Ants can Learn from the Opposite

Nicolás Rojas-Morales[*]
nicolasrojas@acm.org

María-Cristina Riff R.[†]
mcriff@inf.utfsm.cl

Elizabeth Montero[‡]
emontero@inf.utfsm.cl

Universidad Técnica Federico Santa María
Avenida España 1680
Valparaíso, Chile

ABSTRACT

In this work we present different learning strategies focused on detecting candidate solutions that are not interesting to be explored by a metaheuristic, in terms of evaluation function. We include a first step before the metaheuristic. The information obtained from this step is given to the metaheuristic, for visiting candidate solutions that are more promising in terms of their quality. The goal of using these strategies is to learn about candidate solutions that can be discarded from the search space, and thus to improve the search of the metaheuristic. We present two new strategies that differ on how the solutions can be constructed in an opposite way. Our approach is evaluated using Ant Solver, a well-known ant based algorithm for solving Constraint Satisfaction Problems. We show promising results that make our solution as good approach to apply in other metaheuristics.

Keywords

opposite learning strategies; ant algorithms; antipheromone; negative pheromone

1. INTRODUCTION

In order to solve a combinatorial optimization problem it is necessary to search through a huge space of candidate solutions until an acceptable solution is reached. For this reason, most algorithms focus on identifying regions of the search space with promising solutions.

We are involved in a project whose goal is to propose learning strategies to improve the search of metaheuristics. In this work, we focus our attention on the detection of candidate solutions that can initially be considered by the metaheuristic as very promising, but which are finally not good solutions. Our goal is to make an early detection of these solutions and to provide this information to the metaheuristic in order to improve its search.

In our approach we divide the search process into two steps: A prior *first step* for learning about unpromising candidate solutions and, a *second step* where the metaheuristic takes into account the information obtained from the first step to solve the original problem. For the learning step we considered the idea of opposite search. We propose two levels of opposition that define how the search process will be guided and the opposition of the decisions made during the construction of solutions. In this work we present strategies to be used in both, the learning and solving phases, by ant based algorithms. Therefore, the purpose of using opposite knowledge is to detect candidate solutions that are constructed expecting to be promising in terms of the heuristic knowledge, but finally they are not good quality solutions.

Section 2 describes existing approaches that apply opposite knowledge for metaheuristics solving combinatorial optimization problems. Section 3 presents our approach for searching solutions in the opposite direction. *Ant Solver*, the ant based algorithm studied here, is described in Section 4. Section 5 shows the implementation of our approaches in Ant Solver. The experimental setup to study the effect of opposite information is presented in Section 6. Section 7 shows the results of these experiments and Section 9 presents some conclusions and future work.

2. RELATED WORK

Metaheuristics for solving combinatorial optimization problems identify promising regions of the search space and exploiting them to obtain the best quality solution from the region. In this work we are interested in analyzing if the knowledge obtained from the identification of non promising regions can be used to improve the search performed by metaheuristics. In literature, there are several approaches of Opposition-Based Learning (OBL) for metaheuristics like evolutionary algorithms, particle swarm optimization, harmony search and ant colony systems.

First known approaches to OBL were applied to *Differential Evolution (DE)* algorithms [15] [18]. In *Opposite Differential Evolution (ODE)* opposition is understood as opposition in an ordered set of real values. The main objective of these approaches is to accelerate the convergence of

[*]Supported by CONICYT-PCHA/ National Doctorate / 2015-21150696. Supported by FONDECYT Project No. 1151456
[†]Partially supported by the Centro Científico Tecnológico de Valparaíso (CCTVal) Project No. FB0821
[‡]Supported by FONDECYT Initiation into Research Project no. 11150787

GECCO '16, July 20-24, 2016, Denver, CO, USA
© 2016 ACM. ISBN 978-1-4503-4206-3/16/07…$15.00
DOI: http://dx.doi.org/10.1145/2908812.2908927

the search processes by improving the chance of getting best candidate solutions by checking the current and the opposite solutions simultaneously. In ODE, OBL is used during the population initialization and generation jumping. During population initialization, a set of random solutions is created and then their opposites are calculated. Initial population is then conformed by the best solutions from these two sets. During generation jumping, the new population can be generated considering the classical or the current opposite population. In [18], extensive experimentation is presented and authors conclude that ODE has better performance than DE when the size of problems increases. In [1], OBL is applied to a Shuffled Differential Evolution (SDE) algorithm. Here, population is divided into several memeplexes and each one is improved by DE. Four different approaches were tested, combining population initialization and generation jumping. Encouraging results were found using these approaches. Their main disadvantage was the large number of parameter values to tune. Some approaches have also been proposed to study the importance of parameter *jumping rate* [17] [7]. Several approaches related to the scope of opposition [22], interpretation of opposition concept [16], diversity [25] and multiobjective approaches [6] have also been proposed using OBL in Differential Evolution.

Opposition-based Particle Swarm Optimization (OPSO) [10] applies the opposite information in Particle Swarm Optimization (PSO). The objective is accelerate the convergence speed and the capability for handling noisy optimization problems. In this case, the idea of opposition was applied in three components: initialization, generation jumping and local improvement of the best individual in swarm. OPSO considers an opposite swarm obtained from the regular swarm, using a numerical opposite position and opposite velocity. Fittest particles belonging to both swarms will be part of the next generation. This process is repeated iteratively using a jumping rate and a dynamic constriction factor for enhancing the convergence speed. Experiments with well-known optimization functions show that opposition-based ideas improve the performance of PSO. In [24] authors present an extension of OPSO that includes a dynamic Cauchy mutation, applied to the global best particle. The objective of this mutation is to avoid getting trapped in local optima. *Opposition-Based Disturbance* [4] tries to increase the population diversity and search globally in PSO. For this, the approach disturbs the position of particles when the local best is updated.

In ant based algorithms there exist several proposals to use opposite information in order to improve their performance, related to the quality of solutions found or the resources consumed by the algorithm. Schoonderwoerd et al. [19] introduced the concept of *antipheromone* proposing the idea that ants would decrease pheromone rather than reinforce it. Montogomery [14] presented three ideas of antipheromone structure in an Ant Colony System (ACS) algorithm for Traveling Salesman Problem (TSP): (1) subtracting pheromone to the elements which worsen the solutions, (2) using two pheromones matrices: one for good solutions and one for bad solutions; and (3) using *explorer ants*, that are interested in regions with low amount of pheromone.

Malisia et al. [11] presented five extensions of ACS for TSP. Also, they propose that for searching in the opposite way, an ACS can be modified during the solution construction or during the pheromone updating process. Three extensions were presented for modifying the solution construction process: Synchronous Opposition (SO), Free Opposition (FO) and Free Quasi-Opposition (FQO). These algorithms consider that each ant has to be paired: a *leading ant* and an *opposite ant*. The *leading ant* constructs solutions like in ACS. At each step, the leading ant selects each city based on its rank, while the *opposite ant* selects the city with the opposite rank of the leading ant. These algorithms differ in the decision made when both ants are in the same city. Other two extensions modify the pheromone updating process: Opposite Pheromone per Node (OPN) and Opposite Pheromone per Edge (OPE). When an ant is constructing a solution, in both cases, altered pheromone values can be considered. These altered values try to make farther cities more desirable. Both approaches differ in the way the pheromone matrix is altered: OPN modifies a copy of the full current pheromone matrix while OPE only modifies the edge being added to the solution. In [12], Malisia presents a pheromone update framework called Opposite Pheromone Update (OPU). The idea is to speed up the learning process modifying the pheromone content. Pheromone is deposited or removed from opposite edges, depending of what kind of ant algorithm is used and the problem being solved. After pheromone is updated, for each node of a given solution, outgoing edges are ranked.

The main difference between our approaches and most of the OBL mechanisms is related to when the knowledge is generated and how it is used. In the literature related to this topic, opposite search occurs simultaneously to the search process of the original algorithm. In this work we propose to incorporate a first step of search in the opposite direction in order to extract knowledge. This knowledge is related to the discovering of regions of search space that look promising but do not lead to optimal solutions. This is then transmitted to the original algorithm aiming to improve its search. In the following section we will introduce our opposite learning approach.

3. OUR APPROACH

The objective of our approach is to improve metaheuristics search process including a learning phase, for solving combinatorial problems. The main idea is to invest an amount of time searching for opposite solutions and, provide this information to the original metaheuristic. Using this information, the metaheuristic can focus its search on more promising regions of the search space. Before explaining the details of our approach we need to consider the following definitions:

Definition 3.1. *Full Opposite Combinatorial Problem*
Given a combinatorial problem COP $= \{X, D, C, F\}$ where X are the variables, D their discrete domains, C the set of constraints and a function F to be minimized, we define its full opposite combinatorial problem FOCOP $= \{X, D, C, O_F\}$, where O_F maximizes the same function of F.

Note: Here, the same definition can be used when the problem is a maximization one and the full opposite problem corresponds to minimization.

Definition 3.2. *Candidate Solutions*
Given a metaheuristic M designed to solve a COP, we define

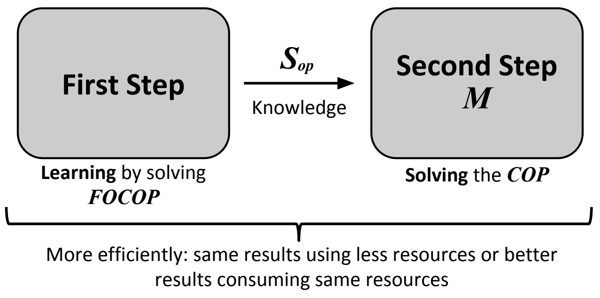
First Step
S_{op}
Knowledge
Second Step
M
Learning by solving *FOCOP*
Solving the *COP*
More efficiently: same results using less resources or better results consuming same resources

Figure 1: Division of the search process

the set S_p as the candidate solutions that M obtains for COP. We define S_{op} as the set of candidate solutions for FOCOP.

We have divided our approach into two steps: The first one searches for a set S_{op} for the $FOCOP$ and the second step solves the COP using the metaheuristic M. We illustrate this procedure in figure 1. Knowledge is transfered from the first to the second step for being incorporated and used. Thus, the task is to define strategies for including this information without modifying the original metaheuristic. For this, we decided to consider the metaheuristic M as a black box, adding a previous step to its execution and maintaining the design decisions made by the authors when they proposed and evaluated it.

In this work we present an application of our learning approach for ant algorithms. Here, the learning and solving phase, are performed using ant algorithms. Knowledge is transfered from the first to the second step using the pheromone matrix. Roughly speaking, during the first step ants deposit pheromone for marking candidate solutions that are not really promising. We named the pheromone used in this step as antipheromone to emphasize the idea of the opposite search. Our strategy for ant algorithms will be explained in the next section. From now on we will refer to first step as *Antipheromone Step*.

3.1 Antipheromone strategy

Real ants use pheromone to share knowledge for finding food efficiently. Ant Colony Optimization (ACO) algorithms use pheromone to share knowledge for guiding ants to regions of the search space where good quality solutions can be found. A main objective of our approach is to answer the question: Would it be possible to use Ant Colony Optimization to initially identify components or assignments related to sub-optimal solutions, and solve a problem more efficiently than without this knowledge?

Our antipheromone strategy requires definitions about how to search in the opposite direction and how to transfer the information obtained from the antipheromone step to the second step, in order to accomplish the goal of improving the search of the metaheuristic.

It is important to note that, it is not necessary that the two ant algorithms be of the same kind, i.e, Ant System, Ant Colony System, $\mathcal{MAX}-\mathcal{MIN}$ Ant System ($\mathcal{MMAS}$). Otherwise, they must use the same pheromone matrix structure in order to simplify the translation. More precisely, we define the following components:

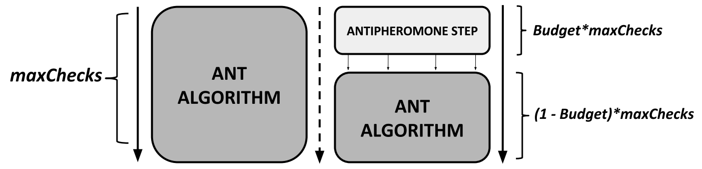

Figure 2: Budget distribution

- The amount of antipheromone $\Delta\mathtt{anti\_\tau}$ that will be deposited by each ant during the *Antipheromone Step*. This amount must be consistent with how much pheromone is deposited by the ants in the original metaheuristic. The amount of antipheromone should not be constant, the worse the solution, the higher amount of antipheromone should be deposited. Moreover, the amount of antipheromone should not exceed the maximum pheromone that can be deposited during the solving step.
- A *translation rule* for including the information during the Antipheromone Step to initialize the pheromone matrix of the metaheuristic. Dispersion of values in antipheromone matrix should be considered to determine the amount of pheromone for each arc. If the ant based algorithm is an $\mathcal{MMAS}$ [21], it is necessary to translate the antipheromone considering the maximum and minimum allowed values for pheromone.
- The *Budget* defines the effort that will be invested in the Antipheromone Step. Figure 2 shows how resources are distributed. It is important to remark that the effort invested in the whole process (both steps) should be the same as the one used by the metaheuristic: the higher the budget for the antipheromone step, the lower the effort during the second step. On the other hand, a very low *Budget* will not allow the antipheromone step to converge to the desired regions of the search space.

In this work we apply the antipheromone strategy to a well-known algorithm for solving Constraint Satisfaction Problems called Ant Solver [20]. Ant Solver is an interesting algorithm for evaluate our strategy because is considered a comparable approach to constraint programming techniques for binary CSP [23]. However, it is important to remark that our objective is not to propose the best algorithm for solving Constraint Satisfaction Problems. Our focus is to evaluate the contribution of including an opposite learning strategy to improve the search process of an ant based algorithm, in this case, for solving CSP's. In the following section we present Ant Solver algorithm and then we explain how our opposite learning approach can be applied to this study case.

4. ANTS FOR CONSTRAINT SATISFACTION PROBLEMS

A *Constraint Satisfaction Problem* (CSP) is composed of a set of *variables* $X = \{X_1, \ldots, X_n\}$, their related *domains* $D_1, \ldots, D_n$ and a set C containing z *constraints* on these variables. The domain of a variable is a set of values to which the variable may be instantiated.

Definition 4.1. *Complete Instantiation*
A *Complete Instantiation* **I** is a mapping from a n-tuple of

variables $(X_1, \ldots, X_n) \rightarrow D_1 \times \ldots \times D_n$, such that it assigns a value from its domain to each variable in X. Instantiation of variable X_i can be represented as $\langle X_{p_i}, v \rangle$.

Definition 4.2. *Partial Instantiation*
Given a subset of variables $X_p \subseteq X$, a *Partial Instantiation* $\mathbf{I_p}$ is a mapping from a j-tuple of variables$(X_{p_1}, \ldots, X_{p_j}) \rightarrow D_{p_1} \times \ldots \times D_{p_j}$, such that it assigns a value from its domain to each variable in X_p.

A solution to the CSP consists of an instantiation of all the variables which does not violate any constraint. *Ant Solver* (AS) algorithm was proposed to solve CSP's [20]. Ant Solver searches for a solution that minimizes the number of unsatisfied constraints. Algorithm 1 shows Ant Solver structure. At each step, each ant constructs a complete instantiation $\mathbf{I}$ for the CSP. Unlike classical ant algorithms, when applied to CSP, the construction of solutions considers two decisions at each assignment: the selection of the next variable, according to some given criteria, and the selection of the value, made probabilistically according to the transition rule. Pheromone information is updated at the end of each cycle. 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 assignment of values.

Algorithm 1: Ant Solver algorithm

```
Input : Set of parameters and Initialized pheromone trails
Output: Overall best solution reached
1 pre-processing($nBest$,$\epsilon$);
2 while $F(\mathbf{I}) > 0$ or maxChecks not reached do
3     for $k = 1$ to $nbAnts$ do
4         $\mathbf{I_k} \leftarrow$ ConstructCompleteInstantiation();
5         post-processing();
6     end
7     UpdatePheromoneTrails($\mathbf{I_1}, \ldots, \mathbf{I_{nbAnts}}$);
8 end
```

Equation 1 shows the transition rule used by Ant Solver. It is important to notice that it not only depends on local relations between the candidate node and the last visited, but also on a global relation between the candidate node and the whole set of visited nodes $\mathbf{I_p}$. Hence, the pheromone factor of node $\langle X_j, v \rangle$ depends on pheromone laid on all edges between $\langle X_j, v \rangle$ and the nodes in $\mathbf{I_p}$.

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

Equation 2 shows the heuristic factor considered by Ant Solver. This factor also depends on the whole set of currently visited nodes in $\mathbf{I_p}$. In Ant Solver it is inversely proportional to the number of new violated constraints when setting $X_j = v$.

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

Ants that find the best quality solutions are allowed to lay pheromone, not only in the path formed by the nodes in constructed solution, but also in each pair of visited nodes. Equation 3 shows the amount of pheromone $\Delta\tau$ deposited by each ant:

$$\Delta\tau(\mathbf{I_k}, i, j) = \frac{1}{F(\mathbf{I_k})} \quad (3)$$

where $\mathbf{I_k}$ is a complete instantiation constructed by ant$_k$.

Ant Solver includes *pre-* and *post-* processing steps that use a *min-conflicts* based local search procedure. The pre-processing phase performs local search repeatedly to collect information that is used to initialized pheromone trails. The post-processing phase performs local search after each ant has constructed a complete assignment.

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 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.

5. OPPOSITE-LEARNING STRATEGIES

This section presents the implementation of the *Antipheromone Step* in Ant Solver for solving CSP [20]. As we mentioned in Section 3.1, it is important to previously define some components for implementing our approach:

- The amount of antipheromone $\Delta\texttt{anti\_}\tau$ that will be deposited by each ant during the *Antipheromone Step*.

  In this case we have defined $\Delta\texttt{anti\_}\tau$ as:

  $$\Delta\texttt{anti\_}\tau = 1/(1 + |worstIter - worstExec|) \quad (4)$$

  where *worstIter* is the quality of the worst solution found during this iteration and *worstExec* is the quality of the worst solution found so far. We have selected this function in order to favor candidate solutions with a higher value of evaluation function and trying to maintain bounded the orders of magnitude of $\Delta\texttt{anti\_}\tau$.

- A *translation rule* for including the collected information during the Antipheromone Step to initialize the pheromone matrix of Ant Solver. In this case, we consider the complement of the antipheromone in each arc to obtain the pheromone according to:

  $$\tau_{ij} = \tau_{max} - \texttt{anti\_}\tau_{ij} \quad (5)$$

  where $\texttt{anti\_}\tau$ corresponds to the antipheromone matrix. We have selected this translation rule in order to clearly transmit the knowledge acquired during the antipheromone step. Hence, if a pair of nodes should not be part of the solution, the solving phase will discard it easily.

- The *Budget* defines the effort that will be invested in the Antipheromone Step. Budget will be considered as a percentage of the total effort devoted to the entire search process. Here, we have tuned this parameter for determining the best budget to execute the process.

Algorithm 2: Antipheromone strategy algorithm

```
  Input  : Translation Rule $t$
  Output: Pheromone Matrix
1 while Budget not reached do
2     for $k = 1$ to nbAnts do
3         $\mathbf{I_k} \leftarrow$ ConstructOpCompleteInstantiation();
4     end
5     UpdateAntipheromoneTrails($\mathbf{I_1}, \ldots, \mathbf{I_{nbAnts}}$);
6 end
7 $\tau \leftarrow$ TranslateAntipheromone($t$);
```

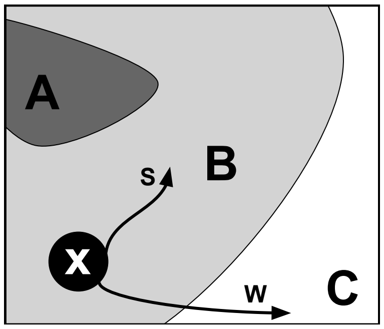

Figure 3: Levels of opposition example

Algorithm 2 shows the general structure of the Antipheromone Step. At every iteration, each ant will construct a solution searching in the opposite way of the problem objective (l.2-4). Then, ants will deposit antipheromone, marking instantiations that are part of opposite solutions (l.5). During the antipheromone step, the same *updating pheromone rule* of Ant Solver is used (l.5). When the *Budget* of resources defined for *Antipheromone Step* has been consumed, the knowledge learned is transmitted to Ant Solver. To do so, the antipheromone matrix is translated into the pheromone matrix using a defined *translation rule* (l.7). Then, *Ant Solver* starts solving the original problem using the obtained pheromone matrix.

In order to explore and compare different ideas of opposition learning, we propose two different levels of opposition. Each level defines how opposed are the decisions made during the construction of solutions in the antipheromone step. Figure 3 shows an example of search space, where quality of solutions decreases from regions A to C. We will suppose that search process starts in candidate solution x. We propose two levels of opposition: *Slow* (S) and *Worst* (W). These levels define where the search process will be guided: level S points to sub-optimal or apparently good solutions in region B and, W points to worst quality solutions in region C. These concepts were formally traduced in the following approaches:

- *Soft Opposite-Learning Ant Solver* (SOL-AS) is related to learning about solutions that are apparently good quality ones in terms of the heuristic knowledge, but they are sub-optimal solutions in terms of the evaluation function. For this, at each iteration, the worst quality solution is marked during the antipheromone step. In SOL-AS the same heuristic knowledge of Ant Solver is used, which is focused on increasing the number of satisfied constraints. The objective is to try to perform a search process similar to AS, but marking poor quality solutions.
- *Worst Opposite-Learning Ant Solver* (WOL-AS) is focused on learning about worst quality solutions. For this, the heuristic knowledge is translated:

$$\eta_{Trans} = Max(\eta_I) + Min(\eta_I) - \eta_I(\langle X_j, v\rangle) \quad (6)$$

  where $Max(\eta_I)$ is the higher heuristic value for $(\langle X_j, v\rangle)$ considering $\mathbf{I}$, $Min(\eta_I)$ the lowest one and, $\eta_I(\langle X_j, v\rangle)$ is the heuristic knowledge of instantiation $(\langle X_j, v\rangle)$. As we defined in 3.1, WOL-AS tries to solve a Full Opposite Combinatorial Optimization Problem. Also, antipheromone is deposited in the worst quality solution at each iteration. The objective is to give priority to the worst decisions during the antipheromone step.

The decision of which level of opposition is better for a specific algorithm will be defined by which knowledge will be useful for solving the problem. We expect that the search process of SOL-AS visit close regions to those Ant Solver does. On the other hand, we expect that WOL-AS search process moves through regions of the search space that Ant Solver could never visit. Both approaches will be evaluated and compared with Ant Solver, using a set of instances belonging to the transition phase.

6. EXPERIMENTS

In this section we present the experiments performed in order to evaluate our antipheromone strategy. The main objective of these experiments is to compare the quality of solutions found by a classical ant algorithm versus the quality found when our approaches are incorporated to the same ant algorithm. Our experiments have been performed using CSP instances belonging to the phase transition, where most complex instances are found [9].

6.1 Instances

For our experiments we have considered binary CSP's. Binary CSP's can be generated at random 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,
- $p_1 \in [0, 1]$ 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. it determines the number of incompatible pairs of values for each constraint.

The CSP instances used 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 [9].

For our experiments we generated hard to solve instances belonging to the transition phase. In order to assure that each instance is solvable, we have added a random generated solution to each one. For our experiments we considered instances with $n = 100$ variables, $m = 8$ domain sizes, $p_1 = 0.14$ and p_2 ranging from 0.22 to 0.31.

6.2 Experimental Setup

For our experiments we used the code of Ant Solver as it has been introduced in [20]. We tuned the parameter values of our approaches considering as training set a selection of 10 problem instances, one from each category ($p_2 \in [0.22, 0.31]$). The average number of conflicts of final solutions was used as success criterion. Table 1 shows the parameter configuration used for the antipheromone step.

Table 1: Parameter values

Algorithm	α	β	ρ	τ_{max}	*Budget*
SOL-AS	2.00	10.00	0.00	2.00	0.05
WOL-AS	9.50	7.00	0.90	2.50	0.05

As our objective is to maintain the design and decisions made for the original algorithm, we considered the same parameter values proposed in [20] for the second phase. Those parameter values are: $nbAnts = 8$, $\alpha = 2.00$, $\beta = 10.00$, $\rho = 0.01$, $\tau_{max} = 4$, $nBest = 200$ and $\epsilon = 0.20$.

As it is usual to evaluate algorithms in the constraint programming community, we considered a fixed number of constraint checks as stopping criteria [8]. On the other hand, Ant Solver considers *pre-* and *post-* processing steps that use a *min-conflicts* based local search procedure for improving solutions. We defined a maximum number of constraint checks for both approaches for fair comparisons and is defined as $4 \times 10E9$. Considering the obtained value for *Budget*, our approaches are consuming only $0.05 \times 4 \times 10E9 = 2 \times 10E8$ constraint checks during the first step.

7. RESULTS

We have evaluated these algorithms using a total number of 100 solvable CSP instances. These instances are classified into categories according to the value of p_2 ranging from 0.22 to 0.31. These categories contain the transition phase for these problems. Each algorithm was tested using 20 different seeds per instance.

Table 2 shows the percentage of successful executions of each approach in each category. We consider a successful execution when the algorithm solves the problem instance. The three algorithms were able to solve all the instances in categories $p2\_0.22$ and $p2\_0.31$, because they contain the easier instances. The results obtained from category $p2\_0.23$ to $p2\_0.30$ show that these instances are the transition phase of these algorithms. In these categories, SOL-AS improved the search process of Ant Solver considering an improvement between a 0.5% on category $p2\_0.30$ and a 22.5% on category $p2\_0.27$.

Table 2: Percentage of Successful Executions

Category	*AS*	*SOL-AS*	*WOL-AS*
p2_0.22	100.0	100.0	100.0
p2_0.23	63.5	**85.5**	68.0
p2_0.24	35.5	**50.5**	32.0
p2_0.25	13.0	**31.5**	12.0
p2_0.26	36.0	**58.0**	33.0
p2_0.27	50.0	**72.5**	47.0
p2_0.28	80.0	**96.5**	79.5
p2_0.29	99.5	**100.0**	100.0
p2_0.30	97.5	**100.0**	95.50
p2_0.31	100.0	100.0	100.0

Table 3: Average number of unsatisfied conflicts of non-solved problems

Category	*AS*	*SOL-AS*	*WOL-AS*
p2_0.22	-	-	-
p2_0.23	1.00	1.00	1.00
p2_0.24	1.16	**1.01**	1.21
p2_0.25	1.70	**1.25**	1.70
p2_0.26	2.65	**2.11**	2.63
p2_0.27	3.10	**1.84**	2.95
p2_0.28	1.98	**1.00**	1.84
p2_0.29	1.00	-	-
p2_0.30	2.20	-	3.18
p2_0.31	-	-	-

Analyzing the performance of WOL-AS, results show that this strategy improves the search process of Ant Solver for categories $p2\_0.23$ and $p2\_0.29$. On the other hand, between categories $p2\_0.24$ and $p2\_0.28$, the Worst Opposite strategy worsen the performance of Ant Solver. Here, the worsening of solutions happens between a 0.5% for $p2\_0.28$ to a 3.5% for category $p2\_0.24$. These results show that searching for the worst quality solutions is not useful for solving these kind of problems.

From another point of view, Table 3 shows the average number of unsatisfied conflicts of non-solved problems. Cells filled with "-" represent those cases where all problems were solved (100% solved instances on Table 2). Here we can observe that solutions obtained by *SOL-AS* show a lower number of unsatisfied conflicts than those obtained by Ant Solver and *WOL-AS*. More specifically, this happens on categories between $p2\_0.24$ to $p2\_0.28$. Then, *SOL-AS* solves more problems and finds better quality solutions than *AS*. For example the biggest improvement percentage of solved problems was obtained in category $p2\_0.27$. Here, *AS* was not able to solve 50% of the executions that have an average number of 3.10 unsatisfied conflicts, while, *SOL-AS* was not able to solve only 27.5% of the executions that have average number of 1.84 unsatisfied conflicts. About the performance of *WOL-AS*, this strategy helps Ant Solver on categories between $p2\_0.26 - p2\_0.28$. Also, for categories $p2\_0.24$ and $p2\_0.30$, worsens the average number of unsatisfied conflicts of AS. Finally, results from Tables 2 and 3 show that *SOL-AS* outperforms the other two studied algorithms.

7.1 Wilcoxon Test

We have used the pair-wise Wilcoxon non-parametric test [2] with the Bonferroni correction [13] for comparing the performance of AS with each our opposite-based approaches. For more details about the application of Wilcoxon test refer to [5]. The hypotheses considered in these tests are:

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

Table 4: Wilcoxon test

Comparison			Neg. Ranks	Pos. Ranks	p-value
AS	-	**SOL-AS**	0	563	0.00
AS	-	**WOL-AS**	55	13	0.00

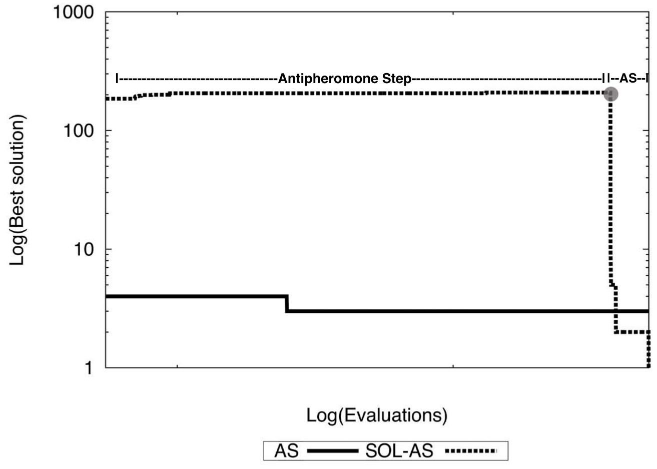

Figure 4: Convergence graph

Table 4 shows the results for two different tests: AS vs $SOL - AS$ and AS vs $WOL - AS$. Results show that AS is statistically different from our approaches. As the idea of solving CSP's is to minimize the number of unsatisfied conflicts, the number of positive ranks shows the cases when our approaches outperform AS. Negative ranks show the cases when AS outperforms our strategies. Comparing AS with *SOL-AS*, results confirm that including the Soft Opposite strategy on Ant Solver is useful for improving its search process. On the other hand, results show that the Worst Opposite strategy is not useful for Ant Solver. This comparison was made considering a confidence level of 95%. All these computations were done using the statistical software package $PSPP$ [1].

7.2 Convergence Graphs

This section shows a convergence graph comparing our best approach and Ant Solver to understand how the search process is modified using this opposite learning strategy. Figure 4 shows the convergence of both algorithms for a problem instance belonging to $p2\_0.26$ category. The graph shows that the search performed by AS converges in the first steps, it shows a small improvement in the middle of the process and finishes with a non-optimal solution. On the other side, *SOL-AS* starts searching for bad quality solutions until the antipheromone step ends. Then, it starts searching for good quality solutions and converges quickly to the optimal. Both steps, the antipheromone step and Ant Solver, are shown in the graph.

8. DISCUSSION

We evaluated two different strategies that consider a different level of opposition, constructing opposite solutions for guiding the search process to: sub-optimal (or apparently good) and worst quality solutions. These levels were traduced to SOL-AS and WOL-AS respectively. For using these strategies, it is important to study how to define the amount of antipheromone to be deposited, a translation rule and the effort budget to be consumed during the antipheromone step.

We have evaluated our antipheromone strategies using Ant Solver and results show that SOL-AS is able to solve more hard problems that belong to the transition phase. As we expected, the regions of the search space visited by SOL-AS and Ant-Solver are close.

[1]Available in https://www.gnu.org/software/pspp/

9. CONCLUSIONS AND FUTURE WORK

We have proposed two opposite-learning strategies to improve the search process of Ant Solver. We have shown the key idea of dividing the search in two steps using our approach can significantly improve the search process, and therefore solutions given by the solver. SOL-AS has shown to be a very effective strategy to improve the Ant Solver performance without any change to the original algorithm. Future works include to apply our approach to the Job Shop Scheduling problem which suffer from the second order deception [3]. We also plan to study the effectiveness of our opposite-learning strategy to this deceptive condition.

10. ACKNOWLEDGMENTS

We would like to thank Christine Solnon for providing us the code for performing our experiments.

11. REFERENCES

[1] M. A. Ahandani and H. Alavi-Rad. Opposition-based learning in the shuffled differential evolution algorithm. *Soft Comput.*, 16(8):1303–1337, 2012.

[2] T. Bartz-Beielstein and M. Preuss. Experimental research in evolutionary computation. In *Genetic and Evolutionary Computation Conference, GECCO 2007*, pages 3001–3020. Springer Berlin, 2007.

[3] C. Blum and M. Dorigo. Deception in ant colony optimization. In M. Dorigo, M. Birattari, C. Blum, L. M. Gambardella, F. Mondada, and T. Stützle, editors, *Ant Colony Optimization and Swarm Intelligence, 4th International Workshop, ANTS 2004*, volume 3172 of *Lecture Notes in Computer Science*, pages 118–129. Springer, 2004.

[4] Y. Chi and G. Cai. Particle swarm optimization with opposition-based disturbance. In *Informatics in Control, Automation and Robotics (CAR), 2010 2nd International Asia Conference on*, volume 2, pages 223–226, 2010.

[5] J. Derracm, S. García, D. Molina, and F. Herrera. A practical tutorial on the use of nonparametric statistical tests as a methodology for comparing evolutionary and swarm intelligence algorithms. *Swarm and Evolutionary Computation*, 1(1):3–18, 2011.

[6] N. Dong and Y. Wang. Multiobjective differential evolution based on opposite operation. In *2009 International Conference on Computational Intelligence and Security, CIS 2009*, pages 123–127. IEEE Computer Society, 2009.

[7] A. Esmailzadeh and S. Rahnamayan. Opposition-based differential evolution with protective generation jumping. In *Differential Evolution (SDE), 2011 IEEE Symposium on*, pages 1–8, 2011.

[8] E. C. Freuder and R. J. Wallace. Partial constraint satisfaction. *Artificial Intelligence*, 58(1-3):21–70, 1992.

[9] I. P. Gent, E. Macintyre, P. Prosser, B. M. Smith, and T. Walsh. Random constraint satisfaction: Flaws and structure. *Constraints*, 6(4):345–372, 2001.

[10] L. Han and X. He. A novel opposition-based particle swarm optimization for noisy problems. In *Natural Computation, 2007. ICNC 2007. Third International Conference on*, volume 3, pages 624–629, 2007.

[11] A. Malisia and H. Tizhoosh. Applying opposition-based ideas to the ant colony system. In *2007 IEEE Swarm Intelligence Symposium, SIS 2007*, pages 182–189, 2007.

[12] A. R. Malisia. Improving the exploration ability of ant-based algorithms. In H. R. Tizhoosh and M. Ventresca, editors, *Oppositional Concepts in Computational Intelligence*, volume 155, pages 121–142. Springer, 2008.

[13] R. G. Miller. *Simultaneous Statistical Inference*. Springer-Verlag, 1981.

[14] J. Montgomery and M. Randall. Anti-pheromone as a tool for better exploration of search space. In M. Dorigo, G. Di Caro, and M. Sampels, editors, *Ant Algorithms, Third International Workshop, ANTS 2002*, volume 2463 of *Lecture Notes in Computer Science*, pages 100–110. Springer, 2002.

[15] S. Rahnamayan, H. Tizhoosh, and M. Salama. Opposition-based differential evolution algorithms. In *IEEE International Conference on Evolutionary Computation, CEC 2006*, pages 2010–2017, 2006.

[16] S. Rahnamayan, H. Tizhoosh, and M. Salama. Quasi-oppositional differential evolution. In *IEEE Congress on Evolutionary Computation (CEC 2007)*, pages 2229–2236, 2007.

[17] S. Rahnamayan, H. R. Tizhoosh, and M. M. A. Salama. Opposition-based differential evolution (ODE) with variable jumping rate. In *IEEE Symposium on Foundations of Computational Intelligence (FOCI 2007)*, pages 81–88. IEEE, 2007.

[18] S. Rahnamayan and G. G. Wang. Solving large scale optimization problems by opposition-based differential evolution (ODE). *WSEAS Trans. on Computation*, 7(10):1792–1804, 2008.

[19] R. Schoonderwoerd, J. L. Bruten, O. E. Holland, and L. J. M. Rothkrantz. Ant-based load balancing in telecommunications networks. volume 5, pages 169–207. MIT Press, 1996.

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

[21] T. Stützle and H. H. Hoos. MAX-MIN Ant System. volume 16, pages 889–914. Elsevier, 2000.

[22] J. Tang and X. Zhao. On the improvement of opposition-based differential evolution. In *Sixth International Conference on Natural Computation, ICNC 2010*, pages 2407–2411. IEEE, 2010.

[23] J. I. van Hemert and C. Solnon. A study into ant colony optimisation, evolutionary computation and constraint programming on binary constraint satisfaction problems. In J. Gottlieb and G. Raidl, editors, *Evolutionary Computation in Combinatorial Optimization, 4th European Conference, EvoCOP 2004*, volume 3004 of *Lecture Notes in Computer Science*, pages 114–123. Springer, 2004.

[24] H. Wang, H. Li, Y. Liu, C. Li, and S. Zeng. Opposition-based particle swarm algorithm with cauchy mutation. In *Proceedings of the IEEE Congress on Evolutionary Computation, CEC 2007*, pages 4750–4756, 2007.

[25] J. Wang, Z. Wu, and H. Wang. Hybrid differential evolution algorithm with chaos and generalized opposition-based learning. In *Advances in Computation and Intelligence*, volume 6382 of *Lecture Notes in Computer Science*, pages 103–111. Springer Berlin Heidelberg, 2010.