Integrating Evolutionary Computation Components in Ant Colony Optimization

S. Alonso, O. Cordón, I. Fernández de Viana, F. Herrera

Sergio Alonso
Department of Computer Science and Artificial Intelligence
E.T.S.I. Ingeniería Informática
University of Granada
c/ Daniel Saucedo Aranda, s/n
18071 - Granada, Spain
E-mail: zerjio@zerjio.com

Oscar Cordón
Department of Computer Science and Artificial Intelligence
E.T.S.I. Ingeniería Informática
University of Granada
c/ Daniel Saucedo Aranda, s/n
18071 - Granada, Spain
Phone: +34-958-246143
Fax: +34-958-243317
E-mail: ocordon@decsai.ugr.es

Iñaki Fernández de Viana
Dpto. de Electrónica Sistemas Informáticos y Automática (IESIA)
Universidad de Huelva
Carretera Huelva - La Rábida
21071 Palos de la Frontera (Huelva), Spain
Phone: +34-959-017371
E-mail: ijfviana@ugr.es

Francisco Herrera
Department of Computer Science and Artificial Intelligence
E.T.S.I. Ingeniería Informática
University of Granada
c/ Daniel Saucedo Aranda, s/n
18071 - Granada, Spain
Phone: +34-958-240598
Fax: +34-958-243317
E-mail: herrera@decsai.ugr.es

This work has been already published:

S. Alonso, O. Cordón, I. Fernández de Viana, F. Herrera, **Integrating Evolutionary Computation Components in Ant Colony Optimization Evolutionary Algorithms: An Experimental Study**. *Recent Developments in Biologically Inspired Computing, L. Nunes de Castro, F.J. Von Zuben (Eds.), Idea Group Publishing, 2004, 148-180*

Integrating Evolutionary Computation Components in Ant Colony Optimization*

ABSTRACT

This chapter introduces two different ways to integrate Evolutionary Computation Components in Ant Colony Optimization (ACO) Metaheuristic. First of all ACO metaheuristic is introduced and compared to Evolutionary Computation to notice their similarities and differences. Then two new models of ACO algorithms that include some Evolutionary Computation concepts (Best-Worst Ant System and exchange of memoristic information in parallel ACO algorithms) are presented with some empirical results, that show improvements in the quality of the solutions when compared with more basic and classical approaches.

KEYWORDS

Heuristics, Algorithms, Ant Colony Optimization, Evolutionary Algorithms, PBIL, Parallel Evolutionary Algorithms,Travelling Salesman Problem.

TABLE OF CONTENTS

INTRODUCTION

Complex combinatorial optimization problems arise in many different fields such as economy, commerce, engineering or industry. These problems are so complex that there is no algorithm known that solves them in polynomial time (Garey & Johnson, 1979). These kinds of problems are called *NP-hard*.

Still, many of these problems have to be solved in a huge number of practical settings and therefore a large number of algorithmic approaches were proposed to tackle them. The existing techniques can roughly be classified into exact and approximate algorithms. *Exact algorithms* try to find an optimal solution and to prove that the solution obtained is actually an optimal one. These algorithms include techniques such as backtracking, branch and bound, dynamic programming, etc. (Papadimitriou & Steiglitz, 1982) (Brassard & Bratley, 1996). Because exact algorithms show poor performance for many problems, several types of approximate algorithms were developed that provide high quality solutions to combinatorial problems in short computation time.

Approximate algorithms can be classified into two main families: deterministic and probabilistic. Deterministic algorithms always produce the same solution when the starting conditions are the same, while the later algorithms are characterized by a non deterministic behaviour, that is, for a specific problem and in the same execution conditions (same seeds from the random number generators, same values of the different parameters, same number of iterations, and so on), they return different solutions.

A different classification for approximate algorithms distinguishes between *construction algorithms* and *local search algorithms*. The former are based on generating solutions from scratch by adding solution components step by step. The best known example are greedy construction heuristics (Brassard & Bratley, 1996). Their advantage is speed: they are typically very quick and, in addition, often return reasonably good solutions. However, these solutions are not guaranteed to be optimal with respect to small local changes. Local search algorithms repeatedly try to improve the current solution by movements to (hopefully better) neighbouring solutions. The simplest case are iterative improvement algorithms: if in the neighbourhood of the current solution s, a better solution s' is found, it replaces the current solution and the search is continued from s'; if no better solution is found, the algorithm terminates in a local optimum. Nowadays, hybridizations of both techniques are usually used: any construction algorithm builds an initial solution which is then improved by a local search algorithm.

Unfortunately, iterative improvement algorithms may become stuck in poor quality local optima. To allow them for a further improvement in solution quality, in the last two decades the research in this field has moved attention to the design of general-purpose techniques for guiding underlying, problem-specific construction or local search heuristics. These techniques are called Metaheuristics (Glover & Kochenberger, 2003). They involve concepts that can be used to define heuristic methods, that is, metaheuristics can be seen as a general algorithmic framework which can be applied to different (combinatorial) optimization problems with relatively few modifications if given some underlying, problem specific heuristic method. In fact, metaheuristics are now widely recognized as the most promising approaches for attacking hard combinatorial optimization problems (Aarts & Lenstra, 1997) (Michalewicz & Fogel, 2000) (Reeves, 1995).

Heuristics Based on Nature or *Bioinspired Algorithms* (Colorni, A., Dorigo, M., Maffioli, F., Maniezzo, V., Righini, G. & Trubian M., 1996) are approximate algorithms that have achieved good results. All of them share at least one quality: they operate simulating some natural processes, although some of them have evolved in order to increase their effectiveness. However, these improvements sometimes include some aspects that do not have a direct natural inspiration.

The family of Bioinspired Algorithms include *Genetic Algorithms* (Michalewicz, 1996) (where a set of chromosomes evolve by means of mutations and crossovers), *Simulated Annealing* (Aarts & Korst, 1990) (that exploits an analogy between the way in which a metal cools and freezes into a minimum energy crystalline structure -the annealing process- and the search for a minimum in a more general system) and *Ant Colony Optimization (ACO)* (Dorigo & Di Caro, 1999), among others.

The ACO Metaheuristic (Dorigo, Di Caro & Gambardella, 1999) (Dorigo & Di Caro, 1999) is one of the most recent bioinspired metaheuristics. The initial models were developed by Dorigo, Maniezzo and Colorni (Dorigo, Maniezzo & Colorni, 1996) a few years ago. Recently, a large number of authors have developed more complex models.

These algorithms take inspiration from the behaviour of real ant colonies to identify shortest paths between the nest and a food source. While walking between their nest ant and the source food, the ants deposit a substance called *pheromone*. When ants arrive to a path intersection, they need to choose the path to follow. They select it applying a probabilistic decision biased by the amount of pheromone: stronger *pheromone trails* are preferred. The most promising paths receive a greater pheromone after some time. This is due to the fact that, because these paths are shorter, the ants following them are able to reach the goal quicker and to start the trip back soon. Finally, the pheromone is evaporated by the environment, and makes less promising paths lose pheromone because they are progressively visited by less ants.

Evolutionary Computation (EC) (Bäck, Fogel & Michalewicz, 1998) is the general term for several computational techniques which are based to some degree on the evolution of biological life in the natural world. All of these techniques share some properties: they use a population of individuals with the ability to reproduce, these individuals suffer some genetic variations (usually mutations and crossovers) and there exists some kind of "natural" selection among them.

In this contribution, two ACO models which incorporate some of these EC specific components to the ACO basic model are introduced in order to improve its performance. These models are the Best-Worst Ant System and the exchange of memoristic information in parallel ACO algorithms.

To do so, in *section 2* we study the principles of ACO and EC and we discuss similarities and differences between both of them. Then, the two new models of ACO algorithms with EC components will be studied in *section 3*. Some future works are presented in *section 4* and finally some concluding remarks are pointed out in *section 5*.

BACKGROUND

In this section, we will introduce both the ACO Metaheuristic (with some specific ACO models as well) and EC (giving special attention to the components which will be introduced in the Best-Worst Ant System and the exchange of memoristic information in parallel ACO algorithms). In the last subsection we will focus on the similarities and differences between both computational techniques.

THE ACO METAHEURISTIC

The ACO metaheuristic belongs to the group of bioinspired metaheuristics. This metaheuristic is composed of different algorithms where several cooperative agent populations try to simulate real ant behaviour.

Specifically, they imitate these insects' behaviour when searching among the space and also how they carry their food around the ground. In their searching process, ants deposit a small amount of pheromone, which is a substance that can be "smelled" afterwards. In the near future, every ant can direct its search (and so direct the search of the whole colony as a group) according to the amounts of this hormone on the ground. The way an ant selects the direction it should follow is easy: it takes any direction randomly, but its decision is biased by the pheromone amount in each possible path. The continuous movement of every ant in the colony causes the best paths (the shortest ones) to have the largest amounts of pheromone –the shorter the path, the faster ants go through it, and consequently more ants walk over the path and more pheromone is left behind-. On the other hand, longer paths are progressively abandoned. Therefore, the pheromone that was deposited before finally evaporates. At the end, the best path (the minimum length one) is found between the ant nest and the food source. This mechanism is illustrated in *Figure 1*.

PROBLEMS THAT CAN BE SOLVED USING ACO

A large range of problems (mainly discrete combinatorial optimization problems) can be solved using this kind of approach. All of these problems belong to the group of (constrained) shortest path problems that can be characterized by the following aspects (see (Dorigo & Di Caro, 1999) and (Dorigo & Stützle, 2003) for more details):

- There is a *set of constraints* Ω defined for the problem being solved.
- There is a finite set of components $N = \{n_1, n_2, ..., n_l\}$.

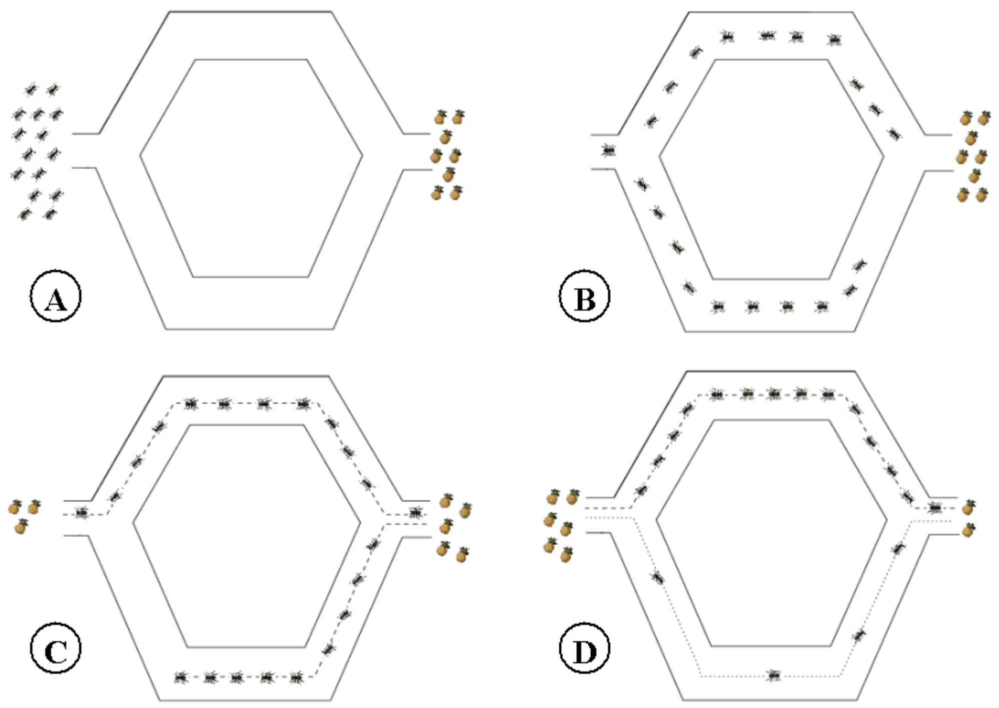


Figure 1: Emergent behaviour of the colony that ends by obtaining the shortest path between two points (mass recruitment).

A) All ants are in the nest and will begin to search for food.

B) The ants choose a random path since they don't know which one is shorter (better).

C) The ants that follow the shorter path return faster to the nest, depositing more pheromone on their way back.

D) The shortest path has more pheromone, and that makes ants follow it with a higher probability.

Based on the figure in (Bonabeau, Dorigo & Theraulaz, 1999).

- The problem presents several *states* defined upon ordered component sequences $\delta = \langle n_r, n_s, \ldots, u_n, \ldots \rangle$ ($\langle r, s, \ldots, u, \ldots \rangle$ to simplify) over the elements of *N*. If $\varDelta$ is the set of all possible sequences, we denote by $\tilde{\varDelta}$ the set of feasible (sub) sequences with respect to the constraints ΩThe elements in $\tilde{\varDelta}$ define the *feasible states*. $|\delta|$ is the length of a sequence δ i.e., the number of components in the sequence.

- There is a *neighbourhood structure* defined as follows: δ_2 is a neighbour of δ_1 if (i) both δ_1 and δ_2 belong to Δ, (ii) the state δ_2 can be reached from δ_1 in one logical movement, i.e., if *r* is the last component of the sequence δ_1, there must exist a component $s \in Y$ such that $\delta_2 = <\delta_1, s>$, i.e., there exists a valid transition between *r* and *s*. The *feasible neighbourhood* of δ_1 is the set containing all sequences $\delta_2 \in \tilde{\Delta}$; if $\delta_2 \notin \tilde{\Delta}$, we say that δ_2 is in the *infeasible neighbourhood* of δ_1.
- A solution *S* is an element of $\tilde{\Delta}$ verifying all the problem requirements.
- There is a cost *C(S)* associated with each solution *S*.
- In some cases, a cost or an estimate of the cost may be associated to states.

As said, all the previous characteristics hold in combinatorial optimization problems that can be represented in the form of a weighted graph *G=(N,A)*, where *A* is the set of edges that connects the set of components *N*. The graph *G* is also called construction graph *G*.[i] Hence, we have that:

- the components n_r are the nodes of the graph,
- the states δ (and hence the solutions *S*) correspond to paths in the graph, i.e., sequences of nodes or edges,
- the edges of the graph, a_{rs}, are connections/transitions defining the neighbourhood structure. $\delta_2 = <\delta_1, s>$ is a neighbour of δ_1 if node *r* is the last component of δ_1 and edge a_{rs} exists in the graph,
- there may be explicit transition costs c_{rs} associated to each edge, and
- the components and connections may have associated pheromone trails τ, which represent some form of indirect, long term memory of the search process, and heuristic values η which represent some heuristic information available on the problem under solution.

Some typical examples of problems that are easily solved using this metaheuristic are the *Travelling Salesman Problem (TSP)* (Bentley, 1992), the *Quadratic Assignment Problem (QAP)* (Maniezzo, Colorni & Dorigo, 1994) (Dorigo, et al., 1996), the *Job Shop Scheduling (JSP)* (Colorni, Dorigo, Maniezzo & Trubian, 1994), *vehicle and network routing* (Schoonderwoerd, Holland, Bruten & Rothkrantz, 1996) (Di Caro & Dorigo, 1998) (Bullnheimer, Hartl & Strauss, 1999b) among many others.

BASICS IF AN ACO ALGORITHM

ACO algorithms are essentially construction algorithms: in each algorithm iteration, every ant constructs a solution to the problem by travelling on a construction graph. Each edge of the graph, representing the possible steps the ant can make, has associated two kinds of information that guide the ant movement:

- *Heuristic information (η_{rs}):* It depends only on the problem that we want to solve. Usually, this information is calculated before the beginning of the run of the algorithm. It represents the a priori quality of the arc. If we compare with natural ants, it corresponds to the difficulty or distance in this part of the path. Clearly, it only depends on the topography of the ground.

- *Memoristic information (τ_{rs}):* This kind of information is modified during the run of the algorithm. It only depends on two factors: the number of ants that followed that arc and on the goodness of the path found by those ants. Comparing with natural ants, the memoristic information are the amounts of pheromone left on the ground. This will be called *pheromone trail (r, s)* from now on.

Exploiting the available pheromone trails and the heuristic information, the artificial ants (which are simple, computational agents) try to build feasible solutions to the problem tackled. However, if necessary, they may also build infeasible solutions that may be penalized depending on the amount of infeasibility. The artificial ant has the following properties (Dorigo & Di Caro, 1999) (Dorigo & Stützle, 2003):

- It searches minimum cost feasible solutions for the problem being solved.

- It has a memory *L* storing information about the path followed until that moment, i.e., *L* stores the generated sequence. This memory can be used to: (i) build feasible solutions, (ii) evaluate the generated solution, and (iii) to retrace the path the ant has followed.
- It has an *initial state* $\delta_{initial}$, that usually corresponds to a unitary sequence, and one or more termination conditions *t* associated.
- It starts in the *initial state* and moves towards feasible states, building its associated solution incrementally.
- When it is in a state $\delta = <\delta_{-1}, r>$ (i.e., it is located in node *r* and has previously followed the sequence δ_{-1}), it can move to any node *s* of its *feasible neighborhood N(r)*, defined as $N(r)=\{s|(a_{rs}\in A)\ and\ (<\delta_r, s>\text{¿}\tilde{\Delta})\}$.
- The movement is made by applying a transition rule, which is a function of the locally available pheromone trails and heuristic values, the ants private memory, and the problem constraints.
- When, during the construction procedure, an ant moves from node *r* to *s*, it can update the pheromone trail τ_{rs} associated to the edge a_{rs}. This process is called *online step-by-step pheromone trail update*.
- The construction procedure ends when any termination condition is satisfied, usually when an *objective state* is reached.
- Once the solution has been built, the ant can retrace the travelled path and update the pheromone trails on the visited edges/components by means of a process called *online delayed pheromone trail update.* This way, the only communication mechanism among the ants is the data structure storing the pheromone levels of each edge/component (shared memory).

In addition to the basic behaviour of the ants in the colony, an ACO algorithm comprises two additional procedures, *pheromone trail evaporation* and *daemon actions*. The pheromone evaporation is triggered by the environment and it is used as a mechanism to avoid search stagnation and to allow the ants to explore new space regions. Daemon actions are optional actions -without a natural counterpart- to implement tasks from a global perspective that is lacking to the local perspective of the ants. Examples are observing the quality of all the solutions generated and releasing an additional pheromone amount only on the transitions/components associated to some of the solutions, or applying a local search procedure to the solutions generated by the ants before updating the pheromone trails. In both cases, the daemon replaces the online delayed pheromone update and the process is called *offline pheromone trail update*. Besides, another usual action performed by the daemon is the restart of the search when it has got stagnated. It is usually done by resetting all the pheromone trails to the initial value τ_0 and go on with the search process.

The structure of a generic ACO algorithm is as follows (Dorigo & Di Caro, 1999) (Dorigo, et al., 1999).

```
Procedure ACO_Metaheuristic
  parameter_initialization
  while (termination_criterion_not_satisfied)
    schedule_activities
      ants_generation_and_activity()
      pheromone_evaporation()
      daemon_actions()      {optional}
    end schedule_activities
  end while
end Procedure
```

```
Procedure ants_generation_and_activity()
  repeat in parallel for k=1 to m (number_of_ants)
    new_ant(k)
  end repeat in parallel
end Procedure
```

```
Procedure new_ant(ant_id)
  initialize_ant(ant_id)
  L = update_ant_memory()
  while (current_state ≠ target_state)
    P = compute_transition_probabilities(A,L,Ω
    next_state = apply_ant_decision_policy(P,Ω
    move_to_next_state(next_state)
    if (on_line_step-by-step_pheromone_update)
      deposit_pheromone_on_the_visited_edge()
    end if
```

```
        L = update_internal_state()
    end while
    if (online_delayed_pheromone_update)
        for each visited edge
            deposit_pheromone_on_the_visited_edge()
        end for
    end if
    release_ant_resources(ant_id)
end Procedure
```

The first step involves the initialization of the parameter values considered by the algorithm. Among others, the initial pheromone trail value associated to each transition, τ_0, which is a small positive value that is typically the same for all connections/components, the number of ants in the colony, *m*, and the weights defining the balance between the heuristic and memoristic information in the probabilistic transition rule have to be set.[ii]

The main procedure of the ACO metaheuristic manages, by means of the `schedule_activities` construct, the scheduling of the three components mentioned in this section: (i) the generation and operation of the artificial ants, (ii) the pheromone evaporation, and (iii) the daemon actions. The implementation of this construct will define the existing syncronism between the three components. While the application to "classical" NP-hard (non distributed) problems typically uses rather a sequential schedule, in distributed problems like network routing, parallelism can be easily and efficiently exploited.

As said, several components are either optional, such as the daemon actions, or strictly dependent on the specific ACO algorithm, e.g., when and where the pheromone is deposited. Generally, the online step-by-step pheromone trail update and the online delayed pheromone trail update are mutually exclusive and they both are not usually present or missing at the same time (if both are missing, typically the daemon updates the pheromone trails).

On the other hand, notice that the procedure `update_ant_memory` involves both specifying the initial state from which the ant starts its path and storing the corresponding component in the ant memory *L*. The decision on which will be that node (it can be a random choice or a fixed one for the whole colony, a random choice or a fixed one for each ant, etc.) depends on the specific application.

Finally, note that the procedures `compute_transition_probabilities` and `apply_ant_decision_policy` consider the current state of the ant, the current values of the pheromone trails visible in that node and the problem constraints Ω to establish the probabilistic transition process to other feasible states.

STEPS TO SOLVE A PROBLEM USING ACO

From the currently known ACO applications, we can identify some guidelines of how to attack new problems by ACO. These guidelines can be summarized by the following six design tasks (Cordón, Herrera & Stützle, 2002):

- Represent the problem in the form of sets of components and transitions or by means of a weighted graph that is travelled by the ants to build solutions.

- Appropriately define the meaning of the pheromone trails τ_{rs}, i.e., the type of decision they bias. This is a crucial step in the implementation of an ACO algorithm and often, a good definition of the pheromone trails is not a trivial task and it typically requires insight into the problem being solved.

- Appropriately define the heuristic preference to each decision that an ant has to take while constructing a solution, i.e., define the heuristic information η_{rs} associated to each component or transition. Notice that heuristic information is crucial for good performance if local search algorithms are not available or can not be applied.

- If possible, implement an efficient local search algorithm for the problem under consideration, because the results of many ACO applications to NP-hard combinatorial optimization problems show that the best performance is achieved when coupling ACO with local optimizers (Dorigo & Di Caro, 1999) (Dorigo & Stützle, 2003).

- Choose a specific ACO algorithm (some of the available ones are described in the next section) and apply it to the problem being solved, taking the previous aspects into account.

- Tune the parameters of the ACO algorithm. A good starting point for parameter tuning is to use parameter settings that were found to be good when applying the ACO algorithm to similar problems or to a variety of other problems. An alternative to time-consuming personal involvement in the tuning task is to use automatic procedures for parameter tuning (Birattari, Stützle, Paquete & Varrentrapp, 2002).

It should be clear that the above steps can only give a very rough guide to the implementation of ACO algorithms. In addition, often the implementation is an iterative process, where with some further insight into the problem and the behaviour of the algorithm, some initially taken choices need to be revised. Finally, we want to insist in the fact that probably the most important of these steps are the first four, because a poor choice at this stage typically can not be made up with pure parameter fine-tuning.

INSTANCES OF ACO ALGORITHMS

As said, different ACO models have been proposed. Among them, we find *Ant System (AS)* (Dorigo, et al., 1996), *Ant Colony System (ACS)* (Dorigo & Gambardella, 1997), *Rank-based Ant System* (Bullnheimer, Hartl & Strauss, 1999a), *Max-Min Ant System* (MMAS) (Stützle & Hoos, 2000) and *Best-Worst Ant System (BWAS)* (Cordón, Fernández de Viana, Herrera & Moreno, 2000). Each algorithm has its own transition and update pheromone rules. In this chapter we will describe two of the first ACO models: AS and ACS.

ANT SYSTEM

AS, developed by Dorigo, Maniezzo and Colorni in 1991, was the first ACO algorithm. AS is characterized by the fact that the pheromone update is triggered once all ants have completed their solutions and it is done as follows. First, all pheromone trails are reduced by a constant factor, implementing in this way the pheromone evaporation. Second, every ant of the colony deposits an amount of pheromone which is a function of the quality of its solution. Initially, AS did not use any daemon actions, but it is very straightforward to, for example, add a local search procedure to refine the solutions generated by the ants.

Solutions in AS are constructed as follows. At each construction step, an ant *k* in AS chooses to go to a next node with a probability that is computed as

$$p_{rs}^k = \begin{cases} \dfrac{[\tau_{rs}]^\alpha \cdot [\eta_{rs}]^\beta}{\sum_{u \in N_r^k} [\tau_{rs}]^\alpha \cdot [\eta_{rs}]^\beta}, & \text{if } s \in N_k(r) \\ 0, & \text{otherwise} \end{cases}$$

where *N*$_k$*(r)* is the feasible neighbourhood of ant *k* when located at node *r*, and $\alpha, \beta \in \Re$ are two parameters that weight the relative importance of the pheromone trail and the heuristic information. Each ant *k* stores the sequence it has followed so far and this memory L_k is, as explained before, exploited to determine *N*$_k$*(r)* in each construction step.

As regards parameters α and β their role is as follows: if $\alpha=0$, those nodes with better heuristic preference have a higher probability of being selected, thus making the algorithm close to a classical *probabilistic greedy algorithm* (with multiple starting points in case ants are located in different nodes at the beginning of each iteration). However, if $\beta=0$, only the pheromone trails are considered to guide the constructive process, which can cause a quick *stagnation*, i.e., a situation where the pheromone trails associated to some transitions are significantly higher than the remainder, thus making the ants always build the same solutions, usually local optima. Hence, there is a need to establish a proper balance between the importance of heuristic and pheromone trail information.

As said, the pheromone deposit is made once all ants have finished to construct their solutions. First, the pheromone trail associated to every arc is evaporated by reducing all pheromones by a constant factor:

$$\tau_{rs} \leftarrow (1-\rho) \cdot \tau_{rs}$$

where $\rho\in[0,1]$ is the evaporation rate. Next, each ant retraces the path it has followed (this path is stored in its local memory L_k) and deposits an amount of pheromone $\Delta\tau_{rs}^{k}$ on each traversed connection:

$$\tau_{rs} \leftarrow \tau_{rs} + \Delta\tau_{rs}^{k}, \quad \forall\, a_{rs} \in S_k$$

where $\Delta\tau_{rs}^{k} = f(C(S_k))$, i.e., the amount of pheromone released depends on the quality $C(S_k)$ of the solution S_k of ant k.

To summarize the description of the AS, we will show the composition of procedure `new_ant` for this particular ACO algorithm:

```
1   Procedure new_ant(ant_id)
2     k = ant_id; r = generate_initial_state; $S_k$ = r
3     $L_k$ = r
4     while (current_state $\neq$ target_state)
5       for each $s \in N_k(r)$ do $p_{rs}^{k} = \frac{[\tau_{rs}]^{\alpha} \cdot [\eta_{rs}]^{\beta}}{\sum_{u \in N_r^k} [\tau_{rs}]^{\alpha} \cdot [\eta_{rs}]^{\beta}}$
6       next_state = apply_ant_decision_policy(P, $N_k$(r))
7       r = next_state; $S_k$ = <$S_k$, r>
8       ---
9       $L_k$ = $L_k \cup r$
10    end while
      { the pheromone_evaporation() procedure triggers and
        evaporates pheromone in every edge $a_{rs}$: $\tau_{rs}=(1-\rho)\cdot\tau_{rs}$    }
11    for each edge $a_{rs} \in S_k$ do
12      $\tau_{rs}=\tau_{rs}$+f(c($S_k$))
13    end for
14    release_ant_resources(ant_id)
15  end Procedure
```

Notice that the empty line *8* is included to remark that no online step-by-step pheromone update is made and that *before the line 12, the pheromone evaporation must have been applied by the daemon*. In fact, this is one example where the `schedule_activities` construct interferes with the functioning of the single main procedures of the ACO metaheuristic.

Before concluding this section, it is important to notice that the creators of the AS also proposed a typically better performing, extended version of this algorithm called *elitist AS* (Dorigo, et al., 1996). In elitist AS, once the ants have released pheromone on the connections associated to their generated solutions, the daemon performs an additional pheromone deposit on the edges belonging to the best solution found until that moment in the search process (this solution is also called global-best solution in the following). The amount of pheromone deposited, which depends on the quality of that global best solution, is weighted by the number of elitist ants considered, *e*, as follows:

$$\tau_{rs} = \tau_{rs} + e \cdot f\left(C\left(S_{global-best}\right)\right), \quad \forall a_{rs} \in S_{global-best}$$

ANT COLONY SYSTEM

ACS is one of the first successors of AS. It introduces three major modifications into AS:

- ACS uses a different transition rule, which is called *pseudo-random proportional rule*: Let *k* be an ant located at a node *r*, $q_0 \notin [0,1]$ be a parameter, and *q* a random value in *[0,1]*. The next node *s* is randomly chosen according to the following probability distribution

  If $q \le q_0$:

$$p_{rs}^{k} = \begin{cases} 1, & if \quad s = \arg \max\limits_{u \in N_k(r)} \left\{ \tau_{ru} ¿ \eta_{ru}^{\beta} \right\} \\ 0, & otherwise \end{cases}$$

  else ($q > q_0$):

$$p_{rs}^{k} = \begin{cases} \dfrac{\left[\tau_{rs}\right]^{\alpha} \cdot \left[\eta_{rs}\right]^{\beta}}{\sum_{u \in N_r^k} \left[\tau_{rs}\right]^{\alpha} \cdot \left[\eta_{rs}\right]^{\beta}}, & if \quad s \in N_k(r) \\ 0, & otherwise \end{cases}$$

As can be seen, the rule has a double aim: when $q \leq q_0$, it exploits the available knowledge, choosing the best option with respect to the heuristic information and the pheromone trail. However, if $q > q_0$, it applies a controlled exploration, as done in AS. In summary, the rule establishes a trade-off between the exploration of new connections and the exploitation of the information available at that moment.

- Only the daemon (and not the individual ants) trigger the pheromone update, i.e., an offline pheromone trail update is done. To do so, ACS only considers a single ant, the one who generated the global best solution, $S_{global\text{-}best}$ (although in early papers, an update based on the iteration-best ant was considered as well (Dorigo, & Gambardella, 1997), ACS almost always applies a global-best update).

  The pheromone update is done by first evaporating the pheromone trails in all the connections used by the global-best ant (it is important to notice that *in ACS, pheromone evaporation is only applied to the connections of the solution that is also used to deposit pheromone*) as follows:

$$\tau_{rs} \leftarrow (1-\rho)\cdot\tau_{rs}, \quad \forall a_{rs} \in S_{global-best}$$

  Next, the daemon deposits pheromone by the rule:

$$\tau_{rs} \leftarrow \tau_{rs} + \rho\cdot f(C(S_{global-best})), \quad \forall a_{rs} \in S_{global-best}$$

  Additionally, the daemon can apply a local search algorithm to improve the ants' solutions before updating the pheromone trails.

- Ants apply an *online step-by-step pheromone trail update* that encourages the generation of different solutions to those yet found. Each time an ant travels an edge a_{rs}, it applies the rule:

$$\tau_{rs} \leftarrow (1-\phi)\cdot\tau_{rs} + \phi\cdot\tau_0$$

where $\varphi \in (0,1]$ is a second pheromone decay parameter. As can be seen, the *online step-by-step update rule* includes both, pheromone evaporation and deposit. Because the amount of pheromone deposited is very small (in fact, τ_0 is the initial pheromone trail value which is chosen in such a way that, in practice, it corresponds to a lower pheromone trail limit, i.e., by the choice of the ACS pheromone update rules, no pheromone trail value can fall below τ_0), the application of this rule makes the pheromone trail on the connections traversed by an ant decrease. Hence, this results in an additional exploration technique of ACS by making the connections traversed by an ant less attractive to following ants and helps to avoid that every ant follows the same path.

The procedures `new_ant` and `daemon_actions` (which in this case interacts with the `pheromone_evaporation` procedure) for ACS are as follows:

```
Procedure daemon_actions
  for each $S_k$ do local_search($S_k$)  {optional}
  $S_{current-best}$ = best_solution (S)
  if (better($S_{current-best}$, $S_{global-best}$))
    $S_{global-best}$ = $S_{current-best}$
  end if
  for each edge $a_{rs} \in S_{global-best}$ do
    { the pheromone_evaporation() procedure triggers and
          evaporates pheromone in edge $a_{rs}$: $\tau_{rs} = (1 - \rho) \cdot \tau_{rs}$  }
    $\tau_{rs} = \tau_{rs} + \rho \cdot f(C(S_{global-best}))$
  end for
end Procedure
```

```
Procedure new_ant(ant_id)
  k = ant_id; r = generate_initial_state; $S_k$ = r
  $L_k$ = r
  while (current_state $\neq$ target_state)
    for each $s \in N_k(r)$ do compute $b_{rs} = \tau_{rs} \cdot \eta_{rs}^{\beta}$
    q = generate_random_value_in_[0,1]
    if (q <= $q_0$)
```

```
          next_state = max(b_rs, N_k(r))
        else
          for each $s \in N_k(r)$ do
                    $p_{rs}^k = \frac{b_{rs}}{\sum_{u \in N_k(r)} b_{ru}}$
          next_state = apply_ant_decision_policy(P, $N_k(r)$ )
        end if
7       r = next_state;  $S_k = < S_k, r >$¿ ¿
8       $\tau_{rs} = (1 - \phi) \cdot \tau_{rs} + \phi \cdot \tau_0$
9       $L_k = L_k \cup r$
10    end while
11    ---
12    ---
13    ---
14    release_ant_resources(ant_id)
15  end Procedure
```

EVOLUTIONARY COMPUTATION

EC (Bäck et al., 1998) is the term that denotes the group of algorithms and techniques which have inspiration on the natural processes that involve evolution. *Evolutionary algorithms (EAs)* (Bäck, 1996) constitute a class of search and optimization methods which share some generic concepts. Some of the techniques that can be included in this class are *Genetic Algorithms*, *Evolution Strategies*, *Evolutionary Programming* and *Genetic Programming*. As said, all of them use a population of competing candidate solutions which reproduce and evolve themselves by means of combinations and alterations, and a selection mechanism which increases the proportion of better solutions in the population. Every different approach has its own genetic structures and own genetic operators which manipulate and generate new candidate solutions.

Since EAs emulate natural evolution, they adopt a biological terminology to describe their structural elements and algorithmic operations, but we should keep in mind that these terms are much more simple than their biological analogous. In *table 1*, we describe a mapping of some of the most common terms in EC from nature to computer science.

Progress in natural evolution is based on three fundamental processes: mutation, recombination and selection of genetic variants. Mutation introduces a random variation into the existing genetic material, thus allowing the appearance of new characteristics that did not exist previously. Recombination is the process which hybridises (usually two) different chromosomes in order to generate a new solution hoping to take advantage of the best characteristics of the parents, that is, to try to obtain a new solution of the problem using the best features of both parents. Finally, selection of genetic variants allow the quality of the chromosomes to increase in the actual population. This later process is usually accomplished suppressing the worst solutions in the population (not allowing them to mutate and reproduce themselves). That is usually known by the term *survival of the fittest*, introduced in some works of Darwin. *Fitness* is the term that describes the quality of an existing solution, and can be used to determine how this solution deserves to continue existing in the actual solutions' population and thus, continue evolving into (hopefully) better solutions.

Nature	***Computer***
Individual	Solution to a problem
Population	Set of solutions
Fitness	Quality of a solution
Chromosome	Representation for a solution (e.g. set of parameters)
Gene	Part of the representation of a solution (e.g. parameter or degree of freedom)
Growth	Decoding of the representation of solutions
Crossover Mutation	Search operators
Natural Selection	Reuse of good (sub-)solutions

Table 1: Nature to Computer terms mapping.

EAs provide a universal optimization technique applicable to a wide range of problem domains. A generic EA operation mode is now described described:

```
1. Initialise population
2. Evaluate population
3. Repeat until termination criteria is fulfilled
      3.1. Select a sub-population for reproduction (Selection)
      3.2. Recombine the "genes" of selected parents
      (Recombination or Crossover)
```

```
        3.3. Mutate the mated population stochastically (*Mutation*)
        3.4. Evaluate the fitness of the new population
        3.5. *Select the survivors* from the actual fitness
```

Note that this is a very general approximation of an EA, and there exist so many variants of EAs that not all of them will fit perfectly in that simple scheme (for example, there are some EAs which do not use all of these steps).

One of the best advantages of this kind of technique is that it needs no particular knowledge about the problem structure but the objective fitness function itself. EAs are robust techniques that are capable of making a good exploitation of the accumulated information about an initial unknown search space, biasing the search into a more useful subspace. They provide an efficient and effective approach to manage large, complex and poorly understood search spaces where other techniques are not useful.

ACO AND EC

Several similarities exist between both the ACO metaheuristic and EC (as well as some important differences distinguish them). In the following two subsections, the relation between ACO and EC is analyzed in more detail.

SIMILARITIES AND DIFFERENCES BETWEEN ACO AND EC

There are many similarities between the ACO metaheuristic and EC. The most important ones are as follows:

1. Both are bioinspired techniques, that is, both of them mimic in some way a natural process to achieve a good solution of the problem being tackled. Of course, this similarity has nothing to do with their effectiveness when solving problems, but at least gives a common framework in which to enclose both techniques.

2. They use a population of individuals to represent problem solutions. Nevertheless, in EC the solutions are in fact the individuals (chromosomes), while in the ACO metaheuristic the individuals are ants, which construct and remember the solutions found so far (but they do not represent the solutions themselves). In EC, every action is made by the system as a whole, and from a global perspective, could be compared to the daemon actions in the ACO metaheuristic.

3. Both techniques use of the knowledge collected at the algorithm run-time to bias the generation of the new population of individuals. However, one main difference is that, in general, EAs use the current population to store the knowledge about the problem (the solutions themselves represent the problem acquired knowledge), whilst in ACO the memoristic structure collecting the pheromone trails (the pheromone matrix) is considered to perform this task and the solutions found are not directly used to represent the obtained information about the problem.

Apart from those mentioned in the previous items, there also exist several differences:

1. Almost every ACO algorithm considers the use of heuristic information, in other words, utilizes some a-priori information about the problem. EAs usually do not take this kind of information into account. This can be an advantage for the ACO metaheuristic because this information can be very useful to guide the algorithm behaviour (specially in the first iterations). If there is no a-priori information about the problem, or if it is very difficult to code it in the form needed by a particular ACO algorithm, this information can usually be omitted without changing the general structure of the algorithm (the component considering the action of the heuristic information in the transition rule is ignored by setting the associated weight to zero).

2. EAs can use different mutation and crossover operators while a particular ACO algorithm uses a single construction rule to generate new solutions to the problem. Using multiple mutation and crossover operators can give more flexibility to the search process and can improve the search tackled on the space of the algorithm (more exploration of the search space).

RELATION BETWEEN ACO AND POPULATION-BASED INCREMENTAL LEARNING

There is a kind of EAs, the so-called Estimation of Distribution Algorithms (EDAs) (Larrañaga & Lozano, 2001), which are based on maintaining a memoristic structure that represents a probability distribution defined on the problem variables. The best known EDA algorithm is *Population-Based Incremental Learning (PBIL)* (Baluja & Caruana, 1995). It is important to study the relation between ACO and PBIL to develop a new hybridized algorithm in *Section 3.1*.

So, as ACO does, PBIL uses a memoristic structure to generate the problem solutions. It consists of a probability vector whose dimension is equal to the number of problem variables. This vector encodes a probabilistic distribution that represents a prototype of good solutions which the algorithm uses to generate a population of new problem solutions in each iteration. The first PBIL model used binary arrays to represent solutions but extended models to solve problems with real variables have been later proposed.

Each position of this probability vector is adapted during the algorithm run-time. The best solution found in the current iteration is used to update the vector. In principle, this update is proportional to the quality of that solution, but more recent models use the *M* best solutions generated in each iteration for the updating process instead of just the best one.

The similarities between ACO algorithms and PBIL (Monmarché, Ramat, Dromel, Slimane & Venturini, 1999) are as follows:

- Both algorithms use some kind of memoristic information to guide the search process.

- The adaptation rules for the structure encoding this information are very similar.

Besides, the components of the probabilistic vector in PBIL suffer random mutations to avoid the chance for premature convergence, while the usual ACO models do not include this action. PBIL also does not use any kind of heuristic information to help in the solution construction process which is a characteristic that has been present in almost every ACO model.

TWO MODELS OF ACO WITH EC COMPONENTS

In the previous sections, we have presented the relations between *ACO* and *EC*. We have also shown that there are some specific components that are used successfully in EAs algorithms but not in ACO. In the next sections, two models which incorporate some of these EC specific components to the ACO basic model are introduced. The aim of those hybrid models is to achieve significantly improved computational results when compared to classical ACO algorithms.

Notice that there have been proposed other hybridizations between EC and ACO, for example, running a meta-level GA to fine-tuning the ACO parameters (Boote & Bonabeau, 1998), but they will not be considered in this work.

BEST-WORST ANT SYSTEM

BWAS is a recent, new ACO model which tries to improve the results obtained by adding some new features from the field of EC. In the next sections, this model will be described and some empirical results will be analyzed.

JUSTIFICATION OF THE APPROACH

Every algorithm that is proposed to solve hard computational problems (usually NP-hard problems) has to do some balance between exploration and exploitation of the search space. If this balance is not good enough, our algorithm can stop finding good solutions after a short period of time, being stuck in a specific search space zone (it would need more exploration), or it would not increase the solutions quality because it is not able to intensify the search in the current space zone to extract the best solutions located on it (it would need more exploitation).

The ACO metaheuristic does not define how this balance should be achieved, because how the different solutions are constructed and how the memoristic information is updated are concrete aspects of every ACO algorithm. Specifically, AS gives too much importance to the exploration facet, because the whole colony updates the pheromone trails, and what is more important, evaporation is made in every arc defined in the pheromone matrix. This makes the algorithm to evolve slowly, with a slow convergence rate, and consequently not to achieve good quality solutions.

ACS offers a better balance between exploration and exploitation of the search space due to its new components. On the other hand, the new transition rule encourages more the exploitation, due to directly selecting the best transition in some of the cases. Besides, ACS performs a global update which only evaporates using the best solution found (which decreases exploration and makes exploitation stronger). Also, the *online step-by-step pheromone trail update* gives more importance to exploration, causing the ants to generate different enough solutions in the current iterations, finally achieving a better balance than AS.

Of course, the use of a local search technique increases the exploitation in both models, and it is usually utilized to increase performance. In this case, the ACO algorithm behaves like a multi-start local search, generating different initial solutions for the local search optimiser, like metaheuristics as ILS (Lourenço, Martin & Stützle, 2003) or VNS (Mladenovic & Hansen, 1997).

In our case, the BWAS introduce some concepts from EDAs (mainly from PBIL), which give a different kind of trade-off between both exploration and exploitation. The evaporation in every arc of the graph gives more diversification to the search process (more exploration) and the update rule, which involves only the global best and the current worst solutions, intensifies the search in the best paths (more exploitation). Besides, a mutation operator is considered to randomly alter the pheromone trails to introduce diversity, as done in PBIL, and a restart is applied when the search is stagned. As will be seen in the next sections, this balance seems to achieve better results than other earlier ACO algorithms.

DESCRIPTION OF THE BWAS

BWAS (Cordón, et al., 2000), proposed by Cordón et al. in 1999, is an ACO algorithm which incorporates EC concepts. It constitutes another extension of AS, which uses its transition rule and pheromone evaporation mechanism. That means that the evaporation is applied to every transition, as in AS, AS_{rank} (Bullnheimer, et al., 1999) and MMAS (Stützle & Hoos, 2000)). Besides, as done in MMAS, BWAS always considers the systematic exploitation of local optimizers to improve the ants' solutions.

At the core of BWAS, the three following daemon actions are found:

- The *best-worst pheromone trail update rule*, which reinforces the edges contained in the global best solution. In addition, the update rule penalizes every connection of the worst solution generated in the current iteration, *current-worst*, that are not present in the global-best one (to avoid penalizing possible good paths) through an additional evaporation of the pheromone trails. Hence, the BWAS update rule becomes:

$$\tau_{rs} \leftarrow \tau_{rs} + \rho \cdot f(C(S_{global-best})), \quad \forall a_{rs} \in S_{global-best}$$

$$\tau_{rs} \leftarrow (1-\rho) \cdot \tau_{rs}, \quad \forall a_{rs} \in S_{current-worst} \quad y \quad a_{rs} \notin S_{global-best}$$

  As we have mentioned before, the first part of this rule intensifies exploitation by giving a higher probability to the paths contained in the best solution found so far, and the second part introduces more exploration by avoiding the worst paths.

- A *pheromone trail mutation* is performed to introduce diversity in the search process. To do so, the pheromone trail associated to one of the transitions starting from each node (i.e., each row of the pheromone trail matrix) is mutated with probability P_m by considering any real-coded mutation operator. This daemon action obviously increases the exploration face of the algorithm.

  The original BWAS proposal applied an operator altering the pheromone trail of every mutated transition by adding or subtracting the same amount in each iteration. The mutation range *mut(it,* $\tau_{threshold}$*)*, which depends on the average of the pheromone trails in the transitions of the global best solution, $\tau_{threshold}$, is less strong in the early stages of the algorithm -when there is no risk of stagnation- and stronger in the latter ones, when the danger of stagnation is stronger:

$$\tau'_{rs} \leftarrow \begin{cases} \tau_{rs} + mut(it, \tau_{threshold}), & if \;\; a=0 \\ \tau_{rs} - mut(it, \tau_{threshold}), & if \;\; a=1 \end{cases}$$

with *a* being a random value in {0, 1} and *it* being the current iteration. Notice that this operator is based on Michalewicz's non uniform mutation for real coded Genetic Algorthms (Michalewicz & Fogel, 2000).

- As other ACO models, BWAS considers the *re-initialization of the pheromone trails* (algorithm restart) when the search gets stuck, which is done by setting every pheromone trail to τ_0. In earlier versions of the algorithm, the number of different arcs between the best and worst solutions in the current population was compared to some threshold value. More actual versions of the algorithm use a different concept of stagnation to know when it should re-initializate the pheromone matrix: the number of iterations without a change of the best global solution.

The BWAS `daemon_actions` procedure is as follows:

```
Procedure daemon_actions
  for each $S_k$ do local_search($S_k$)
  $S_{current-best}$ = best_solution(S)
  if (best($S_{current-best}$, $S_{global-best}$))
    $S_{global-best}$ = $S_{current-best}$
  end if
  for each edge $a_{rs} \in S_{global-best}$ do
    $\tau_{rs} = \tau_{rs} + \rho \cdot f(C(S_{global-best}))$
    sum = sum + $\tau_{rs}$
  end for
  $\tau_{threshold}$ = sum / $|S_{global-best}|$
  $S_{current-worst}$ = worst_solution ($S_k$)
  for each edge $a_{rs} \in S_{current-worst}$ and $a_{rs} \notin S_{global-best}$ do
    $\tau_{rs} = (1 - \rho) \cdot \tau_{rs}$
  end for
```

```
   mut = mut(it, $\tau_{threshold}$)
   for each node/component $r \in$ {1, ..., l} do
     z = generate_random_value_in_[0, 1]
     if (z $\leq P_m$)
       s = generate_random_value_in_[1, ..., l]
       a = generate_random_value_in_[0, 1]
       if (a = 0) $\tau_{rs}$ = $\tau_{rs}$ + mut
       else $\tau_{rs}$ = $\tau_{rs}$ - mut
     end if
   end for
   if (stagnation_condition)
     for each edge $a_{rs}$ do $\tau_{rs}$ = $\tau_0$
   end if
end Procedure
```

The interested reader can refer to (Cordón, Fernández de Viana & Herrera, 2002a) for an analysis of the individual performance of each of the three BWAS components and the different combinations of them on the TSP. In that study, it is shown that the version of BWAS that includes all three components performs better than other variants that include only a single component or a combination of two of the three components in the most of the cases. Thereby, the BWAS model can not ignore any of the three components if we want it to perform effectively. A similar study was done for BWAS applied to the QAP in (Cordón, Fernández de Viana & Herrera, 2002b). Moreover, a new, well-performing ACO model called *Best-Worst Ant Colony System*, which is based on introducing the three BWAS components into the ACS, is proposed in (Cordón, et al., 2002b).

EXPERIMENTS AND ANALYSIS OF RESULTS

In this section, we will present some results obtained with the BWAS applied to the TSP and we will compare them to some other ACO algorithms.

The TSP (Bentley, 1992) is a very well known classical combinatorial optimization problem. It is an NP-hard problem, that is, there is no efficient deterministic algorithm capable of solve it in polynomial time. Since the first proposals of ACO algorithms, this problem has been extensively used to compare the efficiency of every new ACO algorithm.

The problem is an analogy to a real problem. A salesman is supposed to visit -just one time- all cities in a region and to return to the original one using the shortest possible tour (to decrease transport costs).

More formally, the problem is described as: $G = (N, A)$ is a complete weighted graph, where N is the set of cities in the problem and A the set of arcs (roads) between them. Every arc has a value d_{ij}, which represents the length of the arc $(i, j) \in A$. The problem is to find the lowest cost Hamiltonian circuit in G.

One of the most important properties of this problem is the great similarity between the search space for the salesman and the search space for ants. Both are weighted graphs (in TSP the weight corresponds with the length of the roads between cities) where every arc is a path or road with a degree of desirability (the length), and in both cases the goal of the problem is to find the shortest path. This property makes very easy to adapt the ACO metaheuristic to solve this problem.

For this study, we have selected eight different TSP instances. All of them belong to the TSP Library (TSPLIB) (Reinelt, 2001). Specifically, we have selected the following instances: *brazil58*, *gr120*, *d198*, *lin318*, *att532*, *rat783*, *u1060* y *d1291*. The four former ones represent *small* and "easy" instances, while the latter two represent *large* instances. *att532* and *rat783* will be called mid-sized instances from now on.

All the parameters used to solve every instance are shown in *table 2*. These parameters have been proved to be the best possible ones in previous works (Cordón, et al., 2002a).

Parameter	***Value***
Ant number Number of executions Max. Time of Execution Transition Rule Global evaporation Const.	m = 15 10 T = 900 a 3600 sg. α= 1 ; β= 2 ρ= 0.2
Elitist AS	
Elitist Ants	m (15)
ACS	
Local evaporation const.	q_0 = 0.8 φ= 0.2
BWAS	
Mutation Probability Iterations without change	P_m= 0.3, 0.25, 0.2 0.3, 0.2, 0.3
Local Search	
Candidate List Size Selection Rule Algorithm	40 first better 2-opt

Table 2. Parameters used in TSP

For comparison purposes, two different algorithms have also been run on every instance. These are the *elitist AS* (AS variation where the so called *elitist ants* also update pheromone, see *Section 2.1.4.1*), which have been proved to give better results than AS, and the ACS.

All runs have been made using the following conditions:

- We have used the same values for the common parameters of the different algorithms.
- We have used a candidate list. That candidate list is fixed; to construct it, only heuristic information has been considered, which is present at the beginning of the run.
- We have considered the same seeds for the random number generators.
- We have used the same local search algorithm (see below).
- Both *Elitist AS* and *ACS* use the same restart component as BWAS. This two versions will be noted *EAS-Re* and *ACS-Re*.
- All experiments have been run 10 different times and the maximum execution time varies from 900 to 3600 seconds depending on the size of the problem (the run time could be shorter if the best possible solution is found before the end of the execution).

In all cases there has been used a local search in order to improve the results. There exist lots of different local search algorithms for this problem (Johnson & McGeoch, 1997). We have used an improved version of the 2-opt (Bentley, 1992). This local search tries to improve the solutions constructed by the ants by means of exchanges of arcs in the solution. The improvements made to this technique are:

- It only uses a candidate list (a list with the nearest neighbours of every node). This is made to increase the speed of the local search.
- It uses a "don't look bit" to remember which nodes can not be improved.

In *table 3*, we show the results achieved for every algorithm. This table has 6 different columns:

- The first shows the name of the algorithm.
- The second shows the best result obtained.
- The third shows the mean of the results.
- The fourth shows the standard deviation.
- The fifth is the mean error (proportion associated to the difference of the best known solution to the problem and the mean of all solutions found by the

  results

  algorithm): $Err = Best_{Solution_{Known}} - \frac{Mean_{of¿}}{Best_{Solution_{Known}}}$.

  ¿
- The sixth shows the mean of the number of restarts.
- C_{opt} is the length of the optimum tour, n is the number of cities in the problem.

In view of the obtained results, the best algorithm is always the BWAS. The mean error is always smaller than in EAS-Re and ACS-Re. Moreover, comparing the results with some previously published in (Cordón, et al., 2002a) we can see that now they have been improved (although we have not found any better individual solution, the mean error is always smaller).

Problem	**Brazil58 (C_{opt} = 25395, n = 58)**					**Gr120 (C_{opt} = 6942, n = 120)**				
Algorithm	***Best***	***Mean***	***Dev.***	***Err.***	***#R***	***Best***	***Mean***	***Dev.***	***Err.***	***#R***
EAS	25395	25395	0	0	-	6942	**6942**	0	0	-
ACS	25395	25395	0	0	-	6942	6946.1	5.49	0.06	-
EAS + Re	25395	25395	0	0	0	6942	**6942**	0	0	0
ACS + Re	25395	25395	0	0	0	6942	6943.8	3.79	0.03	1.1
BWAS	25395	25395	0	0	0	6942	**6942**	0	0	0.5
Problem	**Lin318 (C_{opt} = 42029, n = 318)**					**D198 (C_{opt} = 15780, n = 198)**				
Algorithm	***Best***	***Mean***	***Dev.***	***Err.***	***#R***	***Best***	***Mean***	***Dev.***	***Err.***	***#R***
EAS	42029	42123.8	67.28	0.22	-	15780	15781	0.70	¿ 0	-
ACS	42029	42230	148.48	0.48	-	15780	15784.9	5.67	0.03	-
EAS + Re	42029	42105	53.79	0.18	3	15780	15781	0.70	¿ 0	0
ACS + Re	42029	42182.4	118.12	0.36	5	15780	15782.9	4.31	0.02	2.3
BWAS	42029	**42084.1**	98.60	0.13	2	15780	**15780.4**	0.51	¿ 0	1.7
Problem	**Att532 (C_{opt} = 27686, n = 532)**					**Rat783 (C_{opt} = 8806, n = 783)**				
Algorithm	***Best***	***Mean***	***Dev.***	***Err.***	***#R***	***Best***	***Mean***	***Dev.***	***Err.***	***#R***
EAS	27745	27823	70.67	0.49	-	8860	8878.6	17.06	0.81	-
ACS	27705	27810.3	64.44	0.45	-	8857	8892.7	20.93	0.97	-
EAS + Re	27793	27825.6	41.59	0.50	3.6	8843	8878.4	32.25	0.81	3.8
ACS + Re	27745	27835	57.56	0.54	7	8875	8899.5	22.33	1.05	7.6
BWAS	**27686**	**27711**	12.16	0.09	7.7	**8816**	**8833.4**	15.37	0.31	9.5
Problem	**U1060 (C_{opt} = 224094, n = 1060)**					**D1291 (C_{opt} = 50801, n = 1291)**				
Algorithm	***Best***	***Mean***	***Dev.***	***Err.***	***#R***	***Best***	***Mean***	***Dev.***	***Err.***	***#R***
EAS	231644	232145.8	297.91	3.46	-	51210	51347.2	138.15	1.06	-
ACS	225675	226387.8	668.9	1.01	-	51901	51953.6	60.01	2.21	-
EAS + Re	230360	230806.4	390.8	2.90	2.9	51073	51236.7	119.56	0.85	4.5
ACS + Re	225243	226501	1059.57	1.06	2	51828	51986.8	105.80	2.28	5
BWAS	**225310**	**225721.1**	476.88	0.72	5.3	**50890**	**50986.3**	74.87	0.36	17.2

Table 3. Results for the TSP

To summarize, we would like to point out that:

- BWAS is a good alternative in ACO algorithms. With the same execution properties, it gives better solutions.
- BWAS gives reasonably good solutions even when the values of its parameters are not very appropriate, that is, BWAS is a *robust* algorithm.
- When the values of the parameters are fine-tuned, the robustness of the BWAS increases, obtaining smaller mean errors.

EXCHANGE OF MEMORISTIC INFORMATION IN PARALLEL ACO ALGORITHMS

The inherent structure of the ACO metaheuristic offers relative eases to parallelize the algorithms based on ants behaviour. This parallelization, as will be seen in the next sub-sections, can be done in several different ways. We will propose a new parallel ACO model which introduces some EC aspects as well as we will analize some computational results achieved with this model.

JUSTIFICATION OF PARALLELISM IN ACO

As said, instances of the ACO metaheuristic are easily parallelizable techniques. This parallelism have been previously used (Middendorf, Reischle & Schmeck, 2002) (Michel & Middendorf, 1998) (Gambardella, Taillard & Agazzi, 1999) in several different ways and with several different targets.

First of all, we should notice that there exist two completely different kinds of parallel approaches in ACO. The first one, which has been more extensively used, deals with the possibility of exploiting more computing power using several machines to solve the problem. That is, we use some (usually inter-connected) computers to solve a single instance of a problem, implementing some kind of parallelism and some kind of information exchange between processors.

Usually simple island models (where a processor works as a master and the rest are slaves) have been applied with this purposes, because of the system simplicity. The master processor usually manages all pheromone updates as well it orders the slave processors to compute new solutions, that is, slave processors generate new solutions (applying the usual ant behaviour and transition rule) while the master only receives the solutions generated, updates pheromone and provides the needed memoristic information to the slaves. Other more complicated models use a tree structure where some slave nodes just apply a local search technique (as it can be slow, it is interesting to spend specialized nodes just for this optimization). Both approaches are shown in *figure 2*.

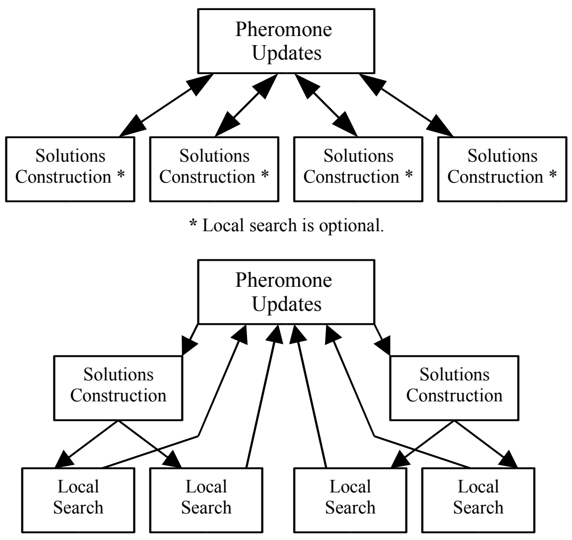


Figure 2. *Different topologies to implement a parallel ACO with several processors.*

This kind of parallelization is often known as a fine-grained parallelization, because each node do have different and very concrete functions. As said, these kinds of approaches try to increase the quality of the solutions by "simply" adding more computer power. The basics of the algorithm are preserved and only vary the location where each element is calculated.

We should be aware of one of the largest problems that this kind of parallelism brings. The speed of our algorithms can not be expected to increase proportionally to the number of processors used. Instead of it, there is a much lower limit that can not be overcome. This side-effect is known as *Amdahl's Law* (Hennesy & Patterson, 2003), and it happens because of communication costs and overheads between nodes. As we can see in *figure 2*, there exist communication between all nodes, specially between the master node to all other processors. We should try to minimize the length of the communications between them to avoid efficiency losses. One of the main problems is that slave nodes which have to generate new solutions need the memoristic information that is stored and updated in the master. Sending the whole pheromone trail matrix can be a very limiting factor for the algorithm. On the other hand, setting the local search process in new processors can be a good improvement, because it does not need memoristic information, just the heuristic one, which can be computed in the early stages of the algorithm by each node (it does not change during the run time).

The other kind of parallelization is not usually made (although in some cases it could be done) through the use of several machines, but in a single one. That new approach involves *several* different and independent colonies that at certain points share some kind of information. In this case, this is an example of coarse-grained parallelization. Usually it does not try to exploit more available computing power, but to offer a more sophisticated scheme that produces (hopefully) better quality solutions with the same computing effort.

In this approach, we have introduced some EC concepts to this kind of parallelization. As it will be explained later, all the different colonies can be seen as chromosomes in a simple *Genetic Algorithm (GA)*.

A NEW PARALLEL ACO MODEL WITH EC COMPONENTS

The first approach made to construct a multi-colony ACO algorithm is to run an existing algorithm several independent times. This kind of approach gives more exploration power to the algorithm, because with every new run, different zones in the search space could be explored. However, the exploitation side of the algorithm would be penalized: every independent execution would have less time to find solutions and the most promising paths could not be explored thoroughly.

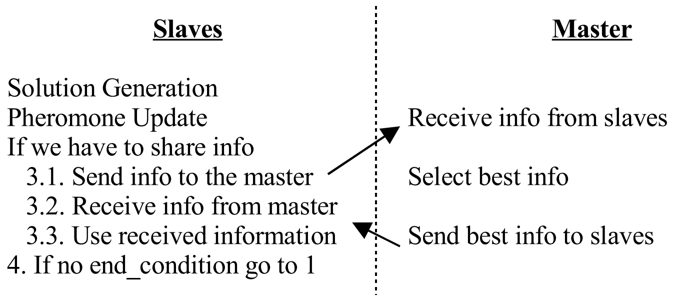
The next step in the construction of the new model was to introduce some kind of information exchange between colonies. They will run independently but, at certain points, they will exchange some useful information which could be used by the other colonies.

One of the simplest information exchange schemes is to exchange the best ant (the best solution) found so far by the colony. The other colonies can use this solution to reinforce those paths with an additional amount of pheromone, what would increase the exploitation of good arcs.

A more complex information exchange scheme involves the sharing of the whole memoristic structure, that is, the pheromone matrix. In this case, there exist different options about what to do with that matrix. A simple option is that the different colonies share their matrices and continue their execution using the best one found so far. Other possibilities involve some crossovers between matrices, and so on. All the different existing approaches will be detailed later.

These approaches share some properties that have to be discussed. Apart from what sort of information to share and when to share it, there exist several different ways of exchanging the information. Three of the most important ones (taken from the parallel GA field (Cantú-Paz, 2000)) are the *master / slave scheme*, the *ring structure* and *random pairs scheme*.

In the *master / slave scheme*, all colonies share their information at a certain point, and the master selects and sends the best piece of that information to the slaves. This scheme is described in more detail in *figure 3*.

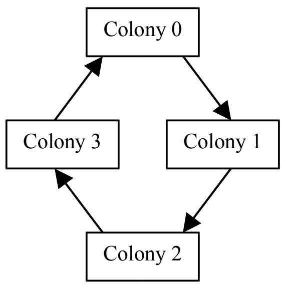


```
Slaves                                   Master

Solution Generation
Pheromone Update                         Receive info from slaves
If we have to share info
  3.1. Send info to the master           Select best info
  3.2. Receive info from master
  3.3. Use received information          Send best info to slaves
4. If no end_condition go to 1
```

Figure 3: Master / Slave scheme to share information. The information shared could be the best solution found so far, the whole pheromone matrix, and so on.

The *ring structure* order all colonies and lets the information exchange occur in pairs of consecutive colonies. This process is detailed in *figure 4*.



```
Colony i

1. Solution Generation
2. Pheromone Update
3. If we have to share info
  3.1. Send info to (i + 1)
  3.2. Receive info from (i – 1)
  3.3. Use received information
4. If no end_condition go to 1
```

Figure 4: Ring structure for sharing information

A *random pairs scheme* groups the colonies in pairs and lets the information exchange happen between both colonies in every pair.

Another problem that we have to solve is that, when colonies share their pheromone matrices, there is no easy way of comparing their fitness, that is, we do not know how good is a pheromone matrix compared with another one. That makes the decision of choosing one or another difficult. There are several possibilities to solve this problem. One of the easiest chances is to consider the fitness of the best solution found as the fitness of the whole matrix. This can not be very precise, because as it has been said, ACO uses a stochastic method to construct solutions, and a poor quality matrix could produce, eventually, a very good solution, while a better matrix could not have been produced very good solutions yet. Anyway, good matrices usually generate good solutions, so we will use this approximation to solve that problem.

The last point we should focus on, is how the information, once it has been shared, is used in the colonies. When colonies share the best solution found (the "best ant"), the only thing that can be usually done with it is to deposit some extra pheromone on the paths contained in that solution. That would increase the desirability of those paths, thus allowing a higher exploitation of these tours.

When the information exchanged is the whole pheromone trail matrix, we could choose among different possibilities. The easiest one is to replace the current pheromone matrix if the one received if it has a better fitness. This option would work as creating "meeting points" where different matrices could evolve independently, but at a certain point all colonies would start using the same good matrix (and from that point every colony would evolve independently again).

Another possibility is to make some kind of crossover between matrices, as if they were very large chromosomes in a sort of GA. This crossover could be done following very different schemes. In our new model, we have used a crossover operator inspired in fuzzy logic elements (Herrera, Lozano & Verdegay, 1996) which tries to preserve the most relevant information of each matrix while also tries to achieve a greater diversity in order to avoid stagnation in the algorithm.

This crossover operator performs a weighted average between all elements in both matrices using the fitness of the matrix as the balancing parameter. That is:

$$\tau_{rs}^{\bar{m}} \leftarrow \mu \cdot \tau_{rs}^{m_1} + (1-\mu) \cdot \tau_{rs}^{m_2} \quad \forall \; arc \; (r,s)$$

where

$$\mu = \frac{fit(BS(m_1))}{fit(BS(m_1)) + fit(BS(m_2))}$$

and where *fit(BS(matrix))* is the *fitness* of the *Best Solution* found by the *matrix*.

Graphically, the result that we would like to obtain is represented in *figure 5*.

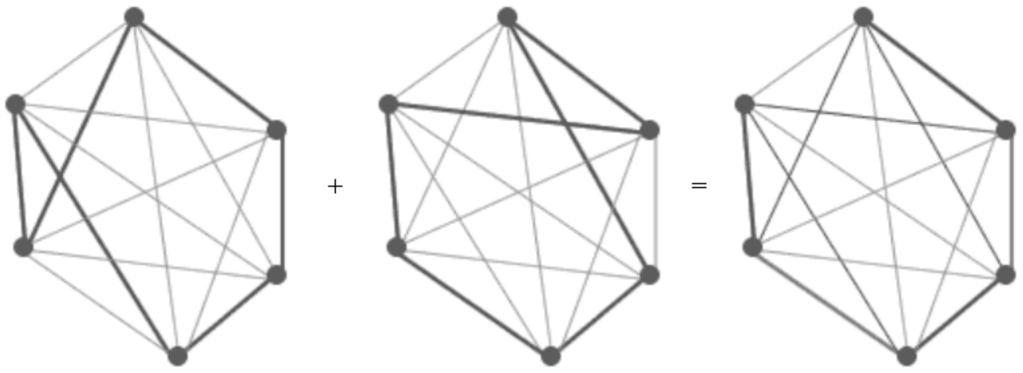


Figure 5: Example of average of the pheromone matrices. Darker arcs have stronger pheromone trails. Following the average, the most promising arcs (the arcs in the shortest tour) have more pheromone, allowing the best solution to be found.

EXPERIMENTS AND ANALYSIS OF RESULTS

To test the performance of the new algorithm some experimentation with different algorithms over instances of the TSP has been done. All of them are based on the Elitist AS (EAS) but include some of the features that we have described in the previous sections. The list of all the algorithms is shown on *table 4*.

Algorithm	**Description**
EASI_T1	Independent Executions of the EAS (1 colony)
EASI_T2	Independent Executions of the EAS (2 colonies)
EASI_T4	Independent Executions of the EAS (4 colonies)
EASI_T8	Independent Executions of the EAS (8 colonies)
EASA_M/S	Best Ant Exchange in EAS ; Master / Slave Configuration (8 colonies)
EASA_Ring	Best Ant Exchange in EAS ; Ring structure (8 colonies)
EASM_M/S	Matrix Exchange in EAS ; Master / Slave Configuration (8 colonies)
EASM_Ring	Matrix Exchange in EAS ; Ring structure (8 colonies)
EASM_Ave	Matrix Averages in EAS ; Random Pairs Scheme (8 colonies)

Table 4: Used algorithms in the comparison.

The first four algorithms are identical but on the number of colonies used. With this four experiments, we wanted to show if just the idea of parallelization obtains better results than using a single colony. The following algorithms try to show the new ideas exposed, from the best ant exchange or pheromone matrix exchange until the different schemes of communication (master / slave, ring topology and random pairs scheme).

Every algorithm has been run 30 times without local search and another 30 times using the enhanced 2-opt local search (see *section 3.1.3*). All other parameters used are shown in *table 5*.

The instances used in this experimentation are *eil101*, *gr120* and *att532*.

Parameter	Value
Global number of ants	*m = 40*
Number of colonies	*t = 8* (Except in EASI)
Ants per Colony	*m / t*
Elitist Ants	*e = m / t*
Run Time	*900 seconds*
Transition Rule	$\alpha=1,\ \beta=2$
Global Evaporation Constant	$\rho=0.2$
Number of executions	*30*
Exchange of Information	every *it = 10*
Candidate List Size	20

Table 5: Parameters used for TSP.

All the results obtained are shown in *table 6*. In that table, we can see the mean of all executions, the standard deviation, best solution and error as they where defined in *section 3.1.3*.

The results are clear. For small instances, our new model with crossover of pheromone matrices is the best performing. In *eil101* and *gr120* without local search, the smaller error was achieved with the EASM_Ave. That gives the highest degree of robustness for this algorithm (although the best solutions were not found by this method).

It is also shown, for the first two problems, that paralellization seems to be a good technique to improve results (the error was being reduced when new colonies were incorporated to the algorithm).

When the problem size increases (*att532*), all parallel algorithms lose efficiency, and the best performing is (with or without local search) the single colony algorithm. The main reason for that behaviour is that the time used to solve the problem was not enough and parallel algorithms had to distribute their execution time over several colonies, thus not allowing to enough exploit their search spaces.

Another important fact is that local search always improve the results (as it was expected) but it does not interfere with algorithms, that is, local search improve the performance of every algorithm almost in the same proportion (as it can be seen from the executions of the *att532* instance). That property allows us for future research to experiment only with algorithms including local search assuring that all benefits acquired are made by means of changes in the algorithms, not by means of the local search technique.

Instance	**eil101 without Local Search** (C_{opt} = 629)								
Algorithm	*EASI_T1*	*EASI_T2*	*EASI_T4*	*EASI_T8*	*EASA_M/S*	*EASA_Ring*	*EASM_M/S*	*EASM_Ring*	*EASM_Ave*
Mean	648.33	644.97	644.56	644.83	645.67	641.73	647.8	646.43	**639.77**
Std. Dev.	6.53	5.39	5.46	5.11	6.33	6.46	7.15	6.73	**6.20**
Best	634	635	634	631	**629**	630	635	632	631
Error	3.07	2.54	2.47	2.52	2.65	2.02	2.99	2.77	**1.71**
Instance	**eil101 WITH Local Search** (C_{opt} = 629)								
Algorithm	*EASI_T1*	*EASI_T2*	*EASI_T4*	*EASI_T8*	*EASA_M/S*	*EASA_Ring*	*EASM_M/S*	*EASM_Ring*	*EASM_Ave*
Mean	629	629	629	629	629	629	629	629	629
Std. Dev.	0	0	0	0	0	0	0	0	0
Best	629	629	629	629	629	629	629	629	629
Error	0	0	0	0	0	0	0	0	0
Instance	**gr120 without Local Search** (C_{opt} = 6942)								
Algorithm	*EASI_T1*	*EASI_T2*	*EASI_T4*	*EASI_T8*	*EASA_M/S*	*EASA_Ring*	*EASM_M/S*	*EASM_Ring*	*EASM_Ave*
Mean	7182.20	7154.2	7161.4	7144.13	7170.5	7146.9	7181.1	7159	**7102.56**
Std. Dev.	81.04	68.55	44.50	38.22	78.00	65.90	65.60	62.16	**45.84**
Best	7066	7034	7040	7067	7026	**7010**	7027	7066	7035
Error	3.46	3.06	3.16	2.91	3.29	2.95	3.44	3.12	**2.31**
Instance	**gr120 WITH Local Search** (C_{opt} = 6942)								
Algorithm	*EASI_T1*	*EASI_T2*	*EASI_T4*	*EASI_T8*	*EASA_M/S*	*EASA_Ring*	*EASM_M/S*	*EASM_Ring*	*EASM_Ave*
Mean	6942	6942	6942	6942	6942	6942	6942	6942	6942
Std. Dev.	0	0	0	0	0	0	0	0	0
Best	6942	6942	6942	6942	6942	6942	6942	6942	6942
Error	0	0	0	0	0	0	0	0	0
Instance	**att532 without Local Search** (C_{opt} = 27686)								
Algorithm	*EASI_T1*	*EASI_T2*	*EASI_T4*	*EASI_T8*	*EASA_M/S*	*EASA_Ring*	*EASM_M/S*	*EASM_Ring*	*EASM_Ave*
Mean	**31272.3**	31383.2	31372.1	31558.5	31604.0	31643.9	31451.9	31304.1	31551.5
Std. Dev.	**353.4**	222.9	266.7	267.1	263.4	321.8	333.9	292.9	237.4
Best	**30581**	30997	30838	30975	31158	31080	30588	30840	30965
Error	**12.95**	13.35	13.3	13.99	14.15	14.30	13.6	13.06	13.9
Instance	**att532 WITH Local Search** (C_{opt} = 27686)								
Algorithm	*EASI_T1*	*EASI_T2*	*EASI_T4*	*EASI_T8*	*EASA_M/S*	*EASA_Ring*	*EASM_M/S*	*EASM_Ring*	*EASM_Ave*
Mean	**28045.9**	28079.3	28077.0	28095.1	28082.63	28100.36	28079.1	28080.4	28156.2
Std. Dev.	**53.39**	39.48	44.04	50.11	62.86	50.18	47.60	45.95	53.91
Best	**27927**	28002	27962	28002	27931	28017	27983	27946	28024
Error	**1.30**	1.42	1.41	1.48	1.43	1.50	1.42	1.42	1.70

Table 6: Results of the experiments.

FUTURE WORKS AND RESEARCH LINES

The new ACO models with EC components proposed are an interesting research field which can produce good and efficient algorithms to solve very different sorts of problems.

More specifically:

- More exhaustive statistic analysis should be done to demonstrate the efficiency of both models. For this purpose, more instances of the TSP problem will be evaluated as well some as new well-known problems as the QAP will be analysed using different statistic tests.

- Both new models can be applied to solve different Bioinformatics problems, concretely the gene regulation process and the construction of genetic networks from DNA sequence analysis and oligonucleotide microarrays.

- Parallelization can also be introduced in different ant algorithms, not just in the EAS, but in ACS or BWAS to achieve better computational results.

- New EC concepts can also be introduced into existing ACO algorithms to obtain, for example, good solutions on multi-modal optimization problems. It will be necessary a deeper insight into EC and ACO algorithms to identify all exploitation and exploration components in both techniques.

CONCLUDING REMARKS

ACO metaheuristic is one of the most recent bioinspired metaheuristics. It simulates ants' behaviour when they search for food to solve complex combinatorial problems. EC is a set of different computational techniques which work by emulating evolution in nature. These techniques include very well studied algorithms, like Genetic Algorithms or Evolutionary Programming.

Despite the differences between ACO metaheuristic and EC, both share properties that allow to improve the results obtained by both approaches constructing special ACO algorithms which include some EC components.

In this work, we have introduced both ACO metaheuristic and EC to afterwards present two new ACO algorithms which integrate some EC concepts to improve their performance; as well as some empirical results on their performance.

We have also emphasized the importance of a good balance between exploration and exploitation of the search space for optimization algorithms (particularly on ACO), and have explained the behaviour of every new EC component that has been added to the new models.

REFERENCES

Aarts, E. H. L., Korst, J.H.M. (1990). *Simulated Annealing and Boltzmann Machines*. Wiley, Chichester.

Aarts, E. H. L. & Lenstra, J. K. (Eds.). (1997). *Local Search In Combinatorial Optimization*. Chichester: John Wiley & Sons.

Bäck, T. (1996). *Evolutionary Algorithms in Theory and Practice*. New York: Oxford University Press.

Bäck, T., Fogel, D. & Michalewicz, Z. (Eds.). (1998). *Handbook of Evolutionary Computation*.

Baluja, S. & Caruana, R. (1995). Removing the Genetics from the Standard Genetic Algorithm. In Prieditis, A. & Rusell, S. (Eds.), *Machine Learning: Proceedings of the Twelfth International Conference*, 38-46. Morgan Kaufmann Publishers.

Bentley, J. L. (1992). Fast algorithms for geometric travelling salesman problem, *ORSA Journal on Computing*, 4: 4, 387-411.

Birattari, M., Stützle, T., Paquete, L. & Varrentrapp, K. (2002). A racing algorithm for configuring metaheuristics. In Langdon, W. B. et al. (Eds.), *GECCO 2002- Proceedings of the Genetic and Evolutionary Computation Conference*, 11-18. San Francisco: Morgan Kaufmann Publishers.

Botee, H. M., Bonabeau, E. (1998). Evolving Ant Colony Optimization. Advances in Complex Systems, Vol. 1, 149-159.

Bonabeau, E., Dorigo, M. & Theraulaz, G. (1999). *Swarm Intelligence. From Natural to Artificial Systems*. Oxford University Press.

Brassard, G. & Bratley, P. (1996). *Fundamentals of Algorithmics*. Englewood Cliffs, NJ: Prentice Hall.

Bullnheimer, B., Hartl, R.F. & Strauss, C. (1999a). *A New Rank Based Version of the Ant System: A Computational Study*. Central European Journal for Ops. Research and Economics, 7(1), 25-38.

Bullnheimer B., Hartl, R. F., & Strauss, C. (1999b). An improved ant system algorithm for the vehicle routing problem. *Annals of Operations Research*, 89, 319-328.

Cantú-Paz, E. (2000). Efficient and Achúrate Parallel Genetic Algorithms. Boston, MA: Kluwer Academic Publishers.

Colorni, A., Dorigo, M., Maffioli, F., Maniezzo, V., Righini, G. & Trubian M. (1996), Heuristics from Nature for Hard Combinatorial Optimization Problems,
International Transactions in Operational Research, 3:1, 1-21.

Colorni, A., Dorigo, M., Maniezzo, V. & Trubian M. (1994). Ant System for job-shop scheduling. *JORBEL - Belgian Journal of Operations Research, Statistics and Computer Science*, 34: 1, 39-53.

Cordón, O., Fernández de Viana, I., Herrera, F. & Moreno, L. (2000), A New ACO Model Integrating Evolutionary Computation Concepts: The Best-Worst Ant System.

Proc. of ANTS'2000, From Ant Colonies to Artificial Ants: Second International Workshop on Ant Algorithms, Brussels, Belgium, 22-29, September 7-9.

Cordón, O., Fernández de Viana, I. & Herrera F. (2002a), Análisis of the Best-Worst Ant System and its variants on the TSP, *Mathware and Soft Computing* 8(3), 177-192.

Cordón, O., Fernández de Viana, I. & Herrera, F. (2002b), Análisis of the Best-Worst Ant System and its variants on the QAP. In Dorigo, M., Di Caro, G., Sampels M. (Eds.), *Ant Algorithms, Lecture Notes in Computer Science*, 2463, 228-234, Berlin, Germany: Springer Verlag.

Cordón, O., Herrera, F. & Stützle, T. (2002), A Review on Ant Colony Optimization Metaheuristic: Basis, Models and New Trends. *Mathware & Soft Computing*, 9: 2-3, 141-175.

Di Caro, G. & Dorigo, M. (1998). AntNet: Distributed stigmergetic control for communications networks. *Journal of Artificial Intelligence Research*, 9, 317-365.

Dorigo, M., Maniezzo, V. & Colorni, A. (1996). The Ant System: Optimization by a Colony of Cooperating Agents. *IEEE Trans. on Systems, Man, and Cybernetics*, Part B, 26(1), 29-41.

Dorigo, M. & Gambardella, L. (1997). Ant Colony System: A Cooperative Learning Approach to the Travelling Salesman Problem. *IEEE Transactions on Evolutionary Computation*, 1(1), 53-66.

Dorigo, M. & Di Caro, G. (1999). The Ant Colony Optimization Meta-heuristic. In Corne, D., Dorigo, M., Glover, F. (Eds.), *New Ideas in Optimization*, 11-32, London, UK: McGraw-Hill.

Dorigo, M., Di Caro, G. & Gambardella, L. M. (1999). Ant Algorithms for Discrete Optimization. *Artificial Life*, 5(2), 137-172.

Dorigo, M., & Stützle, T. (2003). The ant colony optimization metaheuristic: Algorithms, applications and advances. In Glover, F. & Kochenberger, G. (Eds.), *Handbook of Metaheuristics*, 251-285. Kluwer Academic Publishers.

Gambardella, L. M., Taillard, È. D. & Agazzi, G. (1999). MACS-VRPTW: A multiple ant colony system for vehicle routing problems with time windows. In Corne, D., Dorigo M. & Glovar, F. (Eds.), *New Ideas in Optimization*, 63-76. London, UK: McGraw-Hill.

Garey, LM. R. & Johnson, D. S. (1979). *Computers and Intractability: A Guide to the Theory of NP-Completeness*. San Francisco: Freeman.
Glover, F. & Kochenberger, G., (Eds.) (2003). *Handbook of Metaheuristics*. Kluwer Academic Publishers.

Hennesy, J. L. & Patterson, D. A. (2003), *Computer Architecture, A Quantitative Approach*, 40-42, Morgan Kaufmann Publishers.

Herrera, F., Lozano, M. & Verdegay, J. L. (1996). Dynamic and Heuristic Fuzzy Connectives-Based Crossover Operators for Controlling the Diversity and Convergence of Real Coded Genetic Algorithms. *Int. Journal of Intelligent Systems*, 11, 1013-1041.

Johnson, D. S. & McGeoch, L. A. (1997). The Travelling Salesman Problem: A Case Study in Local Optimization, In Arts, E.H.L. & Lenstra, J.K. (Eds.), *Local Search in Combinatorial Optimization*, 215-310, John Wiley & Sons.

Larrañaga, P. & Lozano, J. A. (Eds.). (2001). *Estimation of Distribution Algorithms: A New Tool for Evolutionary Computation*. Dordrecht, The Netherlands: Kluwer Academic Publishers.

Lourenço, H. R., Martin, O. C. & Stützle, T. (2003). Iterated Local Search. In Glover, F., Kochenberger, G. (Eds.), *Handbook of Metaheuristics*, 321-353, Kluwer Academic Publishers.

Maniezzo V., Colorni A., & Dorigo, M. (1994). The Ant System Applied to the quadratic assignment problem. *Technical Report IRIDIA*/94-28, IRIDIA, Université Libre de Bruxelles, Belgium.

Michalewicz, Z. (1996). *Genetic Algorithms + Data Structures = Evolution Programs.* Springer. Third Edition.

Michalewicz, Z. & Fogel, D. B. (2000). *How To Solve It: Modern Heuristics*. Berlin, Germany: Springer Verlag.

Michel, R. & Middendorf, M. (1998). An Island model based Ant system with lookahead for the shortest supersequence problem. In Eiben, A. E., Bäck, T., Schoenauer, M., & Schwefel, H.-P. (Eds.), *Proceedings of PPSN-V, Fifth International Conference on Parallel Problem Solving from Nature,* volume 1498 of *Lecture notes in Computer Science,* 692-701, Berlin, Germany: Springer Verlag.

Middendorf, M., Reischle, F. & Schmeck, H. (2002). Multi colony ant algorithms. *Journal of Heuristics*, 8: 3, 305-320.

Mladenovic, N. & Hansen, P. (1997). Variable Neighborhood Search. *Computers & Operations Research*, 24, 1097-1100.

Monmarché, N., Ramat, E., Dromel, G., Slimane, M. & Venturini, G. (1999). On the similarities Between AS, BSC and PBIL: Toward the Birth of a Now Meta-Heuristic. *Technical Report 215, Ecole d'Ingénieurs en Informatique pour l'Industrie (E3i)*, Université de Tours.

Papadimitriou, C. H. & Steiglitz, K. (1982). *Combinatorial Optimization - Algorithms and Complexity*. Englewood Cliffs, NJ: Prentice Hall.

Reeves, C. (Ed.) (1995). *Modern Heuristic Techniques for Combinatorial Problems*. London, UK: McGraw Hill.

Reinelt, G. (2001, December, 6), TSPLIB, retrieved May 17, 2003 from: http://www.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95/

Schoonderwoerd, R., Holland, O., Bruten, J., & Rothkrantz, L. (1996). Ant-based load balancing in telecommunications networks. *Adaptative Behavior*, 5: 2, 169-207.

Stützle, T. & Hoos, H. (2000). MAX-MIN Ant System. *Future Generation Computer Systems Journal*, 16(8), 889-914.

[*] This work has been supported by the University of Granada under project “Mejora de Metaheurísticas Mediante Hibridación y sus Aplicaciones”.

[i] As said in (Dorigo & Stützle, 2003), the set of edges may fully connect the components. In this case, the implementation of the constraints is fully integrated into the construction policy of the ants.

[ii] This aspect will be analyzed in depth in the next sections when introducing specific ACO algorithms.