Programmation par contraintes avec des fourmis

Madjid Khichane, Patrick Albert, Christine Solnon

Actes JFPC 2008

Programmation par contraintes avec des fourmis

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

ILOG S.A.[1]
9 rue de Verdun, 94253 Gentilly cedex
{mkhichane,palbert}@ilog.fr

LIRIS, CNRS UMR 5205, Université de Lyon I[2]
43 Bd du 11 Novembre
69622 Villeurbanne Cedex
christine.solnon@liris.cnrs.fr

Résumé

On explore dans cet article les possibilités d'intégration de la métaheuristique d'optimisation par colonies de fourmis (ACO) dans un langage de programmation par contraintes (PPC). L'idée est d'utiliser un langage de PPC pour décrire le problème à résoudre, ainsi que pour propager et vérifier les contraintes, et d'utiliser ACO pour guider la recherche de solutions. On illustre notre approche sur le problème d'ordonnancement de voitures.

1 Introduction

Nos motivations trouvent leur origine dans le double constat suivant :

- d'une part, la programmation par contraintes (PPC) permet de décrire des problèmes combinatoires de façon très déclarative... mais les procédures de résolution intégrées aux langages de PPC, traditionnellement basées sur une exploration par "séparation et propagation", ne sont pas toujours capables de résoudre les instances difficiles en un temps acceptable ;
- d'autre part, la métaheuristique d'optimisation par colonies de fourmis (Ant Colony Optimization / ACO) [8] a été utilisée avec succès pour résoudre de très nombreux problèmes d'optimisation combinatoires... mais la majorité des travaux s'est concentrée sur la conception d'algorithmes efficaces pour résoudre différents problèmes, et non sur l'intégration de ces algorithmes dans des langages déclaratifs, de sorte que résoudre un nouveau problème avec ACO demande un important travail de programmation.

Ainsi, nous explorons dans cet article les possibilités d'intégration de la métaheuristique ACO dans un langage de PPC.

1.1 Programmation par contraintes

Les langages de programmation par contraintes permettent de décrire des problèmes de façon déclarative, en termes de contraintes, la résolution de ces problèmes étant prise en charge par des algorithmes de résolution intégrés au langage, sans que l'utilisateur n'ait besoin de « programmer » cette résolution.

Les problèmes décrits en termes de contraintes sont appelés des CSP et sont définis par un triplet (X, D, C) tel que X est un ensemble fini de variables, D est une fonction qui associe à chaque variable $x_i \in X$ son domaine $D(x_i)$, et C est un ensemble fini de contraintes. Chaque contrainte $c_j \in C$ est une relation entre certaines variables de X, restreignant les valeurs que peuvent prendre simultanément ces variables.

Résoudre un CSP (X, D, C) consiste à affecter des valeurs aux variables, de telle sorte que toutes les contraintes soient satisfaites. On note $\langle x_i, v_i \rangle$ l'affectation de la valeur v_i à la variable x_i. Une *affectation* est un ensemble de couples $\langle x_i, v_i \rangle$ et on note $var(A)$ l'ensemble des variables affectées à une valeur dans une affectation A. Une affectation est dite *totale* si elle instancie toutes les variables du problème ; elle est dite *partielle* si elle n'en instancie qu'une partie. Une affectation A *viole* une contrainte c_k si la relation définie par c_k n'est pas vérifiée pour les valeurs des

variables de c_k définies dans A. Une affectation (totale ou partielle) est *consistante* si elle ne viole aucune contrainte, et *inconsistante* si elle viole une ou plusieurs contraintes. Une *solution* est une affectation totale consistante, c'est-à-dire une valuation de toutes les variables du problème qui ne viole aucune contrainte.

La PPC intègre des algorithmes de résolution de CSP à des langages de haut niveau permettant de décrire des CSP. Le succès de cette intégration dépend des deux points fondamentaux suivants :

- le langage doit permettre à l'utilisateur de décrire son problème facilement, de façon déclarative, sans avoir à entrer dans des détails liés à sa résolution ;
- la procédure de résolution de CSP intégrée au langage doit être capable de résoudre efficacement les problèmes ainsi modélisés.

Evidemment, ces deux points ne sont pas faciles à concilier : une résolution est généralement d'autant plus efficace qu'elle exploite les spécificités du problème pour guider la recherche de solutions.

La grande majorité des langages de PPC intègrent des algorithmes de résolution basés sur une exploration exhaustive de l'espace des affectations (jusqu'à trouver une solution ou prouver l'absence de solution). Afin de (tenter de) réduire la combinatoire, l'espace des affectations est structuré en arbre et l'exploration de cet arbre est couplée avec des techniques de propagation de contraintes : à chaque fois qu'une valeur est affectée à une variable, les contraintes portant sur cette variable sont propagées afin d'éliminer du domaine des variables non affectées des valeurs inconsistantes. On peut considérer différents algorithmes de filtrage, qui réduisent plus ou moins fortement les domaines des variables, mais qui sont également plus ou moins coûteux en temps.

Dans cet article, nous utilisons la bibliothèque de programmation par contraintes Ilog solver. Cette bibliothèque contient un grand nombre de méthodes permettant de déclarer des variables et des contraintes ; des procédures de vérification et de propagation sont associées aux contraintes et sont activées au fur et à mesure de la construction d'affectations. Par défaut, la stratégie de recherche de solution est basée sur une recherche exhaustive arborescente. Nous proposons de remplacer cette stratégie de recherche arborescente par une recherche de solutions guidée par l'optimisation par colonies de fourmis.

1.2 Optimisation par colonies de fourmis

Le principe de base de la métaheuristique ACO [8] est de construire itérativement des combinaisons selon un principe glouton aléatoire. Plus précisément, à chaque cycle, chaque fourmi construit une combinaison, en partant d'une combinaison vide et en ajoutant un par un des composants de combinaison jusqu'à obtenir une combinaison complète. A chaque itération, le composant à ajouter est choisi selon une probabilité qui dépend de traces de phéromone : ces traces sont associées aux composants de combinaisons ; elles sont renforcées lorsque les composants ont permis de construire de bonnes combinaisons ; elles sont diminuées à la fin de chaque cycle par évaporation afin d'oublier progressivement les constructions plus anciennes. Ces traces de phéromone représentent l'expérience passée de la colonie concernant le choix des composants et sont utilisées pour biaiser les probabilités de choix de ces composants.

Le premier algorithme à base de fourmis a été proposé par Dorigo en 1992 [5]. Le problème choisi pour ces premières expérimentations est celui du voyageur de commerce. Ce problème a été l'objet de nombreuses études concernant les capacités de résolution des fourmis artificielles [7, 6]. La métaheuristique ACO est une généralisation de ces premiers algorithmes à base de fourmis, et a été appliquée avec succès à de nombreux autres problèmes combinatoires comme par exemple les problèmes d'affectation quadratique [10], de routage de véhicules [3] ou de recherche de cliques maximums [25].

Plusieurs algorithmes ACO ont été appliqués à la résolution de CSP [23, 26]. Dans ces algorithmes, des fourmis construisent itérativement des affectations selon un principe glouton aléatoire : partant d'une affectation vide, elles choisissent à chaque itération une variable non instanciée et une valeur pour cette variable, jusqu'à ce que toutes les variables soient instanciées. Les variables sont choisies en utilisant des heuristiques d'ordre classiques comme, par exemple, l'heuristique "min-domain". Les valeurs sont choisies en fonction de probabilités qui dépendent d'un facteur phéromonal représentant l'expérience passée de la colonie concernant le choix de cette valeur. Ainsi, la phéromone est utilisée dans ces algorithmes comme une heuristique de choix de valeur.

1.3 Principe d'une intégration d'ACO dans un langage de PPC

Nous proposons d'utiliser un langage de PPC, à savoir Ilog solver, pour décrire le problème à résoudre, mais de remplacer la procédure de recherche arborescente prédéfinie par une recherche ACO. On réutilise ainsi tout le travail fait par Ilog à la fois au niveau de la modélisation de problèmes et au niveau des algorithmes de vérification et de propagation de contraintes.

L'utilisation d'Ilog solver pour la modélisation, la

vérification et la propagation de contraintes nous impose d'adapter les algorithmes ACO proposés précédemment pour la résolution de CSP :

- dans les algorithmes introduits dans [23] et [26], chaque fourmi construit une affectation *complète* (telle que chaque variable est affectée à une valeur), cette affectation pouvant éventuellement être *inconsistante*; dans ce cas, la qualité des affectations construites dépend du nombre de contraintes violées, et *l'objectif des fourmis est de construire l'affectation complète violant le moins de contraintes*;
- dans notre nouvel algorithme, appelé *Ant-CP*, chaque fourmi construit une affectation *consistante* (ne violant aucune contrainte), cette affectation pouvant éventuellement être *partielle* (certaines variables n'étant pas affectées); dans ce cas, la qualité des affectations construites dépend du nombre de variables affectées, et *l'objectif des fourmis est de construire l'affectation consistante affectant le plus grand nombre de variables.*

Notons qu'une hybridation d'un algorithme ACO avec des techniques de propagation de contraintes a été proposée dans [17] pour résoudre un problème d'emploi du temps comportant à la fois des contraintes dures et une fonction objectif à optimiser[1] : l'idée est d'utiliser la propagation de contraintes pour permettre aux fourmis de construire des combinaisons satisfaisant toutes les contraintes dures, la phéromone étant utilisée pour guider la recherche par rapport à la fonction objectif. L'approche que nous proposons ici est différente : nous utilisons ACO pour guider la recherche de solutions satisfaisant toutes les contraintes, celles-ci étant difficiles à satisfaire.

Nous décrivons dans la section suivante l'algorithme *Ant-CP*; nous l'illustrons ensuite sur un problème d'ordonnancement de voitures. Nous discutons en conclusion de quelques pistes plus générales pour le succès d'une programmation par contraintes basée sur l'optimisation par colonies de fourmis.

2 Description d'*Ant-CP*

L'algorithme 1 décrit la procédure *Ant-CP* utilisée pour chercher une solution, procédure qui remplace la recherche arborescente classiquement utilisée avec ILOG Solver. Notons bien que cette procédure ACO n'est pas exacte, contrairement aux procédures de recherche intégrées à ILOG Solver : *Ant-CP* n'est pas capable de prouver l'inconsistance du problème s'il n'admet pas de solution, et peut ne pas trouver de solution à un problème admettant une solution; dans ce cas, *Ant-CP* fournira en sortie la meilleure affectation trouvée, c.-à-d. celle affectant le plus grand nombre de variables.

Ant-CP construit itérativement des affectations (lignes 5 à 11) qui sont utilisées pour mettre-à-jour les traces de phéromone (lignes 16 à 20). On décrit dans les paragraphes suivants la stratégie phéromonale Φ utilisée par les fourmis pour guider la recherche, ainsi que les principales étapes de l'algorithme *Ant-CP*, à savoir, le choix de variables et de valeurs, la propagation de contraintes et la mise-à-jour de la phéromone.

2.1 Stratégie phéromonale

La stratégie phéromonale Φ est un paramètre de la procédure *Ant-CP* et est définie par un triplet $(S, \tau, comp)$ tel que :

- S est l'ensemble des composants sur lesquels les fourmis peuvent déposer de la phéromone;
- τ est la fonction qui détermine comment exploiter les traces de phéromone au moment de choisir une valeur pour une variable; plus précisément, étant données une affectation partielle en cours de construction $\mathcal{A}$, une variable x_j et une valeur $v \in D(x_j)$, $\tau(\mathcal{A}, x_j, v)$ retourne la valeur du facteur phéromonal qui évalue l'expérience passée de la colonie concernant le fait d'ajouter $\langle x_j, v\rangle$ à l'affectation partielle $\mathcal{A}$;
- *comp* est la fonction qui détermine l'ensemble des composants phéromonaux sur lesquels déposer de la phéromone quand on veut récompenser une affectation $\mathcal{A}$; plus précisément, étant donnée une affectation partielle $\mathcal{A}$, $comp(\mathcal{A})$ retourne l'ensemble des composants phéromonaux de S qui sont associés à l'affectation $\mathcal{A}$.

L'objectif de la stratégie phéromonale est d'apprendre quelles sont les décisions qui ont permis de construire de bonnes affectations afin de pouvoir utiliser cette information pour biaiser les constructions futures. Par défaut, la stratégie phéromonale considérée, notée Φ_{defaut}, est la suivante :

- les fourmis déposent de la phéromone sur les couples variable/valeur, c.-à-d.,

$$S = \{\tau_{\langle x_i, v\rangle} | x_i \in X, v \in D(x_i)\}$$

  de sorte que chaque trace phéromonale $\tau_{\langle x_i, v\rangle}$ représente l'expérience passée de la colonie concernant le fait d'affecter la valeur v à la variable x_i;
- le facteur phéromonal est défini par

$$\tau(\mathcal{A}, x_j, v) = \tau_{\langle x_j, v\rangle}$$

- l'ensemble des composants phéromonaux sur lesquels de la phéromone est déposée quand on récompense une affectation $\mathcal{A}$ est l'ensemble des

[1] Il s'agit à notre connaissance de la seule approche hybridant ACO avec des techniques de propagation de contraintes.

Algorithme 1 – *Ant-CP*

Entrée :
Un CSP (X, D, C),
Une stratégie phéromonale $\Phi = (S, \tau, comp)$,
Un facteur heuristique η,
Un ensemble de paramètres $\{\alpha, \beta, \rho, \tau_{min}, \tau_{max}, nbAnts, maxCycles\}$

Postrelation :
Retourne une affectation consistante (éventuellement partielle) pour (X, D, C)

Initialiser toutes les traces de S à τ_{max}
$\mathcal{A}_{best} \leftarrow \emptyset$
répéter
  pour *chaque* $k \in 1..nbAnts$ **faire**
    `/* Construction d'une affectation consistante` $\mathcal{A}_k$ `*/`
    $\mathcal{A}_k \leftarrow \emptyset$
    répéter
      Choisir une variable $x_j \in X$ telle que $x_j \notin var(\mathcal{A}_k)$
      Choisir une valeur $v \in D(x_j)$ selon la probabilité $p(x_j, v) = \frac{[\tau(\mathcal{A}_k, x_j, v)]^\alpha [\eta(\mathcal{A}_k, x_j, v)]^\beta}{\sum_{w \in D(x_j)} [\tau(\mathcal{A}_k, x_j, w)]^\alpha [\eta(\mathcal{A}_k, x_j, w)]^\beta}$
      Ajouter $\langle x_j, v \rangle$ à $\mathcal{A}_k$
      Propager les contraintes
    jusqu'à $var(\mathcal{A}_k) = X$ *ou Echec* ;
    si $card(\mathcal{A}_k) \geq card(\mathcal{A}_{best})$ **alors** $\mathcal{A}_{best} \leftarrow \mathcal{A}_k$
    si $var(\mathcal{A}_{best}) = X$ **alors retourner** $\mathcal{A}_{best}$
  finpour
  `/* Mise-à-jour de la phéromone */`
  Evaporer chaque trace de S en la multipliant par $(1 - \rho)$
  pour *chaque affectation partielle* $\mathcal{A}_k \in \{\mathcal{A}_1, \ldots, \mathcal{A}_{nbAnts}\}$ **faire**
    si $\forall \mathcal{A}_i \in \{\mathcal{A}_1, \ldots, \mathcal{A}_{nbAnts}\}, card(\mathcal{A}_k) \geq card(\mathcal{A}_i)$ **alors**
      incrémenter chaque trace de $comp(\mathcal{A}_k)$ de $\frac{1}{1 + card(\mathcal{A}_{best}) - card(\mathcal{A}_k)}$
    finsi
  finpour
  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 *maxCycles atteint* ;
retourner $\mathcal{A}_{best}$

couples variable/valeur contenus dans $\mathcal{A}$, c.-à-d.,

$$comp(\mathcal{A}) = \{\tau_{\langle x_i,v\rangle} | \langle x_i, v\rangle \in \mathcal{A}\}$$

Pour certains problèmes, il peut être souhaitable de considérer d'autres stratégies phéromonales de sorte que l'utilisateur peut définir sa propre stratégie phéromonale. Il doit alors définir le triplet $(S, \tau, comp)$ correspondant à sa stratégie. Cet aspect sera illustré en 3 sur le problème d'ordonnancement de voitures.

2.2 Choix d'une variable et d'une valeur

A chaque itération, la prochaine variable à instancier est choisie selon une heuristique d'ordre donnée (ligne 7). On peut utiliser pour cela les nombreuses heuristiques d'ordre prédéfinies dans ILOG Solver comme, par exemple, l'heuristique `IloChooseMinSizeInt` qui consiste à choisir la variable ayant le plus petit domaine.

Après avoir choisi la prochaine variable à affecter, les fourmis choisissent une valeur pour cette variable (ligne 8). La contribution principale d'ACO pour la résolution de CSP est de fournir une heuristique générique pour ce choix de valeur : la valeur v à affecter à la variable x_j est choisie aléatoirement dans le domaine $D(x_j)$ en fonction d'une probabilité $p(x_j, v)$ dépendant d'un facteur phéromonal $\tau(\mathcal{A}, x_j, v)$ et d'un facteur heuristique $\eta(\mathcal{A}, x_j, v)$ dont les importances relatives sont modulées par deux paramètres α et β. Le facteur phéromonal reflète l'expérience passée de la colonie concernant l'ajout de $\langle x_j, v\rangle$ à l'affectation partielle $\mathcal{A}$; sa définition dépend de la stratégie phéromonale Φ. Le facteur heuristique permet d'introduire des heuristiques d'ordre de choix de valeurs. Il peut s'agir d'heuristiques génériques, comme celles prédéfinies dans ILOG solver. Il peut également s'agir d'heuristiques spécifiques aux problèmes considérés comme, par exemple, l'heuristique introduite dans [26] pour le problème d'ordonnancement de voitures.

2.3 Propagation de contraintes

A chaque fois qu'une variable est affectée à une valeur, les contraintes sont propagées afin de réduire les domaines des variables qui ne sont pas encore affectées (ligne 10). Si le domaine d'une variable est réduit à un singleton, l'affectation partielle en cours de construction est complétée avec l'affectation de cette variable et le processus de propagation de contraintes est continué. Si le domaine d'une variable devient vide, ou si une inconsistance est détectée, alors la propagation se termine sur un échec et la construction de l'affectation partielle $\mathcal{A}_k$ se termine.

On utilise pour cela les algorithmes de propagation intégrés dans ILOG Solver.

2.4 Mise-à-jour de la phéromone

Une fois que chaque fourmi a construit une affectation, les traces de phéromone de S sont mises-à-jour selon la méta-heuristique ACO : elles sont tout d'abord diminuées par évaporation (ligne 15) ; elles sont ensuite récompensées en fonction de leur contribution à la construction des meilleures affectations du cycle, c.-à-d., celles ayant affecté le plus grand nombre de variables (lignes 16 à 20). L'ensemble des composants phéromonaux à récompenser dépend de la stratégie phéromonale considérée. La quantité de phéromone déposée sur ces composants est inversement proportionnelle à la différence entre le nombre de variables affectées dans la meilleure affectation construite depuis le début de l'exécution, $\mathcal{A}_{best}$, et le nombre de variables affectées dans l'affectation récompensée $\mathcal{A}_k$.

Enfin, les traces de phéromone sont bornées entre τ_{min} et τ_{max} (lignes 21 et 22) selon le principe du $\mathcal{MAX} - \mathcal{MIN}$ *Ant System*.

3 Illustration sur le problème d'ordonnancement de voitures

On montre maintenant comment utiliser *Ant-CP* pour résoudre un problème d'ordonnancement de voitures. Ce problème consiste à séquencer des voitures le long d'une chaîne de montage pour installer des options sur les voitures (par exemple, un toit ouvrant ou l'air conditionné). Chaque option est installée par une station différente conçue pour traiter au plus un certain pourcentage de voitures passant le long de la chaîne. Par conséquent, les voitures demandant une même option doivent être espacées de telle sorte que la capacité des stations ne soit jamais dépassée. Plus formellement, on définira une instance de ce problème par un tuple (V, O, p, q, r) tel que

- V est l'ensemble des voitures à produire,
- O est l'ensemble des différentes options possibles,
- $p : O \to \mathbb{N}$ et $q : O \to \mathbb{N}$ sont deux fonctions qui définissent la contrainte de capacité associée à chaque option $o_i \in O$, c.-à-d., pour chaque séquence de $q(o_i)$ voitures consécutives sur la ligne d'assemblage, au plus $p(o_i)$ peuvent demander l'option o_i,
- $r : V \times O \to \{0, 1\}$ est la fonction qui définit les options requises, c.-à-d., pour chaque voiture $v_i \in V$ et pour chaque option $o_j \in O$, si o_j doit être installée sur v_i, alors $r(v_i, o_j) = 1$ sinon $r(v_i, o_j) = 0$.

Il s'agit alors de chercher un ordonnancement des voitures qui satisfait les contraintes de capacité. Ce problème a été montré $\mathcal{NP}$-difficile par Kis [15]. Une variante de ce problème, faisant intervenir, en plus

des contraintes de capacité liées aux options, des contraintes liées à la couleur des voitures, a fait l'objet du challenge ROADEF en 2005 [24].

3.1 Modèle PPC considéré

Le problème d'ordonnancement de voitures à été introduit dans la communauté PPC en 1988, par Dincbas, Simonis et Van Hentenryck [4]. Il est devenu depuis un problème emblématique, souvent utilisé pour évaluer les performances des langages de PPC. Ainsi, ce problème est le premier de la bibliothèque *CSPlib* [11].

Pour évaluer *Ant-CP*, nous avons considéré un modèle PPC classique, correspondant au premier modèle proposé dans le manuel utilisateur d'ILOG Solver pour le problème d'ordonnancement de voitures[2]. Afin de diminuer la taille de l'espace de recherche, ce modèle introduit la notion de classe de voitures : toutes les voitures demandant un même ensemble d'options sont regroupées dans une même classe.

Les variables du CSP sont de deux types :

- on associe à chaque position i dans la séquence de voitures une variable x_i représentant la classe de la voiture à cette position ; le domaine de ces variables est défini par l'ensemble des classes de voitures ;
- on associe à chaque position i dans la séquence de voitures et chaque option j une variable o_i^j indiquant si la voiture à la position i demande l'option j ; le domaine de ces variables est l'ensemble $\{0, 1\}$ de sorte que $o_i^j = 1$ si l'option j doit être installée sur la voiture i, et 0 sinon.

Les contraintes sont de trois types :

- une première série de contraintes fait le lien entre les variables associées aux positions et les variables associées aux options, spécifiant que $o_i^j = 1$ si et seulement si l'option j doit être installée sur la voiture x_i ;
- une deuxième série de contraintes concerne la capacité des stations et spécifie que, pour chaque option j et chaque sous-séquence de q_j voitures consécutives, la somme des variables o_i^j correspondantes doit être inférieure ou égale à p_j ;
- une troisième série de contraintes spécifie pour chaque classe de voitures le nombre de voitures de cette classe devant être séquencées.

3.2 Heuristique d'ordre de choix de variables

On utilise une heuristique d'ordre classique pour ce problème d'ordonnancement de voitures : on affecte les variables x_i associées aux positions dans l'ordre défini par les positions. Notons que les variables o_i^j sont affectées par propagation de contraintes quand la variable x_i correspondante est affectée ou quand les contraintes de capacité permettent d'inférer qu'une option ne peut être installée sur une voiture.

[2]Nous invitons le lecteur intéressé à consulter le manuel utilisateur pour plus d'informations sur ce modèle.

3.3 Stratégies phéromonales considérées

La stratégie phéromonale par défaut Φ_{defaut} associe une trace de phéromone à chaque couple $\langle x_i, v\rangle$ tel que x_i est une variable associée à une position et v est une valeur du domaine de x_i, c.-à-d., une classe de voiture.

Nous comparons les performances de la stratégie phéromonale par défaut avec deux autres stratégies phéromonales dénotées par $\Phi_{classes}$ et Φ_{cars}.

3.3.1 Stratégie $\Phi_{classes}$

Cette stratégie phéromonale a été proposée dans [13]. La structure phéromonale S associe une trace $\tau_{(v,w)}$ à chaque couple de classes de voitures (v, w). Cette trace représente l'expérience de la colonie concernant le fait d'affecter la valeur w à une variable x_i quand x_{i-1} a été affectée à la valeur v ou, autrement dit, de séquencer une voiture de la classe w juste derrière une voiture de la classe v. Pour cette stratégie phéromonale, le facteur phéromonal est égal à la trace de phéromone entre la classe de la dernière voiture séquencée et la classe candidate (le facteur phéromonal est égal à un pour la première variable) :

$$\begin{array}{ll} \tau(\mathcal{A}, x_k, v) = \tau_{(w,v)} & si\ k > 1\ et\ \langle x_{k-1}, w\rangle \in \mathcal{A} \\ \tau(\mathcal{A}, x_k, v) = 1 & si\ k = 1 \end{array}$$

Lors de la mise-à-jour des traces de phéromone, la phéromone est déposée sur les couples de valeurs affectées consécutivement dans les affectations à récompenser, c.-à-d.,

$$comp(\mathcal{A}) = \{\tau_{(v,w)} | \exists x_l \in X, \{\langle x_l, v\rangle, \langle x_{l+1}, w\rangle\} \subseteq \mathcal{A}\}$$

3.3.2 Stratégie Φ_{cars}

Cette stratégie phéromonale a été proposée dans [26]. La structure phéromonale S associe une trace $\tau_{(v,i,w,j)}$ à chaque couple de classes de voitures (v, w) et chaque $i \in [1; \#v]$ et $j \in [1; \#w]$ où $\#v$ et $\#w$ sont les nombres de voitures des classes v et w respectivement. Cette trace représente l'expérience de la colonie concernant le fait de séquencer la j^{eme} voiture de la classe w juste derrière la i^{eme} voiture de la classe v. Dans ce cas, le facteur phéromonal est défini par

$$\begin{array}{ll} \tau(\mathcal{A}, x_k, v) = \tau_{(w,j,v,i+1)} & si\ k > 1\ et\ \langle x_{k-1}, w\rangle \in \mathcal{A} \\ & et\ j = card(\{\langle x_l, w\rangle \in \mathcal{A}\}) \\ & et\ i = card(\{\langle x_l, v\rangle \in \mathcal{A}\}) \\ \tau(\mathcal{A}, x_k, v) = 1 & si\ k = 1 \end{array}$$

Lors de la mise-à-jour des traces de phéromone, la phéromone est déposée sur les composants correspondant à des couples de voitures séquencées consécutivement dans les affectations à récompenser :

$$comp(\mathcal{A}) = \{\tau_{(v,i,w,j)} | \quad \exists x_l \in X, \{\langle x_l, v\rangle, \langle x_{l+1}, w\rangle\} \subseteq \mathcal{A}_k \\ et\ i = card(\{\langle x_m, v\rangle / m \leq l\}) \\ et\ j = card(\{\langle x_m, w\rangle / m \leq l+1\})\}$$

3.4 Facteur heuristique

Dans la probabilité de transition utilisée pour choisir la valeur à affecter à une variable, le facteur phéromonal $\tau(\mathcal{A}_k, x_j, v)$ —qui représente l'expérience passée de la colonie— est combiné à un facteur heuristique $\eta(\mathcal{A}_k, x_j, v)$ —qui dépend du problème. On introduit maintenant deux définitions pour ce facteur heuristique.

3.4.1 Somme des taux d'utilisation (DSU)

Nous avons introduit et comparé dans [12] cinq heuristiques différentes pour le problème d'ordonnancement de voitures. Ces définitions sont toutes liées à la notion de taux d'utilisation d'une option initialement introduite dans [22], notion qui traduit le fait qu'une option est d'autant plus critique qu'elle est demandée par beaucoup de voitures par rapport à sa capacité. La fonction heuristique qui s'est montrée la plus performante en moyenne est basée sur la somme des taux d'utilisation des options demandées par la voiture candidate, c.-à-d.,

$$\eta(\mathcal{A}_k, x_j, v) = \sum_{o_i \in \text{reqOptions}(v)} \frac{\text{reqSlots}(o_i, n_i)}{N}$$

où reqOptions(v) est l'ensemble des options demandées par la classe de voitures v, N est le nombre de voitures n'ayant pas encore été séquencées dans $\mathcal{A}$, n_i est le nombre de voitures demandant l'option o_i et n'ayant pas encore été séquencées dans $\mathcal{A}$, et reqSlots(o_i, n_i) est le nombre minimum de positions permettant de placer n_i voitures demandant l'option o_i sans violer de contraintes de capacité pour l'option o_i. Pour la définition de reqSlots(o_i, n_i), on considère la formule introduite dans [1] (plus précise que celle de [12] qui ne tient pas compte du fait que n_i n'est pas forcément un multiple de q_i)) :

si $n_i \ \% \ p_i = 0$ alors
  $\text{reqSlots}(o_i, n_i) = q_i * n_i / p_i - (q_i - p_i)$
sinon
  $\text{reqSlots}(o_i, n_i) = q_i * (n_i - n_i \ \% \ p_i)/p_i + n_i \ \% \ p_i$

où % est l'opérateur retournant le reste de la division entière.

3.4.2 Somme des taux d'utilisation avec propagation (DSU+P)

On peut exploiter les taux d'utilisation des options pour détecter des inconsistances et pour filtrer les domaines des variables :

- quand le taux d'utilisation d'une option devient supérieur à 1, i.e., quand $reqSlots(o_i, n_i)$ devient supérieur à N, on peut en conclure qu'il n'est pas possible de compléter la séquence sans violer de contraintes, et on peut arrêter la construction en cours sur un échec ;
- quand le taux d'utilisation d'une ou plusieurs options devient égal à 1, on peut supprimer du domaine de la prochaine variable toutes les classes de voitures ne demandant pas toutes les options dont le taux d'utilisation est égal à 1.

4 Résultats expérimentaux

4.1 Jeux d'essais

Les instances satisfiables de la bibliothèque CSPlib [11] à 100 voitures (4 instances proposées par Régin et Puget [21]) et 200 voitures (70 instances proposées par Lee et al dans [16]) sont toutes facilement résolues par *Ant-CP*, quelles que soient la structure phéromonale et l'heuristique considérées.

Nous considérons ici un jeu d'essai plus difficile généré par Perron et Shaw [20]. Les instances de ce jeu d'essai ont toutes 8 options et 20 classes de voitures différentes par instance ; les contraintes de capacité sur les options, définies par les fonctions p et q, sont générées aléatoirement en respectant les contraintes suivantes : $\forall o_i \in O, 1 \leq p(o_i) \leq 3$ et $p(o_i) < q(o_i) \leq p(o_i) + 2$. Le jeu d'essai comporte 32 instances à 100 voitures, 21 instances à 300 voitures et 29 instances à 500 voitures. Toutes ces instances admettent une solution ne violant aucune contrainte.

4.2 Instanciations d'*Ant-CP* considérées

On compare les stratégies phéromonales Φ_{defaut}, $\Phi_{classes}$ et Φ_{cars} introduites en 3.3. Afin d'évaluer l'apport de la phéromone, on considère également une stratégie sans phéromone, notée $\Phi_\emptyset$: dans ce cas, l'ensemble S est l'ensemble vide et le facteur phéromonal $\tau(\mathcal{A}_k, x_j, v)$ est fixé à 1.

On compare également les deux facteurs heuristiques DSU et DSU+P introduits en 3.4.

On note *Ant-CP*(Φ, h) l'instanciation d'*Ant-CP* obtenue avec la stratégie phéromonale $\Phi \in \{\Phi_\emptyset, \Phi_{classes}, \Phi_{cars}, \Phi_{defaut}\}$ et l'heuristique $h \in$ {DSU,DSU+P}.

4.3 Paramétrage

Ant-CP est paramétré par
- le nombre de fourmis *nbAnts*, qui détermine le nombre d'affectations construites à chaque cycle avant chaque étape de mise-à-jour de la phéromone,
- α et β, qui déterminent les poids relatifs des facteurs phéromonal et heuristique dans la probabilité de choix de valeur,
- ρ, qui détermine la vitesse d'évaporation des traces phéromonales,
- τ_{min} et τ_{max} qui fixent les bornes minimale et maximale des traces de phéromone.

La valeur du paramètre β change d'une application à l'autre, en fonction de la fiabilité du facteur heuristique. On a fixé ce paramètre à 6, comme cela a été proposé dans [12] où l'heuristique DSU a été introduite.

Les valeurs des autres paramètres déterminent l'influence de la phéromone sur le processus de résolution, et donc le degré d'intensification de la recherche. Il s'agit là de trouver un bon compromis entre, d'une part, intensifier la recherche aux alentours des zones les plus prometteuses (contenant les meilleures affectations trouvées) et, d'autre part, diversifier la recherche afin de découvrir de nouvelles zones de l'espace de recherche dont on espère qu'elles contiendront de meilleures affectations.

L'intensification d'une recherche ACO est faite par le biais de la phéromone : en déposant de la phéromone sur les composants des meilleures affectations trouvées, on augmente la probabilité de choisir ces composants lors de la construction de nouvelles affectations, et on intensifie donc l'effort de recherche aux alentours de ces composants. Cette intensification peut être modulée en jouant notamment sur le paramètre α (en augmentant α, on augmente la probabilité de choisir les composants ayant reçu plus de phéromone, et on intensifie donc la recherche autour de ces composants) et sur le taux d'évaporation ρ (en augmentant ρ, on augmente l'influence des derniers dépots phéromonaux par rapport aux dépots phéromonaux plus anciens, et on intensifie donc la recherche autour des composants des dernières affectations trouvées).

Ces mécanismes d'intensification sont contrebalancés par des mécanismes de diversification permettant d'éviter une convergence prématurée de la recherche. Cette diversification peut être modulée en jouant notamment sur le nombre de fourmis (en augmentant *nbAnts*, on augmente le nombre de affectations construites à chaque cycle), et sur les bornes τ_{min} et τ_{max} (en limitant l'intervalle $[\tau_{min}; \tau_{max}]$, on limite la formation d'autoroutes phéromonales).

Le bon paramétrage de l'algorithme dépend de la difficulté des instances à résoudre et du temps disponible pour la résolution : plus une recherche est intensifiée, et plus elle convergera rapidement, mais plus elle risque de ne pas trouver de solution pour les instances les plus difficiles. Les instances que nous considérons ici étant relativement difficiles, nous avons choisi un paramétrage permettant une intensification modérée, à savoir, $\alpha = 1$, $\rho = 0.02$, *nbAnts*=30, $\tau_{min} = 0.01$ et $\tau_{max} = 4$.

4.4 Comparaison des différentes instanciations d'*Ant-CP*

La figure 1 compare les différentes stratégies phéromonales pour l'heuristique DSU (courbes du haut), puis pour l'heuristique DSU+P (courbes du bas). On constate qu'après 3000 cycles (chaque cycle correspondant à la construction de *nbAnts* affectations) la phéromone a permis d'augmenter le pourcentage d'instances résolues de 66.32% pour *Ant-CP*($\Phi_\emptyset$,DSU) à 79.39% pour *Ant-CP*(Φ_{cars},DSU), 72.56% pour *Ant-CP*(Φ_{defaut},DSU) et 68.29% pour *Ant-CP*($\Phi_{classes}$,DSU). Ainsi, sur ces instances la meilleure stratégie phéromonale est Φ_{cars} ; la stratégie phéromonale par défaut obtient de moins bons résultats, mais est cependant nettement meilleure que $\Phi_{classes}$.

L'heuristique DSU+P améliore généralement les résultats par rapport à l'heuristique DSU. Cette amélioration est différente selon la stratégie phéromonale considérée : elle est quasiment nulle quand la phéromone est ignorée (le taux de succès pour $\Phi_\emptyset$ passe de 66.32% à 66.97%) ; elle est plus importante pour les stratégies phéromonales $\Phi_{classes}$ et Φ_{defaut} (les taux de succès passent de 68.29% à 74.39% pour $\Phi_{classes}$ et de 72.56% à 78.54% pour Φ_{defaut}) ; elle est plus modérée pour Φ_{cars} (le taux de succès passe de 79.39% à 82.32%).

Notons finalement que l'heuristique DSU+P diminue les temps d'exécution d'un cycle par rapport à l'heuristique DSU car elle filtre plus les domaines des variables et détecte plus tôt certaines inconsistances. Ainsi, pour les instances à 100 voitures, 3000 cycles prennent près de 5 minutes avec l'heuristique DSU, tandis qu'ils prennent près de 3 minutes avec l'heuristique DSU+P (sur un Intel Core Duo cadencé à 2Ghz).

4.5 Comparaison d'*Ant-CP* avec d'autres approches

De très nombreuses approches différentes ont été proposées pour le problème d'ordonnancement de voitures, qui est un problème de référence pour évaluer l'efficacité de nouvelles approches de résolution. Les approches exactes, basées sur la programmation par

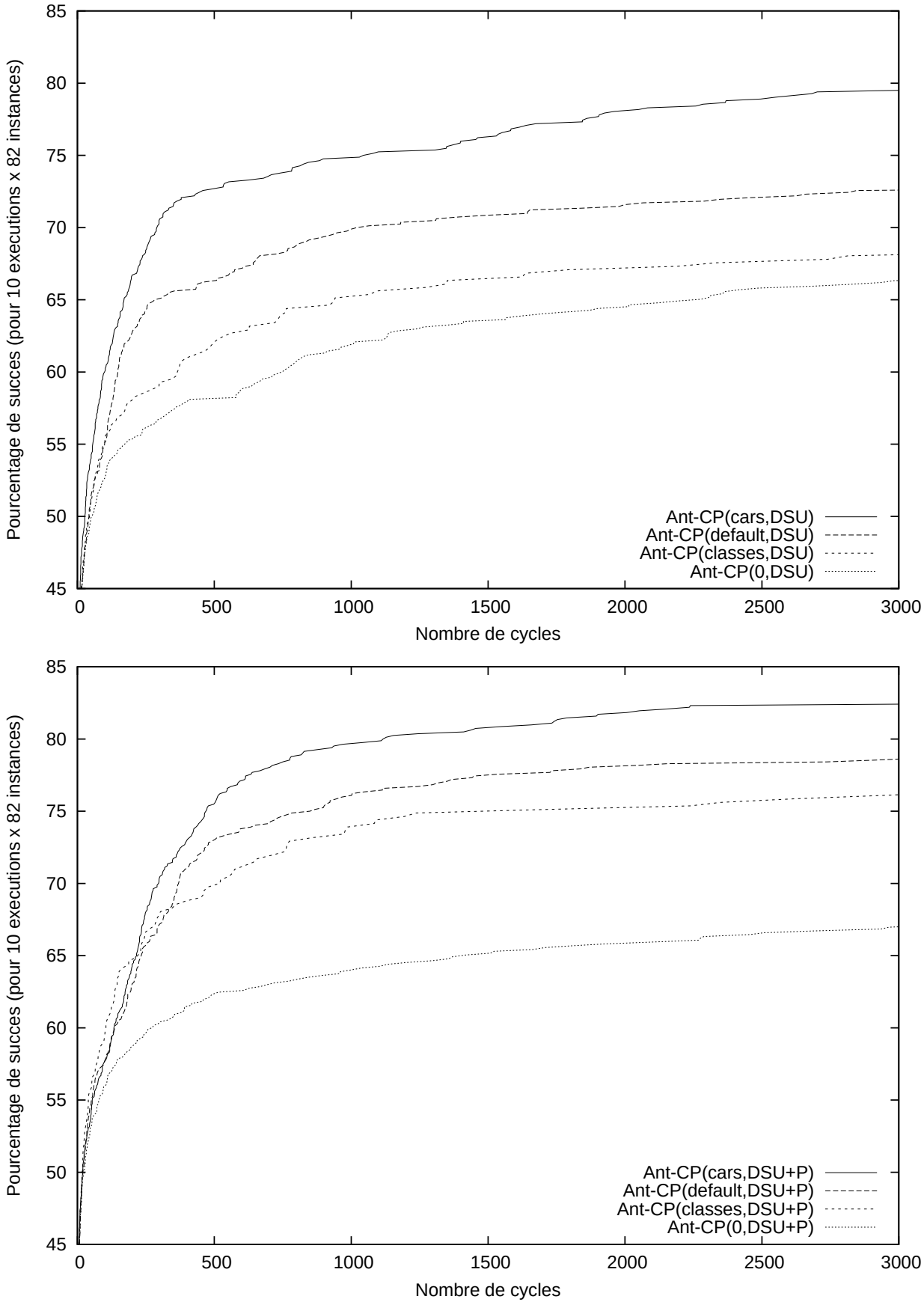

FIG. 1 – Comparaison de différentes instanciations d'*Ant-CP*(Φ, h), avec $\Phi \in \{\Phi_{defaut}, \Phi_{classes}, \Phi_{cars}, \Phi_{\emptyset}\}$ et $h \in \{DSU, DSU + P\}$: chaque courbe trace l'évolution du pourcentage d'exécutions ayant trouvé une solution en fonction du nombre de cycles (pour 10 exécutions sur chacune des 82 instances).

contraintes, ne permettent toujours pas de résoudre en un temps acceptable les instances à 100 et 200 voitures de CSPlib, même en utilisant des algorithmes de filtrage dédiés à ce problème comme, par exemple, ceux proposés dans [21, 27, 2]. Ces instances sont toutes facilement résolues par *Ant-CP*, quelle que soit la stratégie phéromonale considérée (en moins d'une seconde pour les 70 instances à 200 voitures, et moins d'une minute pour les 4 instances satisfiables à 100 voitures, sur un Intel Core Duo cadencé à 2Ghz). A titre de comparaison, aucun des différents filtrages introduits dans [2] pour la contrainte de séquence ne permet de résoudre plus de la moitié de ces instances en moins de 100 secondes avec ILOG solver sur un Pentiun 4 cadencé à 3.2 Ghz.

De nombreuses approches heuristiques ont également été proposées, la plupart d'entre elles étant basées sur une exploration par recherche locale, par exemple, pour n'en citer que quelques unes, [16, 18, 12, 19, 20, 9]. Nous avons par ailleurs proposé plusieurs algorithmes ACO pour ce problème dans [26], et nous avons montré que ces algorithmes obtiennent des résultats très compétitifs avec les meilleures approches heuristiques connues pour ce problème, et notamment VFLS [9] qui a remporté le challenge ROADEF 2005 [24] : la variante ACO la plus performante, appelée ACO(τ_1, τ_2), permet de résoudre un grand nombre d'instances nettement plus rapidement que VFLS ; elle est cependant moins performante pour quelques instances à 500 voitures.

Comme nous l'avons expliqué dans l'introduction, *Ant-CP* diffère des algorithmes ACO introduits dans [24] par le fait qu'il construit des affectations partielles consistantes, et non des affectations totales inconsistantes, l'objectif étant d'intégrer les procédures de propagation et de vérification de contraintes d'ILOG solver. On constate cependant que ces algorithmes ont des performances très similaires si l'on compare le nombre de cycles nécessaires pour résoudre les instances. Plus précisément, *Ant-CP*(Φ_{cars},DSU) obtient des résultats très similaires à ceux de la variante ACO de [26] qui utilise la même structure phéromonale et la même heuristique, à savoir ACO(τ_1, η). En revanche, si on compare le temps nécessaire pour résoudre les instances (et non le nombre de cycles), on constate qu'*Ant-CP* est plus lent que l'algorithme ACO de [26]. En effet, l'utilisation des procédures de propagation et de vérification intégrées à ILOG solver est nettement plus coûteux que l'utilisation de procédures *ad-hoc* définies pour ce problème.

En contrepartie, l'effort de programmation à fournir est différent. Pour la version d'*Ant-CP* qui utilise la stratégie phéromonale par défaut, il s'agit uniquement de décrire le problème à résoudre en déclarant les variables et leurs domaines ainsi que les contraintes. Il est intéressant à ce niveau de noter que nous avons pu utiliser pour cela le modèle décrit dans le manuel utilisateur d'ILOG solver. Pour les deux autres variantes d'*Ant-CP*, qui utilisent des stratégies phéromonales spécifiques, il s'agit en plus de spécifier l'ensemble S des composants phéromonaux ainsi que les deux fonctions τ et *comp* définissant le facteur phéromonal et les composants associés à une affectation à récompenser.

5 Conclusion

Ces premiers essais sur le problème d'ordonnancement de voitures montrent que l'on peut facilement intégrer une stratégie de recherche inspirée par les colonies de fourmis à un langage de programmation par contraintes. Cette intégration permet de bénéficier des facilités offertes par ces langages pour modéliser des problèmes de façon déclarative, en termes de contraintes. Il est intéressant de noter à ce niveau que le modèle utilisé pour décrire le problème à résoudre est le même quelle que soit l'approche de résolution utilisée. Bien évidemment, ces expérimentations devront être poursuivies sur d'autres problèmes pour pouvoir effectivement valider notre approche.

Il est important de remarquer que ces premiers résultats pourraient être améliorés en utilisant des algorithmes de propagation de contraintes dédiés à une recherche ACO. En effet, les algorithmes de propagation de contraintes intégrés dans ILOG Solver ont été conçus pour être utilisés dans un contexte de recherche arborescente avec retours-arrières. Ils maintiennent pour cela des structures de données qui permettent de restaurer, à chaque retour-arrière, le contexte du point de choix précédent. Ces structures de données qui sont coûteuses à maintenir ne sont pas nécessaires dans le contexte d'une recherche ACO où les choix faits ne sont jamais remis en cause.

Un constat similaire peut être fait en ce qui concerne l'utilisation d'un langage comme COMET pour permettre une « recherche ACO basée sur les contraintes »[3]. En effet, Van Hentenryck et Michel ont montré dans [14] que le langage COMET peut être utilisé pour implémenter un algorithme ACO de façon très déclarative. Cependant, COMET est dédié à la recherche locale, c.-à-d., à une exploration de l'espace des affectations par applications successives de transformations élémentaires à une affectation courante. Il maintient pour cela des structures de données permet-

[3] Cette expression de « recherche ACO basée sur les contraintes » est issue d'une discussion avec Pascal Van Hentenryck et Yves Deville, et fait un parallèle avec le principe de « recherche locale basée sur les contraintes » à l'origine de COMET.

tant une évaluation incrémentale et automatique des différentes propriétés invariantes et fonctions objectifs à chaque transformation élémentaire. Ces structures de données peuvent être utilisées pour une recherche ACO (pour évaluer incrémentalement les facteurs heuristiques à chaque étape de la construction) mais elles maintiennent plus de choses que nécessaire dans la mesure où les choix faits lors de la construction gloutonne des affectations ne sont jamais remis en cause. Notons cependant que ces structures de données facilitent, de fait, une hybridation avec des procédures de recherche locale.

Ainsi, une première piste de recherche pour une réelle intégration d'ACO dans un langage de programmation par contraintes concerne la conception d'algorithmes permettant une évaluation automatique et incrémentale des contraintes dans un contexte d'approche constructive où les choix faits ne sont jamais remis en cause (ni par retour-arrière comme dans les approches arborescentes, ni par application d'une transformation élémentaire comme dans la recherche locale).

Une seconde piste de recherche importante concerne le paramétrage d'ACO. Les nombreux paramètres d'un algorithme ACO et leur forte influence sur la résolution sont un frein évident à une utilisation simple dans un contexte de programmation par contraintes. Il s'agit donc d'étudier les possibilités de paramétrage adaptatif, où les valeurs des paramètres sont adaptées dynamiquement au cours du processus de résolution. Il existe des indicateurs de diversification, notamment le taux de ré-échantillonnage et le taux de similarité [25], qui peuvent être utilisés pour détecter dynamiquement des problèmes de stagnation ou de diversification trop fortes. Il s'agit maintenant de concevoir des algorithmes ACO « réactifs », capables d'ajuster dynamiquement leurs paramètres en fonction de l'évolution de ces deux indicateurs.

Références

[1] Nils Boysen and Malte Fliedner. Comments on solving real car sequencing problems with ant colony optimization . *EJOR*, 182(1) :466–468, 2007.

[2] Sebastian Brand, Nina Narodytska, Claude-Guy Quimper, Peter J. Stuckey, and Toby Walsh. Encodings of the sequence constraint. In *CP*, volume 4741 of *LNCS*, pages 210–224. Springer, 2007.

[3] B. Bullnheimer, R.F. Hartl, and C. Strauss. An improved ant system algorithm for the vehicle routing problem. *Annals of Operations Research*, 89 :319–328, 1999.

[4] M. Dincbas, H. Simonis, and P. van Hentenryck. Solving the car-sequencing problem in constraint logic programming. In Y. Kodratoff, editor, *Proceedings of ECAI-88*, pages 290–295, 1988.

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

[6] M. Dorigo and L.M. Gambardella. Ant colony system : A cooperative learning approach to the traveling salesman problem. *IEEE Transactions on Evolutionary Computation*, 1(1) :53–66, 1997.

[7] M. Dorigo, V. Maniezzo, and A. Colorni. Ant System : Optimization by a colony of cooperating agents. *IEEE Transactions on Systems, Man, and Cybernetics – Part B : Cybernetics*, 26(1) :29–41, 1996.

[8] M. Dorigo and T. Stuetzle. *Ant Colony Optimization*. MIT Press, 2004.

[9] B. Estellon, F. Gardi, and K. Nouioua. Real-life car sequencing : very large neighborhood search vs very fast local search. *European Journal of Operational Research (EJOR)*, 2007.

[10] L.M. Gambardella, E. Taillard, and M. Dorigo. Ant colonies for the quadratic assignment problem. *Journal of the Operational Research Society*, 50 :167–176, 1999.

[11] I.P. Gent and T. Walsh. Csplib : a benchmark library for constraints. Technical report, APES-09-1999, 1999. available from http ://csplib.cs.strath.ac.uk/. A shorter version appears in CP99.

[12] J. Gottlieb, M. Puchta, and C. Solnon. A study of greedy, local search and ant colony optimization approaches for car sequencing problems. In *Applications of evolutionary computing*, volume 2611 of *LNCS*, pages 246–257. Springer, 2003.

[13] M. Gravel, C. Gagné, and W.L. Price. Review and comparison of three methods for the solution of the car-sequencing problem. *Journal of the Operational Research Society*, 2004.

[14] Pascal Van Hentenryck and Laurent Michel. *Constraint-based local search*. MIT Press, 2005.

[15] T. Kis. On the complexity of the car sequencing problem. *Operations Research Letters*, 32 :331–335, 2004.

[16] J.H.M. Lee, H.F. Leung, and H.W. Won. Performance of a comprehensive and efficient constraint library using local search. In *11th Australian JCAI*, LNAI. Springer-Verlag, 1998.

[17] B. Meyer and A. Ernst. Integrating aco and constraint propagation. In *ANTS'04*, number 3172 in LNCS, pages 166–177. Springer, 2004.

[18] L. Michel and P. Van Hentenryck. A constraint-based architecture for local search. In *OOPSLA '02 : Proceedings of the 17th ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications*, pages 83–100, New York, NY, USA, 2002. ACM Press.

[19] B. Neveu, G. Trombettoni, and F. Glover. Id walk : A candidate list strategy with a simple diversification device. In *Proceedings of CP'2004*, volume 3258 of *LNCS*, pages 423–437. Springer Verlag, 2004.

[20] L. Perron and P. Shaw. Combining forces to solve the car sequencing problem. In *Proceedings of CP-AI-OR'2004*, volume 3011 of *LNCS*, pages 225–239. Springer, 2004.

[21] J.-C. Regin and J.-F. Puget. A filtering algorithm for global sequencing constraints. In *CP97*, volume 1330 of *LNCS*, pages 32–46. Springer-Verlag, 1997.

[22] B. Smith. Succeed-first or fail-first : A case study in variable and value ordering heuristics. In *third Conference on the Practical Applications of Constraint Technology PACT'97*, pages 321–330, 1996.

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

[24] C. Solnon, V.D. Cung, A. Nguyen, and C. Artigues. The car sequencing problem : overview of state-of-the-art methods and industrial case-study of the ROADEF'2005 challenge problem. *à paraître dans European Journal of Operational Research (EJOR)*, 2008.

[25] C. Solnon and S. Fenet. A study of ACO capabilities for solving the maximum clique problem. *Journal of Heuristics*, 12(3) :155–180, 2006.

[26] Christine Solnon. Combining two pheromone structures for solving the car sequencing problem with ant colony optimization. *European Journal of Operational Research (EJOR)*, 2008 (to appear).

[27] Willem Jan van Hoeve, Gilles Pesant, Louis-Martin Rousseau, and Ashish Sabharwal. Revisiting the sequence constraint. In *12th International Conference on Principles and Practice of Constraint Programming - CP 2006*, volume 4204 of *Lecture Notes in Computer Science*, pages 620–634. Springer, 2006.