Un modèle réactif pour l'optimisation par colonies de fourmis : application à la satisfaction de contraintes

Madjid Khichane, Patrick Albert, Christine Solnon

Un modèle réactif pour l'optimisation par colonies de fourmis : application à la satisfaction de contraintes

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

[1] ILOG An IBM Company
[2] Université de Lyon
Université Lyon 1, LIRIS CNRS UMR5205, France
{mkhichane,palbert}@fr.ibm.com,christine.solnon@liris.cnrs.fr

Résumé

Les algorithmes basés sur l'optimisation par colonies de fourmis (Ant Colony Optimization / ACO) sont paramétrés par deux coefficients α et β qui déterminent respectivement l'importance de l'expérience passée et de l'heuristique dans la probabilité de transition utilisée pour construire des solutions. Nous introduisons une méthode pour adapter dynamiquement les valeurs de ces deux paramètres qui ont une influence déterminante sur l'intensification et la diversification de la recherche. Cette méthode est elle-même basée sur ACO : des traces de phéromone sont associées aux valeurs potentielles des paramètres ; ces traces représentent l'expérience passée concernant l'utilisation de ces valeurs et sont utilisées pour adapter dynamiquement α et β. Nous proposons deux variantes qui diffèrent par la granularité de l'apprentissage : dans un cas l'apprentissage est global à l'ensemble des variables du problème, dans l'autre, il est spécifique à chaque variable. Nous évaluons et comparons expérimentalement ces deux méthodes dans le cadre d'une mise en œuvre de la méthode ACO pour résoudre des problèmes de satisfaction de contraintes.

Abstract

We introduce two reactive frameworks for dynamically adapting some parameters of an Ant Colony Optimization (ACO) algorithm. Both reactive frameworks use ACO to adapt parameters : pheromone trails are associated with parameter values ; these pheromone trails represent the learnt desirability of using parameter values and are used to dynamically set parameters in a probabilistic way. The two frameworks differ in the granularity of parameter learning. We experimentally evaluate these two frameworks on an ACO algorithm for solving constraint satisfaction problems.

1 Introduction

L'Optimisation par Colonies de Fourmis —Ant Colony Optimization / ACO— a démontré sont intérêt pour résoudre de nombreux problèmes combinatoires [4]. Par contre, la résolution d'un problème particulier avec ACO (de même que dans le cas d'autres méta-heuristiques) nécessite la résolution d'un compromis entre deux objectifs contradictoires : d'un coté, il est nécessaire d'intensifier la recherche de solution autour des zones les plus prometteuses alors que de l'autre, il est nécessaire de diversifier la recherche pour découvrir de nouvelles zones de l'espace de recherche qui pourraient s'avérer porteuses de meilleures solutions. C'est en réglant des paramètres de l'algorithme que l'on contrôle le comportement de la recherche vis-à-vis de ce double objectif d'intensification et de diversification (également nommés exploitation et exploration).

Le réglage de ces paramètres est bien entendu un problème délicat car il faut choisir entre deux tendances contradictoires : si l'on choisit des valeurs de paramètres qui privilégient la diversification, la qualité de la solution obtenue est souvent très bonne, mais au prix d'une convergence plus lente, et donc d'un temps de calcul plus long. Si par contre on choisit des valeurs qui favorisent l'intensification, l'algorithme sera amené à trouver de bonnes solutions plus rapidement, mais souvent sans converger vers la ou les meilleures solutions ; il convergera vers des valeurs sous-optimales. De ce fait, les valeurs optimales des paramètres dépendent à la fois de l'instance du problème à traiter et du temps alloué à sa résolution. De surcroit, la réso-

lution n'étant pas nécessairement homogène, il peut s'avérer utile de changer les valeurs des paramétres en cours de résolution afin de s'adapter au mieux aux caractéristiques locales de l'espace de recherche en cours d'exploration.

Pour améliorer le processus de recherche vis-à-vis de la dualité intensification/diversification, Battiti et al [2] ont proposé d'exploiter l'historique de la recherche pour adapter automatiquement et dynamiquement les valeurs de paramètres, donnant ainsi naissance aux approches dites "réactives".

Dans ce papier, nous introduisons une méthode réactive permettant à ACO d'adapter dynamiquement certains de ses paramètres pendant la recherche de solution. Cette adaptation dynamique est réalisée avec ACO lui-même : des traces de phéromone sont associées aux valeurs des paramètres ; elles représentent l'intérêt appris d'utiliser les valeurs des paramètres concernés et sont utilisées dynamiquement pour les affecter de façon probabiliste. Notre approche est évaluée expérimentalement dans le cadre de l'application de ACO pour résoudre des problèmes de satisfaction de contraintes (CSP) dont l'essence est de trouver une affectation de valeurs à des variables tout en respectant les contraintes auxquelles elles sont soumises.

L'article est organisé comme suit. Nous rappelons dans le chapitre 2 les principes de base des deux modèles que nous mettons en œuvre : les CSP et ACO. Le troisième chapitre est consacré à la description de la façon dont nous utilisons ACO pour adapter certains des paramètres pendant la recherche de la solution. Nous introduisons deux cadres réactifs pour ACO : dans le premier, les valeurs de paramètres sont fixées pour la durée de la construction de chaque solution, alors que dans le second, les paramètres sont associés à chaque variable —ils sont alors modifiés dynamiquement pendant la construction de chacune des solutions. Nous exposons au chapitre quatre l'évaluation expérimentale et la comparaison de ces deux variantes. Le dernier chapitre nous donne l'occasion de conclure en présentant des travaux proches ainsi que les évolutions que nous prévoyons d'explorer.

2 Contexte

2.1 Problèmes de satisfaction de contraintes (CSP)

Un *CSP* [12] est défini par un triplet (X, D, C) tel que X est un ensemble fini de variables, D est une fonction qui à chaque variable $X_i \in X$ associe un domaine $D(X_i)$, et C est un ensemble de contraintes, i.e., des relations entre variables qui restreignent l'ensemble des valeurs pouvant être affectées simultanément à ces variables.

Une *affectation*, notée $\mathcal{A} = \{<X_1, v_1>, .., <X_k, v_k>\}$, est un ensemble de couples variable/valeur tels que toutes les variables dans $\mathcal{A}$ sont différentes et chaque valeur appartient au domaine de la variable associée. Cette affectation correspond à l'affectation simultanée des valeurs $v_1, \ldots, v_k$ aux variables $X_1, \ldots, X_k$. Une affectation $\mathcal{A}$ est *partielle* s'il existe des variables de X qui ne sont pas affectées dans $\mathcal{A}$; elle est *complète* si toutes les variables sont affectées.

Le coût d'une affectation $\mathcal{A}$, dénoté par $cost(\mathcal{A})$, est défini par le nombre de contraintes qui sont violées par $\mathcal{A}$. Une *solution d'un CSP* (X, D, C) est une affectation complète de toutes les variables de X, qui satisfait toutes les contraintes dans C, c'est-à-dire, une affectation complète de coût nul.

De nombreux CSP réels, représentant des problèmes concrets, sont sur-contraints de sorte qu'il est impossible de trouver une affectation satisfaisant toutes les contraintes. Pour prendre en compte cette particularité, le cadre des CSP a été généralisé en introduisant la notion de *maxCSP* [5]. Dans ce cas, l'objectif n'est plus de trouver une solution consistante, mais de trouver une affectation qui maximise le nombre de contraintes respectées. La *solution optimale d'un maxCSP* est une affectation complète dont le coût est minimal.

2.2 Optimisation par Colonies de Fourmis (ACO)

La métaheuristique ACO [4] a été appliquée avec succès à un grand nombre de problèmes d'optimisation combinatoire tels que les problèmes de voyageurs de commerce, [3], les problèmes d'affectation quadratique ou bien les problèmes d'ordonnancement de voitures. Le principe de base est de construire des solutions d'une façon gloutonne aléatoire et de tirer parti de la connaissance acquise pour améliorer progressivement les solutions construites. Plus précisément, à chaque cycle, chaque fourmi construit une solution à partir d'une solution vide en ajoutant progressivement des *composants de solution* jusqu'à ce que la solution soit complète. A chaque itération de cette construction, le prochain composant de solution à ajouter est sélectionné relativement à une probabilité qui dépend d'un facteur phéromonal et d'un facteur heuristique :

- le facteur phéromonal reflète l'expérience de la colonie vis-à-vis de la sélection des composants. Il est défini relativement aux traces de phéromone associées aux composants de solutions. Ces traces de phéromone sont *renforcées* lorsque les composants de solutions correspondants on été sélectionnés dans de bonnes solutions. Toutefois, afin de permettre aux fourmis d'oublier progressivement les expériences passées, un mécanisme d'oubli —dual de la mémorisation— atténue les traces anciennes par *évaporation* à la fin de chaque cycle ;
- le facteur heuristique évalue l'intérêt de sélectionner les composants par rapport à la fonction d'objectif.

Ces deux facteurs sont respectivement pondérés par deux paramètres clefs de ACO : α et β. ACO intègre ainsi les deux connaissances complémentaires essentielles à la résolution efficace d'un problème : le facteur phéromonal représente la connaissance progressivement accumulée sur l'instance de problème en cours de résolution tandis que le facteur heuristique représente la connaissance sur la classe de problème dont l'instance est un cas particulier.

En plus de α et β, un algorithme ACO est également paramétré par

- le nombre de fourmis, *nbAnts*, qui détermine le nombre de solutions construites à chaque cycle ;
- le taux d'évaporation, $\rho \in]0;1[$, qui est utilisé à chaque cycle pour multiplier les traces de phéromone par $(1-\rho)$ afin de les atténuer à la fin de chaque cycle ;
- les bornes supérieures et inférieures, τ_{min} et τ_{max}, qui sont utilisées pour borner les traces de phéromone (si l'on considère le système MAX-MIN Ant System [11]).

Le cadre réactif proposé dans cet article vise à adapter α et β qui ont une très forte influence sur le processus de construction de solution.

La valeur du facteur phéromonal α est fondamentale pour articuler l'intensification et la diversification. En effet, plus α est grand, plus la recherche est intensifiée autour des composants de solution portant d'importantes traces de phéromone, i.e., autour des composants qui sont intervenus dans la construction des meilleures solutions passées. En particulier, nous avons montré dans [10] que le réglage du paramètre α doit trouver un compromis entre deux tendances. D'un coté, si l'on limite l'influence de la phéromone par une pondération faible, la qualité de la solution obtenue sera meilleure, mais il faudra plus de temps pour converger vers cette valeur. D'un autre coté, si l'on augmente l'influence du facteur phéromonal en augmentant la valeur du paramètre α, les fourmis trouvent de meilleures solutions pendant les premiers cycles, mais après quelques centaines de cycles, elles stagneront autour d'optima locaux et ne seront pas capables de trouver de meilleures solutions.

Le poids du facteur heuristique β détermine la *gloutonnerie* du processus de recherche ; ses valeurs les plus pertinentes dépendent également de l'instance à résoudre. En effet on constate que l'importance du facteur heuristique varie souvent d'une instance à l'autre. De surcroit, pour une instance donnée son importance peut évoluer pendant la recherche de la solution.

On remarque enfin que les valeur relatives de α et β sont bien sûr importantes, mais que leurs valeurs absolues le sont également —dans le cas contraire un seul paramètre aurait suffit.

Algorithm 1: Ant Solver

Input: A CSP (X, D, C) and a set of parameters $\{\alpha, \beta, \rho, \tau_{min}, \tau_{max}, nbAnts, maxCycles\}$
Output: A complete assignment for (X, D, C)

1 Initialize pheromone trails associated with (X, D, C) to τ_{max}
2 **repeat**
3 **foreach** k **in** $1..nbAnts$ **do**
4 Construct an assignment $\mathcal{A}_k$
5 Improve $\mathcal{A}_k$ by local search
6 Evaporate each pheromone trail by multiplying it by $(1-\rho)$
7 Reinforce pheromone trails of $\mathcal{A}_{best} = arg\min_{\mathcal{A}_k \in \{\mathcal{A}_1,\ldots,\mathcal{A}_{nbAnts}\}} cost(\mathcal{A}_k)$
8 **until** $cost(\mathcal{A}_i) = 0$ *for some* $i \in \{1..nbAnts\}$ **or** *maxCycles reached* ;
9 **return** the constructed assignment with the minimum cost

2.3 Résolution de maxCSP avec ACO

L'algorithme ACO considéré dans notre étude comparative s'appelle Ant Solver (AS) et est décrit dans l'algorithme 1. Nous rappelons rapidement ci-dessous son principe ; une description plus détaillée peut être trouvée dans [8, 9].

Traces de phéromone associées à un CSP (X, D, C) (ligne 1). Nous associons une trace de phéromone à chaque couple variable/valeur $\langle X_i, v\rangle$ tel que $X_i \in X$ et $v \in D(X_i)$. Intuitivement, cette trace de phéromone représente l'intérêt appris d'affecter la valeur v à la variable X_i. Ainsi qu'il est proposé dans [11], les traces de phéromone sont bornées entre τ_{min} et τ_{max}, et sont initialisées à τ_{max}.

Construction d'une affectation par une fourmi (ligne 4) : à chaque cycle (lignes 2-8), chaque fourmi construit une affectation : à partir d'une affectation vide $\mathcal{A} = \emptyset$, elle ajoute itérativement des couples variable/valeur à $\mathcal{A}$ jusqu'à ce que $\mathcal{A}$ soit complète. A chaque étape, pour choisir un couple variable/valeur, la fourmi choisit d'abord une variable X_j qui n'est pas encore affectée dans $\mathcal{A}$. Ce choix est réalisé avec l'heuristique `min-domain` : la fourmi choisit une variable qui a le plus petit nombre de valeurs consistantes vis-à-vis de l'affectation partielle $\mathcal{A}$ en cours de construction. Puis, la fourmi choisit une valeur $v \in D(X_j)$ à affecter à X_j selon la probabilité suivante :

$$p_{\mathcal{A}}(<X_j, v>) = \frac{[\tau(<X_j,v>)]^{\alpha} \cdot [\eta_{\mathcal{A}}(<X_j,v>)]^{\beta}}{\sum_{w \in D(X_j)} [\tau(<X_j,w>)]^{\alpha} \cdot [\eta_{\mathcal{A}}(<X_j,w>)]^{\beta}}$$

où

– $\tau(< X_j, v >)$ est la trace de phéromone liée à $(< X_j, v>)$,
– $\eta_{\mathcal{A}}(< X_j, v >)$ est le facteur heuristique et est inversement proportionnel au nombre de nouvelles contraintes violées par l'affectation de la valeur v à la variable X_j, c.-à-d., $\eta_{\mathcal{A}}(< X_j, v >)) = 1/(1 + cost(\mathcal{A} \cup \{<X_j, v>\}) - cost(\mathcal{A}))$,
– α et β sont les paramètres qui déterminent les poids relatifs des facteurs.

Amélioration locale des affectations (ligne 5) : lorsqu'une affectation complète a été construite par une fourmi, elle est améliorée en exécutant une recherche locale, c.-à-d., en changeant itérativement quelques affectations variable/valeur. Différentes heuristiques peuvent être employées pour choisir la variable à réparer et la nouvelle valeur à lui affecter. Pour toutes les expériences rapportées ci-après, nous avons employé l'heuristique `min-conflicts` [6] : nous choisissons aléatoirement une variable impliquée dans une contrainte violée, et nous affectons cette variable avec la valeur qui minimise le nombre de contraintes violées. De telles améliorations locales sont réitérées jusqu'à atteindre une solution localement optimale qui ne peut pas être améliorée en modifiant une affectation de variable.

Mise à jour des traces de phéromone (lignes 6-7) : lorsque chacune des fourmis de la colonie a construit une affectation, et l'a améliorée par recherche locale, la quantité de phéromone associée à chaque couple variable/valeur est mise à jour selon la méta-heuristique ACO. En premier lieu, toutes les traces de phéromone sont uniformément diminuées (ligne 6) afin de simuler la forme d'évaporation qui permet aux fourmis d'oublier progressivement les constructions passées. Puis, la phéromone est ajoutée sur chaque couple variable/valeur appartenant à la meilleure affectation du cycle, $\mathcal{A}_{best}$ (ligne 7) afin d'attirer les fourmis avec plus de force vers la zone correspondante de l'espace de recherche. La quantité de phéromone déposée est inversement proportionnelle au nombre de contraintes violées dans $\mathcal{A}_{best}$, i.e., $1/cost(\mathcal{A}_{best})$.

3 Utilisation d'ACO pour adapter dynamiquement α et β

Nous proposons d'*employer ACO pour adapter dynamiquement les valeurs* de α et de β. En particulier, nous proposons et comparons dans ce cadre deux méthodes différents. Dans le premier cas, appelé AS($\mathcal{GPL}$) et décrit dans 3.1, les valeurs de α et de β sont fixées au début de la construction d'une solution et sont *adaptées après chaque cycle*, lorsque toutes les fourmis ont construit une solution. Dans la deuxième variante, appelé AS($\mathcal{DPL}$) et décrite dans 3.2, les valeurs de α et de β sont définies pour chaque variable de sorte qu'elles *changent pendant la construction d'une solution*. Ces deux variantes sont expérimentalement évaluées dans 4.

3.1 Description de AS($\mathcal{GPL}$)

AS($\mathcal{GPL}$) (Ant Solver with Global Parameter Learning) suit essentiellement l'algorithme 1 mais intègre de nouveaux dispositifs pour adapter dynamiquement α et β. Par conséquent, α et β ne sont plus à fournir comme paramètres en entrée de l'algorithme, mais leurs valeurs sont choisies avec la méta-heuristique ACO à chaque cycle [1].

Paramétres de AS($\mathcal{GPL}$). En plus des paramètres de Ant Solver, c.-à-d., le nombre de cycles *nbCycles*, le nombre de fourmis *nbAnts*, le taux d'évaporation ρ, et les bornes minimales et maximales des traces de phéromone τ_{min} et τ_{max}, AS($\mathcal{GPL}$) est paramétré par un ensemble de nouveaux paramètres permettant d'adapter dynamiquement α et β, i.e.,
– deux ensembles de valeurs $\mathcal{I}\alpha$ et $\mathcal{I}_\beta$ qui contiennent les ensembles de valeurs qui peuvent être respectivement considérées pour l'affectation de α et β ;
– les bornes minimale et maximale de la quantité de phéromone, $\tau_{min_{\alpha\beta}}$ et $\tau_{max_{\alpha\beta}}$;
– le taux d'évaporation $\rho_{\alpha\beta}$.

Il est à noter que notre cadre réactif suppose que α et β prennent leurs valeurs dans deux ensembles discrets $\mathcal{I}_\alpha$ et $\mathcal{I}_\beta$ qui doivent être connus *a priori*. Ces deux ensembles doivent contenir de bonnes valeurs, *c.-à-d.*, celles qui permettent à Ant Solver de trouver les meilleurs résultats pour chaque exemple. Comme discuté dans la section 4, nous proposons de choisir les valeurs de $\mathcal{I}_\alpha$ et $\mathcal{I}_\beta$ en lançant Ant Solver avec différentes affectations pour α et β sur un ensemble représentatif d'instances, puis d'initialiser $\mathcal{I}_\alpha$ et $\mathcal{I}_\beta$ avec les ensembles des valeurs qui ont permis à Ant Solver de trouver les meilleurs résultats sur ces instances.

Structure de phéromone. Nous associons une trace de phéromone $\tau_\alpha(i)$ à chaque valeur $i \in \mathcal{I}_\alpha$ et une trace de phéromone $\tau_\beta(j)$ à chaque valeur $j \in \mathcal{I}_\beta$. Intuitivement, ces traces de phéromone représentent l'intérêt appris d'affecter respectivement α et β à i et à j. Pendant le processus de recherche, ces traces de phéromone sont maintenues entre les deux limites $\tau_{min_{\alpha\beta}}$ et $\tau_{max_{\alpha\beta}}$. Au début du processus de recherche, elles sont initialisées à $\tau_{max_{\alpha\beta}}$.

1. Nous avons comparé expérimentalement deux variantes de ce cadre réactif : une première variante où les valeurs sont choisies au début de chaque cycle (entre les lignes 2 et 3) de sorte que chaque fourmi considère les mêmes valeurs pendant le cycle, et une deuxième variante où les valeurs sont choisies par des fourmis avant de construire une affectation (entre les lignes 3 et 4). Les deux variantes obtiennent des résultats qui ne sont pas sensiblement différents. Par conséquent, nous considérons seulement la première variante qui est décrite dans cette section.

Choix des valeurs pour α et β. A chaque cycle, (i.e., entre les lignes 2 et 3 de l'algorithme 1), α (resp. β) est affecté en choisissant $i \in \mathcal{I}_\alpha$ (resp. $i \in \mathcal{I}_\beta$) relativement à une probabilité $p_\alpha(i)$ (resp. $p_\beta(i)$) qui est proportionelle à la quantité de phéromone déposée sur i, i.e.,

$$p_\alpha(i) = \frac{\tau_\alpha(i)}{\sum_{j \in \mathcal{I}_\alpha} \tau_\alpha(j)} \quad (\text{resp. } p_\beta(i) = \frac{\tau_\beta(i)}{\sum_{j \in \mathcal{I}_\beta} \tau_\beta(j)})$$

Mise à jour des traces de phéromone. Les traces de phéromone associées à α et β sont mises à jour à chaque cycle, entre les lignes 7 et 8 de l'algorithme 1. D'abord, chaque trace de pheromone $\tau_\alpha(i)$ (resp. $\tau_\beta(i)$) est évaporée en la multipliant par $(1 - \rho_{\alpha\beta})$. Ensuite la trace de pheromone associée à α (resp. β) est renforcée. La quantité de pheromone déposée sur $\tau_\alpha(\alpha)$ (resp. $\tau_\beta(\beta)$) est inversement proportionnelle au nombre de contraintes violées $\mathcal{A}_{best}$ par la meilleure affectation construite pendant le cycle. Ainsi, les valeurs de α et β qui ont permis aux fourmis de construire les meilleures affectations recevront les plus grandes quantité de phéromone.

3.2 Description de AS($\mathcal{DPL}$)

Le cadre réactif décrit à la section précédente adapte dynamiquement α et β à chaque cycle, mais il considère la même affectation pour toutes les décisions élémentaires d'un même cycle. Nous décrivons maintenant une autre variante réactive nommée AS($\mathcal{DPL}$) (Ant Solver with Distributed Parameter Learning). Le principe est de choisir de nouvelles valeurs pour α et β à chacune des étapes de la construction d'une solution, c.-à-d., chaque fois qu'une fourmi doit choisir une valeur pour une variable. Le but est d'ajuster l'affectation de α et de β pour chaque variable du CSP.

Paramètres de AS($\mathcal{DPL}$). Les paramètres de AS($\mathcal{DPL}$) sont les mêmes que ceux de AS($\mathcal{GPL}$).

Structure phéromonale Nous associons une trace de phéromone $\tau_\alpha(X_k, i)$ (resp. $\tau_\beta(X_k, i)$) à chaque variable $X_k \in X$ et chaque valeur $i \in \mathcal{I}_\alpha$ (resp. $i \in \mathcal{I}_\beta$). Intuitivement ces traces de pheromone représentent l'intérêt appris d'affecter la valeur i à α (resp. β) lorsqu'on choisit une valeur pour la variable X_k. Pendant le processus de recherche de solution ces traces de pheromone sont maintenues entre les deux bornes $\tau_{min_{\alpha\beta}}$ et $\tau_{max_{\alpha\beta}}$; elles sont initialisées à $\tau_{max_{\alpha\beta}}$.

Choix des valeurs pour α et β. A chaque étape de la construction d'une affectation, avant de choisir une valeur v pour une variable X_k, α (resp. β) est positionné en choisissant une valeur $i \in \mathcal{I}_\alpha$ (resp. $i \in \mathcal{I}_\beta$) en fonction de la probabilité $p_\alpha(X_k, i)$ (resp. $p_\beta(X_k, i)$) qui est proportionnelle à la quantité de phéromone associée à i pour X_k, i.e.,

$$\begin{cases} p_\alpha(X_k, i) = \dfrac{\tau_\alpha(X_k, i)}{\sum_{j \in \mathcal{I}_\alpha} \tau_\alpha(X_k, j)} \\ \\ p_\beta(X_k, i) = \dfrac{\tau_\beta(X_k, i)}{\sum_{j \in \mathcal{I}_\beta} \tau_\beta(X_k, j)}) \end{cases} \quad (1)$$

Mise à jour des traces de phéromone. Les traces de phéromone associées à α et β sont mises à jour à chaque cycle : entre les lignes 7 et 8 de l'algorithme 1. D'abord chaque trace de phéromone $\tau_\alpha(X_k, i)$ (resp. $\tau_\beta(X_k, i)$) est évaporée en la multipliant par $(1 - \rho_{\alpha\beta})$. Ensuite, une quantité de pheromone est déposée sur les traces associées aux valeurs α et β qui ont été utilisées pour construire la meilleure affectation du cycle ($\mathcal{A}_{best}$) : pour chaque variable $X_k \in X$, si α (resp. β) a été affecté à i pour choisir la valeur affectée à X_k lors de la construction de $\mathcal{A}_{best}$, alors, $\tau_\alpha(X_k, i)$ (resp. $\tau_\beta(X_k, i)$) est incrémenté de $1/cost(\mathcal{A}_{best})$.

4 Résultats expérimentaux

4.1 Instances considérées

Nous illustrons notre ACO réactif sur le benchmark de maxCSP qui a été utilisé pour la compétition [13] CSP 2006, en considérant les 686 instances binaires de maxCSP définies en extension. Parmi ces 686 instances, 641 sont résolues à l'optimal [2], à la fois par la version statique de Ant Solver et par les deux versions réactives, les temps de réponses ne diffèrant pas significativement. Nous avons donc concentré notre expérimentation de la section 4.3 sur les 25 instances les plus difficiles. Parmi ces 25 instances, nous avons sélectionné 10 instances représentatives dont les caractéristiques sont décrites dans la table 1. Nous développons les résultats expérimentaux pour toutes les instances du benchmark de la compétition lorsque nous comparons notre modèle d'ACO réactif avec les meilleurs solveurs de la compétition.

4.2 Contexte d'expérimentation

Pour régler les paramètres de Ant Solver nous l'avons fait tourner en faisant varier les paramètres sur un sous-ensemble représentatif de 100 instances choisies parmi les 686 instances de la compétition. Ces 100 instances contiennent les 25 plus difficiles. Puis nous avons sélectionné le paramétrage avec lequel Ant Solver trouve en

2. Pour la plupart de ces instances, la meilleure solution est connue, cependant pour quelques instances, la solution optimale ne l'est pas. Pour ces instances, nous avons considéré la meilleure solution connue.

Nb	Nom	X	D	C	B
1	brock-400-1	401	2	20477	378
2	brock-400-2	401	2	20414	378
3	mann-a27	379	2	1080	252
4	san-400-0.5-1	401	2	40300	392
5	rand-2-40-16-250-350-30	40	16	250	1
6	rand-2-40-25-180-500-0	40	25	180	1
7	rand-2-40-40-135-650-10	40	40	135	1
8	rand-2-40-40-135-650-22	40	40	135	1

TABLE 1 – Pour chaque instance, Nom, X, D, C, et B indiquent respectivement le nom, le nombre de variables, la taille des domaines des variables, le nombre de contraintes, et le nombre de contraintes violées par la meilleure solution trouvée lors de la compétition de 2006.

moyenne les meilleurs résultats, i.e., $\alpha = 2$, $\beta = 8$, $\rho = 0.01$, $\tau_{min} = 0.1$, $\tau_{max} = 10$, and $nbAnts = 15$. Nous avons limité le nombre de cycles à 10000, mais on constate que le nombre de cycles réellement nécessaire à l'obtention de la meilleure solution est souvent très inférieur. Dans cette section, AS(Static) se réfère à Ant Solver utilisé avec ce paramétrage.

Nous avons également réglé α et β séparément pour chaque instance (tout en conservant inchangées les valeurs des autres paramètres). Dans cette section, AS(Tuned) se réfère à Ant Solver dont les paramètres statiques sont affectés avec la meilleure valeur pour l'instance considérée.

Pour les deux variantes réactives de Ant Solver (AS($\mathcal{GPL}$) et AS($\mathcal{DPL}$)), les paramètres non appris gardent les mêmes valeurs que pour $AS(Tuned)$ et AS $(Static)$, i.e., $\rho = 0.01$, $\tau_{min} = 0.1$, $\tau_{max} = 10$, et $nbAnts = 15$.

Pour les nouveaux paramètres, qui ont été introduits pour adapter dynamiquement α et β, nous avons positionné $\mathcal{I}_\alpha$ et $\mathcal{I}_\beta$ aux valeurs qui donnent de bons résultats pour Static Ant Solver, i.e., $\mathcal{I}_\alpha = \{0, 1, 2\}$ et $\mathcal{I}_\beta = \{0, 1, 2, 3, 4, 5, 6, 7, 8\}$. Quant au taux d'évaporation, et aux bornes supérieures et inférieures des traces de phéromone, nous avons utilisé les même valeurs que pour AS(Static), i.e., $\rho_{\alpha\beta} = 0.01$, $\tau_{min_{\alpha\beta}} = 0.1$, $\tau_{max_{\alpha\beta}} = 10$.

4.3 Comparaison expérimentale de AS(Tuned), AS(Static), AS($\mathcal{GPL}$) et AS($\mathcal{DPL}$)

La table 3 indique la meilleure affectation pour α et β qui a été utilisée pour éxecuter AS(Tuned). Elle montre que cette meilleure affectation est clairement différente d'une instance à l'autre. Nous avons également observé qu'à la fin du processus de recherche de AS($\mathcal{GPL}$), les traces de phéromone utilisée pour les affecter portent des valeurs différentes d'une instance à l'autre. Ceci est particulièrement marqué pour le paramètre β, ce qui indique que la pertinence du facteur heuristique dépend de l'instance considérée.

	1	2	3	4	5	6	7	8
AS($\mathcal{DPL}$)/AS($\mathcal{GPL}$)	=	=	=	=	$>$	$>$	=	$>$
AS($\mathcal{DPL}$)/AS(Static)	=	$>$	$>$	$>$	=	$>$	=	=
AS($\mathcal{DPL}$)/AS(Tuned)	=	$>$	=	=	=	$>$	$<$	$<$
AS($\mathcal{GPL}$)/AS(Static)	=	$>$	$>$	$>$	$<$	=	$<$	=
AS($\mathcal{GPL}$)/AS(Tuned)	=	$>$	=	=	$<$	=	$<$	$<$
AS(Static)/AS(Tuned)	=	=	$<$	$<$	=	=	$<$	$<$

TABLE 2 – Résultats des tests de pertinence statistique : chaque ligne compare deux variantes X/Y et donne pour chaque instance les résultats des tests sur 50 exécutions, *i.e.*, = (resp. $<$ et $>$) si X n'est pas significativement différent de Y (resp. est significativement moins bon ou meilleur que Y).

rée.

La table 3 compare également le nombre de contraintes violées dans la meilleure solution trouvée après 10000 cycles pour les quatre variantes de Ant Solver. Comme les différences entre ces variantes sont plutôt faibles pour certaines instances, nous avons réalisé des tests de pertinence statistique. La table 2 fournit les résultats de ces tests. Elle met en évidence que les variantes réactives sont toujours au moins aussi performantes que AS(Static), excepté pour les instances 5 et 7 qui sont mieux résolues par AS(Static) que par AS($\mathcal{GPL}$). Elle nous indique également que les deux variantes réactives sont au moins au même niveau de performance que AS(Tuned), et qu'elles le surpassent même pour de nombreuses instances – en fait, toutes sauf 2 pour AS($\mathcal{DPL}$) et toutes sauf 3 pour AS($\mathcal{GPL}$)). Enfin la table indique également que AS($\mathcal{DPL}$) n'est pas significativement différent de AS($\mathcal{GPL}$) pour 5 instances, et le dépasse pour 3 instances.

La table 4 compare le nombre de cycles et le temps CPU nécessaire à l'obtention de la meilleure solution. Nous notons d'abord que la surcharge en temps de calcul induite par la mise en œuvre de notre méthode réactive n'est pas significative ; les variantes de Ant Solver mettent un temps comparable pour réaliser un cycle de calcul d'une instance à l'autre, mais également d'une variante de Ant Solver à l'autre. En particulier, AS(Static) converge souvent plus rapidement que AS(Tuned).

Pour comparer les quatre variantes de Ant Solver pendant l'ensemble de la recherche de solution, et non seulement à la fin des 10000 cycles, la figure 1 représente l'évolution du pourcentage d'exécutions qui ont trouvées la solution optimale, en fonction du nombre de cycles [3]. La figure indique que AS(Static) peut trouver la solution optimale pour plus de la moitié des exécutions avant le 2000^{ieme} cycle. Cependant, après 2000 cycles, le pourcentage des exécutions résolues à l'optimalité ne croit que

3. Comme la preuve d'optimalité n'a pas été faite pour toutes les instances, nous considérons la meilleure solution trouvée pour les solutions des instances dont l'optimalité n'a pas été prouvée.

	AS(Tuned)				AS(Static)		AS($\mathcal{GPL}$)		AS($\mathcal{DPL}$)	
Nb	#const	(sdv)	α	β	#const	(sdv)	#const	(sdv)	#const	(sdv)
1	374.84	(0.7)	1	6	374.92	(0.39)	**374.**	(1.01)	**374.**	(1.01)
2	373.12	(0.26)	1	5	374.68	(1.09)	**371.32**	(1.09)	371.48	(1.31)
3	253.88	(0.26)	1	6	254.62	(0.49)	**253.74**	(0.44)	253.96	(0.28)
4	387.2	(0.11)	1	8	388.04	(1.77)	**387.**	(0.)	**387.**	(0.)
5	**1.**	(0.)	2	8	**1.**	(0.)	1.02	(0.14)	**1.**	(0.)
6	1.02	(0.02)	2	6	1.02	(0.14)	1.04	(0.19)	**1.**	(0.)
7	**1.**	(0.)	1	6	1.12	(0.32)	1.66	(0.47)	1.48	(0.5)
8	**1.**	(0.)	1	5	1.08	(0.27)	1.12	(0.32)	1.08	(0.27)

TABLE 3 – Comparaison expérimentale des meilleures solutions trouvées. Pour chaque variante de Ant Solver considérée, chaque ligne indique le nombre de contraintes violées dans la meilleure solution (moyenne et écart-type sur 50 exécutions). Pour AS(Tuned), nous donnons également les valeurs de α et β qui ont été utilisées.

	AS(Tuned)			AS(Static)			AS($\mathcal{GPL}$)			AS($\mathcal{DPL}$)		
	cycles		time	cycles		time	cycles		time	cycles		time
Nb	avg	(sdv)	avg	avg	(sdv)	avg	avg	(sdv)	avg	avg	(sdv)	avg
1	44	(7)	4	30	(4)	2	2717	(516)	98	2501	(491)	93
2	2247	(477)	140	323	(194)	12	2668	(463)	96	3322	(415)	125
3	2193	(309)	542	1146	(335)	160	2714	(432)	399	2204	(295)	328
4	710	(213)	123	347	(174)	39	316	(38)	34	112	(14)	12
5	394	(32)	5	394	(32)	4	379	(44)	5	412	(37)	5
6	476	(19)	26	606	(23)	13	507	(49)	18	579	(23)	15
7	2436	(166)	160	1092	(152)	31	736	(126)	39	1557	(266)	52
8	1944	(120)	140	884	(65)	29	1302	(286)	66	1977	(252)	87

TABLE 4 – comparaison expérimentale du nombre de cycles (moyenne et écart-type sur 50 exécutions) et du temps CPU exprimé en secondes (moyenne sur 50 exécutions) nécessaires pour atteindre la meilleure solution.

faiblement. AS(Tuned) affiche un comportement assez différent : il nécessite plus de cycles pour converger de sorte qu'il trouve moins souvent la solution optimale au début du processus de recherche ; cependant, il présente des performances nettement supérieures à AS(Static) à la fin des 10000 cycles. Considèrons par exemple les instances 2, 3 et 8 : la table 3 nous montre que AS(Tuned) à besoin de plus de cycles que AS(Static) pour les résoudre.

La figure 1 nous indique également que AS($\mathcal{GPL}$) présente de meilleures performances que les trois autres variantes pendant les 2000 premier cycles, alors qu'après 2000 cycles AS($\mathcal{GPL}$), AS($\mathcal{DPL}$) et AS(Tuned) sont assez proches et ont toutes de meilleures performances que AS(Static). Enfin, à la fin du processus de recherche, AS($\mathcal{DPL}$) est légèrement meilleur que AS($\mathcal{GPL}$) qui lui-même est légèrement meilleur que AS(Tuned).

La figure 2 montre l'évolution du pourcentage d'exécutions qui sont proches de l'optimalité, *i.e.*, les solutions optimales ou les solutions qui violent une contrainte de plus que la solution optimale. Elle montre que AS($\mathcal{GPL}$) trouve les solutions quasi-optimales plus rapidement que AS($\mathcal{DPL}$), qui lui-même affiche une meilleure performance que les variantes statiques de Ant Solver. AS(Static) est meilleur que AS(Tuned) au début de la recherche de solution, mais il est généralement dépassé par AS(Tuned) au-delà des 1000 cycles.

4.4 Comparaison expérimentale de AS($\mathcal{DPL}$) avec les solvers de la compétition

Nous comparons maintenant AS($\mathcal{DPL}$) avec les solveurs de maxCSP de la compétition 2006. Il y avait neuf solveurs, parmi lesquels huit sont basés sur des approches complètes, et un est basé sur une méthode de recherche locale incomplète. Pour la compétition, chaque solveur a bénéficié d'une durée allant jusqu'à 40 minutes sur un 3GHz Intel Xeon (voir [13] pour plus de détails). Pour chaque exemple, nous avons comparé AS($\mathcal{DPL}$) avec le meilleur résultat trouvé pendant la compétition (par l'un des 9 solvers). Nous avons également limité AS($\mathcal{DPL}$) à 40 minutes, mais il a été exécuté sur un ordinateur moins puissant (Intel P4 Dual Core à 1.7 GHz). Nous ne rapportons pas les temps CPU car ils ont été obtenus sur différents ordinateurs. Le but ici est d'évaluer la qualité des solutions

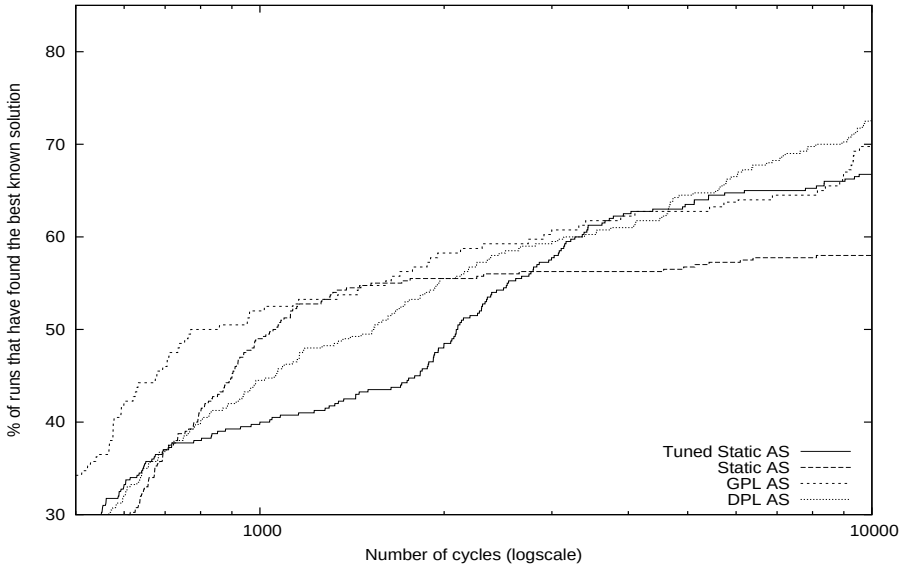

FIGURE 1 – Evolution du pourcentage d'exécutions qui ont trouvé la solution optimale par rapport au nombre de cycles.

				Competition		AS($\mathcal{DPL}$)	
Bench	#I	#C	$\sum$ best known	$\sum cost$	#I^{best}	$\sum cost$	#I^{best}
brock	4	56381	1111	1123	2	**1111**	**4**
hamming	4	14944	460	463	1	**460**	**4**
mann	2	1197	281	**281**	**2**	283	1
p-hat	3	312249	1472	1475	1	**1472**	**3**
san	3	48660	687	692	2	**687**	**3**
sanr	1	6232	182	183	0	**182**	**1**
dsjc	1	736	19	20	0	**19**	**1**
le	2	11428	2869	2925	1	**2869**	**2**
graphw	6	16993	416	420	4	**416**	**6**
scenw	27	29707	809	904	25	**809**	**27**
tightness0.5	15	2700	15	**15**	**15**	16	14
tightness0.65	15	2025	15	**15**	**15**	18	12
tightness0.8	15	1545	21	**22**	**13**	25	10
tightness0.9	15	1260	26	**30**	**11**	31	10

TABLE 5 – Comparaison expérimentale de AS($\mathcal{DPL}$) avec les solveurs de la compétition 2006. Chaque ligne indique le nom du benchmark, le nombre d'instances dans ce benchmark (#I), le nombre total de contraintes dans ces instances (#C), et le nombre de contraintes violées en considérant, pour chaque instance, le meilleur résultat trouvé ($\sum$ best known). Puis, nous donnons les meilleurs résultats obtenus pendant la compétition : pour chaque instance, nous avons considéré le meilleur résultat des 9 solveurs de la compétition et nous donnons le nombre de contraintes qui sont violées ($\sum cost$) suivi du nombre d'instances pour lesquelles la meilleure solution a été trouvée (#I ^{best}). Enfin, nous donnons les résultats obtenus avec AS($\mathcal{DPL}$) : le nombre de contraintes violées ($\sum cost$) suivi du nombre d'instances pour lesquelles la meilleure solution connue a été trouvée (#I^{bests}).

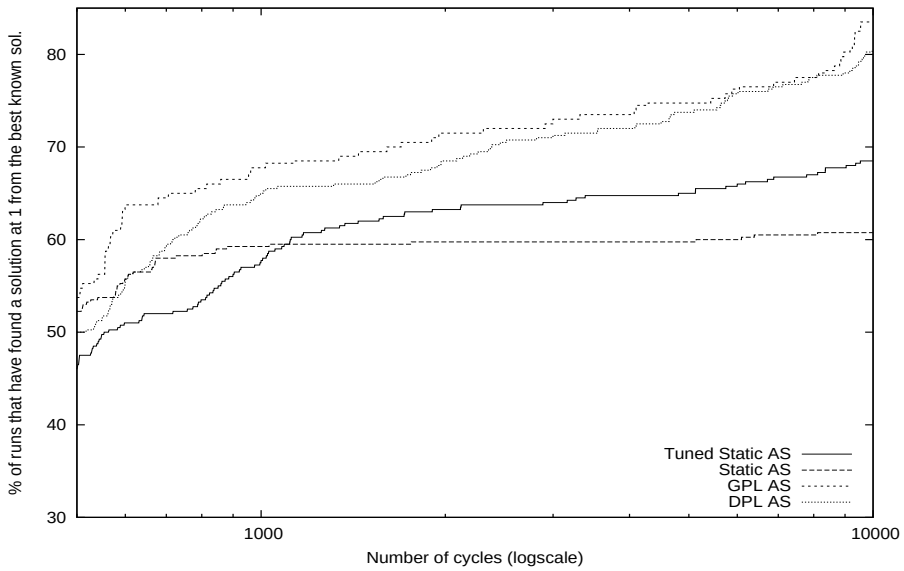

FIGURE 2 – Evolution du pourcentage d'exécutions qui ont trouvé une solution optimale, ou violant une contrainte de plus que la solution optimale, en fonction du nombre de cycles.

trouvées par AS($\mathcal{DPL}$).

Cette comparaison a été faite sur les 686 instances binaires définis en extension. Ces instances ont été regroupés dans 45 benchmarks. Parmi ces 45 benchmarks, il y avait 31 benchmarks pour lesquels AS($\mathcal{DPL}$) et les meilleurs solveurs de la compétition ont trouvé les mêmes valeurs pour chaque instance. Par conséquent, nous ne montrons les résultats que pour les 14 benchmarks pour lesquels AS($\mathcal{DPL}$) et le meilleur solveur de la compétition ont obtenu des résultats différents (pour au moins une instance du benchmarks).

La table 5 donne les résultats pour ces 14 benchmarks. Elle montre que AS($\mathcal{DPL}$) a de meilleures performances que les meilleurs solvers de la compétition pour 9 benchmarks. Plus précisément, AS($\mathcal{DPL}$) a amélioré les meilleures solutions trouvées par un solveur de la compétition pour 19 instances. Par contre, il n'a pas trouvé la meilleure solution pour 15 instances ; parmi ces 15 instances, 14 appartiennent à la série `tightness*` qui apparait clairement plus difficile pour AS($\mathcal{DPL}$) (notons cependant qu'aucun solveur de la compétition n'a été capable de trouver la solution optimale pour 6 de ces instances).

5 Conclusion

Nous avons présenté deux variantes d'un formalisme réactif pour adapter automatiquement et dynamiquement les valeurs du poids α du facteur phéromonal et du poids β du facteur heuristique qui ont une forte influence sur l'articulation intensification/diversification de la recherche dans ACO. Le but est double : nous visons en premier lieu à libérer l'utilisateur du problème délicat du réglage de ces paramètres ; nous visons également à améliorer les résultats sur des instances difficiles.

Les premiers résultats expérimentaux sont très encourageants. En effet, dans la plupart des cas notre ACO réactif atteint les performances d'une variante statique dont le paramétrage a été spécialement réglé pour chaque instance ; il les surpasse même sur quelques instances.

Travaux comparables. Il y a de nombreux travaux relatifs aux approches réactives, qui adaptent dynamiquement des paramètres pendant le processus de recherche [2]. Plusieurs de ces approches réactives ont été proposées pour des approches de recherche locale, et sont à l'origine de la notion de recherche réactive. Par exemple, Battiti et Protasi ont proposé dans [1] d'employer l'information de rééchantillonnage afin d'adapter dynamiquement la longueur de la liste taboue dans une recherche taboue.

Il existe également d'autres approches réactives pour les algorithmes ACO. En particulier, Randall a proposé dans [7] d'utiliser ACO pour adapter dynamiquement des paramètres d'un algorithme ACO. Notre approche emprunte quelques caractéristiques à ce modèle ACO réactif. Ce-

pendant nous apprenons les paramètres à un niveau différent. En effet, dans [7] les paramètres sont appris au niveau des fourmis de sorte que chaque fourmi fait évoluer ses propres paramètres et considère le même paramétrage pendant une construction de solution. Dans ce modèle, chaque fourmi est considérée comme un agent autonome qui améliore ses performances au fur et à mesure qu'il acquiert de la connaissance sur le problème. Notre approche est différente, les paramètres sont appris au niveau de la colonie, de sorte que chaque fourmi emploie les mêmes traces de phéromone pour fixer ses paramètres. Par ailleurs, nous avons étudié et comparé deux variantes : dans la première, les mêmes paramètres sont employés pendant l'ensemble de la construction d'une solution, alors que dans la seconde, les paramètres sont associés à chaque variable. Nous avons montré que cette variante améliore réellement le processus de recherche sur certains instances, mettant en évidence que, lors de la résolution de problèmes de satisfaction de contraintes, la pertinence des facteurs heuristiques et de phéromone dépendent de la variable à assigner.

Futurs développements. Nous prévoyons d'évaluer notre cadre réactif sur d'autres mises en œuvre de ACO afin de qualifier sa généricité. En particulier, il sera intéressant de comparer les variantes réactives sur d'autres problèmes : pour certains problèmes tels que le Voyageur de Commerce, il est plus probable que l'association de paramètres à chaque composant de solution ne soit pas intéressante, tandis que sur d'autres problèmes, tels que le sac à dos multidimensionnel ou les problèmes d'ordonnancement, nous conjecturons que ceci devrait améliorer le processus de recherche.

Une limite de notre cadre réactif se situe dans le fait que l'espace de recherche pour les valeurs de paramètre doit être connu à l'avance et discrétisé. Comme précisé par un relecteur, il serait préférable de résoudre ce méta-problème en tant que tel, *i.e.*, un problème d'optimisation continue. Par conséquent, un développement futur abordera cet aspect.

En conclusion, nous prévoyons d'intégrer un cadre réactif pour adapter dynamiquement les autres paramètres, ρ, τ_{min}, et τ_{max} qui ont des dépendances fortes avec α et β. Ceci pourrait être fait, par exemple, en employant des indicateurs d'intensification/diversification, tels que les taux de similarité ou de rééchantillonnage.

Références

[1] R. Battiti and M. Protasi. Reactive local search for the maximum clique problem. *Algorithmica*, 29(4) :610–637, 2001.

[2] Roberto Battiti, Mauro Brunato, and Franco Mascia. *Reactive Search and Intelligent Optimization*. Operations research/Computer Science Interfaces. Springer Verlag, 2008. in press.

[3] 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.

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

[5] E. Freuder and R. Wallace. Partial constraint satisfaction. *Artificial Intelligence*, 58 :21–70, 1992.

[6] S. Minton, M.D. Johnston, A.B. Philips, and P. Laird. Minimizing conflicts : a heuristic repair method for constraint satistaction and scheduling problems. *Artificial Intelligence*, 58 :161–205, 1992.

[7] Marcus Randall. Near parameter free ant colony optimisation. In *ANTS Workshop*, volume 3172 of *Lecture Notes in Computer Science*, pages 374–381. Springer, 2004.

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

[9] C. Solnon and D. Bridge. An ant colony optimization meta-heuristic for subset selection problems. In *System Engineering using Particle Swarm Optimization*, pages 7–29. Nova Science, 2006.

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

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

[12] E.P.K. Tsang. *Foundations of Constraint Satisfaction*. Academic Press, London, UK, 1993.

[13] Marc van Dongen, Christophe Lecoutre, and Olivier Roussel. Results of the second csp solver competition. In *Second International CSP Solver Competition*, pages 1–10, 2007.