Feasibility and Availability based Heuristics for ACO algorithms solving Binary CSP

Nicolás Rojas-Morales, María-Cristina Riff, Bertrand Neveu

Feasibility and Availability based Heuristics for ACO algorithms solving Binary CSP

Nicolás Rojas-Morales
Universidad Técnica Federico Santa María
Valparaíso, Chile
nicolasrojas@acm.org

María-Cristina Riff
Universidad Técnica Federico Santa María
Valparaíso, Chile
maria.cristina.riff@gmail.com

Bertrand Neveu
LIGM, Ecole des Ponts Paristech
Paris, France
bertrand.neveu@enpc.fr

***Abstract*—A Constraint Satisfaction Problem is composed by a set of variables, their related domains and a set of constraints among the variables that must be satisfied. These are known as hard problems to be solved. Many algorithms have been proposed to solve these problems. Metaheuristics and in particular ant-based algorithms have been used to solve difficult instances. In this paper, we propose new heuristics to be included in an ant-based algorithm in order to improve its performance when tackling hard constraint satisfaction problems. These heuristics are focused on the availability of consistent variable values and to restrict the ants collaborative information to the feasibility. To evaluate these heuristics we used the well-known Ant Solver algorithm and tested with problem instances from the transition phase. Results show that using our heuristics the Ants algorithm increases the number of problems that it is able to solve. Finally, a statistical analysis is presented to compare these approaches.**

***Index Terms*—Heuristics, Ant Colony Optimization, Constraint Satisfaction Problems**

I. Introduction

A Constraint Satisfaction Problem (CSP) is a triple (X, D, C) where X is the set of variables $X = \{X_1, \ldots, X_n\}$, their related domains $D = \{D_1, \ldots, D_n\}$ and C is a set of constraints among variables. The goal is to find a complete instantiation that satisfies all the constraints. Several real-world, industrial and research problems can be represented as a CSP [1]. Many algorithms and meta-heuristics have been proposed to tackle scheduling, planning, energy optimization, routing optimization problems during the last decades [2]. Due to the high complexity that these problems can reach, the meta-heuristic community continues proposing new strategies to solve CSPs more successfully.

In this paper we are interested in ant-based algorithms for solving binary CSPs. Ant Colony Optimization (ACO) [3] is a meta-heuristic that represents the analogy with the organization and communication of colonies of real ants. Ants deposit a chemical substance named pheromone to collaborate and communicate to the colony a path from their nest to a food source. In ACO, these artificial ants walk through a graph $G = (N, L)$ where nodes N are represented by solution components that are connected by arcs in L. In general, pheromone is deposited in paths that are most promising to be visited by the colony. To decide which node will be visited next, pheromone information and heuristic knowledge are considered. Heuristic knowledge measures candidate nodes considering specific information about the problem being tackled. There exist ant-based algorithms in literature proposed for solving CSP [4]–[8]. In general, these approaches differ in the representation and pheromone management. On the other hand, the heuristic knowledge is mostly defined to measure the effect of candidate assignments in terms of the number of conflicts.

The focus of this paper is to introduce heuristics that can improve the search process of ant-based algorithms for solving CSPs. In this work we propose heuristics that discriminate between different candidate assignments, considering information about the feasibility and the availability of variable values. The feasibility is considered in terms of minimizing the number of conflicts and the availability is related to choosing conflicts "easy" to be repaired (in constraints with few forbidden pairs of values).

The idea is to give priority to assignments that can maintain the feasibility throughout the construction process of a partial instantiation. To evaluate these heuristics we used the well-known Ant Solver (AS) [9], an ant-based algorithm proposed for solving Constraint Satisfaction Problems. Our approach will modify the heuristic knowledge and the pheromone management of Ant Solver. Section III introduces details of Ant Solver and Section IV explains our heuristics and their inclusion in AS. Moreover, experiments considering binary CSP instances from the transition phase region were carried out to evaluate these new heuristics. Results, boxplots and statistical analysis are presented in Section V.

The main contributions of this work are:

- New heuristics based on the feasibility and availability of values during the construction process of solutions and,
- Evaluate the inclusion of these heuristics in a well-known ant-based algorithm for solving CSP.

It is important to mention that our objective is not to

First author is supported by CONICYT-PCHA/National Doctoral Scholarship/2015-21150696. Second author is supported by FONDECYT Project No. 1151456 and partially supported by Centro Científico Tecnologico de Valparaíso (CCTVal) No. FB0821

propose the best algorithm for CSP. The idea is to evaluate the inclusion of two heuristics that consider information about the feasibility and the availability of variable values in an ant-based algorithm.

First, the next section presents details of some proposed ant-based algorithms for solving CSPs.

II. Related Work

In this section, we present different ant-based algorithms proposed to solve CSPs. The idea is to compare the main components of our approach with the already existing algorithms.

ACON is an ant-based algorithm with a transition rule and a pheromone management that considers information from low quality solutions [10]. Here, an alternative pheromone matrix is used to obtain information about the worst quality complete instantiations of each interation. This information is used in a transition rule to decrease the attraction to arcs that are related with these unpromising candidates. *ACON* also considers a typical pheromone matrix where best ant deposits pheromone in each iteration. On the other hand, heuristic knowledge represents the effect of a candidate assignment in terms of its produced conflicts. Experiments were made using random binary CSP instances generated with model A and N Queens instances considering different number of queens ranging from 4 to 700. Here, *ACON* solved all the N Queen instances, but only solved CSP instances with the lowest κ values (near to $0.70 - 0.80$).

A particular pheromone management was proposed in [11] for the *ACOU* algorithm. In *ACOU*, newly explored arcs are reinforced during the pheromone update. The idea is to force the algorithm to increase the exploration during the search process. Also, the ant that constructs the best quality solution will deposit pheromone in each iteration. Is not reported how heuristic knowledge is considered in *ACOU*. To evaluate the performance of *ACOU*, model A randomly generated CSP instances are also used. Here, *ACOU* only solved instances with the lowest κ values (near to $0.70-0.80$).

Some works have reported some implementation problems with the graph-based representation for Constraint Satisfaction Problems. *N-Queen ACO* [12] was proposed for solving the n queens problem only for values of n less than 8. Here, authors reported that the size of the used graph-based representation produced resource consuming problems. In other case, as it is reported in [9], specialized algorithms can solve the N-Queens problem than Ant Solver, by using global constraints. More specifically, in problems where global constraints should be considered.

A different graph-based representation for CSP is proposed in [13]. Each node groups different variables that are related by some particular constraint. This strategy produces a reduction of the number of nodes and arcs for CSP problems. To manage the pheromone it is necessary to select which pheromone values will represent arcs in this new structure. For this, an Oblivion Rate is defined to discard pheromones with worst values. This strategy was included in an ant-based algorithm (similar to Ant Solver [9] but without a local search procedure). To evaluate this strategy, N-Queens instances were used considering different number of queens ranging from 25 to 200. Results show that the proposed model was close to obtain optimal solutions for these instances.

With the objective of improving the quality of the solutions found by Ant Solver, Opposite-Inspired Learning strategies were proposed in [14]. Authors proposed to divide the search process of Ant Solver into two steps: a *First Step* to obtain information about assignations related to poor quality solutions and a *Second Step* to solve the problem of interest using this knowledge. Binary CSP instances were used to evaluate the inclusion of two strategies. Results showed that the inclusion of opposite information allowed Ant Solver to reach better quality solutions.

In summary, the main differences among these approaches are related to how the pheromone is managed. On the other hand, it is important to notice that the main structure of these algorithms is similar and also, heuristic knowledge is typically defined to measure the effect of each assignment in terms of the number of conflicts. The novelty of our approach is related to how the heuristic knowledge is defined and how the pheromone is deposited only on feasible arcs.

In the following section we present the baseline algorithm considered in this work called Ant Solver (AS). Ant Solver contains a local search procedure based on *min-conflicts* that improves considerably its performance. However, some works on the literature compare their approaches with Ant Solver without including its local search procedure. In this work, to perform fair comparisons, we consider the complete design of Ant Solver. Moreover, to the best of authors knowledge, Ant Solver is the only ant-based available code for solving CSP's.[1]

III. Ant Solver

This section introduces *Ant Solver* (AS) [9], a well-known $\mathcal{MAX} - \mathcal{MIN}$ Ant System [15] algorithm proposed to solve CSPs. Ant Solver searches for a solution that minimizes the number of unsatisfied constraints. Algorithm 1 shows the structure of Ant Solver. At each step, each ant k constructs a complete instantiation for the CSP (line 5). Unlike classical ant algorithms, when applied to CSP, the construction of a solution considers two decisions at each assignment: the selection of the next variable, according to some given criterion, and the selection of the value, made probabilistically according to the *transition rule*. The pheromone is deposited on a binary graph whose nodes $\langle X_i, v\rangle$ represent the assignment of value v to variable X_i and the edges between two nodes $(\langle X_i, v\rangle, \langle X_j, w\rangle)$ represent those simultaneous assignments of values.

[1] http://www.aco-metaheuristic.org/aco-code/public-software.html

Algorithm 1 Ant Solver

Input: Define parameters values
Output: Overall best solution reached
1: Pre-processing($nBest$,ϵ)
2: InitializePheromoneTrails()
3: **while** $F(\mathbf{I_C}) > 0$ or $maxChecks$ not reached **do**
4: **for** $k = 1 \rightarrow nbAnts$ **do**
5: $\mathbf{I_C^k} \leftarrow$ ConstructCompleteInstantiation()
6: Post-processing($\mathbf{I_C^k}$);
7: **end for**
8: UpdatePheromoneTrails($\mathbf{I_C^{LBest}}$);
9: **end while**
10: **return** $\mathbf{I_C^{GBest}}$

Equation 1 shows the transition rule used by Ant Solver. Here, it does not only depend on local relations between the candidate node and the last visited node, but also on a global relation between the candidate node and the whole set of visited nodes I_p. Hence, the pheromone factor of node $\langle X_j, v\rangle$ depends on pheromone deposited on all edges between $\langle X_j, v\rangle$ and the nodes in I_p.

$$p_{I_p}(\langle X_j, v\rangle) = \frac{[\tau_{I_p}(\langle X_j, v\rangle)]^\alpha * [\eta_{I_p}(\langle X_j, v\rangle)]^\beta}{\sum_{w \in D(X_j)} [\tau_{I_p}(\langle X_j, w\rangle)]^\alpha * [\eta_{I_p}(\langle X_j, w\rangle)]^\beta} \quad (1)$$

The heuristic knowledge in Ant Solver is defined to measure the number of possible new violated constraints for each candidate assignment. This factor also depends on the whole set of currently visited nodes in I_p:

$$\eta_{I_p}(\langle X_j, v\rangle) = \frac{1}{1 + F(\langle X_j, v\rangle \cup I_p) - F(I_p)} \quad (2)$$

where F is the evaluation function of the algorithm that counts the number of conflicts of a partial or complete instantiation. Ants that find the best quality solutions in each iteration are allowed to deposit pheromone. Pheromone is deposited on each pair of assignments in a complete instantiation (line 8). Equation (3) shows the amount of pheromone (Δ_τ) deposited on an edge (i, j) belonging to the best quality complete instantiation of an iteration (I_C^{LBest}).

$$\Delta_\tau(I_C^{LBest}, i, j) = \frac{1}{F(\mathbf{I_C^{LBest}})} \quad (3)$$

Then, the global pheromone updating rule is defined as:

$$\tau_{ij}^{new} = (1 - \rho) * \tau_{ij}^{old} + \Delta_\tau(I_C^{LBest}, i, j) \quad (4)$$

where τ_{ij}^{new} is the updated pheromone value for edge (i, j), τ_{ij}^{old} is the current pheromone value for edge (i, j) and ρ is the decreasing rate used for pheromone evaporation. Ant Solver includes *pre-* and *post-* processing steps that use a *min-conflicts* based local search procedure. The pre-processing phase repeatedly performs a local search to collect information that will be used to initialize pheromone trails (lines 1-2). The post-processing phase performs a local search after each ant has constructed a complete instantiation (line 6).

TABLE I
EXAMPLE OF POSSIBLE ASSIGNMENTS, CONSIDERING DIFFERENT VALUES FOR VARIABLE X_j

Assignment	Conflict with	#Conflicts	# Incompatible values
$\langle X_j, w\rangle$	-	0	-
$\langle X_j, y\rangle$	$\langle X_r, a\rangle$	1	12
$\langle X_j, z\rangle$	$\langle X_s, b\rangle$	1	20

Finally, Ant Solver has seven parameters: α, β, ρ, τ_{max}, $nbAnts$, $nbBest$ and ϵ. α and β determine, respectively, the weight of the pheromone and the weight of the heuristic knowledge in the computation of transition probabilities, ρ represents the level of pheromone evaporation, τ_{max} determines the maximum amount of pheromone allowed in pheromone matrix, $nbAnts$ corresponds to the number of ants used, ϵ determines the quality improvement rate for the pre-processing step and $nbBest$ corresponds to the number of best assignments selected during the pre-processing phase.

IV. OUR APPROACH

In general, heuristic knowledge in ant-based algorithms for Constraint Satisfaction Problems is defined as presented in (2). The consequences of making a variable assignment are measured only in terms of the number of unsatisfied constraints. In some cases, this function can deliver poor information and lead the algorithm to low quality complete instantiations.

We will prefer instantiations easier to be repaired, i.e. where conflicts appear in constraints with a low conflict rate. Without loss of generality, we will assume in the rest of the paper, that the variables have the same domain size, and that the constraint difficulty to be repaired can be measured by the number of pairs of values in conflict.

Let X_j be the next variable to be instantiated in a partial instantiation I_P by Ant Solver and $I_P = \{\langle X_r, a\rangle, \langle X_s, b\rangle\}$. The next possible values to be assigned are w, y and z. Table I shows for each possible assignment: the assignments in I_P with which the candidate assignment is in conflict, the number of conflicts produced and the number of pairs of values in conflict.

The assignment $\langle X_j, w\rangle$ does not have conflicts with I_P. As no conflict can be produced, the heuristic knowledge in Ant Solver will be maximized by choosing this assignment. On the other hand, assigning values y or z to X_j will produce one conflict with $\langle X_r, a\rangle$ or $\langle X_s, b\rangle$, respectively. In this case, the heuristic knowledge in (2) cannot discriminate between these two candidate assignments. If value z is assigned to X_j and if this complete instantiation has the highest quality value of the iteration, all edges in the complete instantiation containing $\langle X_j, z\rangle$ will be marked with pheromone. Then, in next iterations, ants in the colony will consider these edges as

promising although they are related with possible conflicts.

We propose two heuristics that consider this local information about the feasibility and availability of variable values during the construction process. Let $\mathcal{U}(\langle X_j, w\rangle, \langle X_r, a\rangle)$ the number of pairs of values in conflict between two variables X_j and X_r. As shown in Table I, $\mathcal{U}(\langle X_j, w\rangle, \langle X_r, a\rangle) = 12$ and $\mathcal{U}(\langle X_j, w\rangle, \langle X_s, b\rangle) = 20$. Notice that if no conflict exists between $\langle X_j, w\rangle$ and $\langle X_r, a\rangle$ the value is zero. We define a function $\mathcal{E}$ that considers the number of incompatible pairs of values between a candidate assignment and all assignments in a partial instantiation:

$$\mathcal{E}(\langle X_j, w\rangle, I_P) = \sum_{\langle X_r, a\rangle \in I_P} \mathcal{U}(\langle X_j, w\rangle, \langle X_r, a\rangle) \quad (5)$$

where $\langle X_j, w\rangle$ is the candidate assignment, $\langle X_r, a\rangle$ is an assignment in I_P that has a conflict with this candidate assignment. It is important to remark that the amount $\mathcal{U}(\langle X_j, w\rangle, \langle X_r, a\rangle)$ will contribute in the summation if there exist a conflict between $\langle X_j, w\rangle$ and $\langle X_r, a\rangle$.

We propose to include this information in the heuristic knowledge and in the pheromone management of an ant-based algorithm. The idea is to guide the construction process to feasible candidate solutions that can also be easier to be repaired by a local search procedure. For this, the heuristic knowledge and pheromone management of Ant Solver were replaced to evaluate these ideas. In the following we present the details of the implementation.

A. Focused Ant Solver

We will present in this part, how to implement these ideas in an ant-based algorithm for solving Constraint Satisfaction Problems. For this, we selected the well-known Ant Solver (AS) [9]. As Ant Solver has a local search component, it is expected that the complete instantiations constructed will be easier to be repaired considering the information about availability and feasibility of variable values. We propose to use this information in the main components of AS: heuristic knowledge and pheromone management.

1) Heuristic Knowledge: We propose to replace the heuristic knowledge of Ant Solver by:

$$\eta(\langle X_j, w\rangle) = \eta_{I_p}(\langle X_j, w\rangle) + \frac{1}{(1 + \mathcal{E}(\langle X_j, w\rangle, I_P))^2} \quad (6)$$

where $\langle X_j, w\rangle$ is a candidate assignment, $\eta_{I_p}(\langle X_j, w\rangle)$ is the heuristic knowledge in (2) and $\mathcal{E}(\langle X_j, w\rangle, I_P)$ is the function defined in (5). The objective is to give priority to assignments that have a low number of conflicts and also, conflicts with a low number of incompatible pairs of values. In order to better discriminate between scenarios with same number of conflicts the $(1 + \mathcal{E}(\langle X_j, w\rangle, I_P))$ term is squared.

2) Pheromone Management: About the pheromone management, we want to mark solutions that have a low number of conflicts but also that can be easier to be repaired during the search process. For this reason, pheromone will be deposited on the complete instantiation I_C that has the lowest number of conflicts and also the lowest pairs of values in conflict. Moreover, to guide the construction process to prefer feasible edges, pheromone will be deposited only in feasible edges of I_C.

To calculate the number of pairs of values in conflict in a complete instantiation, $\mathcal{E}$ function is redefined considering each pair of variables in conflict:

$$\mathcal{E}(I_C) = \sum_{(\langle X_j, w\rangle, \langle X_r, a\rangle) \in I_C} \mathcal{U}(\langle X_j, w\rangle, \langle X_r, a\rangle) \quad (7)$$

where I_C is a complete instantiation, X_j and X_r are two variables in conflict because the simultaneous instantiation $(\langle X_j, w\rangle, \langle X_r, a\rangle)$ is considered in I_C. Equation 3 is replaced by:

$$\Delta_\tau(I_C^{\mathcal{E}}, (\langle X_j, w\rangle, \langle X_r, a\rangle)) = \frac{1}{F(I_C^{\mathcal{E}})} \quad (8)$$

where $I_C^{\mathcal{E}}$ is the complete instantiation that has the lowest $\mathcal{E}$ function value during the current iteration, $(\langle X_j, w\rangle, \langle X_r, a\rangle)$ is a non conflict pair of assignments and $F(I_C^{\mathcal{E}})$ is the quality of $I_C^{\mathcal{E}}$.

These two new heuristics were included in Ant Solver. We named this new approach Focused Ant Solver (FAS) and in the following section we present an experimental evaluation and comparison of both approaches.[2]

V. Experiments

This section presents the evaluation of our approach and a comparison with the selected baseline algorithm Ant Solver. For this, we considered a set of 100 binary CSP instances from the transition phase, where the most difficult instances are found [16]. In these experiments the stopping criterion is defined as 4×10^9 conflict checks. We defined this amount of resources considering the average conflict checks consumed by the pre- and post-processing phases of Ant Solver. The hardware platform adopted for all these experiments was a Power Edge R630 server with 2 Intel(R) Xeon(R) CPU E5-2680 v3 @ 2.50GHz, 128 GB of RAM under Ubuntu x64 16.10 distribution.

A. Instances

For our experiments, we have considered random binary CSPs. Random binary CSPs can be generated considering four parameters $\langle n, m, p_1, p_2\rangle$:

- n defines the number of variables,
- m corresponds to the number of possible values in the domain of each variable,

[2]The code of FAS can be requested in http://ecco.informaticae.org

TABLE II
κ VALUES FOR THE SELECTED p_2 VALUES

p_2	κ
0.22	0.828
0.23	0.871
0.24	0.915
0.25	0.959
0.26	1.003
0.27	1.049
0.28	1.095
0.29	1.141
0.30	1.189
0.31	1.237

- $p_1 \in [0, 1]$ represents the connectivity, i. e. it determines the number of constraints and
- $p_2 \in [0, 1]$ corresponds to the tightness of the constraints, i. e. the probability that a pair of values is incompatible.

The CSP instances for our experiments were generated using model A. Models differ in how the constraint graph is generated and how incompatible values are chosen. For more details about this method refer to [16].

For our experiments we generated hard to solve instances belonging to the phase transition region. To predict the phase transition, a constrainedness measure (κ) [17] can be calculated as it follows:

$$\kappa(n, m, p_1, p_2) = \frac{n-1}{2} * p_1 * \log_m(\frac{1}{1-p_2}) \qquad (9)$$

where $\kappa \in [0, \infty[$. When $\kappa \approx 0$, problems are under-constrained and soluble and when $\kappa \approx \infty$, problems are over-constrained and insoluble. However, when $\kappa \approx 1$ problems are between solubility and insolubility and is difficult to find a solution or to prove there are none. For our experiments we considered $n = 100$ variables, $m = 8$ domain sizes, $p_1 = 0.14$ and p_2 ranging from 0.22 to 0.31. Ten instances were generated for each category of p_2. Table II shows the values of κ for the different values of p_2. Finally, in order to assure that each instance is solvable, we have added a random generated solution to each one.

B. Parameter Tuning

Parameter values for Ant Solver and Focused Ants were defined using a tuner algorithm called Evolutionary Calibrator (EVOCA) [18]. As we already mentioned, Ant Solver performs a pre-processing step to initialize the pheromone matrix. During this step some instances from $p_2 = [0.22, 0.23, 0.29, 0.30, 0.31]$ categories can be solved. In order to correctly evaluate parameter configurations, instances only from $p_2 = [0.24, 0.25, 0.26]$ categories were used during the tuning process. Table III shows the obtained values by the tuning process.

C. Results

As we already mentioned, 100 instances were considered for these experiments. For each instance, 20 independent runs were executed. To show the effect of the parameter tuning process, we will compare Ant Solver (AS) with the parameter values proposed in [9]. These parameter values are $\alpha = 2$, $\beta = 8$, $\tau_{max} = 4$, $nbAnts = 8$ and $\rho = 0.99$. Moreover, we considered Ant Solver tuned (AS-T) and Focused Ant Solver tuned (FAS) as it was already described. To compare these algorithms we defined a success rate (SR) as:

$$SR = 100 * \frac{SuccessRuns}{TotalRuns} \qquad (10)$$

where $SuccessRuns$ is the number of independent runs where a complete instantiation without conflicts was obtained and $TotalRuns$ is the total number of independent runs executed. Table IV shows the success rates obtained by each algorithm in all the categories. Results in bold show that an algorithm obtains the highest success rate for a category. When a tie occurs, no success rate is highlighted. First, results show the performance improvement obtained by using the parameter values obtained by EVOCA for Ant Solver. This can be seen in 8 of the 10 categories (with p_2 value ranging from 0.23 to 0.30). As a result of this improvement, in the following sections we only will compare FAS with AS-T. Considering AS-T and FAS algorithms, results show that FAS outperforms AS-T in five categories (with p_2 values of 0.23, 0.24, 0.25, 0.27 and 0.28) considering an improvement in the success rate ranging from 5 and 18 percent. Also, AS-T and FAS obtain similar results in the remaining categories.

Table V shows the average execution time in seconds per category, considering the 10 instances and the 20 independent runs. Lower average execution times were obtained in five categories (with p_2 value ranging from 0.23 to 0.27). The time difference in these categories is between 12 and 22 seconds.

Finally, the three algorithms solved all the instances in

TABLE III
PARAMETER VALUES OBTAINED BY EVOCA

Algorithm	$nbAnts$	α	β	ρ	τ_{max}
Ant Solver	9	15.0	4.5	0.0001	17.0000
Focused Ants	46	15.5	14.4	0.0005	26.3931

TABLE IV
SUCCESS RATE OBTAINED BY AS, AS-T AND FAS CONSIDERING DIFFERENT p_2 CATEGORIES WITH $(100, 8, 0.14, p_2)$

p_2	**AS**	**AS-T**	**FAS**
0.22	100,0	100,0	100,0
0.23	63,5	84,0	**93,5**
0.24	35,5	59,5	**73,5**
0.25	13,0	47,5	**65,5**
0.26	36,0	64,0	**65,0**
0.27	50,0	58,0	**65,5**
0.28	80,0	93,5	**99,0**
0.29	99,5	**100,0**	**100,0**
0.30	97,5	**99,0**	98,0
0.31	100,0	100,0	100,0

TABLE V
AVERAGE EXECUTION TIME IN SECONDS AMONG THE 20 INDEPENDENT RUNS AND THE 10 INSTANCES PER CATEGORY (p_2)

p_2	**AS-T**	**FAS**
0.22	3.468	2.881
0.23	35.082	21.188
0.24	53.327	36.798
0.25	66.843	44.443
0.26	53.622	36.364
0.27	48.468	35.723
0.28	21.572	15.890
0.29	3.344	3.189
0.30	4.508	4.541
0.31	0.311	0.340

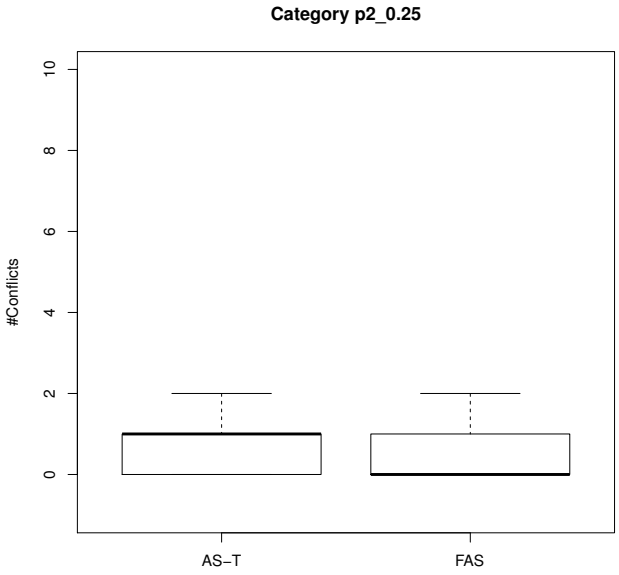

Fig. 1. Boxplot for each algorithm considering the number of conflicts obtained in each independent run in Category 0.25

categories 0.22 and 0.31, considering all the independent runs. This situation confirms that the pre-processing phase can solve most of the problems of these categories.

In general, these results show that the inclusion of these two heuristics produces an improvement of the performance of Ant Solver, in terms of the quality of the solutions obtained. For a further comparison between these algorithms, boxplots and a statistical analysis are presented in the following sections.

D. Boxplots

To analyze the distributional characteristics of the quality of the obtained solutions by both algorithms, we present boxplots of some categories. As in some categories the performance of both algorithms is similar, we only present boxplots for categories where p_2 is: 0.25, 0.26 and 0.27.

Fig.1 shows boxplots for $p_2 = 0.25$ category. Here, both boxes are short and both upper whiskers scores 2 conflicts.

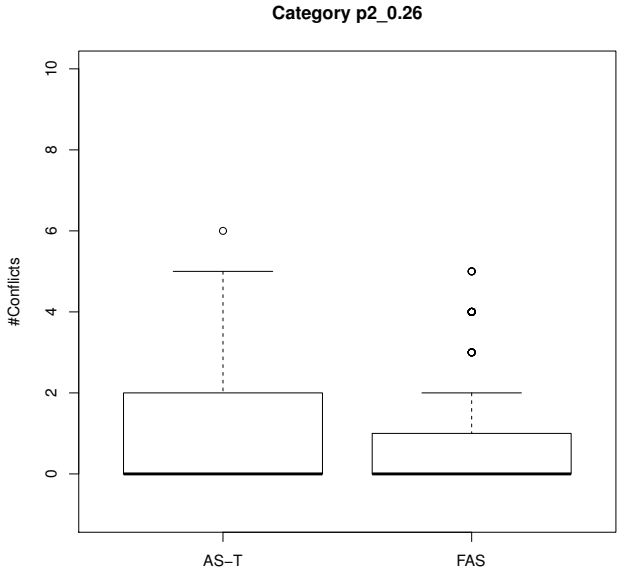

Fig. 2. Boxplot for each algorithm considering the number of conflicts obtained in each independent run in Category 0.26

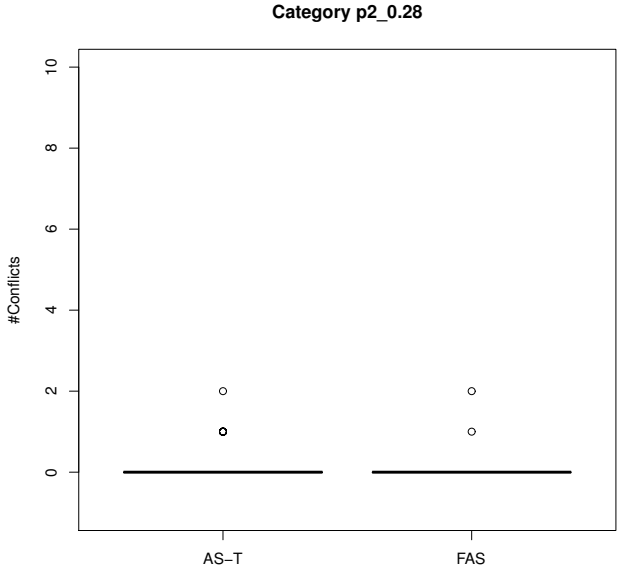

Fig. 3. Boxplot for each algorithm considering the number of conflicts obtained in each independent run in Category 0.28

Moreover, the median of AS-T is near to 1 conflict and the median of FAS is in 0.

Fig.2 shows boxplots for $p_2 = 0.26$ category. In this category, the box of Ant Solver algorithm is taller than FAS's box. The upper quartile of AS-T scores 2 conflicts and its upper whisker scores 5 conflicts. About FAS box, the upper quartile scores 1 conflict and its upper whisker scores 2

TABLE VI
WILCOXON TEST RESULTS. FOR THE COMPARISON THE NUMBER OF POSITIVE RANKS (PR), NEGATIVE RANKS (NR), TIES, TOTAL COMPARISONS AND P-VALUE ARE SHOWED.

Comparison	PR	NR	Ties	Total	p-value
AS-T - FAS	**218**	90	1692	2000	0.00

conflicts. Finally, both algorithms have the same median of 0 conflicts.

Fig.3 shows boxplots for $p_2 = 0.28$ category. Here, for both algorithms, same median and same size of each box can be seen.

E. Statistical Analysis

We have used the pair-wise Wilcoxon non-parametric test [19] to compare the performance of AS-T and FAS. For this, we used the results obtained in all the independent runs obtained by each algorithm on every instance. The hypotheses considered in this test are:

- H_0: Algorithm X found same quality solutions than Algorithm Y
- H_1: Algorithm X found different quality solutions than Algorithm Y

Table VI shows the results of the test. As the solving process of CSP instances tries to find an instantiation minimizing the number of unsatisfied constraints it is a minimization process. The number of positive (resp. negative) ranks show the cases when FAS outperforms AS-T (resp. AS-T outperforms FAS).

Considering a confidence level of 95%, these results show that the search process of AS-T is improved using the proposed heuristics. Moreover, results shows that AS-T and FAS are statistically different algorithms. All these computations were done using the statistical software package $PSPP$.[3]

VI. CONCLUSIONS AND FUTURE WORK

We have proposed two heuristics for ant-based algorithms designed for solving Constraint Satisfaction Problems. These heuristics are based on the feasibility and availability of variable values. The objective is to discriminate between assignments that can lead the construction process to feasible complete instantiations. This information was included in the heuristic knowledge and in the pheromone management of Ant Solver, a well-known ant-based algorithm proposed for solving CSPs. Moreover, pheromone was deposited only in feasible arcs of complete instantiations with the idea of reinforce only paths that could be useful to visit in future iterations.

Results show that the inclusion of these heuristics in Ant Solver improves its search process, allowing the algorithm to obtain non conflict complete instantiations in more cases. As the value of κ tends towards one, the gap between AS-T and FAS decreases. However, there is still an improvement in most categories. Boxplots were also presented to detail how the obtained solutions are distributed. Finally, statistical analysis showed that these algorithms are statistically different and also, that Focused Ant Solver outperforms Ant Solver.

As in this work we only considered CSP problem instances with same size domains, for future work we are also interested in evaluate these heuristics in solving problem instances with different size domains. Also, we are interested in implementing and evaluating these heuristics in other combinatorial problems to improve the search process of meta-heuristics.

On the other hand, we are interested in including an Opposite-Inspired Learning strategy into Focused Ant Solver to improve its search process [20].

[3]Available in http://www.gnu.org/software/pspp/

ACKNOWLEDGMENT

We want to thank Christine Solnon for providing us with the code of Ant Solver [9].

REFERENCES

[1] V. Kumar, "Algorithms for constraint-satisfaction problems: A survey," *AI magazine*, vol. 13, no. 1, p. 32, 1992.
[2] A. E. Eiben and Z. Ruttkay, "Constraint satisfaction problems," 1997.
[3] M. Dorigo, "Optimization, learning and natural algorithms," *Ph. D. Thesis, Politecnico di Milano, Italy*, 1992.
[4] A. Roli, C. Blum, and M. Dorigo, "Aco for maximal constraint satisfaction problems," 2001.
[5] L. Schoofs and B. Naudts, "Ant colonies are good at solving constraint satisfaction problems," in *Proceedings of the 2000 Congress on Evolutionary Computation. CEC00 (Cat. No.00TH8512)*, vol. 2, 2000, pp. 1190–1195.
[6] M. Afshar, "Partially constrained ant colony optimization algorithm for the solution of constrained optimization problems: Application to storm water network design," *Advances in Water Resources*, vol. 30, no. 4, pp. 954 – 965, 2007.
[7] G.-F. Deng and W.-T. Lin, "Ant colony optimization-based algorithm for airline crew scheduling problem," *Expert Systems with Applications*, vol. 38, no. 5, pp. 5787 – 5793, 2011.
[8] M. Khichane, P. Albert, and C. Solnon, "An ACO-Based Reactive Framework for Ant Colony Optimization: First Experiments on Constraint Satisfaction Problems," in *Learning and Intelligent Optimization*, T. Stützle, Ed. Springer Berlin Heidelberg, 2009, pp. 119–133.
[9] C. Solnon, "Ants can solve constraint satisfaction problems," *IEEE Trans. Evolutionary Computation*, vol. 6, no. 4, pp. 347–357, 2002.
[10] K. Ye, C. Zhang, J. Ning, and X. Liu, "Ant-colony algorithm with a strengthened negative-feedback mechanism for constraint-satisfaction problems," *Information Sciences*, vol. 406, pp. 29–41, 2017.
[11] Q. Zhang and C. Zhang, "An improved ant colony optimization algorithm with strengthened pheromone updating mechanism for constraint satisfaction problem," *Neural Computing and Applications*, 2017.
[12] S. Khan, M. Bilal, M. Sharif, M. Sajid, and R. Baig, "Solution of N-Queen problem using ACO," in *Multitopic Conference, 2009. INMIC 2009. IEEE 13th International*. IEEE, 2009, pp. 1–5.
[13] A. González-Pardo and D. Camacho, "A new CSP graph-based representation for ant colony optimization," in *Proceedings of the IEEE Congress on Evolutionary Computation, CEC 2013, Cancun, Mexico, June 20-23, 2013*. IEEE, 2013, pp. 689–696.
[14] N. Rojas-Morales, M.-C. Riff, and E. Montero, "Ants can learn from the opposite," in *Proceedings of the Genetic and Evolutionary Computation Conference 2016*, ser. GECCO '16. ACM, 2016, pp. 389–396.
[15] T. Stützle and H. Hoos, "MAX–MIN Ant System," *Future Generation Computer Systems*, vol. 16, no. 8, pp. 889–914, 2000.

[16] I. P. Gent, E. Macintyre, P. Prosser, B. M. Smith, and T. Walsh, "Random constraint satisfaction: Flaws and structure," *Constraints*, vol. 6, no. 4, pp. 345–372, 2001.

[17] E. MacIntyre, P. Prosser, B. Smith, and T. Walsh, "Random constraint satisfaction: Theory meets practice," in *Principles and Practice of Constraint Programming - CP98, 4th International Conference, Pisa, Italy, October 26-30, 1998, Proceedings*, ser. Lecture Notes in Computer Science, M. Maher and J.-P. Puget, Eds., vol. 1520. Springer, 1998, pp. 325–339.

[18] E. Montero and M.-C. Riff, "A new algorithm for reducing metaheuristic design effort," in *Proceedings of the IEEE Congress on Evolutionary Computation, CEC 2013, Cancun, Mexico, June 20-23, 2013*. IEEE, 2013, pp. 3283–3290.

[19] T. Bartz-Beielstein and M. Preuss, "Experimental research in evolutionary computation," in *Genetic and Evolutionary Computation Conference, GECCO 2007, Proceedings, London, England, UK, July 7-11, 2007, Companion Material*, D. Thierens, Ed. ACM, 2007, pp. 3001–3020.

[20] N. Rojas-Morales, M.-C. Riff, and E. Montero, "A survey and classification of opposition-based metaheuristics," *Computers & Industrial Engineering*, vol. 110, pp. 424–435, 2017.