A Hybrid Memory-based ACO algorithm for the QAP

Guillermo Leguizamón Franco Arito Carlos A. Coello Coello

***Abstract*— The performance of ant colony optimization (ACO) algorithms significantly improves when hybridized with local search procedures which strongly bias the search towards promising regions of the search space. In this work, we study a recently proposed $\mathcal{M}$emory based ACO algorithm ($\mathcal{M}$-ACO) which incorporates some tabu search principles into the solution construction process. This algorithm has also been hybridized with two local search procedures: 2-opt ($\mathcal{M}$-ACO-2opt) and Tabu Search ($\mathcal{M}$-ACO-TS). The performances of the two hybrid versions of $\mathcal{M}$-ACO are analyzed on a set of instances of the Quadratic Assignment Problem (QAP). The results show that the hybrid versions of $\mathcal{M}$-ACO are able to improve the quality of the best known solutions for several of the instances studied.**

I. Introduction

Ant colony optimization (ACO) algorithms generate solutions for an optimization problem through a construction mechanism in which the selection of the solution component to be added at each step is probabilistically influenced by pheromone trails and (in most cases) by heuristic information [6] from which a probabilistic model is evolved to better explore the search space. Accordingly, this construction process probabilistically builds, step by step, the problem solutions. However, a solution construction process can be designed such that the pheromone trails, heuristic values, or any other source of information, can be used to deterministically determine the next problem component to be added to the solution under construction. The proposals of Acan [1], [2], Tsutsui [14] and Wiesemann & Stützle [15] constitute some examples of the use of an external memory as an alternative selection mechanism for the solution components. More recently, Arito and Leguizamón [3] studied the possibility of alternating the way in which the ants select the next solution component. This was done by introducing an external memory as an *auxiliary mechanism* to help in the process of deciding the next problem component to be chosen at each step of the solution construction process.

Such a proposal was inspired by the well-known optimization technique Tabu Search (TS) [7].

The memory-based ACO proposed in [3] included a deterministic mechanism represented by different memory structures that allow the ants (alternatively to the traditional probabilistic approach) to choose solution components in a deterministic way, influenced by the values stored in this memory.

As this memory stores specific information about the search history since the beginning of the algorithm's execution, it allows to focus into non-visited regions of the search space (i.e., regions not yet registered in the memory), while also concentrating on already visited and promising regions (i.e., registered regions). Different sources and combinations of information were considered in this early work, which allowed us to apply alternative intensification/diversification mechanisms in order to avoid premature convergence and improve the algorithm's performance with respect to a well-known ACO algorithm that has been previously applied to the quadratic assignment problem (QAP), the $\mathcal{MAX}-\mathcal{MIN}$ Ant System ($\mathcal{MM}$AS-QAP) algorithm [9], [11].

Hereafter, the paper is organized as follows. Section II presents the QAP. In Section III, we provide a general outline of the variants of the memory-based algorithm presented in [3] and describe the specific variant adopted in this work. The two hybrid versions studied here are shortly described in Section IV. Section V reports the results of the application of the hybrid memory-based ACO algorithms to a set of well-known instances of the QAP including a comparison with the best known solutions for the set of instances considered. Finally, Section VI presents some relevant conclusions of our study and a discussion about future research directions.

II. Quadratic Assignment Problem

The QAP is an $\mathcal{NP}$-hard problem [8], which is usually formulated as the problem of assigning a set of objects to a set of locations with given distances between the locations and given flows between the objects. The objective is to achieve an optimal assignment of these objects to locations in such a way that the sum of the product between flows and distances is minimal:

$$\min_{\phi\in\Phi}\sum_{i=1}^{n}\sum_{r=1}^{n}a_{\phi_i\phi_r}b_{ir}$$

where n is the number of objects and locations. Two $n\times n$ matrices $A=[a_{ir}]$ and $B=[b_{js}]$, represent, respectively, the distance between locations i and r (a_{ir}) and the flow between objects j y s (b_{js}). $\Phi(n)$ is the set of all possible permutations in $\{1,\ldots,n\}$, and ϕ_j gives the location of object j in the current solution $\phi\in\Phi(n)$.

It is worth remarking that a number of real world applications, such as VLSI module placement, scheduling, man-

Guillermo Leguizamón is with LIDIC - Universidad Nacional de San Luis, San Luis, Argentina and with the UMI LAFMIA 3175 CNRS at CINVESTAV-IPN, Departamento de Computación, Av. IPN No. 2508. Col. San Pedro Zacatenco México D.F. 07300, MÉXICO, email: legui@unsl.edu.ar.

Franco Arito is with LIDIC - Universidad Nacional de San Luis, San Luis, Argentina, Av. Ejército de Los Andes 950 (5700), San Luis, Agentina, email:farito@unsl.edu.ar.

Carlos A. Coello Coello is with CINVESTAV-IPN (Evolutionary Computation Group), Departamento de Computación. Av. IPN No. 2508. Col. San Pedro Zacatenco México D.F. 07300, MÉXICO, email: ccoello@cs.cinvestav.mx. He is also affiliated at the UMI LAFMIA 3175 CNRS at CINVESTAV-IPN.

ufacturing, process communications, statistical data analysis, hospital layout, among many others, as well as many combinatorial optimization problems such as the Traveling Salesman Problem (TSP), the Maximum Clique Problem (MCP), and the Graph Partitioning Problem (GPP) can be formulated as instances of the QAP [4]. This reflects the high importance of this problem, and has motivated the design of a wide variety of metaheuristics to solve it.

III. A MEMORY-BASED ACO ALGORITHM

Our memory-based ACO algorithm proposed and studied in [3] shares some characteristics with the $\mathcal{MAX}-\mathcal{MIN}$ Ant System ($\mathcal{MM}$AS-QAP) algorithm [9], [11], which is considered the best performing ACO algorithm for the QAP currently available [12]. The solutions in $\mathcal{MM}$AS-QAP are constructed by assigning at each construction step an element to some location. Pheromone trail τ_{ij} indicates that we want to assign element j to location i. $\mathcal{MM}$AS-QAP does not use any heuristic information in the solution construction procedure. The pheromone updating is done by lowering the pheromone trails by a constant factor ρ and depositing pheromone on the individual solution components of either: i) the best solution in the current iteration, ii) the best solution found so far by the algorithm, or iii) the best solution found since the last re-initialization of the pheromone trails. More precisely, in the original $\mathcal{MM}$AS-QAP two ways of constructing solutions were proposed. The first one makes at each construction step a probabilistic choice similar to the rule in the Ant System algorithm [6], whereas the second one uses the pseudo-random proportional action choice rule similar to the one adopted by the Ant Colony System [5].

In our algorithm. we make use of a mechanism similar to the second one mentioned above (further explained in this paper). Additionally, our approach does not use at all the pheromone reinitialization process.

Similarly to the approach followed in $\mathcal{MM}$AS-QAP in which a local search procedure is applied for improving each candidate solution generated by the ants, we used in our early work [3] an iterative improvement algorithm (the *2−exchange neighborhood*) where two candidate solutions are neighbors if they differ in the assignment of exactly 2 units to locations. This local search procedure uses a best-improvement pivoting rule.

In the following, we present the general design of the memory-based ACO algorithm for QAP. However, for the experimental study presented here, we have chosen one of the variants presented in [3] as will be explained at the end of this section.

A. Memory structures

The memory structures in Tabu Search operate by reference to four main dimensions: recency, frequency, quality, and influence. Our approach makes use of two of them: recency and frequency. Next, we explain how they are used in our memory-based ACO algorithm.

1) Recency based memory: This type of memory stores components of the solutions that have changed in the recent past. The usual way of exploiting this type of memory is labeling the selected components of recently visited solutions. What this approach is trying to achieve is to respectively "forbid" or "encourage" certain choices that prevent us from exploring a larger region of the search space by concentrating on a particular region based on solutions already visited in the recent past.

It is important to have in mind that for some instances, a good search process would result in visiting again a previously found solution. Thus, this mechanism aims at continuously stimulating the discovery of new solutions of high quality.

For the QAP, the recency based memory stores the iteration at which the algorithm[1] assigned object j to location i, for $1 \leq i, j \leq n$. First, this memory allows the ants to make decisions taking into account what objects are the *most recently* assigned to a particular location since they have an associated value close to the current iteration of the algorithm. Second, it allows to make decisions taking into account what objects are the *least recently* assigned to a particular location since they have an associated value far from the current iteration of the algorithm.

When the memory-based ACO algorithm takes into account the *most recently* assigned objects to a particular location, it maintains a $n \times n$ recency based matrix called `recency`, where `recency[i,j]` stores the most recent iteration (the last one) at which the object i has been assigned to the location j during the execution of the algorithm. Therefore, this matrix is used in the process of solution construction in two possible ways:

- **Intensification:** choose the object that was the most recently assigned to the current location (i.e., if the current location is i, the object j is chosen such that `recency[i,j]` has the highest iteration number).
- **Diversification:** choose the object that was the least recently assigned to the current location (i.e., if the current location is i, the object j is chosen such that `recency[i,j]` has the lowest iteration number).

2) Frequency based memory: This type of memory stores components of the solutions that appear more frequently in a solution (i.e., it accounts for the number of times that a component is either present in a solution or in a specific position of the solution). The usual way of exploiting this type of memory is by labeling the selected components of the most frequently chosen solutions. This memory allows to "forbid" that an ant chooses a solution component when it has been frequently chosen in the previous solutions. Thus, the "prohibition" aims at generating solutions that indeed differ from those already generated. In this way, the exploration of the search space is extended. Inversely, this information can be used to "promote" their selection since

[1]This value is not the iteration value of the algorithm itself, but it is a value computed according to the number of ants and to the current iteration number.

most of the ants have chosen them as part of their solutions and therefore they can be considered as *desirable* members for a new solution.

For the QAP, the frequency based memory stores the times that object j has been assigned to location i, for $1 \leq i, j \leq n$. Then, this memory allows the ants to make decisions taking into account those elements that were the *most frequently* assigned to a particular location because they have a high assignment frequency associated to them. On the other hand, it allows to make decisions taking into account those elements that were the *least frequently* assigned to a particular location, since they have the lowest assignment frequency associated to them.

When the memory-based ACO algorithm takes into account the *most recently* assigned objects to a particular location, it maintains a $n \times n$ frequency-based matrix called `frequency`, in which the `frequency[i,j]` stores the number of times that the object j has been assigned to the location i during the execution of the algorithm. Then, this matrix is used in the process of constructing a solution in two possible ways:

- **Intensification:** choose the object that was assigned the highest number of times to the current location (i.e., if the current location is i, the chosen object j is that for which `frequency[i,j]` is the highest).
- **Diversification:** choose the object that was assigned the lowest number of times to the current location (i.e., if the current location is i, the chosen object j is that for which `frequency[i,j]` is the lowest).

B. Constructing Solutions

It must be noticed that during the solution construction process for QAP, the ants assign each object exactly to one location and no location is used by more than one object. Thus, each constructed solution corresponds to a permutation $\phi \in \Phi(n)$. The solution construction process involves two steps. In the first step, a location is chosen and then, in the second step, an object is assigned to that location. To do that, we randomly choose a location i among those not yet occupied. For the second step, we use the pheromone trails, τ_{ij} referring to the desire of assigning the object j to the location i. To assign an object j to an unoccupied location i we use the following rule:

$$j = \begin{cases} T & \text{if } q < q_0 \text{ (Exploitation)} \\ R & \text{if } q \geq q_0 \text{ (Exploration)} \end{cases} \quad (1)$$

This rule is similar to the one used by the Ant Colony System [6]: with a fixed probability q_0 ($0 \leq q_0 \leq 1$) the ant chooses the "best possible element" according to the acquired knowledge (it can be based on the external memory or based on the pheromone trails). With probability $(1-q_0)$, it is carried out a controlled exploration of new solutions (also in this case, it can be based on the external memory or based on the pheromone trails), where q is a random number uniformly distributed in the interval $[0, 1]$. T and R are random variables with a probability distribution given by equations (2) and (3), respectively as explained in the following.

To promote exploitation, we used the rule:

$$T = \begin{cases} arg\max_{l \in \mathcal{N}_i^k}\{\texttt{memory[i,l]}\} & \text{if } r < r_0 \\ arg\max_{l \in \mathcal{N}_i^k}\{\tau_{il}\} & \text{if } r \geq r_0 \end{cases} \quad (2)$$

In this rule, with a fixed probability r_0 ($0 \leq r_0 \leq 1$) the chosen object is the one that was assigned the highest number of times to the current location (`memory = frequency`) or that was most recently assigned to the current location (`memory = recency`). Also, with a probability $(1 - r_0)$, the most desirable object is chosen according to the pheromone trails. Variable r is a random number uniformly distributed in the interval $[0, 1]$ and $\mathcal{N}_i^k$ is the set of still unassigned elements for the ant k, that is, those elements that are still to be assigned to location i.

To promote exploration, we used the rule:

$$R = \begin{cases} arg\min_{l \in \mathcal{N}_i^k}\{\texttt{memory[i,l]}\} & \text{if } p < p_0 \\ \frac{\tau_{ij}(t)}{\sum_{l \in \mathcal{N}_i^k} \tau_{il}(t)} & \text{if } p \geq p_0 \end{cases} \quad (3)$$

In this rule, with a fixed probability p_0 ($0 \leq p_0 \leq 1$), we choose the element that was assigned the lowest number of times to the current location (`memory=frequency`) or the element that was the least recently assigned to the current location (`memory=recency`). Also, with probability $(1 - p_0)$, we choose that element according to the basic selection rule similar to the rule of the Ant System algorithm (notice that in this case we do not use heuristic information at all). Variable p is a uniformly distributed random number in the interval $[0, 1]$.

In the experimental study conducted in [3], we considered 4 variants of the memory-based ACO algorithm which were respectively called $\mathcal{MM}$AS-ff, $\mathcal{MM}$AS-fr, $\mathcal{MM}$AS-rf, and $\mathcal{MM}$AS-rr according to the way of combining the different memories—i.e., the frequency-based memory (equation (2)) and recency-based memory (equation (3)). Thus, $\mathcal{MM}$AS-fr stands for the variant that uses the frequency-based matrix in equation (2) and the recency-based matrix in equation (3). A similar reasoning applies to the remaining algorithms' names.

As a first result, we found the all these variants outperformed to $\mathcal{MM}$AS-QAP in the instances considered (this could be statistically corroborated). In addition, we found that $\mathcal{MM}$AS-rf was the best performer among the 4 variants of the memory-based ACO algorithm (this was also statistically corroborated). From these results we chose $\mathcal{MM}$AS-rf to conduct the experimental study presented here. In the rest of this paper, we will use the name $\mathcal{M}$-ACO to refer to the algorithm $\mathcal{MM}$AS-rf. In order to

improve its performance, in the following, we present an alternative for hybridizing $\mathcal{M}$-ACO which will be applied in our experimental study.

IV. $\mathcal{M}$-ACO AND ITS TWO HYBRID VERSIONS

This section presents a simple alternative to hybridize $\mathcal{M}$-ACO by using a more powerful local search procedure. In our case, we have chosen Tabu Search (TS) to apply it to the solutions found at each iteration of $\mathcal{M}$-ACO. TS has shown to be a powerful metaheuristic technique to solve many combinatorial optimization problems. Indeed, Stützle and Fernandes [10] used this approach to obtain a new benchmark[2] for the set of instances specially created to assess the dependence of the performance of metaheuristics on different instance characteristics as further explained in the next section.

TS (as a local search procedure) is applied to each solution for a very small number of iterations in order to avoid that TS behaves as the most important search engine in the exploration of the search space (i.e., to avoid an excessive bias of TS in $\mathcal{M}$-ACO). According to this, we obtained a highly hybridized algorithm in which the exploration and exploitation is achieved by the the ant colony and an additional exploitation process is performed by the local search procedure. Thus, we have two hybrid versions of $\mathcal{M}$-ACO: (1) the one obtained by using *2−exchange neighborhood* ($\mathcal{M}$-ACO-2opt) which corresponds to $\mathcal{MM}$AS-rf as presented in [3] and (2) another one obtained by applying TS (specifically RoTS[3] [13]) as a local search procedure ($\mathcal{M}$-ACO-TS).

A general outline of the proposed algorithm is presented in Algorithms 1 and 2. In Algorithm 2 is presented, regarding the QAP, the outline of the solution construction process through the use of the external memory and pheromone values.

Algorithm 1 Hybrid Memory-based ACO algorithm

```
Initialize pheromone trails
Initialize external memory
{ Main loop }
while termination conditions not met do
   ConstructAntsSolutions
   ApplyLocalSearch { 2-opt or Tabu Search }
   UpdatePheromones
   UpdateMemory { Recency or Frecuency based }
end while
```

Algorithm 2 ConstructAntsSolutions {for the QAP}

```
for k = 0 to number of ants do
   Generate a random order of location's assignment
   for step = 0 to n do
      if q < q_0 then
         if r < r_0 then
            Choose the object more recently or more frequently assigned to the current location.
            Assign the chosen object to that location.
         else
            Choose the maximum pheromone matrix value for the current location.
            Assign the chosen object (represented by the matrix value) to that location.
         end if
      else
         if p < p_0 then
            Choose the less recently or less frequently object assigned to the current location.
            Assign the chosen object to that location.
         else
            Choose the object j to be assigned to location i by the probabilistic rule:
                                  τ_ij(t) / Σ_{l∈N_i^k} τ_il(t)
         end if
      end if
   end for
end for
```

V. COMPUTATIONAL STUDY

We tested $\mathcal{M}$-ACO-2opt and $\mathcal{M}$-ACO-TS on a set of QAP instances proposed by Stützle and Fernandes [10] of size 50 (we chose 16 instances from each of the four classes). This set of instances was generated in such a way that (i) the characteristics of the instances are systematically varied and (ii) they are large enough to allow systematic studies on the dependence of the performance of the metaheuristics on the different instance characteristics. It is worth noticing that this new benchmark contains the best known solution values for each instance, and many of them are not necessarily the optimal ones. The tested instances include:

- **GridRandom** (GR): Grid-based distance matrix and random flows;
- **GridStructured** (GS): Grid-based distance matrix and structured flows;
- **GridStructuredPlus** (GSP): Grid-based distance matrix and structured flows with connections among clusters of objects.
- **RandomRandom** (RR): Random distance matrix and random flows;
- **RandomStructured** (RS): Random distance matrix and structured flows;
- **RandomStructuredPlus** (RSP): Random distance matrix and structured flows with connections among clusters of objects;

All the experimental results were measured across 30 independent trials of the algorithms and the code was run on an Intel Pentium (R) 4, CPU 3.00Gz, and 1Gb RAM; OS

[2]It was obtained from the corresponding authors upon request.

[3]A re-implementation in C of RoTS was obtained from the Metaheuristics Networks.

Linux. For all the instances we ran both algorithms until they completed 500,000 function evaluations. This number of function evaluations was chosen in order to further compare our results (Section V-A) with the respective benchmark obtained by applying RoTS with the same number of function evaluations.

We compared the algorithms using the average percentage excess (%Error) over the best-known solutions (as mentioned before, they were provided to us by the authors of [10]). In order to assess the statistical significance of the observed differences in the algorithms' performance we applied the non-parametric Mann-Whitney-Wilcoxon test.

Preliminary experimentation was conducted in order to detect the regions of parameters values that produced the best performance. Particularly, we considered the parameters q_0, r_0, p_0, and ρ; all of them varying in the range $(0, 1)$. In order to do that, we used Latin Hypercube Sampling (LHS) to obtain a number of design points in a space filling way and considering a subset of instances randomly selected. From these experiments, we found the following setting for the parameters considered: $q_0 = 0.2$, $r_0 = 0.7$, $p_0 = 0.001$, and $\rho = 0.55$. The number of ants was set to 25, which is a similar value to the one used in [3]. In the case of $\mathcal{M}$-ACO-TS, we applied RoTS for 200 iterations each time that this local search procedure takes place. See Algorithms 1 and 2 for details.

A. Comparison of $\mathcal{M}$-ACO under two hybridization approaches

This section shows the results obtained by $\mathcal{M}$-ACO-2opt and $\mathcal{M}$-ACO-TS for the selected instances. In fact, we chose the complete set of instances for classes GS, GSP, RS, and RSP, except for RR and GR for which 16 instances were selected out of the 36 available for each class. We present the results for each class condensed in Figures 1, 2, 3, 4, 5, and 6 that respectively represent, through boxplots, the percentage error with respect the current benchmark values. Each boxplot displays, for $\mathcal{M}$-ACO-TS (left) and $\mathcal{M}$-ACO-2opt (right), a sample of 16×30 percentage error values corresponding to 16 instances belonging to each class and 30 trials for each of these instances. The caption of each figure includes the respective p-value, which shows in all cases a large statistical significant difference. They clearly indicate that $\mathcal{M}$-ACO-TS outperforms $\mathcal{M}$-ACO-2opt in all the instances considered. Let us analyze first the behavior of $\mathcal{M}$-ACO-TS. It can be readily observed a very robust behavior across the whole set of instances (in the 6 classes). Particularly, for classes RS (Figure 1) and GS (Figure 2), the shape of the respective boxplots indicate that in almost all cases the best known value was reached. However, when the best known values were not achieved, the algorithm still performs well since it gives values (see the outliers) that are below the 3% (RS) and 2% (GS) from the best known. For the same classes, $\mathcal{M}$-ACO-2opt gives also high quality solutions, although it shows a less robust behavior as well as solutions of lower quality. Although the situation is similar when considering the robustness observed above for both algorithms with respect to the remaining instances (classes RS, RSP, GS, and GSP), it can be observed here that the $\mathcal{M}$-ACO-TS algorithm was capable of improving the best known values for at least one instance of each class (in the next section we show each of the improved results). On the side of $\mathcal{M}$-ACO-opt, it can be seen that it was only capable of improving the best values of the instance of class RSP (see Figure 4). It must be recalled that in [3] it was reported one improved result for one instance of this class.

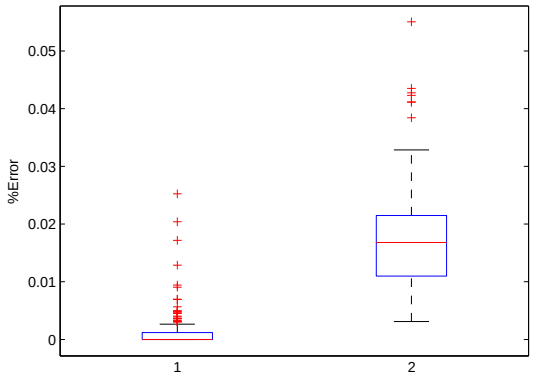

Fig. 1. Random Random (p-value = 8.3723e-162)

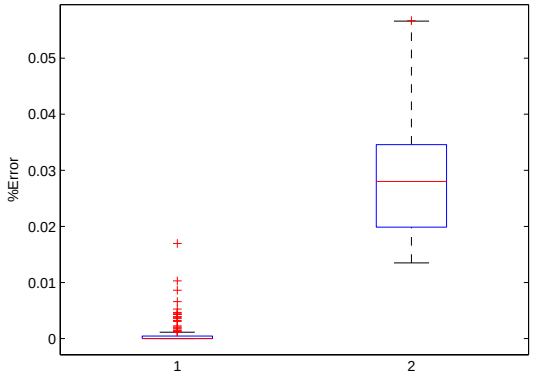

Fig. 2. Grid Random (p-value = 2.9541e-153)

B. Comparison of $\mathcal{M}$-ACO -TS with the QAP benchmark

In order to explicitly show the new best known results, we present in this section a short comparison of the results obtained from $\mathcal{M}$-ACO-TS and the up-to-date best known results for the set of instances considered. Before analyzing the improved results, it should be noticed that for all the remaining instances not shown here, $\mathcal{M}$-ACO-TS was capable to achieve the corresponding best known values.

Table I shows the particular instances for which $\mathcal{M}$-ACO-TS lowered the benchmarks. The improved instances are

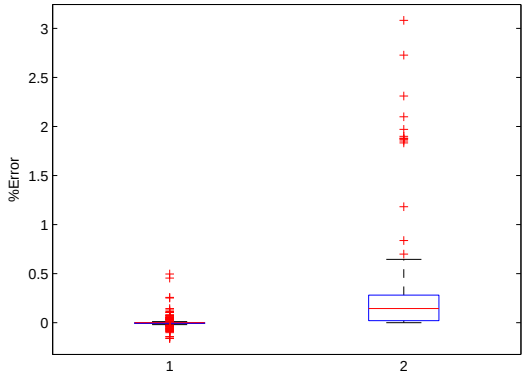

Fig. 3. Random Structured (p-value = 8.6968e-119)

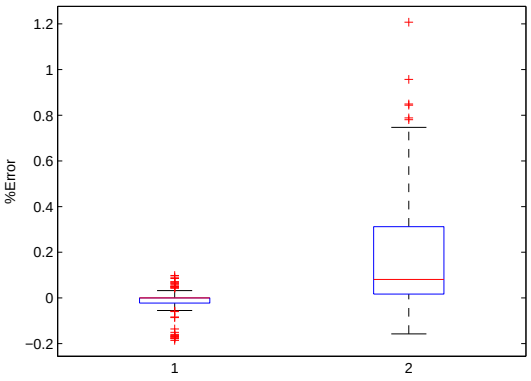

Fig. 4. Random Structured Plus (p-value = 4.5455e-084)

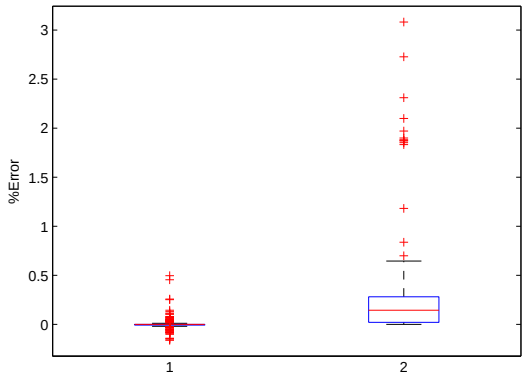

Fig. 5. Grid Structured (p-value = 8.6968e-119)

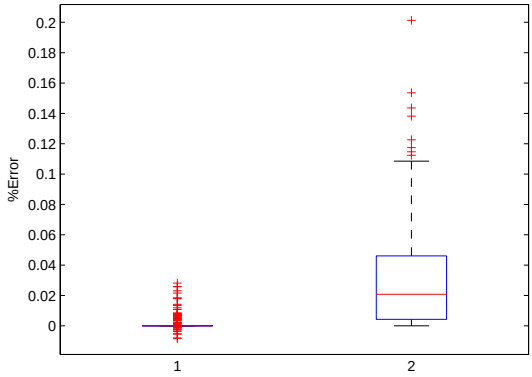

Fig. 6. Grid Structured Plus (p-value = 1.7168e-101)

TABLE I

COMPARISON OF THE IMPROVED RESULTS OF $\mathcal{M}$-ACO WITH RESPECT TO THE BEST-KNOWN SO FAR. DUE TO SPACE CONSTRAINTS, THE INSTANCES ARE IDENTIFIED BY A REFERENCE NAME. THE CORRESPONDING ORIGINAL NAMES CAN BE FOUND IN TABLE II (SEE THE APPENDIX).

Instance	BF (med(BF))	avg(FES)	BK	FES	PoI
rs1	15574 (16643)	26400	16107	251617	3.30%
rs5	5218 (5230)	199720	5427	468518	3.80%
rs9	3672 (3739)	138623	4088	182257	10.17%
rs10	75170 (77577)	143600	76534	213452	1.78%
rs13	4883 (4883)	193200	5824	83812	16.15%
rs14	48875 (48875)	118800	49366	251110	0.99%
rsp1	7053 (7383)	247200	7646	359448	7.75%
rsp5	3913 (4094)	228000	4153	283311	5.53%
rsp9	3675 (3775)	240300	4513	169387	18.56%
rsp10	94804 (94804)	56520	100341	133321	5.51%
rsp13	1410 (1525)	161320	1607	151734	12.58%
rsp14	50907 (53130)	74800	50958	369209	0.10%
gsp10	11250 (11292)	14920	11276	26328	0.23%
gsp14	5598 (5640)	196280	5645	168692	0.83%
gs2	25914 (26048)	122520	25986	428576	0.27%
gs10	9126 (9237)	179200	9152	171611	0.28%
gs14	4729 (4745)	200000	4747	415116	0.37%

identified by a reference name (first column). The corresponding original names can be found in Table II (see the Appendix).

For each instance, Table I shows: the Best Found (BF) value and the median of the best found values out of 30 trials (in parentheses), the average number of function eval-

uations required to reach the best found values (avg(FES)), the Best Known (BK) value, the corresponding number of iterations (or function evaluations) required to reach that value obtained by RoTS (this information can be found in the benchmark), and the percentage of improvement (PoI).

It can be clearly observed a large variation on the values in column `PoI` where the values range from 0.1% to 18.56%. The larger improvements were achieved for instances of classes RS and RSP, i.e., the Random Structured and Random Structured Plus. Regarding the number of function evaluations, the results show that in general, $\mathcal{M}$-ACO-TS was able to reduce the number of visited solutions required to reach the best found values with respect to the numbers reported in the benchmark. However, it should be noticed that we report the average values out of 30 algorithm trials whereas in the benchmark is reported the number of function evaluations necessary to reach the best found value out of 10 trials.

VI. Conclusions

In this paper, we presented the study of two alternative local search procedures to improve the performance of an early proposed memory-based ACO algorithm [3] which showed to be competitive with respect to a state-of-the-art ant algorithm for the QAP. In this work, we have shown, through the validation on an important number of QAP instances, that an ACO algorithm can be easily combined with a powerful technique such as TS to produce a hybrid approach which presents an improved performance with respect to the combined technique working alone.

As part of our future work, we will perform a study focused on developing an advanced strategy to determine *when* and *how many* iterations should be applied of a local search procedure in order to reach an appropriate balance between *intensification* (which is incorporated by the local search process and some components of $\mathcal{M}$-ACO-TS) and *exploration* (which is incorporated by some other components of $\mathcal{M}$-ACO-TS). Furthermore, additional experimental studies are also necessary that consider an extended number and types of QPA instances as well as a comparison with other state-of-the-art hybrid ACO algorithms for QAP.

VII. Acknowledgments

The first author acknowledges the support from the UMI-LAFMIA 3175 CNRS at CINVESTAV-IPN and from the Universidad Nacional de San Luis, Argentina. The second author acknowledges the support from the Universidad Nacional de San Luis, Argentina. The third author gratefully acknowledges support from CONACyT project no 103570. All the authors wish to thank Prof. Thomas Stützle for providing the source code of ACOTSP Version 1.0, which is available online[4], and on which the algorithm reported in this paper is based.

TABLE II
References and complete names of the 17 improved QAP instances

Reference Name	Original Name
rs1	RandomStructured.974823931.n50. K10.m10.A100.00.B1.00.sp10.00.dat
rs5	RandomStructured.974823935.n50. K10.m10.A10.00.B2.00.sp10.00.dat
rs9	RandomStructured.974823939.n50. K10.m10.A4.00.B3.50.sp10.00.dat
rs10	RandomStructured.974823940.n50. K10.m10.A4.00.B3.50.sp20.00.dat
rs13	RandomStructured.974823943.n50. K10.m10.A2.00.B7.00.sp10.00.dat
rs14	RandomStructured.974823944.n50. K10.m10.A2.00.B7.00.sp20.00.dat
rsp1	RandomStructuredPlus.974824391.n50. K10.m10.A100.00.B1.00.sp10.00.dat
rsp5	RandomStructuredPlus.974824395.n50. K10.m10.A10.00.B2.00.sp10.00.dat
rsp9	RandomStructuredPlus.974824399.n50. K10.m10.A4.00.B3.50.sp10.00.dat
rsp10	RandomStructuredPlus.974824400.n50. K10.m10.A4.00.B3.50.sp20.00.dat
rsp13	RandomStructuredPlus.974824403.n50. K10.m10.A2.00.B7.00.sp10.00.dat
rsp14	RandomStructuredPlus.974824404.n50. K10.m10.A2.00.B7.00.sp20.00.dat
gsp10	GridStructuredPlus.974826016.n50. G10.A4.00.B3.50.sp20.00.dat
gsp14	GridStructuredPlus.974826020.n50. G10.A2.00.B7.00.sp20.00.dat
gs2	GridStructured.974825926.n50. G10.A100.00.B1.00.sp20.00.dat
gs10	GridStructured.974825934.n50. G10.A4.00.B3.50.sp20.00.dat
gs14	GridStructured.974825939.n50. G10.A2.00.B7.00.sp20.00.dat

Appendix

References

[1] A. Acan. An external memory implementation in ant colony optimization. In M. Dorigo, M. Birattari, C. Blum, L. M. Gambardella, F. Mondada, and T. Stützle, editors, *ANTS 2004*, volume 3172 of *LNCS*, pages 73–84. Springer, Heidelberg, 2004.

[2] A. Acan. An external partial permutations memory for ant colony optimization. In G. Raidl and J. Gottlieb, editors, *Evolutionary Computation in Combinatorial Optimization*, volume 3448 of *LNCS*, pages 1–11. Springer-LNCS, 2005.

[3] F. Arito and G. Leguizamón. Incorporating Tabu Search Principles into ACO Algorithms. In Maria J. Blesa, Christian Blum, Luca Di Gaspero, Andrea Roli, Michael Sampels, and Andrea Schaerf, editors, *Hybrid Metaheuristics*, volume 5818 of *Lecture Notes in Computer Science*, pages 130–140. Springer, 2009.

[4] E. Cela. *The Quadratic Assignment Problem: Theory and Algorithms*. Kluwer Academic Publisher, Dordrecht, The Netherlands, 1998.

[5] M. Dorigo and L. M. Gambardella. Ant Colony System: A cooperative learning approach to the traveling salesman problem. *IEEE Transactions on Evolutionary Computation*, 1(1):53–66, 1997.

[6] M. Dorigo and T. Stützle. *Ant Colony Optimization*. MIT Press, Cambridge, MA, 2004.

[7] F. Glover and M. Laguna. *Tabu Search*. Kluwer Academic Publishers, Norwell, MA, USA, 1997.

[8] S. Sahni and T. Gonzalez. P-complete approximation problems. *Journal of the ACM*, 23(3):555–565, 1976.

[9] T. Stützle. $\mathcal{MAX}$ – $\mathcal{MIN}$ for the quadratic assignment problem. Technical report, AIDA-97-4, FG Intellektik, FB Informatik, TU Darmstadt, Germany, 1997.

[4] `http://www.aco-metaheuristic.org/aco-code`

[10] T. Stützle and S. Fernandes. New benchmark instances for the QAP and the experimental analysis of algorithms. In J. Gottlieb and G. R. Raidl, editors, *Evolutionary Computation in Combinatorial Optimization: 4th European Conference, EvoCOP 2004*, volume 3004 of *Lecture Notes in Computer Science*, pages 199–209, Berlin, Germany, 2004. Springer-Verlag.

[11] T. Stützle and H. Hoos. $\mathcal{MAX} - \mathcal{MIN}$ ant system. *Future Generation Computer Systems*, 16(8):889–914, 2000.

[12] Thomas Stützle and Marco Dorigo. Aco algorithms for the quadratic assignment problem. In D. Corne, M. Dorigo, and F. Glover, editors, *New Ideas in Optimization*, pages 33–50, London, UK, 1999. McGraw-Hill.

[13] É. D. Taillard. Robust taboo search for the quadratic assignment problem. *Parallel Computing*, 17:443–455, 1991.

[14] S. Tsutsui. cAS: Ant colony optimization with cunning ants. In T. P. Runarsson *et al.*, editor, *Proc. of the 9th Int. Conf. on Parallel Problem Solving from Nature (PPSN IX)*, volume 4193 of *LNCS*, pages 162–171. Springer, Heidelberg, 2006.

[15] W. Wiesemann and T. Stützle. Iterated ants: An experimental study for the quadratic assignment problem. In M. Dorigo, L. M. Gambardella, M. Birattari, A. Martinoli, R. Poli, and T. Stützle, editors, *ANTS 2006*, volume 4150 of *LNCS*, pages 179–190. Springer, Heidelberg, 2006.