30b	0,12	0,11	**0,10**
tai50b	0,28	0,30	**0,27**
wil50	**0,07**	0,07	0,08

TABLEAU 5.5 – Comparaison des résultats selon la taille de fenêtres (pourcentage moyen de différence par rapport à la meilleure valeur connue).

Instances	Uniforme	ParDom d_1^P
bur26a	**0,00**	0,00
chr25a	11,26	**10,74**
kra30a	0,71	**0,67**
kra30b	0,20	**0,16**
nug20	**0,00**	**0,00**
nug30	0,28	**0,10**
sko42	0,26	**0,15**
sko49	0,31	**0,24**
tai30a	1,50	**1,10**
tai50a	2,17	**1,93**
tai30b	0,12	**0,11**
tai50b	0,26	**0,24**
wil50	0,12	**0,10**

TABLEAU 5.6 – Résultats pour dix opérateurs initiaux et cinq opérateurs identité (pourcentage moyen de différence par rapport à la meilleure valeur connue).

sélection uniforme et la sélection proportionnelle associée à l'utilité de Pareto dominance avec la distance d_1^P lorsque l'on utilise les opérateurs O3 et O5 (intensification) et O11 (diversification). Les résultats sont réellement mauvais et les deux sélections se comportent de la même façon.

5.7 Conclusion

Ce chapitre à été consacré à introduire notre approche de sélection d'opérateurs pour la recherche locale à travers des méthodes simples.

Nous avons tout d'abord proposé un cadre général pour la recherche locale en définissant un opérateur de base comme une combinaison d'un voisinage et d'une fonction de sélection. Ceci nous a permis de proposer des opérateurs dans un contexte de chaînes binaires et de permutations, les problème que nous traitons dans ce chapitre étant de ces types.

Instances	Uniforme	ParDom d_1^P
bur26a	0,24	**0,23**
chr25a	35,66	**34,73**
kra30a	6,01	**5,28**
kra30b	**3,28**	3,95
nug20	0,33	**0,25**
nug30	1,39	**1,29**
sko42	1,78	**1,65**
sko49	**1,36**	1,49
tai30a	**0,93**	1,12
tai50a	**1,53**	1,60
tai30b	**6,65**	8,06
tai50b	**5,96**	6,23
wil50	1,04	**0,99**

TABLEAU 5.7 – Résultats pour trois opérateurs (pourcentage moyen de différence par rapport à la meilleure valeur connue).

La performance des opérateurs est déterminée par l'historique de leur applications passée selon des mesures de qualité et de diversité. Nous avons proposé deux mesures de diversité. Les deux caractéristiques divergentes que sont la qualité et la diversité sont combinées grâce à un compromis implicite. Celui-ci est repris du domaine de l'optimisation multiobjectif : la Pareto dominance.

La validation initiale a été faite par rapport à un problème binaire, le One Max. Ceci nous a permis de comparer les résultats empiriques aux résultats théoriques attendus.

Nous sommes ensuite passés à des problèmes de permutation, le problème d'affectation quadratique et celui du voyageur de commerce. Nous avons constaté que selon le type d'ensemble d'opérateurs utilisé la sélection uniforme pouvait avoir de bonnes performances. Avec un ensemble d'opérateurs moins bien choisi, les approches autres que la sélection uniforme prennent le dessus.

L'utilisation du compromis qualité-diversité de la Pareto dominance est validé par rapport à la sélection uniforme et la sélection uniquement basée sur la qualité. Les deux mesures de diversité montrent des caractéristiques intéressantes mais ne sont pas suffisamment différentes pour permettre de privilégier l'une par rapport à l'autre.

Nous avons constaté que l'utilisation des opérateurs changeait au fil de la recherche, et ceci de façon substantielle à certains moments. Ceci valide l'idée de sélectionner dynamiquement les opérateurs.

Chapitre 6

Stratégies de sélection adaptative

Sommaire

6.1 Introduction 90
6.2 Mesure de distance entre opérateurs 90
6.3 Compromis statique : expériences 91
6.3.1 Protocole expérimental 91
6.3.2 Résultats et discussion 93
6.4 Stratégie de contrôle du compromis 93
6.5 Compromis dynamique : expériences 95
6.5.1 Protocole expérimental 95
6.5.2 Résultats et discussion 96
6.6 Conclusion 100

6.1 Introduction

Dans ce chapitre nous nous intéressons à différentes mesures d'utilité, exprimant un compromis entre exploration et exploitation, ainsi qu'à des méthodes de sélection d'opérateurs en fonction de l'utilité.

En particulier, nous souhaitons considérer des mesures d'utilité dont le compromis est explicitement défini. La valeur de compromis peut être fixe ou définie et modifiée pendant l'exécution par une stratégie intégrée au solveur. Ceci implique bien sûr de nouveaux paramètres : pour le compromis lui-même et éventuellement pour la stratégie manipulant le compromis.

Le cœur du problème reste le même que dans le chapitre précédent : la sélection en ligne du prochain mouvement à appliquer dans un processus de recherche locale afin de s'adapter à la réalité du sous-ensemble de l'espace de recherche en cours d'exploration. La Poursuite Adaptative (décrite dans la Section 2.3.3) est un exemple d'une telle stratégie affectant une probabilité de sélection à chaque opérateur pour les algorithmes génétiques. Sa caractéristique principale est l'utilisation du concept du gagnant remporte tout (winner-takes-all) qui se traduit par la forte augmentation de la probabilité de sélection du meilleur opérateur tout en diminuant la probabilité de sélection des autres opérateurs.

Comme nous l'avons déjà évoqué, l'ajout de paramètres complexifie aussi bien la tâche du concepteur de la méthode de recherche que l'utilisation de ladite méthode par l'utilisateur final puisqu'il lui reste généralement des choix paramétriques à faire en fonction du problème qu'il souhaite résoudre. Ceci augmente les difficultés rencontrées lorsque l'on souhaite adapter une méthode à un nouveau problème. La conception de solveurs toujours plus efficaces produit fréquemment des systèmes fortement complexes, qui requièrent une somme non-négligeable de connaissance d'expert, par exemple pour en sélectionner intelligemment les paramètres. Cette tendance mène donc vers un besoin croissant pour des algorithmes de configuration automatique de paramètres.

Dans cette optique, nous utiliserons des méthodes de réglages mentionnées à la Section 2.2 afin de nous aider à choisir les nouveaux paramètres que nous devrons introduire. La configuration automatique est exécutée hors-ligne en faisant tourner l'algorithme sur des instances de test. De plus, nous examinerons la robustesse des méthodes proposées, c'est-à-dire leur aptitude à conserver un comportement correct sur différents problèmes sans demander un effort de réglage supplémentaire. Comme au chapitre précédent, nous nous concentrerons sur deux problèmes de permutations.

La majorité des travaux présentés dans ce chapitre ont également été publiés dans [Veerapen *et al.*, 2012a; Veerapen *et al.*, 2012b; Veerapen *et al.*, 2012c].

6.2 Mesure de distance entre opérateurs

Intéressons-nous à la façon d'introduire un compromis entre intensification et diversification. Dans le chapitre précédent nous avons examiné l'utilité basée sur la Pareto dominance U_P.

L'utilité U_P de l'opérateur o dans l'ensemble O de n opérateurs est alors définie par

$$U_P(o) = |\{o' | o' \in O, o \succ o'\}| + \epsilon \tag{6.1}$$

où ϵ est une valeur positive afin d'obtenir une utilité non nulle et $o \succ o'$ signifie que o domine o'.

Cette utilité ne permet pas de spécifier explicitement un compromis entre l'impact sur la qualité q et sur la distance par rapport au chemin de recherche d. Nous

introduisons un paramètre de pondération, α, afin d'influencer ce compromis.

$$U_\alpha(o) = \alpha d + (1 - \alpha)q \tag{6.2}$$

La mesure *Compass*, présentée à la Section 2.3.3, permet également d'agréger la qualité et la distance mais le compromis entre ces deux mesures est défini par une direction de recherche.

Nous proposons une nouvelle mesure d'utilité basée sur les distances ou, plus correctement, sur les déplacements entre opérateurs puisque l'on considéra aussi bien la magnitude que le signe. Cette nouvelle mesure est inspirée de la Pareto dominance dans le sens où l'on considère des relations entre opérateurs et non les valeurs brutes de qualité q et de distance d. Commençons par définir ce que sont les déplacements.

Définition 6.2.1. Soient deux opérateurs o_1 and o_2 définis par les couples qualité-distance (q_1, d_1) et (q_2, d_2), nous définissons le *déplacement rectilinéaire pondéré* de o_1 vers o_2 tel que

$$d_\alpha(o_1, o_2) = \alpha(d_1 - d_2) + (1 - \alpha)(q_1 - q_2) \tag{6.3}$$

Puisque nous souhaitons considérer les relations entre opérateurs, il nous faut choisir comment nous allons déterminer l'agrégation des d_α en nous rappelant que nous cherchons à maximiser la qualité aussi bien que la distance.

Nous allons examiner trois possibilités :

– la somme des déplacements vers les autres opérateurs

$$U_\alpha^{\Sigma}(o) = max(0, \sum_{i=1}^{n} d_\alpha(o, o_i))$$

– la somme des déplacements positifs vers les autres opérateurs

$$U_\alpha^{\Sigma+}(o) = \sum_{i=1}^{n} max(0, d_\alpha(o, o_i))$$

– la somme des déplacements vers les autres opérateurs dominés

$$U_\alpha^{\Sigma\succ}(o) = \sum_{i=1, o\succ o_i}^{n} d_\alpha(o, o_i)$$

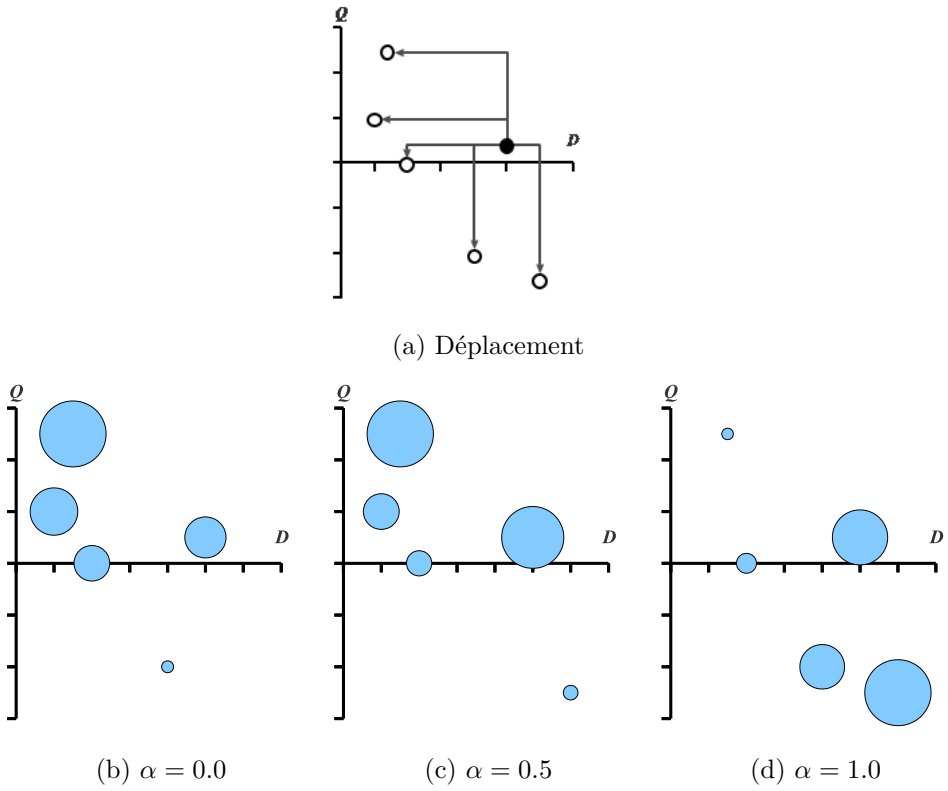
La Figure 6.1 tente d'illustrer l'intérêt d'utiliser la somme des déplacements comme valeur d'utilité d'un opérateur. Tout d'abord, les déplacements, et leur somme, sont utiles puisqu'ils décrivent quantitativement (la magnitude) et, dans une moindre mesure, qualitativement (le signe ou la direction) la relation entre opérateurs. Ensuite, la pondération introduit naturellement un compromis quantifiable vers la qualité et la diversité.

6.3 Compromis statique : expériences

Pour ces premières expériences avec un compromis, nous examinerons d'abord ce compromis de façon statique, c'est-à-dire en le fixant au début de la recherche et en conservant cette valeur tout au long de la recherche sans la modifier.

6.3.1 Protocole expérimental

À nouveau, chaque opérateur est une combinaison d'un voisinage et d'une fonction de sélection. De plus grands voisinages sont créés par composition de ce voisinage avec lui-même.



(a) Déplacement

(b) $\alpha = 0.0$ (c) $\alpha = 0.5$ (d) $\alpha = 1.0$

FIGURE 6.1 – (a) Une population hypothétique de six opérateurs montrant les déplacements non pondérés relativement à un opérateur ; (b)–(d) α détermine l'importance (taille de la bulle) de chaque opérateur pour $U_\alpha^{\Sigma+}$.

Opérateurs

Nous réutilisons les mêmes opérateurs qu'au chapitre précédent : les dix opérateurs O1–O10 pour le QAP, les cinq premiers étant plutôt destinés à l'exploitation et les cinq autres à la diversification. Nous ne considérerons pour le moment que le QAP et nous réintroduirons l'ATSP par la suite.

Ces dix opérateurs sont relativement peu perturbateurs : la portée des transformations qu'ils sont en mesure d'engendrer dans une solution est modeste. En effet, ils modifient un petit nombre de variables (entre deux et six). Afin d'augmenter la difficulté de la tâche du mécanisme de sélection, nous introduisons deux nouveaux opérateurs vraiment perturbateurs modifiant respectivement 25 % et 50 % des variables d'une solution :

O11 $(\sigma_R, \mathcal{N}_E)^{\lfloor n/8 \rfloor}$, $\lfloor n/8 \rfloor$ échanges aléatoires consécutifs entre deux variables,

O12 $(\sigma_R, \mathcal{N}_E)^{\lfloor n/4 \rfloor}$, $\lfloor n/4 \rfloor$ échanges aléatoires consécutifs entre deux variables.

Ces deux opérateurs peuvent participer à l'exploration du paysage de recherche mais seront trop perturbateurs pour être utilisés fréquemment. On peut donc les considérer comme des mécanismes de redémarrage de la recherche. Ceci influencera donc négativement la sélection uniforme qui accorde la même probabilité de sélection à chaque opérateur. Bien entendu, les mécanismes de sélection d'opérateurs que nous allons étudier n'ont pas connaissance de ces informations.

Méthodes testées et paramétrage

Nous effectuerons des comparaisons entre la sélection uniforme, et la sélection proportionnelle (PR) pour les trois sommes des déplacements $U_\alpha^{\Sigma?}$.

Le paramètre pour U_P est $\epsilon = 0.1$ (les valeurs 0.01, 0.05, 0.2 et 1 ont également

FIGURE 6.2 – Les modules Stratégie, Contrôleur et Recherche locale

été testées mais 0.1 a obtenu les meilleurs résultats sur une majorité d'instances). Pour $U_\alpha^{\Sigma+}$, le paramètre α a été testé avec les valeurs 0.2, 0.5 et 0.8.

Pour chaque instance, chaque algorithme est lancé 30 fois et un maximum de 40 000 itérations lui sont accordées par exécution. La longueur de toutes les fenêtres glissantes est fixée à 100 (pour le chemin et les mesures de qualité et de distance pour chaque opérateur).

6.3.2 Résultats et discussion

Les Tableaux 6.1 représentent les mêmes résultats. Dans le premier sont présentées les valeurs obtenues pour chacun des trois compromis α et la comparaison avec la Pareto dominance. Dans le second, on ne retient que les meilleures valeurs obtenues pour tout α confondu. Ceci permet de mieux appréhender l'utilité qui gagnerait à ce que son compromis soit modifié automatiquement au cours de la recherche. Les chiffres en gras indiquent les meilleures valeurs.

Il apparaît clairement que $U_\alpha^{\Sigma+}$ est la meilleure des trois utilités basées sur les déplacements. Ces meilleures valeurs se retrouvent parmi les trois valeurs α utilisées. L'utilité basée sur la Pareto dominance, U_P, donne également de bons résultats. Ces deux utilités arrivent sans peine à devancer la sélection uniforme : ici l'ensemble des opérateurs est plus diversifié, cela paie donc d'utiliser une utilité plus avancée et une sélection proportionnelle. En terme de performance générale, U_P et le meilleur de $U_\alpha^{\Sigma+}$ sont équivalentes.

Si l'on considère U_α^{Σ}, le problème est que l'acceptation de valeurs de déplacements négatives annule souvent les déplacements positifs. Ceci produit des résultats qui sont en général moins bons que la sélection uniforme. La faible performance de $U_\alpha^{\Sigma\succ}$ peut, quant à elle, être éventuellement expliquée par le fait que, comme elle est basée sur la Pareto dominance, et qu'il n'y a pas de ϵ pour assurer une probabilité minimum de sélection, les opérateurs de part et d'autre du spectre d'exploitation-exploration n'ont pas de réelles chances d'être sélectionnés puisqu'il ne domine en général pas les autres opérateurs.

Nous essaierons donc d'améliorer les performances de $U_\alpha^{\Sigma+}$ en lui adjoignant une stratégie permettant de faire varier le compromis.

6.4 Stratégie de contrôle du compromis

Dans la section précédente nous avons utilisé un compromis statique. Nous examinons maintenant comment ce compromis peut s'adapter en ligne à l'état du processus de recherche.

Afin de faire varier α, nous introduisons le module Stratégie qui se fixe au contrôleur comme illustré sur le schéma 6.2. Il communique avec ce dernier et son rôle est de faire varier les paramètres en direct.

Nous considérons une stratégie de diversité « correcte » (CD), un concept développé dans [Maturana et Saubion, 2008b] pour les algorithmes évolutionnaires. Dans cette publication les auteurs comparent plusieurs stratégies de contrôle du compromis dans le contexte des algorithmes évolutionnaires et CD est la plus performante.

Instance	BKV	Uniforme	U_P	$U_\alpha^{\Sigma+}$	U_α^{Σ}	$U_\alpha^{\Sigma\succ}$
bur26a	5426670	0.03	**0.00**	0.02 0.01 **0.00**	0.09 0.13 0.16	0.16 0.09 0.15
chr25a	3796	20.18	**10.67**	13.78 14.94 12.11	33.53 30.75 28.70	34.05 28.68 29.75
kra30a	88900	2.49	**0.79**	1.57 1.63 0.89	5.14 4.39 3.75	4.67 4.55 4.42
kra30b	91420	1.11	0.21	**0.16** 0.45 0.32	3.06 2.78 2.50	3.32 3.03 2.37
nug20	2570	0.12	0.01	**0.00** 0.03 **0.00**	1.02 0.56 1.15	1.09 0.89 1.45
nug30	6124	1.24	0.20	0.31 **0.19** 0.39	1.67 1.43 1.60	1.27 1.54 1.71
sko42	15812	2.28	0.29	**0.19** 0.28 0.67	1.91 1.38 2.01	1.65 1.63 2.14
sko49	23386	2.48	0.36	**0.21** 0.27 0.81	1.37 1.34 2.31	1.46 1.60 1.74
tai30a	1818146	2.59	1.26	**1.17** 1.27 1.68	2.05 1.86 3.18	2.16 1.78 3.41
tai50a	4941410	4.20	2.16	**1.58** 1.59 2.83	2.13 2.61 4.11	2.27 2.80 4.11
tai30b	637117113	0.43	**0.13**	0.44 0.35 0.16	6.65 3.90 3.49	5.21 3.53 1.74
tai50b	458821517	2.36	**0.25**	0.30 0.39 0.37	4.14 3.13 2.56	4.33 3.92 2.66
wil50	48816	1.24	0.16	**0.09** 0.11 0.28	0.95 0.77 1.24	0.94 0.73 0.84

(a) Les résultats pour trois compromis $\alpha = 0.2$, $\alpha = 0.5$ et $\alpha = 0.8$

Instance	BKV	Uniform	U_P	$U_\alpha^{\Sigma+}$	U_α^{Σ}	$U_\alpha^{\Sigma\succ}$
bur26a	5426670	0.03	**0.00**	0.00	0.09	0.09
chr25a	3796	20.18	**10.67**	12.11	28.70	28.68
kra30a	88900	2.49	**0.79**	0.89	3.75	4.42
kra30b	91420	1.11	0.21	**0.16**	2.50	2.37
nug20	2570	0.12	0.01	**0.00**	0.56	0.89
nug30	6124	1.24	0.20	**0.19**	1.43	1.27
sko42	15812	2.28	0.29	**0.19**	1.38	1.63
sko49	23386	2.48	0.36	**0.21**	1.34	1.46
tai30a	1818146	2.59	1.26	**1.17**	1.86	1.78
tai50a	4941410	4.20	2.16	**1.58**	2.13	2.27
tai30b	637117113	0.43	**0.13**	0.16	3.49	1.74
tai50b	458821517	2.36	**0.25**	0.30	2.56	2.66
wil50	48816	1.24	0.16	**0.09**	0.77	0.73

(b) Ici seule est donnée la meilleure valeur obtenue pour les trois compromis.

TABLEAU 6.1 – Résultats des différents déplacements sur le QAP.

De part sa conception pour les algorithmes évolutionnaires, CD utilisent le concept de population. Dans notre framework, CD utilise les solutions présentes dans la fenêtre glissante, la même fenêtre qui est utilisée pour le calcul de la distance. La stratégie CD utilise la qualité des solutions de la fenêtre comme moyen d'évaluation de la diversité du chemin. Si le nombre de solutions ayant la même qualité est au-dessus d'un certain seuil T_{max} alors on suppose que le chemin est très homogène – ceci peu indiquer un bassin d'attraction – et la diversité commandée est incrémentée d'un pas s_{inc}. De façon symétrique, si le nombre de solutions ayant la même qualité est au-dessous d'un autre seuil T_{min} la diversité commandée est décrémentée d'un pas s_{dec}, la supposition étant que le chemin est incapable de rester proche d'optima locaux de bonne qualité, ce qui démontre une faible exploitation.

La stratégie CD vise à améliorer la performance du solveur, toutefois elle introduit quatre nouveaux paramètres afin d'en contrôler un seul. Ceci demande donc un effort conséquent de réglage. On pourra considérer que CD est vraiment utile si le même paramétrage peux être utilisé sur un nouveau problème sans réel effort de réglage supplémentaire.

6.5 Compromis dynamique : expériences

Nous conservons ici les douze opérateurs utilisés précédemment auquel nous ajouterons l'opérateur 3-opt pour le ATSP que nous avions vu au Chapitre précédent.

L'introduction des quatre nouveaux paramètres de la stratégie CD nous oblige à choisir une méthode de réglage. Nous utiliserons F-Race [Birattari, 2004] que nous avons décrit à la Section 2.2.2. F-Race sera également utilisée pour paramétrer la Poursuite Adaptative (AP) qui nous servira de comparaison. Enfin, nous nous comparerons également à un mécanisme où les probabilités d'application des opérateurs sont fixées dès le début. Nous utiliserons ParamILS pour ce réglage.

6.5.1 Protocole expérimental

Le paramètre pour U_P est $\epsilon = 0.1$ (Section 5.4.2). Le gagnant de la procédure F-Race pour AP est ($\alpha' = 0.9$, $\beta = 0.1$, $p_{min} = 0.01$). Les combinaisons de paramètres testées ont été générées à partir de $\{0.1, 0.3, 0.6, 0.9\}$ pour α' et β et $\{0, 0.001, 0.01, 0.1, 0.1\}$ pour p_{min}.

Les paramètres de CD sont testés parmi un ensemble de valeurs prometteuses présentées au Tableau 6.2a. Ceci nous donne 320 combinaisons possibles. La troisième colonne indique le gagnant de la procédure de *racing*.

La combinaison gagnante étant à une extrémité de chaque domaine, on peut être amené à penser qu'une meilleure combinaison peut être obtenue en élargissant les domaines. Nous avons donc lancé un deuxième réglage avec de nouveaux domaines (Tableau 6.2b) et obtenu un nouveau vainqueur ($T_{max} = 0.35$, $T_{min} = 0.15$, $s_{inc} = 0.0001$, $s_{dec} = 0.01$). Ce dernier est relativement différent du précédent mais n'a pas bénéficié des nouvelles valeurs aux extrémités de chaque domaine.

Lorsque les deux gagnants sont testés côte à côte, leurs distributions de résultats sont statistiquement équivalentes. Ceci nous amène à penser que la stratégie n'est pas extrêmement sensible à sa configuration de paramètres et est donc relativement robuste.

Afin d'effectuer la comparaison avec des probabilités d'application d'opérateurs fixées à l'avance, nous utilisons ParamILS. Une valeur est affectée à chaque opérateur, puis est transformée en probabilité en calculant la proportion de cette valeur par rapport à leur somme totale. Le domaine des valeurs pour les opérateurs extrêmement perturbateurs est $\{0.0, 0.001, 0.01, 0.1, 0.5\}$. Pour les autres opérateurs nous

Paramètre	Premiers domaines	Gagnant 1
T_{max}	$\{0.3, 0.4, 0.5, 0.6\}$	0.3
T_{min}	$\{0.05, 0.1, 0.15, 0.20, 0.25\}$	0.25
s_{inc}	$\{0.0001, 0.001, 0.01, 0.1\}$	0.0001
s_{dec}	$\{0.0001, 0.001, 0.01, 0.1\}$	0.1

(a) Domaines de réglage et paramétrage gagnant initial pour CD.

Paramètre	Seconds domaines	Gagnant 2
T_{max}	$\{0.25, 0.3, 0.35, 0.4\}$	0.35
T_{min}	$\{0.15, 0.2, 0.25, 0.3\}$	0.15
s_{inc}	$\{0.00001, 0.0001, 0.001\}$	0.0001
s_{dec}	$\{0.01, 0.1, 0.2\}$	0.01

(b) Nouveaux domaines de réglage et second paramétrage gagnant pour CD.

TABLEAU 6.2 – Configuration par F-Race pour CD.

utilisons le domaine $\{0.0, 0.1, 0.2, \ldots, 1.0\}$. Les valeurs initiales données à ParamILS sont celles se trouvant au milieu des domaines.

Le réglage s'exécute hors ligne dans un laps de temps de 48 heures pendant lequel ParamILS peut explorer l'espace de recherche constitué de $11^{10} \times 5^2$ configurations pour le QAP et $11^{11} \times 5^2$ configurations pour l'ATSP. Les probabilités obtenues pour le QAP et l'ATSP sont données dans les Tableaux 6.3 et 6.4 respectivement.

Comme nous l'avons observé précédemment, les deux métriques de distances vues au chapitre précédent produisent des résultats relativement similaires. Dans l'intérêt de la clarté, nous n'utiliserons donc ici que la distance basée sur les couples variable-valeur d_1^P.

Notons que la méthode de sélection proportionnelle PR avec la mesure d'utilité de Pareto dominance U_p correspond à la méthode vue au chapitre précédent. Les résultats seront un peu différents dans ce chapitre car les ensembles d'opérateurs ne sont pas les mêmes.

À nouveau, pour chaque instance, chaque algorithme est exécuté 30 fois pour un maximum de 40 000 itérations.

6.5.2 Résultats et discussion

Le Tableau 6.5 présente les résultats concernant les expériences sur des instances de QAP issues de QAPLIB [Burkard *et al.*, 1997]. Les valeurs pour chaque algorithme représentent le pourcentage moyen de différence par rapport aux meilleures valeurs connues.

Considérons tout d'abord la qualité générale moyenne des solutions obtenues avec toutes les méthodes. Toutes, même la sélection uniforme, sont généralement à un ou deux pour cents de l'optimale ou de la meilleure valeur connue. Ceci démontre que le nombre alloué d'itérations n'est pas un facteur limitant.

Le Tableau 6.5 nous permet d'observer que U_P (un paramètre) avec une sélection dont la probabilité est directement calculée à partir de l'utilité (PR dans le tableau) se comporte très bien par rapport à AP qui a trois paramètres. Pour AP, le réglage des paramètres ne lui permet que de se comporter à un niveau équivalent ou pire que PR. Le concept du « gagnant remporte tout », utile pour les algorithmes évolutionnaires, ne semble pas apporter de bénéfice dans les conditions de test utilisées

Opérateur	Valeur initiale	Valeur sélectionnée	Probabilité
O1	0.5	0.9	0.20
O2	0.5	0.3	0.07
O3	0.5	0.4	0.09
O4	0.5	0.4	0.09
O5	0.5	0	0.00
O6	0.5	0.6	0.13
O7	0.5	0.8	0.17
O8	0.5	0.4	0.09
O9	0.5	0.8	0.17
O10	0.5	0	0.00
O11	0.01	0.01	0.0022
O12	0.01	0.001	0.0002

TABLEAU 6.3 – Probabilités d'application des opérateurs pour le QAP données par le réglage ParamILS-. Le domaine pour les opérateurs O1–O10 est $\{0.0, 0.1, 0.2, \ldots, 1.0\}$ et le domaine pour les opérateurs O11 et O12 est $\{0.0, 0.001, 0.01, 0.1, 0.5\}$. Dans ce tableau les probabilités sont arrondies à deux chiffres après la décimale pour O1–O10 et à 4 chiffres pour O11 et O12.

Opérateur	Valeur initiale	Valeur sélectionnée	Probabilité
O1	0.5	0.6	0.13
O2	0.5	0.3	0.06
O3	0.5	0.9	0.19
O4	0.5	0.1	0.02
O5	0.5	0.7	0.15
O6	0.5	0.1	0.02
O7	0.5	0.6	0.13
O8	0.5	0.5	0.11
O9	0.5	0.1	0.02
O10	0.5	0.1	0.02
O11	0.01	0.1	0.02
O12	0.01	0.001	0.0002
OA	0.5	0.6	0.13

TABLEAU 6.4 – Probabilités d'application des opérateurs pour l'ATSP données par le réglage ParamILS-. Le domaine pour les opérateurs O1-O10 et OA est $\{0.0, 0.1, 0.2, \ldots, 1.0\}$ et le domaine pour les opérateurs O11 et O12 est $\{0.0, 0.001, 0.01, 0.1, 0.5\}$. Dans ce tableau les probabilités sont arrondies à deux chiffres après la décimale pour O1–O10 et OA, et à 4 chiffres pour O11 et O12.

Instance	BKV	Uniforme	Fixe	U_P PR	U_P AP	$U_\alpha^{\Sigma+}$ α 0.2 / 0.5 / 0.8	$U_\alpha^{\Sigma+}$ CD
bur26a	5426670	0.03	**0.00**	**0.00**	0.02	0.02 / 0.01 / **0.00**	0.01
chr25a	3796	20.20	11.96	**10.70**	13.82	13.8 / 14.9 / 12.1	12.50
kra30a	88900	2.49	0.79	0.79	1.54	1.57 / 1.63 / 0.89	**0.61**
kra30b	91420	1.11	0.18	0.21	0.30	0.16 / 0.45 / 0.32	**0.13**
nug20	2570	0.12	**0.00**	0.01	0.01	**0.00** / 0.03 / **0.00**	0.01
nug30	6124	1.24	0.14	0.20	0.32	0.31 / 0.19 / 0.39	**0.11**
sko42	15812	2.28	0.23	0.29	0.38	0.19 / 0.28 / 0.67	**0.16**
sko49	23386	2.48	0.29	0.36	0.43	**0.21** / 0.27 / 0.81	0.24
tai30a	1818146	2.59	1.25	1.26	1.50	1.17 / 1.27 / 1.68	**0.91**
tai50a	4941410	4.20	1.91	2.16	2.13	**1.58** / 1.59 / 2.83	1.66
tai30b	637117113	0.43	**0.12**	0.13	0.25	0.44 / 0.35 / 0.16	0.15
tai50b	458821517	2.36	0.26	0.25	0.36	0.30 / 0.39 / 0.37	**0.18**
wil50	48816	1.24	0.10	0.16	0.15	0.09 / 0.11 / 0.28	**0.08**

TABLEAU 6.5 – Résultats pour le QAP – le pourcentage moyen de différence par rapport aux meilleures valeurs connues.

ici. Au lieu de donner l'avantage au meilleur opérateur, il semble qu'il soit plus approprié d'avoir une probabilité plus élevée de choisir parmi quelques opérateurs qui se comportent raisonnablement bien.

Les résultats pour la sélection avec les probabilités d'application d'opérateurs fixées à l'avance sont meilleurs que la sélection uniforme. Toutefois ils ne parviennent pas réellement à se différencier des résultats obtenus par une simple sélection proportionnelle.

Les résultats pour la stratégie CD sont présentés à la colonne $U_\alpha^{\Sigma+}CD$. Il semble clair que CD est meilleure que les autres méthodes, ou du moins à 0.05 % du meilleur résultat, sur la majorité des instances. Cette supériorité est confirmée par un test de rangs signés de Wilcoxon avec un seuil de confiance de 95 %. Si nous comparons $U_\alpha^{\Sigma+}CD$ aux meilleures valeurs sur les différents α pour $U_\alpha^{\Sigma+}$, les deux distributions sont statistiquement équivalentes. Ceci nous permet de conclure que la stratégie CD est suffisamment performante pour produire des résultats équivalents aux meilleurs résultats de $U_\alpha^{\Sigma+}$ avec un α préréglé.

Les résultats pour l'ATSP sont donnés dans le Tableau 6.6. Les colonnes 5, 7 et 9 utilisent les paramètres choisis pour le QAP. Les colonnes 6, 8 et 10 (astérisques dans les entêtes des colonnes) utilisent des paramètres réglés pour l'ATSP : $\epsilon = 0.01$ pour U_P avec PR, ($\alpha = 0.3$, $\beta = 0.1$, $p_{min} = 0.001$) pour AP, et ($T_{max} = 0.6$, $T_{min} = 0.25$, $s_{inc} = 0.0001$, $s_{dec} = 0.01$) pour CD.

Regardons d'abord les résultats pour les probabilités d'application fixées à l'avance. À nouveau ces résultats sont meilleurs que la sélection uniforme. Si l'on se réfère au Tableau 6.4, on peut constater que la probabilité de sélection fixée pour l'opérateur

Instances	BKV	Uniforme	Fixe	U_p PR	U_p PR*	U_p AP	U_p AP*	$U_\alpha^{\Sigma+}$ CD	$U_\alpha^{\Sigma+}$ CD*
ry48p	14422	14.0	3.3	0.7	1.5	2.4	1.9	**0.6**	0.7
ft53	6905	32.5	8.2	1.1	2.0	1.1	2.5	0.9	**0.7**
ft70	38673	10.8	5.8	**0.5**	0.6	1.3	0.7	0.8	0.7
ftv38	1530	12.9	3.4	1.0	2.1	2.5	2.5	**0.8**	0.9
ftv44	1613	23.8	5.5	1.2	2.3	2.5	2.6	**0.9**	1.0
ftv47	1776	27.5	6.0	0.7	1.5	1.7	1.1	**0.5**	**0.5**
ftv55	1608	22.5	8.5	1.7	2.8	3.5	3.2	**1.2**	1.3
ftv64	1839	32.5	15.2	2.0	3.5	4.4	3.1	1.7	**1.6**
ftv70	1950	26.9	17.8	2.8	3.9	5.1	3.5	2.2	**2.0**
ftv90	1579	30.8	23.0	4.8	6.0	10.7	4.9	**4.0**	4.4
ftv100	1788	32.5	32.6	5.4	4.8	14.2	5.0	**4.4**	5.3
ftv120	2166	38.4	26.9	7.5	6.8	21.7	**6.7**	8.6	8.5
ftv140	2420	43.0	36.1	10.7	**8.3**	29.6	8.4	11.1	12.6
ftv160	2683	42.4	40.9	14.6	8.3	37.7	**8.2**	15.1	16.6
kro124p	36230	28.0	19.8	4.0	4.5	8.7	3.6	**3.4**	**3.4**
rbg323	1326	31.7	20.9	3.8	0.1	12.3	**0.0**	3.3	3.0
rbg403	2465	36.2	14.2	**0.0**	**0.0**	3.2	**0.0**	0.3	0.3
rbg443	2720	39.9	16.9	0.6	**0.0**	4.5	**0.0**	1.0	0.6

TABLEAU 6.6 – Résultats pour l'ATSP – pourcentage moyen de différence par rapport aux meilleures valeurs connues.

OA, spécifique à l'ATSP, même si elle est parmi les plus élevées, n'est pas celle qui est privilégiée.

Si l'on considère les résultats pour le paramétrage spécifique au QAP appliqué à l'ATSP, les différentes méthodes parviennent à gérer la population d'opérateurs très défavorable. Pour preuve, les résultats pour la sélection uniforme sont très mauvais et rappelons-nous qu'il n'y a qu'un seul opérateur qui soit un bon opérateur pour l'ATSP. D'autre part, AP est peu performante sur les plus grandes instances. Bien entendu, de meilleurs résultats bruts auraient été obtenus si la population d'opérateurs avait été pensée pour l'ATSP. Comme avec le QAP, la stratégie CD produit de bons résultats mais ici elle ne parvient à dépasser les autres méthodes que sur les plus petites instances.

Considérons maintenant les résultats pour le paramétrage spécifique à l'ATSP. Par rapport à PR, PR* produit une amélioration sur les plus grandes instances contrebalancée par des résultats de qualité inférieure sur les autres instances. De fait, PR* n'est pas une amélioration en général mais sert plutôt à souligner l'importance du paramètre ϵ. Une valeur plus petite signifie une faible probabilité de sélection des opérateurs très perturbateurs et donc une meilleure performance sur les plus grandes instances. Pour U_P avec AP, AP* donne une amélioration significative. Cette méthode de sélection est très sensible à la configuration de ses paramètres et à la population d'opérateurs. En effet, ici il y a un seul opérateur qui puisse bénéficier de la stratégie du gagnant remporte tout. Néanmoins, les résultats sont équivalents à ceux de la sélection PR* qui est bien plus simple.

Les nouveaux paramètres pour CD ne semblent pas apporter de réel changement positif ou négatif. La faible performance de CD sur les plus grandes instances provient du fait qu'à partir du moment où un fort compromis vers l'exploration apparaît, ce compromis ne diminue pas suffisamment rapidement, ce qui implique trop

d'applications des opérateurs perturbateurs.

La stratégie CD est basée sur plusieurs paramètres, toutefois les résultats pour CD et CD* montrent que l'algorithme n'est pas hypersensible à leurs valeurs. CD est robuste. Bien que les paramètres demandent un réglage, les résultats tendent à montrer que ce réglage n'est pas forcément à refaire lorsqu'un nouveau problème se présente.

Comparaison par rapport au chapitre précédent

Si l'on compare les résultats obtenus dans ce chapitre avec ceux du chapitre précédent, on remarque que la meilleure méthode, CD, avec douze opérateurs dont deux très perturbateurs, obtient des résultats similaires à ceux de la sélection uniforme avec dix opérateurs sur le QAP. Pour l'ATSP, CD avec treize opérateurs retrouve les performances de la sélection uniforme sur onze opérateurs seulement sur les instances de taille plus petite.

Comme nous l'avions déjà remarqué au chapitre précédent, la nature de l'ensemble d'opérateurs est importante. Ceci est d'autant plus flagrant dans le cas de l'ATSP où un seul des opérateurs, le 3-opt, est utilisé dans la littérature pour résoudre ce problème. Lorsque l'on compare les résultats des sélections uniformes, on remarque une perte drastique en qualité avec les deux opérateurs perturbateurs. Toutefois le gain obtenu avec la sélection proportionnelle ou la stratégie CD est aussi beaucoup plus grand. L'objectif de l'ajout d'opérateurs perturbateurs était de compliquer la tâche des méthodes de sélection.

Ce constat sur l'influence de l'ensemble des opérateurs indique donc qu'il serait judicieux de générer automatiquement les opérateurs à partir du modèle. Ceci permettrait d'obtenir des opérateurs vraisemblablement bons, dont une stratégie de contrôle simple pourrait optimiser l'application.

6.6 Conclusion

Dans ce chapitre nous nous sommes intéressés à faire évoluer dynamiquement le compromis qualité et diversité au fil de la recherche.

Nous avons d'abord présenté une nouvelle façon d'appréhender ce compromis à travers une mesure d'utilité tenant compte des opérateurs les uns par rapport aux autres. Nous avons testé plusieurs variantes pour en trouver une qui fonctionne de façon satisfaisante.

Une stratégie de variation du compromis, la stratégie de diversité correcte CD, a été empruntée au domaine évolutionnaire et adaptée au contexte de la recherche locale.

Cette stratégie a été comparée à la sélection proportionnelle du chapitre précédent et l'approche appelée poursuite adaptative qui cherche à augmenter fortement les chances de l'opérateur ayant la meilleure utilité. Nous avons également utilisé la méthode de réglage ParamILS afin de déterminer des probabilités statiques de sélections des opérateurs. C'est la stratégie CD qui se révèle la meilleure dans la majorité des cas.

Nous avons commenté le fait que les résultats obtenus ici semblent moins bons que ceux du chapitre précédent. En effet nous avons introduit deux opérateurs perturbateurs qui rendent le contrôle beaucoup plus difficile mais qui permettent en même temps de mieux en apprécier le bénéfice. Cela démontre par la même occasion l'importance de l'ensemble d'opérateurs et pointe vers la génération automatique d'opérateurs à partir du modèle.

Chapitre 7

Résolution de CSP

Sommaire

7.1 Introduction 102
7.2 Contraintes 102
7.3 Application 104
7.3.1 Modèle PPP 104
7.3.2 Recherche tabou en Comet 105
7.3.3 Opérateurs de CSP 107
7.3.4 Protocole expérimental 108
7.3.5 Résultats et discussion pour la sélection proportionnelle . . 109
7.3.6 Résultats et discussion pour les stratégies adaptatives . . . 114
7.4 Conclusion 114

7.1 Introduction

Nous avons évoqué au Chapitre 3 différentes méthodes proposées pour la gestion de contraintes pour des algorithmes de recherche locale. À l'instar des méthodes utilisées pour les problèmes d'optimisation, celles-ci demandent à l'utilisateur de les paramétrer. De surcroît, il appartient également à l'utilisateur d'identifier les contraintes critiques dans sa modélisation du problème. Ceci est en général réalisé par l'intermédiaire d'une pondération appliquée aux fonctions de violation des contraintes (Section 3.2) afin de spécifier l'importance d'une contrainte ou d'un groupe de contraintes.

Dans ce chapitre nous souhaitons étudier l'opportunité d'utiliser la sélection autonome d'opérateurs afin de faire les bons choix lorsqu'il s'agit de résoudre des problèmes de satisfaction de contraintes. Nous étudierons l'effet de l'utilisation de la pondération et nous essaierons de voir si l'utilisation de plusieurs opérateurs peut contre balancer l'absence de pondération et simplifier la tâche de paramétrage de l'utilisateur.

Bien entendu, la programmation par contraintes (PPC) est le paradigme idoine pour résoudre les problèmes de satisfaction de contraintes à travers des solveurs complets. La PPC peut être hybridée avec d'autres méthodes, notamment la recherche locale, afin d'en améliorer les performances. Ceci étant dit, nous nous restreindrons ici à une approche de recherche locale utilisant des éléments de PPC telles que les variables incrémentales. Ces mécanismes sont rendus facilement utilisables grâce au langage de programmation que nous utilisons, Comet. En effet Comet permet de manipuler les contraintes et leurs violations de façon relativement transparente puisque tous les mécanismes de la PPC, comme la propagation, sont cachés. Un utilisateur demandant plus de contrôle sur son programme a toutefois toujours la possibilité de modifier ces processus internes.

Au delà de la PPC, la gestion des contraintes a également été traitée dans le cadre des algorithmes évolutionnaires [Coello Coello, 2002].

Ce chapitre a trois objectifs principaux :

- intégrer la possibilité de résoudre des problèmes de satisfaction de contraintes à notre méthode de contrôle ;
- prendre en compte des fonctions objectifs plus complexes, en l'occurrence les fonctions de violation, en simplifiant la tâche de l'utilisateur en agrégeant simplement ces fonctions ;
- essayer de concurrencer les algorithmes existants de recherche locale.

La Figure 7.1 présente le schéma général des différents composants qui rentrent en jeu lorsqu'un mécanisme de contrôle autonome est utilisé pour résoudre un problème de satisfaction de contraintes. Comme pour les problèmes d'optimisation le modèle influence les voisinages et les sélecteurs disponibles qui seront combinés pour créer des opérateurs. Les contraintes du modèle induisent les fonctions de violations et des méthodes de propagation plus ou moins spécifiques qui seront utilisées au sein du moteur de résolution. Le contrôle est l'agent extérieur qui dirige le moteur de résolution à travers l'application des opérateurs.

Nous nous intéresserons d'abord aux contraintes dans le contexte de la recherche locale. Nous proposerons par la suite notre méthode de résolution, avec des opérateurs adaptés à la PPC, que nous testerons sur le Progressive Party Problem.

7.2 Contraintes

Dans les problèmes d'optimisation que nous avons évoqués dans les chapitres précédents, le but était de trouver la meilleure solution ou, du moins, une solution

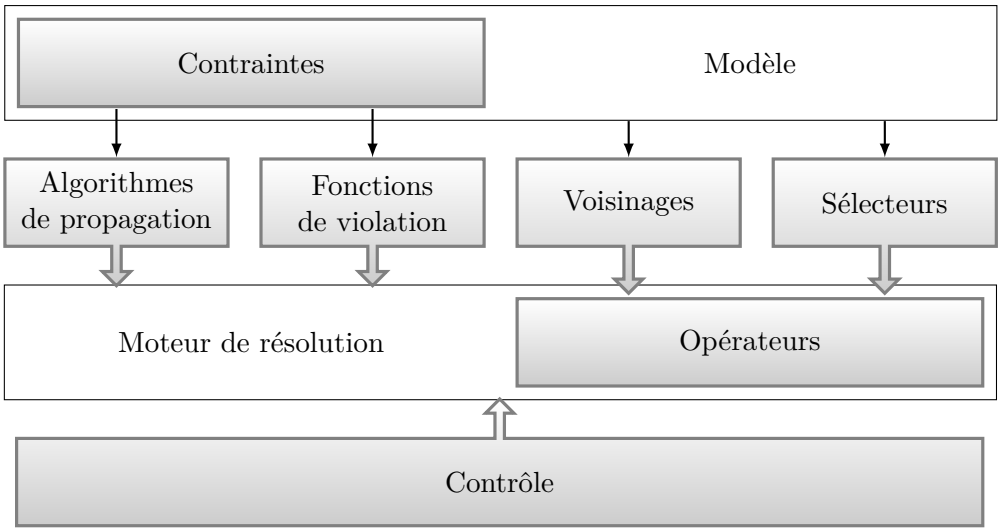


FIGURE 7.1 – Schéma synthétique d'un système pour la résolution de problèmes de satisfaction de contraintes avec la recherche autonome.

se rapprochant de la meilleure solution. Dans le cas de problèmes de satisfaction de contraintes, l'objectif est de trouver une solution qui satisfasse toutes les contraintes du problème. Il existe également le cas où certaines contraintes ne doivent absolument pas être violées – elles sont dites dures – et d'autres contraintes doivent être respectées dans la mesure du possible – elles sont dites faibles. Nous n'aborderons pas ici ce second cas.

Afin de gérer les contraintes dans le contexte de la recherche locale, il est nécessaire de définir des fonctions de violation pour chacune des contraintes du problème. Ces fonctions de violations expriment le degré de violations d'une contrainte. L'association de toutes les fonctions de violations correspondant aux contraintes du problème permet d'obtenir une fonction pouvant servir de fonction objectif pour le processus de recherche locale. On se ramène alors à un problème d'optimisation où l'on cherche à minimiser la fonction objectif. Définir la fonction de violation associée à une contrainte n'est donc pas un choix trivial. [1]

Dans sa forme la plus basique, la fonction de violation d'une contrainte peut tout simplement être la valeur de vérité de la contrainte, c'est-à-dire *vrai* ou *faux*, à laquelle on fera correspondre les valeurs 0 et 1 respectivement. En général il est plus intéressant de considérer des fonctions de violations plus « fines », notamment en considérant les variables de la contrainte ainsi que sa sémantique. Par exemple, pour la contrainte globale `alldifferent`, qui est vraie si toutes ses variables sont différentes, une fonction de violation peut être le nombre d'égalités lorsque l'on considère les variables deux à deux ou encore le nombre valeurs qui ont des doublons.

L'intérêt principal de fonctions de violations plus précises vient du fait qu'elles permettent de mieux diriger la recherche. Prenons le cas le plus extrême où la fonction de violation englobant toutes les contraintes du problème renvoie systématiquement 1 qu'il y ait une ou plusieurs contraintes violées. Le paysage de recherche ressemblera alors vraisemblablement à un plateau immense avec de petites abysses ici ou là lorsqu'une solution est réalisable. Cette topographie est extrêmement nuisible à une procédure de recherche locale puisqu'aucun gradient ne permet de guider la recherche vers un optimum local.

Lorsque l'on considère un système de contraintes $C = (c_1, c_2, \ldots, c_n)$, les fonctions de violations de chacune des contraintes du système doivent être combinées

1. Notons que sans fonction de violation, nous nous retrouverions face à des contraintes dûres. Le paysage de recherche correspondant ne contiendrait que les solutions réalisables.

afin de créer la fonction de violation du système. En général, cette combinaison pourra être la somme, voire la valeur maximale. Par ailleurs, dans cette agrégation, on peut imaginer que les contraintes n'aient pas toutes la même importance, ce qui introduit naturellement une pondération. Ainsi la fonction de violation d'un système de contraintes sera de la forme

$$V_s(C) = \sum_{i \in \{1,\ldots,n\}} \alpha_i v_s(c_i) \tag{7.1}$$

où α_i est une pondération et $v_s(c_i)$ est la fonction de violation de c_i pour la solution s.

7.3 Application

Nous nous intéressons ici à l'application des méthodes présentées dans les chapitres précédents au Progressive Party Problem (PPP). L'intérêt du PPP est qu'il comporte différents types de contraintes globales et donc des fonctions de violation différentes. Ces fonctions doivent être aggrégées en une seule fonction et il existe des pondérations sur les fonctions de violations qui fonctionnent mieux que d'autres. Ceci nous donne l'occasion de tester le bénéfice apporté par notre méthode avec et sans pondération.

Le PPP est également un problème que l'on retrouve régulièrement dans les publications présentant des systèmes de recherche locale avec des contraintes [Galinier et Hao, 2004; Van Hentenryck et Michel, 2005; Ågren *et al.*, 2009].

Ce problème est présenté à la Section 4.4. À notre connaissance, la meilleure méthode pour résoudre le PPP provient de [Van Hentenryck et Michel, 2005]. C'est une recherche tabou. Elle est par ailleurs utilisée pour démontrer la puissance du langage Comet. Elle se prête donc bien à une comparaison avec nos approches également écrites en Comet.

7.3.1 Modèle PPP

Toutes les méthodes testées utiliseront la même modélisation (Figure 7.2) modulo la pondération des contraintes que nous utiliserons ou non selon le contexte (lignes 12 et 14).

Les variables de décisions sont déclarées à la ligne 9 et `boat[g,p]` représente le bateau dans lequel se trouve l'invité `g` pendant la période `p`. Les contraintes `alldifferent` de la ligne 12 indiquent qu'un invité ne peut pas se rendre plus d'une fois sur le même bateau. Les contraintes `knapsack` de la ligne 14 spécifient les contraintes de capacité. À la ligne 16, les contraintes `atmost` spécifient que deux invités ne peuvent se rencontrer plus d'une fois.

Notons que les pondérations exprimées aux lignes 12 et 14 apparaissent pour la première fois dans [Galinier et Hao, 2004]. Les auteurs indiquent avoir testé plusieurs valeurs. Ces pondérations ont par la suite été reprises dans le travaux de Van Hentenryck et Michel, notamment [Van Hentenryck et Michel, 2005].

Comme nous reprenons les fonctions de violations par défaut de Comet (Sec-

```
1  //inputs:
2  // range Guests;
3  // range Periods;
4  // range Hosts;
5  // int cap[Hosts];
6  // int crew[Guests];

7  Solver<LS> m();
8  UniformDistribution distr(Hosts);
9  var{int} boat[Guests,Periods](m,Hosts) := distr.get();
10 ConstraintSystem<LS> S(m);

11  forall (g in Guests)
12     S.post(2 * alldifferent(all(p in Periods) boat[g,p]));
13  forall (p in Periods)
14     S.post(2 * knapsack(all(g in Guests) boat[g,p],crew,cap));
15  forall (i in Guests,j in Guests : j > i)
16     S.post(atmost(1,all(p in Periods) boat[i,p] == boat[j,p]));
17 m.close();
```

FIGURE 7.2 – Modélisation en COMET.

tion 3.3.3), résoudre le problème revient alors a

$$\begin{aligned}\min \quad & 2 \sum_{\substack{g \in \texttt{Guests} \\ h \in \texttt{Hosts}}} \max(\#(h, \{\texttt{boat}[g,p] | p \in \texttt{Periods}\}) - 1, 0) \\ & + 2 \sum_{\substack{p \in \texttt{Periods} \\ h \in \texttt{Hosts}}} \max\left(\sum_{i \in \rho(h, \{\texttt{boat}[g,p] | g \in \texttt{Guests}\})} \texttt{crew}[i] - \texttt{cap}[h], 0 \right) \\ & + \sum_{\substack{i,j \in Guests \\ j > i}} \max(\#(v, \{\texttt{boat}[i,p] = \texttt{boat}[j,p] | p \in \texttt{Periods}\}) - 1, 0)\end{aligned} \tag{7.2}$$

où $\#(v, a)$ est le nombre d'occurrences de v dans a et $\rho(v, a)$ est l'ensemble des indices de a dont les valeurs sont égales à v.

7.3.2 Recherche tabou en Comet

L'algorithme avec lequel nous nous comparerons est donné dans la Figure 7.3. Il s'agit d'une recherche tabou qui utilise certaines notions de généricité. Pour résoudre le modèle présenté plus haut il suffit d'un appel à `search(m,S,Hosts)`. À l'interieur de la fonction, au lieu d'utiliser la modélisation avec le tableau `boat`, les variables des contraintes sont récupérées directement (ligne 2). Cette même notion d'abstraction des structures est employée dans notre solveur.

L'algorithme se base sur une heuristique de conflit minimum qui sélectionne la variable de décision ayant le plus de violations (ligne 15) et qui cherche la valeur minimisant ses violations (ligne 17). La longueur de la liste tabou est variable et évolue en fonction des changements de violations d'une itération à l'autre. Les couples variable-valeur sont tabous pendant un certain nombre d'itérations (ligne 19). Si les nouvelles violations sont inférieures à celles de l'itération précédente, ce nombre d'itérations est décrémenté (ligne 21). À l'inverse il est incrémenté s'il n'y a pas eu

```
1  function void search(Solver<LS> m, ConstraintSystem<LS> S, range D) {
2      var{int}[] x = S.getVariables();
3      range X = x.getRange();
4      UniformDistribution distr(D);
5      int it = 1;
6      int tabu[X,D] = -1;
7      int tbl = 2; int tblMin = 2; int tblMax = 10;
8      var{int} violations = S.violations();
9      int best = violations;
10     Solution solution(m);
11     int stable = 0; int stableLimit = 2000;
12     int restartFreq = 100000;
13     while (violations > 0 && it <= 1000000) {
14         int old = violations;
15         selectMax (i in X) (S.violations(x[i])) {
16             int gap = best - violations;
17             selectMin (j in D, d = S.getAssignDelta(x[i],j):{
18                        tabu[i,j] <= it || d < gap) (d) {
19                 tabu[i,x[i]] = it + tbl;
20                 x[i] := j;
21                 if (violations < old && tbl > tblMin) tbl--;
22                 if (violations >= old && tbl < tblMax) tbl++;
23             }
24             if (violations < best) {
25                 best = violations;
26                 solution = new Solution(m);
27                 stable = 0;
28             } else {
29                 if (stable == stableLimit) {
30                     solution.restore();
31                     stable = 0;
32                 } else stable++;
33                 if (it%restartFreq == 0) {
34                     with delay(m) {
35                      forall (i in X)
36                         x[i] := distr.get();
37                     }
38                     best = violations;
39                     solution = new Solution(m);
40                 }
41             }
42             it++;
43         }
44     }
45 }
```

FIGURE 7.3 – Procédure de recherche tabou en COMET avec liste tabou dynamique, intensification et redémarrage.

d'amélioration (ligne 22). Un critère classique d'aspiration est utilisé : un mouvement tabou est accepté s'il produit une nouvelle meilleure solution.

Cette recherche tabou est améliorée par un processus d'intensification qui retourne à la meilleure solution trouvée jusqu'alors si aucune amélioration ne s'est produite pendant un certain nombre d'itérations (ligne 29). Ceci permet d'explorer des parties intéressantes de l'espace de recherche plus exhaustivement et d'éviter de tourner en rond ou simplement de ne faire aucun progrès dans des portions de l'espace de recherche moins intéressantes. L'algorithme intègre également un composant de redémarrage. Si aucune solution réalisable (aucune violation) n'a été trouvée pendant un certain nombre d'itérations, la recherche repartira d'une nouvelle solution aléatoire, en espérant ainsi se libérer de la région de l'espace de recherche dans laquelle elle était prisonnière (ligne 33).

Au final, cet algorithme est plus qu'une simple recherche tabou puisqu'il intègre différents mécanismes pour améliorer la recherche tabou classique. Par ailleurs, il repose sur plusieurs paramètres : pour la longueur minimum et maximum du tabou, pour le déclenchement de l'intensification et du redémarrage. Enfin on peut aussi considérer la pondération des contraintes du modèle comme faisant partie de la paramétrisation.

7.3.3 Opérateurs de CSP

Contrairement aux problèmes précédemment traités, ici il ne s'agit pas de variables binaires ni de problèmes de permutations mais d'un problème aux variables entières, chacune dans un domaine précis. Nous changeons donc de population d'opérateurs.

Nous introduisons notamment une fonction de sélection sur les variables permettant de les sélectionner en fonction du nombre de contraintes violées dont elles font partie. Nous ajoutons également la possibilité de rendre tabou certains mouvements.

Voisinage

Définition 7.3.1. Soient D un domaine discret fini, $x = (x_1, \ldots, x_n) \in X$ une solution du problème avec $x_i \in D$ alors $\mathcal{N}_D$ est la relation de voisinage où chaque voisin diffère de l'autre dans la valeur d'une de ses variables x_i : $(x, x') \in \mathcal{N}_D$ ssi $\exists! i \in \{1, \ldots, n\}$ tel que $x_i \neq x'_i$.

Définition 7.3.2. Soit f_k^{mv} la fonction qui retourne les k indices des k variables subissant le plus de violations. Nous pouvons alors créer la relation de voisinage $\mathcal{N}_D^{f_k^{mv}}$ qui restreint $\mathcal{N}_D$ aux voisins ayant les plus fortes violations.

Sélecteur

Nous reprenons le sélecteur de la k-meilleure amélioration : σ_{BIk} tel que $\sigma(s, \mathcal{N})$ est un élément uniformément sélectionné $s' \in K$, K étant l'ensemble des k-meilleurs éléments selon l'ordre $<$, tel que $(s, s') \in \mathcal{N}$. Nous avons choisi de reprendre ce sélecteur car il allie l'exploitation (sélection parmi les meilleurs) à plus ou moins d'exploration (la sélection se fait aléatoirement parmi k valeurs).

Le sélecteur σ_{BIk} est modifié en introduisant une fonctionnalité tabou. Ainsi σ_{BIk}^{Tl} sera le sélecteur σ_{BIk} associé à une liste tabou. Une solution est tabou si elle contient une affection interdite dans la liste tabou. Le nombre d'itérations pendant lequel une affection est interdite est donné par l. Si des opérateurs utilisent la fonctionnalité tabou, ils partageront la même liste.

Opérateur

Nous souhaitons avoir un ensemble d'opérateurs corrects pour le problèmes : certains pourront être utilisés pour améliorer une solution alors que d'autres doivent pouvoir apporter de la diversité. Avec le voisinage et le sélecteur proposés, il est possible d'influencer ces deux aspects en utilisant différentes valeurs de k et de l.

En combinant la relation de voisinage $\mathcal{N}_D^{f_k^{mv}}$ pour les valeurs $k \in \{1, 5\}$, et le sélecteur σ_{BIk}^{Tl} pour les valeurs $k \in \{1, 5\}$ et $l \in \{7, 8, 9\}$ nous générons combinatoirement douze opérateurs. Les nombres l d'itérations tabou sont choisis pour cadrer avec ceux de la recherche tabou COMET avec laquelle nous nous comparerons (variant de 2 à 10).

Notons que nous avons initialement essayé, puisque notre mise en œuvre nous le permet, de générer des opérateurs qui ne considèrent que les variables utilisées dans une seule contrainte. Ceci nous fait donc au moins un opérateur par contrainte (soit environ 440), et combinatoirement beaucoup plus si l'on considère les autres choix dans la génération des opérateurs, par exemple différents sélecteurs.

L'approche « une contrainte – un opérateur » n'a pas fonctionné car le processus de recherche ne parvenait pas à converger. Nous présenterons donc ici des opérateurs qui considèrent toutes les variables du problème globalement.

7.3.4 Protocole expérimental

Dans un premier temps nous comparerons la sélection uniforme (Uni), la sélection proportionnelle (PR) associée à l'utilité Pareto U_p et la recherche tabou (TS). Nous modifions chaque algorithme pour pouvoir analyser l'importance de la pondération des fonctions de violations (dans la modélisation) ainsi que l'impact du composant d'intensification. À cette fin, nous intégrons cette fonctionalité dans notre mécanisme de contrôle.

Sélection proportionnelle. Pour PR U_p nous fixons $\epsilon = 0.1$ (Section 5.4.2). Au vu des expériences dans les chapitres précédents, la seule autre valeur à avoir été testée est $\epsilon = 1$ mais elle donnait de moins bon résultats.

Recherche tabou. Les paramètres de TS sont ceux proposés originellement : nombre d'itérations taboues compris entre 2 et 10 inclus, retour à la meilleure solution si aucune amélioration pendant 2 000 itérations (intensification) et redémarrage si aucune amélioration pendant 100 000.

Intensification. Lorsque PR U_p est appareillée du composant d'intensification, le même nombre d'itérations est conservé (2 000).

Poursuite adaptative. Une fois ces expériences initiales effectuées, nous ajouterons les comparaisons avec la poursuite adaptative (AP) associée à l'utilité Pareto U_p et avec la stratégie dynamique de diversité correcte (CD) associée à l'utilité de la somme des déplacements positifs pondéres. À nouveau, AP et CD sont réglés à l'aide de F-Race.

Le gagnant de la procédure F-Race pour AP est ($\alpha' = 0.9$, $\beta = 0.9$, $p_{min} = 0.01$). Les combinaisons de paramètres testées ont été générées à partir de $\{0.1, 0.3, 0.6, 0.9\}$ pour α' et β et $\{0, 0.001, 0.01, 0.1, 0.1\}$ pour p_{min}.

Stratégie adaptative. Les paramètres de CD sont testés parmi un ensemble de valeurs prometteuses présentées au Tableau 7.1. Ceci nous donne 144 combinaisons possibles. La troisième colonne indique le gagnant de la procédure de *racing*.

Paramètre	Domaine	Gagnant
T_{max}	$\{0.2, 0.3, 0.4, 0.5\}$	0.2
T_{min}	$\{0.15, 0.2, 0.25, 0.3\}$	0.25
s_{inc}	$\{0.0001, 0.001, 0.01\}$	0.0001
s_{dec}	$\{0.01, 0.1, 0.2\}$	0.1

TABLEAU 7.1 – Configuration par F-Race pour CD pour le PPP.

	Pourcentage de résolution											
	Sans pondération						Avec pondération					
	Sans intens.			Avec intens.			Sans intens.			Avec intens.		
Config	Uni	PR	TS	Uni	PR	TS	Uni	PR	TS	Uni	PR	TS
1-6	100	100	100	100	100	100	100	100	100	100	100	100
1-7	100	100	100	100	100	100	100	100	100	100	100	100
1-8	100	100	100	100	100	100	100	100	100	100	100	100
1-9	100	100	100	100	100	100	100	100	100	100	100	100
1-10	12	**95**	93	**100**	99	**100**	69	**100**	97	100	100	100
2-6	100	100	100	100	100	100	100	100	100	100	100	100
2-7	100	100	100	100	100	100	100	100	100	100	100	100
2-8	100	100	100	100	100	100	100	100	100	100	100	100
2-9	0	**12**	0	45	**85**	51	2	**18**	3	89	97	**99**
3-6	100	100	100	100	100	100	100	100	100	100	100	100
3-7	100	100	100	100	100	100	100	100	100	100	100	100
3-8	100	100	100	100	100	100	100	100	100	100	100	100
3-9	0	**4**	0	49	**72**	36	0	**14**	2	88	98	**100**
4-6	100	100	100	100	100	100	100	100	100	100	100	100
4-7	100	100	100	100	100	100	100	100	100	100	100	100
4-8	84	**100**	68	100	100	100	100	100	100	100	100	100
4-9	0	**1**	0	15	**48**	10	0	**7**	0	76	**96**	93
5-6	100	100	100	100	100	100	100	100	100	100	100	100
5-7	48	**100**	1	74	**99**	65	**100**	**100**	22	98	**100**	**100**
6-6	**100**	**100**	99	100	100	100	100	100	100	100	100	100
6-7	4	**63**	0	43	**78**	7	72	**100**	0	86	**96**	90

TABLEAU 7.2 – Pourcentage de configurations résolues en fonction de l'utilisation de contraintes pondérées et du composant d'intensification. Les meilleures valeures pour chaque groupe sont indiquées en gras.

Notons que comme un grand nombre d'instances sont facilement résolues, le critère d'optimisation pour F-Race est le temps d'exécution et non le taux de résolution.

Dans tous les cas, le nombre maximum d'itérations par exécution est fixé à 1 000 000 et chaque algorithme est exécuté 100 fois. Toutes les fenêtres glissantes sont de longueur 100.

7.3.5 Résultats et discussion pour la sélection proportionnelle

Le Tableau 7.2 présente le pourcentage de configurations résolues en fonction de l'utilisation de contraintes pondérées et du composant d'intensification. Le Tableau 7.3 présente le nombre moyen d'itérations correspondant.

Taux de résolution

Comme le montre le Tableau 7.2 la majorité des configurations est facilement résolue avec 100 % de succès dans le nombre d'itérations imparti, même avec la sélection uniforme. Le nombre d'itérations nécessaires pour arriver à ces résultats

nous permettra de mieux en appréhender les différences. Les configurations 1-10, 2-9, 3-9, 4-9, 5-7 et 6-7 sont les plus difficiles.

Sans pondération des contraintes. Considérons d'abord le cas où le modèle n'utilise pas de pondération afin d'accentuer l'importance de certaines contraintes. Sans intensification, Uni fait réellement mieux que TS sur les configurations 4-8 et 5-7. C'est l'inverse pour 1-10. Par contre PR arrive à obtenir les meilleurs résultats de ce groupe.

Le groupe sans pondération mais avec intensification peut représenter la situation très concrète dans laquelle un utilisateur sans expérience du problème ne saurait pas quelle pondération donner aux contraintes du modèle. Dans ce cas on peut voir que PR parvient encore à obtenir de meilleurs résultats et Uni devance légèrement TS.

Avec pondération des contraintes. Pour le groupe avec pondération mais sans intensification, on retrouve à peu près les mêmes différences entre les méthodes, à ceci près que c'est la sélection uniforme qui semble gagner le plus de l'introduction de la pondération.

Enfin dans le dernier groupe qui reprend la pondération et l'intensification, on retrouve les performances optimales des différentes méthodes. On notera que, comme dans le cas du QAP, la sélection uniforme produit des résultats très honorables. Cependant, une population d'opérateurs corrects, en elle-même, n'obtient pas les meilleurs résultats. Privilégier certains opérateurs apporte un bénéfice comme nous le montrent les résultats de PR. Il n'y a pas de différence statistique entre PR et TS. PR avec intensification arrive donc à retrouver les performances de l'algorithme tabou TS. Toutefois, il est intéressant de se rappeler que les mécanismes de diversifaction de ces deux méthodes ne sont pas similaires. TS se repose sur le tabou et des redémarrages alors que PR n'utilise que des opérateurs intégrant le tabou et une sélection uniforme parmi k variables de violations maximum et k meilleures solutions. Ce mécanisme de diversification est basé sur des perturbations très modérées alors qu'un redémarrage est beaucoup plus violent.

Nombre d'itérations

Si l'on s'intéresse au Tableau 7.3 qui rapporte le nombre moyen d'itérations, on observe facilement que la sélection proportionnelle (PR) apporte un net avantage dans ce sens par rapport à la sélection uniforme. De plus le nombre moyen d'itérations de TS est en général supérieur à PR. Ceci s'explique par l'utilisation de voisinages différents et de stratégies de diversification différentes. En effet TS fait moins de diversification en général (au moyen de la liste tabou) mais fait des redémarrages ponctuels. Si TS est dans une region de l'espace de recherche dont elle a du mal à s'échapper par la diversification, elle doit attendre le redémarrage. PR utilise des mécanismes de diversification (légèrement) plus variés à travers les sélecteurs des opérateurs et également grâce au tabou. Ceci semble être une stratégie utile pour le problème étudié.

Analyse d'une exécution

Dans les Figures 7.4–7.6 nous illustrons l'exécution de PR sur la configuration 6 avec 7 périodes en utilisant les pondérations des contraintes et l'intensification. Cette exécution se termine avec succès en 15 525 itérations.

La Figure 7.4 montre les courbes empilées de la fréquence cumulée de la sélection de chaque opérateur. Les fréquences à la dernière itération représentent donc les fréquences de sélection totales des opérateurs. Tous les opérateurs ne sont pas

Config	Nombre moyen d'itérations (secondes)											
	Sans pondération						Avec pondération					
	Sans intensification			Avec intensification			Sans intensification			Avec intensification		
	Uni	PR	TS	Uni	PR	TS	Uni	PR	TS	Uni	PR	TS
1-6	250	**221**	249	250	**219**	249	216	**196**	226	216	**204**	226
1-7	416	**376**	388	416	**372**	388	350	**322**	376	350	**306**	376
1-8	970	**840**	861	970	904	**861**	756	**697**	1725	756	**710**	750
1-9	4630	**3158**	8019	4370	3144	**2944**	2843	**2381**	2996	2644	2300	**1998**
1-10	943830	**316586**	341073	155296	88567	**33914**	554440	**227515**	247284	50695	28679	**19338**
2-6	942	**816**	881	942	**842**	881	761	761	761	761	**720**	761
2-7	3033	**1969**	3218	2893	**2050**	3036	1864	**1504**	2265	1875	**1486**	2189
2-8	47279	**10852**	133170	20057	**10912**	28156	17660	**7157**	50900	10140	**7103**	15199
2-9	1000000	**946882**	1000000	726768	**362940**	732799	991618	**901524**	990532	269584	**123307**	218156
3-6	909	**775**	929	909	**822**	929	711	**636**	769	711	**664**	769
3-7	3015	**2277**	3799	2900	**2506**	3523	1807	**1742**	2517	1783	**1598**	2357
3-8	60580	**13174**	143121	22918	**11365**	37724	18922	**8778**	41309	11535	**7271**	16433
3-9	1000000	**987745**	1000000	767415	**502047**	820982	1000000	**906003**	993045	377672	**175256**	280598
4-6	2051	2356	**1149**	2098	1829	**1149**	1413	1716	**976**	1413	1397	**976**
4-7	6491	**4544**	6336	7017	**3708**	5028	2889	**2740**	3639	2735	**2587**	3469
4-8	481965	**32987**	614348	98033	**27707**	66246	61635	**13693**	120563	21742	**11443**	20962
4-9	1000000	**997385**	1000000	922735	**705899**	949235	1000000	**970613**	1000000	494544	**215666**	431750
5-6	10757	**7583**	68981	11127	**7380**	28239	5173	**4443**	17380	6494	**3703**	9526
5-7	755250	**143354**	997439	464109	**139078**	684932	211945	**44965**	905812	142557	**36924**	141654
6-6	25340	**13294**	199386	25974	**15654**	74758	8666	**7795**	58470	9357	**6157**	21678
6-7	988873	**626927**	1000000	692064	**431470**	981352	570480	**176577**	1000000	270446	**143494**	362559

TABLEAU 7.3 – Nombre moyen d'itérations en fonction de l'utilisation de contraintes pondérées et du composant d'intensification. Les meilleures valeurs pour chaque groupe sont indiquées en gras.

sélectionné avec la même fréquence. On voit par ailleurs que cette fréquence n'est pas constante. L'histogramme de la Figure 7.5 indique les taux de sélection sur toute l'exécution. Les cinq opérateurs les plus sélectionnés ont comme caractéristique commune de considérer les cinq variables subissant le plus de violations et non une seule. Parmi ces cinq, les deux premiers choisissent ensuite le meilleur voisin et non un parmi les cinq meilleurs.

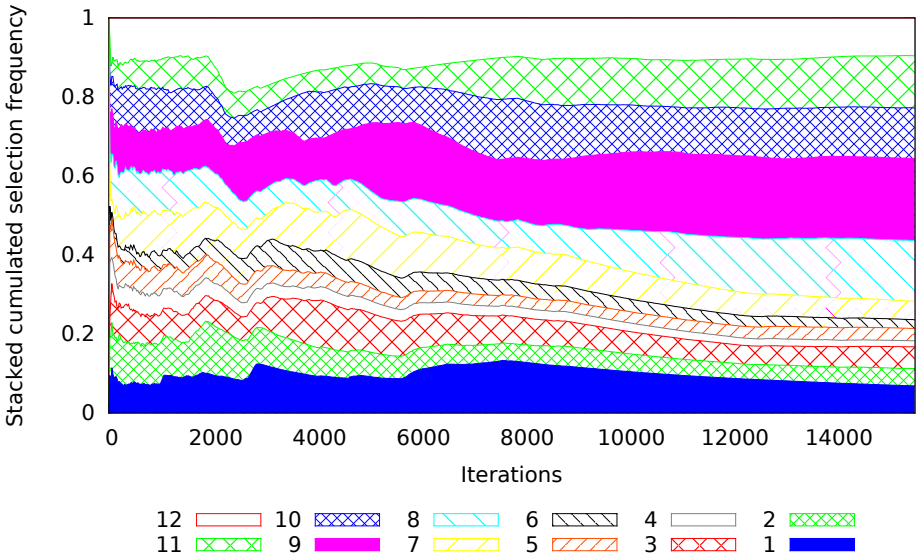
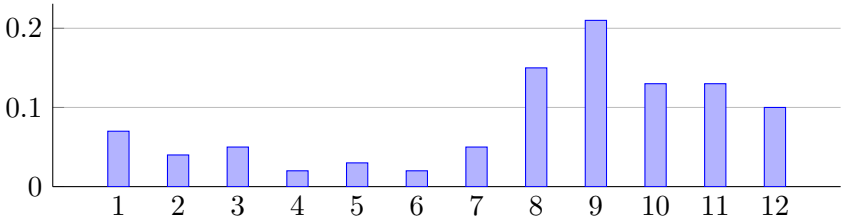


FIGURE 7.4 – Courbes empilées de la fréquence cumulée de sélection des opérateurs fonction de l'itération pour une exécution de la configuration 6 avec 7 périodes.

Ces représentations peuvent toutefois induire l'observateur en erreur en donnant l'impression que les probabilités de sélection sont relativement fixes. Ce n'est pas le cas. La Figure 7.6 montre les courbes empilées des probabilités de sélection des opérateurs en fonction de l'itération pour la même exécution qu'avant.

Sur les 2 000 premières itérations les changements sont relativement rapides et tous les opérateurs ont l'occasion de tenter leur chance. Cela s'explique en partie car au début tous les opérateurs peuvent facilement améliorer la solution précédente.

Pendant la première moitié de la recherche (moins de 8 000 itérations) l'opérateur 1 – qui sélectionne la variable la plus violée et sélectionne le meilleur voisin avec la plus faible longueur tabou – a une probabilité de sélection non négligeable, voire parfois majoritaire. Par la suite sa probabilité de sélection est quasi nulle car



Opérateur	1	2	3	4	5	6	7	8	9	10	11	12
Tabou	7	8	9	7	8	9	7	8	9	7	8	9
K-best	1	1	1	5	5	5	1	1	1	5	5	5
K-violations	1	1	1	1	1	1	5	5	5	5	5	5

FIGURE 7.5 – Taux de sélection des opérateurs pour une exécution de la configuration 6 avec 7 périodes.

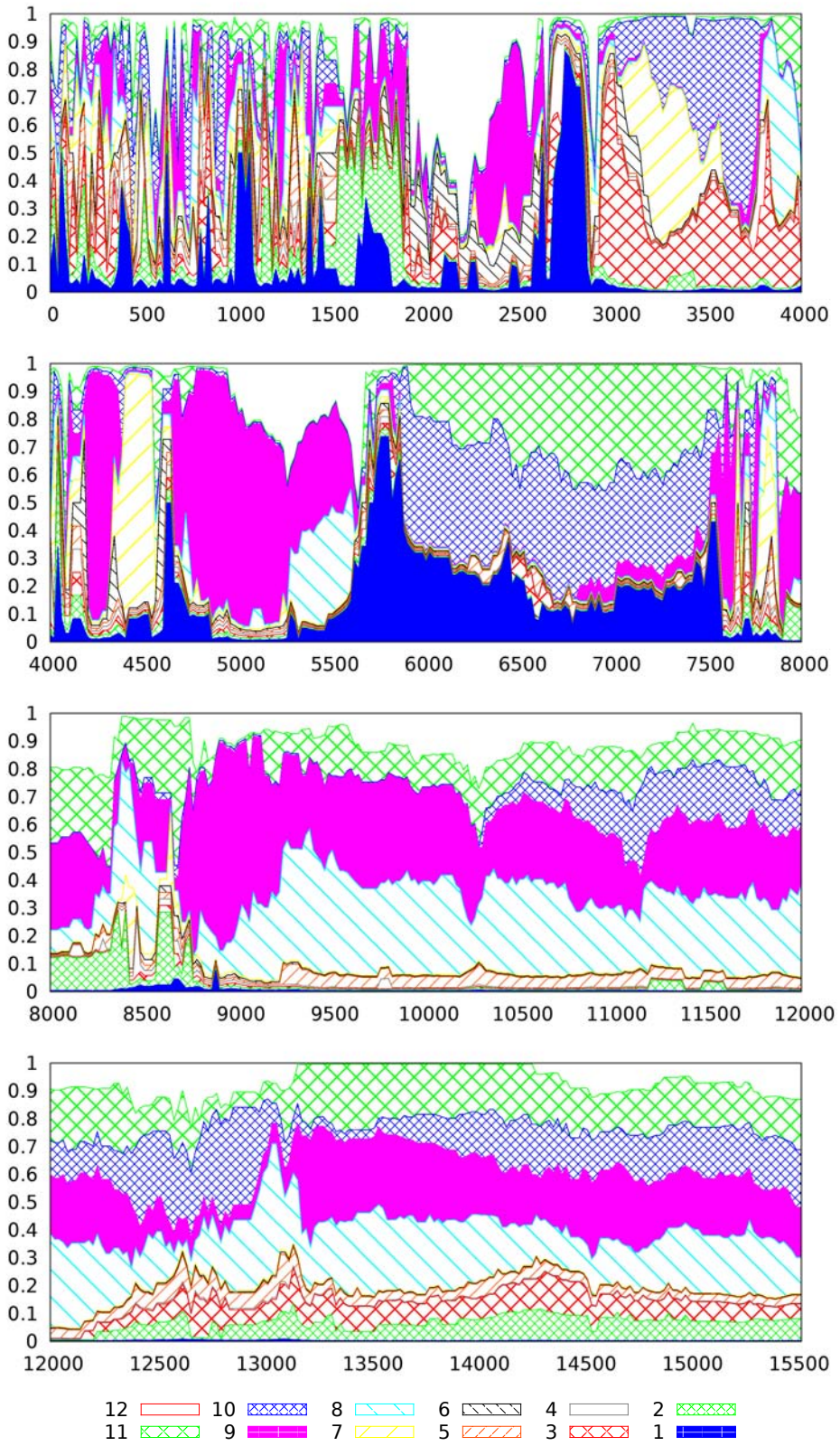


FIGURE 7.6 – Courbes empilées des probabilités de sélection des opérateurs en fonction de l'itération pour une exécution de la configuration 6 avec 7 périodes.

il n'apporte pas assez de diversification – dû à la longueur tabou ou le choix parmi les cinq meilleurs voisins.

Les probabilités de sélection sur la seconde moitié de la recherche sont monopolisées par les opérateurs 8-12 qui font plus de diversification à travers les mécanismes mentionnés juste avant.

7.3.6 Résultats et discussion pour les stratégies adaptatives

Maintenant que nous avons vu qu'il était intéressant d'utiliser plusieurs opérateurs sélectionnés proportionnellement, passons aux mécanismes plus avancés évoqués dans les chapitres précédents : la Poursuite Adaptative (AP) sur le principe du gagnant-remporte-tout utilisant l'utilité U_p et la stratégie de diversité correcte (CD) utilisant comme utilité la somme positive des déplacements pondérés. Le Tableau 7.4 présente les résultats de ces méthodes et reprend les résultats de PR et TS pour le cas où l'on considère un modèle avec des contraintes pondérées et un composant d'intensification.

Remarquons tout d'abord, et en considérant le taux de réussite, que CD semble améliorer les résultats de PR mais sans que cette amélioration ne soit réellement significative. AP quant à elle se retrouve dernière, à un niveau équivalent à celui de la sélection uniforme. La stratégie du gagnant-remporte-tout ne fonctionne pas réellement.

Au niveau des itérations moyennes, PR et CD font plus ou moins jeu égal. Les nettes différences sur les configurations les plus difficiles viennent essentiellement du fait qu'il est pénalisant de ne pas avoir trouvé une solution dans le nombre d'itérations imparti, ce qui fait remonter défavorablement la moyenne des itérations. Dans les cas où une solution est trouvée à chaque fois, les différences sont moins flagrantes.

Enfin examinons les temps d'exécution. Ici la recherche tabou est nettement plus rapide : d'environ un ordre de grandeur. Les autres méthodes reposent sur l'architecture de notre solveur qui est une surcouche de Comet, ainsi les processus de gestion des fenêtres glissantes, de calcul de l'impact, de calcul de l'utilité et les différents mécanismes de communications entre les composants jouent en leur défaveur. Notons que CD est plus rapide que PR. PR a bénéficié du réglage par F-Race basé sur l'optimisation du temps d'exécution.

Dans le cas du PPP, la sélection d'opérateurs nous permet donc d'atteindre un niveau équivalent de performance par rapport à la meilleure méthode connue mais, comme nous pouvions nous y attendre, au prix d'un temps de résolution supérieur.

7.4 Conclusion

Dans ce chapitre nous avons étudié l'application des méthodes des chapitres précédents à un problème de satisfaction de contraintes. Ceci a demandé de traduire les contraintes, et plus explicitement leurs violations, en une fonction à minimiser et pouvant donc servir de fonction objectif. Nous avons évoqué le fait que définir cette fonction n'est pas trivial, notamment, lorsque cette fonction est l'agrégation de plusieurs fonctions de violation et qu'il faut choisir une pondération des contraintes.

Nous avons étudié le Progressive Party Problem en nous comparant, à notre connaissance, à la meilleure méthode de recherche locale développée pour ce problème. Cette méthode, une recherche tabou ajustée pour le problème, a été décrite en détail.

De nouveaux voisinages, sélecteurs et opérateurs ont été introduits pour pouvoir gérer les problèmes à variables entières qui ne sont pas des permutations. En sus

Config	Résolution				Itérations				Temps (secondes)			
	AP	PR	CD	TS	AP	PR	CD	TS	AP	PR	CD	TS
1-6	100	100	100	100	215	**204**	209	226	0.39	0.36	0.24	**0.13**
1-7	100	100	100	100	328	**306**	321	376	0.55	0.51	0.37	**0.20**
1-8	100	100	100	100	757	710	**681**	750	1.11	1.07	0.66	**0.25**
1-9	100	100	100	100	2847	2300	2104	**1998**	3.68	3.19	1.74	**0.44**
1-10	100	100	100	100	48124	28679	37889	**19338**	60.29	38.70	28.33	**2.15**
2-6	100	100	100	100	807	**720**	772	761	1.00	0.92	0.57	**0.18**
2-7	100	100	100	100	1853	1486	**1343**	2189	2.16	1.87	1.01	**0.36**
2-8	100	100	100	100	10752	7103	**5384**	15199	12.16	8.74	3.75	**1.50**
2-9	87	97	**99**	**99**	282528	123307	**101930**	218156	329.74	154.96	70.96	**19.70**
3-6	100	100	100	100	710	**664**	754	769	0.89	0.87	0.55	**0.18**
3-7	100	100	100	100	1813	**1598**	1580	2357	2.15	2.01	1.14	**0.36**
3-8	100	100	100	100	11940	7271	**6593**	16433	13.58	8.94	4.56	**1.59**
3-9	81	98	95	**100**	423701	**175256**	194428	280598	498.38	220.23	134.30	**25.27**
4-6	100	100	100	100	1492	1397	2200	**976**	1.69	1.66	1.35	**0.20**
4-7	100	100	100	100	3019	**2587**	2991	3469	3.48	3.13	1.98	**0.44**
4-8	100	100	100	100	20922	**11443**	11796	20962	24.02	14.01	7.89	**1.97**
4-9	75	96	**98**	93	487386	215666	**156460**	431750	570.36	270.38	107.63	**39.04**
5-6	100	100	100	100	5261	**3703**	4802	9526	5.51	4.19	2.81	**0.82**
5-7	96	100	100	100	144820	36924	**34298**	141654	150.28	42.06	21.09	**11.14**
6-6	100	100	100	100	8686	**6157**	8362	21678	8.98	6.74	4.80	**1.69**
6-7	84	96	**98**	90	300979	143494	**116907**	362559	302.79	157.72	70.83	**28.07**

TABLEAU 7.4 – Pourcentage de configurations résolues, nombre moyen d'itérations, temps moyen d'exécution en secondes pour la Poursuite Adaptative (AP), le choix proportionnel (PR), la stratégie dynamique (CD) et la recherche tabou (TS). Les meilleures valeurs pour chaque groupe sont indiquées en gras.

nous avons introduit la possibilité d'utiliser un critère tabou pour configurer un opérateur. Un mécanisme d'intensification similaire à celui employé dans la méthode concurrente est aussi ajouté.

En terme de résultats, ce chapitre à montré qu'une sélection proportionnelle en fonction d'une utilité basée sur un compromis implicite entre qualité et diversité donne de bons résultats. En particulier, lorsque les pondérations sont supprimées ou que le mécanisme d'intensification est absent, la sélection proportionnelle bat la méthode tabou concurrente. Avec les pondérations et l'intensification, les deux méthodes obtiennent des taux de résolution équivalents.

La stratégie adaptative CD apporte une légère amélioration des taux de résolution mais également du temps d'exécution par rapport à la sélection proportionnelle. Elle permet également de se rapprocher des temps d'exécution de la recherche tabou.

En analysant le résultat d'une exécution, nous avons observé que tous les opérateurs ne se comportent pas de la même façon. Ce genre d'information pourra être utile si l'utilisateur souhaite utiliser notre solveur comme une aide à la conception d'algorithmes plus spécifiques et efficaces.

Comme nous l'avons déjà dit et montré, la pondération des contraintes est un élément important de la résolution. Il sera donc intéressant d'intégrer aux opérateurs la possibilité de manipuler ces pondérations.

Conclusion générale

Le problème de la configuration d'algorithmes se présente dans de nombreuses situations et notamment dans le domaine de l'optimisation et des problèmes de satisfaction de contraintes. La configuration d'un algorithme se réfère au système de composants et de paramètres dont il est constitué. La recherche autonome regroupe les différentes techniques permettant de configurer un algorithme avant ou pendant son exécution et notamment de pouvoir utiliser le même algorithme sur des problèmes variés, ayant des instances diverses, en automatisant le processus de configuration des paramètres internes d'un algorithme.

D'un point de vue théorique, il est impossible de concevoir un algorithme fonctionnant de manière optimale sur tous les problèmes. Néanmoins, dans le contexte de la recherche autonome, nous pouvons chercher à développer des algorithmes relativement robustes pour fonctionner sur une palette plus ou moins large de problèmes.

Les algorithmes de recherche locale sont des algorithmes généralement stochastiques qui fonctionnent en appliquant de façon itérative des perturbations locales à une solution dans la quête de la meilleure solution. Ces perturbations sont générées par des opérateurs. Ces opérateurs permettent d'améliorer la solution ou de l'altérer, afin de mieux explorer l'espace de recherche. Il faut donc prendre en compte un compromis entre exploitation et exploration car ces deux objectifs sont généralement divergeant.

Les algorithmes de recherche locale sont conçus pour être utilisables sur différents problèmes mais leur adaptation et leur configuration pour un problème donné demande souvent une expérience non négligeable de la part de l'utilisateur.

La sélection adaptative d'opérateurs s'inscrit dans le contexte de la recherche autonome et a pour but de sélectionner les opérateurs à appliquer au cours de la recherche dans l'objectif de faciliter la tâche de l'utilisateur en limitant le nombre de choix qu'il doit faire pour utiliser l'algorithme (les paramètres à configurer).

En nous inspirant de travaux récents en sélection adaptative d'opérateur dans le contexte des algorithmes évolutionnaires, nous avons proposé des approches pour l'optimisation dans le contexte de la recherche locale.

Nous avons notamment introduit la notion de compromis entre qualité et diversité. La diversité est intuitivement présente dans les algorithmes évolutionnaires et plus difficile à appréhender dans le contexte de la recherche locale. Ainsi au lieu de considérer un population de solutions nous considérerons une partie du chemin de recherche. Ces solutions nous permettent de baser les choix d'opérateurs sur leurs performances antérieures.

Nos premières expériences montrent qu'avec un ensemble d'opérateurs relativement corrects il est possible d'obtenir des performances elles aussi correctes, simplement en faisant un choix uniforme parmi ces opérateurs. Notre approche qui utilise un compromis implicite entre qualité et diversité grâce à la Pareto dominance issue de l'optimisation multiobjectif permet d'améliorer légèrement ces résultats. La différence est plus explicite lorsque l'ensemble des opérateurs est moins bien choisi.

Nous nous sommes ensuite intéressés à introduire une stratégie permettant de faire évoluer automatiquement le compromis. Nos expériences, sur un ensemble d'opérateurs non optimaux, montrent que cette stratégie est en mesure de produire de meilleurs résultats qu'avec des approches sans stratégie. Cette stratégie s'avère robuste par rapport aux valeurs de ses paramètres.

Une extension de notre approche a été proposée pour les problèmes de satisfaction de contraintes. Des opérateurs intégrant plus de fonctionnalités ont été introduits. Sur le problème testé notre approche permet de concurrencer, en terme de taux de résolution, une approche de recherche locale développée spécifiquement pour le problème.

Perspectives de recherche

Ce travail ouvre de nombreuses perspectives de recherche. À nos yeux une perspective majeure est la génération automatique des opérateurs à partir du modèle du problème. En effet, dans le travail présenté, il est possible de générer combinatoirement des opérateurs en utilisant différentes briques de bases. Toutefois l'ensemble des opérateurs est un ensemble par défaut intégré au solveur qui peut être modifié selon le souhait de l'utilisateur. Comme le montre nos expériences, le choix des opérateurs est important : avec des opérateurs bien choisis, c'est-à-dire qui correspondent au modèle et à sa sémantique, même des mécanismes de sélection simples permettent d'obtenir de bons résultats. Ainsi, si la génération des opérateurs pouvait se faire automatiquement à partir du modèle, cela permettrait vraisemblablement d'obtenir des ensembles d'opérateurs plus en phase avec le problème.

Pour mettre en avant le bénéfice de la sélection d'opérateurs nous avons considéré des ensembles d'opérateurs que nous savions ne rien apporter à la recherche, voire même lui être nuisible. À chaque fois ces opérateurs sont conservés. Il serait possible d'employer des techniques de gestion adaptative d'opérateurs, comme dans [Maturana *et al.*, 2010], pour gérer l'ensemble des opérateurs et ne conserver que les meilleurs. Les mécanismes existants pourraient être réutilisés afin de « trier » les opérateurs et trouver un ensemble de bons opérateurs. Des évaluations périodiques des « mauvais » permettraient de décider s'il est approprié de les réintroduire dans le processus de sélection. Par ailleurs, la gestion des opérateurs sera aussi utile lorsque les opérateurs seront automatiquement générés à partir du modèle car il y en aura potentiellement beaucoup.

Du point de vue de la robustesse, il est bien évident que, si nous souhaitons mettre cette caractéristique vraiment en avant, il sera nécessaire de tester d'avantage de problèmes aux modèles variés et aux sémantiques différentes. De plus, ici nous n'avons considéré que des problèmes d'optimisation sans réelles contraintes et qu'un problème de satisfaction de contraintes. Il sera pertinent de considérer également des problèmes d'optimisation sous contraintes. Bien évidemment il se posera alors la question de comment combiner fonction objectif et fonction de violation.

Une autre piste de recherche pour la sélection d'opérateurs est l'optimisation multiobjectif. En effet le fait d'avoir plusieurs objectifs se prête bien à l'utilisation de différents opérateurs qui fonctionneraient plus ou moins bien selon l'objectif.

Enfin, d'un point de vue d'ingénierie logicielle, notre solveur actuel demande à être optimisé et mieux documenté afin de pouvoir espérer le distribuer.

Publications personnelles

Articles de conférences internationales avec comité de lecture

- **An Exploration-Exploitation Compromise-Based Adaptive Operator Selection for Local Search**, avec Jorge Maturana et Frédéric Saubion. Dans Terence Soule, éditeur, *Proceedings of the Fourteenth International Genetic and Evolutionary Computation Conference (GECCO'12)*, pages 1277–1284. ACM, New York, NY, USA, 2012.
- **A Comparison of Operator Utility Measures for On-line Operator Selection in Local Search**, avec Jorge Maturana et Frédéric Saubion. Dans Youssef Hamadi et Marc Schoenauer, éditeurs, *Learning and Intelligent Optimization, Proceedings of the 6th Learning and Intelligent Optimization Conference (LION6)*, Lecture Notes in Computer Science, vol. 7219, pp. 497–502. Springer Berlin / Heidelberg, 2012.
- **Pareto Autonomous Local Search**, avec Frédéric Saubion. Dans Carlos A. Coello Coello, éditeur, *Learning and Intelligent Optimization, Proceedings of the 5th Learning and Intelligent Optimization Conference (LION5)*, Lecture Notes in Computer Science, vol. 6683, pp. 392–406. Springer Berlin / Heidelberg, 2011.

Articles de conférence nationale avec comité de lecture

- **Sélection adaptative d'opérateurs pour la recherche locale basée sur un compromis exploration-exploitation**, avec Jorge Maturana et Frédéric Saubion, *Actes des Huitièmes Journées Francophones de Programmation par Contraintes (JFPC 2012)*, pp. 318–327. Toulouse, France, mai 2012.
- **Sélection autonome d'opérateurs par dominance pour la recherche locale**, avec F. Saubion, *Actes des Septièmes Journées Francophones de Programmation par Contraintes (JFPC 2011)*, pp. 307–316. Lyon, France, juin 2011.

Article de workshop

- **Hyperheuristic as Component of a Multi-Objective Metaheuristic**, avec Dario Landa-Silva et Xavier Gandibleux. Dans Frank Hutter et Marco A. Montes de Oca, éditeurs, *SLS-DS 2009 : Doctoral Symposium on Engineering Stochastic Local Search Algorithms*, Technical Report TR/IRIDIA/2009-024, IRIDIA, Université Libre de Bruxelles, pp. 51–55. Bruxelles, Belgique, septembre 2009.

Références bibliographiques

[Acan, 2005] Adnan Acan. An external partial permutations memory for ant colony optimization. Dans Günther Raidl et Jens Gottlieb, éditeurs, *Evolutionary Computation in Combinatorial Optimization*, volume 3448 of *Lecture Notes in Computer Science*, pages 1–11. Springer Berlin / Heidelberg, 2005. (cité page 56)

[Ackley, 1987] David. H. Ackley. *A connectionist machine for genetic hillclimbing*. Kluwer Academic Publishers, 1987. (cité page 54)

[Auer *et al.*, 2002] Peter Auer, Nicolò Cesa-Bianchi, et Paul Fischer. Finite-time analysis of the multiarmed bandit problem. *Machine Learning*, 47(2) :235–256, 2002. (cité page 40)

[Back, 1996] Thomas Back. *Evolutionary Algorithms in Theory and Practice : Evolution Strategies, Evolutionary Programming, Genetic Algorithms*. Oxford University Press, janvier 1996. (cité page 20)

[Battiti *et al.*, 2008] Roberto Battiti, Mauro Brunato, et Franco Mascia. *Reactive Search and Intelligent Optimization*. Springer Publishing Company, Incorporated, 2008. (cité page 32)

[Battiti et Tecchiolli, 1994] Roberto Battiti et Giampietro Tecchiolli. The reactive tabu search. *INFORMS Journal on Computing*, 6(2) :126–140, janvier 1994. (cité page 32)

[Beldiceanu et Contejean, 1994] Nicolas Beldiceanu et Evelyne Contejean. Introducing global constraints in CHIP. *Mathematical and Computer Modelling*, 20(12) :97–123, décembre 1994. (cité page 48)

[Benoist *et al.*, 2011] Thierry Benoist, Bertrand Estellon, Frédéric Gardi, Romain Megel, et Karim Nouioua. LocalSolver 1.x : a black-box local-search solver for 0-1 programming. *4OR : A Quarterly Journal of Operations Research*, 9(3) :299–316, 2011. (cité page 45)

[Birattari *et al.*, 2010] Mauro Birattari, Zhi Yuan, Prasanna Balaprakash, et Thomas Stützle. F-race and iterated f-race : An overview. Dans Thomas Bartz-Beielstein, Marco Chiarandini, Luis Paquete, et Mike Preuss, éditeurs, *Experimental Methods for the Analysis of Optimization Algorithms*, pages 311–336. Springer-Verlag, Berlin, Heidelberg, 2010. (cité page 29)

[Birattari, 2004] Mauro Birattari. *The Problem of Tuning Metaheuristics as seen from a machine learning perspective*. Thèse de doctorat, Université Libre de Bruxelles, Bruxelles, Belgique, décembre 2004. (cité pages 29, 31 et 95)

[Blum et Mastrolilli, 2007] Christian Blum et Monaldo Mastrolilli. Using branch & bound concepts in construction-based metaheuristics : Exploiting the dual problem knowledge. Dans Thomas Bartz-Beielstein, María Blesa Aguilera, Christian Blum, Boris Naujoks, Andrea Roli, Günther Rudolph, et Michael Sampels, éditeurs, *Hybrid Metaheuristics*, volume 4771 of *Lecture Notes in Computer Science*, pages 123–139. Springer Berlin / Heidelberg, 2007. (cité page 9)

[Blum et Roli, 2003] Christian Blum et Andrea Roli. Metaheuristics in Combinatorial Optimization : Overview and Conceptual Comparison. *ACM Computing Surveys*, 35(3) :268–308, 2003. (cité page 13)

[Bräysy, 2003] Olli Bräysy. A reactive variable neighborhood search for the vehicle-routing problem with time windows. *INFORMS Journal on Computing*, 15(4) :347–368, décembre 2003. (cité page 33)

[Burkard *et al.*, 1997] Rainer E. Burkard, Stefan E. Karisch, et Franz Rendl. QAPLIB – a quadratic assignment problem library. *Journal of Global Optimization*, 10(4) :391–403, juin 1997. (cité pages 55, 81, 82 et 96)

[Burke *et al.*, 2003] Edmund Burke, Graham Kendall, Jim Newall, Emma Hart, Peter Ross, et Sonia Schulenburg. Hyper-Heuristics : an emerging direction in modern search technology. Dans Fred Glover et Gary A. Kochenberger, éditeurs, *Handbook of Metaheuristics*, pages 457–474. Springer, 2003. (cité page 34)

[Burke *et al.*, 2010] Edmund K. Burke, Matthew Hyde, Graham Kendall, Gabriela Ochoa, Ender Özcan, et John R. Woodward. A classification of hyper-heuristic approaches. Dans Michel Gendreau et Jean-Yves Potvin, éditeurs, *Handbook of Metaheuristics*, volume 146 of *International Series in Operations Research & Management Science*, pages 449–468. Springer US, 2010. (cité page 34)

[Candan *et al.*, 2012] Caner Candan, Adrien Goeffon, Frédéric Lardeux, et Frédéric Saubion. A dynamic island model for adaptive operator selection. Dans *Proceedings of the fourteenth international conference on Genetic and evolutionary computation conference*, GECCO '12, page 1253–1260, New York, NY, USA, 2012. ACM. (cité page 54)

[Chakhlevitch et Cowling, 2008] Konstantin Chakhlevitch et Peter Cowling. Hyperheuristics : Recent developments. Dans Carlos Cotta, Marc Sevaux, et Kenneth Sörensen, éditeurs, *Adaptive and Multilevel Metaheuristics*, volume 136 of *Studies in Computational Intelligence*, pages 3–29. Springer Berlin / Heidelberg, 2008. (cité page 34)

[Chiarandini *et al.*, 2007] Marco Chiarandini, Luís Paquete, Mike Preuss, et Enda Ridge. Experiments on metaheuristics : Methodological overview and open issues. Technical Report DMF-2007-03-003, The Danish Mathematical Society, 2007. (cité page 81)

[Cirasella *et al.*, 2001] Jill Cirasella, David S. Johnson, Lyle A. McGeoch, et Weixiong Zhang. The asymmetric traveling salesman problem : Algorithms, instance generators, and tests. Dans Adam L. Buchsbaum et Jack Snoeyink, éditeurs, *Algorithm Engineering and Experimentation*, volume 2153, pages 32–59. Springer Berlin Heidelberg, Berlin, Heidelberg, 2001. (cité pages 60 et 82)

[Codognet et Diaz, 2001] Philippe Codognet et Daniel Diaz. Yet another local search method for constraint solving. Dans Kathleen Steinhöfel, éditeur, *Stochastic Algorithms : Foundations and Applications*, volume 2264 of *Lecture Notes in Computer Science*, pages 342–344. Springer Berlin / Heidelberg, 2001. (cité page 44)

[Coello Coello, 2002] Carlos 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, janvier 2002. (cité page 102)

[Cowling *et al.*, 2001] Peter Cowling, Graham Kendall, et Eric Soubeiga. A hyperheuristic approach to scheduling a sales summit. Dans Edmund Burke et Wilhelm Erben, éditeurs, *Practice and Theory of Automated Timetabling III*, volume 2079 of *Lecture Notes in Computer Science*, pages 176–190. Springer Berlin / Heidelberg, 2001. (cité page 33)

[Da Costa *et al.*, 2008] Luis Da Costa, Álvaro Fialho, Marc Schoenauer, et Michèle Sebag. Adaptive operator selection with dynamic multi-armed bandits. Dans *Proceedings of the 10th annual conference on Genetic and evolutionary computation*, pages 913–920, Atlanta, GA, USA, 2008. ACM. (cité page 41)

[Davis, 1989] Lawrence Davis. Adapting operator probabilities in genetic algorithms. Dans *Proceedings of the 3rd International Conference on Genetic Algorithms*, page 61–69, San Francisco, CA, USA, 1989. Morgan Kaufmann Publishers Inc. (cité pages 35 et 37)

[De Jong, 2006] Kenneth A. De Jong. *Evolutionary Computation : A Unified Approach*. The MIT Press, février 2006. (cité page 20)

[Dorigo, 1992] Marco Dorigo. *Optimization, Learning and Natural Algorithms*. Thèse de doctorat, Politecnico di Milano, Italy, 1992. (cité page 20)

[Drezner, 2005a] Zvi Drezner. Compounded genetic algorithms for the quadratic assignment problem. *Operations Research Letters*, 33(5) :475–480, septembre 2005. (cité page 56)

[Drezner, 2005b] Zvi Drezner. The extended concentric tabu for the quadratic assignment problem. *European Journal of Operational Research*, 160(2) :416–422, janvier 2005. (cité page 56)

[Dueck, 1993] Gunter Dueck. New optimization heuristics : The great deluge algorithm and the record-to-record travel. *Journal of Computational Physics*, 104(1) :86–92, janvier 1993. (cité page 35)

[Dumitrescu et Stützle, 2010] Irina Dumitrescu et Thomas Stützle. Usage of exact algorithms to enhance stochastic local search algorithms. Dans Vittorio Maniezzo, Thomas Stützle, Stefan Voß, Ramesh Sharda, et Stefan Voß, éditeurs, *Matheuristics*, volume 10 of *Annals of Information Systems*, pages 103–134. Springer US, 2010. (cité page 9)

[Eiben *et al.*, 1999] Ágoston E. Eiben, R. Hinterding, et Zbigniew Michalewicz. Parameter control in evolutionary algorithms. *IEEE Transactions on Evolutionary Computation*, 3(2) :124 –141, 1999. (cité pages 26 et 27)

[Eiben *et al.*, 2007] Ágoston E. Eiben, Zbigniew Michalewicz, Marc Schoenauer, et James E. Smith. Parameter control in evolutionary algorithms. Dans Fernando Lobo, Cláudio Lima, et Zbigniew Michalewicz, éditeurs, *Parameter Setting in Evolutionary Algorithms*, volume 54 of *Studies in Computational Intelligence*, pages 19–46. Springer Berlin / Heidelberg, 2007. (cité pages 26 et 27)

[Eiben et Smit, 2012] Ágoston E. Eiben et Selmar K. Smit. Evolutionary algorithm parameters and methods to tune them. Dans Youssef Hamadi, Éric Monfroy, et Frédéric Saubion, éditeurs, *Autonomous Search*, pages 15–36. Springer Berlin Heidelberg, 2012. (cité page 28)

[Fang *et al.*, 1994] Hsiao-Lan Fang, Peter Ross, et Dave Corne. A promising hybrid GA/Heuristic approach for open-shop scheduling problems. Dans *Proceedings of the 11th European Conference on Artificial Intelligence*, pages 590–594. John Wiley and Sons, 1994. (cité page 35)

[Fialho *et al.*, 2008] Álvaro Fialho, Luís Da Costa, Marc Schoenauer, et Michèle Sebag. Extreme value based adaptive operator selection. Dans *Parallel Problem Solving from Nature – PPSN X*, pages 175–184. Springer, 2008. (cité page 37)

[Fialho, 2010] Álvaro Fialho. *Adaptive Operator Selection for Optimization*. Thèse de doctorat, Université Paris-Sud 11, Orsay, France, décembre 2010. (cité pages 35, 37 et 42)

[Filho *et al.*, 2009] Carmelo Filho, Fernando de Lima Neto, Anthony Lins, Antônio Nascimento, et Marília Lima. Fish school search. Dans Raymond Chiong, éditeur, *Nature-Inspired Algorithms for Optimisation*, volume 193 of *Studies in Computational Intelligence*, pages 261–277. Springer Berlin / Heidelberg, 2009. (cité page 20)

[Fischetti *et al.*, 2003] Matteo Fischetti, Andrea Lodi, et Paolo Toth. Solving real-world ATSP instances by branch-and-cut. Dans Michael Jünger, Gerhard Reinelt, et Giovanni Rinaldi, éditeurs, *Combinatorial Optimization — Eureka, You Shrink !*, volume 2570, pages 64–77. Springer Berlin Heidelberg, Berlin, Heidelberg, 2003. (cité page 60)

[Galinier et Hao, 2004] Philippe Galinier et Jin-Kao Hao. A general approach for constraint solving by local search. *Journal of Mathematical Modelling and Algorithms*, 3 :73–88, 2004. (cité pages 44, 62 et 104)

[Gamboa *et al.*, 2006] Dorabela Gamboa, César Rego, et Fred Glover. Implementation analysis of efficient heuristic algorithms for the traveling salesman problem. *Computers & Operations Research*, 33(4) :1154–1172, 2006. (cité page 60)

[Garey et Johnson, 1990] Michael R. Garey et David S. Johnson. *Computers and Intractability : A Guide to the Theory of NP-Completeness.* W. H. Freeman & Co., 1990. (cité pages 12, 54 et 57)

[Glover et McMillan, 1986] Fred Glover et Claude McMillan. The general employee scheduling problem. an integration of MS and AI. *Computers & Operations Research*, 13(5) :563–573, 1986. (cité page 17)

[Glover, 1986] Fred Glover. Future paths for integer programming and links to artificial intelligence. *Computers & Operations Research*, 13(5) :533–549, 1986. (cité page 13)

[Glover, 1996] Fred Glover. Ejection chains, reference structures and alternating path methods for traveling salesman problems. *Discrete Applied Mathematics*, 65(1–3) :223–253, mars 1996. (cité page 60)

[Goldberg, 1989] David E. Goldberg. *Genetic algorithms in search, optimization, and machine learning.* Addison-Wesley, 1989. (cité page 20)

[Goldberg, 1990] David E. Goldberg. Probability matching, the magnitude of reinforcement, and classifier system bidding. *Machine Learning*, 5(4) :407–425, 1990. (cité page 38)

[Gomory, 1958] Ralph E. Gomory. Outline of an algorithm for integer solutions to linear programs. *Bulletin of the American Mathematical Society*, 64(5) :275–278, septembre 1958. (cité page 60)

[Hamadi *et al.*, 2011] Youssef Hamadi, Eric Monfroy, et Frédéric Saubion. What is autonomous search ? Dans Pascal van Hentenryck et Michela Milano, éditeurs, *Hybrid Optimization*, volume 45, pages 357–391. Springer New York, New York, NY, 2011. (cité pages 25 et 27)

[Hamadi *et al.*, 2012] Youssef Hamadi, Éric Monfroy, et Frédéric Saubion. *Autonomous Search.* Springer Berlin Heidelberg, 2012. (cité pages 24 et 42)

[Hartland *et al.*, 2006] Cédric Hartland, Sylvain Gelly, Nicolas Baskiotis, Olivier Teytaud, et Michèle Sebag. Multi-armed bandit, dynamic environments and meta-bandits. *Online Trading of Exploration and Exploitation Workshop, NIPS*, novembre 2006. (cité page 41)

[Holland, 1975] John H. Holland. *Adaptation in natural and artificial systems.* University of Michigan Press, 1975. (cité page 20)

[Hoos et Stützle, 2004] Holger H. Hoos et Thomas Stützle. *Stochastic Local Search : Foundations & Applications*. Elsevier, septembre 2004. (cité page 13)

[Hoos, 2012] Holger H. Hoos. Programming by optimization. *Communications of the ACM*, 55(2) :70, février 2012. (cité page 31)

[Hutter *et al.*, 2007] Frank Hutter, Holger H. Hoos, et Thomas Stützle. Automatic algorithm configuration based on local search. Dans *Proceedings of the 22nd national conference on Artificial intelligence - Volume 2*, AAAI'07, page 1152–1157. AAAI Press, 2007. (cité pages 24 et 29)

[Hutter *et al.*, 2010] Frank Hutter, Holger Hoos, et Kevin Leyton-Brown. Automated configuration of mixed integer programming solvers. Dans Andrea Lodi, Michela Milano, et Paolo Toth, éditeurs, *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, volume 6140 of *Lecture Notes in Computer Science*, pages 186–202. Springer Berlin / Heidelberg, 2010. (cité page 31)

[Johnson *et al.*, 2002] David S. Johnson, Gregory Gutin, Lyle A. McGeoch, Anders Yeo, Weixiong Zhang, et Alexey Zverovich. Experimental analysis of heuristics for the ATSP. Dans Gregory Gutin et Abraham P. Punnen, éditeurs, *The Traveling Salesman Problem and its Variations*. Springer, 2002. (cité page 59)

[Julstrom, 1997] Bryant A. Julstrom. Adaptive operator probabilities in a genetic algorithm that applies three operators. Dans *Proceedings of the 1997 ACM symposium on Applied computing*, SAC '97, page 233–238, New York, NY, USA, 1997. ACM. (cité page 37)

[Kaelbling *et al.*, 1996] Leslie Pack Kaelbling, Michael L Littman, et Andrew W Moore. Reinforcement learning : A survey. *Journal of Artificial Intelligence Research*, 4 :237—285, 1996. (cité page 35)

[Kanellakis et Papadimitriou, 1980] Paris-C. Kanellakis et Christos H. Papadimitriou. Local search for the asymmetric traveling salesman problem. *Operations Research*, 28(5) :1086–1099, octobre 1980. (cité page 60)

[Karaboga et Akay, 2009] Dervis Karaboga et Bahriye Akay. A survey : algorithms simulating bee swarm intelligence. *Artificial Intelligence Review*, 31(1) :61–85, 2009. (cité page 20)

[Kendall et Hussin, 2005] Graham Kendall et Naimah Hussin. A tabu search hyper-heuristic approach to the examination timetabling problem at the MARA university of technology. Dans Edmund Burke et Michael Trick, éditeurs, *Practice and Theory of Automated Timetabling V*, volume 3616 of *Lecture Notes in Computer Science*, pages 270–293. Springer Berlin / Heidelberg, 2005. (cité page 35)

[Khudabukhsh *et al.*, 2009] Ashiqur R. Khudabukhsh, Lin Xu, Holger H. Hoos, et Kevin Leyton-Brown. SATenstein : automatically building local search SAT solvers from components, 2009. (cité page 31)

[Kirkpatrick *et al.*, 1983] Scott Kirkpatrick, C. Daniel Gelatt, et Mario P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598) :671–680, mai 1983. (cité page 17)

[Koopmans et Beckmann, 1957] Tjalling C. Koopmans et Martin J. Beckmann. Assignment problems and the location of economic activities. *Econometrica*, 25(1) :53–76, janvier 1957. (cité page 54)

[Lai et Robbins, 1985] Tze Leung Lai et Herbert Robbins. Asymptotically efficient adaptive allocation rules. *Advances in applied mathematics*, 6(1) :4–22, 1985. (cité page 40)

[Langdon et Poli, 2002] William B. Langdon et Riccardo Poli. *Foundations of Genetic Programming*. Springer, mars 2002. (cité page 20)

[Laurière, 1978] Jean-Louis Laurière. A language and a program for stating and solving combinatorial problems. *Artificial Intelligence*, 10(1) :29–127, 1978. (cité page 48)

[Lin et Kernighan, 1973] Shen Lin et Brian W. Kernighan. An effective heuristic algorithm for the traveling-salesman problem. *Operations Research*, 21(2) :498–516, 1973. (cité page 59)

[Linhares, 2004] Alexandre Linhares. The structure of local search diversity. Dans *Proceedings of the 5th WSEAS International Conference on Applied Mathematics*, Math'04, page 32 :1–32 :5, Stevens Point, Wisconsin, USA, 2004. World Scientific and Engineering Academy and Society (WSEAS). (cité page 74)

[Loiola *et al.*, 2007] Eliane Maria Loiola, Nair Maria Maia de Abreu, Paulo Oswaldo Boaventura-Netto, Peter Hahn, et Tania Querido. A survey for the quadratic assignment problem. *European Journal of Operational Research*, 176(2) :657–690, janvier 2007. (cité pages 54, 55 et 56)

[Mairy *et al.*, 2010] Jean-Baptiste Mairy, Pierre Schaus, et Yves Deville. Generic adaptive heuristics for large neighborhood search. Dans *Seventh International Workshop on Local Search Techniques in Constraint Satisfaction (LSCS2010). A Satellite Workshop of CP 2010*, Scotland, 2010. (cité page 33)

[Mairy *et al.*, 2011] Jean-Baptiste Mairy, Yves Deville, et Van Hentenryck, Pascal. Reinforced adaptive large neighborhood search. Dans *Eighth International Workshop on Local Search Techniques in Constraint Satisfaction (LSCS2011). A Satellite Workshop of CP 2011*, Italy, 2011. (cité page 33)

[Marmion *et al.*, 2011] Marie-Eleonore Marmion, Clarisse Dhaenens, Laetitia Jourdan, Arnaud Liefooghe, et Sebastien Verel. The road to VEGAS : guiding the search over neutral networks. Dans *Proceedings of the 13th annual conference on Genetic and evolutionary computation*, GECCO '11, page 1979–1986, New York, NY, USA, 2011. ACM. (cité page 33)

[Maron et Moore, 1994] Oded Maron et Andrew W. Moore. Hoeffding races : Accelerating model selection search for classification and function approximation. Dans *Advances in Neural Information Processing Systems*, volume 6, pages 59–66. Morgan Kaufmann Publishers, Inc., 1994. (cité page 29)

[Maturana *et al.*, 2009] Jorge Maturana, Álvaro Fialho, Frédéric Saubion, Marc Schoenauer, et Michèle Sebag. Extreme compass and dynamic multi-armed bandits for adaptive operator selection. Dans *Evolutionary Computation, 2009. CEC '09. IEEE Congress on*, pages 365 –372, 2009. (cité page 37)

[Maturana *et al.*, 2010] Jorge Maturana, Frédéric Lardeux, et Frédéric Saubion. Autonomous operator management for evolutionary algorithms. *Journal of Heuristics*, 16(6) :881–909, 2010. (cité pages 37 et 118)

[Maturana et Saubion, 2008a] Jorge Maturana et Frédéric Saubion. A compass to guide genetic algorithms. Dans Günter Rudolph, Thomas Jansen, Simon Lucas, Carlo Poloni, et Nicola Beume, éditeurs, *Parallel Problem Solving from Nature – PPSN X*, volume 5199 of *Lecture Notes in Computer Science*, pages 256–265. Springer Berlin / Heidelberg, 2008. (cité page 37)

[Maturana et Saubion, 2008b] Jorge Maturana et Frédéric Saubion. On the design of adaptive control strategies for evolutionary algorithms. Dans Nicolas Monmarché, El-Ghazali Talbi, Pierre Collet, Marc Schoenauer, et Evelyne Lutton, éditeurs, *EA 2007*, volume 4926, pages 303–315. Springer, Heidelberg, 2008. (cité page 93)

[Metropolis *et al.*, 1953] Nicholas Metropolis, Arianna Rosenbluth, Marshall N. Rosenbluth, Augusta H. Teller, et Edward Teller. Equation of state calculations by

fast computing machines. *Journal of Chemical Physics*, 21(6) :1087–1092, juin 1953. (cité page 16)

[Michel et Van Hentenryck, 2000] Laurent Michel et Pascal Van Hentenryck. Localizer. *Constraints*, 5(1) :43–84, 2000. (cité page 44)

[Misevicius, 2011] Alfonsas Misevicius. An implementation of the iterated tabu search algorithm for the quadratic assignment problem. *OR Spectrum*, 34(3) :665–690, octobre 2011. (cité page 56)

[Misevičius, 2003] Alfonsas Misevičius. A modified simulated annealing algorithm for the quadratic assignment problem. *Informatica*, 14(4) :497–514, janvier 2003. (cité page 56)

[Misir *et al.*, 2012] Mustafa Misir, Katja Verbeeck, Patrick De Causmaecker, et Greet Vanden Berghe. An intelligent hyper-heuristic framework for CHeSC 2011. Dans Youssef Hamadi et Marc Schoenauer, éditeurs, *Proceedings of the 6th Learning and Intelligent Optimization Conference (LION6)*, Paris, France, 2012. To appear. (cité page 37)

[Mladenović et Hansen, 1997] Nenad Mladenović et Pierre Hansen. Variable neighborhood search. *Computers & Operations Research*, 24(11) :1097–1100, novembre 1997. (cité page 19)

[Moscato, 1989] Pablo Moscato. On evolution, search, optimization, genetic algorithms and martial arts : Towards memetic algorithms. Technical Report 826, California Institute of Technology, 1989. (cité page 20)

[Nagata et Soler, 2012] Yuichi Nagata et David Soler. A new genetic algorithm for the asymmetric traveling salesman problem. *Expert Systems with Applications*, 39(10) :8947–8953, août 2012. (cité page 60)

[Nannen et Eiben, 2006] Volker Nannen et Ágoston E. Eiben. A method for parameter calibration and relevance estimation in evolutionary algorithms. Dans *Proceedings of the 8th annual conference on Genetic and evolutionary computation*, GECCO '06, page 183–190, New York, NY, USA, 2006. ACM. (cité page 28)

[Nannen et Eiben, 2007] Volker Nannen et Ágoston E. Eiben. Efficient relevance estimation and value calibration of evolutionary algorithm parameters. Dans *IEEE Congress on Evolutionary Computation, 2007. CEC 2007*, pages 103 –110, septembre 2007. (cité page 28)

[Nareyek, 2004] Alexander Nareyek. Choosing search heuristics by non-stationary reinforcement learning. Dans *Metaheuristics : computer decision-making*, pages 523–544. Kluwer Academic Publishers, 2004. (cité pages 35 et 76)

[Newton *et al.*, 2011] Hakim Newton, Duc Pham, Abdul Sattar, et Michael Maher. Kangaroo : An efficient Constraint-Based local search system using lazy propagation. Dans Jimmy Lee, éditeur, *Principles and Practice of Constraint Programming – CP 2011*, volume 6876 of *Lecture Notes in Computer Science*, pages 645–659. Springer Berlin / Heidelberg, 2011. (cité page 45)

[Niehaus et Banzhaf, 2001] Jens Niehaus et Wolfgang Banzhaf. Adaption of operator probabilities in genetic programming. Dans Julian Miller, Marco Tomassini, Pier Lanzi, Conor Ryan, Andrea Tettamanzi, et William Langdon, éditeurs, *Genetic Programming*, volume 2038 of *Lecture Notes in Computer Science*, pages 325–336. Springer Berlin / Heidelberg, 2001. (cité page 37)

[Ovacik *et al.*, 2000] Irfan M. Ovacik, Srikanth Rajagopalan, et Reha Uzsoy. Integrating interval estimates of global optima and local search methods for combinatorial optimization problems. *Journal of Heuristics*, 6(4) :481–500, 2000. (cité page 81)

[Page, 1954] E. S. Page. Continuous inspection schemes. *Biometrika*, 41(1/2) :100–115, juin 1954. (cité page 41)

[Papadimitriou et Steiglitz, 1982] Christos H. Papadimitriou et Kenneth Steiglitz. *Combinatorial Optimization, Algorithms and Complexity*. Prentice-Hall, 1982. (cité pages 8 et 12)

[Poli et Graff, 2009] Riccardo Poli et Mario Graff. There is a free lunch for hyper-heuristics, genetic programming and computer scientists. Dans *Proceedings of the 12th European Conference on Genetic Programming*, pages 195–207, Tübingen, Germany, 2009. Springer-Verlag. (cité page 25)

[Puchinger *et al.*, 2010] Jakob Puchinger, Günther R. Raidl, et Sandro Pirkwieser. MetaBoosting : enhancing integer programming techniques by metaheuristics. Dans Vittorio Maniezzo, Thomas Stützle, Stefan Voß, Ramesh Sharda, et Stefan Voß, éditeurs, *Matheuristics*, volume 10 of *Annals of Information Systems*, pages 71–102. Springer US, 2010. (cité page 9)

[Raidl, 2006] Günther Raidl. A unified view on hybrid metaheuristics. Dans Francisco Almeida, María Blesa Aguilera, Christian Blum, José Moreno Vega, Melquíades Pérez Pérez, Andrea Roli, et Michael Sampels, éditeurs, *Hybrid Metaheuristics*, volume 4030 of *Lecture Notes in Computer Science*, pages 1–12. Springer Berlin / Heidelberg, 2006. (cité page 9)

[Rego *et al.*, 2011] César Rego, Dorabela Gamboa, Fred Glover, et Colin Osterman. Traveling salesman problem heuristics : Leading methods, implementations and latest advances. *European Journal of Operational Research*, 211(3) :427–441, juin 2011. (cité page 60)

[Reinelt, 1991] Gerhard Reinelt. TSPLIB - a traveling salesman problem library. *INFORMS Journal on Computing*, 3(4) :376–384, janvier 1991. (cité pages 59 et 81)

[Rice, 1976] John R. Rice. The algorithm selection problem. Dans *Advances in computers*, volume 15, pages 65–118. Academic Press, Inc., 1976. (cité page 24)

[Robbins, 1952] Herbert Robbins. Some aspects of the sequential design of experiments. *Bulletin of the American Mathematical Society*, 58 :527–535, 1952. (cité page 40)

[Roberti et Toth, 2012] Roberto Roberti et Paolo Toth. Models and algorithms for the asymmetric traveling salesman problem : an experimental comparison. *EURO Journal on Transportation and Logistics*, 1(1-2) :113–133, mai 2012. (cité page 59)

[Robinson, 1949] Julia B. Robinson. On the hamiltonian game (A traveling-salesman problem). RAND Research Memorandum RM-303, 1949. (cité page 57)

[Rodman, 1978] Leiba Rodman. On the many-armed bandit problem. *The Annals of Probability*, 6(3) :491–498, juin 1978. (cité page 40)

[Régin, 1994] Jean-Charles Régin. A filtering algorithm for constraints of difference in CSPs. Dans *Proceedings of the twelfth national conference on Artificial intelligence (vol. 1)*, AAAI '94, page 362–367, Menlo Park, CA, USA, 1994. American Association for Artificial Intelligence. (cité page 48)

[Schiavinotto et Stützle, 2007] Tommaso Schiavinotto et Thomas Stützle. A review of metrics on permutations for search landscape analysis. *Comput. Oper. Res.*, 34(10) :3143–3153, 2007. (cité pages 73 et 75)

[Schrijver, 2005] Alexander Schrijver. On the history of combinatorial optimization (Till 1960). Dans Karen Aardal, George L. Nemhauser, et Robert Weismantel, éditeurs, *Discrete Optimization*, volume 12, pages 1–68. Elsevier, 2005. (cité page 57)

[Schumacher *et al.*, 2001] C. Schumacher, Michael D. Vose, et L. Darrell Whitley. The no free lunch and problem description length. Dans *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2001)*, page 565–570. Morgan Kaufmann, 2001. (cité page 24)

[Shaw, 1998] Paul Shaw. Using constraint programming and local search methods to solve vehicle routing problems. Dans Michael Maher et Jean-Francois Puget, éditeurs, *Principles and Practice of Constraint Programming — CP98*, volume 1520 of *Lecture Notes in Computer Science*, pages 417–431. Springer Berlin / Heidelberg, 1998. (cité page 33)

[Shaw, 2004] Paul Shaw. A constraint for bin packing. Dans Mark Wallace, éditeur, *Principles and Practice of Constraint Programming – CP 2004*, volume 3258 of *Lecture Notes in Computer Science*, pages 648–662. Springer Berlin / Heidelberg, 2004. (cité page 49)

[Sidaner *et al.*, 2002] Alain Sidaner, Olivier Bailleux, et Jean-Jacques Chabrier. Measuring the spatial dispersion of evolutionary search processes : Application to walksat. Dans Pierre Collet, Cyril Fonlupt, Jin-Kao Hao, Evelyne Lutton, et Marc Schoenauer, éditeurs, *Artificial Evolution*, volume 2310 of *Lecture Notes in Computer Science*, pages 77–87. Springer Berlin / Heidelberg, 2002. (cité page 74)

[Simonis, 2009] Helmut Simonis. Progress on the progressive party problem. Dans Willem-Jan van Hoeve et John Hooker, éditeurs, *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, volume 5547 of *Lecture Notes in Computer Science*, pages 328–329. Springer Berlin / Heidelberg, 2009. (cité page 62)

[Smith *et al.*, 1996] Barbara M. Smith, Sally C. Brailsford, Peter M. Hubbard, et H. Paul Williams. The progressive party problem : Integer linear programming and constraint programming compared. *Constraints*, 1(1) :119–138, 1996. (cité pages 60, 61 et 62)

[Smith-Miles, 2008] Kate A. Smith-Miles. Cross-disciplinary perspectives on meta-learning for algorithm selection. *ACM Computing Surveys*, 41(1) :1–25, décembre 2008. (cité page 24)

[Soubeiga, 2003] Eric Soubeiga. *Development and application of hyperheuristics to personnel scheduling.* Thèse de doctorat, University of Nottingham, Nottingham, Royaume-Uni, 2003. (cité pages 35 et 37)

[Sprent, 1992] Peter Sprent. *Pratique des statistiques nonparamétriques.* Institut national de recherche agronomique, 1992. Traduit de Applied nonparametric statistical methods. (cité page 81)

[Stattenberger *et al.*, 2007] Günther Stattenberger, Markus Dankesreiter, Florian Baumgartner, et Johannes Schneider. On the neighborhood structure of the traveling salesman problem generated by local search moves. *Journal of Statistical Physics*, 129(4) :623–648, novembre 2007. (cité page 59)

[Stützle, 1998] Thomas Stützle. *Local Search Algorithms for Combinatorial Problems : Analysis, improvements and New Applications.* Thèse de doctorat, Technischen Universität Darmstadt, Darmstadt, Allemagne, 1998. (cité page 18)

[Stützle, 2006] Thomas Stützle. Iterated local search for the quadratic assignment problem. *European Journal of Operational Research*, 174(3) :1519–1539, novembre 2006. (cité page 56)

[Taillard, 1991] Éric Taillard. Robust taboo search for the quadratic assignment problem. *Parallel Computing*, 17(4–5) :443–455, 1991. (cité page 56)

[Thierens, 2005] Dirk Thierens. An adaptive pursuit strategy for allocating operator probabilities. Dans *Proceedings of the 2005 conference on Genetic and evolutionary computation*, GECCO '05, page 1539–1546, New York, NY, USA, 2005. ACM. (cité pages 39 et 40)

[Van Hentenryck et Michel, 2005] Pascal Van Hentenryck et Laurent Michel. *Constraint-Based Local Search*. The MIT Press, 2005. (cité pages 1, 44, 51, 62 et 104)

[Veerapen *et al.*, 2012a] Nadarajen Veerapen, Jorge Maturana, et Frédéric Saubion. A comparison of operator utility measures for on-line operator selection in local search. Dans *Learning and Intelligent Optimization, Proceedings Sixth International Conference on Learning and Intelligent Optimization (LION6)*, volume 7219 of *Lecture Notes in Computer Science*, pages 497–502, Paris, France, 2012. Springer Berlin / Heidelberg. (cité page 90)

[Veerapen *et al.*, 2012b] Nadarajen Veerapen, Jorge Maturana, et Frédéric Saubion. An exploration-exploitation compromise-based adaptive operator selection for local search. Dans *Proceedings of the fourteenth international conference on Genetic and evolutionary computation conference*, GECCO '12, page 1277–1284, New York, NY, USA, 2012. ACM. (cité page 90)

[Veerapen *et al.*, 2012c] Nadarajen Veerapen, Jorge Maturana, et Frédéric Saubion. Sélection adaptative d'opérateurs pour la recherche locale basée sur un compromis exploration-exploitation. Dans *Actes des Huitièmes Journées Francophones de Programmation par Contraintes (JFPC 2012)*, pages 318–327, Toulouse, France, mai 2012. (cité page 90)

[Veerapen et Saubion, 2011a] Nadarajen Veerapen et Frédéric Saubion. Pareto autonomous local search. Dans Carlos A. Coello Coello, éditeur, *Learning and Intelligent Optimization, Proceedings Fifth International Conference on Learning and Intelligent Optimization (LION5)*, volume 6683 of *Lecture Notes on Computer Science*, pages 392–406. Springer, Heidelberg, 2011. (cité page 71)

[Veerapen et Saubion, 2011b] Nadarajen Veerapen et Frédéric Saubion. Sélection autonome d'opérateurs par dominance pour la recherche locale. Dans *Actes des Septièmes Journées Francophones de Programmation par Contraintes (JFPC 2011)*, pages 307–316, Lyon, France, juin 2011. (cité page 71)

[Voudouris, 1997] Christos Voudouris. *Guided Local Search for Combinatorial Optimization Problems*. Thèse de doctorat, University of Essex, Colchester, Royaume-Uni, 1997. (cité page 19)

[Walser, 1997] Joachim P. Walser. Solving linear pseudo-boolean constraint problems with local search. Dans *Proceedings of the fourteenth national conference on artificial intelligence and ninth conference on Innovative applications of artificial intelligence*, AAAI'97/IAAI'97, page 269–274. AAAI Press, 1997. (cité pages 61 et 62)

[Weinberger, 1990] E. Weinberger. Correlated and uncorrelated fitness landscapes and how to tell the difference. *Biological Cybernetics*, 63(5) :325–336, 1990. (cité page 13)

[Wolpert et Macready, 1997] David H. Wolpert et William G. Macready. No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, 1(1) :67—82, 1997. (cité page 24)

[Ågren *et al.*, 2009] Magnus Ågren, Pierre Flener, et Justin Pearson. Revisiting constraint-directed search. *Information and Computation*, 207(3) :438–457, mars 2009. (cité pages 62 et 104)

l'université nantes angers le mans
PÔLE DE RECHERCHE ET D'ENSEIGNEMENT SUPÉRIEUR

université angers

Thèse de Doctorat

Nadarajen VEERAPEN

Contrôle autonome d'opérateurs pour la recherche locale

Résumé
Au fil des années, un nombre croissant de méthodes de résolution ont été proposées afin de traiter des problèmes plus grands et plus complexes. Parmi ces méthodes, les métaheuristiques sont largement utilisées dans le monde académique et industriel afin de résoudre efficacement des problèmes d'optimisation et de satisfaction de contraintes. Toutefois la conception de métaheuristiques de plus en plus performantes produit souvent des systèmes fortement complexes dont l'utilisation demande une expertise non négligeable aussi bien du problème lui-même que de la façon de paramétrer la méthode de résolution. Concevoir des algorithmes de recherche autonomes est donc une question importante.
Cette thèse traite du problème de la gestion et de la sélection d'opérateurs dans le contexte de la recherche locale, au sein d'un contrôleur générique. Celui a pour but de pouvoir être réutilisé facilement pour traiter différents problèmes. Nous nous attachons donc à concevoir des méthodes simples et robustes. La sélection des opérateurs se base sur un apprentissage des performances antérieures de chaque opérateur afin de déterminer les opérateurs vraisemblablement les plus bénéfiques à chaque pas de la recherche. Pour effectuer ces choix, le contrôleur se base sur la capacité des opérateurs à améliorer la qualité des solutions ainsi que sur la faculté de produire des solutions qui diffèrent de celles déjà obtenues.
Les méthodes proposées sont testées sur différents problèmes théoriques et pratiques d'optimisation combinatoire et de satisfaction de contraintes. Les résultats obtenus montrent qu'il est possible d'obtenir des résultats corrects avec des méthodes simples. Les mécanismes adaptatifs proposés se révèlent robustes sur différents problèmes.

Mots clés
Recherche autonome, recherche locale, sélection adaptative d'opérateurs, optimisation combinatoire, satisfaction de contraintes.

Abstract
Over the course of the years, an increasing number of resolution methods have been proposed to deal with larger and more complex problems. Among those methods, metaheuristics are commonly used in academia and the industry to efficiently solve optimisation and constraint satisfaction problems. Nevertheless the design of increasingly efficient metaheuristics often leads to highly complex systems which require a non negligible amount of expert knowledge of the problem itself and of the parameterisation of the solving method. Designing automous search algorithms is thus an important topic.
This thesis deals with the problem of managing and selection operators in the context of local search, within a generic controller. The latter should be easily adapted to deal with different problems. We therefore focus on designing simple and robust methods. Operator selection is based on learning the past performance of each operator to determine which operators are likely to be the most beneficial at each step of the search. In order to carry out this selection, the controller uses information about the capacity of the operators to improve the quality of solutions as well as their propensity to produce solutions which differ from ones already obtained.
The proposed methods are tested on different theoretical problems and ones with practical applications. Both combinatorial optimisation and constraint satisfaction problems are considered. The results show that it is possible to obtain good results with simple methods. The proposed adaptive mecanisms are shown to be robust across different problems.

Key Words
Autonomous search, local search, adaptive operator selection, combinatorial optimisation, constraint satisfaction.