Optimisation par colonies de fourmis pour le problème du sac à dos multidimensionnel

Inès Alaya* — **Christine Solnon**** — **Khaled Ghédira***

* *SOIE, Institut Supérieur de Gestion de Tunis*
41 Rue de la Liberté- Cité Bouchoucha-2000 Le Bardo-Tunis
{ines.alaya,khaled.ghedira}@isg.rnu.tn

** *LIRIS, CNRS FRE 2672, University Lyon 1*
43 bd du 11 novembre, 69622 Villeurbanne cedex
christine.solnon@liris.cnrs.fr

RÉSUMÉ. *Dans cet article, nous proposons d'utiliser la métaheuristique d'optimisation par colonies de fourmis (Ant Colony Optimization / ACO) pour résoudre le problème du sac à dos multidimensionnel. L'objectif est de sélectionner un sous-ensemble d'objets qui maximise une fonction utilité donnée tout en respectant certaines contraintes de ressources. Nous proposons un algorithme ACO générique pour ce problème. L'idée est de construire des solutions de façon incrémentale, par ajouts successifs d'objets à une solution partielle. A chaque itération, l'objet à ajouter est choisi selon une probabilité dépendant de traces de "phéromone" et d'une information heuristique locale. On étudie trois façons de déposer (et d'exploiter) les traces de phéromone : sur les objets sélectionnés, sur les couples d'objets sélectionnés consécutivement ou sur tous les couples de sommets sélectionnés. On compare le comportement de ces trois variantes sur un ensemble d'instances "benchmarks" et on étudie l'influence de la phéromone sur le processus de résolution. On compare enfin l'algorithme ACO proposé avec d'autres approches.*

ABSTRACT. *We propose an algorithm based on the Ant Colony Optimization (ACO) meta-heuristic for solving the Multidimensional Knapsack Problem (MKP), the goal of which is to find a subset of objects that maximizes a given objective function while satisfying some resource constraints. The proposed ACO algorithm is generic, and we propose three different instantiations, corresponding to three different ways of laying and exploiting pheromone trails. We experimentally compare these three different instanciations. We then experimentally compare our ACO algorithm with two state-of-the-art genetic algorithms, showing that it obtains competitive results.*

MOTS-CLÉS : *Optimisation par colonies de fourmis, Problème du sac à dos multidimensionnel .*

KEYWORDS: *Ant colony optimization, Multidimensional knapsack problem.*

1re soumission à *Technique et science informatiques*, le 25 février 2005.

1. Introduction

Le problème du sac-à-dos multidimensionnel (MKP) est un problème NP-difficile qui possède de nombreuses applications pratiques comme l'allocation des processeurs dans les systèmes distribués, le chargement des cargaisons, le découpage de stocks. L'objectif du MKP est de trouver un sous-ensemble d'objets qui maximise un profit total tout en satisfaisant certaines contraintes de capacité. Plus formellement, un MKP est modélisé comme suit :

$$\begin{array}{rl} \text{maximiser} & \sum_{j=1}^{n} p_j x_j \\ \text{sous les contraintes} & \sum_{j=1}^{n} r_{ij} x_j \leq b_i, \forall i \in 1..m \\ & x_j \in \{0,1\}, \forall j \in 1..n \end{array}$$

où r_{ij} est la quantité de ressource i consommée par l'objet j, b_i est la quantité totale disponible de la ressource i, p_j est le profit associé à l'objet j, et x_j est la variable de décision associée à l'objet j, i.e., $x_j = 1$ si j est sélectionné et $x_j = 0$ sinon.

Plusieurs travaux ont été développés pour résoudre le MKP. Les méthodes exactes, généralement basées sur une approche par séparation et évaluation sont limitées à des instances de petites tailles, ce qui justifie le recours aux méthodes heuristiques.

1.1. *Approches heuristiques*

De nombreux algorithmes heuristiques ont été proposés pour résoudre le MKP. Ces algorithmes explorent l'espace de recherche du problème de façon incomplète et opportuniste ; ils trouvent généralement de bonnes solutions rapidement mais ne garantissent pas l'optimalité des solutions trouvées. Beaucoup de ces algorithmes sont basés sur une approche gloutonne : partant d'un sac-à-dos vide, des objets sont itérativement sélectionnés en respectant un critère gradient, et en assurant la faisabilité de la solution.

D'autres algorithmes approchés combinent la programmation linéaire avec une heuristique en résolvant plusieurs relaxations du MKP. Magazine et Oguz [MAG 84] ont présenté un algorithme qui combine une heuristique duale avec l'approche du multiplicateur lagrangien généralisé en utilisant les multiplicateurs générés comme borne supérieure. Les heuristiques duales commencent avec une solution dont les variables sont initialisées à 1, ensuite les variables sont mises à 0 relativement à des règles heuristiques.

Récemment plusieurs métaheuristiques ont été développées pour le MKP. Ces métaheuristiques ont permis d'avoir des résultats très compétitifs avec des instances larges. Plusieurs algorithmes basés sur la recherche taboue ont été proposées. Glover et Kochenberger [F.G 96] ont présenté un algorithme basé sur la recherche taboue qui utilise une mémoire flexible intègrant des informations de fréquence.

Hanafi et Fréville [HAN 98] ont proposé un algorithme étroitement lié au travail de Glover et Kochenberger [F.G 96]. Ils trouvent des résultats meilleurs que ceux trouvés

par Glover et Kochenberger sur un sous-ensemble des mêmes jeux de tests. Vasquez et Hao [VAS 01] ont présenté une approche hybride qui combine la recherche taboue avec la programmation linéaire. Cet algorithme utilise une recherche taboue qui emploie la méthode d'élimination inverse pour la gestion de la liste tabou. Cette dernière méthode a été mise en œuvre pour la première fois sur le MKP par Damneyers et Voß[DAM 93]. L'algorithme proposé améliore les meilleurs résultats connus jusqu'à présent sur des instances jugées difficiles.

Plusieurs algorithmes génétiques ont été développés pour le MKP, nous citons l'algorithme de Chu et Beasley [CHU 98] qui trouve parmi les meilleurs résultats sur des instances larges de MKP. Cet algorithme incorpore, en plus des opérateurs classiques d'un algorithme génétique, un opérateur heuristique qui utilise des connaissances spécifiques au problème.

Tout récemment trois algorithmes basés sur l'optimisation par colonies de fourmis ont été proposés. Un objectif essentiel de cet article est de comparer ces trois algorithmes.

1.2. *Optimisation par colonies de fourmis*

Les fourmis sont capables de résoudre collectivement des problèmes complexes, comme trouver le plus court chemin entre deux points dans un environnement accidenté. Pour cela, elles communiquent entre elles de façon locale et indirecte, grâce à une hormone volatile, appelée phéromone : au cours de leur progression, les fourmis déposent une trace de phéromone ; elles choisissent ensuite leur chemin de façon aléatoire, selon une probabilité dépendant de la quantité de phéromone précédemment déposée.

Ce mécanisme, qui permet aux fourmis de résoudre collectivement des problèmes complexes, est à l'origine des algorithmes à base de fourmis artificielles. Ces algorithmes ont été initialement proposés dans [DOR 92, DOR 96], comme une approche multi-agents pour résoudre des problèmes d'optimisation combinatoire. L'idée est de représenter le problème à résoudre sous la forme de la recherche d'un meilleur chemin dans un graphe, puis d'utiliser des fourmis artificielles pour rechercher de bons chemins dans ce graphe. Le comportement des fourmis artificielles est inspiré des fourmis réelles : elles déposent des traces de phéromone sur les composants du graphe et elles choisissent leurs chemins relativement aux traces de phéromone précédemment déposées ; ces traces sont évaporées au cours du temps. Intuitivement, cette communication indirecte fournit une information sur la qualité des chemins empruntés afin d'attirer les fourmis, dans les itérations futures, vers les zones correspondantes de l'espace de recherche.

Ces caractéristiques du comportement des fourmis artificielles définissent la "métaheuristique d'optimisation par une colonie de fourmis" ou "Ant Colony Optimization (ACO) metaheuristic" [DOR 99]. Cette métaheuristique a permis de résoudre différents problèmes d'optimisation combinatoire, comme le problème du voyageur

de commerce [DOR 97], le problème d'affectation quadratique [GAM 99] ou le problème de routage de véhicules [BUL 99].

1.3. *Motivations et organisation de l'article*

On propose dans cet article d'étudier les capacités de la métaheuristique ACO pour résoudre le problème du sac-à-dos multidimensionnel. En effet, trois algorithmes ACO pour le MKP ont été proposés récemment. Dans ces trois algorithmes, les fourmis construisent des solutions selon une heuristique gloutonne : partant d'un sac-à-dos vide, elles sélectionnent à chaque itération un nouvel objet, cet objet étant choisi en fonction d'un critère heuristique (caractérisant la "qualité" des objets candidats) et de traces de phéromone (caractérisant l'expérience passée de la colonie). Ces trois algorithmes diffèrent dans leur façon de déposer et d'exploiter les traces de phéromone : Leguizamon et Michalewizc [LEG 99] ont proposé de déposer des traces de phéromone sur les objets sélectionnés ; Fidanova [FID 02] a proposé de déposer de la phéromone sur le chemin formé par la suite d'objets sélectionnés ; Alaya, Solnon et Ghédira [ALA 04] ont proposé de déposer de la phéromone sur les couples d'objets sélectionnés dans une même solution.

Un objectif essentiel de cet article est de comparer ces trois stratégies différentes de dépôt et exploitation des traces de phéromones pour résoudre le MKP avec la métaheuristique ACO. Ainsi, nous décrivons dans la section suivante un algorithme ACO générique pour le MKP, et nous proposons trois instantiations différentes de cet algorithme, correspondant aux trois stratégies phéromonales. Nous discutons ensuite brièvement 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'intéret 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.

2. Un algorithme ACO générique pour le MKP

Pour résoudre le MKP 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 Ant-Knapsack (AK), un algorithme ACO générique pour le MKP, et nous décrivons trois instantiations 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.

2.1. *L'algorithme générique Ant-Knapsack*

L'algorithme générique Ant-Knapsack est décrit dans la Figure 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 [STÜ 00] : on impose des bornes inférieures et supérieures τ_{min} et τ_{max} sur les traces de phéromone (avec $0 < \tau_{min} < \tau_{max}$). L'objectif est de garantir une bonne exploration de l'espace de recherche, et prévénir la colonie de fourmis de "stagner" sur une solution sub-optimale, en empêchant que les différences relatives entre les traces de phéromone ne deviennent trop importantes durant l'exécution. De plus, les traces de phéromone sont initialisées à la borne supérieure τ_{max} afin de garantir une exploration plus large de l'espace de recherche durant les premiers cycles.

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

Algorithme Ant-Knapsack :

Initialiser les traces de phéromone à τ_{max}
répéter
pour chaque fourmi k **dans** $1..nbAnts$, construire une solution $\mathcal{S}_k$ comme suit :
Choisir aléatoirement 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 sélectionné sans violer des contraintes de ressources$\}$
tant que $Candidats \neq \emptyset$ **faire**
Choisir un objet $o_i \in Candidats$ avec la probabilité
$$p_{S_k}(o_i) = \frac{[\tau_{S_k}(o_i)]^{\alpha} . [\eta_{S_k}(o_i)]^{\beta}}{\sum_{o_j \in Candidates} [\tau_{S_k}(o_j)]^{\alpha} . [\eta_{S_k}(o_j)]^{\beta}}$$
$\mathcal{S}_k \leftarrow \mathcal{S}_k \cup \{o_i\}$
enlever de $Candidats$ chaque objet qui viole des contraintes de ressources
fin tant que
fin pour
mettre à jour les traces de phéromone en fonction de $\{\mathcal{S}_1, \ldots, \mathcal{S}_{nbAnts}\}$
si une trace de phéromone est inférieure à τ_{min} **alors** la mettre à τ_{min}
si une trace de phéromone est supérieure à τ_{max} **alors** la mettre à τ_{max}
jusqu'à nombre maximal de cycles atteint **ou** solution optimale trouvée

Figure 1. *Algorithme ACO générique pour le MKP*

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 [LEG 99] et [ALA 04] : 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

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

2.2. *Définition des composants phéromonaux*

Les fourmis de l'algorithme générique 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 $\tau(o_i, o_j) = \tau(o_j, o_i)$.

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 depend 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 $\tau(o_i, o_j)$.

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 $1/(1 + \mathrm{profit}(S_{\mathrm{best}}) - \mathrm{profit}(S_k))$. 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 $\mathcal{O}(|S_k|^2)$

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).

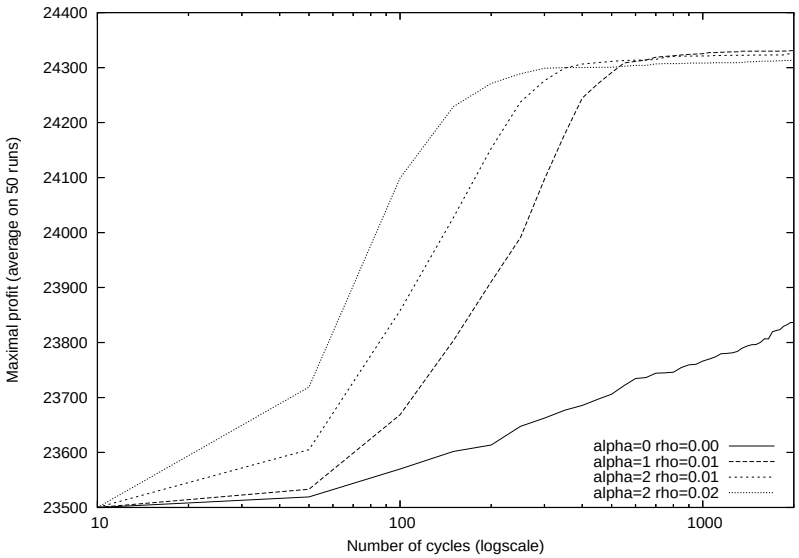

Figure 2. *Influence de α et ρ sur la qualité des solutions trouvées par Edge-AK pour une instance du MKP ayant 100 objets et 5 contraintes de ressource : chaque courbe trace l'évolution du profit de la meilleure solution quand le nombre de cycles augmente pour une affectation donnée de α et ρ (moyenne sur 50 exécutions). Les autres paramètres ont été affectés à $\beta = 5$, $nbAnts = 30$, $\tau_{min} = 0.01$, and $\tau_{max} = 6$.*

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 comportement des fourmis est identique pour les trois versions proposées de Ant-Knapsack. Nous l'illustrons dans la Figure 2 pour Edge-AK ; des courbes très similaires auraient été générées pour Vertex-AK et Path-AK. 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 glou-

ton classique, les solutions trouvées sont très nettement moins bonnes que lorsque la phéromone est utilisée.

4. Expérimentations et résultats

Nous présentons dans cette section les résultats des trois instances de l'algorithme générique Ant-Knapsack, i.e., Vertex-AK, Path-AK et Edge-AK. Nous comparons aussi ces résultats avec ceux de Leguizemon et Michalewicz [LEG 99] et de Fidanova [FID 02] ainsi qu'avec deux approches de l'état de l'art sur le sujet.

4.1. *Ensemble de tests et conditions d'expérimentation*

Nous avons considéré des instances larges de MKP OR-Library accessibles à partir du site `http ://mscmga.ms.ic.ac.uk/`. Ces instances sont groupées en 5 classes avec $m \in \{5, 10, 30\}$ et $n \in \{100, 250\}$. Ces instances sont notées dans ce qui suit `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.

4.2. *Comparaison des algorithmes Vertex-AK, Path-AK et Edge-AK*

Le tableau 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 (ou un écart-type) sur 100 exécutions.

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 précise 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.

Tableau 1. *Comparaison de la qualité des solutions trouvées par Vertex-AK, Path-AK et Edge-AK sur les 15 classes de problèmes considérées. 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 (moyenne et écart-type sur 10 exécutions pour chacune des 10 instances de la classe).*

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

Le tableau 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. 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.

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

Tableau 2. *Comparaison des temps d'exécution de Vertex-AK, Path-AK et Edge-AK sur les les 15 classes de problèmes considérées. 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 et le nombre de cycles nécessaires pour trouver la solution (moyenne et écart-type sur 10 exécutions de chaque instance de la classe).*

m.n	rd	**Vertex-AK**				**Path-AK**				**Edge-AK**			
		Temps		Nb cycles		Temps		Nb cycles		Temps		Nb cycles	
		Moy	Ecart	Moy	Ecart	Moy	Ecart	Moy	Ecart	Moy	Ecart	Moy	Ecart
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	39,40	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

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.

4.3. *Comparaison avec d'autres algorithmes ACO*

L'algorithme générique Ant-Knapsack s'instancie en trois versions Vertex-AK, Path-AK et Edge-AK en fonction de la stratégie phéromonale choisie. L'instance Edge-AK correspond à l'algorithme décrit dans [ALA 04]. Les instances Vertex-AK et Path-AK adoptent les stratégies phéromonales respectivement proposées dans [LEG 99] et [FID 02], mais présentent quelques différences par ailleurs. Nous comparons dans cette section les résultats obtenus par ces différents algorithmes ACO pour le MKP.

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

L'algorithme Vertex-AC utilise la même stratégie phéromonal que l'algorithme proposé par Leguizamon et Michalewizc dans [LEG 99]. 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-AC 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 [DOR 96] tandis que Vertex-AK est basés sur le schéma ACO "MAX-MIN Ant System" proposé dans [STÜ 00].

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 même valeurs (i.e., $\alpha = 1$ et $\beta = 5$). En revanche, les résultats présentés dans [LEG 99] ont été obtenus avec un taux d'évaporation ρ fixé à 0.3 (au lieu de 0.01 pour Vertex-AC), 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 $30 * 2000 = 60000$.

4.3.2. *Comparaison de Path-AK avec l'algorithme de Leguizamon et Michalewizc*

L'algorithme Path-AK utilise la même stratégie phéromonale que l'algorithme proposé par Fidanova dans [FID 02]. 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, dans l'algorithme de Fidanova l'information heuristique 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 dont le ratio de dureté est $rd = 0,25$, l'algorithme de Fidanova obtient un profit moyen de 23939. Si l'on compare ces résultats avec ceux de Path-AK (qui obtient 24192), 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 $30 * 2000 = 60000$.

4.4. *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 98] et une approche proposée par Gottlieb [GOT 00]. Ces deux approches combinent un algorithme génétique avec des techniques de réparation (pour transformer

une solution non consistante en un solution consistante) et des techniques de recherche locale (pour améliorer la qualité des solutions consistantes). 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.

Le tableau 3 compare Ant-knapsack avec ces deux approches. 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 optimales de la relaxation du problème linéaire sur les réels, i.e., $cart = 1 - Best/opt^{LP}$ où opt^{LP} est la valeur optimale du problème linéaire sur les réels et $Best$ est la meilleure solution trouvée par l’algorithme considéré.

Tableau 3. *Résultats 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 Ant-knapsack, Chu et Beasley(*`CB`*), et Gottlieb (*`G`*. Pour Ant-knapsack et Gottlieb, on reporte également le nombre moyen de solutions construites pour trouver la meilleure solution.*

Classe			`CB`	`G`		**Ant-knapsack**	
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	0.80	17147
		0.75	0.48	0.48	53528	0.48	15137
Moyenne pour 10.100			0.943	0.965	117848	0.96	16919

Si l’on considère la qualité des solutions trouvées, on remarque que Ant-knapsack obtient les même résultats que `CB` et `G` sur trois classes. Cependant, il obtient des résultats légèrement moins bons sur les 3 autres classes (à part pour 10.100 avec $rd = 0,5$ pour laquelle Ant-knapsack obtient de meilleurs résultats que `G` mais de moins bons résultats que `CB`). 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 Ant-Knapsack. De plus, quand on considère le nombre de solutions construites, on remarque que Ant-knapsack construit au plus soixante mille solutions tandis que les deux algorithmes génétiques en construisent au plus un million. Le tableau 3 montre que `G` construit toujours plus de solutions que Ant-Knapsack.

5. Conclusion

Nous avons proposé dans cet article un algorithme ACO générique pour résoudre le problème du sac-à-dos multidimensionnel. Cet algorithme est générique dans le sens où il est paramétré par les composants sur lesquels les traces de phéromone sont déposées. On a proposé trois instanciations différentes de cet algorithme générique : 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 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.

Comme nous l'avons souligné dans l'introduction de cet article, 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. Ainsi, pour ce genre de problème, il semble que la stratégie la plus intéressante soit de considérer l'ensemble des éléments déjà sélectionnés pour choisir le prochain élément à entrer dans le sous-ensemble, plutot que de considérer l'intéret de chaque élément "séparément".

Lorsque l'on compare les résultats obtenus par Edge-AK avec les algorithmes de l'état de l'art sur le sujet on constate que, s'ils sont globalement compétitifs, ils sont tout de même légèrement moins bons sur la moitié des classes considérées. Notons toutefois qu'Edge-AK n'utilise pas de technique de recherche locale pour améliorer les solutions construites par les fourmis. Ainsi, les travaux futurs concerneront essentiellement l'intégration de telles techniques dans Ant-Knapsack afin de le rendre plus compétitif avec les autres algorithmes pour ce problème.

6. Bibliographie

[ALA 04] ALAYA I., SOLNON C., GHÉDIRA K., « Ant algorithm for the multi-dimensional knapsack problem », *Proceedings of International Conference on Bioinspired Optimization Methods and their Applications (BIOMA 2004)*, 2004, p. 63–72.

[BUL 99] BULLNHEIMER B., HARTL R., STRAUSS C., « An Improved Ant system Algorithm for the Vehicle Routing Problem », *Annals of Operations Research*, vol. 89, 1999, p. 319–328.

[CHU 98] CHU P., BEASLEY J., « A genetic algorithm for the multidimentional knapsack problem », *Journal of heuristics*, vol. 4, 1998, p. 63–86.

[DAM 93] DAMMEYER F., S.VOSS, « Dynamic tabu list management using the reverse elimination method », *Annals of Operations Research*, vol. 41, 1993, p. 31–46.

[DOR 92] DORIGO M., « Optimization, Learning and Natural Algorithms *(in Italian)* », PhD thesis, Dipartimento di Elettronica, Politecnico di Milano, Italy, 1992.

[DOR 96] DORIGO M., MANIEZZO V., COLORNI A., « The Ant System : Optimization by a Colony of Cooperating Agents », *IEEE Transactions on Systems, Man, and Cybernetics – Part B*, vol. 26, n° 1, 1996, p. 29–41.

[DOR 97] DORIGO M., GAMBARDELLA L., « Ant Colony System : A Cooperative Learning Approach to the Traveling Salesman Problem », *IEEE Transactions on Evolutionary Computation*, vol. 1, n° 1, 1997, p. 53–66.

[DOR 99] DORIGO M., DI CARO G., « The Ant Colony Optimization Meta-Heuristic », CORNE D., DORIGO M., GLOVER F., Eds., *New Ideas in Optimization*, p. 11–32, McGraw Hill, UK, 1999.

[F.G 96] F.GLOVER, G.A.KOCHENBERGER, « Critical event tabu search for multidimensional knapsack problems », I.H.OSMAN, J.P.KELLY, Eds., *Metaheuristics : the Theory and Applications*, p. 407–427, Kluwer Academic Publishers, Dordrecht, The Netherlands, 1996.

[FID 02] FIDANOVA S., « Evolutionary Algorithm for Multidimensional Knapsack Problem », *Proceedings of PPSNVII*, 2002.

[GAM 99] GAMBARDELLA L., TAILLARD E., DORIGO M., « Ant Colonies for the Quadratic Assignment Problem », *Journal of the Operational Research Society*, vol. 50, 1999, p. 167–176.

[GOT 00] GOTTLIEB J., « Permutation-based evolutionary algorithms for multidimensional knapsack problems », *Proceedings of the 2000 ACM symposium on Applied computing*, 2000, p. 408–414.

[HAN 98] HANAFI S., FRÉVILLE A., « An approximate algorithm for multidimentional zero-one knapsack problems a parametric approach », *Management Science*, vol. 43, 1998, p. 402–410.

[LEG 99] LEGUIZAMON G., MICHALEWICZ Z., « A new version of Ant System for Subset Problem », *Proceedings of Congress on Evolutionary Computation*, 1999, p. 1459–1464.

[MAG 84] MAGAZINE M., OGUZ O., « A Heuristic Algorithm for the Multidimensional Zero-One Knapsack Problem », *European Journal of Operational Research*, vol. 16, 1984, p. 319–326.

[STÜ 00] STÜTZLE T., HOOS H., « $\mathcal{MAX} - \mathcal{MIN}$ Ant System », *Journal of Future Generation Computer Systems*, vol. 16, 2000, p. 889–914.

[VAS 01] VASQUEZ M., HAO J., « A Hybrid Approach for the 0-1 Multidimensional Knapsack Problem », *Proceedings of IJCAI'01*, 2001.