Optimisation multi-objectif par colonies de fourmis : cas des problèmes de sac à dos

Inès Alaya

THÈSE

en cotutelle entre

UNIVERSITE DE LA MANOUBA
ECOLE NATIONALE DES SCIENCES DE L'INFORMATIQUE

Et

L'UNIVERSITE CLAUDE BERNARD LYON1

Présentée en vue de l'obtention du diplôme de

DOCTEUR EN INFORMATIQUE

par

Inès Alaya

Optimisation multi-objectif par colonies de fourmis
Cas des problèmes de sac à dos

Sous la direction de

Khaled GHEDIRA et Christine SOLNON

Réalisée au sein de

et

Soutenu le : 5 mai 2009

Devant le jury composé de :

Président : Jean-Michel JOLION, Professeur, Université de Lyon
Rapporteur : Yves DEVILLE, Professeur, Université catholique de Louvain
Rapporteur : Béchir AYEB, Professeur, Faculté des sciences de Monastir
Examinateur : Patrick ALBERT, Directeur Scientifique, ILog
Directeurs de thèse : Khaled GHEDIRA, Professeur, Institut Supérieur de Gestion
Christine SOLNON, Maître de Conférences, Université Lyon1

Table des matières

Introduction Générale

Les problèmes d'optimisation NP-difficiles ne possédent pas, à ce jour, un algorithme général permettant de les résoudre en un temps polynomial. Ce problème de l'explosion combinatoire limite l'utilisation de méthodes exactes pour la résolution à des problèmes de petites tailles. Dans le cas de problèmes de grande taille, comme cela est souvent le cas dans les applications réelles, les méthodes incomplètes, qui sacrifient la complétude pour gagner l'efficacité, deviennent une alternative intéressante.

Ces dernières années, la plupart des métaheuristiques présentées dans la littérature pour résoudre des problèmes combinatoires sont d'inspiration biologique. Parmi ces métaheuristiques, nous pouvons citer les algorithmes génétiques, le recuit simulé, les réseaux de neurones, les algorithmes à estimation de distribution, l'optimisation par essaim de particules et l'optimisation par colonie de fourmis.

Dans cette thèse, nous nous intéressons à l'étude des capacités de la métaheuristique d'optimisation par colonie de fourmis ('Ant Colony Optimization' - ACO) [Dorigo & Di Caro, 1999, Moyson & Manderick, 1988, Dorigo & Stüzle, 2004]. Les algorithmes à base de colonies de fourmis ont été initialement proposés dans [Dorigo, 1992, Dorigo *et al.* , 1996]. Depuis son apparition, l'ACO requiert de plus en plus l'attention de la communauté scientifique vu le succès qu'elle a réalisé. Elle a été appliquée à plusieurs problèmes combinatoires comme le problème du voyageur de commerce [Dorigo & Gambardella, 1997], affectation quadratique [Gambardella

et al. , 1999a], routage de véhicules [Bullnheimer *et al.* , 1999], sac à dos multidimensionnel [Alaya *et al.* , 2004a],...

Nous nous sommes intéressés, dans cette thèse, à l'étude des capacités de cette métaheuristique, et nous avons tenté de dégager des pistes concernant le choix d'une stratégie phéromonale. Cette stratégie consiste à un mécanisme d'apprentissage utilisé par les algorithmes ACO pour permettre à la colonie de converger vers les solutions qui est basé sur le dépôt de traces de phéromone. Ces traces sont ensuite utilisées pour biaiser les décisions lors de la construction de solutions. Le choix d'une telle stratégie est à la fois déterminant et délicat en fonction des problèmes. Initialement, la métaheuristique ACO a été introduite pour résoudre un problème de recherche de chemin hamiltonien dans un graphe, i.e. voyageur de commerce. Dans ce cas, la stratégie phéromonale consiste à déposer la phéromone sur les chemins construits. Pour les problèmes de sélection de sous-ensemble comme par exemple les problèmes de sac à dos, de clique maximale ou de satisfaction de contraintes, il n'est pas évident de choisir une stratégie phéromonale appropriée. En effet, ces problèmes ne peuvent pas être ramenés, généralement, à des problèmes de recherche de chemins, puisque l'ordre dans lequel les composants de solutions sont sélectionnés n'est pas important. Nous nous sommes plus particulièrement intéressés au problème du sac-à-dos multidimensionnel qui est un problème classique de sélection de sous-ensemble, et qui a été largement étudié dans la littérature. Nous étendons cette étude par la suite aux cas des problèmes d'optimisation multi-objectif, où le choix d'une structure phéromonale est encore plus délicat.

Dans la plupart des problèmes du monde réel, il ne s'agit pas d'optimiser seulement un seul critère mais plutôt d'optimiser simultanément plusieurs critères et qui sont généralement conflictuels. L'optimisation multi-objectif consiste donc à optimiser simultanément plusieurs fonctions. La notion de solution optimale unique dans l'optimisation uni-objectif disparait pour les problèmes d'optimisation multi-objectif

au profit de la notion d'*ensemble de solutions Pareto optimales*.

L'utilisation des métaheuristiques pour résoudre des problèmes multi-objectifs a fait l'objet d'un intérêt de plus en plus croissant. A présent, les métaheuristiques constituent un des champs de recherche les plus actifs dans l'optimisation multi-objectifs. La plupart des travaux existants concernent l'optimisation bi-objectif. Le cas multi-objectif reste difficile à résoudre, non seulement à cause de la complexité de ces problèmes, mais aussi à cause du nombre élevé des solutions Pareto optimales à un problème d'optimisation multi-objectif.

Nous proposons, dans cette thèse, d'étudier les capacités de la métaheuristique ACO pour la résolution des problèmes d'optimisation multi-objectif. Pour ce faire, il est indispensable de choisir la stratégie phéromonale appropriée. De plus, pour ce genre de problèmes, le choix des structures phéromonales est un autre point clé dans la résolution. En effet, pour les problèmes mono-objectif, généralement une seule structure de phéromone est utilisée correspondant à l'unique l'objectif. Cependant, lorsqu'on a plusieurs objectifs à optimiser simultanément, il n'est plus évident comment définir les structures phéromonales, comment les associer aux fonctions objectifs, ni comment les exploiter lors de la construction de solutions.

Ce mémoire de thèse est organisé de la façon suivante. La première partie est consacrée à la présentation des problèmes d'optimisation combinatoire et l'étude des méthodes de résolution existantes. Nous introduisons dans le premier chapitre les problèmes d'optimisation combinatoire et les approches de résolution complètes et heuristiques pour le cas mono-objectif. Dans le deuxième chapitre, nous nous focalisons plus particulièrement sur l'optimisation par colonies de fourmis. Nous définissons dans le troisième chapitre les problèmes d'optimisation multi-objectif et nous présentons les principales approches de résolution proposées dans la litérature. Nous consacrons le quatrième chapitre à la description des approches ACO proposées dans la littérature pour résoudre des problèmes multi-objectifs. Nous proposons dans

ce chapitre une taxonomie qui classe ces différentes approches suivant plusieurs critères.

La deuxième partie du mémoire présente nos contributions. Nous proposons dans le cinquième chapitre une étude de différentes stratégies phéromonales qui porte sur le problème du sac à dos multidimensionnel mono-ojectif (en anglais Multidimensional Knapsack Problem : MKP). Nous proposons un algorithme fourmi générique permettant de mettre en évidence l'importance du choix de la stratégie phéromonale utilisée. Nous instancions cet algorithme en trois variantes qui présentent trois différentes stratégies phéromonales possibles. En effet, nous avons proposé dans le cadre du travail de mastère un algorithme fourmi pour la résolution du MKP utilisant une seule stratégie phéromonale [Alaya *et al.* , 2004a, Alaya *et al.* , 2004b]. Nous comparons dans cette partie cet algorithme avec d'autres stratégies phéromonales et nous montrons l'importance du choix de ces stratégies sur la qualité des solutions et le temps d'exécution. Nous montrons aussi l'influence des paramètres de l'algorithme fourmi sur la qualité des solutions construites ainsi que sur la similarité de ces solutions.

Nous pensons que cette étude expérimentale de différentes stratégies phéromonales dépasse le cadre du MKP, et peut être utile pour la résolution d'autres problèmes de "recherche d'un meilleur sous-ensemble" à l'aide de la métaheuristique ACO. De plus, cette étude va nous servir de base, pour le cas multi-objectif, dans le choix de la stratégie phéromonale, et aussi dans le choix des valeurs des paramètres de l'algorithme ACO. L'essentiel de ces travaux a été publié dans [Alaya *et al.* , 2007b, Alaya *et al.* , 2005].

Les sixièmes et septièmes chapitres regroupent les contributions les plus fondamentales de ce travail. Nous proposons, dans le sixième chapitre, un algorithme générique basé sur l'optimisation par colonies de fourmis pour résoudre des problèmes d'optimisation multi-objectif. Cet algorithme est paramétré par le nombre

de colonies de fourmis et le nombre de structures de phéromone considérées. Cet algorithme permet de tester et de comparer, dans un même cadre, plusieurs approches nouvelles et existantes. Ainsi nous pouvons comparer ces approches indépendamment du schéma ACO considéré et des détails d'implémentation. Une partie des travaux présentés dans ce chapitre a été publiée dans [Alaya *et al.* , 2007a]. Dans le septième chapitre, nous appliquons et testons les différentes variantes de l'algorithme générique sur le problème du sac à dos multidimensionnel multi-objectif.

Première partie

Optimisation combinatoire

Chapitre 1

Introduction

1.1 Introduction

Nous définissons dans ce chapitre les problèmes d'optimisation combinatoire et les éléments de base relatifs ainsi que quelques exemples de ces problèmes. Nous présentons par la suite les principales approches de résolution complètes et heuistiques.

1.2 Définition

On qualifie généralement de « combinatoires » les problèmes dont la résolution se ramène à l'examen d'un nombre fini de combinaisons. Bien souvent cette résolution se heurte à une explosion du nombre de combinaisons à explorer.

Un problème d'optimisation combinatoire peut être défini comme suit :

Etant donné un ensemble S de combinaisons, et une fonction $f : S \rightarrow IR$, il s'agit de trouver la combinaison de S minimisant f, i.e., $s^* \in S$,
$f(s^*) \leq f(s_i), \forall s_i \in S$.

Il est à noter que pour les problèmes de maximisation, il suffit de multiplier la fonction coût par -1. Selon le contexte, f est appelée la fonction de coût, la fonction

économique ou la fonction objectif,...

L'optimisation combinatoire trouve des applications dans des domaines aussi variés que la gestion, l'ingénierie, la conception, la production, les télécommunications, les transports, l'énergie, les sciences sociales et l'informatique elle-même.

1.3 Complexité théorique d'un problème

On entend ici par «complexité d'un problème»[1] une estimation du nombre d'instructions à exécuter pour résoudre les instances de ce problème, cette estimation étant un ordre de grandeur par rapport à la taille de l'instance. Il s'agit là d'une estimation dans le pire des cas dans le sens où la complexité d'un problème est définie en considérant son instance la plus difficile. Les travaux théoriques dans ce domaine ont permis d'identifier différentes classes de problèmes en fonction de la complexité de leur résolution [Papadimitriou, 1994]. En effet, il existe un très grand nombre de classes différentes, et on se limitera ici à une présentation succincte nous permettant de caractériser formellement la notion de problème combinatoire.

Problèmes de décision. Les classes de complexité ont été introduites pour les problèmes de décision, c'est-à-dire les problèmes posant une question dont la réponse est « oui » ou « non ». Pour ces problèmes, on définit notamment les deux classes $\mathcal{P}$ et $\mathcal{NP}$:

- La classe $\mathcal{P}$ contient l'ensemble des problèmes polynomiaux, i.e., pouvant être résolus par un algorithme de complexité polynomiale. Cette classe caractérise l'ensemble des problèmes que l'on peut résoudre « efficacement ».
- la classe $\mathcal{NP}$ contient l'ensemble des problèmes polynomiaux non déterministes, i.e., pouvant être résolus par un algorithme de complexité polynomiale

[1] Cette section a été inspirée du cours de mastère de Christine Solnon disponible à l'url http ://www710.univ-lyon1.fr/ csolnon/

pour une machine non déterministe (que l'on peut voir comme une machine capable d'exécuter en parallèle un nombre fini d'alternatives). Intuitivement, cela signifie que la résolution des problèmes de $\mathcal{NP}$ peut nécessiter l'examen d'un grand nombre (éventuellement exponentiel) de cas, mais que l'examen de chaque cas doit pouvoir être fait en un temps polynomial.

Les relations d'inclusion entre les classes $\mathcal{P}$ et $\mathcal{NP}$ sont à l'origine d'une très célèbre conjecture que l'on peut résumer par «$\mathcal{P} \neq \mathcal{NP}$ ». En effet, si la classe $\mathcal{P}$ est manifestement inclue dans la classe $\mathcal{NP}$, la relation inverse n'a jamais été ni montrée ni infirmée.

Cependant, certains problèmes de $\mathcal{NP}$ apparaissent plus difficiles à résoudre dans le sens où l'on ne trouve pas d'algorithme polynomial pour les résoudre avec une machine déterministe. Les problèmes les plus difficiles de $\mathcal{NP}$ définissent la classe des problèmes $\mathcal{NP}$-complets : un problème de $\mathcal{NP}$ est $\mathcal{NP}$-complet s'il est au moins aussi difficile à résoudre que n'importe quel autre problème de $\mathcal{NP}$, i.e., si n'importe quel autre problème de $\mathcal{NP}$ peut être transformé en ce problème par une procédure polynomiale.

Ainsi, les problèmes $\mathcal{NP}$-complets sont des problèmes combinatoires dans le sens où leur résolution implique l'examen d'un nombre exponentiel de cas. Notons que cette classe des problèmes $\mathcal{NP}$-complets contient un très grand nombre de problèmes (dont quelques-uns sont décrits dans la section 4). Pour autant, tous les problèmes combinatoires n'appartiennent pas à cette classe. En effet, pour qu'un problème soit $\mathcal{NP}$-complet, il faut qu'il soit dans la classe $\mathcal{NP}$, i.e., que l'examen de chaque cas puisse être réalisé efficacement, par une procédure polynomiale. Si on enlève cette contrainte d'appartenance à la classe $\mathcal{NP}$, on obtient la classe plus générale des problèmes $\mathcal{NP}$-difficiles, contenant l'ensemble des problèmes qui sont « au moins aussi difficiles » que n'importe quel problème de $\mathcal{NP}$, sans nécessairement appartenir à $\mathcal{NP}$.

Problèmes d'optimisation. A chaque problème d'optimisation on peut associer un problème de décision dont le but est de déterminer s'il existe une solution pour laquelle la fonction objectif soit supérieure (resp. inférieure) ou égale à une valeur donnée. La complexité d'un problème d'optimisation est liée à celle du problème de décision qui lui est associé. En particulier, si le problème de décision est $\mathcal{NP}$-complet, alors le problème d'optimisation est dit $\mathcal{NP}$-difficile.

1.4 Modélisation

Parmi les principaux outils de modélisation uilisés pour la résolution des problèmes d'optimisation combinatoire, nous trouvons la théorie des graphes. En effet, la modélisation par les graphes est naturelle dans certains problèmes (plus court chemin,...).

Les problèmes d'optimisation combinatoire peuvent aussi se formuler comme des programmes linéaires en nombres entiers où les variables sont astreintes à prendre des valeurs entières. Les problèmes de type sac à dos peuvent être représentés par un programme linéaire en nombres entiers, i.e., un ensemble de contraintes linéaires et une fonction objectif linéaire portant sur des variables à valeurs entières.

1.5 Exemples de problèmes d'optimisation combinatoire

1.5.1 Le problème du voyageur de commerce

Dans ce problème, il s'agit de trouver le chemin le plus court reliant n villes données, chaque ville ne devant être visitée qu'une et une seule fois, et revenir à la ville de départ. La difficulté du problème vient de l'explosion combinatoire du

nombre de chemins à explorer lorque l'on accroît le nombre de villes à visiter.

Plus formellement, ce problème peut être modélisé comme suit : Etant donné un graphe non orienté complet $G = (S, A)$, et une fonction de distance $d : A \rightarrow IR^+$, on cherche à déterminer un cycle hamiltonien [2] de distance totale minimale.

Pour ce problème, trouver s'il existe un chemin hamiltonien est un problème $\mathcal{NP}$-complet, la recherche du plus court chemin hamiltonien est un problème $\mathcal{NP}$-difficile.

1.5.2 Le problème du sac à dos multidimensionnel

Le problème du sac à dos multidimensionnel (Multidimensional Knapsack Problem : MKP) est un problème d'optimisation combinatoire sous contraintes NP-difficile. Plusieurs problèmes pratiques peuvent être formalisés comme un MKP. Nous citons l'allocation des processeurs et des bases de données dans les systèmes informatiques distribués, le chargement des cargaisons, les problèmes de découpage de stock, etc.

Le MKP permet de modéliser une grande variété de problèmes où il s'agit de maximiser un profit tout en ne disposant que de ressources limitées. De manière formelle, il est présenté comme suit :

Soit

- m le nombre de ressources
- n le nombre d'objets
- p_j (j $\in$ 1..n) le profit apporté par l'objet j
- r_{ij} (i $\in$ 1..m, j $\in$ 1..n) la consommation de la ressource i par l'objet j
- b_i (i $\in$ 1..m) la quantité totale disponible de la ressource i.

On définit le MKP(n, m) par :

[2] un cycle qui passe une et une seule fois par chaque sommet du graphe

$$MKP \left\{ \begin{array}{l} maximiser \sum_{j=1}^{n} p_j.x_j \\ tq \sum_{j=1}^{n} r_{ij}.x_j \leq b_i, i \in 1..m \\ \qquad x_j \in \{0,1\}^n, \text{j} \in 1..n \end{array} \right\}$$

x_j (j $\in$ 1..n) est une variable de décision qui peut prendre pour valeur 0 (si l'objet j n'est pas sélectionné) ou 1 (s'il est sélectionné).

1.5.3 Le problème du sac à dos quadratique

Le problème du sac à dos quadratique, Quadratic Knapsack Problem (QKP), consiste à sélectionner un sous-ensemble d'objets dont le poids total ne dépasse pas une capacité donnée c du sac à dos et de maximiser le profit total. Plus formellement, le QKP est défini comme suit : Supposons $N = \{1..n\}$ un ensemble d'objets donné où chaque objet $j \in N$ a un poids positif w_j. De plus nous disposons d'une matrice d'ordre n d'entiers non négatifs $P = \{p_{ij}\}$, où le p_{ij} est un profit accompli en sélectionnant deux objets différents i et $j \in N$. En outre, le profit p_{ii} est accompli si l'objet $i \in N$ est choisi. Des variables binaires x_j seront introduites et indiqueront si l'objet $j \in N$ est sélectionné. Ce problème peut être formulé comme suit :

$$\begin{array}{rl} \text{Maximiser} & \sum_{i \in N} \sum_{j \in N} p_{ij} x_i x_j \\ \text{tel que} & \sum_{j \in N} w_j x_j \leq c \\ & x_j \in \{0,1\}, j \in N \end{array}$$

Ce problème peut être considéré comme une variante du MKP, où on a une seule dimension, et où la fonction objectif est quadratique et non plus linéaire.

1.6 Approches pour la résolution de problèmes d'optimisation combinatoire

La majorité des problèmes d'optimisation combinatoire, sont des problèmes NP-difficiles et donc ne possèdent pas à ce jour un algorithme efficace, i.e ; de complexité polynomiale, valable pour toutes les données. Ceci a motivé les chercheurs à développer de nombreuses méthodes de résolution en recherche opérationnelle et en intelligence artificielle. Ces méthodes appartiennent à deux grandes familles : les méthodes complètes et les méthodes approchées.

Nous présentons dans ce qui suit les principales méthodes complètes et heuristiques connues dans la littérature.

1.7 Les approches complètes

Le principe essentiel d'une méthode exacte consiste généralement à énumérer, souvent de manière implicite, l'ensemble des combinaisons de l'espace de recherche. Parmi les méthodes exactes, nous trouvons les méthodes de séparation et évaluation, dites méthodes de Branch and Bound, et la programmation dynamique.

1.7.1 Branch and bound

Le Branch and Bound est une technique qui effectue un parcours en profondeur de l'arbre de recherche afin de fournir une ou plusieurs solutions optimales à partir d'un ensemble de solutions potentielles. A chaque étape de la recherche, correspondant à un noeud de l'arbre de recherche, l'algorithme utilise une fonction Bound pour calculer une borne de l'ensemble des solutions du sous-arbre prenant sa racine à ce noeud. En début de résolution, cette borne est initialisée à une valeur maximale (en cas de minimisation). Si cette évaluation est moins bonne que la meilleure solution

trouvée jusqu'à ce niveau de recherche, tout le sous-arbre peut être coupé.

Il importe de souligner que l'efficacité de l'algorithme Branch and Bound dépend étroitement du calcul de la borne utilisée.

1.7.2 Programmation dynamique

La programmation dynamique est une méthode ascendante : On commence d'habitude par les sous problèmes les plus petits et on on remonte vers les sous problèmes de plus en plus difficiles. Elle est utilisée pour les problèmes qui satisfont au principe d'optimalité de Bellman : "Dans une séquence optimale (de décisions ou de choix), chaque sous-séquence doit aussi être optimale". Un exemple de ce type de problème est le plus court chemin entre deux sommets d'un graphe.

L'idée de base est d'éviter de calculer deux fois la même chose, généralement en utilisant une table de résultats déjà calculés, remplie au fur et à mesure qu'on résout les sous problèmes.

Il est à noter que la programmation dynamique est utilisée pour résoudre des problèmes polynomiaux (et non $\mathcal{NP}$-difficiles).

1.8 Les approches heuristiques

Nous distinguons parmi les approches heuristiques les approches par voisinage et les approches par construction. Les approches par voisinage partent d'une ou plusieurs solutions initiales qu'elles cherchent à améliorer à partir d'un voisinage défini, alors que les approches constructives partent d'une solution vide et génèrent de nouvelles solutions de façon incrémentale en ajoutant itérativement des composants aux solutions en cours de construction.

1.8.1 Approches par voisinage

L'idée de ces approches est de structurer l'ensemble des solutions en terme de voisinage, et d'explorer cet ensemble en partant d'une ou plusieurs solutions initiales et en choisissant à chaque itération une ou plusieurs solutions voisines d'une ou plusieurs solutions courantes. Le voisinage d'une solution S est l'ensemble des solutions qui peuvent être atteintes à partir de S en un seul "mouvement", un mouvement étant par exemple, le changement de valeur d'une variable.

Un algorithme de recherche locale glouton cherche dans le voisinage une solution qui améliore la qualité de la solution courante. Si une telle solution est trouvée, elle remplace la solution courante et la recherche locale continue. Ces étapes sont répétées jusqu'à ce qu'aucune solution meilleure n'est trouvée dans le voisinage de la solution courante et l'algorithme termine avec un optimum local. L'optimum local est la meilleure solution parmi les solutions du voisinage, contrairement à l'optimum global qui est la meilleure solution parmi toutes les solutions possibles.

Plusieurs améliorations à cet algorithme glouton ont été proposées dans la littérature pour essayer d'échapper aux optima locaux. Nous présentons, dans ce qui suit, quelques unes de ces améliorations comme la recherche tabou, le recuit simulé, les algorithmes génétiques et l'optimisation par essaim particulaire.

Lors de l'utilisation de ces approches heuristiques, un point critique est de trouver un équilibre entre l'exploitation de l'expérience de recherche (la notion d'intensifcation) et l'exploration de nouvelles zones de l'espace de recherche non encore visitées (la diversification). Une bonne recherche doit être capable de trouver un bon compromis entre ces deux notions duales.

Recherche Taboue

La Recherche Taboue est une méthode heuristique d'optimisation combinatoire développée par Glover [Glover, 1986]. La méthode Taboue effectue des mouvements

d'une solution S à une meilleure solution S' appartenant au voisinage de S noté N(S). Si ce voisinage est trop grand, seulement une partie $V(S) \subseteq N(S)$ sera examinée.

Pour éviter de rester bloqué dans le premier minimum local rencontré, la méthode Tabou accepte des solutions voisines dont la qualité est moins bonne que la solution courante. Elle se dirige alors vers la solution voisine qui dégrade le moins possible la fonction objectif. Cette dégradation de f, que l'on espère être temporaire, devrait permettre à l'algorithme d'atteindre des régions de l'espace contenant des solutions meilleures que celles rencontrées jusque-là. Cependant, l'algorithme Tabou risque de boucler en revisitant des solutions antérieures pendant l'itération suivante (ou après quelques itérations intermédiaires).

Pour éviter ce phénomène, l'algorithme utilise une structure de mémoire dans laquelle il stocke les mouvements interdits à chaque étape afin de ne pas obtenir une solution déjà rencontrée récemment. Ces mouvements interdits à une itération donnée sont dits tabous, d'où le nom attribué à la méthode par Glover [Glover, 1986]. Le caractère tabou d'un mouvement doit être temporaire afin de donner une plus grande flexibilité à l'algorithme en lui permettant de remettre en question les choix effectués, une fois que les risques de bouclage ont été écartés. La façon la plus simple de mettre en oeuvre ce système d'interdictions consiste à garder en mémoire les derniers mouvements effectués et à empêcher l'algorithme de faire les mouvements inverses à ces derniers mouvements. Ces mouvements sont mémorisés dans une liste T, appelée liste taboue. La gestion de cette liste se fait de manière circulaire en remplaçant à chaque itération le mouvement le plus ancien de la liste par le mouvement courant.

Cependant, il peut se trouver des cas où il est intéressant de révoquer le statut tabou d'un mouvement lorsque son application à la solution courante permet d'atteindre une solution meilleure. La mise en oeuvre d'un tel mécanisme est réalisée à l'aide d'une fonction auxiliaire A qui est appelée fonction d'aspiration.

Dans cette méthode, le compromis entre les mécanismes d'intensification/diversification peut être géré par la taille de la liste taboue. Pour favoriser l'intensification de la recherche il suffit de diminuer la taille de la liste tabou, et contrairement, pour favoriser la diversification il s'agit d'augmenter la taille de cette liste.

Recuit simulé

La méthode du Recuit Simulé repose sur une analogie avec la thermodynamique. En effet, la 'recherche' par un système physique des états d'énergie les plus bas est l'analogue formel d'un processus d'optimisation combinatoire.

Le recuit est un processus physique de chauffage. Un système physique porté à une température assez élevée devient liquide dans ce cas le degré de liberté des atomes qui le composent augmente. Inversement lorsque l'on baisse la température le degré de liberté diminue jusqu'à obtenir un solide. Suivant la façon dont on diminue la température on obtient des configurations d'énergie différentes :

- Baisse brutale de la température, la configuration atteinte est le plus souvent un état métastable, dont l'énergie est supérieure à celle du minimum absolu. Le système est en quelque sorte piégé dans ce minimum local.
- Baisse progressive de la température de façon à éviter de piéger le système dans des vallées d'énergie élevée, pour l'envoyer vers les bassins les plus importants et les plus profonds, là où les baisses ultérieures de température le précipiteront vers les fonds.

La méthode d'optimisation du recuit simulé a été proposée en 1982 par S. Kirkpatrick et al. [Kirkpatrick *et al.* , 1983] à partir de la méthode de Metropolis et al. [Metropolis *et al.* , 1953] qui était utilisée pour modéliser les processus physiques. Kirkpatrick s'est inspiré de la mécanique statistique en utilisant en particulier la distribution de Bolzmann. Le recuit simulé permet de sortir d'un minimum local en acceptant avec une certaine probabilité une dégradation de la fonction. Ainsi, la

probabilité p qu'un système physique passe d'un niveau d'énergie E_1à un niveau E_2 est donnée par :

$$p = e^{-\frac{E_2 - E_1}{k_b \cdot T}} \tag{1.1}$$

k_b est la constante de Bolzmann

T est la température du système

Comme le montre cette formule la probabilité d'observer une augmentation de l'énergie est d'autant plus grande que la température est élevée, donc au niveau du recuit simulé :

- Une diminution de la fonction sera toujours acceptée ;
- Une augmentation de la fonction sera acceptée avec une probabilité définie selon une formule du type 1.1.

Dans la méthode du recuit simulé, les mécanismes d'intensification et de diversification sont contrôlés par la température. En effet, la température décroît au fil du temps de sorte que la recherche tend à s'intensifier vers la fin de l'algorithme.

Les algorithmes génétiques

Les algorithmes génétiques sont des méthodes évolutionnistes qui s'inspirent fortement des mécanismes biologiques liés aux principes de sélection et d'évolution naturelle. Développés initialement par Holland et al. [Holland, 1992] pour répondre à des besoins spécifiques en biologie, les algorithmes génétiques ont rapidement été adaptés à des contextes très variés. Dans un algorithme génétique simple, un individu (une solution) est caractérisé par une structure de données qui représente son code génétique. La force de ce dernier, appelée fitness, peut être mesurée par la valeur de la fonction objectif correspondante. La recherche est réglée par trois opérateurs qui sont appliqués successivement. La phase de coopération est gouvernée par un opérateur de sélection et un opérateur de croisement (ou crossover) alors que

la phase d'adaptation individuelle fait appel à un opérateur de mutation.

La création d'une nouvelle génération est obtenue par itération de l'algorithme génétique qui va créer de nouveaux individus et en détruire d'autres (mécanisme de sélection naturelle) ce qui permet le renouvellement de la population (l'ensemble des solutions courantes).

La sélection La phase de sélection spécifie les individus de la population P qui sont amenés à se reproduire par croisement. La méthode de base, appelée roue de loterie attribue à chaque individu une probabilité de reproduction proportionnelle à son adaptation dans la population. L'espoir étant de faire reproduire les individus les plus forts.

Le Croisement Etant donné deux individus sélectionnés, l'opérateur de croisement crée deux nouveaux individus. Cet opérateur coupe les deux chaînes juxtaposées en un ou plusieurs points et produit deux nouvelles chaînes après avoir échangé les parties coupées. Cette opération symbolise dans l'algorithme génétique la reproduction.

La mutation Une mutation est un changement aléatoire d'un ou plusieurs bits de la chaîne codant l'individu. L'opérateur de croisement devient moins efficace avec le temps, car les individus deviennent similaires. C'est à ce moment que le phénomène de mutation prend toute son importance : ces mutations ne créent généralement pas de meilleures solutions au problème, mais elles évitent l'établissement de populations uniformes incapables d'évoluer.

Il est important de souligner que les concepts qui sont à la base des algorithmes génétiques sont extrêmement simples. En effet, ils font uniquement intervenir des nombres générés aléatoirement et un ensemble de règles probabilistes très générales qui ne tiennent pas forcément compte de toutes les particularités du problème traité.

L'optimisation par essaim particulaire

L'optimisation par essaim particulaire est une technique d'optimisation stochastique qui a été développée initialement en 1995 sous l'appelation anglaise de "Particle Swarm Optimization" (PSO) par Russel Eberhart et James Kennedy, inspirée par le comportement social des essaims d'oiseaux ou des bans de poissons.

PSO partage plusieurs similarités avec les algorithmes évolutionnaires comme les algorithmes génétiques (AGs). Le système est initialisé par une population de solutions aléatoires et cherche des optima à travers les générations. Cependant, contrairement aux AGs, PSO n'utilise aucun opérateur évolutionnaire comme le croisement ou la mutation. Dans PSO, les solutions potentielles, appelées particules, volent à travers l'espace du problème en suivant les particules optimales courantes.

En effet l'algorithme général de l'OEP peut être décrit de manière non formelle comme suit : Au départ, un essaim est réparti aléatoirement dans l'espace de recherche, chaque particule ayant également une vitesse aléatoire. Ensuite, à chaque pas de temps :

- Chaque particule est capable d'évaluer la qualité de sa position et de garder en mémoire sa meilleure performance, c'est-à-dire la meilleure position qu'elle a atteinte jusqu'ici et sa qualité.
- Chaque particule est capable d'interroger un certain nombre de ses congénères (ses informatrices, dont elle même) et d'obtenir de chacune d'entre elles sa propre meilleure performance (et la qualité afférente).
- chaque particule choisit la meilleure des meilleures performances dont elle a connaissance, modifie sa vitesse en fonction de cette information et de ses propres données et se déplace en conséquence.

Les informatrices peuvent être définies de la manière suivante : On suppose toutes les particules disposées (symboliquement) en cercle et, pour la particule étudiée, on inclut progressivement dans ses informatrices, d'abord elle même, puis les plus

proches à sa droite et à sa gauche, de façon à atteindre le total requis. Ces dernières années, l'OEP a été appliqué avec succès à plusieurs domaines de recherche et application.

1.8.2 Approches constructives

Les approches constructives commencent à partir d'une solution vide qu'elles construisent par incréments au fur et à mesure de la recherche. Nous commençons par définir les algorithmes gloutons qui présentent la méthode de base à partir de laquelle sont dérivés les algorithmes fourmi, par la suite nous présentons les algorithmes à estimation de distribution.

Algorithmes gloutons

Le principe d'un algorithme glouton (appelé en anglais greedy algorithm) est de partir d'une solution initiale vide et d'ajouter, d'une manière incrémentale, des composants de solutions sans remettre en cause les choix antérieurs jusqu'à obtenir une solution complète.

Dans le cas le plus simple, les composants de solutions sont ajoutés d'une manière aléatoire. De meilleurs résultats sont généralement obtenus en utilisant une heuristique du bénéfice qu'apporte le composant à ajouter, c'est ce qu'on appelle le critère gradient. Un inconvénient majeur de ces méthodes est que l'utilisation du critère gradient dans les premières étapes peut mener à des déplacements trés faibles dans les dernières phases de construction de solutions et engendrer des solutions de qualité médiocre.

La performance des algorithmes gloutons dépend étroitement de la pertinence de l'heuristique utilisée, i.e. leur capacité à exploiter les connaissances du problème.

Algorithmes à estimation de distribution

Les algorithmes à estimation de distribution (ou en anglais « Estimation of Distribution Algorithms », EDA) forment une famille de métaheuristiques inspirée aussi des algorithmes génétiques. Cependant, ils n'emploient pas d'opérateurs de croisement ni de mutation, mais plutôt ils sélectionnent les meilleures solutions de la population courante et extraient des informations statistiques globales des solutions extraites. Un modèle de probabilité de distribution des solutions prometteuses est construit en se basant sur les informations extraites. Par la suite, les solutions construites remplacent en partie ou totalement les solutions de la population courante. Plus précisement, l'algorithme général de EDA peut être décrit comme suit :

- Etape 0 : Prendre aléatoirement un ensemble de solutions pour construire la population initiale
- Etape 1 : Sélectionner quelques solutions de la population courante relativement à une méthode de sélection. Construire le modèle de probabilité des solutions sélectionnées. Utiliser le modèle probabiliste pour construire de nouvelles solutions selon un principe glouton.
- Etape 2 : Remplacer quelques ou tous les membres de la population courante par les nouvelles solutions construites à partir du modèle de probabilité.
- Etape 3 : Si la condition d'arrêt n'est pas satisfaite, aller à Etape 1.

1.9 Conclusion

Nous avons défini dans la première partie de ce chapitre les problèmes d'optimisation combinatoire mono-objectif ainsi que quelques exemples de ces problèmes. Nous avons présenté par la suite différentes approches pour la résolution de problèmes d'optimisation combinatoire. Nous avons commencé par présenter brièvement quelques méthodes exactes. Par la suite, nous avons décrit des approches heuris-

tiques en présentant des méthodes par voisinage et d'autres par construction. Nous consacrons le chapitre prochain pour la définition de l'optimisation par colonies de fourmis, qui appartient à la classe des approches par construction, et qui présente l'approche de base de notre travail.

Chapitre 2

Optimisation par colonies de fourmis

2.1 Introduction

La métaheuristique d'optimisation par colonies de fourmis a été initialement introduite par Dorigo, Maniezzo et Colorni [Dorigo, 1992, Dorigo *et al.* , 1996] et a été inspirée par les études sur le comportement des fourmis réelles effectuées par Deneubourg et al [Deneubourg *et al.* , 1983].

A l'origine, l'optimisation par colonie de fourmis a été conçue pour résoudre le problème du voyageur de commerce en proposant le premier algorithme ACO : 'Ant System' (AS) [Dorigo & Gambardella, 1997]. Par la suite, un nombre considérable d'applications de ACO a été proposé telles que l'affectation quadratique [Gambardella *et al.* , 1999a], le routage des véhicules [Bullnheimer *et al.* , 1999], le problème de satisfaction de contraintes [Solnon, 2002],...

Dans la section suivante nous expliquons l'analogie biologique à partir de laquelle ACO a été inspirée. Nous présentons par la suite l'algorithme 'Ant System, le premier algorithme fourmi proposé dans la littérature, ainsi que ses extensions. Nous définissons dans la section 4 la métaheuristique d'optimisation par colonies de fourmis. Nous décrivons par la suite quelques exemples d'hybridation de ACO avec

la recherche locale. Enfin, nous discutons dans la section 6 de l'influence de quelques paramètres sur l'intensification/diversification de la recherche.

2.2 Analogie biologique

La métaheuristique ACO a été inspirée, essentiellement, par les études sur le comportement des fourmis réelles effectuées par Deneubourg et al [Deneubourg *et al.* , 1983]. L'un des problèmes étudiés était de comprendre comment des insectes, comme les fourmis, peuvent trouver le chemin le plus court du nid à la source de nourriture et le chemin de retour. Il a été trouvé que le moyen utilisé pour communiquer l'information entre les fourmis qui cherchent des chemins, est le dépôt de traces de phéromone,i.e., une substance chimique que les fourmis arrivent à détecter.

En se déplaçant, une fourmi dépose de la phéromone marquant ainsi le chemin par une trace de cette substance. Tandis qu'en abscence de traces une fourmi se déplace aléatoirement, une fourmi qui rencontre une trace de phéromone déjà déposée peut la détecter et décider de la suivre avec une probabilité proportionnelle à l'intensité de la trace, et renforce ainsi cette trace avec sa propre phéromone.

Le comportement collectif émerge d'un processus autocatalytique où plus les fourmis suivent une trace, plus cette trace devient attirante : c'est le principe de stigmergie.

Ce processus peut être illustré par l'exemple de la figure 2.1. Dans la figure 2.1.a, des fourmis se déplacent dans un réseau qui connecte une source de nourriture (A) à un nid (E). Supposons que les fourmis prennent une unité de temps pour une distance de longueur 1 et que à chaque unité de temps, 30 fourmis sortent du nid. Les premières fourmis qui arrivent à la position D doivent choisir d'aller vers C (avec une longueur totale du chemin de 3) ou vers H (avec une longueur totale du chemin de 4). Supposons qu'à l'instant t=0, 30 fourmis sont au point D. Comme

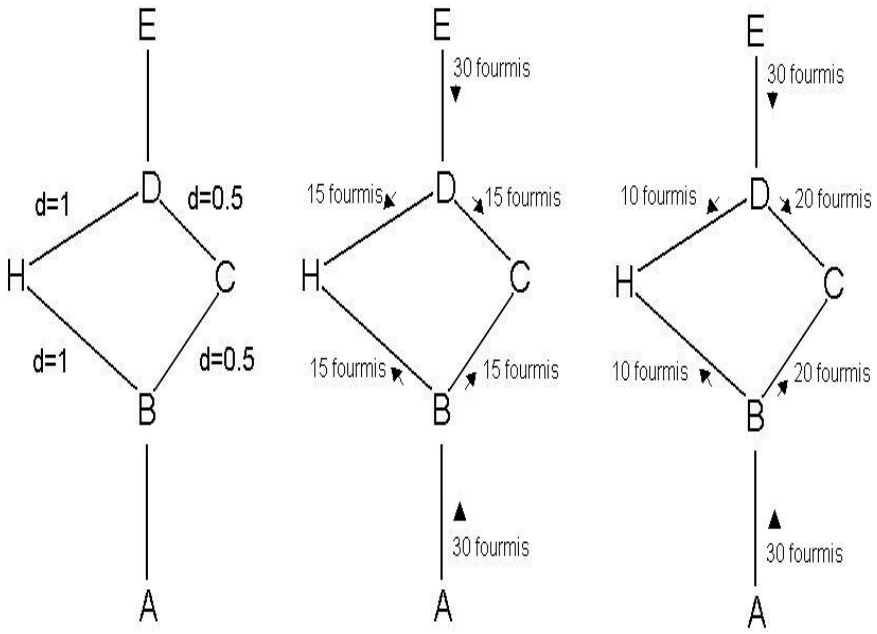

FIG. 2.1 – Comment les fourmis trouvent le plus court chemin

il n'existe pas de trace de phéromone sur les deux chemins, elles vont choisir une des deux alternatives avec la même probabilité. Donc environ 15 fourmis décident d'aller à C et les 15 autres fourmis décident d'aller à H. A t=1, les 30 prochaines fourmis sortant de B (et de D) peuvent détecter les traces de phéromone (figure 2.1.c). La trace de phéromone sur BCD est deux fois plus intense que celle sur BHD, car 30 fourmis sont passées par BCD (15 de A et 15 de D), tandis que 15 fourmis seulement sont passées par BHD. C'est pourquoi maintenant 20 fourmis prennent BCD et 10 prennent BHD (de même 20 prennent DCB et 10 prennent DHB à partir du point D). Une fois encore, plus de phéromone est déposée sur le plus court chemin. Ce processus est répété à chaque unité de temps et ainsi les traces de phéromone sont renforcées. Grâce à ce processus autocatalytique, toutes les fourmis vont trés rapidement choisir le chemin le plus court.

Pour transposer ce comportement à un algorithme général d'optimisation combinatoire, on fait une analogie entre l'environnement dans lequel les fourmis cherchent de la nourriture et l'ensemble des solutions admissibles du problème(i.e., l'espace de recherche du problème), entre la quantité ou la qualité de la nourriture et la fonction objectif à optimiser et enfin entre les traces et une mémoire adaptative.

Les fourmis artificielles dans les algorithmes ACO se comportent de la même manière. Elles diffèrent des fourmis naturelles dans le fait qu'elles ont une sorte de mémoire, pour assurer la génération de solutions faisables. En plus, elles ne sont pas complètement aveugles, i.e. elles ont des informations sur leur environnement.

Nous allons décrire, dans ce qui suit, le développement historique de l'algorithme Ant System appliqué au problème de voyageur de commerce, ensuite nous définissons de façon plus générique la métaheuristique ACO.

2.3 Optimisation par colonie de fourmis et problème de voyageur de commerce

Le problème de voyageur de commerce (PVC) a fait l'objet de la première implémentation d'un algorithme de colonie de fourmis : le " Ant System " [Dorigo, 1992].

2.3.1 Ant System

Dans l'algorithme Ant System (AS), chaque fourmi est initialement placée sur une ville choisie aléatoirement, chacune possède une mémoire qui stocke la solution partielle qu'elle a construite auparavant. Initialement, la mémoire contient la ville de départ. Commençant à partir de cette ville, une fourmi se déplace itérativement d'une ville à une autre. Quand elle est à une ville i, une fourmi k choisit d'aller à

une ville non encore visitée j avec une probabilité donnée par :

$$p_{ij}^{k}(t) \;=\; \frac{[\tau_{ij}\;(t)]^{\alpha} \times\; [\,\eta_{ij}]^{\beta}}{\sum\limits_{y\;\in\;N_i^k}\Big([\tau_{iy}\;(t)]^{\alpha}\;\times\;[\,\eta_{iy}]^{\beta}\Big)}\;si\;\;j \in N_i^k\;\;;0\;\;sinon \qquad (2.1)$$

- τ_{ij} (t) est l'intensité de la trace de phéromone dans l'arête (i, j) à l'instant t,
- $\eta_{ij} = \frac{1}{d_{ij}}$ est une information heuristique à priori valable, où d_{ij} est la distance entre la ville i et la ville j ; l'idée étant d'attirer les fourmis vers les villes les plus proches,
- α, β sont deux paramètres qui déterminent l'influence relative de la trace de phéromone et de l'information heuristique,
- N_i^k est le voisinage faisable de la fourmi k c'est à dire l'ensemble des villes non encore visitées par la fourmi k.

La construction de solution se termine après que chaque fourmi ait complété un tour. Ensuite, les traces de phéromone sont mises à jour. Dans AS, la mise à jour se fait, d'abord, en réduisant les traces de phéromone avec un facteur constant ρ (c'est l'évaporation de phéromone) et, ensuite, en permettant à chaque fourmi de déposer de la phéromone sur les arêtes qui appartiennent à son tour. Ainsi la formule de mise à jour de phéromone est comme suit :

$$\tau_{ij}(t+1) = (1-\rho) \times \tau_{ij}(t) + \sum_{k=1}^{nbAnts} \Delta\tau_{ij}^{k} \qquad (2.2)$$

Avec $0 \leq \rho \leq 1$ et nbAnts le nombre de fourmis.

Le paramètre ρ est ainsi utilisé pour éviter l'accumulation illimitée de phéromone et permet à l'algorithme d'oublier les mauvaises décisions précédemment prises. Sur les arêtes qui n'ont pas été choisis par les fourmis la force associée va décroître rapidement avec le nombre d'itérations.

$\Delta\tau_{ij}^{k}$ est la quantité de phéromone que la fourmi k dépose sur l'arête (i,j).

Il est défini par :

$$\Delta\tau_{ij}^{k} = \left\{ \begin{array}{cc} \frac{Q}{L^k} & si\ (i,j) \in Tabou^k \\ 0 & sinon \end{array} \right\} \tag{2.3}$$

où L^k est la longueur du tour généré par la fourmi k, Q une constante de l'algorithme et $Tabou^k$ est la liste des villes déjà visitées.

Avec cette formule, les arêtes du tour le plus court recevront la plus grande quantité de phéromone. En général, les arêtes qui sont utilisées par plusieurs fourmis et qui appartiennent aux tours les plus courts recevront plus de phéromone et en conséquence seront plus favorisés dans les itérations futures de l'algorithme.

2.3.2 Extensions de Ant System

Malgré ses résultats encourageants, AS n'était pas compétitif avec les algorithmes de l'état de l'art du PVC. Des recherches sont alors entreprises pour l'étendre et essayer d'améliorer ses performances en contrôlant mieux l'intensification et la diversification de la recherche.

Rank-based Ant system (AS-rank)

C'est une extension proposée par Bullnheimer, Hartl, and Strauss [Bullnheimer *et al.* , 1999]. Cet algorithme trie les fourmis selon les longueurs de tours générés. Après chaque phase de construction de tours, seulement les w meilleures fourmis sont autorisées à la mise à jour de phéromone. Ainsi, la $r^{ième}$ fourmi contribue à la mise à jour de phéromone avec un poids donné par le max {0, w - r} tandis que la fourmi correspondant au meilleur tour global renforce la trace de phéromone avec un poids w. L'objectif de cette version est d'intennsifier plus la recherche vers les meilleures solutions.

Ant Colony System

L'algorithme "Ant Colony System" (ACS) [Dorigo & Gambardella, 1997] a été introduit par Dorigo et Gambardella pour améliorer les performances du premier algorithme sur des problèmes de grandes tailles.

ACS introduit une règle qui dépend d'un paramètre q_0 ($0 \leq q_0 \leq 1$), qui définit une balance *diversification/intensification*. Cette règle permet aux fourmis de favoriser plus, lors de leurs déplacements, l'arête qui contient le maximum de traces suivant q_0, sinon la règle de transition habituelle est utilisée. Ainsi ACS permet d'intensifier plus la recherche vers les zones les plus prometteuses, i.e., qui contiennent plus de traces.

$\mathcal{MAX} - \mathcal{MIN}$ *Ant Sytem* (MM**AS**)

Stützle et Hoos ont introduit l'algorithme MMAS [Stützle & Hoos, 2000] qui, pour améliorer les performances de ACO, combine une exploitation améliorée des meilleures solutions trouvées durant la recherche avec un mécanisme efficace pour éviter une stagnation prématurée de la recherche. MMAS diffère en trois aspects clé de AS :

- Pour exploiter les meilleures solutions trouvées durant une itération ou durant l'exécution de l'algorithme, après chaque itération, une seule fourmi ajoute de la phéromone. Cette fourmi peut être celle qui a trouvé la meilleure solution depuis le début de l'exécution, ou bien celle durant le dernier cycle.
- Pour éviter une stagnation de la recherche, les valeurs possibles de la trace de phéromone sur chaque composant de solution sont limitées à l'intervalle $[\tau_{min},\tau_{max}]$.
- De plus, les traces de phéromone sont initialisées à τ_{max} pour assurer une exploration élevée de l'espace des solutions au début de l'algorithme.

2.4 La métaheuristique d'optimisation par colonie de fourmis

En utilisant la métaheuristique ACO, l'une des tâches principales est d'encoder le problème à résoudre sous la forme d'un problème de recherche d'un chemin optimal dans un graphe, appelé graphe de construction, où la faisabilité est définie en respectant un ensemble de contraintes. Les fourmis artificielles peuvent être caractérisées comme des procédures de construction gloutonnes stochastiques qui construisent des solutions en se déplaçant dans le graphe de construction.

Dans cette section, nous définissons d'abord une représentation générique du problème que les fourmis exploitent pour construire des solutions. Ensuite, nous détaillons le comportement des fourmis durant la construction de solution et finalement nous définissons la métaheuristique d'optimisation par colonie de fourmis.

2.4.1 Représentation du problème

Considérons le problème de minimisation (S, f, Ω) où S est l'ensemble des solutions candidates, f la fonction objectif qui associe à chaque solution candidate s $\in$ S une valeur de la fonction objectif f(s) et Ω est l'ensemble des contraintes. L'objectif est de trouver une solution optimale globale $s_{opt} \in$ S qui est une solution minimisant f et qui satisfait les contraintes Ω. Les différents états du problème sont caractérisés comme une séquence de composants.

Les fourmis artificielles construisent les solutions en se déplaçant sur le graphe construit G(C,L) tel que les sommets sont les composants C et L est l'ensemble qui connecte les composants de C (un élément de L est une connexion). Les contraintes du problème sont implémentées directement dans les règles de déplacement des fourmis (soit en empêchant les mouvements qui violent les contraintes, soit en pénalisant de telles solutions).

2.4.2 Comportement des fourmis

Les fourmis peuvent être caractérisées comme des procédures de construction stochastiques qui construisent des solutions en se déplaçant sur le graphe construit $G(C, L)$. Les fourmis ne se déplacent pas arbitrairement, mais elles suivent une politique de construction qui est une fonction des contraintes du problème.

Les traces de phéromone τ peuvent être associées aux composants ou aux connexions du graphe (τ_i pour un composant, τ_{ij} pour une connexion) créant une mémoire à long terme sur la totalité du processus de recherche de la fourmi, qui change dynamiquement durant la résolution afin de refléter l'expérience de recherche acquise par les agents, et une valeur heuristique η (η_i, η_{ij} respectivement) représentant une information sur l'instance du problème ou une information sur le temps d'exécution fourni par une source autre que les fourmis. Dans la plupart des cas, η est le coût ou une estimation du coût de l'extension de l'état courant.

2.4.3 La métaheuristique ACO

Dans les algorithmes ACO [Dorigo & Di Caro, 1999, Dorigo & Stüzle, 2004], les fourmis se déplacent en appliquant une politique de décision stochastique locale qui se base sur l'utilisation des traces de phéromone et d'une information heuristique spécifique au problème traité. Des coefficients α et β permettent de contrôler l'importance relative des deux éléments. Chaque fourmi possède une forme de mémoire afin d'obliger celle-ci à former une solution admissible.

En se déplaçant, les fourmis construisent incrémentalement des solutions au problème d'optimisation. Une fois qu'une fourmi a construit une solution, ou lorsque la solution est en cours de construction, la fourmi évalue la solution (partielle) et dépose la phéromone dans le composant ou la connexion qu'elle a utilisé. Cette information de phéromone dirigera la recherche des futures fourmis.

L'intensité de ces traces décroît dans le temps par un facteur constant appelé coefficient d'évaporation. D'un point de vue pratique, l'évaporation de phéromone est nécessaire pour éviter une convergence prématurée de l'algorithme vers une région sous-optimale. Ce processus implémente une forme utile d'oubli favorisant l'exploration de nouvelles aires de recherche.

2.4.4 Applications

Les algorithmes ACO ont été appliqué à plusieurs problèmes combinatoires NP-difficiles. L'application de ACO au PVC a été déjà présentée dans la section précédente. Dans cette section, nous discutons des applications à d'autres problèmes d'optimisation combinatoire.

Problème de Tournée de Véhicules (VRP)

Parmi les applications les plus intéressantes de la métaheuristique ACO au VRP, nous citons l'algorithme AS-VRP conçu par Bullnheimer, Harlt et Strauss [Bullnheimer *et al.* , 1999] qui est basée sur l'algorithme AS_{rank}.

Gambardella, Taillard et Agazzi [Gambardella *et al.* , 1999b] ont traité le VRP en adaptant l'approche ACS pour définir MACS-VRPTW, et en considérant l'extension de la fenêtre temps au VRP qui introduit un intervalle de temps dans lequel un client doit être servi.

Le Problème de k-Partitionnement de Graphes

Le problème de k-partitionnement de graphes (k-PPG) consiste à partitionner les noeuds d'un graphe donné en k partitions de façon à minimiser les coûts des arcs inter-partitions tout en essayant d'assurer l'équilibre des poids des partitions.

Ce problème a été traité par la métaheuristique ACO par Alaya, Sammoud, Hammami et Ghédira [Alaya *et al.* , 2003]. Dans l'algorithme proposé, le graphe de

construction décrivant ce problème est un graphe complet dont les noeuds sont des couples $\langle X, P\rangle$ avec X est un noeud du graphe à partitionner et P est le numéro de partition à laquelle X peut appartenir. Ainsi, un couple $\langle X, P\rangle$ choisi par une fourmi signifie que le noeud X du graphe est affecté à la partition P, et par conséquent tout couple $\langle X, Q\rangle$ telque Q $\neq P$ devient interdit (n'appartient plus au voisinage faisable) puisque qu'un noeud ne peut pas appartenir à plus qu'une partition à la fois.

Au cours de chaque cycle, chaque fourmi produit un partitionnement en se déplaçant sur le graphe de construction : à partir du nœud courant $\langle X, P\rangle$, elle choisit de se délacer vers le nœud $\langle Y, Q\rangle$ selon une probabilité qui dépend : (i) du coût des arcs inter-partitions apporté par ce nouveau composant (l'information heuristique) et (ii) des traces de phéromone. La quantité de phé romone à déposer est inversement proportionnelle au coût de la solution générée.

Le problème du sac à dos multidimensionnel

Le MKP présente l'un des problèmes sur lesquels nous travaillons. Dans cette section, nous présentons les travaux proposés dans la littérature pour résoudre ce problème avec ACO.

Une application de ACO au MKP dans [Leguizamon & Michalewicz, 1999] définit le graphe de construction comme étant un graphe complet composé des objets du MKP (qui représentent les noeuds). La mise à jour de phéromone se fait sur les composants des solutions (c'est à dire les noeuds du graphe). Les traces de phéromone reflète donc la désirabilité de prendre un objet qui a fait partie de plusieurs solutions. La quantité de phéromone à déposer est égale au profit de la solution construite. L'information heuristique est calculée de façon dynamique en fonction des capacités des ressources (les contraintes) et du profit de l'objet à prendre.

Une autre application d'ACO au MKP a été présentée dans [Fidanova, 2002]. Dans cette application, les traces de phéromone sont déposées sur les arêtes visités.

Aprés que toutes les fourmis aient complété leurs tours, les fourmis déposent de la phéromone sur les chemins qu'elle viennent de parcourir. De même dans cette application la quantité de phéromone à déposer est égale au profit de la solution construite. Ensuite, la meilleure fourmi du cycle renforce les traces de phéromone sur les composants de la meilleure solution. De plus lorsque certains mouvements ne sont pas utilisés dans la solution courante, un renforcement de phéromone additionnelle est utilisé pour ces composants afin de les favoriser dans les itérations futures. L'information heuristique est calculée de façon statique.

2.5 ACO et la recherche locale

Dans plusieurs applications à des problèmes d'optimisation combinatoire NP-difficiles, les algorithmes ACO réalisent de meilleures performances lorsqu'ils sont combinés avec des algorithmes de recherche locale [Dorigo & Gambardella, 1997, Stützle & Hoos, n.d., Alaya *et al.* , 2006]. Ces algorithmes optimisent localement les solutions des fourmis et ces solutions optimisées localement sont utilisées par la suite dans la mise à jour de phéromone. L'utilisation de la recherche locale dans les algorithmes ACO peut être très intéressante comme les deux approches sont complémentaires. En effet, la combinaison peut améliorer largement la qualité des solutions produites par les fourmis.

D'un autre côté, générer des solutions initiales pour les algorithmes de recherche locale n'est pas une tâche facile. Dans les algorithmes de recherche locale, où les solutions initiales sont générées aléatoirement, la qualité des solutions peut être médiocre. En combinant la recherche locale avec l'ACO, les fourmis génèrent stochastiquement, en exploitant les traces de phéromone, des solutions initiales prometteuses pour la recherche locale.

2.6 Contrôle de l'intensification/diversification

Un point critique, lors de l'application d'un algorithme ACO, est de trouver un équilibre entre l'exploitation de l'expérience de recherche acquise par les fourmis et l'exploration de nouvelles zones de l'espace de recherche non encore visitées. ACO possède plusieurs manières pour accomplir une telle balance, essentiellement par la gestion des traces de phéromone. En effet, les traces de phéromone déterminent dans quelles parties de l'espace de recherche les solutions construites auparavant sont localisées.

Une manière de mettre à jour la phéromone, et pour exploiter l'expérience de recherche acquise par les fourmis, est de déposer une quantité de phéromone fonction de la qualité de la solution trouvée par chaque fourmi. Ainsi, les zones correspondant aux meilleures solutions recevront une quantité de phéromone plus élevée et seront plus sollicitées par les fourmis durant leurs déplacements.

Pour contrôler la balance entre l'exploitation de l'espace de recherche et l'exploration de nouvelles zones, ACO dispose de plusieurs paramètres pour gérer l'importance relatives des traces de phéromone, essentiellement les deux paramètres α et ρ.

α détermine le poids des traces de phéromone dans le calcul des probabilités de transition, et ρ détermine l'évaporation de phéromone. Ces deux paramètres ont une influence sur le comportement exploratoire ; pour favoriser la stratégie d'exploration, on peut diminuer le coefficient α, ainsi les fourmis deviennent moins sensibles aux phéromones lors de leurs déplacements et peuvent visiter des zones non explorées, ou diminuer ρ, ainsi la phéromone s'évapore plus lentement et l'intensité des traces de phéromone devient similaire sur les arêtes et donc les fourmis deviennent moins influencées par la phéromone.

Pour favoriser la stratégie d'exploitation, on augmente les valeurs respectives de α et/ou de ρ, ainsi les fourmis seront plus guidées par la phéromone vers des zones

'prometteuses' de l'espace de recherche. Après un certain nombre d'itérations, toutes les fourmis vont converger vers un ensemble de 'bonnes' solutions. Ainsi, on favorise une convergence plus rapide de l'algorithme.

Toutefois, il faut faire attention et éviter une intensification trop forte dans les zones qui apparaissent prometteuses de l'espace de recherche car cela peut causer une situation de stagnation : une situation dans laquelle toutes les fourmis génèrent la même solution.

Pour éviter une situation pareille de stagnation, une autre solution serait de maintenir un niveau raisonnable d'exploration de l'espace de recherche. Par exemple, dans ACS les fourmis utilisent une règle de mise à jour locale de phéromone durant la construction de solution pour rendre le chemin qu'elles viennent de prendre moins désirable pour les futures fourmis et ainsi, diversifier la recherche. MMAS introduit une limite inférieure pour l'intensité des traces de phéromone pour garantir toujours la présence d'un niveau minimal d'exploration. MMAS utilise aussi une réinitialisation des traces de phéromone, qui est une manière d'enforcer l'exploration de l'espace de recherche.

2.7 Conclusion

ACO est une méthode stochastique qui requiert de plus en plus l'attention de la communauté scientifique. Cette métaheuristique a prouvé sa performance pour plusieurs problèmes d'optimisation combinatoire NP-difficiles.

L'utilisation des traces de phéromone permet d'exploiter l'expérience de recherche acquise par les fourmis et ainsi renforcer l'apprentissage pour la construction de solutions dans les itérations futures de l'algorithme. En même temps, l'information heuristique peut guider les fourmis vers les zones prometteuses de l'espace de recherche.

Un autre avantage de ACO est que son domaine d'application est vaste. En principe, ACO peut être appliquée à n'importe quel problème d'optimisation discret pour lequel un mécanisme de construction de solution peut être conçu.

Toutefois, ACO a ses propres inconvénients qui résident, principalement, dans les problèmes de paramétrage. En effet, ACO nécessite une instanciation des différents paramètres qui dépend du problème. Généralement, les paramètres correspondant aux meilleures solutions prouvées expérimentalement sont retenus.

Chapitre 3

Optimisation multi-objectifs

3.1 Introduction

Dans la plupart des problèmes du monde réel, il ne s'agit pas d'optimiser seulement un seul critère mais plutôt d'optimiser simultanément plusieurs critères et qui sont généralement conflictuels. Dans les problèmes de conception, par exemple, il faut le plus souvent trouver un compromis entre des besoins technologiques et des objectifs de coût. L'optimisation multi-objectif consiste donc à optimiser simultanément plusieurs fonctions. La notion de solution optimale unique dans l'optimisation uni-objectif disparait pour les problèmes d'optimisation multi-objectif au profit de la notion d'*ensemble de solutions Pareto optimales*.

3.2 Définitions

3.2.1 Formulation

Un PMO(F,D) peut être défini comme suit :

$$\begin{array}{ll} min & F(x) = (f_1(x), f_2(x), ..., f_m(x)) \qquad m \geq 2 \\ avec & F = (f_1 : D \rightarrow IR, ...f_m : D \rightarrow IR) \end{array} \tag{3.1}$$

où m est le nombre de fonctions à optimiser, $x = (x_1, .., x_n)$ est le vecteur des variables de décisions,$F(x) = (f_1(x), f_2(x), .., f_m(x))$ est le vecteur des critères à optimiser.

L'image d'une solution x dans l'espace des critères est le point $y = (y_1, .., y_m)$ avec $y_i = f_i(x)$, $i = 1..m$, et $Y = F(D)$ représente les points réalisables dans l'espace des objectifs. On impose sur cet ensemble une relation d'ordre partiel appelée relation de dominance que nous allons définir dans la section suivante.

3.2.2 Notion de dominance

Définition

Une solution $y = (y_1, .., y_m)$ *domine faiblement* une solution $z = (z_1, .., z_m)$
Ssi $\forall i \in 1, .., m \ f_i(y) \leq f_i(z)$

Si la solution y domine faiblement la solution z nous allons noter $y \preceq z$

Définition

Une solution $y = (y_1, .., y_m)$ *domine* une solution $z = (z_1, .., z_m)$
Ssi $\forall i \in 1, .., m \ f_i(y) \leq f_i(z)$ et $\exists j \in 1, .., m$ tq $f_j(y) < f_j(z)$

Si la solution y domine la solution z nous notons alors $y \prec z$

Notons que pour toute paire de solutions y et z un et un seul des cas suivants peut se présenter :

– y domine z

– y est dominé par z

– y et z sont équivalentes au sens de la dominance.

Les solutions équivalentes au sens de la dominance sont appelées dans ce qui suit, solutions équivalentes au sens de Pareto ou solutions pareto équivalentes ou, encore, solutions non-dominées.

Propriétés de la relation de dominance

La relation de dominance telle qu'elle est définie ci-dessus :

– *n'est pas réflexive*, car une solution ne se domine pas elle même.

– *n'est pas symétrique*, car on n'a jamais $y \prec z$ et $z \prec y$

– *est transitive*, car si $y \prec z$ et $z \prec w$ alors $y \prec w$

Optimalité de Pareto

Soit P un ensemble de solutions candidates d'un problème d'optimisation multi-objectif. L'ensemble $P' \subseteq P$, composé de tous les éléments de P qui ne sont dominés par aucun élément de P est dit sous-ensemble non dominé de l'ensemble P.

– Optimalité locale au sens de Pareto : Une solution y est *optimale localement au sens de pareto* s'il existe un réel $\delta > 0$ telqu'il n'y ait pas une solution z dominant y et vérifiant $\|z - y\| \leq \delta$

– Optimalité globale au sens de Pareto : Une solution y est *optimale globalement au sens de pareto*, ou *optimale au sens de Pareto*, ou encore *Pareto-optimale* s'il n'existe aucun point de l'espace faisable D qui la domine. L'ensemble des solutions Pareto optimale est appelé *l'ensemble de Pareto* ou également *l'ensemble des compromis optimaux*. L'image de l'ensemble de Pareto dans l'espace des critères est appelé la *surface de Pareto* (ou le front de Pareto pour le cas bi-objectif) ou encore la *surface des compromis optimaux*.

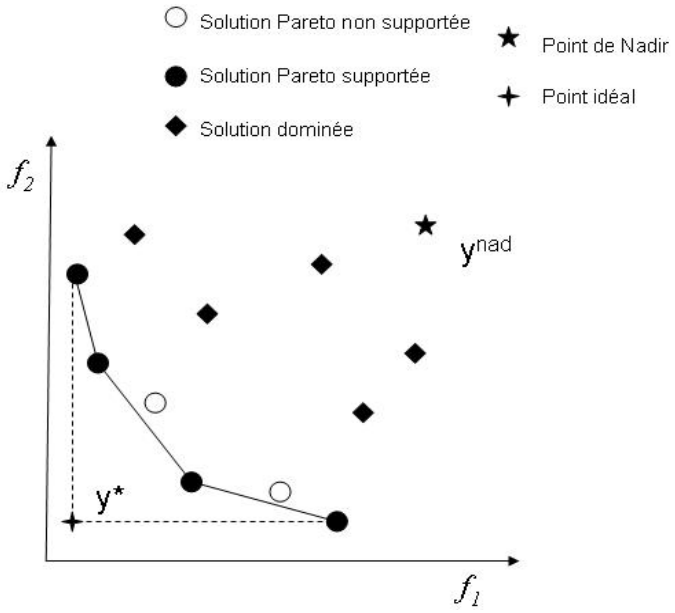

FIG. 3.1 – Représentation des solutions supportées et non supportées, point idéal et point de Nadir

3.2.3 Point idéal et Point de Nadir

Le vecteur idéal $y^* = (y_1^*, .., y_m^*)$ est obtenu en optimisant séparément chaque fonction objectif f_i, i.e. $y_i^* = f_i^*(x), x \in D$ (voir figure 3.1). Généralement ce vecteur n'appartient pas à l'espace objectif réalisable mais il est dans certains cas utile en tant que référence, par exemple, pour normaliser les valeurs des objectifs.

A la différence du vecteur idéal qui représente les bornes inférieures de chaque objectif dans l'espace faisable, le vecteur de Nadir correspond à leurs bornes supérieures sur la surface de Pareto, et non pas dans tout l'espace faisable (voir figure 3.1). Ce vecteur sert à restreindre l'espace de recherche ; il est utilisé dans certaines méthodes d'optimisation interactives.

3.2.4 Convexité

Comme nous le verrons plus tard, certaines méthodes d'optimisation multi-objectif nécessitent de travailler sur un espace Y des valeurs de F qui soit convexe.

Définition

Un ensemble Y est convexe si, pour n'importe quels deux points distincts de cet ensemble, le segment qui relie ces deux points est contenu dans l'ensemble S.

3.2.5 Mesures de performance

Dans l'optimisation multi-objectif, l'évaluation de la performance est beaucoups plus complexe que dans l'optimisation uni-objectif. La mesure des surfaces de compromis est un problème délicat car elle doit représenter la qualité des solutions, la taille du front, la répartition des solutions sur le front,...

Deux types de métriques ont été proposés dans la littérature : les métriques relatives, qui comparent deux ensembles, et les métriques absolues, qui évaluent un ensemble sans avoir besoin d'autres points ou ensemble de référence.

Ces mesures peuvent aussi être classées suivant qu'elles nécessitent la connaissance de la surface de Pareto optimale ou qu'elles n'exigent pas cette information.

Distance générationnelle

La distance générationnelle (Generational Distance) est une métrique qui nécessitent la connaissance de la surface de Pareto optimale. Elle permet de mesurer à quelle distance de la surface de compromis se situe un ensemble de solutions. La définition de cette métrique tirée de [Collette & Siarry, 2002] est la suivante :

Définition Distance générationnelle

$$G = \frac{(\sum_{i=1}^{n} d_i^p)^{\frac{1}{p}}}{n}$$

où d_i est la distance entre la solution i et la solution la plus proche appartenant à la surface de compromis optimale, et n est le nombre d'éléments de l'ensemble Pareto généré.

La métrique C

Cette métrique a été introduite dans [Zitzler & Thiele, 1999]. C'est une métrique relative, qui compare deux surfaces de compromis A et B. La valeur $C(A, B)$ correspond au pourcentage d'éléments de B faiblement dominés par au moins un des éléments de A. Le calcul de ce ratio s'effectue grâce à la formule suivante :

Définition La couverture de deux ensembles

Etant donnés deux ensembles de solutions X' et X'',

$$C(X', X'') = \frac{|\{a'' \in X'' : \exists a' \in X', a' \succeq a''\}|}{|X''|}$$

La valeur de $C(X', X'') = 1$, signifie que tous les points dans X'' sont dominés ou égaux aux points dans X', tandis que $C(X', X'') = 0$, signifie qu'aucun point dans X'' n'est dominé par un point de X'. Il est à noter qu'il est nécessaire de considérer les deux valeurs $C(X', X'')$ et $C(X'', X')$ car $C(X', X'') \neq 1 - C(X'', X')$.

L'hypervolume

C'est une métrique qui a été aussi introduite par Zitzler [Zitzler & Thiele, 1999]. Elle calcule une approximation du volume compris sous la courbe formée par les points de l'ensemble à évaluer. Ainsi, lorsque le problème comporte deux critères, ce calcul correspond au calcul d'une aire, lorsque le problème comporte trois critères, la valeur calculée est un volume... Elle est définie comme suit :

Définition Hypervolume

Soit $A = (x_1, x_2, ..., x_n) \subseteq X$ un sous-ensemble de n éléments. La fonction d'hypervolume calcule le volume borné par l'union de tous les polytopes $p_1, p_2, ..., p_n$, ou chaque p_i est formé par l'intersection des hyperplans des x_i par rapport aux axes du repère : pour chaque axe de l'espace des objectifs, il existe un hyperplan perpendiculaire à l'axe et passant par le point $(f_1(x_i), f_2(x_i), ..., f_k(x_i))$. Pour le cas à deux dimensions, chaque p_i représente un rectangle défini par les points de coordonnées $(0, 0)$ et $(f_1(x_i), f_2(x_i))$.

D'autres métriques ont été décrites dans l'ouvrage [Collette & Siarry, 2002].

3.3 Classification des méthodes de résolution

Dans le contexte de la résolution de PMO, très peu sont les travaux sur les méthodes exactes. Parmi ces méthodes, qui ont porté essentiellement sur les problèmes à deux critères, nous trouvons le "branch and bound" [Sen *et al.* , 1988, Sayin & Karabati, 1999, Visée *et al.* , 1998], l'algorithme A* [Stewart & White, 1991] et la programmation dynamique [White, 1982, Carraway *et al.* , 1990]. Ces méthodes restent efficaces pour des problèmes de petites tailles. Pour des problèmes de grandes tailles ou à plus de deux critères, il n'existe pas de procédures exactes efficaces, étant donné la complexité NP-difficile et le cadre multi-critère de ces problèmes. Ceci justifie le recourt aux méthodes heuristiques qui sacrifient la complétude pour gagner l'efficacité.

Dans la littérature, nous trouvons deux classifications différentes des méthodes de résolution des PMO. Le premier classement adopte un point de vue décideur. Le deuxième classe les méthodes suivant leur façon de traiter les fonctions objectifs.

Le premier classement divise ces méthodes en trois familles suivant la coopération entre la méthode d'optimisation et le décideur :

- Les méthodes *a priori* : Dans ces méthodes, le décideur définit le compromis qu'il désire réaliser. Ainsi, il est supposé connaitre a priori le poids de chaque objectif, ce qui revient à transformer le PMO en un problème uni-objectif. On retrouve dans cette méthode la plupart des méthodes par agrégation.
- Les méthodes *interactives* : Dans ce cas, il y a coopération progressive entre la méthode d'optimisation et le décideur. Le décideur affine son choix de compromis au fur et à mesure du déroulement du processus de résolution à partir des connaissances acquises à chaque étape du processus.
- Les méthodes *a posteriori* : Dans ce cas, le décideur choisit une solution parmi toutes les solutions de l'ensemble PO fourni par la méthode d'optimisation. Cette méthode est utilisable lorsque la cardinalité de l'ensemble PO est réduite.

On peut trouver des méthodes d'optimisation multi-objectif qui n'entrent pas exclusivement dans une famille. On peut utiliser, par exemple, une méthode a priori en lui fournissant des préférences choisies aléatoirement. Le résultat sera alors un ensemble PO à cardinalité élevée qui sera présenté au décideur pour qu'il choisisse une solution. Cette combinaison forme alors une méthode a posteriori.

Cette classification permet donc seulement de se faire une idée sur la démarche que l'on devra suivre pour obtenir la solution. La deuxième classification divise les méthodes de résolution de PMO en deux classes :

- la classe de méthodes transformant le problème initial en un problème uni-objectif, ainsi ces méthodes retournent une solution unique, elles utilisent généralement des méthodes d'agrégation
- et la classe de méthodes fournissant un ensemble de solution PO : cette classe est aussi divisée en deux sous-classes

  * Les méthodes Pareto : ces méthodes utilisent directement la notion de dominance au sens de Pareto dans la sélection des solutions générées. Cette idée a été initialement introduite par Goldberg [Goldberg, 1989] pour résoudre les

problèmes proposés par Schaffer [Schaffer, 1985].

* Les méthodes non Pareto : ces méthodes optimisent séparément les objectifs Nous allons décrire dans la suite de cette section des méthodes appartenant à la première classe. Nous allons nous focaliser pour les méthodes de la deuxième classe sur les métaheuristiques.

3.4 Méthodes transformant le problème multi-objectif en un problème uni-objectif

Cette classe de méthodes, transformant le problème initial en un problème uni-objectif, génère donc une seule solution et non pas un ensemble de solutions Pareto-optimales. Les solutions générées par ces méthodes sont dites supportées (voir figure 3.1). Ainsi les solutions non supportées de la figure 3.1 ne peuvent pas être atteintes par ces méthodes.

3.4.1 Agrégation pondérée

Cette méthode de résolution de PMO est la plus évidente et, probablement, la plus largement utilisée en pratique. Elle consiste à ramener le problème multi-objectif au problème de l'optimisation d'une combinaison linéaire des objectifs initiaux. Ainsi, il s'agit d'associer à chaque fonction objectif un coefficient de pondération et à faire la somme des fonctions objectifs pondérées pour obtenir une nouvelle et unique fonction objectif.

Un PMO, comme défini dans 3.1, se transforme alors de la manière suivante :

$$min \quad \sum_{i=1..m} w_i f_i(x) \quad m \geq 2 \tag{3.2}$$

où les poids $w_i \geq 0$ sont tels que $\sum_{i=1..m} w_i = 1$

Cette méthode peut, en faisant varier les valeurs du vecteur poids w, retrouver un ensemble de solutions supportées, si le domaine réalisable est convexe. Cependant, elle ne permet pas de trouver les solutions enfermées dans une concavité (les solutions non supportées).

Les résultats obtenus avec de telles méthodes dépendent fortement des paramètres choisis pour le vecteur de poids w. Les poids w_i doivent également être choisis en fonction des préférences associées aux objectifs, ce qui est une tâche délicate. Une approche généralement utilisée consiste à répéter l'exécution de l'algorithme avec des vecteurs poids différents.

3.4.2 Méthode $\epsilon-$contrainte

Dans cette approche, le PMO 3.1 est reformulé de la manière suivante :

$$\begin{array}{lll} min & f_k(x) & k \in 1..m \\ avec & f_j(x) \leq \epsilon_j, & j = 1..m, j \neq k \end{array} \tag{3.3}$$

où $\epsilon = (\epsilon_1, .., \epsilon_{k-1}, \epsilon_{k+1}, .., \epsilon_m)$. En d'autres termes, le problème consiste à retenir une fonction f_k comme critère unique à optimiser, tandis que les critères restants sont transformés en contraintes. Différentes valeurs de ϵ peuvent être données pour pouvoir générer plusieurs solutions Pareto optimales. La connaissance à priori des intervalles appropriés pour les valeurs ϵ_i est requise pour tous les objectifs.

3.4.3 Méthode programmation par but

Dans cette méthode, le décideur doit définir les buts qu'il désire atteindre pour chaque objectif. Ces valeurs sont introduites dans la formulation du problème le transformant en un problème uni-objectif. Par exemple, la fonction coût peut intégrer une norme pondérée qui minimise les déviations par rapport aux buts. Le

problème est formulé comme suit :

$$min \quad (\sum_{i=1..m} w_i |f_i(x) - y_i|^p)^{\frac{1}{p}} \quad m \geq 2 \tag{3.4}$$

où $1 \leq p \leq \infty$, et y est le vecteur de référence (but) ou le vecteur idéal.

Plusieurs métaheuristiques ont utilisé l'approche transformant le PMO en un problème uni-objectif :

- Algorithmes génétiques : nous citons l'algorithme HLGA [Hajela & Lin, 1992]dont la sélection est basée sur la performance des individus calculée comme la somme pondérée des objectifs. A la différence de la méthode d'agrégation classique, ici, un vecteur des poids différent est associé à chaque individu.
- Recherche taboue : la recherche taboue a été utilisée pour résoudre un problème d'affectation de fréquences [Dahl *et al.* , 1995].
- Recuit simulé : Le recuit simulé a été appliqué au problème de voyageur de commerce multi-objectif [Serafini, 1992], au problème du sac à dos multi-objectif [Ulungu *et al.* , 1998], au problème de design de réseaux de transport [Friesz *et al.* , 1993].

3.5 Approches non-agrégées

Nous présentons dans cette section quelques métaheuristiques appartenant à cette classe de méthodes.

3.5.1 Les algorithmes génétiques

Les AGs ont été largement utilisés dans la communauté multi-objectif. Ils sont trés appropriés pour résoudre des PMOs grâce à l'utilisation d'une population de solutions. Les AGs peuvent chercher plusieurs solutions Pareto-optimales dans la même exécution.

Vector Evaluated Genetic Algorithm (VEGA)

Le premier algorithme génétique (AG) proposé dans la littérature pour résoudre des PMOs est l'algorithme VEGA développé par Schaffer [Schaffer, 1985]. VEGA sélectionne les individus de la population courante suivant chaque objectif séparément. A chaque génération, la population est divisée en m sous-populations, où m est le nombre d'objectifs à optimiser. Chaque sous-population i est sélectionnée suivant l'objectif f_i. Les m sous-populations sont ensuite mélangées pour construire une nouvelle population, à laquelle sont appliqués les opérateurs génétiques (croisement et mutation).

Dans l'algorithme VEGA la méthode de sélection ne tient compte que d'un seul objectif, elle favorise les meilleurs individus par rapport à cet objectif. Ainsi, ces individus ne seront sélectionnés que lorsque la sélection est effectuée sur cet objectif. Les individus ayant une performance générale acceptable, appelé par Schaffer les individus "milieu", vont être éliminés car ils ne seront sélectionnés dans aucune sous-population. Ceci empêche la méthode d'atteindre les solutions "compromis".

Les AGs présentés dans la suite utilisent la notion de dominance de Pareto dans la sélection des solutions générées.

Multi-Objective GA (MOGA)

MOGA [Fonseca & Fleming, 1993], proposé par Fonseca et Fleming, est le premier algorithme qui utilise directement la notion de dominance de Pareto. L'algorithme utilise une procédure de ranking dans laquelle, pour chaque solution i, le nombre n_i de solutions la dominant est calculé, et le rang $r_i = (1 + n_i)$ lui est associé. Ainsi, le rang de chaque solution non-dominée est égal à 1 et le rang maximal ne peut pas être plus grand que la taille de la population N. La performance (fitness) F_i de chaque individu est basée sur son rang. La performance est calculée de sorte que tous les individus du même rang aient la même performance et que cette

performance soit à maximiser.

Niched-Pareto Genetic Algorithm (NPGA)

Horn et al [Horn *et al.* , 1994] ont proposé en 1994 la méthode NPGA. Cette méthode utilise un nichage [1] mis à jour dynamiquement [Oei *et al.* , 1991], et une sélection par tournoi. Une paire de solutions i et j est choisie aléatoirement de la population P. Ensuite ces deux solutions sont comparées au sens de la dominance à une sous-population T choisie aussi aléatoirement :

- Si une des solutions i ou j domine toutes les solutions de T alors que l'autre solution est dominée par au moins une solution de T, alors la première gagne le tournoi (elle est choisie)
- Si les deux solutions i et j sont soit dominées par au moins une solution de T soit dominent tout l'ensemble T, alors elles sont mises dans la population des enfants Q et leur compteur de niche (niche count) dans Q est calculé. La solution qui a le compteur de niche le plus petit gagne le tournoi.

Le "Non Sorting Dominated Genetic Algorithm II" (NSGA-II)

Srinivas et Deb [Srinivas & Deb, 1994] ont implémenté en 1994 la première procédure de ranking qui a été initialement introduite par Goldberg [Goldberg, 1989], nommé le Non Sorting Genetic Algorithm (NSGA). Tous les individus non-dominés de la population possèdent le rang 1. Ces individus sont ensuite enlevés de la population et l'ensemble suivant d'individus non-dominés est identifié et on leur attribue le rang 2. Ce processus est réitéré jusqu'à ce que tous les individus de la population aient un rang. Une nouvelle version élitiste de cet algorithme, nommée Non Sorting Genetic Algorithm II (NSGA-II), a été présentée dans [Deb *et al.* , 2002]. NSGA-II intègre un opérateur de sélection, basé sur un calcul de la distance de "crowding"

[1] une niche écologique est un ensemble d'individus situés dans un espace restreint

(ou "surpeuplement")[2] qui estime la densité de chaque individu dans la population. Comparativement à NSGA , NSGA-II obtient de meilleurs résultats sur toutes les instances présentées dans les travaux de K. Deb, ce qui fait de cet algorithme un des plus utilisés aujourd'hui.

Pour gérer l'élitisme, NSGA-II n'utilise pas d'archive externe pour stocker l'élite, comme le font d'habitude les autres AGs. NSGA-II assure qu'à chaque nouvelle génération, les meilleurs individus rencontrés soient conservés.

A chaque itération t, l'algorithme utilise deux populations séparés P_t et Q_t de taille N. La population P_t contient les meilleurs individus rencontrés jusqu'à l'itération t, et la population Q_t est formée des autres individus, issus des phases précédentes de l'algorithme.

La première étape consiste à créer une population combinée $R_t = P_t \bigcup Q_t$ et à appliquer une procédure de ranking. Puisque tous les individus précédents et courants de la population sont inclus dans R_t, l'élitisme est assuré. La deuxième phase consiste à construire une nouvelle population P_{t+1} contenant les N meilleurs individus de R_t. La troisième phase consiste à remplir la population Q_{t+1}. Il suffit alors d'utiliser les opérateurs de sélection, croisement et mutation sur les individus de P_{t+1}, puis d'insérer les descendants dans Q_{t+1}.

Le "Strength Pareto Evolutionary Algorithm 2" (SPEA2)

L'algorithme SPEA2, proposé par Zitler et al dans [Zitzler *et al.* , 2001], a été développé comme une amélioration de SPEA [Zitzler & Thiele, 1999]. SPEA est un algorithme élitiste qui maintient une population externe, appelée aussi archive, contenant le meilleur front de compromis rencontré durant la recherche. Cette po-

[2]L'approche par "crowding" consiste à déterminer un représentant par niche découverte, seuls les représentants participeront aux différentes étapes de l'algorithme. Le "crowding" fut introduit initialement par De Jong [Jong, 1975].

pulation est destinée à contenir un nombre limité de solutions non-dominées trouvées depuis le début de l'exécution. A chaque itération de nouvelles solutions non-dominées sont comparées aux individus de cette population au sens de dominance et seules les solutions non dominées résultantes restent. Il est à noter que SPEA non seulement préserve l'élite mais aussi la fait participer aux opérations génétiques. Une procédure de clustering est appliquée à cette population externe pour réduire sa taille si elle dépasse la taille limite.

SPEA2 diffère de son prédécesseur en plusieurs aspects. D'abord, la taille de l'archive est fixe, c'est à dire que s'il n'y a pas suffisamment d'individus non-dominés, l'archive est complété par ceux dominés. Notons qu'une telle situation ne peut se produire que durant les premières générations et durant une période relativement courte.

Nous allons noter dans ce qui suit P la poulation courante de taille N, et P' l'archive de taille N'.

Le calcul de performance est plus raffiné que celui de SPEA, au sens qu'il tient plus compte de la densité des solutions. Pour attribuer une valeur de fitness à un individu p de la population P et P', l'algorithme utilise la règle suivante :

$$fitness(p) = R(p) + D(p)$$

Dans cette équation, $R(p)$ est la performance préliminaire (raw fitness) de l'individu p, et $D(p)$ est sa densité. La performance préliminaire est calculée suivant des valeurs de '$Strength$' des solutions. $D(p)$ est calculée en utilisant une adaptation de la technique du "$k^{ième}$ plus proche voisin" [Silverman, 1986].

Les modifications apportées répondent à certaines critiques soulevées par SPEA. La procédure de clustering qui contrôlait la taille de l'archive dans SPEA est remplacée dans SPEA2 par une méthode de troncation qui, contrairement au clustering, préserve les individus situés aux extrémités du front Pareto. Une autre différence importante est que dans SPEA2 seuls les membres de l'archive participent au processus

de reproduction.

3.5.2 Recuit simulé

Le premier algorithme multi-objectif basé sur le recuit simulé a été proposé par Serafini [Serafini, 1992] en 1992. La méthode utilise le schéma standard du recuit simulé avec une seule solution courante. Le résultat de l'algorithme est un ensemble de solutions Pareto-optimales contenant toutes les solutions non-dominées générées par l'algorithme.

Serafini considère un certain nombre de règles d'acceptance définissant la probabilité d'acceptance des nouvelles solutions voisines. Le recuit simulé uni-objectif accepte avec une probabilité égale à 1 les nouvelles solutions meilleures ou égales à la solution courante. Si la nouvelle solution est inférieure à la solution courante elle est acceptée avec une probabilité inférieure à 1. Dans le cas multi-objectif, trois situations sont possibles en comparant une nouvelle solution x' avec la solution courante x :

- x' domine x,
- x' est dominée par x,
- x et x' sont mutuellement non-dominées.

Dans le premier cas, il est évident d'accepter x' avec une probabilité égale à 1. Dans le deuxième cas la probabilité d'acceptance doit être inférieure à 1. La question se pose pour la troisième situation quelle doit être la valeur de la probabilité d'acceptance. Serafini propose des règles d'acceptance multi-objectif qui traitent différemment la troisième situation. Ces règles correspondent à certaines fonctions d'agrégation locales.

Ulungu et al [Ulungu *et al.* , 1999] ont proposé une méthode appelé MOSA. Ils uilisent aussi des règles d'acceptance multi-objectif. MOSA utilise un nombre de vecteurs poids prédifinis. Chaque vecteur est associé à un processus de recuit in-

dépendant. Chaque processus commence d'une solution aléatoire ou d'une solution construite par une heuristique spécialisée. Ensuite, la solution réalise des mouvements qui sont acceptés avec une probabilité définie par une règle d'acceptance. Le résultat de l'algorithme est un ensemble de solutions Pareto-optimales contenant toutes les solutions non-dominées générées par tous les processus de recuit. Ainsi, ces processus, qui travaillent indépendamment, coopèrent dans la génération d'un ensemble commun de solutions Pareto-optimales.

Czyzak et Jaszkiewicz [Czyzak & Jaszkiewicz, 1998] ont proposé la méthode "Pareto simulated annealing". Cette méthode utilise des fonctions scalaires basées sur les probabilités pour l'acceptance des nouvelles solutions voisines. Une population de solutions est utilisée, chacune d'elles explorant l'espace de recherche relativement aux règles du recuit simulé. Les solutions peuvent être traitées comme des agents travaillant indépendamment mais échangeant des informations sur leurs positions. A chaque solution est associé un vecteur poids séparé. La méthode met à jour les vecteurs poids des solutions en construction en utilisant une règle pour assurer la dispersion des solutions sur toutes les régions du front Pareto.

Suppapitnarm et Parks [Suppapitnarm & Parks, 2001] ont proposé une méthode dans laquelle la probabilité d'acceptance d'une nouvelle solution dépend du fait qu'elle est ajoutée ou non à l'ensemble des solutions Pareto-optimales potentielles. Si elle est ajoutée à cet ensemble, ce qui signifie qu'elle n'est dominée par aucune autre solution trouvée, elle est acceptée pour être la solution courante avec une probabilité égale à 1. Sinon, une règle d'acceptance multi-objectif est utilisée.

3.5.3 Recherche tabou

Grandibleux et al [Gandibleux *et al.* , 1997] ont proposé une version multi-objectif de la recherche taboue. La méthode utilise des fonctions pondérées scalaires dont les poids sont changés périodiquement. La modification du vecteur poids dégrade

les poids des objectifs qui ont été nettement améliorés. Deux listes taboues sont utilisées. La première est une liste taboue régulière qui empêche de revenir aux solutions déjà visitées. La deuxième contient les vecteurs poids.

Hansen [Hansen, 1998] a proposé une recherche taboue basée sur l'idée de la méthode "Pareto simulated annealing" [Czyzak & Jaszkiewicz, 1998]. La méthode utilise une population de solutions explorant chacune d'elles différentes régions de l'ensemble Pareto. Le vecteur poids est utilisé dans une fonction scalaire. De plus, à chaque solution est associée une liste taboue. La dispersion des solutions est assurée par la modification de leurs vecteurs poids. Pour chaque solution, la modification du vecteur poids vise à la déplacer des autres solutions proches dans l'espace des objectifs. Hansen a appliqué cette méthode au problème du sac à dos multi-objectif.

Ben Abdelaziz et Krichen [Ben Abdelaziz & Krichen, 1997] ont proposé un algorithme de recherche taboue multi-objectif conçu spécialement au problème du sac à dos multi-objectif. La méthode est guidée par des fonctions scalaires pondérées linéaires et par une relation de dominance. La relation de dominance est appliquée non seulement aux solutions mais aussi aux objets. La recherche locale est basée sur l'idée de l'insertion des objets non-dominés au sac à dos.

3.5.4 Approches hybrides

Plusieurs approches hybrides ont été proposées dans la littérature essayant de tirer profit des avantages de chacune des méthodes utilisées ou bien pour combler certaines de ses lacunes. Plusieurs travaux ont proposé des méthodes hybrides pour résoudre des PMOs.

Jaszkiewicz a proposé l'algorithme "Multi-Objective Genetic Local Search" (MOGLS) qu'il a appliqué au problème de voyageur de commerce multi-objectif [Jaszkiewicz, 2002a] et au problème du sac à dos multi-objectif [Jaszkiewicz, 2002b]. Cet algorithme utilise une méthode de descente pure comme opérateur de recherche lo-

cale. A chaque itération, l'algorithme construit une fonction d'utilité aléatoire ainsi qu'une population temporaire composée d'un nombre de meilleures solutions à partir des solutions générées. Ensuite, une paire de solutions sélectionnée aléatoirement est recombinée. La recherche locale est appliquée à chaque solution générée.

Barichard et Hao [Barichard & Hao, 2003] ont proposé l'algorithme GTS^{MOKP} pour résoudre le problème du sac à dos multi-objectif. Cet algorithme combine une procédure génétique avec un opérateur de recherche tabou.

Récemment, plusieurs algorithmes évolutionnaires qui intègre un calcul d'indicateur d'hypervolume ont été proposés dans la littérature. Une analyse de ces algorithmes a été proposée dans [Brockhoff *et al.* , 2008].

3.6 Conclusion

Nous avons défini dans ce chapitre les problèmes d'optimisation multi-objectifs et les éléments de base relatifs. Nous avons présenté plusieurs approches heuristiques qui ont été proposées pour résoudre ces problèmes. Nous consacrons le prochain chapitre à la présentation des approches ACO proposées dans la littérature pour des PMOs.

Chapitre 4

Optimisation par colonies de fourmis pour la résolution de problèmes multi-objectifs

4.1 Introduction

Récemment plusieurs travaux ont proposé des algorithmes basés sur l'optimisation par colonies de fourmis pour résoudre des PMO, appelés dans la littérature algorithmes MOACO.

4.2 Algorithmes MOACO

Nous présentons dans cette section les algorithmes MOACO les plus connus qui ont été proposés dans la littérature.

4.2.1 Multiple Objective Ant-Q

L'algorithme "Multiple Objective Ant-Q" (MOAQ) a été proposé par Mariano et Morales dans [Mariano & Morales, 1999] pour résoudre le prolème de distribution de l'eau dans les réseaux d'irrigation. Cet algorithme est basé sur un algorithme d'apprentissage renforcé distribué appelé Ant-Q [Gambardella & Dorigo, 1995].

L'idée de base de MOAQ est de réaliser un algorithme d'optimisation avec une famille d'agents (fourmis) pour chaque objectif. Chaque famille k essaye d'optimiser un objectif en considérant les solutions trouvées pour les autres objectifs et leur fonction correspondante HE^k. De cette manière, tous les agents des différentes familles travaillent dans le même environnement en proposant des actions et une valeur de récompense r qui dépend de comment leurs actions ont participé à trouver des solutions de compromis parmi les autres agents.

MOAQ présente d'autres caractéristiques spécifiques. D'abord, lors de la construction de solution, la $j^{ème}$ fourmi de la $i^{ème}$ famille utilise la solution trouvée par la $j^{ème}$ fourmi de la famille $i-1$. De plus, lorsqu'une solution trouvée n'est pas faisable, l'algorithme applique une pénalité à ses composants sur les valeurs de Q. Finalement, tout au long du processus, les solutions non-dominées sont enregistrées dans un ensemble externe.

4.2.2 L'algorithme BicriterionAnt

L'algorithme BicriterionAnt a été proposé par Iredi et al. [Iredi *et al.* , 2001] spécialement pour résoudre le problème de tournée de véhicules bi-critère. Pour ce faire, il utilise deux structures de traces de phéromone différentes, τ et τ', une pour chaque critère considéré.

A chaque génération, chacune des m fourmis de la colonie génère une solution au problème. Durant l'étape de construction, la fourmi choisit le prochain sommet

j à visiter relativement à la probabilité suivante :

$$p(j) = \frac{\tau_{ij}^{\lambda\alpha}.{\tau'_{ij}}^{(1-\lambda)\alpha}.\eta_{ij}^{\lambda\alpha}.{\eta'_{ij}}^{(1-\lambda)\alpha}}{\sum_{u\in\Omega}\tau_{iu}^{\lambda\alpha}.{\tau'_{iu}}^{(1-\lambda)\alpha}.\eta_{iu}^{\lambda\alpha}.{\eta'_{iu}}^{(1-\lambda)\alpha}}$$

où η_{ij} et η'_{ij} sont les valeurs heuristiques associées à l'arête a_{ij} relativement au premier et second objectif, respectivement, Ω est le voisinage faisable de la fourmi, et λ est calculé pour chaque fourmi h, $f \in 1, ..., m$, comme suit :

$$\lambda_h = \frac{h-1}{m-1}$$

Une fois que toutes les foumis ont construit leurs solutions, les traces de phéromone sont évaporées par la règle habituelle. Ensuite, chaque fourmi qui a généré une solution dans le front Pareto au cycle courant est autorisée à mettre à jour les deux structures phéromone τ et τ', en déposant une quantité égale à $\frac{1}{l}$, où l est le nombre de fourmis qui sont en train de mettre à jour les traces de phéromone. Les solutions non-dominées générées tout au long de l'exécution de l'algorithme sont mises dans un ensemble externe.

4.2.3 L'algorithme BicriterionMC

Iredi et al. ont proposé dans le même travail [Iredi *et al.* , 2001] un autre algorithme MOACO, "BicriterionMC", très similaire au précédent BicriterionAnt. La différence principale est que chaque foumi met à jour une seule structure de phéromone dans le nouvel algorithme. Les auteurs introduisent une définition générale pour l'algorithme basée sur l'utilisation de p structures de traces de phéromone et par la suite mettent $p = 2$ pour résoudre des problèmes bi-critères. Pour ce faire, ils considèrent deux méthodes différentes pour la mise à jour des traces de phéromone :

- Méthode 1. *Mise à jour par origine* : une fourmi dépose des traces de phéromone seulement dans sa colonie. Cette méthode force les deux colonies à

chercher dans des régions différentes du front Pareto. L'algorithme utilisant cette première méthode est appelé "UnsortBicriterion".

– Méthode 2. *Mise à jour par région* : la séquence des solutions sur le front Pareto est divisée en p parties de taille égale. Les fourmis qui ont trouvé des solutions dans la $i^{ème}$ partie dépose de la phéromone pour la colonie i, $i \in [1, p]$. Le but est de guider explicitement les fourmis des colonies de chercher dans des régions différentes du front Pareto, chacune dans une région. L'algorithme utilisant cette méthode est appelé "BicriterionMC".

4.2.4 Pareto Ant Colony Optimization

"Pareto Ant Colony Optimization" (P-ACO) a été proposé par Doerner et al. dans [Doerner *et al.* , 2004]. Il a été initialement appliqué au problème de sélection de portefeuille multi-objectif. Il suit le schéma général de l'algorithme ACS, mais la mise à jour globale de phéromone est réalisée en utilisant deux fourmis, la meilleure et la deuxième-meilleure solution générée dans le cycle courant pour chaque objectif k. Dans P-ACO, plusieurs structures phéromone τ^k sont considérées, une pour chaque objectif. A chaque cycle de l'algorithme, chaque fourmi calcule un ensemble de poids $p = (p_1, ..., p_k)$, et l'utilise pour combiner les traces de phéromone et l'information heuristique.

Les solutions non-dominées trouvées tout au long de l'exécution sont là aussi enregistrées dans un ensemble externe.

4.2.5 Multiple Ant Colony System pour le problème de tournées de véhicules avec fenêtres de temps

l'algorithme "Multiple Ant Colony System for Vehicle Routing Problem with Time Windows" (MACS-VRPTW) a été introduit par Gambardella et al. [Gambar-

della *et al.* , 1999b]. Il utilise, comme P-ACO, le schéma de ACS.

MACS-VRPTW est organisé avec une hiérarchie des colonies de fourmis conçue pour optimiser successivement une fonction multi-objectif : la première colonie, ACS-VEI, minimise le nombre de véhicules alors que la deuxième,ACS-TIME, optimise les solutions faisables trouvées par la première. Chaque colonie utilise une structure phéromone indépendante pour son objectif spécifique et elles collaborent en partageant la meilleure solution globale trouvée ψ^{gb}. Lorsque ACS-VEI est activée, elle essaie de trouver une solution faisable avec un véhicule en moins que ceux utilisés dans ψ^{gb}. Le but de ACS-TIME est d'optimiser le temps total de la tournée des solutions qui utilisent le même nombre de véhicules utilisé dans ψ^{gb}. A chaque fois une meilleure solution est trouvée l'algrithme tue les deux colonies, et le processus est réitéré : deux nouvelles colonies sont activées travaillant avec la nouvelle solution.

4.2.6 Multiple Ant Colony System

Multiple Ant Colony System (MACS) [Barán & Schaerer, 2003] est une variante de MACS-VRPTW décrit dans la section précédente. Il utilise aussi ACS, mais contrairement à son prédécesseur, MACS utilise une seule structure phéromone, τ, et plusieurs informations heuristiques, η^k, initialement deux, η^0 et η^1. De cette manière, une fourmi se déplace d'un sommet i à un sommet j en appliquant la règle suivante :

$$j = \begin{cases} arg \max\limits_{j \in \Omega} (\tau_{ij}.[\eta^0_{ij}]^{\lambda\beta}.[\eta^1_{ij}]^{(1-\lambda)\beta}) & \text{si } q \leq q_0, \\ \hat{i}, & \text{sinon.} \end{cases}$$

où λ est calculée pour chaque fourmi h comme $\lambda = \frac{h}{m}$, avec m est le nombre de fourmis, et $\hat{i}$ est le sommet choisi relativement à la probabilité suivante :

$$p(j) = \frac{\tau_{ij}.[\eta^0_{ij}]^{\lambda\beta}.[\eta^1_{ij}]^{(1-\lambda)\beta}}{\sum\limits_{u \in \Omega} \tau_{iu}.[\eta^0_{iu}]^{\lambda\beta}.[\eta^1_{iu}]^{(1-\lambda)\beta}}$$

A chaque fois une fourmi traverse une arête a_{ij}, elle réalise une mise à jour de phéromone locale comme suit :

$$\tau_{ij} = (1 - \rho).\tau_{ij} + \rho.\tau_0,$$

Initialement, τ_0 est calculé d'un ensemble de solutions heuristiques en prenant leurs moyennes dans chaque fonction objectif, f^0 et f^1, et en appliquant l'expression suivante :

$$\tau_0 = \frac{1}{\hat{f}_0.\hat{f}_1}$$

Cependant, la valeur de τ_0 ne reste pas fixe, comme habituellement dans ACS, mais elle est mise à jour durant l'exécution de l'algorithme, avec la formule précédente avec les nouvelles valeurs des fonctions objectifs des solutions non dominées trouvées.

La mise à jour globale est réalisée avec chaque solution S de l'ensemble Pareto courant en appliquant la règle suivante :

$$\tau_{ij} = (1 - \rho).\tau_{ij} + \frac{\rho}{f^0(S).f^1(S)}.$$

4.2.7 COMPETants

Doerner et al. ont introduit l'algorithme COMPETants [Doerner *et al.* , 2003] pour traiter les problèmes de transport bi-objectif. L'algorithme adopte le schéma de "rank-based AS" [Bullnheimer *et al.* , 1999]. Il utilise deux colonies de fourmis, chacune avec sa propre structure de phéromone et sa propre information heuristique. Les deux colonies sont en compétition sur le nombre de fourmis. Le nombre de fourmis dans chaque colonie n'est pas fixe, la colonie qui trouve de meilleures solutions reçoit plus de fourmis dans la prochaine itération. De plus, l'information passe entre les deux colonies, puisque les fourmis observent et utilisent non seulement leur propre trace de phéromone mais aussi la trace étrangère. La décision des fourmis d'utiliser ou non la trace de phéromone étrangère est basée sur la meilleure

solution trouvée dans chaque colonie. Les fourmis qui décident d'utiliser la trace étrangère sont appelées "*spies*".

4.2.8 Multi-Objective Network ACO

Multi-objective Network ACO (MONACO) [Cardoso *et al.* , 2003] a été conçu pour être appliqué à un problème dynamique, l'optimisation du trafic d'un message dans un réseau. Cet algorithme est différent des autres MOACO présentés puisqu'il traite un problème dynamique, la "policy" du réseau change suivant les étapes de l'algorithme et elle n'attend pas la fin de l'exécution. MOACO suit le scéma de base de AS classique mais utilise plusieurs traces de phéromone et une seule information heuristique. Chaque fourmi, qui est représentée comme un message, utilise pour se déplacer, dans la probabilité de transition, une agrégation des traces de phéromone.

L'objectif de l'algorithme n'est pas de trouver un bon ensemble de solution non-dominées mais de rendre le réseau travaille efficacement.

4.2.9 Crowding Population-based Ant Colony Optimisation

Crowding Population-based Ant Colony Optimisation (CPACO) [Angus, 2007] a été proposé pour résoudre le problème de voyageur de commerce multi-objectif. CPACO étend l'algorithme Population-based Ant Colony Optimisation (PACO) [Guntsch & Middendorf, 2002] en utilisant, d'abord, un schéma de remplacement de "crowding" (ou surpeuplement)[Jong, 1975]. CPACO utilise une seule structure phéromone avec des structures heuristiques individuelles, alors que PACO utilise des structures phéromone et heuristiques individuelles pour chaque objectif. A chaque itération une nouvelle structure phéromone est calculée comme suit :

- A toutes les solutions (s) dans la population (S) est affecté un rang relativement au rang de dominance défini dans l'algorithme NSGAII [Deb *et al.* , 2002].

– Tous les éléments dans la structure phéromone sont initialisés à une certaine valeur (τ_{init}).
– Toutes les solutions incrémentent leurs éléments correspondants dans la structure phéromone relativement à l'inverse de leur rang, i.e. $\Delta\tau_{ij}^{s} = 1/(s_{rank})$.

Pour déterminer les poids pour chaque objectif, CPACO affecte à chaque fourmi un ensemble unique de facteurs heuristiques (λ). Ceci permet à chaque fourmi d'exploiter les structures heuristiques avec différentes quantités tout en utilisant une structure phéromone commune.

4.3 Taxonomie des algorithmes MOACO

Pour résoudre un problème d'optimisation multi-objectif (PMO) avec ACO, plusieurs points sont à définir pour lesquels différents choix sont possibles. Une taxonomie de plusieurs algorithmes MOACO a été proposée dans [Garcia-Martinez *et al.* , 2007]. Cette taxonomie classe les algorithmes MOACO par le nombre de structures de phéromone et le nombre d'informations heuristiques. Nous proposons dans ce travail une autre taxonomie présentée au tableau 4.1. En effet, nous considérons dans cette taxonomie que ces algorithmes diffèrent essentiellement dans les points suivants :

Structures de phéromone. Les traces de phéromone représentent le moyen qu'utilise une fourmi pour attirer les autres fourmis de sa colonie vers les aires correspondantes. Pour le cas uni-objectif, ces traces reflètent la désirabilité de visiter une zone de l'espace de recherche suivant la fonction à optimiser. En traitant un PMO, où on a plusieurs objectifs à optimiser simultanément, deux choix s'offrent :

1. On peut utiliser une seule structure de phéromone comme proposé dans [Angus, 2007, McMullen, 2001, Gravel *et al.* , 2002, Barán & Schaerer, 2003, Ma-

riano & Morales, 1999]. Dans ce cas, les traces de phéromone déposées par une fourmi sont définies relativement à une agrégation des objectifs à optimiser.

2. Une deuxième possibilité consiste à considérer plusieurs structures de phéromone comme proposé dans [Doerner *et al.* , 2004, Doerner *et al.* , 2003, Cardoso *et al.* , 2003, Iredi *et al.* , 2001, Gambardella *et al.* , 1999b]. Dans ce cas, généralement on utilise plusieurs colonies de fourmis, chacune utilise sa propre phéromone.

Définition du facteur phéromone. A chaque étape de la construction de solution d'une fourmi, un candidat est choisi relativement à une probabilité de transition qui dépend de deux facteurs : un facteur phéromone et un facteur heuristique. La définition du facteur phéromone dépend de la définition de la structure de phéromone, comme discuté dans le premier point.

1. Lorsqu'une seule structure phéromone est utilisée, le facteur phéromone est défini relativement à cette structure comme dans [McMullen, 2001, Gravel *et al.* , 2002].

2. Lorsque plusieurs structures phéromone sont utilisées, on peut soit utiliser pour chaque objectif à optimiser la structure phéromone correspondante comme proposé dans [Doerner *et al.* , 2003, Gambardella *et al.* , 1999b], soit utiliser toutes les structures phéromone. Dans ce dernier cas, généralement une agrégation des structures phéromonales est utilisée : une somme pondérée comme proposée dans [Doerner *et al.* , 2004] ou un produit pondéré comme dans [Iredi *et al.* , 2001, Cardoso *et al.* , 2003].

Définition du facteur heuristique. Ce facteur heuristique doit donner une idée sur l'apport du candidat en question à la solution en cours de construction. Deux stratégies différentes ont été considérées :

1. une première stratégie est de considérer une agrégation des différents objectifs dans une seule information heuristique et d'utiliser cette information comme facteur heuristique, comme proposé dans [Doerner *et al.* , 2004, Cardoso *et al.* , 2003, McMullen, 2001, Gravel *et al.* , 2002],

2. une deuxième stratégie consiste à associer à chaque objectif une structure heuristique. Dans ce cas, comme dans la définition du facteur phéromone, on peut soit utiliser séparement chaque fonction heuristique pour l'objectif correspondant, comme proposé dans [Doerner *et al.* , 2003, Gambardella *et al.* , 1999b], soit agréger les différentes structures heuristiques comme dans [Mariano & Morales, 1999, Iredi *et al.* , 2001, Barán & Schaerer, 2003, Angus, 2007].

Solutions à récompenser. Lors de la mise à jour de phéromone, on doit décider sur quelles solutions construites déposer la phéromone.

1. Une première possibilité, appelée l'approche Pareto, consiste à récompenser les solutions non-dominées de l'ensemble Pareto du cycle courant. Pour cette dernière possibilité on peut encore décider si on va récompenser toutes les solutions de l'ensemble Pareto comme proposé dans [Barán & Schaerer, 2003, Angus, 2007] ou bien seulement les nouvelles solutions non-dominées qui entrent dans l'ensemble au cycle courant comme dans [Iredi *et al.* , 2001].

2. Une deuxième possibilité, appelée l'approche non Pareto, est de récompenser chacune des solutions qui trouve la meilleure valeur pour chaque critère du cycle courant comme proposé dans [Doerner *et al.* , 2004, Gambardella *et al.* , 1999b, Doerner *et al.* , 2003, Mariano & Morales, 1999].

Algorithmes	**Structures phéromonales**	**Facteur phéromonal**	**Facteur heuristique**	**Solutions à récompenser**
CPACO [Angus, 2007]	Unique	Unique	Plusieurs, Agrégé	Pareto
ACOAMO [McMullen, 2001]	Unique	Unique	Unique	Non Pareto
MOACOM [Gravel *et al.* , 2002]	Unique	Unique	Unique	Non Pareto
MACS [Barán & Schaerer, 2003]	Unique	Unique	Plusieurs, Agrégé	Pareto
MOAQ [Mariano & Morales, 1999]	Unique	Unique	Plusieurs, Agrégé	Non Pareto
PACO [Doerner *et al.* , 2004]	Plusieurs	Plusieurs, Agrégé	Unique	Non Pareto
COMPETants [Doerner *et al.* , 2003]	Plusieurs	Plusieurs, Non agrégé	Plusieurs, Non agrégé	Non Pareto
MONACO [Cardoso *et al.* , 2003]	Plusieurs	Plusieurs, Agrégé	Unique	Non Pareto
BicriterionAnt [Iredi *et al.* , 2001]	Plusieurs	Plusieurs, Agrégé	Plusieurs, Agrégé	Pareto
MACS-VRPTW [Gambardella *et al.* , 1999b]	Plusieurs	Plusieurs, Non agrégé	Plusieurs, Non agrégé	Non Pareto

TAB. 4.1 – Taxonomie des algorithmes ACO pour la résolution de PMO

4.4 Discussion

Nous avons présenté plusieurs algorithmes MOACO proposés dans la littérature. Ces algorithmes proposent différentes stratégies pour la résolution des problèmes multi-objectifs. La plupart de ces algorithmes ne traitent que le cas bi-objectif. Ils utilisent des schémas ACO différents. De plus, ces algorithmes ont été conçus pour résoudre des problèmes spécifiques ce qui rend difficile leur comparaison. Il est nécessaire, pour pouvoir les comparer, de les unifier dans un même cadre en utilisant un même schéma ACO, les mêmes conditions d'expérimentations et les appliquer à un même problème multi-objectif.

4.5 Conclusion

Nous avons présenté dans ce chapitre plusieurs algorithmes MOACO de la littérature ainsi que nous avons proposé une taxonomie qui les classent suivant différents points. Nous avons remarqué, cependant, qu'il est difficile de comparer ces algorithmes. D'où, nous nous proposons de concevoir un algorithme générique paramétrable permettant de comparer différentes approches dans un même cadre. Nous proposons, d'abord, dans le prochain chapitre de comparer différentes stratégies phéromonales pour le problème du sac à dos multidimensionnel mono-objectif.

Deuxième partie

Etude de stratégies phéromonales

Chapitre 5

Stratégies phéromonales pour le problème du sac à dos multidimensionnel mono-objectif

5.1 Introduction

Dans ce chapitre, nous comparons trois stratégies différentes de dépôt et exploitation des traces de phéromone. Nous utilisons comme application un problème d'optimisation combinatoire uniobjectif connu dans la littérature NP-difficile : le problème du sac à dos multidimensionnel (MKP)que nous avons décrit dans le premier chapitre. Ainsi, nous décrivons dans la section suivante l'algorithme ACO proposé pour le MKP, et nous proposons trois variantes différentes de cet algorithme, correspondant aux trois stratégies phéromonales. Nous discutons ensuite de l'influence des paramètres de l'algorithme sur l'intensification/diversification de la recherche. Enfin, nous comparons expérimentalement les trois stratégies proposées, ainsi que d'autres approches évolutionnaires, sur un ensemble de problèmes issus d'un benchmark classique.

L'intérêt de ce travail dépasse le cadre de la résolution du problème du sac-à-dos multidimensionnel. En effet, de nombreux problèmes d'optimisation combinatoire consistent à sélectionner un sous-ensemble d'objets optimisant une fonction objectif donnée tout en respectant un certain nombre de contraintes, e.g., la recherche de cliques maximum, les problèmes de satisfaction de contraintes. Pour résoudre ces problèmes avec la métaheuristique ACO, il s'agit essentiellement de décider sur quels composants de solutions déposer de la phéromone. Ainsi, notre étude comparative sur le MKP pourrait aider à choisir une stratégie pour la résolution d'autres problèmes de ce type. L'essentiel des travaux réalisés dans ce chapitre est publié dans [Alaya *et al.* , 2007b] et [Alaya *et al.* , 2005].

5.2 Algorithme ACO pour le MKP

Dans les algorithmes ACO, les fourmis déposent les traces de phéromone sur les composants des meilleures solutions construites pour attirer les autres fourmis vers les aires correspondantes de l'espace de recherche. Ainsi pour résoudre un problème avec la métaheuristique ACO, un point clé est de décider sur quels composants des solutions construites les traces de phéromone vont être déposées et comment exploiter ces traces lors de la construction de nouvelles solutions. Une solution d'un MKP est un ensemble d'objets sélectionnés $S = \{o_1, \ldots, o_k\}$ (on considèrera qu'un objet o_i est sélectionné si la variable de décision correspondante x_{o_i} a été mise à 1). Etant donnée une telle solution S, trois différentes manières de déposer la phéromone peuvent être considérées :

- Une première possibilité est de déposer les traces de phéromone sur chaque objet sélectionné dans S. Dans ce cas, lors de la construction des solutions suivantes, la probabilité de sélectionner chaque objet de S sera augmentée.
- Une deuxième possibilité est de déposer les traces de phéromone sur chaque

couple (o_i, o_{i+1}) de deux objets successivement ajoutés dans S. Dans ce cas, l'idée est d'augmenter la probabilité de choisir l'objet o_{i+1} si le dernier objet sélectionné est o_i.

– Une troisième possibilité est de déposer les traces de phéromone sur toutes les paires (o_i, o_j) de deux objets appartenant à S. Dans ce cas, lors de la construction d'une nouvelle solution S', la probabilité de choisir l'objet o_i sera augmentée si o_j a déjà été sélectionné dans S'. Plus précisément, plus S' contiendra de sommets appartenant déjà à S, et plus les autres sommets de S auront de chances d'être sélectionnés.

Pour comparer ces trois stratégies phéromonales différentes, nous introduisons maintenant l'algorithme Ant-Knapsack (AK) et nous décrivons trois variantes différentes de cet algorithme : Vertex-AK dans lequel la phéromone est déposée sur les sommets, Path-AK dans lequel la phéromone est déposée sur les arcs du chemin visité, et Edge-AK dans lequel la phéromone est déposée sur les arêtes reliant tous les sommets appartenant à une solution.

5.2.1 L'algorithme Ant-Knapsack

L'algorithme Ant-Knapsack est décrit dans la Figure 5.1. A chaque cycle de cet algorithme, chaque fourmi construit une solution. Lorsque toutes les fourmis ont construit une solution, les traces de phéromone sont mises à jour. L'algorithme s'arrête lorsqu'une fourmi a trouvé une solution optimale (si la valeur optimale est connue), ou lorsqu'un nombre maximal de cycles a été atteint. Notons que cet algorithme est inspiré du $\mathcal{MAX} - \mathcal{MIN}$ Ant System [Stützle & Hoos, 2000] dans lequel les traces de phéromone sont limitées à l'intervalle $[\tau_{\min}, \tau_{\max}]$ et sont initialisées à $\tau_{\max}$.

Pour construire une solution, les fourmis choisissent aléatoirement un objet initial, puis ajoutent itérativement des objets qui sont choisis à partir d'un ensemble

Algorithme Ant-Knapsack :

Initialiser les traces de pheromone a τ_{max}

repeter

pour chaque fourmi k **dans** $1..nbAnts$, construire une solution $\mathcal{S}_k$ comme suit :

Choisir aleatoirement un premier objet $o_1 \in 1..n$

$\mathcal{S}_k \leftarrow \{o_1\}$

$Candidats \leftarrow \{o_i \in 1..n / o_i$peut être selectionne sans violer des contraintes de ressources$\}$

tant que $Candidats \neq \emptyset$ **faire**

Choisir un objet $o_i \in Candidats$ avec la probabilite

$$p_{S_k}(o_i) = \frac{[\tau_{S_k}(o_i)]^{\alpha}.[\eta_{S_k}(o_i)]^{\beta}}{\sum_{o_j \in Candidats}[\tau_{S_k}(o_j)]^{\alpha}.[\eta_{S_k}(o_j)]^{\beta}}$$

$\mathcal{S}_k \leftarrow \mathcal{S}_k \cup \{o_i\}$

enlever de $Candidats$ chaque objet qui viole des contraintes de ressources

fin tant que

fin pour

mettre a jour les traces de pheromone en fonction de $\{\mathcal{S}_1, \ldots, \mathcal{S}_{nbAnts}\}$

si une trace de pheromone est inferieure a τ_{min} **alors** la mettre a τ_{min}

si une trace de pheromone est superieure a τ_{max} **alors** la mettre a τ_{max}

jusqu'a nombre maximal de cycles atteint **ou** solution optimale trouvee

FIG. 5.1 – Algorithme ACO pour le MKP

Candidats qui contient tous les objets qui peuvent être sélectionnés sans violer de contraintes de ressources. A chaque étape, l'objet o_i à ajouter à la solution en cours de construction S_k est choisi parmi l'ensemble de sommets *Candidats* relativement à une probabilité $p_{S_k}(o_i)$. Cette probabilité est définie proportionnellement à un facteur phéromonal $\tau_{S_k}(o_i)$ et un facteur heuristique $\eta_{S_k}(o_i)$, ces deux facteurs étant pondérés par deux paramètres α et β qui déterminent leur importance relative. Le facteur phéromonal dépend de la stratégie phéromonale choisie et est décrit plus tard. Le facteur heuristique $\eta_{S_k}(o_i)$ est défini de façon similaire à [Leguizamon & Michalewicz, 1999] et [Alaya *et al.* , 2004a] : soit $d_{S_k}(i) = b_i - \sum_{g \in S_k} r_{ig}$ la quantité restante de la ressource i lorsque la fourmi a construit la solution S_k ; nous définissons le ratio

$$h_{S_k}(j) = \sum_{i=1}^{m} \frac{r_{ij}}{d_{S_k}(i)}$$

qui présente la dureté de l'objet j par rapport à toutes les contraintes $i \in 1..m$ et relativement à la solution construite S_k, de sorte que plus ce ratio est faible plus l'objet est intéressant. Nous intégrons alors le profit p_j de l'objet j pour obtenir un ratio profit/ressource, et nous définissons le facteur heuristique par

$$\eta_{S_k}(j) = \frac{p_j}{h_{S_k}(j)}$$

5.2.2 Définition des composants phéromonaux

Les fourmis de l'algorithme Ant-Knapsack évoluent et déposent de la phéromone sur le graphe complet $G = (V, E)$ tel que V est l'ensemble des objets. Cet algorithme s'instancie en trois versions différentes en fonction des composants du graphe sur lesquels les fourmis déposent des traces de phéromone.

– Dans Vertex-AK, les fourmis déposent la phéromone sur les sommets V du graphe. La quantité de phéromone sur un objet $o_i \in V$ est notée $\tau(o_i)$. Cette quantité représente la "désirabilité" de choisir l'objet o_i lors de la construction

d'une solution.

- Dans Path-AK, les fourmis déposent de la phéromone sur les couples de sommets sélectionnés consécutivement, i.e., sur les arcs du graphe. La quantité de phéromone sur un arc $(o_i, o_j) \in E$ est notée $\tau(o_i, o_j)$. Cette quantité représente la "désirabilité" de choisir l'objet o_i juste après avoir choisi l'objet o_j lors de la construction d'une solution. Il est à noter que dans ce cas le graphe est orienté.
- Dans Edge-AK, les fourmis déposent la phéromone sur les paires de sommets sélectionnés dans une même solution, i.e., sur les arêtes E du graphe. La quantité de phéromone sur une arête $(o_i, o_j) \in E$ est notée $\tau(o_i, o_j)$. Cette quantité représente la désirabilité de choisir un objet o_i lors de la construction d'une solution qui contient déjà l'objet o_j. Il est à noter que, dans ce cas le graphe est non orienté, et donc $\tau(o_i, o_j) = \tau(o_j, o_i)$.

5.2.3 Définition du facteur phéromonal

Le facteur phéromonal $\tau_{S_k}(o_i)$ traduit l'expérience passée de la colonie et est utilisé dans la probabilité de transition des fourmis pour les guider vers des zones de l'espace de recherche prometteuses. Ce facteur phéromonal dépend de la quantité de phéromone déposée sur les composants phéromonaux du graphe :

- Dans Vertex-AK, il dépend de la quantité déposée sur l'objet candidat, i.e.,

$$\tau_{S_k}(o_i) = \tau(o_i)$$

- Dans Path-AK, il dépend de la quantité présente sur l'arc connectant le dernier objet ajouté à la solution partielle S_k et l'objet candidat o_i, i.e., si le dernier objet ajouté dans S_k est o_j,

$$\tau_{S_k}(o_i) = \tau(o_j, o_i)$$

- Dans Edge-AK, il dépend de la quantité déposée sur les arêtes connectant l'objet candidat o_i avec les différents objets présents dans la solution partielle

S_k, i.e.,

$$\tau_{S_k}(o_i) = \sum_{o_j \in S_k} \tau(o_i, o_j)$$

Notons que ce facteur phéromonal peut être calculé de façon incrémentale : lorsque le premier objet o_1 a été sélectionné, le facteur phéromonal $\tau_{S_k}(o_j)$ est initialisé à $\tau(o_1, o_j)$ pour chaque objet candidat o_j ; ensuite, à chaque fois qu'un nouvel objet o_i est ajouté à la solution courante S_k, le facteur phéromonal $\tau_{S_k}(o_j)$ de chaque objet candidat o_j est incrémenté de $\tau(o_i, o_j)$.

5.2.4 Mise à jour de la phéromone

Après que toutes les fourmis aient fini la construction de leurs solutions, les traces de phéromone sont mises à jour. Cette mise-à-jour s'effectue en deux étapes. Dans une première étape, toutes les traces de phéromone sont diminuées, pour simuler l'évaporation, en multipliant chaque composant phéromonal par un ratio de persistance $(1 - \rho)$ tel que $0 \leq \rho \leq 1$. Notons que dans Vertex-AK, les composants phéromonaux sont associés aux objets de sorte que l'étape d'évaporation s'effectue en $\mathcal{O}(n)$ tandis que dans Path-AK et Edge-AK, les composants phéromonaux sont associés aux couples d'objets de sorte que l'étape d'évaporation s'effectue en $\mathcal{O}(n^2)$.

Dans un deuxième temps, la meilleure fourmi du cycle dépose de la phéromone. Plus précisément, soit $\mathcal{S}_k \in \{\mathcal{S}_1, \ldots, \mathcal{S}_{nbAnts}\}$ la meilleure solution (celle ayant un profit maximal, les ex-aequo étant départagés aléatoirement) construite durant le cycle courant et $\mathcal{S}_{best}$ la meilleure solution construite depuis le début de l'exécution (y compris le cycle courant). La quantité de phéromone déposée par la fourmi k est inversement proportionnelle à la différence de profit entre $\mathcal{S}_k$ et $\mathcal{S}_{best}$, i.e., elle est égale à $1/(1 + \text{profit}(S_{\text{best}}) - \text{profit}(S_k))$. Cette quantité de phéromone est déposée sur les composants phéromonaux de $\mathcal{S}_k$, i.e.,

– dans Vertex-AK, elle est déposée sur chaque sommet de $\mathcal{S}_k$,

– dans Path-AK, elle est déposée sur les arcs du chemin visité par la fourmi k lors de la construction de $\mathcal{S}_k$, i.e., sur tout couple de sommets (o_i, o_j) tel que o_j a été ajouté dans S_k juste après o_i.
– dans Edge-AK, elle est déposée sur chaque arête reliant deux sommets différents de $\mathcal{S}_k$, i.e., sur la clique définie par $\mathcal{S}_k$.

Comme pour la première étape, la complexité en temps de cette deuxième étape dépend de la version considérée : pour Vertex-AK et Path-AK elle s'effectue en $\mathcal{O}(|S_k|)$ tandis que pour Edge-AK elle s'effectue en $\mathcal{O}(|S_k|^2)$

5.3 Influence des paramètres α et ρ sur la résolution

Quand on résoud un problème d'optimisation combinatoire avec une approche heuristique, il s'agit de trouver un bon compromis entre deux objectifs relativement duaux : d'une part il s'agit d'intensifier la recherche autour des zones de l'espace de recherche les plus prometteuses, qui sont généralement proches des meilleures solutions trouvées ; d'autre part il s'agit de diversifier la recherche et favoriser l'exploration afin de découvrir de nouvelles et si possible meilleures zones de l'espace de recherche. Le comportement des fourmis par rapport à cette dualité entre intensification et diversification peut être influencé en modifiant les valeurs des paramètres. En particulier, la diversification peut être augmentée soit en diminuant la valeur du poids du facteur phéromonal α (de sorte que les fourmis deviennent moins sensibles aux traces phéromonales), soit en diminuant le taux d'évaporation ρ (de sorte que la phéromone s'évapore plus doucement et les écarts d'une trace à l'autre évoluent plus doucement). Lorsque l'on augmente ainsi la capacité exploratoire des fourmis, on trouve généralement de meilleures solutions, mais en contrepartie ces solutions sont plus longues à trouver.

Nous avons pu constater que cette influence des paramètres α et ρ sur le compor-

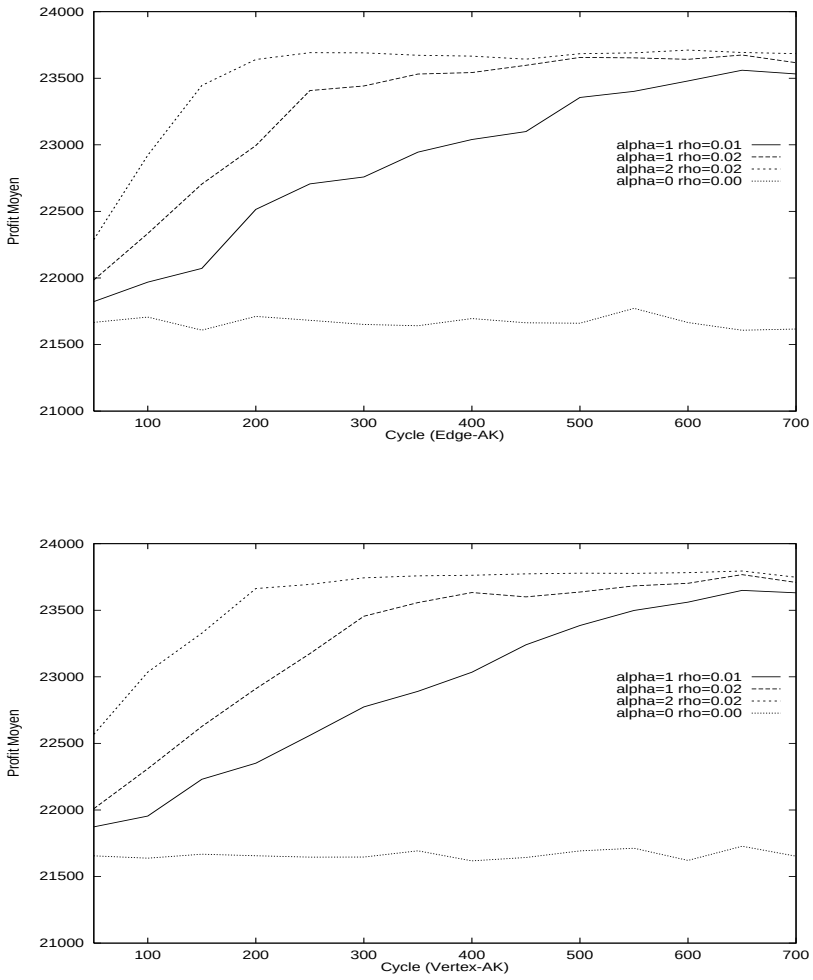

FIG. 5.2 – Influence de α et ρ sur la qualité des solutions trouvées par Edge-AK et Vertex-AK pour une instance du MKP ayant 100 objets et 5 contraintes de ressource

tement des fourmis est identique pour les trois versions proposées de Ant-Knapsack. Nous l'illustrons dans la Figure 5.2 pour Edge-AK et Vertex-AK. Dans cette figure, chaque courbe trace l'évolution du profit moyen quand le nombre de cycles augmente pour une affectation donnée de α et ρ (moyenne sur 10 exécutions). Les autres paramètres ont été affectés à $\beta = 5$, $nbAnts = 30$, $\tau_{min} = 0.01$, and $\tau_{max} = 6$. Cette figure montre que quand on augmente l'intensification, en choisissant des valeurs telles que $\alpha = 2$ et $\rho = 0.02$, Edge-AK trouve rapidement de bonnes solutions mais stagne plus vite sur des solutions légèrement moins bonnes. A l'inverse, en choisissant des valeurs pour α et ρ qui favorisent l'exploration, telles que $\alpha = 1$ et $\rho = 0.01$, les fourmis trouvent de meilleures solutions en fin d'exécution, mais elles ont besoin de plus de cycles pour converger sur ces solutions.

Notons finalement que lorsque la phéromone n'est pas utilisée du tout, i.e., quand $\alpha = 0$ et $\rho = 0$, de sorte que l'algorithme se comporte comme un algorithme glouton classique, les solutions trouvées sont très nettement moins bonnes que lorsque la phéromone est utilisée.

5.4 Influence des traces de phéromone sur la similarité des solutions calculées

Pour expliciter l'influence de la phéromone sur la capacité des fourmis à explorer l'espace de recherche, nous proposons dans cette section de mesurer la similarité des solutions construites durant l'exécution. Plusieurs mesures de diversité ou de similarité ont été introduites pour les approches évolutionnaires, le maintien de la diversité de la population étant un point clé pour éviter une convergence prématurée et une stagnation de ces algorithmes.

Pour mesurer l'effort de diversification de Ant-Knapsack pendant l'exécution, nous proposons de calculer le taux de similarité. Ce taux correspond à la mesure

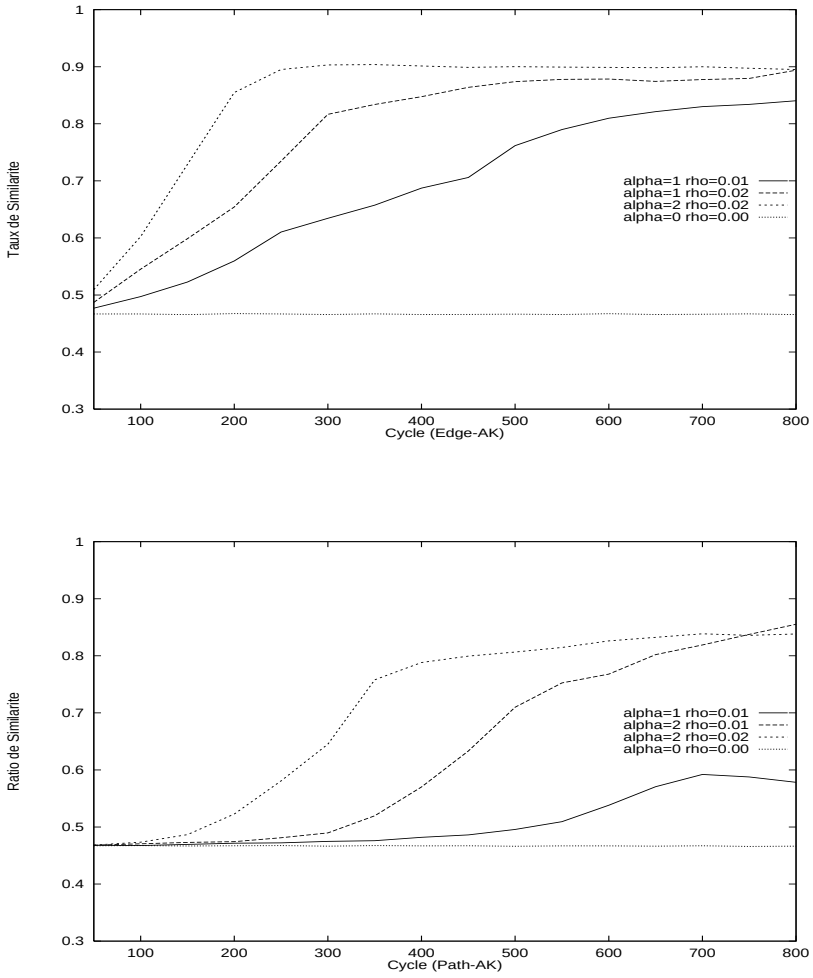

FIG. 5.3 – Influence de α et ρ sur la similarité des solutions trouvées par Edge-AK et Path-AK pour une instance du MKP ayant 100 objets et 5 contraintes de ressource.

de diversité de la population introduite pour les approches génétiques [Morrison & DeJong, 2001]. Plus précisément, le taux de similarité d'un ensemble de solutions S est défini par le nombre moyen d'objets qui sont partagés par n'importe quelle paire de solutions dans S, divisé par le nombre moyen d'objets sélectionnés dans les solutions de S. Ainsi, ce taux est égal à un si toutes les solutions de S sont identiques, tandis qu'il est égal à zéro si l'intersection de chaque paire de solutions de S est vide.

Il est à noter que ce ratio peut être calculé très rapidement en maintenant un tableau freq tel que, pour chaque noeud $v_i \in V$, freq[i] est égal au nombre de solution de S qui ont sélectionné le noeud v_i. Dans ce cas, le ratio de similarité de S est égal à $\frac{\sum_{v_i \in V}(freq[i].(freq[i]-1))}{(|S|-1).\sum_{S_k \in S}|S_k|}$ et il peut être calculé facilement de manière incrémentale en construisant les solutions.

Dans les algorithmes ACO, ce sont les traces de phéromone qui dirigent les fourmis vers les zones "prometteuses" de l'espace de recherche. La figure 5.3 montre l'influence de ces traces sur la similarité des solutions construites durant l'exécution de l'algorithme Ant-Knapsack, pour les deux variantes Path-AK et Edge-AK, et pour différentes valeurs des paramètres α et ρ qui déterminent l'influence de la phéromone sur le comportement des fourmis. Dans cette figure, chaque courbe trace l'évolution du ratio de similarité des solutions construites quand le nombre de cycles augmente pour une affectation donnée de α et ρ (moyenne sur 10 exécutions). Les autres paramètres ont été affectés à $\beta = 5$, $nbAnts = 30$, $\tau_{min} = 0.01$, and $\tau_{max} = 6$. Ces courbes mettent en évidence trois phases dans les exécutions de Ant-Knapsack.

Durant les premiers cycles, on observe une phase d'*exploration*, où les fourmis calculent des solutions assez différentes (les solutions construites ont 42% d'objets en commun en moyenne). La durée de cette phase d'exploration dépend du paramétrage considéré : elle est d'autant plus longue que l'on réduit l'influence de la phéromone en diminuant α et ρ. La durée de cette phase d'exploration dépend aussi de la

stratégie phéromonale considérée : on observe qu'elle est plus longue pour Path-AK que pour Edge-AK. Notons que cette phase d'exploration initiale est accentuée par la stratégie du $\mathcal{MAX} - \mathcal{MIN}$ Ant System [Stützle & Hoos, 2000], qui impose des bornes minimales et maximales τ_{min} et τ_{max} aux traces de phéromone, et initialise ces traces à τ_{max} au début de l'exécution : ainsi, après n cycles de Ant-knapsack, la trace de phéromone τ_i sur un composant phéromonal (arête ou sommet) i est telle que $(1-\rho) \cdot \tau_{max} \leq \tau_i \leq \tau_{max}$, et comme $(1-\rho)$ est généralement très proche de 1, les traces de phéromone ont des valeurs très similaires durant les premiers cycles et n'influencent pas significativement le calcul des probabilités de transition.

On observe ensuite une deuxième phase de *convergence*, où la similarité des solutions construites augmente progressivement, mettant en évidence le fait que les fourmis sont de plus en plus influencées par la phéromone et intensifient l'effort de recherche autour d'une zone plus petite de l'espace de recherche : la similarité des solutions construites augmente de 42% à près de 90%. Notons que l'augmentation de la similarité est d'autant plus rapide que l'influence de la phéromone est forte.

On observe enfin une troisième phase de *stagnation*, où la similarité des solutions construites se stabilise autour d'une valeur assez élevée (entre 80% et 90%), montrant que les fourmis se sont focalisées sur une toute petite zone de l'espace de recherche qu'elles explorent intensément.

5.5 Expérimentations et résultats

Nous présentons dans cette section les résultats des trois variantes de l'algorithme Ant-Knapsack, i.e., Vertex-AK, Path-AK et Edge-AK. Nous comparons aussi ces résultats avec ceux de deux algorithmes ACO qui ont été proposés dans la littérature et qui utilise chacun une stratégie phéromonale différente. L'algorithme de Leguizemon et Michalewicz [Leguizamon & Michalewicz, 1999] utilise la même stratégie

que Vertex-AK et l'algorithme de Fidanova [Fidanova, 2002] qui utilise la même stratégie que Path-AK. Nous comparons enfin ces résultats avec deux approches de l'état de l'art sur le sujet qui trouvent parmi les meilleurs résultats connus pour ces instances.

5.5.1 Ensemble de tests et conditions d'expérimentation

Nous avons considéré des instances larges de MKP OR-Library accessibles à partir du site `http ://www.brunel.ac.uk/depts/ma/research/jeb/info.html`. Ces instances sont groupées en 5 classes avec $m \in \{5, 10, 30\}$ et $n \in \{100, 250\}$. Dans la suite, ces classes sont notées m.n. Chaque classe contient 30 instances groupées en 3 sous-classes de 10 instances en fonction d'un ratio de dureté rd : $rd = 0,25$ pour les 10 premières instances, $rd = 0,5$ pour les 10 instances suivantes, et $rd = 0,75$ pour les 10 dernières instances.

Tous les algorithmes ont été implémentés en Visual C++ et exécutés sur un processeur PIV 2GH. Dans toutes les expérimentations, nous avons mis α à 1, β à 5, ρ à 0,01, le nombre de fourmis à 30 et les bornes de phéromone τ_{min} et τ_{max} à $0,01$ et 6. Nous avons limité le nombre de cycles à 2000 cycles.

5.5.2 Comparaison des algorithmes Vertex-AK, Path-AK et Edge-AK

Le tableau 5.1 compare la qualité des résultats obtenus par Vertex-AK, Path-AK et Edge-AK sur les 15 classes de problèmes considérées. Chaque classe contient 10 instances, et chaque algorithme a été exécuté 10 fois sur chaque instance, de sorte que chaque résultat est une moyenne suivi d' un écart-type sur 100 exécutions. Dans ce tableau, chaque ligne donne les résultats pour les 10 instances de la classe `m.n` ayant le ratio de dureté `rd`. Pour chacune de ces classes et pour chaque algorithme

considéré, le tableau donne le profit trouvé.

Ce tableau montre que les deux algorithmes Vertex-AK et Edge-AK trouvent des résultats très nettement meilleurs que ceux trouvés par Path-AK, pour toutes les classes d'instances considérées. Ainsi, la stratégie consistant à déposer de la phéromone sur des objets consécutivement sélectionnés guide les fourmis de manière moins profitable que les deux autres stratégies.

En comparant Vertex-AK et Edge-AK, nous constatons que les solutions trouvées par Edge-AK sont meilleures pour 10 classes d'instances et inférieures pour les 5 autres classes. Notons que ces résultats dépendent de la taille des instances (par rapport au nombre de variables et de contraintes) : pour les instances des classes 5.100, 5.250 et 10.100, Edge-AK est clairement meilleur que Vertex-AK tandis que pour les instances des classes 10.250 et 30.100 cette tendance s'inverse.

Le tableau 5.2 compare les temps d'exécution et le nombre de cycles effectués par Vertex-AK, Path-AK et Edge-AK sur les 15 classes de problèmes considérées. Dans ce tableau, chaque ligne donne les résultats pour les 10 instances de la classe `m.n` ayant le ratio de dureté `rd`. Pour chacune de ces classes et pour chaque algorithme considéré, le tableau donne le temps (en secondes) et le nombre de cycles nécessaires pour trouver la solution (moyenne (Moy) et écart-type (Ec) sur 10 exécutions de chaque instance de la classe). On constate que les résultats pour les trois versions sont du même ordre de grandeur. Path-AK effectue légèrement moins de cycles et est un peu plus rapide que les deux autres versions, mais cela s'explique probablement par le fait qu'il trouve de bien moins bonnes solutions : pour Path-AK, la phéromone n'est pas capable de guider la recherche vers de bonnes zones de l'espace de recherche.

On constate que les nombres de cycles effectués par Vertex-AK et Edge-AK sont très similaires. Ce nombre dépend essentiellement du nombre de variables `n`, tandis que le nombre de contraintes `m` et la dureté de ces contraintes `rd` ne semblent pas avoir une influence directe.

m.n	rd	**Vertex-AK**		**Path-AK**		**Edge-AK**	
		Moyenne	Ecart	Moyenne	Ecart	Moyenne	Ecart
5.100	0,25	24192,0	33,13	24139,3	149,52	**24197,2**	37,52
5.100	0,50	43243,3	24,89	43137,9	137,11	**43246,4**	35,87
5.100	0,75	60470,5	25,37	60435,7	68,86	**60471,0**	28,35
5.250	0,25	**60335,9**	56,09	59110,6	911,77	60334,9	50,16
5.250	0,50	109203,4	39,09	107818	554,38	**109231,6**	47,97
5.250	0,75	151536,6	36,93	150578,6	152,68	**151536,9**	37,86
10.100	0,25	22553,0	60,97	22418,2	193,32	**22572,2**	61,81
10.100	0,50	42641,0	50,60	42383,0	192,37	**42658,5**	60,97
10.100	0,75	59540,0	34,71	59464,9	89,42	**59555,6**	44,70
10.250	0,25	**58847,6**	124,66	57367,3	593,35	58826,0	93,55
10.250	0,50	**108600,7**	109,50	106834,2	506,34	108594,1	78,03
10.250	0,75	151272,5	51,03	150066,9	170,66	**151276,0**	55,46
30.100	0,25	**21618,1**	73,33	21411,4	213,22	21608,4	75,68
30.100	0,50	**41406,1**	69,05	41013,0	228,18	41403,3	68,53
30.100	0,75	59172,5	40,56	59049,9	111,09	**59173,2**	36,67
Moyenne		67642,2	55,33	67015,26	284,8	**67645,7**	54,21

TAB. 5.1 – Comparaison de la qualité des solutions trouvées par Vertex-AK, Path-AK et Edge-AK sur les 15 classes de problèmes considérées.

m.n	rd	**Vertex-AK**				**Path-AK**				**Edge-AK**			
		Temps		Nb cycles		Temps		Nb cycles		Temps		Nb cycles	
		Moy	Ec	Moy	Ec	Moy	Ec	Moy	Ec	Moy	Ec	Moy	Ec
5.100	0,25	20	25	619	219	24	9	759	296	25	19	496	209
5.100	0,50	38	50	585	215	44	18	739	321	47	39	490	157
5.100	0,75	37	49	381	142	63	28	713	315	56	60	333	125
5.250	0,25	327	111	1062	357	182	97	720	384	343	89	1237	321
5.250	0,50	715	184	1115	287	438	238	779	426	807	202	1321	335
5.250	0,75	1044	304	1038	276	745	392	828	437	1185	328	1197	333
10.100	0,25	28	12	751	335	26	12	756	366	21	11	614	325
10.100	0,50	52	22	630	261	49	25	699	351	43	23	508	297
10.100	0,75	56	24	489	224	79	41	766	401	37	24	358	242
10.250	0,25	373	172	1054	364	267	165	806	428	413	129	1167	360
10.250	0,50	813	276	1151	390	489	274	743	415	971	235	1315	317
10.250	0,75	1232	396	1132	364	840	514	812	499	1364	360	1230	318
30.100	0,25	234	250	808	325	93	81	774	309	316	248	810	372
30.100	0,50	284	291	749	302	113	56	769	386	400	213	797	310
30.100	0,75	262	292	515	188	162	78	853	428	437	267	587	251
Moyenne		368	164	805	283	241	135	768	384	431	150	831	285

TAB. 5.2 – Comparaison des temps d'exécution de Vertex-AK, Path-AK et Edge-AK sur les 15 classes de problèmes considérées.

Enfin, on constate qu'un cycle de Vertex-AK est effectué légèrement plus rapidement qu'un cycle de Edge-AK. Par exemple, pour la classe 30.100 avec $rd = 0,25$, les deux algorithmes ont besoin en moyenne de 810 cycles pour converger ; Vertex-AK met 234 secondes pour cela tandis que Edge-AK met 316 secondes. Cette différence, qui est d'autant plus sensible que le nombre de variables `n` augmente, s'explique par le fait que l'évaporation est en $\mathcal{O}(n^2)$ pour Edge-AK alors qu'elle est en $\mathcal{O}(n)$ pour Vertex-AK.

5.5.3 Comparaison avec d'autres algorithmes ACO

L'algorithme Ant-Knapsack s'instancie en trois versions Vertex-AK, Path-AK et Edge-AK en fonction de la stratégie phéromonale choisie. Les instances Vertex-AK et Path-AK adoptent les stratégies phéromonales respectivement proposées dans [Leguizamon & Michalewicz, 1999] et [Fidanova, 2002], mais présentent quelques différences par ailleurs. Nous comparons dans cette section les résultats obtenus par ces différents algorithmes ACO pour le MKP.

Comparaison de Vertex-AK avec l'algorithme de Leguizamon et Michalewizc

L'algorithme Vertex-AK utilise la même stratégie phéromonale que l'algorithme proposé par Leguizamon et Michalewizc dans [Leguizamon & Michalewicz, 1999]. Ces deux algorithmes diffèrent cependant sur les deux points suivants :

- dans l'algorithme de Leguizamon et Michalewizc, la quantité de phéromone déposée est égale au profit de la solution construite tandis que dans Vertex-AK elle est inversement proportionnelle à l'écart de profit avec la meilleure solution trouvée ;
- l'algorithme de Leguizamon et Michalewiz est basé sur le schéma ACO "Ant-System" proposé initialement dans [Dorigo *et al.* , 1996] tandis que Vertex-AK

est basé sur le schéma ACO "MAX-MIN Ant System" proposé dans [Stützle & Hoos, 2000].

Pour les deux algorithmes, les paramètres α et β, qui déterminent l'importance relative du facteur heuristique et du facteur phéromonal, ont été fixés aux mêmes valeurs (i.e., $\alpha = 1$ et $\beta = 5$). En revanche, les résultats présentés dans [Leguizamon & Michalewicz, 1999] ont été obtenus avec un taux d'évaporation ρ fixé à 0.3 (au lieu de 0.01 pour Vertex-AK), un nombre de fourmis égal au nombre d'objets, i.e., n (au lieu de 30 pour Vertex-AK) et un nombre de cycles fixé à 100 (au lieu de 2000 pour Vertex-AK).

Sur les instances des classes 5.100 et 10.100 dont le ratio de dureté est $rd = 0,25$, l'algorithme de Leguizamon et Michalewiz obtient des profits de 24144 et 22457 respectivement (moyenne sur 10 exécutions pour chacune des 10 instances de la classe considérée). Si l'on compare ces résultats avec ceux de Vertex-AK (qui obtient 24192 et 22553 sur ces deux classes), on constate que Vertex-AK obtient de meilleurs résultats. Notons toutefois que sur ces instances chaque exécution de l'algorithme de Leguizamon et Michalewiz calcule $100*100 = 10000$ solutions tandis que Vertex-AK en calcule $30 * 2000 = 60000$.

Comparaison de Path-AK avec l'algorithme de Fidanova

L'algorithme Path-AK utilise la même stratégie phéromonale que l'algorithme proposé par Fidanova dans [Fidanova, 2002]. Ces deux algorithmes diffèrent cependant sur les même points que Vertex-AK par rapport à l'algorithme de Leguizamon et Michalewizc, i.e., sur la quantité de phéromone déposée et sur le schéma ACO considéré. De plus, l'algorithme de Fidanova renforce la meilleure solution trouvée par une quantité de phéromone additionnelle. L'information heuristique dans cet algorithme est calculée de façon statique, i.e., sans prendre en compte les quantités de ressources déjà consommées par la solution en cours de construction.

Sur les instances de la classe 5.100 avec les valeurs du ratio de dureté $rd = 0,25$ $rd = 0,5$ et $rd = 0,75$, l'algorithme de Fidanova obtient des profits moyens respectivement de 23939 43037 et 60373. Si l'on compare ces résultats avec ceux de Path-AK (qui obtient 24139 43137 et 60435), on constate que Path-AK obtient de meilleurs résultats. Notons toutefois que sur ces instances chaque exécution de l'algorithme de Fidanova calcule $200 * 100 = 20000$ solutions tandis que Path-AK en calcule $30 * 2000 = 60000$.

Comparaison de Edge-AK avec les deux algorithmes ACO

On compare maintenant les algorithmes Edge-AK, l'algorithme de Leguizamon et Michalewizc et celui de Fidanova, qui utilisent chacun une stratégie phéromonale différente. Nous trouvons que Edge-AK trouve des résultats qui sont nettement meilleurs que ceux trouvés par l'algorithme de Fidanova sur toutes les instances testées : pour la classe d'instances 5.100 l'algorithme de Fidanova trouve un profit de 42449 (moyenne des 30 instances testées) tandis que Edge-AK trouve un profit de 42638. Edge-AK trouve aussi des résultats meilleurs que ceux reportés par l'algorithme de Leguizamon et Michalewizc sur la plupart des instances testées : sur les instances des classes 5.100 et 10.100 dont le ratio de dureté est $rd = 0,25$, l'algorithme de Leguizamon et Michalewiz obtient des profits de 24144 et 22457, alors que Edge-AK trouve des profits de 24197 et 60471 respectivement.

5.5.4 Comparaison avec d'autres approches

On compare maintenant Edge-AK, qui est la version de Ant-Knapsack qui obtient les meilleurs résultats en moyenne avec une approche proposée par Chu et Beasley [Chu & Beasley, 1998] et une approche proposée par Gottlieb [Gottlieb, 2000]. Ces deux approches combinent un algorithme génétique avec des techniques de réparation (pour transformer une solution violant des contraintes en une solu-

tion cohérente) et des techniques de recherche locale (pour améliorer la qualité des solutions cohérentes). Dans les deux approches la méthode de remplacement utilisée est une méthode "steady state", la méthode de sélection est le tournoi binaire standard. Les deux approches ont une population de 100 individus et stoppent une exécution lorsque 10^6 solutions différentes ont été générées. Les différences entre les deux approches concernent essentiellement l'opérateur de croisement et la procédure de génération de la population initiale. Il est à noter que les résultats trouvés par ces deux approches pour la plupart des instances considérées sont des valeurs optimales (pour les instances avec des valeurs optimales connues).

Le tableau 5.3 compare Ant-knapsack avec ces deux approches sur 6 classes de 10 instances du MKP. Pour chaque classe, on donne l'écart moyen entre la solution sur les réels et la solution trouvée par Edge-AK, Chu et Beasley (`CB`), et Gottlieb (`G`). Pour Edge-AK et Gottlieb, on reporte également le nombre moyen de solutions construites pour trouver la meilleure solution. Dans ce tableau, la qualité des solutions est mesurée par l'écart (en pourcentage) entre le profit de la meilleure solution trouvée et la valeur optimale de la relaxation du problème linéaire sur les réels, i.e., $Ecart = 1 - Best/opt^{LP}$ où opt^{LP} est la valeur optimale du problème linéaire sur les réels et $Best$ est la meilleure solution trouvée par l'algorithme considéré.

Si l'on considère la qualité des solutions trouvées, on remarque que Edge-AK obtient les même résultats que `CB` sur quatre classes et les même résultats que `G` sur trois classes. Il obtient de meilleurs résultats que `G` sur la classe 10.100 avec $rd = 0,5$. Sur les deux autres classes, il obtient des résultats légèrement moins bons. Notons toutefois que les deux algorithmes génétiques considérés utilisent des techniques de recherche locale pour améliorer la qualité des solutions construites, ce qui n'est pas le cas de Edge-AK. De plus, quand on considère le nombre de solutions construites, on remarque que Edge-AK construit au plus soixante mille solutions tandis que les deux algorithmes génétiques en construisent au plus un million. Le tableau 5.3

Classe			CB	G		Edge-AK	
m	n	rd	écart	écart	#sol	écart	#sol
5	100	0.25	0.99	0.99	74595	0.99	15736
		0.50	0.45	0.45	37353	0.47	16053
		0.75	0.32	0.32	40690	0.32	11131
Moyenne pour 5.100			0.587	0.587	50879	0.59	14306
10	100	0.25	1.56	1.60	190979	1.69	17874
		0.50	0.79	0.81	109036	0.79	17147
		0.75	0.48	0.48	53528	0.48	15137
Moyenne pour 10.100			0.943	0.965	117848	0.96	16919

TAB. 5.3 – Comparaison de Ant-knapsack avec deux approches évolutionnaires sur 6 classes de 10 instances du MKP.

montre que G construit toujours plus de solutions que Ant-Knapsack.

5.6 Conclusion

Nous avons proposé dans ce chapitre un algorithme ACO générique pour résoudre le problème du sac à dos multidimensionnel. Nous avons proposé trois versions différentes de cet algorithme suivant les composants sur lesquels les traces de phéromone sont déposées : Vertex-AK où la phéromone est déposée sur les objets, Path-AK où la phéromone est déposée sur les couples d'objets sélectionnés consécutivement, et Edge-AK où la phéromone est déposée sur les paires d'objets sélectionnés dans une même solution. Un objectif principal était de comparer ces trois stratégies phéromonales, indépendamment des détails d'implémentation et du schéma ACO considéré.

Les expérimentations ont montré que Path-AK obtient de bien moins bons résultats que Vertex-AK et Edge-AK : pour ce problème, l'ordre dans lequel les objets

sont sélectionnés n'est pas significatif, et il n'est clairement pas intéressant de ne considérer que le dernier objet sélectionné pour choisir le prochain objet à entrer dans le sac à dos. Les expérimentations ont également montré que Edge-AK obtient de meilleurs résultats que Vertex-AK sur deux tiers des classes d'instances considérées. Cependant, les différences de résultats entre ces deux versions ne sont pas aussi importantes qu'avec Path-AK. De plus, Edge-AK prend plus de temps CPU que Vertex-AK lors de l'exécution. Ceci est dû à la phase de mise à jour de phéromone qui est de complexité quadratique pour Edge-AK alors qu'elle est de complexité linéaire pour Vertex-AK.

Comme nous l'avons souligné dans l'introduction de ce chapitre, nous pensons que cette étude expérimentale de différentes stratégies phéromonales dépasse le cadre du MKP, et peut être utile pour la résolution d'autres problèmes de "recherche d'un meilleur sous-ensemble" à l'aide de la métaheuristique ACO.

En comparant Edge-AK avec d'autres algorithmes ACO de l'état de l'art, qui utilisent des stratégies phéromonales différentes, on trouve qu'il donne de meilleurs résultats pour toutes les instances considérées.

Lorsque l'on compare les résultats obtenus par Edge-AK avec d'autres approches de l'état de l'art, qui trouvent parmi les meilleurs résultats connus pour les instances testées, on constate qu'ils sont globalement compétitifs même s'ils sont légèrement moins bons sur quelques instances, puisqu'il retrouvent la plupart de ces résultats. Notons toutefois qu'Edge-AK n'utilise pas de technique de recherche locale pour améliorer les solutions construites par les fourmis.

Chapitre 6

Un algorithme ACO générique pour la résolution de problèmes multi-objectifs

6.1 Introduction

Pour résoudre un problème d'optimisation multi-objectif (PMO) avec ACO, plusieurs points sont à définir pour lesquels différents choix sont possibles. Nous avons étudié dans le chapitre 4 plusieurs algorithmes MOACO qui ont été proposés dans la littérature pour résoudre des PMO. Nous avons établi une taxonomie de ces algorithmes suivant différents critères de classement.

Dans ce chapitre, nous présentons un algorithme ACO générique, appelé m-ACO, pour la résolution de PMO, et plus particulièrement les problèmes de sélection de sous-ensembles. Cet algorithme générique nous permet de mettre dans un même cadre différentes approches que nous pouvons tester et comparer indépendamment des détails d'implémentation et du schéma ACO considéré. Nous instancions l'alorithme m-ACO par six variantes : nous proposons trois nouvelles approches et nous

reprenons le schéma général de trois autres algorithmes MOACO présentés dans la littérature. Il est à noter que nous considérons, dans cet algorithme, les problèmes de maximisation (dans le cas de minimisation il suffit de multiplier la fonction objectif par -1). Une partie des travaux présentés dans ce chapitre est publiée dans [Alaya *et al.* , 2007a]. Une version plus étendue de cet article, reprenant plus largement les résultats de ce chapitre et du chapitre suivant est actuellement en cours de soumission à un journal international.

6.2 L'algorithme générique m-ACO

Dans cette section, nous présentons un algorithme ACO générique pour des problèmes multiobjectifs. Cet algorithme sera instancié par six variantes qui seront décrites par la suite.

L'algorithme générique, appelé m-ACO, est implicitement paramétré par le problème multi-objectif à résoudre. Nous considérons que les fourmis construisent les solutions dans un graphe $G = (V, E)$ dont la définition dépend du problème à résoudre et que les traces de phéromone sont associées aux sommets ou arêtes de ce graphe.

m-ACO est aussi paramétré par le nombre de colonies de fourmis $\#Col$ et le nombre de structures de phéromone considérées $\#\tau$. La figure 6.1 décrit l'algorithme générique de m-ACO($\#Col$,$\#\tau$). L'algorithme suit le schéma de $\mathcal{MAX}-\mathcal{MIN}$ Ant System [Stützle & Hoos, 2000]. Les traces de phéromone sont initialisées d'abord à une borne supérieure donnée τ_{max}. Par la suite, à chaque cycle chaque fourmi construit une solution et les traces de phéromone sont mises à jour. Pour éviter une convergence prématurée, les traces de phéromone sont bornées avec deux bornes τ_{min} et τ_{max} telles que $0 < \tau_{min} < \tau_{max}$. L'algorithme s'arrête lorsqu'un nombre maximum de cycles est atteint.

Algorithme m-ACO($\#Col$,$\#\tau$) :

Initialiser les traces de pheromone a τ_{max}

repeter

pour chaque colonie c **dans** 1..$\#Col$

pour chaque fourmi k **dans** 1..nbf

construire une solution

pour i **dans** 1..$\#\tau$

mettre a jour la $i^{ème}$ structure de traces de pheromone

si une trace est inferieure a τ_{min} alors la mettre a τ_{min}

si une trace est superieure a τ_{max} alors la mettre a τ_{max}

jusqu'a un nombre maximal de cycles atteint

FIG. 6.1 – Algorithme ACO generique pour POM

6.2.1 Construction de solutions

La figure 6.2 décrit l'algorithme utilisé par les fourmis pour construire des solutions dans le graphe de construction $G = (V, E)$ dont la définition dépend du problème à résoudre. A chaque itération, un sommet de G est choisi parmi un ensemble de sommets candidats $Cand$; il est ajouté à la solution S et l'ensemble des sommets candidats est mis à jour en supprimant les sommets qui violent des contraintes de C. Le sommet v_i à ajouter à la solution S par les fourmis de la colonie c est choisi relativement à la probabilité $p_S^c(v_i)$ définie comme suit :

$$p_S^c(v_i) = \frac{[\tau_S^c(v_i)]^\alpha . [\eta_S^c(v_i)]^\beta}{\sum_{v_j \in Cand} [\tau_S^c(v_j)]^\alpha . [\eta_S^c(v_j)]^\beta}$$

où $\tau_S^c(v_i)$ et $\eta_S^c(v_i)$ sont respectivement les facteurs phéromonal et heuristique du sommet candidat v_i, et α et β sont deux paramètres qui déterminent leur importance relative. La définition du facteur phéromone dépend des paramètres $\#Col$ et $\#\tau$, il va être détaillé par la suite. La définition du facteur heuristique dépend du problème

Construction d'une solution S :

$S \leftarrow \emptyset$

$Cand \leftarrow V$

tant que $Cand \neq \emptyset$ **faire**

choisir $v_i \in Cand$ avec probabilite $p_S(v_i)$

ajouter v_i a S

supprimer de $Cand$ les sommets qui violent des contraintes

fin tq

FIG. 6.2 – Construction de solution

à résoudre. Il doit être défini lors de l'application de l'algorithme à un problème spécifique.

6.2.2 Mise à jour de phéromone

Une fois que toutes les fourmis ont construit leurs solutions, les traces de phéromone sont mises à jour : d'abord, les traces de phéromone sont réduites par un facteur constant pour simuler l'évaporation ; par la suite, des traces de phéromone sont déposées sur les meilleures solutions. Plus précisement, la quantité $\tau^i(c)$ de la $i^{ème}$ structure de phéromone déposée sur le composant c est mise à jour comme suit :

$$\tau^i(c) \longleftarrow (1-\rho) \times \tau^i(c) + \ \Delta\tau^i(c)$$

où ρ est le facteur évaporation, tel que $0 \leq \rho \leq 1$, et $\Delta\tau^i(c)$ est la quantité de phéromone déposée sur le composant c. La définition de cette quantité dépend des paramètres $\#Col$ et $\#\tau$; elle va être détaillée par la suite.

6.3 Description de 6 variantes de m-ACO

Nous décrivons, dans cette section, six variantes de notre algorithme générique qui considèrent différentes valeurs pour les paramètres nombre de colonies de fourmis $\#Col$ et structures de phéromone $\#\tau$. Ces variantes diffèrent aussi dans la phase de mise à jour de phéromone. Parmi ces variantes, nous retrouvons des approches ACO qui existaient déjà dans la littérature, à savoir les variantes m-ACO_2, m-ACO_3 et m-ACO_4, et d'autres nouvelles approches proposées, à savoir les variantes m-ACO_1, m-ACO_5 et m-ACO_6. Etant donné qu'il est difficile d'implémenter tous les algorithmes MOACO proposés dans la littérature, nous avons repris l'idée de base des meilleurs algorithmes retournés par la comparaison de plusieurs MOACO réalisé dans [Garcia-Martinez *et al.* , 2007] sur le problème du voyageur de commerce multi-objectif.

Il est à noter que nous utilisons le même facteur heuristique η pour toutes les variantes qui est une agrégation des informations heuristiques η_i relatives à tous les objectifs. Etant donné que cette information est spécifique au problème à résoudre, nous laissons la définition de ce facteur lors de l'application de l'algorithme à un problème spécifique. Nous avons choisi d'utiliser le même facteur heuristique pour pouvoir distinguer l'influence des paramètres de l'algorithme m-ACO.

6.3.1 Variante 1 : m-ACO_1(m+1,m)

Pour cette nouvelle variante, le nombre de colonies $\#Col$ est mis à $m + 1$ et le nombre de structures de phéromone est mis à m, où $m = |F|$ est le nombre d'objectifs à optimiser : chaque colonie considère un objectif différent, et utilise sa propore structure de phéromone ; une autre colonie multi-objectif est ajoutée, qui optimise tous les objectifs.

Définition des facteurs phéromone. Le facteur phéromone $\tau_S^i(v_j)$ considéré par la $i^{ème}$ colonie uni-objectif, qui optimise la $i^{ème}$ fonction objectif f_i, est défini relativement à la $i^{ème}$ structure phéromone ; en fonction de l'application considérée, il peut être défini comme étant la quantité de phéromone déposée sur le sommet v_j ou comme étant la quantité de phéromone déposée sur les arêtes entre v_j et quelques sommets dans la solution partielle S.

Le facteur phéromone $\tau_S^{m+1}(v_j)$ considéré par la colonie multi-objectif est le facteur phéromone $\tau_S^r(v_j)$ de la $r^{ème}$ colonie uni-objectif, où r est une colonie choisie aléatoirement. Ainsi cette colonie considère, à chaque étape de construction, un objectif à optimiser choisi aléatoirement.

Mise à jour de phéromone. Pour chaque colonie uni-objectif, la phéromone est déposée sur les composants de la meilleure solution trouvée par la $i^{ème}$ colonie durant le cycle courant, où la qualité de solutions est évaluée relativement au $i^{ème}$ objectif f_i seulement. Ainsi, soit S^i la meilleure solution du cycle courant de la $i^{ème}$ colonie qui optimise la fonction f_i, et soit S^i_{meil} la meilleure solution de la $i^{ème}$ colonie parmi toutes les solutions construites par les fourmis de la $i^{ème}$ colonie depuis le début de l'exécution de l'algorithme. La quantité de phéromone déposée sur un composant de solution c pour la $i^{ème}$ structure de phéromone est définie comme suit

$$\Delta\tau^i(c) = \begin{cases} \frac{1}{(1+f_i(S^i)-f_i(S^i_{meil})} & \text{si c est un composant de } S^i, \\ 0 & \text{sinon.} \end{cases}$$

La colonie multi-objectif maintient un ensemble de solutions : une meilleure solution pour chaque objectif. Elle dépose de la phéromone sur chaque structure de phéromone relativement à l'objectif correspondant avec la même formule définie pour les autre colonies.

6.3.2 Variante 2 : m-ACO$_2$(m+1,m)

Cette deuxième variante est trés similaire à la première, et considère $m+1$ colonies et m structures de phéromone : une colonie uni-objectif est associée à chaque objectif, et le comportement de ces colonies est défini comme dans la première variante ; il y a aussi une autre colonie supplémentaire multi-objectif qui optimise tous les objectifs. La seule différence entre la variante 1 et 2 réside dans la manière avec laquelle cette colonie multi-objectif exploite les structures de phéromone des autres colonies pour construire des solutions. Pour cette colonie multi-objectif, le facteur phéromone $\tau_S^{m+1}(v_j)$ est égal à la somme de chaque facteur phéromone $\tau_S^c(v_j)$ de chaque colonie $c \in \{1, ..., m\}$, aggrégeant ainsi toutes les informations phéromonales dans une seule valeur.

Cette variante reprend l'idée de base de l'algorithme CompetAnts [Doerner *et al.* , 2003]. Cependant CompetAnts possède quelques spécificités. D'abord, le nombre de fourmis dans les colonies n'est pas fixe. De plus, les auteurs ne définissent pas une colonie $m+1$ qui optimise tous les objectifs mais un certain nombre de fourmis dans chaque colonie qu'ils appellent "spies".

6.3.3 Variant 3 : m-ACO$_3$(m,m)

Pour cette variante, le nombre de colonies $\#Col$ et le nombre de structures de phéromone sont mis à m. Chaque colonie considère un objectif différent, et utilise une structure de phéromone différente. Cette variante ressemble aussi à la première sauf qu'elle n'utilise pas une colonie multi-objectif. Le comportement de ces colonies est défini comme dans la première variante. L'interêt d'une telle approche est de laisser chaque colonie chercher dans une région différente de l'espace de recherche.

Cette variante implémente le schéma général de l'algorithme UnsortBicriterion [Iredi *et al.* , 2001].

6.3.4 Variant 4 : m-ACO$_4$(1,1)

Dans cette instanciation de m-ACO, il y a une seule colonie de fourmis et une seule structure de phéromone, i.e., $\#Col = 1$ et $\#\tau = 1$.

Définition des facteurs phéromone. Le facteur phéromone $\tau_S^1(v_j)$ considéré par les fourmis de l'unique colonie est défini relativement à l'unique structure de phéromone ; en fonction de l'application considérée, il peut être défini comme étant la quantité de phéromone déposée sur le sommet v_j ou comme étant la quantité de phéromone déposée sur les arêtes entre v_j et quelques sommets dans la solution partielle S.

Mise à jour de phéromone. Une fois que la colonie a construit un ensemble de solutions, chaque solution non-dominée (appartenant à l'ensemble Pareto) est récompensée.

Soit S_P l'ensemble des solutions non-dominées. La quantité de phéromone déposée sur un composant de solution c est défini comme suit

$$\Delta\tau^1(c) = \begin{cases} 1 & \text{si c est un composant de solution de } S_P, \\ 0 & \text{sinon.} \end{cases}$$

Chaque commposant appartenant à au moins une solution de l'ensemble Pareto reçoit la même quantité de phéromone. En effet, ces solutions appartiennent à des solutions non comparables.

Cette variante reprend l'idée de base de l'algorithme MACS [Barán & Schaerer, 2003]. La seule différence est dans la définition de la quantité à déposer lors de la mise à jour de phéromone. Dans MACS, le dépôt de phéromone est aussi autorisé aux solutions non dominées mais avec une quantité obtenue en agrégeant les fonctions objectifs et non pas une quantité fixe.

6.3.5 Variante 5 : m-ACO$_5$(1,m)

Dans cette nouvelle variante, il y a une seule colonie mais m structures de phéromone, i.e., $\#Col = 1$ et $\#\tau = m$.

Facteur phéromone. A chaque étape de construction de solution, les fourmis choisissent aléatoirement un objectif $r \in \{1, ..., m\}$ à optimiser. Le facteur phéromone $\tau_S^1(v_j)$ est ainsi défini par le facteur phéromone associé l'objectif choisi aléatoirement r.

Mise à jour de phéromone. Une fois que la colonie a construit un ensemble de solutions, les m meilleures solutions relativement aux m objectifs sont utilisées pour récompenser les m structures de phéromone. Soit S^i la solution de la colonie qui maximise le $i^{ème}$ objectif f_i pour le cycle courant, et soit S^i_{meil} la solution qui maximise f_i parmi toutes les solutions construites par les fourmis depuis le début de l'exécution. La quantité de phéromone déposée sur un composant de solution c pour la $i^{ème}$ structure de phéromone est définie comme suit :

$$\Delta\tau^i(c) = \begin{cases} \frac{1}{(1+f_i(S^i)-f_i(S^i_{meil}))} & \text{si c est un composant de } S^i, \\ 0 & \text{sinon.} \end{cases}$$

6.3.6 Variante 6 : m-ACO$_6$(1,m)

Cette dernière variante est trés similaire à la précédente et utilise une seule colonie et m structures de phéromone. Le comportement des fourmis de la colonie est défini comme dans la variante précédente : à chaque étape transition, les fourmis choisissent aléatoirement un objectif $r \in \{1, ..., m\}$ à optimiser.

Lors de la mise à jour de phéromone, les m meilleures solutions relativement aux m objectifs sont utilisées pour récompenser les m structures de phéromone

exactement comme dans la variante précédente. De plus, toutes les solutions non-dominées de l'ensemble pareto sont aussi autorisées à déposer sur les m structures de phéromone.

Soit S_P l'ensemble des solutions non-dominées. La quantité de phéromone $\Delta\tau^i(c)$ déposée sur un composant c d'une solution non-dominée pour la $i^{ème}$ structure de phéromone est défini comme suit :

$$\Delta\tau^i(c) = \begin{cases} 1 & \text{si c est un composant de solution de } S_P, \\ 0 & \text{sinon.} \end{cases}$$

6.4 Conclusion

Nous avons proposé dans ce chapitre un algorithme générique basé sur l'optimisation par colonie de fourmis pour la résolution des problèmes d'optimisation multi-objectifs. Cet algorithme est paramétré par le nombre de colonies de fourmis et le nombre de traces de phéromone. Nous considérons que le nombre de colonies de fourmis peut être différent du nombre de structures de phéromone, alors que généralement dans les algorithmes MOACO ces deux paramètres sont considérés comme étant un seul. Nous avons proposé six variantes de cet algorithme. Parmi ces variantes, trois sont de nouvelles approches, et trois autres reprennent le schéma général d'autres algorithmes MOACO présentés dans la littérature. Les trois nouvelles approches que nous proposons utilisent de nouvelles idées. La variante m-ACO_1(m+1,m) utilise une colonie multi-objectif qui considère à chaque étape de construction un objectif choisi aléatroirement. Les deux variantes m-ACO_5(1,m) et m-ACO_6(1,m) emploient une seule colonie avec plusieurs structures phéromonales. De plus, les fourmis de la colonie considèrent à chaque étape de construction un objectif aléatoirement.

Comme nous l'avons souligné au début de ce chapitre, cet algorithme générique

nous permet de mettre dans un même cadre différentes approches que nous pouvons tester et comparer indépendamment des détails d'implémentation et du schéma ACO considéré. Dans le chapitre suivant, nous appliquons ces différentes variantes sur le problème du sac à dos multi-objectif.

Chapitre 7

Application de m-ACO au problème du sac à dos multidimensionnel multi-objectif

7.1 Introduction

Nous montrons dans ce chapitre comment notre algorithme générique m-ACO peut être appliqué au problème du sac à dos multi-objectif (MOKP). Nous testons et comparons les variantes décrites dans le chapitre précédent sur des instances benchmark du MOKP. Nous comparons aussi notre approche avec des algorithmes évolutionnaires de l'état de l'art.

7.2 Le problème du sac à dos multidimensionnel multi-objectif

L'objectif de ce problème est de maximiser un vecteur de fonction profit tout en satisfaisant un ensemble de contraintes de capacité du sac à dos. Plus formellement, un MOKP est défini comme suit :

$$\begin{array}{rl} \text{maximiser} & f_k = \sum_{j=1}^{n} p_j^k x_j, \forall k \in 1..m \\ \text{sous les contraintes} & \sum_{j=1}^{n} w_j^i x_j \leq b_i, \forall i \in 1..q \\ & x_j \in \{0,1\}, \forall j \in 1..n \end{array} \tag{7.1}$$

où m est le nombre de fonctions objectif, n est le nombre d'objets, x_j est la variable de décision associée à l'objet o_j, q est le nombre de contraintes de ressource, w_j^i est la quantité de la ressource i consommé par l'objet o_j, b_i est la quantité totale disponible de la ressource i, p_j^k est le profit associé à l'objet o_j relativement l'objectif k.

7.3 Choix de la stratégie phéromonale

Nous avons étudié dans le chapitre 5 différentes stratégies pour la définition de la structure de phéromone pour le problème du sac à dos multidimensionnel uni-objectif. Nous rappelons brièvement dans ce qui suit ces différentes stratégies. En effet, pour résoudre un problème de type sac à dos, un point clé est de décider quels composants des solutions construites doivent être récompensés, et comment exploiter ces récompenses lors de la construction de nouvelles solutions. Etant donnée une solution d'un problème sac à dos $S = \{o_1, \ldots, o_k\}$, nous pouvons considérer trois manières différentes de déposer les traces de phéromone :

- Une première possibilité est de déposer la phéromone sur chaque objet sélectionné dans S.

– Une deuxième possibilité est de déposer la phéromone sur chaque couple (o_i, o_{i+1}) d'objets sélectionné succéssivement dans S.
– Une troisième possibilité est de déposer la phéromone sur chaque paire (o_i, o_j) d'objets différents de S.

Nous avons trouvé que la première et troisième stratégie trouvent des résultats nettement meilleurs que la deuxième. Ceci est expliqué par le fait que pour les problèmes de type sac à dos l'ordre dans lequel les objets sont sélectionnés n'est pas significatif. Cependant, la différence entre les deux premières stratégies n'est pas aussi importante même si on note un léger avantage pour la première stratégie sur la qualité des solutions. Cependant la deuxième stratégie trouve des résultats trés proches avec un temps de calcul nettement inférieur à la première, puisque cette dernière est de complexité quadratique alors que la deuxième est de complexité linéaire.

Etant donné la difficulté simultanée du cadre multi-objectif et de la complexité de la deuxième stratégie, nous avons choisi d'utiliser, pour le problème du sac à dos multidimensionnel multi-objectif, la première possibilité puisqu'elle offre un bon compromis entre la qualité des solutions et le temps d'exécution.

7.4 Définition du facteur heuristique

Nous définissons le facteur heuristique pour le problème du sac à dos multidimensionnel multiobjectif comme étant une agrégation du facteur heuristique que nous avons utilisé pour le problème du sac à dos multidimensionnel mono-objectif défini dans le chapitre 5. Ainsi, le facteur heuristique $\eta_S(o_j)$ utilisé dans la probabilité de transition pour le choix d'un objet candidat o_j est dfini comme suit : soit $d_S(i) = b_i - \sum_{g \in S} r_{ig}$ la quantité restante de la ressource i lorsque la fourmi a

construit la solution S ; nous définissons le ratio

$$h_S(o_j) = \sum_{i=1}^{q} \frac{w_j^i}{d_S(i)}$$

qui présente la dureté de l'objet o_j par rapport à toutes les contraintes $i \in 1..q$ et relativement à la solution construite S, de sorte que plus ce ratio est faible plus l'objet est intéressant. Nous intégrons alors le profit p_j^k de l'objet o_j par rapport à l'objectif k pour obtenir un ratio profit/ressource correspondant à l'information heuristique relative à l'objectif k,

$$\eta_S^k(o_j) = \frac{p_j^k}{h_S(o_j)}$$

nous définissons alors le facteur heuristique par une agrégation des informations heuristiques relatives à tous les objectifs,

$$\eta_S(o_j) = \sum_{k=1}^{m} \eta_S^k(o_j).$$

7.5 Expérimentations et résultats

Nous présentons dans cette section les résultats des six variantes de l'algorithme m-ACO, i.e., m-ACO$_1(m{+}1, m)$, m-ACO$_2(m{+}1, m)$, m-ACO$_3(m, m)$, m-ACO$_4(1, 1)$, m-ACO$_5(1, m)$ et m-ACO$_6(1, m)$. Nous comparons aussi ces résultats avec plusieurs algorithmes évolutionnaires de l'état de l'art : SPEA [Zitzler & Thiele, 1999], NSGA [Srinivas & Deb, 1994], HLGA [Hajela & Lin, 1992], NPGA [Horn *et al.* , 1994], VEGA [Schaffer, 1985], SPEA2 [Zitzler *et al.* , 2001] et NSGAII [Deb *et al.* , 2002]. Ces algorithmes évolutionnaires ont été décrits dans le chapitre 3.

7.5.1 Ensemble de tests

Les expérimentations sont effectuées sur les 9 instances définies dans [Zitzler & Thiele, 1999]. Ces benchmarks ont 2, 3 et 4 objectifs combinés avec 250, 500 et 750 objets. De plus, pour chaque instance, le nombre de contraintes est égal au nombre d'objectifs. Ces instances ont été générées de manière aléatoire avec des poids et des profits décorélés, et les capacités de chaque contrainte sont réglées à environ la moitié de la somme des poids de la contrainte considérée. Pour celà le nombre d'objets attendus dans les solutions optimales est d'environ la moitié du nombre total d'objets du problème. Nous allons noter ces instances dans ce qui suit n.m avec n étant le nombre d'objets et m le nombre d'objectifs.

Les instances considérées ainsi que les résultats des différents algorithmes évolutionnaires sont accessibles sur le site http ://www.tik.ee.ethz.ch/∼zitzler/testdata.html.

7.5.2 Conditions d'expérimentation et paramétrage

Tous les algorithmes ont été implémentés en C++. Pour toutes les variantes de m-ACO($\#Col$,$\#\tau$), nous avons considéré les mêmes conditions d'expérimentations et les mêmes valeurs pour les paramètres de l'algorithme. Nous avons mis α, le coefficient de pondération du facteur phéromone, à 1, et ρ, le ratio d'évaporation, à 0.01. Nous nous sommes basés pour le choix de ces paramètres sur l'étude que nous avons menée au chapitre 5 sur l'influence de ces paramètres sur la résolution pour le cas du problème du sac à dos multidimensionnel uniobjectif. En effet, nous avons montré dans ce chapitre que la recherche d'un bon compromis entre intensification et diversification de la recherche peut être influencée par ces paramètres. Nous avons trouvé que, avec ces valeurs de α et ρ qui favorisent l'exploration, les fourmis trouvent de meilleures solutions en fin d'exécution, mais elles ont besoin de plus de cycles pour converger sur ces solutions.

Pour le choix des valeurs des autres paramètres nous avons mené une série d'expérimentations sur quelques instances du MOKP. Nous avons mis β, le coefficient de pondération du facteur heuristique, à 4, le nombre de cycles à 3000 et les bornes de phéromone τ_{min} et τ_{max} à 0.01 et 6. Nous avons mis le nombre de fourmis pour chaque colonie employée à $\frac{100}{\#Col}$, ainsi nous obtenons le même nombre de solutions générées pour toutes les variantes.

Chaque variante a été testée dix fois pour chaque instance reportée. Donc nous présentons par la suite pour les métriques considérées les résultats calculés de dix exécutions de chaque algorithme.

7.5.3 Mesures de performance considérées

Pour comparer les performances des différents algorithmes, nous utilisons d'abord pour les instances représentables graphiquement, i.e. les instances bi-objectifs, une analyse visuelle des surfaces de compromis générées par les différents algorithmes. Ces graphiques peuvent aussi nous aider à comprendre les valeurs des métriques utilisées par la suite.

Nous utilisons deux métriques qui appartiennent aux deux types de métriques qui ont été définies dans le chapitre 3. La première est la métrique C qui est une métrique relative, qui compare deux algorithmes, et la deuxième est la métrique d'hypervolume [1] qui est une métrique absolue, qui évalue un algorithme de manière indépendante.

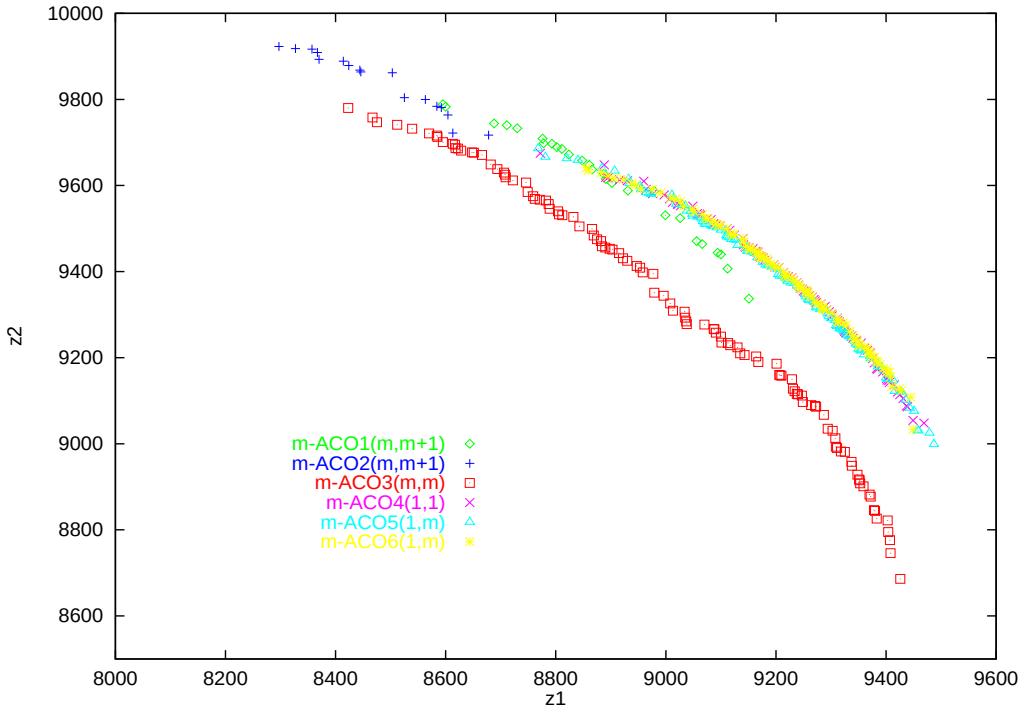

FIG. 7.1 – Les fronts Pareto retournés par dix exécutions de chaque variante sur l'instance avec 2 objectifs et 250 objets.

7.5.4 Comparaison des différentes variantes de m-ACO

Analyse visuelle des surfaces de compromis

Pour représenter graphiquement la performance des différentes variantes sur les instances biobjectives testées, nous allons procéder dans cette section selon le même protocol expérimental que celui de Zitzler et al. [Zitzler *et al.* , 2000]. Tous les ensembles Pareto générés par chaque algorithme dans les dix exécutions réalisées vont être fusionnés dans un seul ensemble Pareto en supprimant les solutions dominées

[1]Il est à noter que la fonction d'hypervolume que nous avons utilisée dans les expérimentations a été tirée de la page de Zitzler http ://www.tik.ee.ethz.ch/∼zitzler.

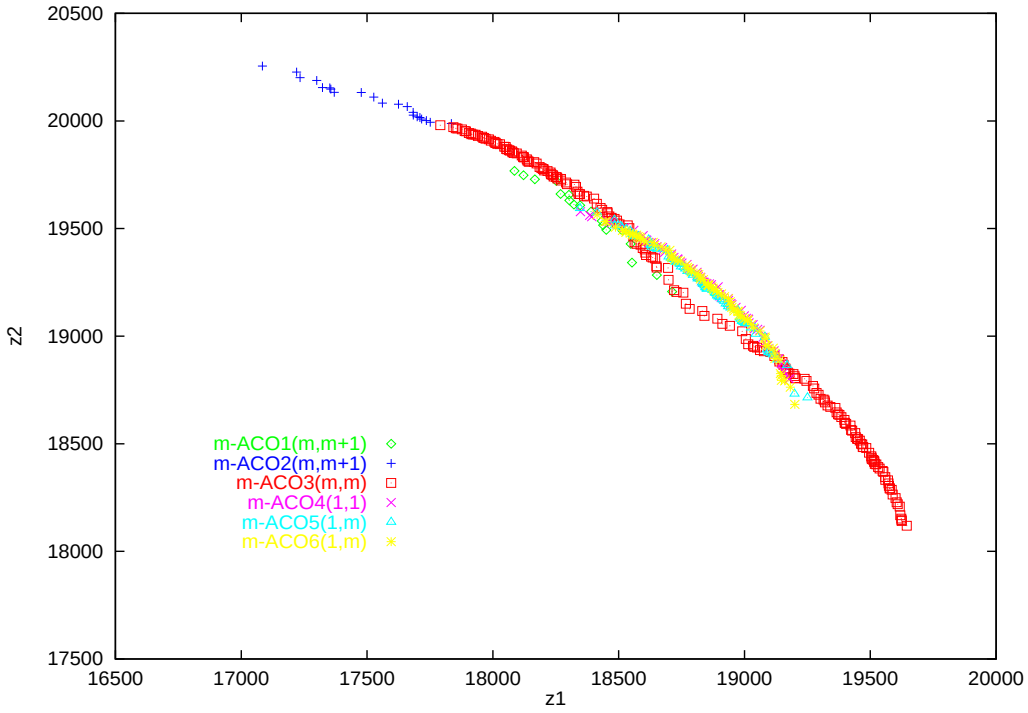

FIG. 7.2 – Les fronts Pareto retournés par dix exécutions de chaque variante sur l'instance avec 2 objectifs et 500 objets.

de l'ensemble fusionné.

Soit $\overline{P_i^j}$ l'ensemble Pareto retourné par l'algorithme i durant la $j^{ème}$ exécution, $P_i = \overline{P_i^1} \bigcup \overline{P_i^2} \bigcup ... \bigcup \overline{P_i^{10}}$, l'union des ensembles de solutions retournés par les dix exécutions de l'algorithme i, et finalement $\overline{P_i}$ l'ensemble de toutes les solutions non dominées de P_i.

Les figures 7.1 et 7.2 représentent les ensembles $\overline{P_i}$ des différentes variantes sur les instances 250.2 et 500.2 respectivement. Ces graphiques offrent une information visuelle qui n'est pas mesurable mais qui est parfois plus parlante que les évaluations numériques.

Pour l'instance 250.2 (figure 7.1), nous remarquons d'abord que la variante

m-ACO_3(m,m) obtient des solutions nettement dominées par les autres variantes puisque la surface de compromis procurée par cette variante est largement au dessous des autres surfaces. Cependant cette variante trouve des solutions qui sont bien réparties sur tout le front Pareto.

La variante m-ACO_2(m,m+1) trouve des solutions qui sont au dessus de certaines solutions de m-ACO_3(m,m) mais qui se trouvent sur l'extrémité du front Pareto. Les solutions générées par cette variante optimisent un des deux objectifs, mais elles ne couvrent pas le milieu du front Pareto. Ainsi, elle ne trouve pas des solutions de bonne qualité qui établissent un compromis entre les deux objectifs.

Le front Pareto procuré par la variante m-ACO_1(m,m+1), qui est meilleur que les deux précédents, domine dans une petite région les fronts retournés par les trois autres variantes m-ACO_4(1,1), m-ACO_5(1,m) et m-ACO_6(1,m) mais dans le reste ces dernières le dominent. Les fronts Pareto des variantes m-ACO_4(1,1), m-ACO_5(1,m) et m-ACO_6(1,m) sont trés proches et plus similaires que les autres variantes. Ces variantes trouvent des solutions qui couvrent le milieu du front Pareto. Cependant, le front Pareto de m-ACO_6(1,m) est légèrement au dessus des deux autres variantes.

Pour l'instance 500.2 (figure 7.2), les différentes variantes ont un comportement trés similaire à celui de l'instance 250.2. Toutefois, les différences entre ces fronts ne sont pas aussi importantes pour cette instance. En effet, là aussi m-ACO_2(m,m+1) trouve des solutions qui se trouvent sur l'extrémité du front Pareto. m-ACO_3(m,m) trouve des solutions non dominées qui sont bien réparties sur tout le front Pareto mais qui sont, dans la partie centrale, au dessous des fronts des variantes m-ACO_4(1,1), m-ACO_5(1,m) et m-ACO_6(1,m). La variante m-ACO_6(1,m) est meilleure pour cette instance aussi que les autres variantes.

Analyse de la métrique C

Dans cette section nous comparons les six variantes de m-ACO suivant la métrique C. Les tableaux 7.1 et 7.2 représentent les résultats de cette métrique sur respectivement les instances avec 250 et 500 objets combinés avec 2, 3 et 4 objectifs. Chaque ligne de ces tableaux affiche successivement les noms des deux variantes comparées, et les valeurs minimales, moyennes et maximales de la métrique C parmi dix exécutions des deux variantes.

En comparant les différentes variantes sur les instances avec 250 objets (tableau 7.1), nous constatons que, pour cette métrique, m-ACO_6(1,m) trouve les meilleurs résultats. En effet, les valeurs de la métrique C obtenues par m-ACO_6(1,m) sont toujours supérieures à celles trouvées par les autres variantes. Plus particulièrement, certaines solutions retournées par les variantes m-ACO_1(m,m+1), m-ACO_2(m,m+1) et m-ACO_3(m,m) sont dominées par m-ACO_6(1,m), alors que pour la plupart des cas aucune solution de m-ACO_6(1,m) n'est dominée par d'autres de ces variantes, i.e. les cas où les valeurs de la métrique C sont égales à 0.

La différence des résultats n'est pas aussi importante pour les deux autres variantes m-ACO_4(1,1) et m-ACO_5(1,m), puisque ces variantes trouvent des solutions qui dominent d'autres solutions de m-ACO_6(1,m). Cependant, les valeurs de la métrique C trouvées par m-ACO_6(1,m) sont supérieures ou égales à celles trouvées par ces deux variantes.

Pour les instances avec 500 objets (tableau 7.2), nous remarquons que les résultats de m-ACO_6(1,m) sont encore plus élevés que les autres variantes. En effet, cette variante trouve des valeurs de la métrique C supérieures à celles trouvées pour les instances avec 250 objets.

En comparant les autres variantes avec m-ACO_6(1,m), nous trouvons qu'elles trouvent pour la plupart des cas des valeurs de la métrique C égales ou proches de 0. Par contre, m-ACO_6(1,m) trouve des valeurs supérieures de C, égales parfois

même à 1 pour l'instance la plus difficile avec 4 objectifs. Pour ces cas, toutes les solutions de m-ACO_6(1,m) dominent toutes les solutions retournées par la variante comparée. Il est à noter que pour ces instances aussi les deux variantes m-ACO_5(1,m) et surtout m-ACO_4(1,1) trouvent des résultats assez bons.

Nous concluons cette section en notant que ces résultats de la métrique C confirment ce que nous avons constaté en analysant visuellement les surfaces de compromis retournées par les différentes variantes pour les instances biobjectives. De plus, les résultats de m-ACO_6(1,m) s'améliorent encore plus, pour la plupart des cas, pour les instances les plus difficiles avec 3 et 4 objectifs.

Analyse de la métrique d'hypervolume

Nous comparons maintenant les différentes variantes par la métrique d'hypervolume qui appartient à une autre classe de métriques : les métriques absolues. Le tableau 7.3 présente les résultats des moyennes de l'hypervolume sur les 10 exécutions de chacune des 6 variantes de m-ACO sur les instances avec 250 et 500 objets combinés avec 2, 3 et 4 objectifs.

En comparant ces résultats, nous remarquons que la variante m-ACO_3(m,m) obtient globalement les valeurs les plus élevées (donc les meilleures) de l'hypervolume. Ensuite, nous trouvons les trois variantes m-ACO_4(1,1), m-ACO_5(1,m) et m-ACO_6(1,m) qui obtiennent des valeurs assez proches. Et enfin les deux variantes m-ACO_1(m,m+1) et m-ACO_2(m,m+1) trouvent les valeurs d'hypervolume les moins bonnes.

Ainsi, nous trouvons que la variante m-ACO_3(m,m), qui est largement dominée suivant la métrique C par les autres variantes, obtient une bonne moyenne d'hypervolume sur les instance testées.

Pour comprendre ces résultats, nous revenons sur les figures 7.1 et 7.2 qui représentent graphiquement la surface de compromis des différentes variantes sur les

Variantes comparées	250.2			250.3			250.4		
	min	**moy**	**max**	**min**	**moy**	**max**	**min**	**moy**	**max**
C(m-ACO$_1$, m-ACO$_2$)	0	0,03125	0,25	0	0,0153	0,0909	0	0,0071	0,0714
C(m-ACO$_2$, m-ACO$_1$)	0	0	0	0	0,0177	0,0725	0	0,0004	0,0020
C(m-ACO$_1$, m-ACO$_3$)	0,1341	0,1951	0,2683	0	0,0003	0,0020	0,0009	0,001	0,0011
C(m-ACO$_3$, m-ACO$_1$)	0	0,0069	0,0556	0	0	0	0	0,0002	0,002
C(m-ACO$_1$, m-ACO$_4$)	0	0,0056	0,0291	0	0,0001	0,0010	0	0,0001	0,0003
C(m-ACO$_4$, m-ACO$_1$)	0	0,0254	0,1429	0,0156	0,1	0,2188	0,0151	0,0242	0,0449
C(m-ACO$_1$, m-ACO$_5$)	0	0,0052	0,0278	0	0,0001	0,0013	0	0,0006	0,0012
C(m-ACO$_5$, m-ACO$_1$)	0	0,1071	0,2143	0	0,0089	0,0250	0	0,0059	0,0163
C(m-ACO$_1$, m-ACO$_6$)	0	0,0019	0,0097	0	0,0001	0,0009	0	0,0001	0,0006
C(m-ACO$_6$, m-ACO$_1$)	0	0,0009	0,0085	0	**0,0416**	**0,1277**	0	**0,0214**	**0,0387**
C(m-ACO$_2$, m-ACO$_3$)	0	0,0386	0,0824	0	0,0001	0,0013	0	0,0002	0,0008
C(m-ACO$_3$, m-ACO$_2$)	0	0	0	0	0,0267	0,0667	0	0	0
C(m-ACO$_2$, m-ACO$_4$)	0	0	0	0	0	0	0	0	0
C(m-ACO$_4$, m-ACO$_2$)	0	0	0	0	0,1647	0,4118	0,0667	0,1303	0,3333
C(m-ACO$_2$, m-ACO$_5$)	0	0	0	0	0	0	0	0	0
C(m-ACO$_5$, m-ACO$_2$)	0	0	0	0	0,0882	0,3529	0	0,02667	0,2
C(m-ACO$_2$, m-ACO$_6$)	0	0	0	0	0	0	0	0	0
C(m-ACO$_6$, m-ACO$_2$)	0	0	0	0	**0,0235**	**0,1176**	0	**0,0626**	**0,2**
C(m-ACO$_3$, m-ACO$_4$)	0	0	0	0	0,0001	0,0009	0	0	0
C(m-ACO$_4$, m-ACO$_3$)	0,2188	0,2862	0,3291	0,0035	0,006	0,0094	0,0015	0,0029	0,0039
C(m-ACO$_3$, m-ACO$_5$)	0	0	0	0	0,001	0,0071	0	0,001	0,002
C(m-ACO$_5$, m-ACO$_3$)	0,1591	0,273	0,3797	0,0018	0,0061	0,0105	0,0002	0,0007	0,0013
C(m-ACO$_3$, m-ACO$_6$)	0	0	0	0	0,0003	0,0028	0	0	0
C(m-ACO$_6$, m-ACO$_3$)	**0,2911**	**0,308**	**0,3291**	**0,0041**	**0,0081**	**0,0154**	**0**	**0,1214**	**0,4**
C(m-ACO$_4$, m-ACO$_5$)	0	0,0119	0,0278	0,0013	0,0055	0,0108	0	0,0009	0,0036
C(m-ACO$_5$, m-ACO$_4$)	0	0,0109	0,0482	0	0	0	0	0	0
C(m-ACO$_4$, m-ACO$_6$)	0	0,001	0,0103	0	0,0004	0,0019	0	0,0002	0,0011
C(m-ACO$_6$, m-ACO$_4$)	0	**0,0072**	**0,0388**	0	**0,0010**	**0,0057**	0	**0,0003**	**0,003**
C(m-ACO$_5$, m-ACO$_6$)	0	0,0019	0,0105	0	0,0002	0,0009	0	0	0,0003
C(m-ACO$_6$, m-ACO$_5$)	0	**0,0264**	**0,0833**	**0,0012**	**0,0045**	**0,0105**	0	**0,002**	**0,0104**

TAB. 7.1 – Comparaison des 6 variantes de m-ACO suivant la métrique C sur les instances avec 250 objets et 2, 3 et 4 objectifs

Variantes comparées	500.2			500.3			500.4		
	min	**moy**	**max**	**min**	**moy**	**max**	**min**	**moy**	**max**
C(m-ACO$_1$, m-ACO$_2$)	0	0,1	1	0	0,0158	0,0909	0,4167	0,6276	0,7778
C(m-ACO$_2$, m-ACO$_1$)	0	0,0001	0,0006	0	0,0003	0,002	0	0	0
C(m-ACO$_1$, m-ACO$_3$)	0	0,092799	0,365789	0	0,0007	0,0031	0	0,0383	0,1764
C(m-ACO$_3$, m-ACO$_1$)	0	0	0	0	0,0014	0,004	0	0,0005	0,0070
C(m-ACO$_1$, m-ACO$_4$)	0	0,0032	0,0213	0	0	0	0	0	0
C(m-ACO$_4$, m-ACO$_1$)	0	0,0168	0,0909	0	0,0535	0,1	0,0046	0,0392	0,0951
C(m-ACO$_1$, m-ACO$_5$)	0	0,0013	0,0132	0	0,0004	0,0015	0	0	0,0004
C(m-ACO$_5$, m-ACO$_1$)	0	0,1495	0,8889	0	0,0117	0,0370	0	0,0081	0,0186
C(m-ACO$_1$, m-ACO$_6$)	0	0,0025	0,025	0	0,0002	0,002	0	0	0
C(m-ACO$_6$, m-ACO$_1$)	0	**0,0444**	**0,4444**	0	**0,0618**	**0,2**	**0,0046**	**0,034**	**0,1021**
C(m-ACO$_2$, m-ACO$_3$)	0	0	0	0	0,0004	0,0038	0	0,0546	0,2828
C(m-ACO$_3$, m-ACO$_2$)	0	0	0	0	0,0007	0,0044	0,2	0,4361	0,6111
C(m-ACO$_2$, m-ACO$_4$)	0	0	0	0	0	0	0	0	0
C(m-ACO$_4$, m-ACO$_2$)	0	0	0	0	0,0548	0,2941	0,8462	0,9231	1
C(m-ACO$_2$, m-ACO$_5$)	0	0	0	0	0	0	0	0	0
C(m-ACO$_5$, m-ACO$_2$)	0	0,1	1	0	0,0248	0,1818	0,7692	0,9297	1
C(m-ACO$_2$, m-ACO$_6$)	0	0	0	0	0	0	0	0	0
C(m-ACO$_6$, m-ACO$_2$)	0	0	0	**0**	**0,1087**	**0,3125**	**1**	**1**	**1**
C(m-ACO$_3$, m-ACO$_4$)	0	0	0	0	0	0	0	0	0
C(m-ACO$_4$, m-ACO$_3$)	0,1447	0,2164	0,3026	0	0,0066	0,0105	0,0005	0,0527	0,28
C(m-ACO$_3$, m-ACO$_5$)	0	0,0032	0,0317	0	0,0006	0,0028	0	0,0003	0,0016
C(m-ACO$_5$, m-ACO$_3$)	0,0128	0,0832	0,3824	0,0032	0,0079	0,0219	0	0,0488	0,2278
C(m-ACO$_3$, m-ACO$_6$)	0	0	0	0	0,0005	0,0015	0	0	0
C(m-ACO$_6$, m-ACO$_3$)	**0**	**0,0358**	**0,0764**	**0,0032**	**0,0075**	**0,0107**	**0,0007**	**0,0507**	**0,289**
C(m-ACO$_4$, m-ACO$_5$)	0	0,0331	0,1053	0	0,0136	0,0392	0,0004	0,0038	0,0095
C(m-ACO$_5$, m-ACO$_4$)	0	0,0537	0,3553	0	0	0	0	0	0,0002
C(m-ACO$_4$, m-ACO$_6$)	0	0,0023	0,0227	0	0,0021	0,0137	0	0,0001	0,0012
C(m-ACO$_6$, m-ACO$_4$)	0	0,0064	0,0213	0	0,0022	0,0111	0	**0,0003**	**0,0023**
C(m-ACO$_5$, m-ACO$_6$)	0	0,0099	0,0581	0	0	0	0	0	0
C(m-ACO$_6$, m-ACO$_5$)	0	**0,0189**	**0,0714**	0	**0,0125**	**0,036**	0	**0,0029**	**0,0073**

TAB. 7.2 – Comparaison des 6 variantes de m-ACO suivant la métrique C sur les instances avec 500 objets et 2, 3 et 4 objectifs

instances biobjectives. Suivant ces figures, nous avons conclu que la surface de compromis procurée par la variante m-ACO$_3$(m,m) est largement au dessous des autres surfaces mais elle est bien répartie sur la totalité du front Pareto. Ceci explique la valeur élevée trouvée pour l'hypervolume, puisque cette métrique, comme nous l'avons définie au chapitre 3, calcule une approximation du volume compris sous la courbe formée par les points de l'ensemble à évaluer.

Nb d'objets	250			500		
Nb d'objectifs	**2**	**3**	**4**	**2**	**3**	**4**
m-ACO$_1$(m,m+1)	8,81E+07	7,48E+11	5,96E+15	3,66E+08	6,28E+12	9,55E+16
m-ACO$_2$(m,m+1)	8,42E+07	7,17E+11	5,45E+15	3,54E+08	5,87E+12	8,24E+16
m-ACO$_3$(m,m)	9,12E+07	8,41E+11	6,94E+15	3,88E+08	6,7E+12	1,05E+17
m-ACO$_4$(1,1)	9,08E+07	7,62E+11	5,98E+15	3,74E+08	6,18E+12	9,6E+16
m-ACO$_5$(1,m)	9,12E+07	7,68E+11	6,04E+15	3,75E+08	6,2E+12	9,63E+16
m-ACO$_6$(1,m)	9,06E+07	7,62E+11	5,97E+15	3,74E+08	6,18E+12	9,58E+16

TAB. 7.3 – Comparaison des 6 variantes de m-ACO suivant l'hypervolume sur les instances avec 250 et 500 objets combinés avec 2, 3 et 4 objectifs

7.5.5 Comparaison de m-ACO$_6$(1,m) avec les algorithmes génétiques

Nous comparons dans cette section la variante m-ACO$_6$(1,m), qui trouve les meilleurs résultats comparée aux autres variantes, avec des algorithmes génétiques de l'état de l'art.

Analyse visuelle des surfaces de compromis

Les figures 7.3 et 7.4 représentent les ensembles $\overline{P_i}$ de la variante m-ACO$_6$(1,m) et des algorithmes génétiques (AG) SPEA, NSGA, HLGA et NPGA sur les instances

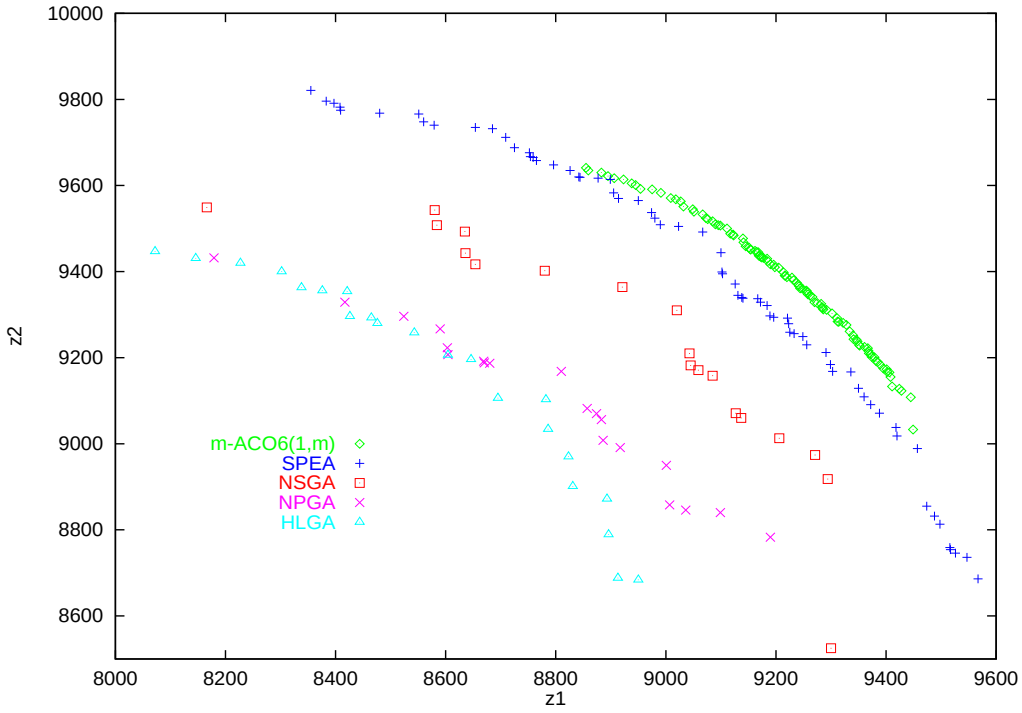

FIG. 7.3 – Les fronts Pareto retournés par dix exécutions de chaque algorithme sur l'instance avec 2 objectifs et 250 objets.

250.2 et 500.2 respectivement. La figure 7.5 compare les ensembles Pareto de m-ACO_6(1,m) avec les algorithmes SPEA, SPEA2 et NSGAII sur l'instance 750.2.

En comparant les solutions non dominées retournées par m-ACO_6(1,m) avec celles des autres AG sur les instances 250.2 et 500.2, nous constatons que ces solutions dominent largement celles des AG. Pour l'instance 750.2, nous remarquons aussi que les solutions non dominées de m-ACO_6(1,m) dominent nettement celles de tous les AG comparés. De plus, aucune solution de tous ces AG ne domine une autre solution de m-ACO_6(1,m). Notons toutefois que ces AG, plus particulièrement SPEA2 et NSGAII, trouvent des solutions qui sont mieux réparties sur le front Pareto.

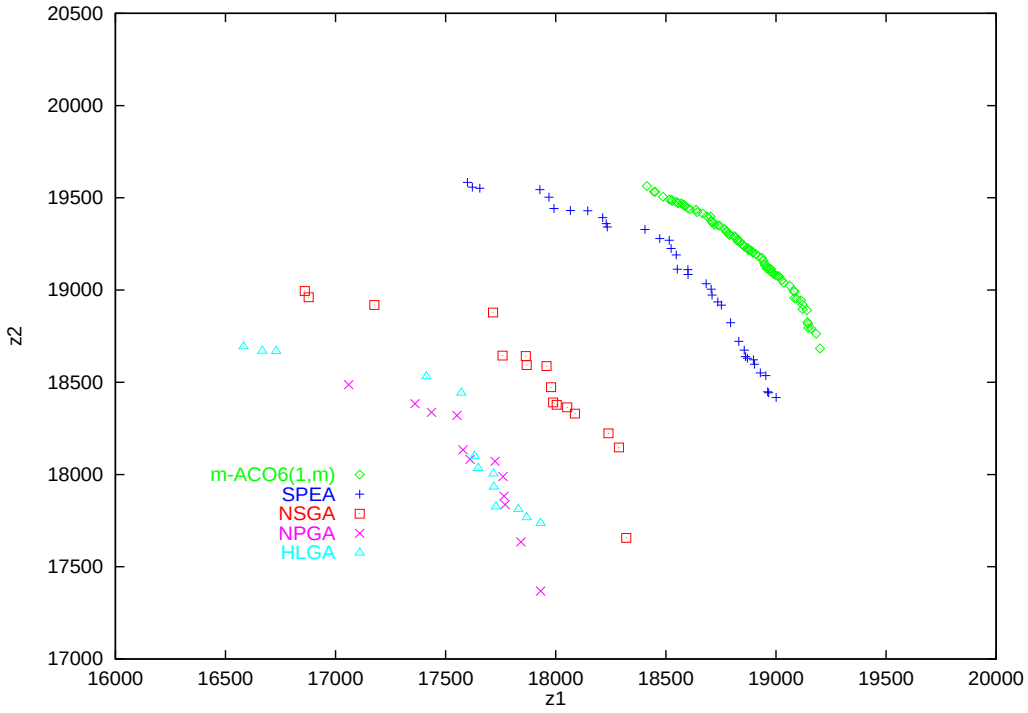

FIG. 7.4 – Les fronts Pareto retournés par dix exécutions de chaque algorithme sur l'instance avec 2 objectifs et 500 objets.

Analyse de la métrique C

Dans cette section nous comparons la variante m-ACO_6 avec les algorithmes évolutionnaires suivant la métrique C. Les tableaux 7.4 et 7.5 comparent m-ACO_6, sur respectivement les instances avec 250 et 500 objets combinés avec 2, 3 et 4 objectifs, avec les résultats des algorithmes VEGA, SPEA, NSGA, HLGA et NPGA. Le tableau 7.6 présente les résultats de m-ACO_6 et les alorithmes VEGA, SPEA, SPEA2, NSGA, NSGAII, HLGA et NPGA sur les instances avec 750 objets avec 2, 3 et 4 objectifs. Chaque ligne de ces tableaux affiche successivement les noms des deux algorithmes comparées, et les valeurs minimales, moyennes et maximales de la

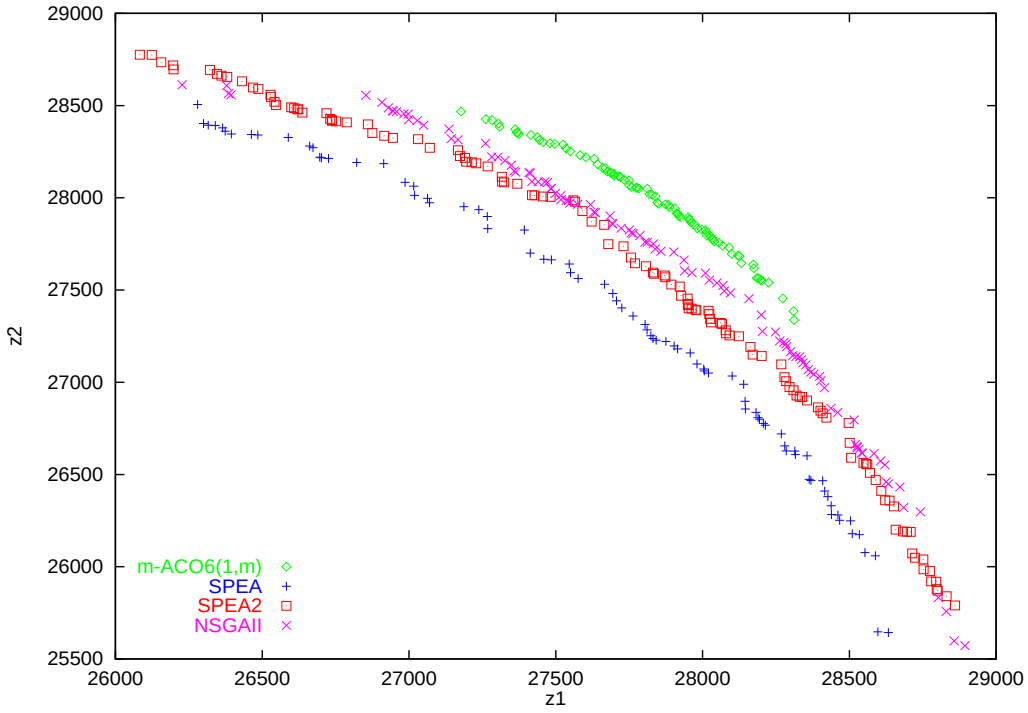

FIG. 7.5 – Les fronts Pareto retournés par dix exécutions de chaque algorithme sur l'instance avec 2 objectifs et 750 objets.

métrique C parmi dix exécutions des deux algorithmes.

En comparant m-ACO$_6$ avec tous les algorithmes évolutionnaires, nous constatons qu'elle est nettement meilleure sur toutes les instances testées. En effet, les solutions de m-ACO$_6$ dominent largement les solutions retournées par les autres algorithmes. De plus, aucune solution de tous les AG comparés ne domine une autre de m-ACO$_6$ puisque les valeurs de la métrique C pour tous ces algorithmes et pour toutes les instances sont égales à 0.

Nous constatons aussi que plus les instances sont difficiles et grandes, meilleurs sont les résultats de m-ACO$_6$. En effet, pour les instances avec 500 objets, les valeurs de la métrique C de m-ACO$_6$ avec tous les AG sont égales ou proches de 1 sauf pour

SPEA qui est l'algorithme le plus performant parmi la liste des AG comparés pour ces instances. Pour les instances avec 750 objets, les valeurs C de m-ACO_6 avec les AG sont égales à 1 sauf pour SPEA, SPEA2 et NSGAII. En effet, ces deux derniers algorithmes sont parmi les algorithmes évolutionaires les plus performants et les plus connus dans la littérature. Les résultats de ces algorithmes ne sont disponibles que pour les instances avec 750 objets.

Il est à noter que nous avons comparé aussi toutes les autres variantes de m-ACO avec ces algorithmes évolutionnaires sur les instances avec 250 et 500 objets. Nous avons trouvé que toutes ces variantes sont meilleures que les AG sur toutes les instances, et nous trouvons toujours que m-ACO_6 trouve des valeurs plus élevées de la métrique C que les autres variantes.

	250.2			250.3			250.4		
Algorithmes comparés	**min**	**moy**	**max**	**min**	**moy**	**max**	**min**	**moy**	**max**
C(m-ACO_6 , VEGA)	0,7895	0,9105	0,9474	0,4810	0,8539	0,9806	0,3004	0,6617	0,9009
C(VEGA , m-ACO_6)	0	0	0	0	0	0	0	0	0
C(m-ACO_6 , SPEA)	0,1333	0,17	0,2	0,0485	0,1676	0,2283	0,1782	0,2571	0,3228
C(SPEA , m-ACO_6)	0	0	0	0	0	0	0	0	0
C(m-ACO_6 , NSGA)	0,6154	0,6731	0,7308	0,1743	0,6321	0,8129	0,3374	0,5574	0,7756
C(NSGA , m-ACO_6)	0	0	0	0	0	0	0	0	0
C(m-ACO_6 , HLGA)	0,6923	0,7308	0,8462	0,8793	0,9841	1	0,8947	0,9787	1
C(HLGA , m-ACO_6)	0	0	0	0	0	0	0	0	0
C(m-ACO_6 , NPGA)	0,9444	0,9833	1	0,4510	0,7632	0,9639	0,2780	0,6259	0,8338
C(NPGA , m-ACO_6)	0	0	0	0	0	0	0	0	0

TAB. 7.4 – Comparaison de m-ACO_6(1,m) avec les algorithmes génétiques suivant la métrique C sur les instances avec 250 objets et 2, 3 et 4 objectifs

Algorithmes comparés	500.2			500.3			500.4		
	min	**moy**	**max**	**min**	**moy**	**max**	**min**	**moy**	**max**
C(m-ACO$_6$, VEGA)	1	1	1	0,9855	0,9986	1	1	1	1
C(VEGA , m-ACO$_6$)	0	0	0	0	0	0	0	0	0
C(m-ACO$_6$, SPEA)	0,3514	0,5795	0,7188	0,3707	0,5321	0,6946	0,6688	0,7598	0,8620
C(SPEA , m-ACO$_6$)	0	0	0	0	0	0	0	0	0
C(m-ACO$_6$, NSGA)	1	1	1	0,9489	0,9790	1	0,9975	0,999	1
C(NSGA , m-ACO$_6$)	0	0	0	0	0	0	0	0	0
C(m-ACO$_6$, HLGA)	1	1	1	1	1	1	1	1	1
C(HLGA , m-ACO$_6$)	0	0	0	0	0	0	0	0	0
C(m-ACO$_6$, NPGA)	1	1	1	1	1	1	1	1	1
C(NPGA , m-ACO$_6$)	0	0	0	0	0	0	0	0	0

TAB. 7.5 – Comparaison de m-ACO$_6$(1,m) avec les algorithmes génétiques suivant la métrique C sur les instances avec 500 objets et 2, 3 et 4 objectifs

Analyse de la métrique d'hypervolume

Le tableau 7.7 présente les résultats des moyennes de l'hypervolumes sur 10 exécutions de chaque algorithme comparé sur toutes les instances des jeux de tests. En comparant m-ACO$_6$(1,m) avec les AG, nous remarquons d'abord que cet algorithme trouve des valeurs de l'hypervolume qui sont largement meilleures que les algorithmes NSGA, NPGA, HLGA et VEGA sur toutes les instances testées. En le comparant avec l'algorithme SPEA, nous trouvons que SPEA trouve des valeurs légèrement supérieures pour les instances avec 250 objets. Mais pour les instances les plus grandes avec 500 et 750 objets, nous trouvons que m-ACO$_6$(1,m) trouve des valeurs largement supérieures, spécialement pour les instances les plus difficiles avec 4 objectifs.

En comparant m-ACO$_6$(1,m) avec NSGAII et SPEA2 sur les instances avec 750 objets, nous constatons qu'ils sont trés proches pour les trois instances testées avec un trés leger avantage pour SPEA2.

Algorithmes comparés	750.2			750.3			750.4		
	min	**moy**	**max**	**min**	**moy**	**max**	**min**	**moy**	**max**
C(m-ACO$_6$, VEGA)	1	1	1	1	1	1	-	-	-
C(VEGA , m-ACO$_6$)	0	0	0	0	0	0	-	-	-
C(m-ACO$_6$, SPEA)	0,2143	0,6641	0,9545	0,17	0,245	0,2867	0,7629	0,826	0,8857
C(SPEA , m-ACO$_6$)	0	0	0	0	0	0	0	0	0
C(m-ACO$_6$, SPEA2)	0,108	0,164	0,196	0,08	0,1277	0,1567	0,5029	0,56	0,6657
C(SPEA2 , m-ACO$_6$)	0	0	0	0	0	0	0	0	0
C(m-ACO$_6$, NSGA)	1	1	1	1	1	1	-	-	-
C(NSGA , m-ACO$_6$)	0	0	0	0	0	0	-	-	-
C(m-ACO$_6$, NSGAII)	0,092	0,2192	0,268	0,1467	0,1957	0,2367	0,4886	0,5769	0,6371
C(NSGAII , m-ACO$_6$)	0	0	0	0	0	0	0	0	0
C(m-ACO$_6$, HLGA)	1	1	1	1	1	1	1	1	1
C(HLGA , m-ACO$_6$)	0	0	0	0	0	0	0	0	0
C(m-ACO$_6$, NPGA)	1	1	1	1	1	1	1	1	1
C(NPGA , m-ACO$_6$)	0	0	0	0	0	0	0	0	0

TAB. 7.6 – Comparaison de m-ACO$_6$(1,m) avec les algorithmes génétiques suivant la métrique C sur les instances avec 750 objets et 2, 3 et 4 objectifs

Nb d'objets	250			500			750		
Nb d'objectifs	**2**	**3**	**4**	**2**	**3**	**4**	**2**	**3**	**4**
m-ACO$_6$	9,06E+07	7,62E+11	5,97E+15	3,74E+08	6,18E+12	9,59E+16	8,01E+08	2,17E+13	4,97E+17
SPEA	9,18E+07	7,75E+11	5,98E+15	3,68E+08	6,053E+12	8,97E+16	7,73E+08	2,06E+13	4,72E+17
NSGA	8,78E+07	7,23E+11	5,53E+15	3,42E+08	5,48E+12	8,09E+16	7,13E+08	1,84E+13	-
NPGA	8,52E+07	7,01E+11	5,36E+15	3,27E+08	5,24E+12	7,78E+16	6,76E+08	1,75E+13	3,82E+17
HLGA	8,31E+07	6,42E+11	4,68E+15	3,21E+08	5,26E+12	6,51E+16	6,64E+08	1,56E+13	3,18E+17
VEGA	8,56E+07	6,93E+11	5,28E+15	3,35E+08	4,72E+12	7,75E+16	6,96E+08	1,77E+13	-
SPEA2	-	-	-	-	-	-	8,21E+08	2,21E+13	4,97E+17
NSGAII	-	-	-	-	-	-	8,09E+08	2,19E+13	4,92E+17

TAB. 7.7 – Comparaison de m-ACO$_6$(1,m) avec les algorithmes génétiques suivant l'hypervolume

Ces résultats se confirment aussi en revenant sur les constatations tirées de l'analyse visuelle des surfaces de compromis. En effet, nous avons trouvé que la surface procurée par m-ACO_6(1,m) est toujours au dessus de tous les autres AG, mais nous avons noté que SPEA, SPEA2 at NSGAII trouvent des solutions qui sont mieux réparties sur le front Pareto.

7.5.6 Analyse globale

Dans cette section nous essayons de tirer quelques conclusions générales résumant toutes les analyses réalisées dans cette étude expérimentale.

Dans la première partie de cette étude, nous avons comparé les six variantes proposées de m-ACO. Nous rappelons que les trois variantes m-ACO_2(m,m+1), m-ACO_3(m,m) m-ACO_4(1,1) reprennent respectivement les schémas généraux des algorithmes MOACO de la littérature CompetAnts [Doerner *et al.* , 2003], Unsort-Bicreterion [Iredi *et al.* , 2001] et MACS [Barán & Schaerer, 2003]. Les trois autres variantes m-ACO_1(m,m+1), m-ACO_5(1,m) et m-ACO_6(1,m) représentent de nouvelles approches que nous avons proposées.

En comparant ces différentes variantes, nous avons trouvé que la nouvelle variante m-ACO_6(1,m) trouve les meilleurs résultats. Nous avons comparé ces variantes suivant trois manières différentes. D'abord, suivant l'analyse visuelle des surfaces de compromis procurées sur les instances biobjectives, nous avons trouvé que m-ACO_6(1,m) retourne la surface qui est au dessus de toutes les autres surfaces. Nous avons constaté aussi que les deux variantes m-ACO_4(1,1) et m-ACO_5(1,m) trouvent des surfaces plus proches que les autres variantes. Cependant, m-ACO_3(m,m) obtient des solutions bien réparties sur le front Pareto. Tandis que m-ACO_1(m,m+1) et surtout m-ACO_2(m,m+1) trouvent des solutions qui sont sur l'extrémité du front.

Ces informations visuelles on été confirmées par les deux métriques utilisées par la suite non seulement sur les instances biobjectives mais aussi sur les autres instances

avec 3 et 4 objectifs. D'abord, suivant la métrique C, le front de m-ACO_6(1,m) domine les fronts de toutes les autres variantes. En effet, les valeurs de la métrique C de m-ACO_6(1,m) sont toujours supérieures aux autres variantes. De plus, les variantes m-ACO_4(1,1) et m-ACO_5(1,m) trouvent des valeurs de C assez bonnes. Ensuite, suivant la métrique d'hypervolume, nous avons trouvé que m-ACO_3(m,m) trouvent les valeurs les plus élevées.

Nous pouvons expliquer, d'abord, les résultats trouvés par m-ACO_3(m,m) par le fait que cette variante favorise une diversification importante. En effet, cette variante utilise m colonies de fourmis qui utilise chacune et indépendamment sa propore structure de phéromone. D'où chaque colonie peut chercher des solutions dans une région différente. Ceci explique les solutions qui sont diversifiées et réparties sur le front Pareto. Cependant ces solutions sont largement dominées particulièrelement par les variantes m-ACO_4(1,1), m-ACO_5(1,m) et m-ACO_6(1,m).

En comparant les deux variantes m-ACO_1(m,m+1) et m-ACO_2(m,m+1), qui trouvent globalement les résultats les moins bons, nous remarquons que m-ACO_1(m,m+1) est meilleure que m-ACO_2(m,m+1) surtout pour les instances les plus difficiles. Nous rappelons que la seule différence entre ces deux variantes est dans la colonie $m+1$ qui utilise, dans la première variante, la trace de phéromone correspondant à l'objectif choisi aléatoirement à chaque étape de construction, alors que pour la deuxième, elle utilise une agrégation de toutes les traces de phéromone. Ainsi, il semble que l'utilisation d'une agrégation des traces de phéromone n'est pas un bon choix, pour cet algorithme et pour le MOKP.

La variante m-ACO_4(1,1), qui reprend l'idée de base de l'algorithme MACS [Barán & Schaerer, 2003], trouve des résultats qui sont assez bons. D'ailleurs, dans [Garcia-Martinez *et al.* , 2007] les auteurs, qui ont comparé plusieurs algorithmes MOACO sur le problème du voyageur de commerce biobjectifs, ont trouvé que cet algorithme trouve parmi les meilleurs résultats. Cette variante utilise les solutions

du front Pareto pour la mise à jour de phéromone.

Les deux nouvelles variantes m-ACO$_5$(1,m) et m-ACO$_6$(1,m) utilisent une idée relativement nouvelle. En effet, elles utilisent une seule colonie de fourmis et plusieurs structures de phéromone. De plus, les fourmis considèrent à chaque étape de construction une structure de phéromone choisie aléatoirement.

m-ACO$_6$(1,m) n'utilise de plus, par rapport à m-ACO$_5$(1,m), que la mise à jour de phéromone sur le front Pareto. Ainsi, l'apport de cette mise à jour sur la qualité des solutions générées est clair. Ceci peut expliquer aussi la qualité des solutions de m-ACO$_4$(1,1) et aussi de MACS sur le problème du voyageur de commerce [Garcia-Martinez *et al.* , 2007].

Enfin, nous pouvons conclure pour cette étude que la variante m-ACO$_6$(1,m) trouve les meilleurs résultats non seulement grâce à la nouvelle idée qu'elle utilise mais aussi grâce à la mise à jour de phéromone sur le front Pareto. En effet, m-ACO$_6$(1,m) et m-ACO$_4$(1,1) appartiennent à la classe des approches Pareto (définies aux chapitre 3). Ceci peut rejoindre les études qui ont été réalisées sur les algorithmes évolutionnaires dans la littérature qui classent les approches Pareto parmi les meilleures approches dans l'optimisation multi-objectif.

Dans la deuxième partie de cette étude, nous avons comparé m-ACO$_6$(1,m) avec plusieurs algorithmes évolutionnaires de l'état de l'état de l'art. Nous avons trouvé que m-ACO$_6$(1,m) obtient des fronts Pareto qui dominent toujours et largement tous les fronts des AG sur toutes les instances testées. De plus, les fronts Pareto générés par ces AG ne dominent jamais ceux de m-ACO$_6$(1,m). Il est à noter que tous les AG comparés sont des approches Pareto, sauf VEGA et HLGA, qui trouvent d'ailleurs les résultats les moins bons.

7.6 Conclusion

Nous avons appliqué, dans ce chapitre, les six variantes de l'algorithme générique m-ACO sur le problème du sac à dos multi-objectif. Nous avons testé et comparé les différentes variantes sur plusieurs instances benchmarks du MOKP. Pour comparer ces différentes variantes, nous avons procédé d'abord à une analyse visuelle des surfaces de compromis procurées sur les instances bi-objectifs. Par la suite, nous avons utilisé deux métriques de performance différentes : la métrique C et l'hypervolume. Les résultats trouvés par les deux métriques ont confirmé les constatations tirées par l'analyse visuelle des surfaces de compromis et qui étaient en faveur de la nouvelle variante proposée m-ACO_6(1,m).

m-ACO_6(1,m) utilise une idée relativement nouvelle puisqu'elle utilise une seule colonie de fourmis et m structures de phéromone, i.e. une structure pour chaque objectif. De plus, les fourmis considèrent à chaque étape de construction une structure de phéromone choisie aléatoirement. Cependant, nous avons constaté que cette nouvelle variante trouve les meilleurs résultats grâce à cette nouvelle idée et aussi grâce à la mise à jour de phéromone sur le front Pareto.

En effet, nous avons pu conclure que pour les algorithmes MOACO, les approches Pareto trouvent généralement les meilleurs résultats.

En comparant m-ACO_6(1,m) avec plusieurs algorithmes génétiques de l'état de l'art, nous avons trouvé que m-ACO_6(1,m) est nettement meilleur sur toutes les instances testées.

Conclusion Générale

Dans cette thèse, nous avons investigué les capacités de l'optimisation par colonies de fourmis pour la résolution des problèmes d'optimisation combinatoire et multi-objectif. Après une étude de l'état de l'art des problèmes d'optimisation combinatoire et multi-objectif, nous avons proposé une taxonomie des algorithmes fourmis présentés dans la littérature pour résoudre des problèmes de ce type. Cette taxonomie classe les différents algorithmes suivant la définition de quatre points : les strucures phéromone, le facteur phéromone, le facteur heuristique et les solutions à récompenser.

Nous avons mené, par la suite, une étude de différentes stratégies phéromonales que nous avons appliquées sur le problème du sac à dos multidimensionnel (MKP) pour le cas uni-objectif. Nous avons proposé trois versions différentes d'un algorithme générique que nous avons défini. Les trois versions diffèrent suivant les composants sur lesquels les traces de phéromone sont déposées : Vertex-AK où la phéromone est déposée sur les objets, Path-AK où la phéromone est déposée sur les couples d'objets sélectionnés consécutivement, et Edge-AK où la phéromone est déposée sur les paires d'objets sélectionnés dans une même solution.

Les expérimentations ont montré que Path-AK obtient de bien moins bons résultats que Vertex-AK et Edge-AK et que Edge-AK obtient des résultats légèrement meilleurs que Vertex-AK. Cependant, Edge-AK prend plus de temps CPU que Vertex-AK lors de l'exécution, puisque la phase de mise à jour de phéromone est

de complexité quadratique pour Edge-AK alors qu'elle est de complexité linéaire pour Vertex-AK. Nous avons étudié aussi l'influence des paramètres de l'algorithme fourmi sur la qualité des solutions construites ainsi que sur la similarité de ces solutions.

Nous nous sommes basés sur cette étude, pour le cas multi-objectif, pour choisir la stratégie phéromonale appropriée, et aussi pour choisir les valeurs des paramètres de l'algorithme ACO.

Nous avons proposé par la suite un algorithme générique, appelé m-ACO, basé sur l'optimisation par colonies de fourmis pour résoudre des problèmes d'optimisation multi-objectifs. Cet algorithme est paramétré par le nombre de colonies de fourmis et le nombre de structures de phéromone. En effet, nous considérons que le nombre de colonies de fourmis peut être différent du nombre de structures de phéromone. Cet algorithme permet de tester et de comparer, dans un même cadre, différentes approches existantes ainsi que de nouvelles approches indépendamment du schéma ACO considéré et des détails d'implémentation. Nous avons proposé six variantes de cet algorithme. Parmi ces variantes, nous avons proposé trois nouvelles approches, et nous avons repris le schéma général de trois autres algorithmes MOACO présentés dans la littérature.

Nous avons appliqué et testé les différentes variantes, par la suite, sur le problème du sac à dos multi-dimensionnel multi-objectif. Aprés une étude comparative des différentes variantes suivant différentes mesures de performance, nous avons trouvé que la nouvelle variante m-ACO_6(1,m) trouve globalement les meilleurs résultats. Cette variante utilise une idée relativement nouvelle puisqu'elle utilise une seule colonie de fourmis et une structure de phéromone pour chaque objectif, et les fourmis considèrent aléatoirement à chaque étape de construction un objectif à optimiser. De plus, la mise à jour de phéromone se fait non seulement sur les meilleures solutions trouvées pour chaque objectif, comme dans la variante m-ACO_5(1,m), mais aussi

sur le front Pareto.

Nous avons pu constater l'apport de la mise à jour de phéromone sur le front Pareto en comparant les deux variantes m-ACO_5(1,m) et m-ACO_6(1,m). Nous avons remarqué aussi que la variante m-ACO_4(1,1), qui utilise une approche Pareto, trouve des résultats assez bons et proches pour plusieurs instances de m-ACO_6(1,m). En effet, l'algorithme MACS, que m-ACO_4(1,1) reprend l'idée de base, trouve aussi parmi les meilleurs résultats comparé avec plusieurs algorithmes MOACO de l'état de l'art dans [Garcia-Martinez *et al.* , 2007] pour le problème du voyageur de commerce. Ainsi, nous avons pu conclure que pour les algorithmes MOACO, tout comme les algorithmes génétiques, les approches Pareto semblent trouver les meilleurs résultats.

Nous avons comparé par la suite la variante m-ACO_6(1,m) avec plusieurs algorithmes génétiques de l'état de l'art, qui utilisent pour la plupart une approche Pareto. Nous avons trouvé que m-ACO_6(1,m) est nettement meilleure sur toutes les instances testées.

Nos futurs travaux de recherche essayeront d'appliquer l'algorithme m-ACO à d'autres problèmes d'optimisation mutli-objectif. Nous nous sommes focalisés durant cette thèse sur des problèmes de type sac à dos. Nous envisageons de traiter d'autres problèmes comme par exemple le problème du voyageur de commerce multi-objectif.

Nous avons proposé six variantes de l'algorithme générique m-ACO. D'autres variantes peuvent être proposées pour cet algorithme qui, soit proposent de nouvelles idées, soit reprennent d'autres algorithmes MOACO de l'état de l'art.

Nous avons comparé nos approches avec les algorithmes évolutionnaires les plus connus et les plus performants de l'état de l'art. Une autre perspective serait de nous comparer avec d'autres algorithmes basés sur d'autres métaheuristiques de l'état de l'art comme le recuit simulé ou la recherche taboue.

Table des figures

Liste des tableaux

Bibliographie

[Alaya *et al.* , 2003] Alaya, I., Sammoud, O., Hammami, M., & Ghédira, K. 2003. Ant Colony Optimization for the k-Graph Partitioning Problem. *In : The 3rd Tunisia-Japan Symposium on Science and Technology, TJASST2003.*

[Alaya *et al.* , 2004a] Alaya, I., Solnon, C., & Ghédira, K. 2004a. Ant algorithm for the multi-dimensional knapsack problem. *Pages 63–72 of : Proceedings of International Conference on Bioinspired Optimization Methods and their Applications (BIOMA 2004).*

[Alaya *et al.* , 2004b] Alaya, I., Solnon, C., & Ghédira, K. 2004b. Des fourmis pour le sac à dos multidimensionnel. *Pages 29–30 of : Actes de la Conférence Internationale en Recherche Opérationnelle (FRANCORO IV).*

[Alaya *et al.* , 2005] Alaya, I., Solnon, C., & Ghédira, K. 2005. Différentes stratégies phéromonales pour le sac à dos multidimensionnel. *Pages 151–160 of : Méthodologies et Heuristiques pour l'Optimisation des Systèmes Industriels(MHOSI 2005).*

[Alaya *et al.* , 2006] Alaya, I., Solnon, C., & Ghédira, K. 2006. Algorithme fourmis hybride pour le sac à dos multidimensionnel. *In : Métaheuristiques (META 2006).*

[Alaya *et al.* , 2007a] Alaya, I., Solnon, C., & Ghédira, K. 2007a. Ant Colony Optimization for Multi-objective Optimization Problems. *Pages 450–457 of : 19th*

IEEE International Conference on Tools with Artificial Intelligence (ICTAI'07).

[Alaya *et al.* , 2007b] Alaya, I., Solnon, C., & Ghédira, K. 2007b. Optimisation par colonies de fourmis pour le problème du sac à dos multidimensionnel. *Techniques et Sciences Informatique*, **26**(3-4), 371–390.

[Angus, 2007] Angus, D. 2007. Crowding population-based ant colony optimisation for the multi-objective travelling salesman problem. *Pages 333–340 of : In 2007 IEEE Symposium on Computational Intelligence in Multi-Criteria Decision-Making (MCDM 2007)*. IEEE.

[Barichard & Hao, 2003] Barichard, V., & Hao, J.K. 2003. Genetic tabu search for the multi-objective knapsack problem. *Tsinghua Science and Technology*, **8**(1), 8–13.

[Barán & Schaerer, 2003] Barán, B., & Schaerer, M. 2003. A Multiobjective Ant Colony System for Vehicle Routing Problem with Time Windows. *Pages 97–102 of : Proc. Twenty first IASTED International Conference on Applied Informatics, Insbruck, Austria.*

[Ben Abdelaziz & Krichen, 1997] Ben Abdelaziz, F., & Krichen, S. 1997. *A tabu search heuristic for multiple objective knapsack problems.* Ructor Research Report RR 28-97.

[Brockhoff *et al.* , 2008] Brockhoff, D., Friedrich, T., & Neumann, F. 2008. Analyzing Hypervolume Indicator Based Algorithms. *Pages 651–660 of : Conference on Parallel Problem Solving From Nature (PPSN X).* Springer.

[Bullnheimer *et al.* , 1999] Bullnheimer, B., Hartl, R.F., & Strauss, C. 1999. An Improved Ant system Algorithm for the Vehicule Routing Problem. *Annals of Operations Research*, **89**, 319–328.

[Cardoso *et al.* , 2003] Cardoso, P., Jesus, M., & Márquez, A. 2003. MONACO - Multi-Objective Network Optimisation based on ACO. *In : In X En-*

cuentros de Geometría Computacional, Seville, Spain.

[Carraway *et al.* , 1990] Carraway, R. L., Morin, T. L., & Moskowitz, H. 1990. Generalized dynamic programming for multicriteria optimization. *European Journal of Operational Research*, **44**, 95–104.

[Chu & Beasley, 1998] Chu, P.C., & Beasley, J.E. 1998. A genetic algorithm for the multidimentional knapsack problem. *Journal of heuristics*, **4**, 63–86.

[Collette & Siarry, 2002] Collette, Y., & Siarry, P. 2002. *Optimisation multiobjectif.* Eyrolles.

[Czyzak & Jaszkiewicz, 1998] Czyzak, P., & Jaszkiewicz, A. 1998. Pareto simulated annealing a metaheuristic technique for multiple-objective combinatorial optimisation. *Journal of Multi-Criteria Decision Analysis*, **7**, 34–47.

[Dahl *et al.* , 1995] Dahl, G., Jornsten, K., & Lokketangen, A. 1995. A tabu search approach to the channel minimization problem. *Pages 369–377 of :* In G. Liu, K-H. Phua, J. Ma J. Xu F. Gu, & C. He, editors (eds), *Optimization- Techniques ans Applications, ICOTA'95*, vol. 1. Chengdu, China : World Scientific.

[Deb *et al.* , 2002] Deb, K., Pratap, A., Agarwal, S., & Meyarivan, T. 2002. A Fast and Elitist Multiobjective Genetic Algorithm : NSGA-II. *Pages 182–197 of : IEEE Transactions on Evolutionary Computation*, vol. 6 :2.

[Deneubourg *et al.* , 1983] Deneubourg, J.L, Goss, S, & Verhaeghe, J.C. 1983. Probabilistic behaviour in ants. *Journal of Theorical Biology*, **105**, 259–271.

[Doerner *et al.* , 2003] Doerner, K., Hartl, R. F., & Teimann, M. 2003. Are COMPETants More Competent for Problem Solving ? The Case of Full Truckload Transportation. *Central European Journal of Operations Research*, **11**(2), 115–141.

[Doerner *et al.* , 2004] Doerner, K., Gutjahr, W. J., Hartl, R. F., Strauss, C., & Stummer, C. 2004. Pareto Ant Colony Optimization : A Metaheuristic Approach to Multiobjective Portfolio Selection. *Annals of Operations Research.*

[Dorigo, 1992] Dorigo, M. 1992. *Optimization, Learning and Natural Algorithms* (in Italian). Ph.D. thesis, Dipartimento di Elettronica, Politecnico di Milano, Italy.

[Dorigo & Di Caro, 1999] Dorigo, M., & Di Caro, G. 1999. The Ant Colony Optimization Meta-Heuristic. *Pages 11–32 of :* Corne, D., Dorigo, M., & Glover, F. (eds), *New Ideas in Optimization.* McGraw Hill, UK.

[Dorigo & Gambardella, 1997] Dorigo, M., & Gambardella, L.M. 1997. Ant Colony System : A Cooperative Learning Approach to the Traveling Salesman Problem. *IEEE Transactions on Evolutionary Computation*, **1**(1), 53–66.

[Dorigo & Stüzle, 2004] Dorigo, M., & Stüzle, T. 2004. *Ant Colony Optimization.* Springer-Verlag.

[Dorigo *et al.* , 1996] Dorigo, M., Maniezzo, V., & Colorni, A. 1996. The Ant System : Optimization by a Colony of Cooperating Agents. *IEEE Transactions on Systems, Man, and Cybernetics – Part B*, **26**(1), 29–41.

[Fidanova, 2002] Fidanova, S. 2002. Evolutionary Algorithm for Multidimensional Knapsack Problem. *In : Proceedings of PPSNVII.*

[Fonseca & Fleming, 1993] Fonseca, C.M., & Fleming, P.J. 1993. Genetic algorithms for multi-objective optimization : formulation, discussion and generalization. *Pages 416–423 of : The Fifth International Conference on Genetic Algorithms.*

[Friesz *et al.* , 1993] Friesz, T. L., Anandalingam, G., Mehta, N. J., Nam, K., Shah, S. J., & Tobin, R. L. 1993. The multiojective equilibrium network

design problem revisited : A simulated annealing approach. *European Journal of Operational Research*, 44–57.

[Gambardella *et al.* , 1999a] Gambardella, L., Taillard, E., & Dorigo, M. 1999a. Ant Colonies for the Quadratic Assignment Problem. *Journal of the Operational Research Society*, **50**, 167–176.

[Gambardella & Dorigo, 1995] Gambardella, L. M., & Dorigo, M. 1995. Ant-Q : A Reinforcement Learning Approach to the Traveling Salesman Problem. *Pages 252–260 of : Proc. Twelfth International Conference on Machine Learning (ML-95).*

[Gambardella *et al.* , 1999b] Gambardella, L.M., Taillard, E.D., & Agazzi, G. 1999b. MACS-VRPTW : A Multiple Ant Colony System for Vehicle Routing Problems with Time Windows. *Pages 63–76 of :* Corne, D., Dorigo, M., & Glover, F. (eds), *New Ideas in Optimization.* McGraw Hill, London, UK.

[Gandibleux *et al.* , 1997] Gandibleux, X., Mezdaoui, N., & Fréville, A. 1997. A multiobjective tabu search procedure to solve combinatorial optimization problems. *Advances in Multiple Objective and Goal Programming. Lecture Notes in Economics and Mathematical Systems*, **455**, 291–300.

[Garcia-Martinez *et al.* , 2007] Garcia-Martinez, C., Cordon, O., & Herrera, F. 2007. A taxonomy and an empirical analysis of multiple objective ant colony optimization algorithms for bi-criteria tsp. *Pages 116–148 of : European Journal of Operational Research.*

[Glover, 1986] Glover, F. 1986. Future Paths for Integer Programming and Links to Artificial Intelligence. *Computers and Operations Research*, **13**(5), 533–549.

[Goldberg, 1989] Goldberg, D.E. 1989. Genetic algorithms for search, optimization, and machine learning. *In :* Addison-Wesley, MA : (ed), *Reading.*

[Gottlieb, 2000] GOTTLIEB, J. 2000. Permutation-based evolutionary algorithms for multidimensional knapsack problems. *Pages 408–414 of : Proceedings of the 2000 ACM symposium on Applied computing.*

[Gravel *et al.* , 2002] GRAVEL, M., PRICE, W. L., & GAGNÉ, C. 2002. Scheduling Continuous Casting of Aluminium using a Multiple Objective Ant Colony Optimization Metaheuristic. *European Journal of Operational Research*, **143**(1), 218–229.

[Guntsch & Middendorf, 2002] GUNTSCH, M., & MIDDENDORF, M. 2002. A population based approach for ACO. *Page 7281 of :* IN S. CAGNONI, J. GOTTLIEB, E. HART M. MIDDENDORF, & G. R. RAIDL, EDITORS (eds), *EvoWorkshops*. Springer-Verlag.

[Hajela & Lin, 1992] HAJELA, P., & LIN, C-Y. 1992. Genetic search strategies in multicriterion optimal design. *Structural Optimization*, 99–107.

[Hansen, 1998] HANSEN, M.P. 1998. *Metaheuristics for multiple objective combinatorial optimization*. Ph.D. Thesis, Technical University of Denmark, Lyngby.

[Holland, 1992] HOLLAND, JOHN H. 1992. *Adaptation in Natural and Artificial Systems : An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence*. The MIT Press.

[Horn *et al.* , 1994] HORN, J., NAFPLIOTIS, N., & GOLDBERG, D.E. 1994. A niched pareto genetic algorithm for multiobjective optimization. *Pages 82–87 of :* CENTER, IEEE SERVICE (ed), *First IEEE Conference on Evolutionary Computation*, vol. 1. Piscataway : IEEE World Congress on Computational Computation.

[Iredi *et al.* , 2001] IREDI, S., MERKLE, D., & MIDDENDORF, M. 2001. Bi-Criterion Optimization with Multi Colony Ant Algorithms. *Pages 359–372 of : First International Conference on Evolutionary Multi-criterion Optimization (EMO'01)*, vol. 1993. Lecture Notes in Computer Science.

[Jaszkiewicz, 2002a] Jaszkiewicz, A. 2002a. Genetic local search for multiple objective combinatorial optimization. *European Journal of Operational Research*, **137**(1), 50–71.

[Jaszkiewicz, 2002b] Jaszkiewicz, A. 2002b. On the Performance of Multiple-Objective Genetic Local Search on the 0/1 Knapsack Problem-A Comparative Experiment. *IEEE Transactions on Evolutionary Computation*, **6**(4), 402–412.

[Jong, 1975] Jong, K.A. De. 1975. *An analysis of the behaviour of a class of genetic adaptive systems.* PhD thesis, University of Michigan.

[Kirkpatrick *et al.* , 1983] Kirkpatrick, S., Gelatt, C. D., & Vecchi, M. P. 1983. Optimization by Simulated Annealing. *Science*, **220**(4598), 671–680.

[Leguizamon & Michalewicz, 1999] Leguizamon, G., & Michalewicz, Z. 1999. A new version of Ant System for Subset Problem. *Pages 1459–1464 of : Proceedings of Congress on Evolutionary Computation.*

[Mariano & Morales, 1999] Mariano, C. E., & Morales, E. 1999 (June). *A Multiple Objective Ant-Q Algorithm for the Design of Water Distribution Irrigation Networks.* Tech. rept. HC-9904. Instituto Mexicano de Tecnologa del Agua, Mexico.

[McMullen, 2001] McMullen, P. R. 2001. An Ant Colony Optimization Approach to Addressing a JIT Sequencing Problem with Multiple Objectives. *Artificial Intelligence in Engineering*, **15**(3), 309–317.

[Metropolis *et al.* , 1953] Metropolis, N., Rosenbluth, A., Rosenbluth, M., Teller, A., & Teller, E. 1953. Equation of State Calculations by Fast Computing Machines. *J. Chem. Phys.*, **21**, 1087–1092.

[Morrison & DeJong, 2001] Morrison, R.W, & DeJong, K.A. 2001. Measurement of population diversity. *Pages 31–41 of : In 5th International Conference EA 2001*, vol. 2310 of LNCS. Springer-Verlag.

[Moyson & Manderick, 1988] Moyson, & Manderick. 1988. The Collective Behaviour of Ants : an Example of Self Organisation in Massive Parallelism. *In : Proceedings of AAAI Spring Symposium on Parallel Models of Intelligence.*

[Oei *et al.* , 1991] Oei, C. K., Goldberg, D., & Chang, S6J. 1991. *Tournament delection, niching, and the preservation of diversity.* Tech. rept. University of Illinois, Urbana-Champaign.

[Papadimitriou, 1994] Papadimitriou, C. 1994. *Computational Complexity.* AddisonWesley.

[Sayin & Karabati, 1999] Sayin, S., & Karabati, S. 1999. A bicriteria approach to the two-machine flow shop scheduling problem. *European Journal of Operational Research*, **133**, 435–449.

[Schaffer, 1985] Schaffer, J.D. 1985. Multiple objective optimization with vector evaluated genetic algorithms. *Pages 93–100 of :* Grefenstette, J.J (ed), *ICGA International Conference on Genetic Algorithms.* Lecture Notes in Computer Science.

[Sen *et al.* , 1988] Sen, T., Raiszadeh, M. E., & Dileepan, P. 1988. A branch and bound approach to the bicriterion scheduling problem involving total flowtime and range of lateness. *Management Science*, **34**(2), 254–260.

[Serafini, 1992] Serafini, P. 1992. Simulated annealing for multiple objective optimization problems. *Pages 87–96 of : Tenth Int. Conf. on Multiple Criteria Decision Making.*

[Silverman, 1986] Silverman, B. W. 1986. Density Estimation for Statistics and Data Analysis.

[Solnon, 2002] Solnon, C. 2002. Ants can Solve Constraint Satisfaction Problems. *IEEE Transactions on Evolutionary Computation*, **6**(4), 347–357.

[Srinivas & Deb, 1994] Srinivas, N., & Deb, K. 1994. Multiobjective optimization using non dominated sorting in genetic algorithms. *Evolutionary Computation*, **2**, 221–248.

[Stewart & White, 1991] Stewart, B. S., & White, C. C. 1991. Multiobjective A*. *Journal of the ACM*, **38**(4), 775–814.

[Stützle & Hoos, n.d.] Stützle, T., & Hoos, H.H. The MAX-MIN Ant System and local search for the traveling salesman problem. *Pages 309–314 of :* B T. (ed), *Proceedings of the 1997 IEEE International Conference on Evolutionary Computation (ICEC97).*

[Stützle & Hoos, 2000] Stützle, T., & Hoos, H.H. 2000. $\mathcal{MAX} - \mathcal{MIN}$ Ant System. *Journal of Future Generation Computer Systems*, **16**, 889–914.

[Suppapitnarm & Parks, 2001] Suppapitnarm, A., & Parks, T. 2001. Simulated annealing : an alternative approach to true multiobjective optimization. *In : Genetic and Evolutionary Computation Conference.*

[Ulungu *et al.* , 1998] Ulungu, E. L., Teghem, J., Frtemps, P., & Tuyttens, D. 1998. *MOSA method : A tool for solving multi-objective combinatorial optimization problems.* Tech. rept. Laboratory of Mathematic and Operational Research, Faculté polytechnique de Mons.

[Ulungu *et al.* , 1999] Ulungu, E.L., Teghem, J., Fortemps, Ph, & Tuyttens, D. 1999. MOSA method : a tool for solving multiobjective combinatorial optimization problems. *Journal of MultiCriteria Decision Analysis*, **8**, 221–236.

[Visée *et al.* , 1998] Visée, M., Teghem, J., Pirlot, M., & Ulungo, E. L. 1998. Two-phases method and branch and bound procedures to solve knapsack problem. **12**, 139–155.

[White, 1982] White, D. J. 1982. The set of efficient solutions for multiple-objective shortest-path problems. *Computers and Operations Research*, **9**, 101–

107.

[Zitzler & Thiele, 1999] Zitzler, E., & Thiele, L. 1999. Multiobjective evolutionary algorithms : a comparative case study and the strength Pareto approach. *IEEE Transactions on Evolutionary Computation*, 257–271.

[Zitzler *et al.* , 2000] Zitzler, E., Deb, K., & Thiele, L. 2000. Comparison of Multiobjective Evolutionary Algorithms : Empirical Results. *Evolutionary Computation*, **8**(2), 173–195.

[Zitzler *et al.* , 2001] Zitzler, E., Laumanns, M., & Thiele, L. 2001. SPEA2 : Improving the Strength Pareto Evolutionary Algorithm. *Pages 12–21 of : In : K. Giannakoglou et al. (Eds.) EUROGEN 2001, Evolutionary Methods for Design, Optimization and Control with Applications to Industrial Problems, Athens, Greece.*

Résumé

Dans cette thèse, nous nous intéressons à l'étude des capacités de la métaheuristique d'optimisation par colonie de fourmis (*Ant Colony Optimization* - ACO) pour résoudre des problèmes d'optimisation combinatoire multi-objectif. Dans ce cadre, nous avons proposé une taxonomie des algorithmes ACO proposés dans la littérature pour résoudre des problèmes de ce type. Nous avons mené, par la suite, une étude expérimentale de différentes stratégies phéromonales pour le cas du problème du sac à dos multidimensionnel mono-objectif. Enfin, nous avons proposé un algorithme ACO générique pour résoudre des problèmes d'optimisation multi-objectif. Cet algorithme est paramétré par le nombre de colonies de fourmis et le nombre de structures de phéromone considérées. Il permet de tester et de comparer, dans un même cadre, plusieurs approches. Nous avons proposé six variantes de cet algorithme dont trois présentent de nouvelles approches et trois autres reprennent des approches existantes. Nous avons appliqué et comparé ces variantes au problème du sac à dos multidimensionnel multi-objectif.

Mot clés : optimisation par colonies de fourmis, optimisation combinatoire, problèmes multi-objectif, stratégies phéromonales, problèmes sac à dos

Abstract

In this thesis, we investigate the capabilities of Ant Colony Optimization (ACO) metaheuristic to solve combinatorial and multi-objective optimization problems. First, we propose a taxonomy of ACO algorithms proposed in the literature to solve multi-objective problems. Then, we study different pheromonal strategies for the case of mono-objective multidimensional knapsack problem. We propose, finally, a generic ACO algorithm to solve multi-objective problems. This algorithm is parameterised by the number of ant colonies and the number of pheromone structures. This algorithm allows us to evaluate and compare new and existing approaches in the same framework. We compare six variants of this generic algorithm on the multi-objective multidimensional knapsack problem.

Key words: ant colony optimization, combinatorial optimization, multi-objective problems, pheromone strategies, knapsack problems