An ACO-Based Reactive Framework for Ant Colony Optimization: First Experiments on Constraint Satisfaction Problems

Madjid Khichane, Patrick Albert, Christine Solnon

An ACO-based Reactive Framework for Ant Colony Optimization: First Experiments on Constraint Satisfaction Problems

Madjid Khichane[1,2], Patrick Albert[1], and Christine Solnon[2⋆]

[1] ILOG SA, France {mkhichane,palbert}@ilog.fr
[2] Université de Lyon
Université Lyon 1, LIRIS CNRS UMR5205, France
christine.solnon@liris.cnrs.fr

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

Ant Colony Optimization (ACO) has shown to be very effective to solve a wide range of combinatorial optimization problems [1]. However, when solving a problem (with ACO like with other metaheuristics), one usually has to find a compromise between two dual goals. On the one hand, one has to intensify the search around the most promising areas, that are usually close to the best solutions found so far. On the other hand, one has to diversify the search and favor exploration in order to discover new, and hopefully more successful, areas of the search space. The behavior of the algorithm with respect to this intensification/diversification duality (also called exploitation/exploration duality) can be influenced by modifying parameter values.

Setting parameters is a difficult problem which usually lets the user balance between two main tendencies. On the one hand, when choosing values which emphasize diversification, the quality of the final solution is often better, but the time needed to converge on this solution is also often higher. On the other hand, when choosing values which emphasize intensification, the algorithm often finds better solutions quicker, but it often converges on sub-optimal solutions. Hence, the best parameter values both depend on the instance to be solved and

⋆ This work has been partially financed by ILOG under the research collaboration contract ILOG/UCBL-LIRIS.

on the time allocated for solving the problem. Moreover, it may be better to change parameter values during the solution process, depending on the search landscape around the current state, than to keep them fixed.

To improve the search process with respect to the intensification/diversification duality, Battiti et al [2] propose to exploit the past history of the search to automatically and dynamically adapt parameter values, thus giving rise to reactive approaches.

In this paper, we introduce a reactive framework for ACO to dynamically adapt some parameters during the search process. This dynamic adaptation is done with ACO: 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. Our approach is experimentally evaluated on constraint satisfaction problems (CSPs) which basically involve finding an assignment of values to variables so that a given set of constraints is satisfied.

The paper is organized as follows. We first recall in section 2 some background on CSPs and ACO. We show in section 3 how to use ACO to dynamically adapt some parameters during the search process. In particular, we introduce two different reactive frameworks for ACO: a first framework where parameter values are fixed during the construction of a solution, and a second framework where parameters are tailored for each variable so that parameters are dynamically changed during the construction of a solution. We experimentally evaluate and compare these two reactive frameworks in section 4, and we conclude on some related work and further work in section 5.

2 Background

2.1 Constraint satisfaction problems (CSPs)

A *CSP* [3] is defined by a triple (X, D, C) such that X is a finite set of variables, D is a function that maps every variable $X_i \in X$ to its domain $D(X_i)$, that is, the finite set of values that can be assigned to X_i, and C is a set of constraints, that is, relations between some variables which restrict the set of values that can be assigned simultaneously to these variables.

An *assignment*, noted $\mathcal{A} = \{< X_1, v_1 >, \ldots, < X_k, v_k >\}$, is a set of variable/value couples such that all variables in $\mathcal{A}$ are different and every value belongs to the domain of its associated variable. This assignment corresponds to the simultaneous assignment of values $v_1, \ldots, v_k$ to variables $X_1, \ldots, X_k$, respectively. An assignment $\mathcal{A}$ is *partial* if some variables of X are not assigned in $\mathcal{A}$; it is *complete* if all variables are assigned.

The *cost* of an assignment $\mathcal{A}$, denoted by $cost(\mathcal{A})$, is defined by the number of constraints that are violated by $\mathcal{A}$. A *solution of a CSP* (X, D, C) is a complete assignment for all the variables in X, which satisfies all the constraints in C, that is, a complete assignment with zero cost.

Most real-life CSPs are over-constrained, so that no solution exists. Hence, the CSP framework has been generalized to maxCSPs [4]. In this case, the goal

is no longer to find a consistent solution, but to find a complete assignment that maximizes the number of satisfied constraints. Hence, an *optimal solution of a maxCSP* is a complete assignment with minimal cost.

2.2 Ant Colony Optimization (ACO)

ACO is a metaheuristic [1] which has been successfully applied to a wide range of combinatorial optimization problems such as, *e.g.*, travelling salesman problems [5], quadratic assignment problems [6], or car sequencing problems [7]. The basic idea is to iteratively build solutions in a greedy randomized way. More precisely, at each cycle, each ant builds a solution, starting from an empty solution, by iteratively adding solution components until the solution is complete. At each iteration of this construction, the next solution component to be added is chosen with respect to a probability which depends on two factors:

- The pheromone factor reflects the past experience of the colony regarding the selection of this component. This pheromone factor is defined with respect to pheromone trails associated with solution components. These pheromone trails are reinforced when the corresponding solution components have been selected in good solutions; they are decreased by evaporation at the end of each cycle, thus allowing ants to progressively forget older experiments.
- The heuristic factor evaluates the interest of selecting this component with respect to the objective function.

These two factors are respectively weighted by two parameters α and β.

Besides α and β, an ACO algorithm is also parameterized by

- the number of ants, $nbAnts$, which determines the number of constructed solutions at each cycle;
- the evaporation rate, $\rho \in]0;1[$, which is used at the end of each cycle to decrease all pheromone trails by multiplying them by $(1-\rho)$;
- the lower and upper pheromone bounds, τ_{min} and τ_{max}, which are used to bound pheromone trails (when considering the MAX-MIN Ant System [6]).

The reactive framework proposed in this paper focuses on α and β which have a great influence on the solution construction process.

The weight of the pheromone factor, α, is a key parameter for balancing intensification and diversification. Indeed, the greater α, the stronger the search is intensified around solutions containing components with high pheromone trails, i.e., components that have been previously used to build good solutions. In particular, we have shown in [8] that the setting of α let us balance between two main tendencies. On the one hand, when limiting the influence of pheromone with a low pheromone factor weight, the quality of the final solution is better, but the time needed to converge on this value is also higher. On the other hand, when increasing the influence of pheromone with a higher pheromone factor weight, ants find better solutions during the first cycles, but after a few hundreds or so cycles, they are no longer able to find better solutions.

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 A_k
5         Improve A_k by local search
6     Evaporate each pheromone trail by multiplying it by (1 − ρ)
7     Reinforce pheromone trails of A_best = arg min_{A_k ∈ {A_1,...,A_nbAnts}} cost(A_k)
8 until cost(A_i) = 0 for some i ∈ {1..nbAnts} or maxCycles reached ;
9 return the constructed assignment with the minimum cost
```

The weight of the heuristic factor, β, determines the greedyness of the search and its best setting also depends on the instance to be solved. Indeed, the relevancy of the heuristic factor usually varies from an instance to another. Moreover, for a given instance, the relevancy of the heuristic factor may vary during the solution construction process.

Note finally that not only the ratio between α and β matters, but also their absolute value. Let us consider for example the two following parameter settings: $p_1 = \{\alpha = 1, \beta = 2\}$ and $p_2 = \{\alpha = 2, \beta = 4\}$. In both settings, β is twice as high as α. However, p_2 emphasizes more strongly differences than p_1. Let us consider for example the case where ants have to choose between two components a and b which pheromone factors respectively are $\tau(a) = 1$ and $\tau(b) = 2$, and heuristic factors respectively are $\eta(a) = 2$ and $\eta(b) = 3$. When considering the p_1 setting, choice probabilities are

$$p(a) = \frac{1^1 \cdot 2^2}{1^1 \cdot 2^2 + 2^1 \cdot 3^2} = 0.18 \text{ and } p(b) = \frac{2^1 \cdot 3^2}{1^1 \cdot 2^2 + 2^1 \cdot 3^2} = 0.82$$

whereas when considering the p_2 setting, choice probabilities are

$$p(a) = \frac{1^2 \cdot 2^4}{1^2 \cdot 2^4 + 2^2 \cdot 3^4} = 0.05 \text{ and } p(b) = \frac{2^2 \cdot 3^4}{1^2 \cdot 2^4 + 2^2 \cdot 3^4} = 0.95$$

2.3 Solving max-CSPs with ACO

The ACO algorithm considered in our comparative study is called Ant Solver (AS) and is described in algorithm 1. We briefly describe below the main features of this algorithm; more information can be found in [9, 10].

Pheromone trails associated with a CSP (X, D, C) (line 1). We associate a pheromone trail with every variable/value couple $\langle X_i, v\rangle$ such that $X_i \in X$ and $v \in D(X_i)$. Intuitively, this pheromone trail represents the learned desirability of assigning value v to variable X_i. As proposed in [6], pheromone trails are bounded between τ_{min} and τ_{max}, and they are initialized at τ_{max}.

Construction of an assignment by an ant (line 4): At each cycle (lines 2-8), each ant constructs an assignment: starting from an empty assignment $\mathcal{A} = \emptyset$, it iteratively adds variable/value couples to $\mathcal{A}$ until $\mathcal{A}$ is complete. At each step, to select a variable/value couple, the ant first chooses a variable $X_j \in X$ that is not yet assigned in $\mathcal{A}$. This choice is performed with respect to the smallest-domain ordering heuristic, i.e., the ant selects a variable that has the smallest number of consistent values with respect to the partial assignment $\mathcal{A}$ under construction. Then, the ant chooses a value $v \in D(X_j)$ to be assigned to X_j with respect to the following probability:

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

where

- $\tau(<X_j, v>)$ is the pheromone trail associated with $<X_j, v>$,
- $\eta_{\mathcal{A}}(< X_j, v >)$ is the heuristic factor and is inversely proportional to the number of new violated constraints when assigning value v to variable X_j, i.e., $\eta_{\mathcal{A}}(<X_j, v>) = 1/(1 + cost(\mathcal{A} \cup \{<X_j, v>\}) - cost(\mathcal{A}))$,
- α and β are the parameters that determine the relative weights of the factors.

Local improvement of assignments (line 5): Once a complete assignment has been constructed by an ant, it is improved by performing some local search, i.e., by iteratively changing some variable/value assignments. Different heuristics can be used to choose the variable to be repaired and the new value to be assigned to this variable. For all experiments reported below, we have used the min-conflict heuristics [11], i.e., we randomly select a variable involved in some violated constraint, and then we assign this variable with the value that minimizes the number of constraint violations. Such local improvements are iterated until reaching a locally optimal solution which cannot be improved by modifying one variable assignment.

Pheromone trails update (lines 6-7): Once every ant has constructed an assignment, and improved it by local search, the amount of pheromone laying on each variable/value couple is updated according to the ACO metaheuristic. First, all pheromone trails are uniformly decreased (line 6) in order to simulate some kind of evaporation that allows ants to progressively forget worse constructions. Then, pheromone is added on every variable/value couple belonging to the best assignment of the cycle, $\mathcal{A}_{best}$ (line 7) in order to further attract ants towards the corresponding area of the search space. The quantity of pheromone laid is inversely proportional to the number of constraint violations in $\mathcal{A}_{best}$, i.e., $1/cost(\mathcal{A}_{best})$.

3 Using ACO to dynamically adapt α and β

We now propose to use ACO to dynamically adapt the values of α and β. In particular, we propose and compare two different reactive frameworks. In the

first framework, called AS($\mathcal{GPL}$) and described in 3.1, the setting of α and β is fixed during the construction of a solution and is adapted after each cycle, once every ant has constructed a solution. In the second framework, called AS($\mathcal{DPL}$) and described in 3.2, the setting of α and β is personalized for each variable so that it changes during the construction of a solution. These two frameworks are experimentally evaluated in 4.

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

AS($\mathcal{GPL}$) (Ant Solver with Global Parameter Learning) basically follows algorithm 1 but integrates new features for dynamically adapting α and β. Hence, α and β are no longer given as input parameters of the algorithm, but their values are chosen with respect to the ACO metaheuristic at each cycle[1].

Parameters of AS($\mathcal{GPL}$). Besides the parameters of Ant Solver, i.e., the number of cycles *nbCycles*, the number of ants *nbAnts*, the evaporation rate ρ, and the lower and upper pheromone bounds τ_{min} and τ_{max}, AS($\mathcal{GPL}$) is parameterized by a set of new parameters that are used to set α and β, i.e.,

- two sets of values $\mathcal{I}_\alpha$ and $\mathcal{I}_\beta$ which respectively contain the set of values that may be considered for setting α and β;
- a lower and an upper pheromone bound, $\tau_{min_{\alpha\beta}}$ and $\tau_{max_{\alpha\beta}}$;
- an evaporation rate $\rho_{\alpha\beta}$.

Note that our reactive framework supposes that α and β take their values within two given discrete sets of values $\mathcal{I}_\alpha$ and $\mathcal{I}_\beta$ which must be known *a priori*. These two sets should contain good values, *i.e.*, those which allow Ant Solver to find the best results for every possible instance. As discussed in Section 4, we propose to choose the values of $\mathcal{I}_\alpha$ and $\mathcal{I}_\beta$ by running Ant Solver with different settings for α and β on a representative set of instances, and by keeping in $\mathcal{I}_\alpha$ and $\mathcal{I}_\beta$ the values that allowed Ant Solver to find the best results on these instances.

Pheromone structure. We associate a pheromone trail $\tau_\alpha(i)$ with every value $i \in \mathcal{I}_\alpha$ and a pheromone trail $\tau_\beta(j)$ with every value $j \in \mathcal{I}_\beta$. Intuitively, these pheromone trails represent the learnt desirability of setting α and β to i and j respectively. During the search process, these pheromone trails are bounded between the two bounds $\tau_{min_{\alpha\beta}}$ and $\tau_{max_{\alpha\beta}}$. At the beginning of the search process, they are initialized to $\tau_{max_{\alpha\beta}}$.

[1] We have experimentally compared two variants of this reactive framework: a first variant where the values are chosen at the beginning of each cycle (between lines 2 and 3) so that every ant uses the same values during the cycle, and a second variant where the values are chosen by ants before constructing an assignment (between lines 3 and 4). The two variants obtain results that are not significantly different. Hence, we only consider the first variant which is described in this section.

Choice of values for α and β. At each cycle (i.e., between lines 2 and 3 of algorithm 1), α (resp. β) is set by choosing a value $i \in \mathcal{I}_\alpha$ (resp. $i \in \mathcal{I}_\beta$) with respect to a probability $p_\alpha(i)$ (resp. $p_\beta(i)$) which is proportional to the amount of pheromone laying on i, i.e.,

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

Pheromone trails update. The pheromone trails associated with α and β are updated at each cycle, between lines 7 and 8 of algorithm 1. First, each pheromone trail $\tau_\alpha(i)$ (resp. $\tau_\beta(i)$) is evaporated by multiplying it by $(1 - \rho_{\alpha\beta})$. Then the pheromone trail associated with α (resp. β) is reinforced. The quantity of pheromone laid on $\tau_\alpha(\alpha)$ (resp. $\tau_\beta(\beta)$) is inversely proportional to the number of constraint violations in $\mathcal{A}_{best}$, the best assignment built during the cycle. Therefore, the values of α and β that have allowed ants to build better assignments will receive more pheromone.

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

The reactive framework described in the previous section dynamically adapts α and β at every cycle, but it considers the same setting for all assignment constructions within a same cycle. We now describe another reactive framework called AS($\mathcal{DPL}$) (Ant Solver with Distributed Parameter Learning). The basic idea is to choose new values for α and β at each step of the construction of an assignment, i.e., each time an ant has to choose a value for a variable. The goal is to tailor the setting of α and β for each variable of the CSP.

Parameters of AS($\mathcal{DPL}$). The parameters of AS($\mathcal{DPL}$) are the same as the ones of AS($\mathcal{GPL}$).

Pheromone structure. We associate a pheromone trail $\tau_\alpha(X_k, i)$ with every variable $X_k \in X$ and every value $i \in \mathcal{I}_\alpha$ and a pheromone trail $\tau_\beta(X_k, j)$ with every variable $X_k \in X$ and every value $j \in \mathcal{I}_\beta$. Intuitively, these pheromone trails respectively represent the learnt desirability of setting α and β to i and j when choosing a value for variable X_k. During the search process, these pheromone trails are bounded between the two bounds $\tau_{min_{\alpha\beta}}$ and $\tau_{max_{\alpha\beta}}$. At the beginning of the search process, they are initialized to $\tau_{max_{\alpha\beta}}$.

Choice of values for α and β. At each step of the construction of an assignment, before choosing a value v for a variable X_k, α (resp. β) is set by choosing a value $i \in \mathcal{I}_\alpha$ (resp. $i \in \mathcal{I}_\beta$) with respect to a probability $p_\alpha(X_k, i)$ (resp. $p_\beta(X_k, i)$) which is proportional to the amount of pheromone laying on i for X_k, i.e.,

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

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

Table 1. For each instance, Name, X, D, C, and B respectively give the name, the number of variables, the size of the variable domains, the number of constraints, and the number of violated constraints in the best solution found during the 2006 competition.

Pheromone trails update. The pheromone trails associated with α and β are updated at each cycle, between lines 7 and 8 of algorithm 1. First, each pheromone trail $\tau_\alpha(X_k, i)$ (resp. $\tau_\beta(X_k, i)$) is evaporated by multiplying it by $(1 - \rho_{\alpha\beta})$. Then, some pheromone is laid on the pheromone trails associated with the values of α and β that have been used to build the best assignment of the cycle ($\mathcal{A}_{best}$): for each variable $X_k \in X$, if α (resp. β) has been set to i for choosing the value to assign to X_k when constructing $\mathcal{A}_{best}$, then $\tau_\alpha(X_k, i)$ (resp. $\tau_\beta(X_k, i)$) is incremented by $1/cost(\mathcal{A}_{best})$.

4 Experimental results

4.1 Test suite

We illustrate our reactive framework on a benchmark of maxCSP which has been used for the CSP 2006 competition [12]. We have considered the 686 binary maxCSP instances defined in extension. Among these 686 instances, 641 are solved to optimality[2] both by the static version of Ant Solver and the two reactive versions, whereas CPU times are not significantly different. Hence, we concentrate our experimental study of section 4.3 on the 25 harder instances that are not always solved to optimality. Among these 25 hard instances, we have chosen 10 representative ones which features are described in Table 1. We shall give more experimental results, for all instances of the benchmark of the competition, in section 4.4, when comparing our reactive ACO framework with the best solvers of the competition.

4.2 Experimental setup

We have tuned parameters for Ant Solver by running it on a representative subset of 100 instances (including the 25 hardest ones) among the 686 instances of the competition, with different parameter settings. We have selected the setting that allowed Ant Solver to find the best results on average, i.e., $\alpha = 2$, $\beta = 8$, $\rho = 0.01$,

[2] For most of these instances, the optimal solution is known. However, for a few instances optimal solutions are not known. For these instances, we have considered the best known solution.

	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 2. Experimental comparison of best found solutions. Each line gives, for each variant of Ant Solver, the number of violated constraints in the best found solution (average on 50 runs and standard deviation). For AS(Tuned), we also give the values of α and β that have been considered.

$\tau_{min} = 0.1$, $\tau_{max} = 10$, and $nbAnts = 15$. We have set the maximum number of cycles to 10000, but the number of cycles needed to converge to the best solution is often much smaller. In this section, AS(Static) refers to Ant Solver with this static parameter setting.

We also have tuned α and β for every instance separately (while keeping the other parameters to the same values). In this section, AS(Tuned) refers to Ant Solver with the best static parameter setting for the considered instance.

For the two reactive variants of Ant Solver (AS($\mathcal{GPL}$) and AS($\mathcal{DPL}$)), we have kept the same parameter setting for the "old" parameters, i.e., $\rho = 0.01$, $\tau_{min} = 0.1$, $\tau_{max} = 10$, and $nbAnts = 15$. For the new parameters, that have been introduced to dynamically adapt α and β, we have set $\mathcal{I}_\alpha$ and $\mathcal{I}_\beta$ to the set of values that gave reasonably good results with Static Ant Solver, i.e., $\mathcal{I}_\alpha = \{0, 1, 2\}$ and $\mathcal{I}_\beta = \{0, 1, 2, 3, 4, 5, 6, 7, 8\}$. For the evaporation rate and the lower and upper pheromone bounds, we have used the same values as for static AS, i.e., $\rho_{\alpha\beta} = 0.01$, $\tau_{min_{\alpha\beta}} = 0.1$, $\tau_{max_{\alpha\beta}} = 10$.

4.3 Experimental comparison of AS(Tuned), AS(Static), AS($\mathcal{GPL}$) and AS($\mathcal{DPL}$)

Table 2 gives the best setting for α and β that have been considered when running AS(Tuned). It shows us that this best setting is clearly different from one instance to another. We also noticed that, at the end of the search process of AS($\mathcal{GPL}$), pheromone trails used to set α and β have rather different values from one instance to another. This is more particularly true for β, thus showing that the relevancy of the heuristic factor depends on the considered instance.

Table 2 also compares the number of violated constraints in the best found solution after 10000 cycles, for the four variants of Ant Solver. As differences between the different variants are rather small on some instances, we have performed statistical significance tests. Table 3 gives the results of these statis-

	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 3. Results of statistical significance tests: each line compares two variants X/Y and gives for every instance the result of the test for 50 runs, *i.e.*, = (resp. < and >) if X is not significantly different from Y (resp. worse and better than Y).

	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. Experimental comparison of the number of cycles (average and standard deviation on 50 runs) and the CPU time in seconds (average on 50 runs) spent to find the best solution.

tical tests. It shows us that reactive variants are always at least as good as AS(Static), except for instances 5 and 7 which are better solved by AS(Static) than AS($\mathcal{GPL}$). It also shows us that both reactive variants are able to reach the performances of AS(Tuned), and even outperform it, on many instances (all but 2 for AS($\mathcal{DPL}$) and all but 3 for AS($\mathcal{GPL}$)). Finally, it also shows us that AS($\mathcal{DPL}$) is not significantly different from AS($\mathcal{GPL}$) for 5 instances, and outperforms it on 3 instances.

Table 4 compares the number of cycles and the CPU time spent to find the best solution. We first note that the computational overhead due to the reactive framework is not significant so that the four Ant Solver variants spend comparable CPU times for performing one cycle on one given instance. We note also that the number of cycles needed to converge is different from one instance to another, but also from one variant of Ant Solver to another. In particular, AS(Static) often converges quicker than AS(Tuned).

In order to allow us to compare the four Ant Solver variants during the whole search process, and not only at the end of the 10000 cycles, Figure 1 plots the evolution of the percentage of runs that have found the optimal solution

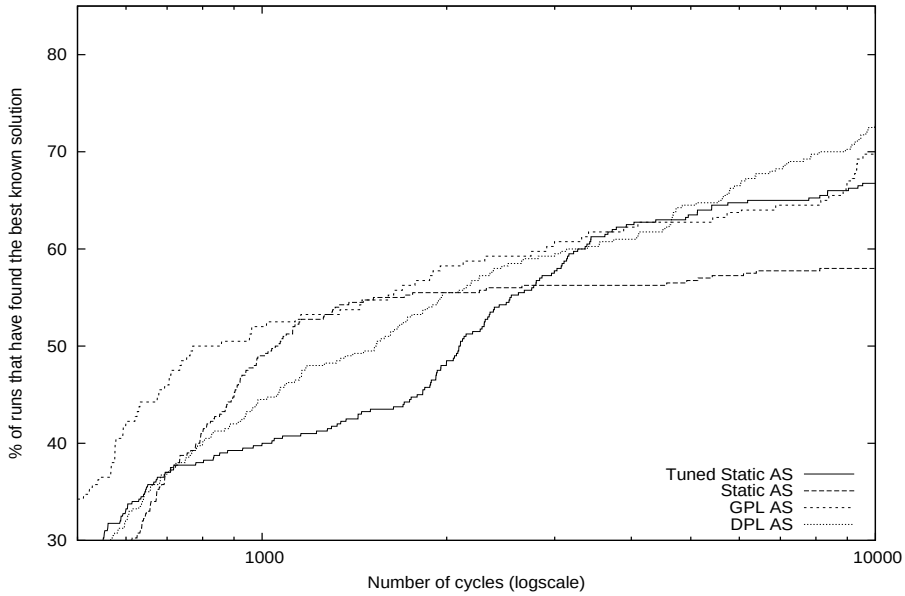

Fig. 1. Evolution of the percentage of runs that have found the optimal solution with respect to the number of cycles.

with respect to the number of cycles[3]. It shows that AS(Static) is able to solve to optimality more than half of the runs within the 2000 first cycles. However, after 2000 or so cycles, the percentage of runs solved to optimality by AS(Static) does not increase a lot. AS(Tuned) exhibits a rather different behavior: if it is able to solve to optimality less runs at the beginning of the search process, it has significantly outperformed AS(Static) at the end of the 10000 cycles. Let us consider for example instances 2, 3 and 8: Table 2 shows us that AS(Tuned) is able to find better solutions than AS(Static); however, Table 4 shows us that AS(Tuned) needs much more cycles than AS(Static) to find these solutions.

Figure 1 also shows us that AS($\mathcal{GPL}$) outperforms the three other variants during the 2000 first cycles, whereas after 2000 cycles AS($\mathcal{GPL}$), AS($\mathcal{DPL}$) and AS(Tuned) are rather close and all of them clearly outperform AS(Static). Finally, at the end of the search process AS($\mathcal{DPL}$) slightly outperforms AS($\mathcal{GPL}$) which itself slightly outperforms AS(Tuned).

Figure 2 plots the evolution of the percentage of runs that have found solutions that are close to optimality, *i.e.*, optimal solutions or solutions which violates one more constraint than the optimal solution. It shows that AS($\mathcal{GPL}$) more quickly finds nearly optimal solutions than AS($\mathcal{DPL}$), which itself is better than the static variants of Ant Solver. AS(Static) is better than AS(Tuned) at

[3] As proof of optimality has not been done for all the considered instances, we consider the best known solution for the instances which optimal solution is not known.

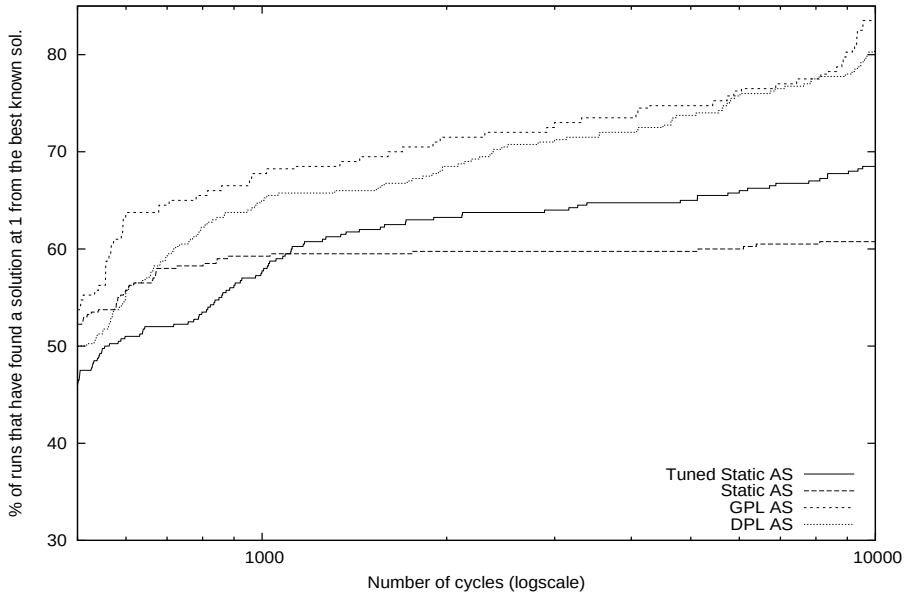

Fig. 2. Evolution of the percentage of runs that have found a solution that violates one more constraint than the optimal solution with respect to the number of cycles.

the beginning of the search process, but it is outperformed by AS(Tuned) after 1000 cycles or so.

4.4 Experimental comparison of AS($\mathcal{DPL}$) with state-of-the-art solvers

We now compare AS($\mathcal{DPL}$) with the MaxCSP solvers of the 2006 competition. There was 9 solvers, among which 8 are based on complete branch and propagate approaches, and 1 is based on incomplete local search. For the competition, each solver has been given a time limit of 40 minutes on a 3GHz Intel Xeon (see [12] for more details). For each instance, we have compared AS($\mathcal{DPL}$) with the best result found during the competition (by any of the 9 solvers). We have also limited AS($\mathcal{DPL}$) to 40 minutes, but it has been run on a less powerful computer (a 1.7 GHz P4 Intel Dual Core). We do not report CPU times as they have been obtained on different computers. The goal here is to evaluate the quality of the solutions found by AS($\mathcal{DPL}$).

This comparison has been done on the 686 binary instances defined in extension. These instances have been grouped into 45 benchmarks. Among these 45 benchmarks, there was 31 benchmarks for which AS($\mathcal{DPL}$) and the best solver of the competition have found the same values for every instance of the benchmark. Hence, we only display results for the 14 benchmarks for which AS($\mathcal{DPL}$)

				Competition		AS($\mathcal{DPL}$)	
Bench	#I	#C	$\sum_{\text{best known}}$	$\sum cost$	#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. Experimental comparison of AS($\mathcal{DPL}$) with the solvers of the 2006 competition. Each line gives the name of the benchmark, the number of instances in this benchmark (#I), the total number of constraints in these instances (#C), and the total number of violated constraints when considering, for each instance, its best known solution ($\sum_{\text{best known}}$). Then, we give the best results obtained during the competition: for each instance, we have considered the best result over the 9 solvers of the competition and we give the total number of constraints that are violated ($\sum cost$) followed by the number of instances for which the best known solution has been found (#I^{best}). Finally, we give the results obtained by AS($\mathcal{DPL}$): the total number of constraints that are violated ($\sum cost$) followed by the number of instances for which the best known solution has been found (#I^{best}).

and the best solver of the competition obtained different results (for at least one instance of the benchmark).

Table 5 gives results for these 14 benchmarks. It shows that AS($\mathcal{DPL}$) outperforms the best solvers of the competition for 9 benchmarks. More precisely, AS($\mathcal{DPL}$) has been able to improve the best solutions found by a solver of the competition for 19 instances. However, it has not found the best solution for 15 instances; among these 15 instances, 14 belong to the tightness* benchmarks which appear to be difficult ones for AS($\mathcal{DPL}$).

5 Conclusion

We have introduced two reactive frameworks for dynamically and automatically tuning the pheromone factor weight α and the heuristic factor weight β which have a strong influence on intensification/diversification of ACO searches. The goal is twofold: first, we aim at freeing the user from the unintuitive problem of tuning these parameters; second, we aim at improving the search process and reaching better performances on difficult instances.

First experimental results are very encouraging. Indeed, in most cases our reactive ACO reaches performances of a static variant, and even outperforms it on some instances.

Related work. There exists a lot of work on reactive approaches, that dynamically adapt parameters during the search process [2]. Many of these reactive approaches have been proposed for local search approaches, thus giving rise to reactive search. For example, Battiti and Protasi have proposed in [13] to use resampling information in order to dynamically adapt the length of the tabu list in a tabu search.

There also exists reactive approaches for ACO algorithms. In particular, Randall has proposed in [14] to dynamically adapt ACO parameters by using ACO and our approach borrows some features from this reactive ACO framework. However, parameters are learnt at a different level. Indeed, in [14] parameters are learnt at the ant level so that each ant evolves its own parameters and considers the same parameter setting during a solution construction. In our approach, paramaters are learnt at the colony level, so that every ant uses the same pheromone trails to set parameters. Moreover, we have compared two frameworks, a first one where the same parameters are used during a solution construction, and a second one where parameters are tailored for every variable, and we have shown that this second framework actually improves the search process on some instances, thus bringing to the show that, when solving constraint satisfaction problems, the relevancy of the heuristic and the pheromone factors depend on the variable to be assigned.

Further work. We plan to evaluate our reactive framework on other ACO algorithms in order to evaluate its genericity. In particular, it will be interesting to compare the two reactive frameworks on other problems: for some problems such as the Traveling Salesman Problem, it is most probable that tailoring parameters for every solution component is not interesting, whereas on other problems, such as the multidimensional knapsack or the car sequencing problems, we conjecture that this should improve the search process.

A limit of our reactive framework lies in the fact that the search space for the parameter values must be known in advance and discretized. As pointed out by a reviewer, it would be preferable to solve the meta-problem as what it is, *i.e.*, a continuous optimization problem. Hence, further work will address this issue.

Finally, we plan to integrate a reactive framework for dynamically adapting the other parameters, ρ, τ_{min}, and τ_{max} which have strong dependencies with α and β. This could be done, for example, by using intensification/diversification indicators, such as the similarity ratio or resampling information.

References

1. Dorigo, M., Stuetzle, T.: Ant Colony Optimization. MIT Press (2004)
2. Battiti, R., Brunato, M., Mascia, F.: Reactive Search and Intelligent Optimization. Operations research/Computer Science Interfaces. Springer Verlag (2008) in press.

3. Tsang, E.: Foundations of Constraint Satisfaction. Academic Press, London, UK (1993)
4. Freuder, E., Wallace, R.: Partial constraint satisfaction. Artificial Intelligence **58** (1992) 21–70
5. Dorigo, M., Gambardella, L.: Ant colony system: A cooperative learning approach to the traveling salesman problem. IEEE Transactions on Evolutionary Computation **1**(1) (1997) 53–66
6. Stützle, T., Hoos, H.: $\mathcal{MAX} - \mathcal{MIN}$ Ant System. Journal of Future Generation Computer Systems **16** (2000) 889–914
7. Solnon, C.: Combining two pheromone structures for solving the car sequencing problem with Ant Colony Optimization. European Journal of Operational Research (EJOR) **191** (2008) 1043–1055
8. Solnon, C., Fenet, S.: A study of aco capabilities for solving the maximum clique problem. Journal of Heuristics **12**(3) (2006) 155–180
9. Solnon, C.: Ants can solve constraint satisfaction problems. IEEE Transactions on Evolutionary Computation **6**(4) (2002) 347–357
10. Solnon, C., Bridge, D.: An ant colony optimization meta-heuristic for subset selection problems. In: System Engineering using Particle Swarm Optimization, Nova Science (2006) 7–29
11. Minton, S., Johnston, M., Philips, A., Laird, P.: Minimizing conflicts: a heuristic repair method for constraint satistaction and scheduling problems. Artificial Intelligence **58** (1992) 161–205
12. van Dongen, M., Lecoutre, C., Roussel, O.: Results of the second csp solver competition. In: Second International CSP Solver Competition. (2007) 1–10
13. Battiti, R., Protasi, M.: Reactive local search for the maximum clique problem. Algorithmica **29**(4) (2001) 610–637
14. Randall, M.: Near parameter free ant colony optimisation. In: ANTS Workshop. Volume 3172 of Lecture Notes in Computer Science., Springer (2004) 374–381