Intégration de l'optimisation par colonies de fourmis dans CP Optimizer

Madjid Khichane, Patrick Albert, Christine Solnon

▶ **To cite this version:**

Madjid Khichane, Patrick Albert, Christine Solnon. Intégration de l'optimisation par colonies de fourmis dans CP Optimizer. JFPC 2010 - Sixièmes Journées Francophones de Programmation par Contraintes, Jun 2010, Caen, France. pp.177-186. inria-00520276

HAL Id: inria-00520276
https://inria.hal.science/inria-00520276

Submitted on 22 Sep 2010

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

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

Intégration de l'optimisation par colonies de fourmis dans CP Optimizer

Madjid Khichane[1,2], Patrick Albert[1] et Christine Solnon[2]

[1] IBM
9, rue de Verdun, Gentilly 94253, France
[2] Université de Lyon
Université Lyon 1, LIRIS CNRS UMR5205, France
{madjid.khichane,palbert}@fr.ibm.com,christine.solnon@liris.cnrs.fr

Résumé

Nous présentons un algorithme générique pour la résolution de problèmes d'optimisation combinatoires. Cet algorithme est hybride et combine une approche heuristique, à savoir l'optimisation par colonies de fourmis (ACO), avec une approche complète de type "Branch &Propagate &Bound" (B&P&B), à savoir CP Optimizer (produit commercialisé par IBM Ilog). Le problème à résoudre est modélisé avec le langage de modélisation de CP Optimizer. Il est ensuite résolu par un algorithme générique qui fonctionne en deux phases séquentielles. La première phase sert à échantillonner l'espace des solutions. Pendant cette première phase, CP Optimizer est utilisé pour construire des solutions satisfaisant toutes les contraintes du problème, et le mécanisme d'apprentissage phéromonal d'ACO est utilisé pour identifier les zones prometteuses de l'espace de recherche par rapport à la fonction objectif à optimiser. Durant la deuxième phase, CP Optimizer effectue une recherche arborescente exhaustive (le "restart" [11]) guidée par les traces de phéromone accumulées lors de la première phase. Nous avons testé l'algorithme proposé sur des problèmes de sac à dos, d'affectation quadratique et d'ensemble stable maximum. Les premiers résultats sur ces trois problèmes montrent que ce nouvel algorithme améliore les performances de CP Optimizer.

1 Introduction

Les problèmes d'optimisation combinatoires (COP) sont d'une importance conséquente aussi bien dans le monde scientifique que dans le monde industriel. La plupart des COP sont *NP*-difficiles. Par conséquent, à moins que $P = NP$, ils ne peuvent pas être résolus de façon exacte en un temps polynomial. Parmi les COP qui sont *NP*-difficiles nous pouvons citer le problème du voyageur de commerce (TSP), le problème du sac-à-dos multidimensionnel (MKP) et le problème d'affectation quadratique (QAP). Pour résoudre ces COP, deux types d'approches complémentaires peuvent être utilisées, à savoir, les approches complètes et les approches heuristiques (ou métaheuristiques).

Les approches complètes explorent l'espace des combinaisons de façon exhaustive et procèdent généralement par *Branch & Bound* (B&B) : l'espace des combinaisons est structuré en un arbre qui est exploré de façon systématique, et des fonctions d'évaluation sont utilisées pour élaguer les sous-arbres dont l'évaluation de la fonction objectif est moins bonne que la meilleure solution trouvée. Cette approche permet de trouver la solution optimale en une limite de temps finie [9, 10]. Cependant, sa complexité est exponentielle dans le pire des cas. Quand le COP comporte des contraintes à satisfaire, en plus de la fonction objectif à optimiser, on peut raffiner l'approche B&B en ajoutant une phase de propagation des contraintes, ce que l'on peut résumer par *Branch & Propagate & Bound* (B&P&B) : à chaque nœud de l'arbre, les contraintes sont exploitées pour réduire l'espace de recherche en enlevant des domaines des variables les valeurs inconsistantes par rapport à une consistance locale. La programmation par contraintes (PPC) est généralement basée sur une approche B&P&B ; elle offre des langages de haut niveau pour la modélisation de COP en termes de contraintes et elle intègre toute une gamme d'algorithmes dédiés à la propagation de contraintes. Par conséquent, résoudre des COP avec la PPC ne nécessite pas beaucoup de travail de programmation. La PPC est géné-

ralement très efficace lorsque les contraintes sont suffisamment fortes pour que leur propagation réduise l'espace de recherche à une taille raisonnable. Toutefois, sur certaines instances, elle peut ne pas trouver de solutions de bonne qualité en un temps acceptable.

Les métaheuristiques contournent ce problème d'explosion combinatoire en explorant l'espace de recherche de façon incomplète : elles ignorent délibérément certaines zones qui leur semblent moins prometteuses. Un point clé réside dans leur capacité à équilibrer la diversification et l'intensification de la recherche : la diversification a pour objectif de garantir un bon échantillonnage de l'espace de recherche limitant le risque d'ignorer des zones contenant des solutions optimales ; l'intensification a pour objectif d'augmenter l'effort de recherche dans les zones prometteuses, aux alentours des bonnes solutions. Les métaheuristiques obtiennent généralement des solutions de très bonne qualité en des temps de calcul acceptables. Cependant, elles ne garantissent pas l'optimalité des solutions trouvées, car elles n'assurent pas le parcours entier de l'espace de recherche. Un autre inconvénient des métaheuristiques réside dans le fait que leur utilisation pour résoudre de nouveaux COP nécessite généralement un important travail de programmation. En particulier, la prise en compte des contraintes particulières à l'application demande de concevoir des structures de données appropriées pour évaluer rapidement les violations de contraintes avant de prendre une décision.

Beaucoup de métaheuristiques sont basées sur la recherche locale où l'espace de recherche est exploré par des perturbations élémentaires de combinaisons, e.g., le recuit simulé [5] ou la recherche Tabou [2]. Pour faciliter l'implémentation des algorithmes basés sur la recherche locale, Van Hentenryck et Michel ont conçu un langage de haut niveau basé sur les contraintes nommé Comet [3]. Comet introduit notamment la notion de variable incrémentale permettant au programmeur de concevoir de façon déclarative des structures de données qui permettent d'évaluer efficacement le voisinage de solutions.

Dans ce papier, nous proposons une approche générique pour la résolution de COP qui combine une approche complète de type B&P&B avec une métaheuristique. Nous avons implémenté et validé notre approche à l'aide de CP Optimizer, qui est un solveur générique, développé par IBM Ilog, qui utilise la technique de B&P&B. Notre objectif est de montrer qu'en combinant ce type d'approche avec une métaheuristique on améliore ses performances tout en conservant ses avantages principaux, à savoir la déclarativité et la complétude : dans notre nouvelle approche (de même qu'avec CP Optimizer), le COP à résoudre est défini de façon déclarative en termes de contraintes et il est résolu par un algorithme générique intégré au système ; cet algorithme est complet et garantit donc l'optimalité des solutions trouvées (dans la mesure où on ne borne pas les temps d'exécution).

La métaheuristique que nous utilisons pour améliorer les performances de CP Optimizer est l'optimisation par colonies de fourmis (ACO) [1]. Il s'agit d'une approche constructive qui explore l'espace de recherche par constructions itératives de nouvelles combinaisons. ACO a montré qu'elle est très efficace pour trouver rapidement de bonnes solutions, mais elle souffre des mêmes inconvénients que les autres métaheuristiques, c'est à dire, il n'y a aucune garantie quant à l'optimalité des solutions trouvées, et elle demande un important travail de programmation à chaque fois qu'elle est appliquée à un nouveau problème.

En combinant ACO avec CP Optimizer, nous prenons le meilleur des deux approches. Pour résoudre un problème dans cette nouvelle approche on procède de la même manière que lorsqu'on utilise CP Optimizer, c'est-à-dire on commence par modéliser le problème dans le langage de modélisation de CP Optimizer. Ensuite, on demande au solveur de rechercher la solution optimale. Cette recherche est décomposée en deux phases. Dans la première phase, ACO est utilisée pour échantillonner l'espace des solutions et identifier les régions de l'espace de recherche les plus prometteuses. Au cours de cette première phase, CP Optimizer est utilisé pour propager les contraintes et proposer des solutions réalisables à ACO. Dans la deuxième phase, CP Optimizer réalise une recherche basée sur le "restart" [11] et qui utilise l'approche B&P&B pour trouver la solution optimale. Au cours de cette deuxième phase, les traces de phéromones recueillies lors de la première phase sont utilisées par CP Optimizer comme heuristique d'ordre de valeurs. Cela lui permet de se concentrer en premier sur les régions les plus prometteuses de l'espace des solutions.

Rappelons que notre objectif principal n'est pas de rivaliser avec les algorithmes existants dans l'état de l'art qui sont dédiés à la résolution de problèmes spécifiques, mais de montrer qu'ACO peut améliorer le processus de recherche d'un algorithme générique utilisant la technique de B&P&B. Pour cela, nous avons choisi CP Optimizer comme référence, et nous l'avons utilisé comme une "boite noire" avec sa configuration par défaut correspondant à la stratégie Restart proposée par Refalo dans [11] : cette stratégie relance régulièrement de nouvelles recherches et utilise des *noGoods* pour apprendre de ses échecs passés ; elle utilise aussi les impacts comme heuristique de choix de variables et de valeurs.

La suite de cet article est organisée comme suit :

dans la section 2, nous rappelons quelques définitions sur les COP, la PPC et ACO. La section 3 décrit l'algorithme CPO-ACO. Dans la section 4, nous donnons quelques résultats expérimentaux sur les problèmes du sac-à-dos multidimensionnel, de l'affectation quadratique et de l'ensemble stable maximum. Nous concluons par une discussion sur certains travaux connexes et les perspectives de ce travail.

2 Contexte

2.1 Problème d'optimisation combinatoire (COP)

Un COP est défini par un quadruplet $P = (X, D, C, F)$ tel que

- $X = \{x_1, \ldots, x_n\}$ est un ensemble de n variables de décisions ;
- à chaque variable $x_i \in X$ est associé un domaine $D(x_i)$ qui est un ensemble fini d'entiers ;
- C est un ensemble de contraintes à satisfaire ;
- $F : D(x_1) \times \ldots \times D(x_n) \longrightarrow \mathbb{R}$ est la fonction objectif à optimiser.

Une affectation $\mathcal{A}$ est un ensemble de couples variable-valeur notés $< x_i, v_i >$ et correspondant à l'affectation de la valeur $v_i \in D(x_i)$ à la variable x_i. Une affectation $\mathcal{A}$ est complète si toutes les variables de X sont affectées dans $\mathcal{A}$, sinon, elle est partielle. Une affectation est inconsistante si elle viole une ou plusieurs contraintes de C, sinon elle est consistante. Une solution est une affectation consistante et complète. Une solution $\mathcal{A}$ de P est optimale si pour toute autre solution $\mathcal{A}^{'}$ de P, $F(\mathcal{A}) \leq F(\mathcal{A}^{'})$ si P est un problème de minimisation ou $F(\mathcal{A}) \geq F(\mathcal{A}^{'})$ si P est un problème de maximisation.

2.2 Résolution d'un COP par B&P&B

L'approche B&P&B résout un COP en construisant un arbre de recherche : à chaque noeud, elle choisit une variable non encore affectée x_i et elle choisit une valeur $v_j \in D(x_i)$ pour créer le point de choix $x_i = v_j \vee x_i \neq v_j$. Elle explore ensuite séparément chaque sous-arbre induit par chacune de ces deux alternatives. Cette recherche arborescente est combinée avec la propagation de contraintes et les techniques d'approximation de la fonction objectif.

La propagation de contraintes assure un certain niveau de consistance en filtrant les domaines des variables non affectées. Ce filtrage se traduit par la suppression des valeurs qui sont inconsistantes par rapport à une certaine consistance locale comme, par exemple, la consistance d'arc [7].

Les techniques d'encadrement des bornes de la fonction objectif permettent également de filtrer les domaines des variables [6]. En effet, une valeur d'une variable qui ne pourra pas figurer dans une solution qui soit meilleure que la meilleure solution trouvée jusqu'à présent sera supprimée.

Lorsque la propagation de contraintes ou l'approximation des bornes de la fonction objectif détecte un échec (un domaine d'une variable est devenu vide), cette procédure fait marche arrière pour exploiter un autre noeud de l'arbre de recherche. Cette méthode est efficace et générique. En revanche, elle ne parvient pas à résoudre certains COP pour lesquels la propagation de contraintes et/ou les techniques d'approximation ne sont pas capables de réduire suffisamment l'espace de recherche.

2.3 Heuristique d'ordre basée sur les impacts

A chaque étape d'une résolution B&P&B, il s'agit de choisir la prochaine variable à affecter ainsi qu'une valeur appartenant à son domaine. Ces choix ont une forte influence sur la taille de l'arbre de recherche et on utilise généralement des heuristiques d'ordre pour guider ces choix.

Refalo [11] a proposé des heuristiques d'ordre basées sur la notion d'impact. L'impact de l'affectation d'une valeur à une variable est défini comme étant la proportion de l'espace de recherche supprimée par la propagation de contraintes causée par cette affectation. L'impact d'une valeur est défini comme étant la moyenne de ses impacts observés et l'impact d'une variable, comme étant la moyenne de l'impact des valeurs restantes dans son domaine. Ces impacts sont utilisés pour définir des heuristiques d'ordre : à chaque nœud de l'arbre de recherche, la prochaine variable à affecter est celle ayant le plus fort impact, et la valeur affectée est celle ayant le plus petit impact.

Ces heuristiques d'ordre basées sur les impacts sont celles utilisées par défaut par CP Optimizer.

2.4 Optimisation par Colonies de Fourmis (ACO)

ACO est une métaheuristique constructive qui explore l'espace de recherche en construisant itérativement de nouvelles combinaisons. Cette métaheuristique a été appliquée à la résolution de nombreux COP, et il existe de nombreuses déclinaisons de ce principe très général. Nous invitons le lecteur intéressé à se référer à [1] pour plus de détails. Nous décrivons ici l'algorithme du MAX-MIN Ant System (MMAS) proposé par Stützle et Hoos dans [13], qui a montré de très bonnes performances sur de nombreux problèmes, et nous le présentons dans notre contexte de résolution d'un COP modélisé par un quadruplet (X, D, C, F).

L'idée de base est de construire des affectations complètes selon un principe glouton aléatoire, et de progressivement biaiser le modèle probabiliste utilisé pour

construire une affectation en fonction de l'expérience passée. On utilise pour cela un mécanisme simple d'apprentisage par renforcement de traces de phéromone.

Structure phéromonale. La phéromone est utilisée dans un algorithme ACO pour identifier progressivement les composants de solution les plus prometteurs, et biaiser en conséquence le modèle probabiliste utilisé pour construire des combinaisons. De façon générale, on associe à chaque composant de solution c une trace de phéromone τ_c. Cette trace de phéromone représente l'expérience passée de la colonie concernant l'utilisation de c lors de la construction d'une combinaison : plus c a participé à de bonnes combinaisons, plus τ_c a une valeur importante, et plus la probabilité de choisir c lors de la construction d'une combinaison est importante.

Le choix de la structure phéromonale (i.e., des composants sur lesquels on dépose de la phéromone) dépend du problème à résoudre. Si on considère un COP définit par un quadruplet (X, D, C, F), on peut associer une trace de phéromone $\tau(x_i, v_j)$ à chaque couple variable-valeur $< x_i \in X, v_j \in D(x_i) >$. Intuitivement, la quantité de phéromone sur $< x_i, v_j >$ représente l'intéret appris d'affecter la valeur v_j à la variable x_i. Cette structure phéromonale a démontré son efficacité à résoudre certains COP, par exemple, le QAP [13], les CSP [12] ou le problème d'ordonnancement de voitures [4].

Les traces de phéromone sont utilisées pour intensifier la recherche autour des meilleures affectations trouvées. Afin d'équilibrer l'intensification et la diversification, ces traces de phéromone sont bornées entre deux paramètres τ_{min} et τ_{max} de sorte que les différences entre deux traces soient limitées. De plus, les traces de phéromone sont initialisées à τ_{max} au début de l'exécution d'un algorithme ACO.

Construction des affectations. A chaque cycle de l'algorithme, chaque fourmi construit une affectation complète selon un principe glouton aléatoire : partant d'une affectation vide, on choisit à chaque itération une variable non affectée et une valeur à affecter à cette variable. Ce processus est répété jusqu'à ce que toutes les variables soient affectées. Le choix de la prochaine variable à affecter se fait par rapport à une heuristique donnée. Par exemple, pour le problème d'ordonnancement de voitures, les variables sont affectées par ordre croissant d'indice dans la séquence de voitures. Pour un CSP, on peut choisir la variable ayant le plus petit domaine.

Une fois qu'une fourmi a sélectionné une variable x_i, elle choisit une valeur $v_j \in D(x_i)$ en fonction de la probabilité

$$p(x_i, v_j) = \frac{[\tau(x_i, v_j)]^\alpha \cdot [\eta(x_i, v_j)]^\beta}{\sum_{v_k \in D(x_i)} [\tau(x_i, v_k)]^\alpha \cdot [\eta(x_i, v_k)]^\beta} \quad (1)$$

où $\eta(x_i, v_j)$ est le facteur heuristique associé à l'affectation de v_j à x_i. La définition de ce facteur heuristique dépend du problème considéré. Généralement, ce facteur heuristique évalue l'impact de l'affectation de v_j à x_i sur la fonction objectif. α et β sont deux paramètres qui déterminent l'influence de la phéromone et de l'heuristique dans le choix de la valeur.

Les contraintes sont généralement gérées différemment selon que le COP est faiblement ou fortement contraint. Quand le COP est faiblement contraint, il est très facile de construire une solution satisfaisant toutes les contraintes, et la difficulté réside dans le fait que l'on cherche la solution optimisant la fonction objectif. C'est le cas par exemple du QAP ou du MKP. Dans ce cas, les contraintes sont propagées après chaque affectation de sorte que les domaines ne contiennent que des valeurs consistantes, et les fourmis ne construisent que des affectations consistantes. Par exemple, pour le QAP, la contrainte imposant de ne pas implanter une même usine à plusieurs endroits est propagée en enlevant les emplacements déjà utilisés des domaines ; pour le MKP, les contraintes de capacité sont propagées en affectant à 0 les variables associées à des objets ne pouvant être sélectionnés sans violer une contrainte de capacité.

Quand le COP est fortement contraint, la construction d'une solution satisfaisant toutes les contraintes (indépendamment de la fonction objectif) peut devenir un problème difficile de sorte que les fourmis peuvent ne pas y arriver. Pour éviter cela, les contraintes peuvent être intégrées dans la fonction objectif : l'objectif est alors de trouver l'affectation satisfaisant au mieux les contraintes [12].

Mise-à-jour de la phéromone. Une fois que chaque fourmi a construit une solution, les traces de phéromone sont mises-à-jour. Dans un premier temps, toutes les traces de phéromone sont diminuées en les multipliant par $(1-\rho)$, où $\rho \in [0; 1]$ est le taux d'évaporation. Ce processus d'évaporation permet aux fourmis d'oublier progressivement les constructions anciennes pour ainsi mettre l'accent sur les constructions les plus récentes.

Dans un deuxième temps, on récompense les meilleurs affectations construites lors du dernier cycle et/ou la meilleure affectation construite depuis le début de l'exécution. Cette récompense est traduite par le dépôt d'une quantité de phéromone sur les composants de ces affectations. Lorsque les traces

de phéromone sont associées à des couples variable-valeur, la phéromone est déposée sur les couples <variable/valeur> de l'affectation à récompenser. La quantité de phéromone déposée est généralement proportionnelle à la qualité de l'affectation à récompenser. Cette quantité est souvent normalisée entre 0 et 1 et elle est généralement définie comme un ratio entre la qualité de l'affectation à récompenser et la valeur optimale (si elle est connue) ou la qualité de la meilleure affectation trouvée jusqu'alors.

3 Description de $CPO - ACO$

ACO s'est révélée très efficace pour trouver rapidement de bonnes solutions à de nombreux COP. Cependant, la conception d'un algorithme ACO pour résoudre un nouveau COP peut exiger un grand effort de programmation. En effet, si les procédures de gestion et d'exploitation de la phéromone sont très semblables d'un COP à l'autre, la résolution d'un nouveau COP nécessite d'écrire des procédures de propagation et de contrôle des contraintes dépendantes du problème considéré. Ce constat est à l'origine de notre travail : en combinant ACO avec la PPC, on peut réutiliser les nombreuses procédures disponibles pour la gestion des contraintes.

L'algorithme proposé $CPO - ACO$ fonctionne en deux phases :

- Pendant la première phase, ACO utilise CP Optimizer pour construire des solutions et le mécanisme d'apprentissage phéromonal est utilisé pour intensifier progressivement la recherche autour des meilleures solutions trouvées.
- Pendant la deuxième phase, CP Optimizer est utilisé pour rechercher la solution optimale, et les traces de phéromone recueillies au cours de la première phase sont utilisées pour guider CP Optimizer dans sa recherche.

3.1 Première phase de $CPO - ACO$

L'algorithme 1 décrit la première phase de $CPO - ACO$, les grandes lignes de cet algorithme sont décrites ci-après.

Construction d'une affectation

Lors de chaque cycle (lignes 3-14), chaque fourmi demande à CP Optimizer de construire une solution (ligne 5). Notez que pendant cette première phase, nous ne demandons pas à CP Optimizer d'optimiser la fonction objectif, mais simplement de trouver des solutions satisfaisant toutes les contraintes. Chaque nouvel appel à CP Optimizer correspond à une nouvelle recherche (restart) et CP Optimizer construit une solution selon le principe B&P&B : il propage les contraintes après chaque affectation et si un échec est détecté, il fait marche arrière pour prendre une autre décision jusqu'à ce qu'il trouve une solution. Lors de cette première phase, CP Optimizer est utilisé avec ses paramètres par défaut à l'exception de l'heuristique de choix de valeur qui lui est transmise en paramètre. Cette heuristique est basée sur ACO et définit la probabilité de choisir la valeur v_j pour une variable x_i par

$$p(v_j) = \frac{[\tau(x_i, v_j)]^\alpha \cdot [1/impact(v_j)]^\beta}{\sum_{v_k \in D(x_i)} [\tau(x_i, v_k)]^\alpha \cdot [1/impact(v_k)]^\beta}$$

où $impact(v_j)$ est l'impact observé de la valeur v_j [11]. Comme toutes les traces de phéromone sont initialisées à la même valeur (i.e., τ_{max}), au cours des premiers cycles, les impacts sont plus déterminants que les traces de phéromone dans le choix des valeurs. Toutefois, à la fin de chaque cycle les traces de phéromone sont mises à jour, et les traces de phéromones influencent de plus en plus le choix des valeurs.

Notons que CPO-ACO est plutôt destiné à résoudre des COP pour lesquels la difficulté n'est pas de construire des solutions, mais de trouver la solution qui optimise la fonction objectif. Pour ces problèmes, CP Optimizer est capable de construire très rapidement une solution (en faisant très peu de retours arrières) de sorte que l'on peut rapidement collecter un nombre significatif de solutions qui peuvent alors être utilisées par ACO pour biaiser la recherche. CPO-ACO pourrait être utilisé pour résoudre des COP plus contraints mais, dans ce cas, CP Optimizer peut avoir besoin de plus de temps pour calculer une solution satisfaisant toutes les contraintes ce qui fait que l'apprentissage phéromonal sera basé sur trop peu de solutions pour être intéressant.

Mise-à-jour de la phéromone

Une fois que chaque fourmi a construit une affectation, les traces de phéromone sont évaporées en les multipliant par $(1 - \rho)$ où $\rho \in [0; 1]$ est le taux d'évaporation des phéromones (lignes 6-7).

À la fin de chaque cycle, les meilleures solutions (par rapport à la fonction l'objectif) sont récompensées dans le but d'intensifier la recherche autour d'elles. Les meilleures solutions du cycle sont systématiquement récompensées (lignes 9-11). En revanche, la meilleure solution construite depuis le début de la recherche est récompensée seulement si elle est meilleure que les meilleures solutions du dernier cycle (lignes 12-13).

Une solution $\mathcal{A}$ est récompensée en augmentant la quantité de phéromone sur chaque couple $< x_i, v_j >$ de $\mathcal{A}$, ainsi, la probabilité d'affecter x_i à v_j lors des futures

Entrées : $P = (X, D, C, F)$ et les paramètres $\{t_{max1}, d_{min}, it_{max}, \alpha, \beta, \rho, \tau_{min}, \tau_{max}, nbAnts\}$
Sorties : une solution $\mathcal{A}_{best}$ et une matrice de phéromone $\tau : X \times D \rightarrow [\tau_{min}; \tau_{max}]$

début
  pour chaque $x_i \in X$ **et chaque** $v_j \in D(x_i)$ **faire** $\tau(x_i, v_j) \leftarrow \tau_{max}$
  répéter
    `/* Construction des solutions */`
    pour chaque $k \in \{1, \ldots, nbAnts\}$ **faire**
      Construire une solution $\mathcal{A}_k$ en utilisant CP Optimizer
    `/* Evaporation de la phéromone */`
    pour chaque $x_i \in X$ **et chaque** $v_i \in D(x_i)$ **faire**
      $\tau(x_i, v_i) \leftarrow max(\tau_{min}, (1 - \rho) \cdot \tau(x_i, v_i))$
    `/* Dépôt de phéromone */`
    Soit $\mathcal{A}_{best}$ la meilleure solution construite jusqu'à présent (y compris le dernier cycle)
    pour chaque $k \in \{1, \ldots, nbAnts\}$ **faire**
      si $\forall l \in \{1, \ldots, nbAnts\}, \mathcal{A}_k$ *est au moins aussi bon que* $\mathcal{A}_l$ **alors**
        pour chaque $< x_i, v_i > \in \mathcal{A}_k$ **faire** $\tau(x_i, v_i) \leftarrow min(\tau_{max}, \tau(x_i, v_i) + \frac{1}{1+|F(\mathcal{A}_k) - F(\mathcal{A}_{best})|})$
    si $\mathcal{A}_{best}$ *est strictement meilleur que toutes les solutions* $\{\mathcal{A}_1, ..., \mathcal{A}_{nbAnts}\}$ **alors**
      pour chaque $< x_i, v_i > \in \mathcal{A}_{best}$ **faire** $\tau(x_i, v_i) \leftarrow min(\tau_{max}, \tau(x_i, v_i) + 1)$
  jusqu'à *temps utilisé* $\geq t_{max1}$ **ou** *nombre de cycle sans amélioration de* $\mathcal{A}_{best} \geq it_{max}$ **ou** *distance moyenne de* $\{\mathcal{A}_1, ..., \mathcal{A}_{nbAnts}\} \leq d_{min}$;
  retourne $\mathcal{A}_{best}$ **et** τ
fin

Algorithme 1 – Phase 1 de $CPO - ACO$

affectations est augmentée. La quantité de phéromone ajoutée est inversement proportionnel à l'écart entre $F(\mathcal{A})$ et $F(\mathcal{A}_{best})$.

Conditions d'arrêt

La première phase s'arrête dans l'un des cas suivants : si le temps CPU t_{max1} a été atteint ; si la meilleure solution $\mathcal{A}_{best}$ n'a pas été améliorée depuis it_{max} itérations ; ou bien si la distance moyenne entre les affectations calculées pendant le dernier cycle est plus petite que d_{min}, ce qui indique que les traces de phéromone ont permis à la recherche de converger. Nous définissons la distance entre deux affectations comme étant le taux de couples variable-valeur sur lesquels les deux affectations sont différentes : la distance entre $\mathcal{A}_1$ et $\mathcal{A}_2$ est $\frac{|X| - |\mathcal{A}_1 \cap \mathcal{A}_2|}{|X|}$.

3.2 Deuxième phase de $CPO - ACO$

À la fin de la première phase, la meilleure solution construite $\mathcal{A}_{best}$ et la structure phéromonale τ sont transmises à la deuxième phase. Le coût de $\mathcal{A}_{best}$ est utilisé pour borner la fonction objectif. Ensuite, CP Optimizer est lancé pour chercher la solution optimale : dans cette deuxième phase, chaque fois que CP Optimizer trouve une meilleure solution, il borne la fonction objectif avec son coût et il fait marche arrière pour trouver de meilleures solutions ou prouver l'optimalité de la dernière solution trouvée.

Comme dans la première phase, CP Optimizer est utilisé comme une boîte noire avec ses paramètres de recherche par défaut et utilise la structure phéromonale τ et les impacts comme heuristiques d'ordre de variables. Cependant, cette fois, il ne calcule pas les probabilités de sélection des valeurs pour une variable x_i, mais il choisit la valeur qui maximise la formule

$$[\tau(x_i, v_i)]^\alpha \cdot [1/impact(v_i)]^\beta.$$

Nous avons comparé expérimentalement différentes variantes de CPO-ACO :

- Dans le but de montrer l'intérêt d'utiliser un mécanisme d'apprentissage phéromonal lors de la première phase, nous avons testé la variante où, pendant la première phase, la recherche est effectuée sans utiliser de phéromone de sorte que l'heuristique de choix de valeurs est définie par les impacts uniquement. Cette variante donne des résultats significativement moins bons.
- Pour évaluer l'intérêt de l'utilisation des traces de phéromone lors de la deuxième phase, nous

avons testé la variante où, à la fin de la première phase, nous ne retenons que $\mathcal{A}_{best}$ qui est utilisé pour borner la fonction objectif au début de la deuxième phase. La deuxième phase est alors constituée par la recherche par défaut de CP Optimizer (qui utilise uniquement les impacts comme heuristique de choix de valeurs). Cette variante obtient également des résultats beaucoup moins bons que lorsque la structure de phéromone est utilisée lors de cette deuxième phase.
- Enfin, nous avons testé la variante où, au cours de la deuxième phase, le choix de valeur pour une variable donnée est fait en utilisant la règle de transition probabiliste de ACO (voir la section 3.1) comme dans la première phase au lieu de sélectionner la valeur qui maximise la formule donnée en section 3.2. Cette variante obtient, sur la plupart des tests effectués, des résultats qui ne sont pas significativement différents de ceux qu'on donne dans la section suivante.

4 Evaluation expérimentale de CPO-ACO

4.1 Les problèmes considérés

Nous avons évalué l'algorithme CPO-ACO sur trois COP bien connus, à savoir : le problème de sac-à-dos multidimensionnel (MKP) ; le problème d'affectation quadratique (QAP) et le problème de l'ensemble stable maximum (MIS).

Le problème de sac à dos multidimensionnel (MKP) consiste à sélectionner un sous-ensemble d'objets qui satisfait un ensemble de contraintes linéaires de capacité et qui maximise la somme des profits des objets sélectionnés. Le modèle PPC que nous avons utilisé est le suivant :

- X associe une variable x_i à chaque objet i ;
- $\forall x_i \in X$, $D(x_i) = \{0, 1\}$ de sorte que $x_i = 0$ si i n'est pas sélectionné, et 1 sinon ;
- C est un ensemble de m contraintes de capacité tel que chaque contrainte $C_j \in C$ est de la forme $\sum_{i=1}^{n} c_{ij} \cdot x_i \leq r_j$ où c_{ij} est la quantité de ressource j requise par l'objet i et r_j est la quantité disponible de la ressource j ;
- la fonction objectif à maximiser est $F = \sum_{i=1}^{n} u_i \cdot x_i$ où u_i est le profit de l'objet i.

Nous avons considéré les instances académic avec 100 objets disponibles sur http ://people.brunel.ac.uk/˜mastjjb/jeb/orlib/mknapinfo.html. Nous avons considéré les 20 premières instances avec 5 contraintes de ressources (de 5-100-00 à 5-100-19), les 20 premières instances avec 10 contraintes de ressources (de 10-100-00 à 10-100-19) et les 20 premières instances avec 30 contraintes de ressources (de 30-100-00 à 30-100-19).

Le problème d'affectation quadratique (QAP) consiste à déterminer l'emplacement d'usines de façon à minimiser les distances entre les usines et les flux de produits entre les usines. Le modèle PPC que nous avons utilisé est le suivant :

- X associe une variable x_i à chaque usine i ;
- $\forall x_i \in X$, $D(x_i) = \{1, \ldots, n\}$ tel que $x_i = j$ si l'usine i est construite à l'emplacement j ;
- C contient uniquement la contrainte all-different sur toutes les variables, assurant ainsi que chaque usine est construite à un seul emplacement.
- la fonction objectif à optimiser est $F = \sum_{i=1}^{n} \sum_{j=1}^{n} a_{x_i x_j} b_{ij}$ où $a_{x_i x_j}$ est la distance entre les emplacements des usines i et j, et b_{ij} est le flux entre les usines i et j.

Pour nos tests, nous avons utilisé les instances académiques de la QAPLIB qui sont disponibles sur www.opt.math.tu-graz.ac.at/qaplib/inst.html.

Le problème de l'ensemble stable maximum (MIS) consiste à sélectionner le plus grand sous-ensemble de sommets d'un graphe de sorte que deux sommets sélectionnés dans ce sous-ensemble ne sont pas reliés par une arête (ce problème est équivalent à la recherche d'une clique maximum dans le graphe inverse). Le modèle PPC que nous avons utilisé est le suivant :

- X associe une variable x_i à chaque sommet i ;
- $\forall x_i \in X$, $D(x_i) = \{0, 1\}$ de sorte que $x_i = 0$ si le sommet i n'est pas sélectionné et $x_i = 1$ sinon ;
- C associe une contrainte binaire c_{ij} à chaque arête (i, j) du graphe. Cette contrainte assure que les sommets i et j ne seront pas sélectionnés dans le même ensemble, i.e., $c_{ij} = (x_i + x_j < 2)$.
- la fonction objectif à maximiser est $F = \sum_{i=1}^{n} x_i$.

Les instances de MIS que nous avons utilisées pour nos tests sont toutes disponibles sur http ://www.nlsde.buaa.edu.cn/∼kexu/benchmarks/graph-benchmarks.htm.

4.2 Conditions expérimentales

Pour chaque problème, le modèle PPC a été implémenté en utilisant le langage de modélisation de CP Optimizer.

Nous comparons CPO-ACO avec CP Optimizer (noté dans la suite CPO). Dans les deux cas, nous avons utilisé la version V2.3 de CPO avec ses paramètres de recherche par défaut. Toutefois, pour CPO-ACO, les fonctions de choix de valeurs sont transmises en paramètres à CPO comme décrit dans la section

précédente. Pour CPO, les impacts sont utilisés comme heuristique de choix de variable et de valeur [11].

Pour toutes les expériences, le temps CPU total a été limité à 300 secondes sur une machine Pentium-4 2.2 Gz. Pour CPO-ACO, cette durée totale est partagée entre les deux phases : la durée de la phase 1 est au plus égale à $t_{max1} = 25\%$ du temps total. Nous avons par ailleurs fixé d_{min} à 0.05 (de sorte que la phase 1 est arrêtée dès que la distance moyenne entre les solutions du même cycle est inférieure à 5%) et it_{max} à 500 (de sorte que la phase 1 est arrêtée si $\mathcal{A}_{best}$ n'a pas été améliorée depuis 500 cycles). Le nombre de fourmis est $nbAnts = 20$; le poids du facteur phéromonal est $\alpha = 1$ et le poids du facteur heuristique est $\beta = 2$. Les traces de phéromone sont bornées entre $\tau_{min} = 0.01$ et $\tau_{max} = 1$.

Notons que nous n'avons pas utilisé le même taux d'évaporation de la phéromone pour toutes les expérimentations. En effet, pour le MKP et le MIS les variables sont binaires, ce qui fait qu'à chaque cycle une des deux valeurs possibles (0 ou 1) est récompensée, et ACO converge assez rapidement. Pour le QAP, tous les domaines sont de taille n (où n est égal au nombre de variables), donc à chaque cycle, seulement une valeur sur les n valeurs possibles est récompensée. Dans ce cas, nous avons délibérément augmenté le taux d'évaporation afin d'accélérer la convergence de ACO lors de la première phase. Par conséquent, $\rho = 0.01$ pour le MKP et le MIS et $\rho = 0.1$ pour le QAP.

Pour les deux algorithmes, CPO et CPO-ACO, nous avons effectué 30 exécutions par instance de chaque problème.

4.3 Résultats expérimentaux

Le tableau 1 donne les résultats expérimentaux obtenus par CPO et CPO-ACO sur le MKP, le QAP et le MIS. Pour chaque classe d'instances et pour chaque algorithme, ce tableau donne l'écart (en pourcentage) par rapport à la meilleure solution connue. Notons d'abord que CPO et CPO-ACO n'atteignent (presque) jamais la solution optimale connue : les meilleures solutions connues sont en effet, obtenues avec des approches dédiées. CPO et CPO-ACO sont des approches complètement génériques qui ne visent pas à concurrencer les approches dédiées. Aussi, nous avons choisi une limite raisonnable de temps CPU (300 secondes) afin de nous permettre d'effectuer un nombre suffisant de tests, pour pouvoir utiliser les tests statistiques. Avec cette limite de temps, CPO-ACO obtient des résultats compétitifs avec des approches dédiées sur le MKP (moins de 1% d'écart par rapport aux meilleures solutions connues), mais il est assez loin des meilleures solutions connues sur de nombreuses instances du QAP et MIS.

Comparons maintenant CPO avec CPO-ACO. Le tableau 1 nous montre que l'utilisation d'ACO pour guider CPO améliore le processus de recherche sur toutes les classes d'instances sauf deux. Toutefois, cette amélioration est plus importante pour le MKP que pour les deux autres problèmes. Comme les deux approches ont obtenu des résultats assez proches sur certaines classes d'instances, nous avons fait des tests statistiques (t-test avec un niveau de confiance de 95%) pour déterminer si les résultats sont significativement différents ou non. Pour chaque classe, nous avons indiqué le pourcentage d'instances pour lesquelles une approche a obtenu des résultats significativement meilleurs que l'autre (colonne $>_{t-test}$ du tableau 1). Pour le MKP, CPO-ACO est nettement meilleur que CPO sur 57 instances, alors qu'il n'est pas significativement différent pour 3 instances. Pour le QAP, CPO-ACO est nettement meilleur que CPO sur un grand nombre d'instances. Toutefois, CPO est meilleur que CPO-ACO sur une instance de la classe tai* du QAP. Pour le MIS, CPO-ACO est nettement meilleur que CPO sur 35% d'instances, mais il n'est pas significativement différent sur toutes les autres.

5 Conclusion

Nous avons proposé une approche générique pour résoudre les COP définis par un ensemble de contraintes et une fonction objectif. Cette approche générique combine une approche B&P&B avec ACO. L'idée principale de cette combinaison est de bénéficier de l'efficacité (i) de ACO pour explorer l'espace de recherche et identifier les zones prometteuses (ii) de CP Optimizer pour exploiter fortement le voisinage des meilleures solutions trouvées par ACO. Cette combinaison nous permet d'atteindre un bon équilibre entre la diversification et l'intensification de la recherche : la diversification est principalement assurée au cours de la première phase par ACO et l'intensification est assurée par CP Optimizer au cours de la deuxième phase. Nous avons montré par des expériences sur trois COP différents que cette approche hybride améliore les performances de CP Optimizer.

Il est à noter que grâce à la nature modulaire de CP Optimizer qui sépare clairement la partie modélisation du problème de la partie résolution, la combinaison de ACO et CP Optimizer a été faite de manière naturelle. Par conséquent, le programme CPO-ACO utilisé était exactement le même pour les expériences sur les différents problèmes utilisés dans cet article.

D'autres travaux ont également proposé des approches intégrant ACO avec des approches de type B&P&B. En particulier, B. Meyer a proposé, dans [8], deux algorithmes hybrides où ACO a été couplée avec

Résultats pour le MKP

			CPO				$CPO-ACO$			
Name	# I	# X	avg	(sd)	$>_{avg}$	$>_{t-test}$	avg	(sd)	$>_{avg}$	$>_{t-test}$
5.100-*	20	100	1.20	(0.30)	0%	0%	**0.46**	(0.23)	**100%**	**100%**
10.100-*	20	100	1.53	(0.31)	0%	0%	**0.83**	(0.34)	**100%**	**100%**
30.100-*	20	100	1.24	(0.06)	5%	0%	**0.86**	(0.08)	**95%**	**85%**

Résultats pour le QAP

			CPO				$CPO-ACO$			
Name	# I	# X	avg	(sd)	$>_{avg}$	$>_{t-test}$	avg	(sd)	$>_{avg}$	$>_{t-test}$
bur*	7	26	1.17	(0.43)	0%	0%	**0.88**	(0.43)	**100%**	**57 %**
chr*	11	19	12.11	(6.81)	45%	9%	**10.99**	(6.01)	**55 %**	**45 %**
had*	5	16	1.07	(0.89)	0%	0%	**0.54**	(1.14)	**100%**	**60 %**
kra*	2	30	17.46	(3.00)	0%	0%	**14.99**	(2.79)	**100%**	**100%**
lipa*	6	37	22.11	(0.82)	0%	0%	**20.87**	(0.75)	**100%**	**100%**
nug*	15	20	8.03	(1.59)	7%	0%	**5.95**	(1.44)	**93 %**	**80 %**
rou*	3	16	5.33	(1.15)	33%	0%	**3.98**	(1.00)	**67 %**	**67 %**
scr*	3	16	**4.60**	(2.4)	33%	0%	5.12	(2.60)	**67 %**	0 %
tai*	4	16	6.06	(1.35)	25%	25%	**4.84**	(1.25)	**75 %**	**50 %**

Résultats pour le MIS

			CPO				$CPO-ACO$			
Name	# I	# X	avg	(sd)	$>_{avg}$	$>_{t-test}$	avg	(sd)	$>_{avg}$	$>_{t-test}$
frb-30-15-*	5	450	9.83	(1.86)	0%	0%	**9.46**	(2.00)	**80%**	**20%**
frb-35-17-*	5	595	**11.62**	(2.05)	**60%**	0%	11.82	(2.31)	40%	0%
frb-40-19-*	5	760	13.47	(1.92)	20%	0%	**12.85**	(2.22)	**80%**	**20%**
frb-45-21-*	5	945	15.40	(2.43)	0%	0%	**14.35**	(1.82)	**100%**	**80%**
frb-50-23-*	5	1150	16.24	(2.32)	20%	0%	**15.84**	(2.00)	**80%**	**20%**
frb-53-24-*	5	1272	18.15	(2.55)	0%	0%	**16.86**	(1.84)	**100%**	**80%**
frb-56-25-*	5	1400	17.85	(2.37)	20%	0%	**16.89**	(1.08)	**80%**	**40%**
frb-59-26-*	5	1534	18.40	(2.44)	40%	0%	**18.37**	(2.16)	**60%**	**20%**

TABLE 1 – Comparaison de CPO et CPO-ACO sur le MKP, le QAP et le MIS. Chaque ligne donne successivement : le nom de la classe d'instances, le nombre d'instances dans la classe ($\#I$), le nombre moyen de variables dans ces instances ($\#X$), les résultats obtenus par CPO (resp. CPO-ACO), à savoir, le pourcentage d'écart par rapport à la meilleure solution connue (moyenne (avg) et écart type(sd)), le pourcentage d'instances pour lesquelles CPO (resp. CPO-ACO) a obtenu de meilleurs résultats en moyenne ($>_{avg}$), et le pourcentage d'instances pour lesquelles CPO (resp. CPO-ACO) est donné meilleur par le test statistique t-test.

la PPC. Il a proposé un premier couplage faible où les deux approches fonctionnent en parallèle en échangeant seulement les solutions et les bornes de la fonction objectif. Il a proposé un deuxième couplage plus fort, où la propagation de contraintes a été incorporée dans ACO pour permettre à une fourmi de revenir en arrière (backtrack) lorsqu'une affectation d'une valeur à une variable donnée échoue. Toutefois, la procédure de retour en arrière a été limitée au niveau de la dernière variable choisie. Cela signifie que, si toutes les valeurs possibles de la dernière variable choisie ont été essayées sans succès, la recherche d'une fourmi se termine par un échec (pas de solution construite). Les résultats de ce travail montrent sur le problème d'ordonnancement de tâches que le couplage plus fort est meilleur. Notons que le couplage fort proposé n'est pas une approche complète.

Nous avons également proposé dans [4] une approche hybride, notée Ant-CP, qui combine ACO avec la PPC pour résoudre les problèmes de satisfaction de contraintes (sans fonction à optimiser). Comme CPO-ACO, Ant-CP utilise le langage de modélisation de la PPC pour définir le problème, et les fourmis utilisent les procédures prédéfinies de la PPC pour propager les contraintes. Cependant, contrairement à CPO-ACO, Ant-CP effectue une recherche incomplète.

Plusieurs points méritent d'être mentionnés en tant que futures améliorations de CPO-ACO. À l'heure actuelle, CPO-ACO fonctionne plutôt bien sur les problèmes pour lesquels la recherche d'une solution réalisable est facile. Dans ce papier, nous avons appliqué CPO-ACO sur trois problèmes différents sans pour autant utiliser d'heuristiques dédiées à ces problèmes. Nous proposons d'étudier l'intérêt d'ajouter des heuristiques dépendantes des problèmes à résoudre et voir si cela lui permet de devenir compétitif avec les approches existantes dans l'état de l'art.

Le réglage des paramètres est un autre point intéressant. Pour l'instant les paramètres de CPO-ACO sont réglés en utilisant notre expérience. Cependant, on pense qu'une version adaptative qui change dynamiquement les valeurs des paramètres pendant l'exécution peut augmenter l'efficacité de l'algorithme et sa robustesse.

6 Remerciements

Les auteurs remercient Renaud Dumeur, Jérôme Rogerie, Philippe Refalo et Philippe Laborie pour les discussions fructueuses et animées sur l'optimisation combinatoire et, en particulier, les discussions sur les stratégies de recherche. Aussi, nous remercions Jérôme Rogerie et Philippe Laborie pour leur relecture de cet article.

Références

[1] Marco Dorigo and Thomas Stützle. *Ant Colony Optimization*. MIT Press, 2004.

[2] F. Glover and M. Laguna. *Tabu Search*. Kluwer Academic Publishers, 1997.

[3] Pascal Van Hentenryck and Laurent Michel. *Constraint-Based Local Search*. MIT Press, 2005.

[4] M. Khichane, P. Albert, and C. Solnon. Integration of *aco* in a constraint programming language. *6th International Conference on Ant Colony Optimization and Swarm Intelligence (A ANTS2008)*, (5217) :84–95, 2008.

[5] S. Kirkpatrick, C. Gellat, , and M. Vecchi. Optimization by simulated annealing. *science*, 220 :671–680, 1983.

[6] A.H. Land and A.G. Doig. An automatic method of solving discrete programming problems. *Econometrica*, 28 :497–520, 1960.

[7] A.K. Mackworth and E.C. Freuder. The complexity of some polynomial network consistency algorithms for constraint satisfaction problems[ac1-3]. *Artificial Intelligence*, (25) :65–74, 1985.

[8] B. Meyer. *Hybrids of constructive meta-heuristics and constraint programming : A case study with ACO*. In Chr. Blum, M.J.Blesa, A. Roli, and M. Sampels, editors, Hybrid Metaheuristics-An emergent approach for optimization. Springer Verlag, New York, 2008.

[9] GL. Nemhauser and AL. Wolsey. *Integer and combinatorial optimization*. New York : Jhon Wiley and & Sons ;, 1988.

[10] CH. Papadimitriou and K. Steiglitz. *Combinatorial optimization–Algorithms and complexity*. New York :Dover ;, 1982.

[11] P. Refalo. Impact-based search strategies for constraint programming. *In CP04, 10th International Conference on Principales and Practice of Constraint Programming,*, (10) :557–571, 2004.

[12] C. Solnon. Ants can solve constraint satisfaction problems. *IEEE Transactions on Evolutionary Computation*, 6(4) :347–357, 2002.

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