Globally optimal triangulations of minimum weight using Ant Colony Optimization metaheuristic

Maria Gisela Dorzán, Edilma Olinda Gagliardi, Mario Guillermo Leguizamón
Facultad de Ciencias Físico Matemáticas y Naturales
Universidad Nacional de San Luis
San Luis, Argentina
Email: mgdorzan, oli, legui @unsl.edu.ar
and
Gregorio Hernández Peñalver
Facultad de Informática
Universidad Politécnica de Madrid
Madrid, España
Email: gregorio@fi.upm.es

ABSTRACT
Globally optimal triangulations are difficult to be found by deterministic methods as, for most type of criteria, no polynomial algorithm is known. In this work, we consider the Minimum Weight Triangulation (MWT) problem of a given set of n points in the plane. Our aim is to show how the Ant Colony Optimization (ACO) metaheuristic can be used to search for globally optimal triangulations of minimum weight. We present an experimental study for a set of instances for MWT problem. We create these instances since no reference to benchmarks for this problem were found in the literature. We assess through the experimental evaluation the applicability of the ACO metaheuristic for MWT problem.

Keywords: Triangulation, Minimum Weight Triangulation, Computational Geometry, ACO Metaheuristic.

I. INTRODUCTION
In Computational Geometry there are many problems that either are NP-hard or no polynomial algorithms are known. Therefore, we can find approximate solutions using metaheuristics. The optimization problems related to special geometric configurations are interesting to research due to their use in many fields of application. Triangulations are planar partitions, which received considerable attention mainly due to their applications, e.g., visibility, ray-shooting, kinetic collision detection, rigidity, guarding, etc. Minimizing the total length has been one of the main optimality criteria for triangulations. Indeed, the Minimum Weight Triangulation (MWT) minimize the sum of the edge lengths, providing a quality measure for determining how good is a structure. The complexity of computing a minimum weight triangulation has been one of the most longstanding open problems in Computational Geometry, introduced by Garey and Johnson [10] in their open problems list, and various approximation algorithms were proposed over time. Mulzer and Rote [23] recently showed that MWT problem is NP-hard.

Given the inherent difficulty of the above mentioned problem, the approximate algorithms arise as alternative candidates for solving MWT problem. These algorithms can obtain approximate solutions to the optimal solutions, and they can be specific for a particular problem or they can be part of a general applicable strategy in the resolution of different problems. The metaheuristic methods satisfy these properties.

A metaheuristic is an iterative generation process that guides the search of solutions intelligently combining different concepts of diverse fields as artificial intelligence [24], biological evolution [2], swarm intelligence [13], among others. These algorithms have a simple implementation and they can efficiently find good solutions for NP-hard optimization problems [22]. For the experimental study presented in this work we use the Ant Colony Optimization (ACO) metaheuristic.

According to the current state-of-the-art with respect to theoretical results about the problems considered in this investigation, we adopted to solve them a metaheuristic technique as the more appropriate approach to find nearly optimal solutions. Previous works about approximations on MWT problem using metaheuristic, were presented in [6] and [7], where we described the design of the ACO algorithms and gave the first steps in this research.

This paper is organized as follows. In the next Section, we present the theoretical aspects of triangulations. In Section III, we present the general overview of the ACO metaheuristic and in Section IV, we present the proposed ACO algorithm for solving the MWT problem. In Section V, we describe the MWT instances used and the details and results of the experimental study in which we analyze the sensitivity of some important parameters on the performance of the proposed ACO algorithm. Last section is reserved for the conclusions and future vision.

II. MINIMUM WEIGHT TRIANGULATION
Let S be a set of points in the plane. A triangulation of S is a partition of the convex hull of S into triangles whose set of vertices is exactly S. The weight of a triangulation T is the sum of the Euclidean lengths of all the edges of T. The triangulation that minimizes this sum is named a Minimum Weight Triangulation of S and it is denoted by MWT(S).

Triangulation is one of the main topics in Computational Geometry and it is commonly used in a large set of applications, such as computer graphics, scientific visualization,
robotics, computer vision, and image synthesis, as well as in mathematical and natural science. MWT problem has a long and rich history, dating back to the 1970s. As far as authors’ knowledge, the MWT problem was first considered by Düppel and Gottschalk [8] who proposed a greedy algorithm which always adds the shortest edge to the triangulation. Later, Shamos and Hoey [29] suggested using the Delaunay triangulation as a minimum weight triangulation. Lloyd [20] provided examples which show that both proposed algorithms usually do not compute the MWT. Similarly, Gilbert [11] and Klimcsek [16] independently showed how to compute a minimum weight triangulation of a simple polygon in $O(n^3)$ time by dynamic programming.

Approaching the problem from other direction, researchers were looking for triangulations that approximate the MWT. The Delaunay triangulation is not a good candidate, since it may be longer by a factor of $\Omega(n)$ (see Figure 1) [14] [21]. The greedy triangulation approximates the MWT by a factor of $\Omega(n)$ (see Figure 2) [21] [18] [19]. Plaisted and Hong [25] showed how to approximate the MWT up to a factor of $O(\log n)$ in $O(n^2\log n)$ time. Lev-copoulos and Krznaric [19] introduced quasi-greedy triangulations, which approximate the MWT within a constant factor. Remy and Steger [27] discovered an approximation scheme for MWT that runs in quasi-polynomial time: for every fixed $\varepsilon$, it finds a $(1 + \varepsilon)$-approximation in $n^{O((\log^3 n)}$ time.

Fig. 1. Two examples of possible triangulations for the same set of points

(a) Delaunay triangulation  (b) Minimum weight triangulation

Fig. 2. Two examples of possible triangulations for the same set of points

(a) Greedy triangulation  (b) Minimum weight triangulation

From the point of view of metaheuristics, many papers present solutions to problems at the field of Graphical Computation. In 1992, Sen and Zheng [28] proposed an algorithm to approximate the minimum weight triangulation using Simulated Annealing but in many proofs they obtain solutions “near” to the ideal ones. The neighborhood is obtained with a flip in a random edge of the current triangulation. In 1993, Wu and Wainwright [32] approximated the minimum weight triangulation using a genetic algorithm where the recombination and mutation operators are the same, such that both of them make a flip to obtain the neighbors. Qin et al. [26] also use a genetic algorithm and they propose new operators for recombination and mutation. Capp and Julstrom [3] present a new weight codification of the triangulations to use it in a genetic algorithm. In the previous mentioned works, the experimental evaluation is rather poor and they do not describe the quality of the obtained solutions. In 2001, Kolingerova and Ferko [17] presented a genetic optimization, which recombination operator is named DeWall and the mutation operator makes a flip in the selected individual. The principal weakness of this method is the time demand.

The complexity of the computation was one of the more interesting opened problems in Geometry Computational until Mulzer and Rote demonstrated in 2006 that MWT’s construction is a NP-hard problem [23].

III. ANT COLONY OPTIMIZATION METAHEURISTIC

The Ant Colony Optimization metaheuristic involves a family of algorithms in which a colony of artificial ants cooperate in finding good solutions to difficult discrete optimization problems. Cooperation is a key design component of ACO algorithms: the choice is to allocate the computational resources to a set of relatively simple agents (artificial ants) that communicate indirectly by stigmergy. Thus, good quality solutions are an emergent property of the agents cooperative interaction.

An artificial ant in an ACO algorithm is a stochastic constructive procedure that incrementally builds a solution by adding opportually defined solution components to a partial solution under construction. Therefore, the ACO metaheuristic can be applied to any combinatorial optimization problem for which a constructive graph can be defined. Each edge $(i, j)$ in the graph represents a possible path and it has associated two information sources that guide the ant moves: pheromone trails and heuristic information. The pheromone trail, denoted by $\tau_{ij}$, encodes a long-term memory about the entire ant search process, and is updated by the ants themselves. The heuristic information, denoted by $\eta_{ij}$, represents a priori information about the problem instance or run-time information provided by a source different from the ants. In many cases $\eta$ is the cost, or an estimate of the cost, of adding the component or connection to the solution under construction. These values are used by the ants to make probabilistic decisions on how to move on the graph. The ants act concurrently and independently and although each ant is complex enough to find a solution to the problem, which is probably poor, good-quality solutions can only emerge as the result of the collective interaction among the ants. This is obtained via indirect communication mediated by the information that ants read or write in the variables storing pheromone trail values. It is a distributed learning process in which the single agents, the ants, are not adaptive.

ACO metaheuristic has also been successfully applied to continuous problems, however, in this paper we only consider and describe its application to discrete problems.
themselves but, on the contrary, adaptively modify the way the problem is represented and perceived by other ants [5]. There are two additional process for updating pheromone and the daemon actions. The pheromone updating is the process by which the pheromone trails are modified. The trail values can either increase, as ants deposit pheromone on the components or connections they use, or decrease, due to pheromone evaporation. The daemon procedure is used to implement centralized actions which cannot be performed by single ants. Examples of daemon actions are the activation of a local optimization procedure, or the collection of global information that can be used to decide whether it is useful or not to deposit additional pheromone to bias the search process from a nonlocal perspective. The daemon can observe the path found by each ant in the colony and select one or a few ants, like those that built the best solutions in the algorithm iteration that allowed to deposit additional pheromone on the connections they used.

A. The general ACO algorithm
In this section we present a general ACO algorithm (Algorithm 1) and a description of its main components. After that, the next two sections respectively describe in detail the specific component of the general ACO algorithm (function BuildSolutionk) that have to be adapted for each of the problem studied here, i.e., MWT.

Algorithm 1 General-ACO

<table>
<thead>
<tr>
<th>Initialize</th>
<th>for c ∈ {1, ..., C} do</th>
</tr>
</thead>
<tbody>
<tr>
<td>for k ∈ {1, ..., K} do</td>
<td></td>
</tr>
<tr>
<td>BuildSolutionk</td>
<td></td>
</tr>
<tr>
<td>EvaluateSolution</td>
<td></td>
</tr>
<tr>
<td>end for</td>
<td></td>
</tr>
<tr>
<td>SaveBestSolutionSoFar</td>
<td></td>
</tr>
<tr>
<td>UpdateTrails</td>
<td></td>
</tr>
<tr>
<td>end for</td>
<td></td>
</tr>
<tr>
<td>ReturnBestSolution</td>
<td></td>
</tr>
</tbody>
</table>

Main components of Algorithm 1:
- **Initialize**: this process initializes the parameters considered for the algorithm. The initial trail of pheromone is associated to each edge, \( \tau_0 \); it is an small positive value, in general, the same for all edges. The quantity of ants of the colony, \( K \). The weights that define the proportion in which they will affect the heuristic information and pheromone trails in the probabilistic transition rule, named respectively \( \beta \) y \( \alpha \). \( C \) is the maximum number of cycles.
- **BuildSolutionk**: this process begins with a partial empty solution which is extended at each step by adding a feasible solution component chosen from the current solution neighbors; i.e., to find a route on the construction graph guided by the mechanism that defines the set of feasible neighbors with regard to the partial solution. The choice of a feasible neighbor is done in a probabilistic way in every step of the construction, depending on the used ACO variant. In this work, the selection rule for the solutions construction is based on the following probabilistic model:

\[
P_{ij} = \begin{cases} 
\frac{\tau_{ij}^{\alpha} \eta_{ij}^{\beta}}{\sum_{h \in F(i)} \tau_{ih}^{\alpha} \eta_{ih}^{\beta}} & j \in F(i); \\
0, & \text{otherwise.}
\end{cases}
\]

- \( F(i) \) is the set of feasible points for point \( i \).
- \( \tau_{ij} \) is the pheromone value associated to edge \( (i, j) \).
- \( \eta_{ij} \) is the heuristic value associated to edge \( (i, j) \).
- \( \alpha \) and \( \beta \) are positives parameters for determining the relative importance of the pheromone with respect to the heuristic information.
- **EvaluateSolution**: evaluates and saves the best solution found by ant \( k \) in the current cycle.
- **SaveBestSolutionSoFar**: saves the best solution found for all cycles so far.
- **UpdateTrails**: increases the pheromone level in the promising paths, and is decreased in other case. First, all the pheromone values are decreased by means of the process of evaporation. Then, the pheromone level is increased when good solutions appear. The following equation is used:

\[
\tau_{ij} = (1 - \rho)\tau_{ij} + \Delta \tau_{ij} \tag{2}
\]

- \( \rho \in (0, 1) \) is the factor of persistence of the trail.
- \( \Delta \tau_{ij} = \sum_{k=1}^{K} \Delta^k \tau_{ij} \) is the accumulation of trail, proportional to the quality of the solutions.
- \( \Delta^k \tau_{ij} = \begin{cases} 
Q/L_k, & \text{when ant } k \text{ used edge } (i, j); \\
0, & \text{in other case.}
\end{cases} \)
- \( Q \) is a constant depending of the problem; it usually set to 1.
- \( L_k \) is the objective value of the solution \( k \).

Pheromone evaporation avoids a fast convergence of the algorithm. In addition, this way of forgetting allows the exploration of new areas of the search space. The update of the pheromone trail can be done according to one of the following criterions: *elitist* and *not elitist*. In the elitist case, the best found solution is used to give an additional reinforcement to the levels of pheromone. The not elitist one uses the solutions found by all the ants to give an additional reinforcement to the levels of pheromone.

IV. THE PROPOSED ACO ALGORITHM FOR MWT (ACO-MWT)

Considering Algorithm 1 described in the previous section, we present BuildSolutionk function for the MWT problem described in Algorithm 2. The other functions in the General-ACO (Algorithm 1) remain the same for ACO-MWT. BuildSolutionk process showed below is an improvement of the algorithm described in [6]. The current version is better than the previous one in the sense that the intersection procedure is invoked fewer times. It must be noticed that in the initial version of the algorithm, we start with an empty solution, i.e., we just have the set of
points $S$. In this improvement, we take the $\beta$-skeleton idea. Let $p$ and $q$ be two distinct points in $S$ with $\beta > 1$. Following Kirkpatrick and Radke [15], the edge $pq$ is included in the $\beta$-skeleton $\beta(S)$ of $S$ if the two circles of diameter $\beta \cdot |pq|$ and passing through both $p$ and $q$ do not enclose any point in $S$. Interestingly, $\beta(S)$ is a subgraph of each MWT(S) provided that $\beta$ is large enough. The original bound $\beta \geq \sqrt{2}$ in Keil [12] was improved later in Cheng and Xu [4] to $\beta > 1.1768$. So, we start with a partial solution composed by the edges of the Relative Neighborhood Graph (RNG) which is called 2-skeleton because is easier to compute than the 1.1768-skeleton.

BuildSolution works as follows, each ant builds a triangulation of a given set or instance $S$, starting from an initial random point. At each step, the algorithm adds a new edge $(i, j)$ if there is no intersection between $(i, j)$ and the edges of the (partial) solution $S_k$. In this case, $i$ is a feasible point for $j$ and vice versa. If the current point has no feasible points, it selects the next reference point according to one of the following criterions: i) random selection; ii) select the point with the largest quantity of feasible points; or, iii) select the point with the lowest quantity of feasible points.

Algorithm 2 BuildSolution

$S_k \leftarrow$ edges of RNG

$i \leftarrow \text{SelectInitialPoint}(S)$

while $S$ is not triangulated do

$F_i \leftarrow \text{FeasiblePoints}(i, S_k)$

if $F_i = \emptyset$ then

$i \leftarrow \text{SelectPoint}(S, S_k)$

$F_i \leftarrow \text{FeasiblePoints}(i, S_k)$

end if

$j \leftarrow \text{SelectPointProb}(F_i)$

if not $\text{IntersectSolution}(i, j, S_k)$ then

$S_k \leftarrow S_k \cup (i, j)$

$i \leftarrow j$

end if

UpdateFeasiblePoints$(i, j)$

end while

In order to better understand the behavior of Algorithm 2, the respective main components are described in the following:

- **SelectInitialPoint**(S): returns a point $p \in S$, randomly selected.
- **FeasiblePoints**(i, $S_k$): returns a set of points $p \in S$, such that the edge $(i, p)$ could not intersect with the edges of the solution $S_k$. Note that this function may return points that are not feasible for $p$.
- **SelectPoint**(S, $S_k$): returns a point $p \in S$, such that **FeasiblePoints**(p, $S_k$) has at least a point. $p$ is selected according yo one of the criteria mentioned previously.
- **SelectPointProb**(F_i): returns a point $j \in F_i$ chosen according to Equation 1, where $\eta_{ij}$ is 1/|d_{ij}|, and $d_{ij}$ is the Euclidean distance between $i$ and $j$.
- **IntersectSolution**(i, j, $S_k$): returns true if the edge $(i, j)$ intersects at least one edge of the solution $S_k$; returns false, in other case.
- UpdateFeasiblePoints(i, j): updates the feasible points of $i$ and $j$, i.e., the points $i$ and $j$ are not more feasible with respect to each other.

V. Experimental Evaluation

In this work we present an ACO algorithm for the MWT problem. The proposed ACO algorithm is represented by an Ant System (AS), a particular instance of the class of ACO algorithms. We show the experimental phase for sets of instances of size 40, 80, 120, and 160 points. Through the experimental evaluation, we assess the applicability of the ACO metaheuristic for MWT problem.

To the best knowledge of the authors, there not exist collections of instances in the literature for MWT problem. According to that, we design an instances generator. Therefore, we have generated respectively a collection of 10 instances of size 40/80/120/160/200; i.e., a total of 50 instances for the problem. Each instance is called LDn-i where $n$ denotes the size of the $i$-instance, with $1 \leq i \leq 10$. The instance generator uses different functions of CGAL Library [1]. The points are randomly generated, uniformly distributed and for each point $(x, y)$, the coordinates $x, y \in [0, 1000]$. Especially for MWT problem, these instances must be pre-processed to guarantee non collinear points which a sine qua non property that has to be met to find suitable solutions for the problem.

Particularity, in this work, we present the experimental research for four instances of size 40, 80, 120, and 160 respectively (LD40-1, LD40-2, LD40-3, LD40-4, LD80-1, LD80-2, LD80-3 LD80-4, LD120-1, LD120-2, LD120-3, LD120-4, LD160-1, LD160-2, LD160-3, and LD160-4). The ACO-MWT algorithm was implemented in C and run on BACO parallel cluster, composed by 60 PCs, with a 3.0 GHz Pentium-4 processor each one and 90 PCs with a 2.4 GHz Core 2 Quad processor each one, under CONDOR batch queuing system.

In this evaluation phase, we used the following parameter values: $\alpha = 1$; $\beta = 1$ and 5; and $\rho = 0.10$, 0.25, and 0.50. elit = 1 and 0, where 1 means that the trail is updated in a elitist way; in other case, the updating is done in a not elitist way. criterion = 1, 2, and 3, is used for selecting a point in the SelectPoint(S, $S_k$) procedure in ACO-MWT algorithm. For criterion = 1 the point is chosen randomly; for criterion = 2, the chosen point has the largest quantity of feasible points; and for criterion = 3, the chosen point has the lowest quantity of feasible points. The number of cycles, $C$ is 1000; the number of ants, $K$ is 50. For each parameter setting, given below, 30 runs were performed by using different random seeds. For each instance, the experiment were done with the twelve parameter setting, according to combinations of parameters before detailed. We obtain average, median, best, and standard deviation values, considering the objective function (weight). We show the results according to the four best parameter settings considering the smaller weights, i.e., the four Best values. We only show the results for the best four parameter settings since the results for the remaining ones were in general of lower quality with respect to the best found values.

Next, we show the experimental results, considering the above presented settings. Each parameter setting is denoted
We carried out the boxplot method to visualize the distribution of the results. It can be seen that the ACO-MWT algorithm found the smaller weights for all cases. ACO-MWT managed to reduce the weights between 1% and 5% with regard to the DT strategy, but for LD40-4 instance achieved a reduction of 8%.

We used the ACO-MWT algorithm among the Delaunay Triangulation (DT). The Table VI shows the smaller weights found of each strategy. From the displayed results it can be seen that the ACO-MWT algorithm found the smaller weights for all cases. ACO-MWT managed to reduce the weights between 1% and 5% with regard to the DT strategy, but for LD40-4 instance achieved a reduction of 8%

We carried out the boxplot method to visualize the distribution of the results. It can be seen that the ACO-MWT algorithm found the smaller weights for all cases. ACO-MWT managed to reduce the weights between 1% and 5% with regard to the DT strategy, but for LD40-4 instance achieved a reduction of 8%.

We carried out the boxplot method to visualize the distribution of the results. It can be seen that the ACO-MWT algorithm found the smaller weights for all cases. ACO-MWT managed to reduce the weights between 1% and 5% with regard to the DT strategy, but for LD40-4 instance achieved a reduction of 8%.

We carried out the boxplot method to visualize the distribution of the results. It can be seen that the ACO-MWT algorithm found the smaller weights for all cases. ACO-MWT managed to reduce the weights between 1% and 5% with regard to the DT strategy, but for LD40-4 instance achieved a reduction of 8%.

We carried out the boxplot method to visualize the distribution of the results. It can be seen that the ACO-MWT algorithm found the smaller weights for all cases. ACO-MWT managed to reduce the weights between 1% and 5% with regard to the DT strategy, but for LD40-4 instance achieved a reduction of 8%.

We carried out the boxplot method to visualize the distribution of the results. It can be seen that the ACO-MWT algorithm found the smaller weights for all cases. ACO-MWT managed to reduce the weights between 1% and 5% with regard to the DT strategy, but for LD40-4 instance achieved a reduction of 8%.

We carried out the boxplot method to visualize the distribution of the results. It can be seen that the ACO-MWT algorithm found the smaller weights for all cases. ACO-MWT managed to reduce the weights between 1% and 5% with regard to the DT strategy, but for LD40-4 instance achieved a reduction of 8%.

We carried out the boxplot method to visualize the distribution of the results. It can be seen that the ACO-MWT algorithm found the smaller weights for all cases. ACO-MWT managed to reduce the weights between 1% and 5% with regard to the DT strategy, but for LD40-4 instance achieved a reduction of 8%.

We carried out the boxplot method to visualize the distribution of the results. It can be seen that the ACO-MWT algorithm found the smaller weights for all cases. ACO-MWT managed to reduce the weights between 1% and 5% with regard to the DT strategy, but for LD40-4 instance achieved a reduction of 8%.

We carried out the boxplot method to visualize the distribution of the results. It can be seen that the ACO-MWT algorithm found the smaller weights for all cases. ACO-MWT managed to reduce the weights between 1% and 5% with regard to the DT strategy, but for LD40-4 instance achieved a reduction of 8%.

We carried out the boxplot method to visualize the distribution of the results. It can be seen that the ACO-MWT algorithm found the smaller weights for all cases. ACO-MWT managed to reduce the weights between 1% and 5% with regard to the DT strategy, but for LD40-4 instance achieved a reduction of 8%.

We carried out the boxplot method to visualize the distribution of the results. It can be seen that the ACO-MWT algorithm found the smaller weights for all cases. ACO-MWT managed to reduce the weights between 1% and 5% with regard to the DT strategy, but for LD40-4 instance achieved a reduction of 8%.

We carried out the boxplot method to visualize the distribution of the results. It can be seen that the ACO-MWT algorithm found the smaller weights for all cases. ACO-MWT managed to reduce the weights between 1% and 5% with regard to the DT strategy, but for LD40-4 instance achieved a reduction of 8%.

We carried out the boxplot method to visualize the distribution of the results. It can be seen that the ACO-MWT algorithm found the smaller weights for all cases. ACO-MWT managed to reduce the weights between 1% and 5% with regard to the DT strategy, but for LD40-4 instance achieved a reduction of 8%.

We carried out the boxplot method to visualize the distribution of the results. It can be seen that the ACO-MWT algorithm found the smaller weights for all cases. ACO-MWT managed to reduce the weights between 1% and 5% with regard to the DT strategy, but for LD40-4 instance achieved a reduction of 8%.

We carried out the boxplot method to visualize the distribution of the results. It can be seen that the ACO-MWT algorithm found the smaller weights for all cases. ACO-MWT managed to reduce the weights between 1% and 5% with regard to the DT strategy, but for LD40-4 instance achieved a reduction of 8%.

We carried out the boxplot method to visualize the distribution of the results. It can be seen that the ACO-MWT algorithm found the smaller weights for all cases. ACO-MWT managed to reduce the weights between 1% and 5% with regard to the DT strategy, but for LD40-4 instance achieved a reduction of 8%.
VI. Conclusion

In this work, we present the design of approximation algorithms for solving the Minimum Weight Triangulation problem for sets of points in the plane. The proposed ACO algorithm for the MWT is represented by an Ant System (AS), a particular instance of the class of ACO algorithms. We also detailed the generation of instances for the experimental evaluation, being this another contribution of this paper, since there are not available instances with special properties for building triangulations. Currently, we are continuing the experimental evaluation over a large collection of instances, where the obtained results could be used as benchmarks. From this initial experimental phase we obtained preliminary results that will guide future experimentation. Currently, we are in the phase of applying a more methodological approach (e.g., SPOT toolbox[30]) for the experimental design following the directions of Bartz-Beielstein [31] and Fang et al.[9]. This methodological approach for parameter optimization will let us assess more accurately the behavior of our proposed algorithms under different parameter setting.

Future work will address the use different parameter setting for the ACO algorithms and the experimentation with the whole collection of instances generated. In addition, we aim at comparing the proposed algorithms against other strategies like greedy algorithms or alternative meta-heuristics (e.g., evolutionary algorithm, particle swarm optimization algorithms).

Fig. 3. MWT: Boxplots for (a) LD40-1, (b) LD40-2, (c) LD40-3, and (d) LD40-4

Fig. 4. MWT: Boxplots for (a) LD80-1, (b) LD80-2, (c) LD80-3, and (d) LD80-4

Fig. 5. MWT: Boxplots for (a) LD120-1, (b) LD120-2, (c) LD120-3, and (d) LD120-4

Fig. 6. MWT: Boxplots for (a) LD160-1, (b) LD160-2, (c) LD160-3, and (d) LD160-4

bution of the weights for each environment. The Figures 3, 4, 5, and 6 show the boxplots of the weights obtained for the 30 seeds for four instances for 40, 80, 120, and 160 points for the twelve configurations. The medians are similar for $\beta$ between 0.10, 0.25 and 0.50. The algorithm is more robust when elit = 1 because the 50% of the values (values between the first and third quartile) are very closed around the median value. We obtained better results with $\beta = 5$ and elit = 1 (see parameter settings 7 to 12).
ACKNOWLEDGEMENT

The authors would like to thank to Research Project Tecnologías Avanzadas de Bases de Datos 22/F614 financed by Universidad Nacional de San Luis, San Luis, Argentina; Project AL09-PAC-12 financed by Universidad Politécnica de Madrid, Madrid, España; and, Instituto de Física Aplicada, Universidad Nacional de San Luis - CONICET, San Luis, Argentina, to allow us to use the cluster. Likewise, to the colleagues who contributed with opinions and technical advice.

REFERENCES