Boosting ACO with a Preprocessing Step

Christine Solnon

LISI - University Lyon 1 - 43 bd du 11 novembre, 69 622 Villeurbanne cedex, France

Abstract. When solving a combinatorial optimization problem with the Ant Colony Optimization (ACO) metaheuristic, one usually has to find a compromise between guiding or diversifying the search. Indeed, ACO uses pheromone to attract ants. When increasing the sensibility of ants to pheromone, they converge quicker towards a solution but, as a counterpart, they usually find worse solutions. In this paper, we first study the influence of ACO parameters on the exploratory ability of ants. We then study the evolution of the impact of pheromone during the solution process with respect to its cost's management. We finally propose to introduce a preprocessing step that actually favors a larger exploration of the search space at the beginning of the search at low cost. We illustrate our approach on Ant-Solver, an ACO algorithm that has been designed to solve Constraint Satisfaction Problems, and we show on random binary problems that it allows to find better solutions more than twice quicker.

1 Introduction

When solving a combinatorial optimization problem with the ACO metaheuristic [1], one usually has to find a compromise between two dual goals, i.e., guiding or diversifying the search.

Guiding the search with ACO: ACO aims at guiding the search towards "promising" states, that are close to the best solutions found so far. Indeed, it has been shown for a large number of combinatorial optimization problems that higher quality local minima tend to be closer to global minima than lower quality local minima. Actually, this property underlies the motivation of evolutionary approaches which are of little interest on problems for which the correlation between solution fitness and distance to optimal solutions is too low [2, 3].

In order to reach this guidance goal, ACO uses a stigmergetic communication mechanism: ants lay pheromone on graph edges in order to attract other ants, and the quantity of pheromone laid is proportional to the path quality so that edges participating to the best paths become more attractive. Improved performances can be obtained when introducing an elitist strategy where only the best ants deposit pheromone, as proposed in the $\mathcal{MAX}-\mathcal{MIN}$ Ant System [3].

Diversifying the search with ACO: While guiding the search, one must also favor exploration to discover new, and hopefully more successful, areas of the search space. Diversification is particularly important at the beginning of the search in order to avoid premature convergence towards local optima.

In order to reach this diversification goal, ants choose their way to go in a stochastic way —with respect to transition probabilities— so that an edge with a small quantity of pheromone can be chosen —with a small probability— even though there are other edges with higher quantities of pheromone. To emphasize exploration, [3] imposes lower and upper bounds τ_{min} and τ_{max} on pheromone trails so that for all pheromone trails τ_{ij}, $\tau_{min} \leq \tau_{ij} \leq \tau_{max}$. These bounds prevent the relative differences between pheromone trails from becoming too extreme so that the probability of choosing an edge never becomes null. Also, pheromone trails are initialized at τ_{max} at the beginning of the algorithm, thus achieving a higher exploration of the search space during the first cycles.

Influence of parameters on guidance/diversification: The behaviour of ants can be influenced by modifying parameter values [4]. In particular, diversification can be emphasized both by decreasing the value of the pheromone factor weight α —so that ants become less sensitive to pheromone trails— and by increasing the value of the pheromone persistence rate ρ —so that pheromone evaporates more slowly. When increasing the exploratory ability of ants in this way, one usually finds better solutions, but as a counterpart it takes longer time to find them. Different approaches have been proposed for finding "good" parameter values, such as [5]. However, one always has to choose between values that give priority to solution quality —by favoring diversification — or values that give priority to computation time —by favoring guidance.

Overview of the paper: The goal of the preprocessing step presented in this paper is to increase diversification at low cost, so that ants can find better solutions in much less time. The basic idea is to collect a significant number of paths without using pheromone. These collected paths constitute a kind of sampling of the search space and they are used to initialize pheromone trails before running the ACO algorithm. The paper is organized as follows: we first study the impact of ACO parameters with respect to guidance and diversification; we then study the evolution of the impact of pheromone trails during solution process, and we show that during the first cycles pheromone does not influence much on ants behaviour whereas its management is tedious and expensive; we finally describe the preprocessing step that actually favors a larger exploration of the search space at the beginning of the search at low cost. This approach is illustrated with Ant-Solver, an ACO algorithm designed to solve Constraint Satisfaction Problems (CSPs). The next section recalls some definitions and terminology on CSPs and random binary CSPs and briefly describes Ant-Solver.

2 Background

CSPs: A *CSP* [6] is defined by a triple (X, D, C) such that $X = \{X_1, ..., X_n\}$ is a finite set of n variables, D is a function which maps every variable $X_i \in X$ to its domain $D(X_i)$, i.e., the set of values that can be assigned to X_i, and C is a set of constraints, i.e., relations between variables that restrict the set of values that can be assigned simultaneously to the variables. An *assignment*, denoted by $\mathcal{A} = \{<X_1, v_1>, <X_2, v_2>, \ldots, <X_k, v_k>\}$, is a set of variable-value pairs and corresponds to the simultaneous assignment of values $v_1, v_2, \ldots, v_k$ to variables $X_1, X_2, \ldots, X_k$ respectively. The *cost* of an assignment $\mathcal{A}$ is defined by the number of violated constraints in $\mathcal{A}$. A *solution of a CSP* (X, D, C) is a complete assignment for all the variables in X that satisfies all the constraints in C, i.e., the cost of which is 0.

Random binary CSPs: Binary CSPs only have binary constraints, i.e., each constraint involves exactly two variables. Binary CSPs can be generated at random. A class of randomly generated CSPs is characterized by 4 components $< n, m, p_1, p_2 >$ where n is the number of variables, m is the uniform domain size, p_1 is a measure of the connectivity and p_2 is a measure of the tightness of the constraints. Experiments reported in this paper have been obtained with random binary CSPs generated according to model A as described in [7], i.e., p_1 is the probability of adding a constraint between two different variables, and p_2 is the probability of ruling out a pair of values between two constrained variables. As incomplete approaches cannot detect inconsistency, we report experiments performed on feasible instances only, i.e., CSPs that do have at least one solution. Instances are forced to be feasible by first randomly generating a solution.

Description of Ant-Solver: The algorithm for solving CSPs, called Ant-Solver, is based on the ACO metaheuristic [1] and is sketched below.

procedure Ant-Solver(X, D, C)
    $\tau \leftarrow$ InitializePheromoneTrails()
    repeat
        for k **in** $1..nbAnts$ **do**
            $\mathcal{A}_k \leftarrow \emptyset$
            while $\mid \mathcal{A}_k \mid < \mid X \mid$ **do**
                $X_j \leftarrow$ SelectVariable$(X, \mathcal{A}_k)$
                $v \leftarrow$ ChooseValue$(\tau, X_j, D(X_j), \mathcal{A}_k)$
                $\mathcal{A}_k \leftarrow \mathcal{A}_k \cup \{<X_j, v>\}$
            $\mathcal{A}_k \leftarrow$ ApplyLocalSearch$(\mathcal{A}_k)$
        $\tau \leftarrow$ UpdatePheromoneTrails$(\tau, \{\mathcal{A}_1, \ldots, \mathcal{A}_{nbAnts}\})$
    until $cost(\mathcal{A}_i) = 0$ for some $i \in \{1..nbAnts\}$ **or** max cycles reached

We now describe the pheromone graph on which artificial ants lay pheromone trails, and the different used functions. More details can be found in [8].

The **pheromone graph** associates a vertex with each variable-value pair $<X_i, v>$ such that $X_i \in X$ and $v \in D(X_i)$. There is an edge between any pair of vertices corresponding to two different variables. The amount of pheromone laying on an edge $(<X_i, v>, <X_j, w>)$ is noted $\tau(<X_i, v>, <X_j, w>)$. Intuitively, this trail represents the learned desirability of assigning simultaneously value v to variable X_i and value w to variable X_j. Lower and upper bounds τ_{min} and τ_{max}, such that $0 < \tau_{min} \leq \tau_{max}$, are explicitly imposed on pheromone trails.

The function "**InitializePheromoneTrails**()" initializes the amount of pheromone laying on each edge of the pheromone graph to τ_{max}.

The function "**SelectVariable**$(X, \mathcal{A}_k)$" returns a variable $X_j \in X$ that is not yet assigned in $\mathcal{A}_k$. This choice can be performed randomly, or with respect to some commonly used variable ordering. Experiments reported in this paper have been performed with the smallest-domain ordering, i.e., the function returns a variable that has the smallest number of consistant values with respect to the already assigned variables (ties are broken randomly).

The function "**ChooseValue**$(\tau, X_j, D(X_j), \mathcal{A}_k)$" returns a value $v \in D(X_j)$ to be assigned to X_j. The choice of v is done according to the ACO metaheuristic, i.e., with respect to a probability $p(v, \tau, X_j, D(X_j), \mathcal{A}_k)$ which depends on a pheromone factor $\mathcal{P}$ and a quality factor $\mathcal{Q}$:

$$p(v, \tau, X_j, D(X_j), \mathcal{A}_k) = \frac{[\mathcal{P}(\tau, \mathcal{A}_k, X_j, v)]^\alpha [\mathcal{Q}(\mathcal{A}_k, X_j, v)]^\beta}{\sum_{w \in D(X_j)} [\mathcal{P}(\tau, \mathcal{A}_k, X_j, w)]^\alpha [\mathcal{Q}(\mathcal{A}_k, X_j, w)]^\beta}$$

where α and β are two parameters which determine the relative importance of pheromone and quality factors. The pheromone factor $\mathcal{P}(\tau, \mathcal{A}_k, X_j, v)$ corresponds to the sum of all pheromone trails laid on all edges between $<X_j, v>$ and the assignments in $\mathcal{A}_k$, i.e.,

$$\mathcal{P}(\tau, \mathcal{A}_k, X_j, v) = \sum_{<X_l, m> \in \mathcal{A}_k} \tau(<X_l, m>, <X_j, v>)$$

and the quality factor $\mathcal{Q}(\mathcal{A}_k, X_j, v)$ is inversely proportional to the number of new violated constraints when assigning value v to variable X_j, i.e.,

$$\mathcal{Q}(\mathcal{A}_k, X_j, v) = 1/(1 + cost(\{<X_j, v>\} \cup \mathcal{A}_k) - cost(\mathcal{A}_k))$$

The function "**ApplyLocalSearch**$(\mathcal{A}_k)$" improves the constructed assignment $\mathcal{A}_k$ 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. Experiments reported in this paper have been performed with the min-conflict heuristic [9], i.e., for each repair, we first randomly select a conflicting variable —involved in some violated constraints— and then choose a value for this variable which minimizes the number of conflicts (ties are broken randomly).

The function "**UpdatePheromoneTrails**$(\tau, \{\mathcal{A}_1, \ldots, \mathcal{A}_{nbAnts}\})$" updates pheromone trails according to the $\mathcal{MAX} - \mathcal{MIN}$ Ant System: all pheromone trails are uniformely decreased and then pheromone is added on edges participating to the construction of the best assignment. Hence, at the end of each cycle, the quantity of pheromone laying on each edge (i, j) is updated as follows:

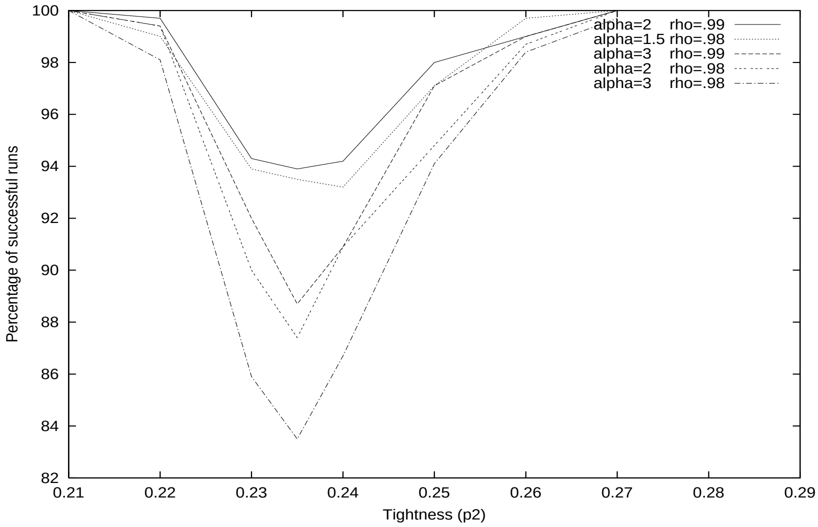
$\tau(i, j) \leftarrow \rho * \tau(i, j)$
if $i \in \mathcal{A}_{Best}$ and $j \in \mathcal{A}_{Best}$ then $\tau(i, j) \leftarrow \tau(i, j) + 1/cost(\mathcal{A}_{Best})$
if $\tau(i, j) < \tau_{min}$ then $\tau(i, j) \leftarrow \tau_{min}$
if $\tau(i, j) > \tau_{max}$ then $\tau(i, j) \leftarrow \tau_{max}$

where ρ is the trail persistence parameter such that $0 \leq \rho \leq 1$ and $\mathcal{A}_{Best}$ is the best assignment —the cost of which is minimal— of $\{\mathcal{A}_1, \ldots, \mathcal{A}_{nbAnts}\}$. One should remark that all possible pairs $(i, j) \in \mathcal{A}^2_{Best}$ are rewarded.

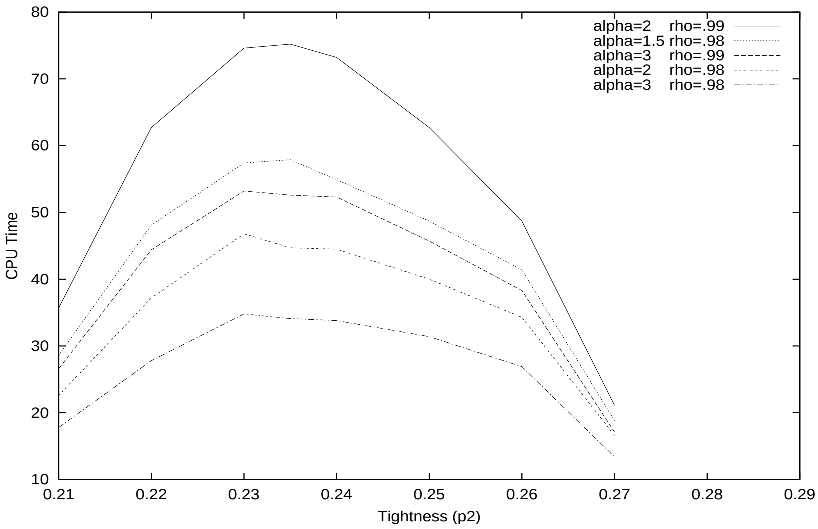
3 Impact of α and ρ on the ants behaviour

α determines the weight of pheromone factors when computing transition probabilities: when increasing α, ants become more sensitive to pheromone. ρ determines trail persistence: a value close to 0 implies a high evaporation so that ants have a "shorter-term" memory and recent experiments are emphasized with respect to older ones. The influence of these two parameters is illustrated below on $< 100, 8, 0.14, p_2 >$ random binary CSPs (for each tightness value p_2, we display results obtained on 300 different feasible problem instances; the other parameters have been setted to $\beta = 10$ and $nbAnts = 8$).

- Influence on the success rate (within a limit of 1000 cycles):



- Influence on the CPU time spent to find a solution (average results on the successful runs only):

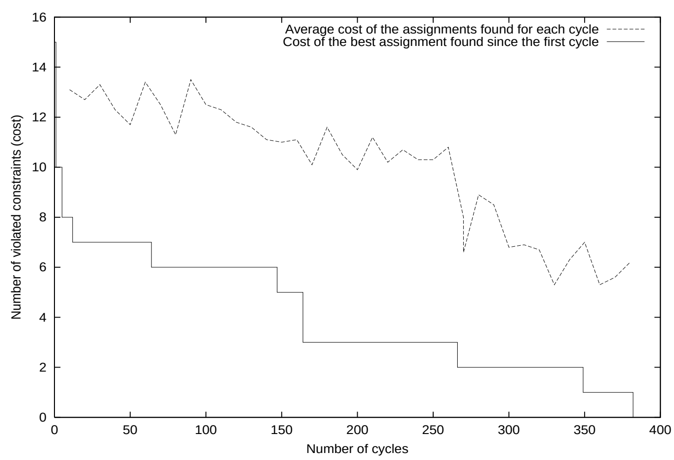


Hence, when considering the success rate criterion, the best results are obtained with low values of α and high values of ρ such as $\alpha = 2$ and $\rho = 0.99$. However, in this case the convergence is rather slow so that the CPU time spent to find a solution is more than twice more than when $\alpha = 3$ and $\rho = 0.98$. Moreover, with even smaller values of α and greater values of ρ —i.e., when $\alpha \leq 1.5$ and $\rho \geq 0.99$— ants converge so slowly that the success rate, within a limit of 1000 cycles decreases —e.g., less than 80% for $< 100, 8, 0.14, 0.23 >$ instances when $\alpha = 1.5$ and $\rho = 0.99$.

As a conclusion, when setting α and ρ one can either choose values that favor a quicker convergence —such as $\alpha = 3$ and $\rho = 0.98$— or values that slow down the convergence —such as $\alpha = 2$ and $\rho = 0.99$. In the first case, Ant-Solver will quickly find rather good assignments, eventhough it may not find the best one. In the last case, Ant-Solver will need more time to find good assignments, but it will more often succeed in actually finding the best one. In the rest of the paper, we have set α to 2 and ρ to 0.99 in order to give priority to the success rate. The next section shows that with such values pheromone only starts to influence ants after a significant number of cycles so that the convergence is rather slow.

4 Impact versus cost of pheromone

In order to favor a larger exploration, and more particularly at the beginning of the search, Ant-Solver derives its features from the $\mathcal{MAX} - \mathcal{MIN}$ Ant System, i.e., pheromone trails are limited to an interval $[\tau_{min}, \tau_{max}]$ and are initialized to τ_{max}. Hence, after n cycles of Ant-Solver, the quantity of pheromone laying on an edge (i, j) is such that $\rho^n * \tau_{max} \leq \tau(i, j) \leq \tau_{max}$ where ρ is the pheromone persistence parameter and usually has a value very close to 1. As a consequence, during the first cycles, pheromone factors nearly all have the same values and do not significantly influence the computation of transition probabilities. This is illustrated below on a typical example of execution of Ant-Solver on a $<100, 8, 0.14, 0.23>$ random binary instance (with α=2 and ρ=0.99).



During a first step, roughly corresponding to the first 100 cycles, the average cost of the constructed assignments is rather stable: in average, around 12 constraints are violated in each constructed assignment. This first step corresponds to a learning step, where the search space is widely explored and pheromone information is collected. Then, in a second step, pheromone information gathered during the first 100 cycles starts to influence the collective behaviour of ants, so that they become able to find better paths: the average number of violated constraints decreases from 12 at cycle 100 to 6 at cycle 350 while the cost of the best assignment found decreases from 6 at cycle 100 to 0 at cycle 382.

However, if the influence of pheromone is deliberately reduced at the beginning of the search so that pheromone information only becomes effective after some 100 or so cycles, managing pheromone is tedious and expensive in time. Indeed, let us consider a CSP (X, D, C), and let $n = \mid X \mid$ be the number of variables and $p = \sum_{X_i \in X} \mid D(X_i) \mid$ be the number of vertices of the associated graph of pheromone. To construct a complete assignment, the computation of pheromone factors of all transition probabilities will require $\mathcal{O}(n * p)$ operations. Then, at the end of each cycle, pheromone laying will require $\mathcal{O}(n^2)$ operations for each rewarded assignment and pheromone evaporation will require $\mathcal{O}(p^2)$ operations. The other operations, that do not deal with pheromone, are the selection of variables, the computation of quality factors and the local repair phase. The complexity of these operations mainly depends on the kind of constraints considered. However, when choosing appropriate data structures that allow incremental computations, the complexity of these operations is usually lower than $\mathcal{O}(p^2)$. Actually, when n and p increase, most of the computation time is used to manage pheromone. For example, on $< 100, 8, 0.14, 0.23 >$ CSPs, assignments are constructed nearly 3 times faster when pheromone is not used.

One should remark that the cost of managing pheromone is more important for Ant-Solver than for many other ACO algorithms. Let us consider for example Ant System described in [10] and designed to solve Traveling Salesman Problems. If the pheromone graph has n vertices, then the computation of all pheromone factors for constructing a complete path will require $\mathcal{O}(n)$ operations; pheromone laying will also require $\mathcal{O}(n)$ operations for each rewarded path and pheromone evaporation will require $\mathcal{O}(n^2)$ operations. However, for large instances with thousands of cities, pheromone management can also become expansive. In this case, a solution can be to deliberately reduce the set of candidate vertices to the closest cities.

5 Description of the preprocessing step

The introduction of a preprocessing step is motivated by the fact that pheromone is rather expensive to manage whereas it actually allows to improve the collective behaviour only after some 100 or so cycles. The idea is to collect a significant number of local minima by performing "classical" local search, i.e., by iteratively constructing complete assignments —without using pheromone— and repairing them. These assignments constitute a kind of sampling of the search space. Then, we select from this sample set the n_{Best} best local minima and use them to initialize pheromone trails. Finally, we continue the search guided by pheromone trails with Ant-Solver.

We shall call *SampleSet* the set of all local minima collected during the preprocessing step, and $BestOf(n_{Best}, SampleSet)$ the n_{Best} best local minima in *SampleSet*.

To determine the number of local minima that should be collected in *SampleSet*, one has to find a tradeoff between computing a large number of local minima, that are more representative of the search space, and a smaller number of local minima, that are more quickly computed. Actually, experiments showed us that the harder a problem is, the more local minima should be computed. Therefore, instead of computing a fixed number of local minima within *SampleSet*, we only fix n_{Best} —the number of local minima that will be selected from *SampleSet* to initialize pheromone trails— and we introduce a limit rate ϵ on the quality improvement of these best local minima, i.e., we stop collecting new local minima in *SampleSet* when the average cost of $BestOf(n_{Best}, SampleSet)$ has not been improved by more than ϵ% for a while. The preprocessing procedure is more precisely described below:

procedure preprocessing
  compute n_{Best} local minima and store them in *SampleSet*
  repeat
    $OldCost \leftarrow \sum_{\mathcal{A} \in BestOf(n_{Best}, SampleSet)} cost(\mathcal{A})$
    compute n_{Best} more local minima and add them to *SampleSet*
    $NewCost \leftarrow \sum_{\mathcal{A} \in BestOf(n_{Best}, SampleSet)} cost(\mathcal{A})$
  until $NewCost/OldCost > 1 - \epsilon$ **or** a solution has been found

In this procedure, local minima are computed by performing local search on complete assignments that are constructed like in Ant-Solver, i.e., by iteratively selecting a variable to be assigned and then choosing a value for this variable. However, the function "ChooseValue" is modified so that values are chosen with respect to quality factors only, i.e., the probability of choosing a value v for a variable X_j is $p(v, \tau, X_j, D(X_j), \mathcal{A}_k) = \mathcal{Q}(\mathcal{A}_k, X_j, v)^\beta / \sum_{w \in D(X_j)} \mathcal{Q}(\mathcal{A}_k, X_j, w)^\beta$

Finally, at the end of the preprocessing step, if no solution has been found, the set $BestOf(n_{Best}, SampleSet)$ —that contains the n_{Best} best local minima of *SampleSet*— is used to initialize pheromone trails: the quantity of pheromone laying on each edge (i, j) of the pheromone graph is set to

$$\tau(i,j) \leftarrow \sum_{\mathcal{A}_k \in BestOf(n_{Best}, SampleSet)} \Delta\tau(\mathcal{A}_k, i, j)$$

where $\Delta\tau(\mathcal{A}_k, i, j)$ is the quantity of pheromone deposited on edge (i, j) for the complete assignment $\mathcal{A}_k$ and is defined as follows:

$$\begin{array}{ll} \Delta\tau(\mathcal{A}_k, i, j) = 1/cost(\mathcal{A}_k) & \text{if } i \in \mathcal{A}_k \text{ and } j \in \mathcal{A}_k \\ \Delta_\tau(\mathcal{A}_k, i, j) = 0 & \text{otherwise} \end{array}$$

One should remark that if a pheromone trail is initialized to 0 (resp. to a high value greater than τ_{max}) then the updatePheromoneTrails function will bound it at the end of the first cycle to τ_{min} (resp. τ_{max}).

6 Experimental results

The preprocessing step is parameterized by n_{Best}, the number of assignments that are extracted from the sample set to initialize pheromone trails, and ϵ, the limit rate on the quality improvement. The influence of these two parameters has been experimentally studied on random binary CSPs. With respect to CPU times, the best results are obtained with the highest values of ϵ and the smallest values of n_{Best}. With respect to success rates, the best results are usually obtained with the smallest values of ϵ and the largest values of n_{Best}. However, we have noticed that, when $n_{Best} \geq 300$, success rates usually decrease. A good tradeoff between CPU time and success rate appears when n_{Best}=200 and ϵ=2%.

The table below displays experimental results on $< 100, 8, 0.14, p_2 >$ random binary CSPs, with n_{Best}=200, ϵ=2%, α=2, β=10, ρ=0.99 and $nbAnts$=8.

p_2	Success rate		Nb of assignments		CPU Time	
	AS with P	AS	AS with P	AS	AS with P	AS
0.21	$52.7 + 47.3 = 100.0$	100.0	$1354 + 153 = 1507$	1145	$10.6 + 4.7 = 15.3$	35.7
0.22	$8.6 + 90.9 = 99.5$	99.7	$1356 + 506 = 1862$	1984	$11.5 + 15.4 = 26.9$	62.7
0.23	$0.4 + 95.2 = 95.6$	94.3	$1603 + 664 = 2267$	2288	$15.4 + 20.3 = 35.7$	74.6
0.24	$0.7 + 94.6 = 95.3$	94.2	$1610 + 746 = 2356$	2237	$15.2 + 20.2 = 35.4$	73.2
0.25	$2.9 + 95.3 = 98.2$	98.0	$1486 + 343 = 1829$	1880	$15.3 + 10.6 = 25.9$	62.7
0.26	$10.6 + 88.7 = 99.3$	99.0	$1392 + 147 = 1539$	1432	$14.8 + 4.9 = 19.7$	48.7
0.27	$69.1 + 30.9 = 100.0$	100.0	$747 + 9 = 756$	614	$8.2 + 0.4 = 8.6$	21.1

For each value of p_2, we have considered 300 different feasible instances and the table successively displays results obtained by Ant-Solver with preprocessing (AS with P) and Ant-Solver without preprocessing (AS), with respect to 3 criteria: the success rate, the number of constructed assignments (average on successful runs only) and the CPU time spent to find a solution (average on successful runs only). For the results obtained with preprocessing, we successively display the part due to the preprocessing step, the part due to the solution process after preprocessing and the total (both for preprocessing and solving).

In this table, one can remark that success rates are slightly improved by preprocessing (except when p_2=0.22). One can also note that the total number of constructed assignments is comparable whereas the CPU time is always twice as small when using preprocessing. Let us consider for example $< 100, 8, 0.14, 0.23 >$ instances. In average, the number of constructed assignments is nearly the same (2267 and 2288). However, for AS with P, 1603 of these assignments are constructed during preprocessing, without using pheromone, so that they are rather quickly computed (in 15.2s). Then, after preprocessing, solutions are found in 20.3s in average, after the construction of 664 assignments, corresponding to 83 cycles of Ant-Solver. Hence, the total time spent to find a solution is 35.7s. As a comparison, when running Ant-Solver without preprocessing, solutions are found in 74.6s in average, after the construction of 2288 assignments, corresponding to 286 cycles of Ant-Solver. Finally, one can also note that if nearly half of the easiest instances are solved during preprocessing, only very few of the hardest instances have been solved during preprocessing.

This shows that random restart of local search with the min-conflict heuristic gives very low performances on hard problems and that ACO actually boosts local search in this case.

As a conclusion, these experimental results show that the preprocessing step introduced in this paper actually boosts Ant-Solver so that it can find solutions twice as fast. This acceleration of the convergence is not performed to the detriment of the exploratory behaviour of ants —as it is usually the case when one emphasizes the influence of pheromone by increasing α or decreasing ρ. Actually, it also allows Ant-Solver to find a few more solutions.

Another possibility to improve the convergence behaviour could be to reduce ρ so that pheromone evaporates quicker and convergence is boosted, but at the same time to introduce occasional pheromone re-initialization. Such an approach, that uses data mining technics to extract information from pheromone matrices, has been proposed in [11] and actually allowed us to boost ACO when solving permutation Constraint Satisfaction Problems (the goal of which is to find an hamiltonian path in a graph that satisfies a given set of constraints). Further works will investigate similar approaches on Ant-Solver, and compare them with preprocessing.

References

1. M. Dorigo and G. Di Caro. the Ant Colony Optimization Meta-Heuristic. In *New Ideas in Optimization*, McGraw Hill, pages 245–260, 1999.
2. P. Merz and B. Freisleben. Fitness landscapes and memetic algorithm design. In *New Ideas in Optimization*, McGraw Hill, pages 245–260, 1999.
3. T. Stutzle and H.H. Hoos. MAX-MIN Ant System. In *Journal of Future Generation Computer Systems*, 16:889–914, 2000.
4. A. Colorni, M. Dorigo and V. Maniezzo. An investigation of some properties of an "Ant algorithm". In *PPSN'92*, pages 509–520, Elsevier, 1992.
5. H. M. Botee and E. Bonabeau. Evolving Ant Colony Optimization. In *Advanced complex systems*, 1:149–159, 1998.
6. E.P.K. Tsang. Foundations of Constraint Satisfaction. *Academic Press*, 1993.
7. E. MacIntyre, P. Prosser, B. Smith and T. Walsh. Random Constraints Satisfaction: theory meets practice. In *CP'98*, volume 1520 of LNCS, Springer-Verlag, pages 325–339, 1998.
8. C. Solnon. Ants can solve Constraint Satisfaction Problems. *Research report*, 2001.
9. S. Minton, M.D. Johnston, A.B. Philips and P. Laird. Minimizing Conflicts: a Heuristic Repair Method for Constraint Satistaction and Scheduling Problems. In *Artificial Intelligence*, 58:161–205, 1992.
10. M. Dorigo, V. Maniezzo and A. Colorni. The Ant System: Optimization by a Colony of Cooperating Agents. In *IEEE Transactions on Systems, Man, and Cybernetics-Part B*, 26(1):29–41, 1996.
11. A. Stuber and C. Solnon. Boosting ACO algorithms with data mining technics (in french). In *JNPC'01*, pages 271–282, 2001.