ANT COLONY OPTIMIZATION: AN OVERVIEW

VITTORIO MANIEZZO, ANTONELLA CARBONARO

Scienze dell’Informazione, University of Bologna, Italy.

Abstract. Ant Colony Optimization (ACO) is a class of constructive meta-heuristic algorithms sharing the common approach of constructing a solution on the basis of information provided both by a standard constructive heuristic and by previously constructed solutions. This tutorial is composed of three parts. The first one frames the ACO approach in current trends of research on metaheuristic algorithms for combinatorial optimization; the second outlines current research within the ACO framework, reporting recent results obtained on different problems, while the third part focuses on a particular research line, the ANTS metaheuristic, providing some details on the algorithm and presenting results recently obtained on the quadratic and on the frequency assignment problems.

1. Introduction

Complex societal systems pose a lot of problems of combinatorial nature. Trucks have to be routed, depots or sale points have to be located, communication networks have to be designed, containers have to be filled, radio links must have an associated frequency, wood or leather masters have to be cut: the list could get long indeed. Complexity theory tells us that the combinatorial problems underlying these applications are often not polynomial. Actual practice tells us that, unfortunately, the size of the instances met in real world applications rules out the possibility of solving them to optimality in an acceptable time. Nevertheless, these instances must be solved, thus the need to bow down to suboptimal solutions, provided that they are of acceptable quality and that they can be found in acceptable time.

Several approaches have been proposed for designing "acceptable quality" solutions under the "acceptable time" constraint. Besides studying the particular instances of interest, in the hope of being in a polynomial special case, one can in fact try to design an algorithm that guarantees to find a solution within a known gap from optimality, i.e. an approximation algorithm, or try to design an algorithm that guarantees that for instances big enough, the probability of getting a bad solution is very small, i.e. a probabilistic algorithm. Or one can lower even more the expectations and accept an algorithm that offers no guarantees, in front of some evidence that historically, on the average, that algorithm has the best record track on
the quality/time trade off for the problem of interest. This is the area of heuristic algorithms.

Heuristic algorithms have a long tradition, the actual necessity to solve difficult instances has pushed research in the area. It is possible to identify different approaches to how to design such algorithms. We maintain in this paper that it is possible to identify in the literature three main classes of heuristic algorithms. The historically first one concentrates on structural properties of the problem to solve and tries to use them for defining constructive or local search algorithms. Algorithms following the second approach are those usually denoted by the term "metaheuristics", a name first proposed by Glover in [24], and focus on the guidance of a constructive or a local search algorithm in order to allow it to overcome critical situations (which usually means to escape from local optima). Finally, a recent trend is focused on incorporating strong results from mathematical programming practice, usually from exact solution method design, into an heuristic framework.

The paper presents a brief overview on the essential features of these three types of heuristic paradigms, then focuses on one particular family of algorithms, those that follow the Ant Colony Optimization (ACO) paradigm. Finally, the third part of this overview describes in some more detail one algorithm of the ACO class, called ANTS.

The paper is structured as follows. Section 2 introduces the general issues that contribute to define the three classes of heuristic algorithm design approaches introduced above. Section 3 describes the common elements of the heuristics belonging to the ACO class. Section 4 concentrates on the ANTS approach, describing its essential ingredients, while section 5 presents computational results obtained by ANTS algorithms on two well known combinatorial optimization (CO) problems: the Quadratic Assignment and the Frequency Assignment problems. Finally, Section 6 presents a brief discussion on the issues raised by this overview.

2. Heuristic framework

This section introduces a classification of heuristic algorithms into three classes, defining some essential traits of each of them and presenting typical examples of algorithms of each class. Notice that, although the three classes of algorithms were successively presented in the literature, this does not necessary reflect on the comparative effectiveness of the algorithms: specific problems are best solved by either of them.

All algorithms will be described according to the following common problem framework. A combinatorial optimization problem is defined over a set \( C = \{c_1, \ldots, c_n\} \) of basic components. A subset \( S \) of components represents a solution of the problem; \( F \subseteq 2^C \) is the subset of feasible solutions, thus a solution \( S \) is feasible if and only if \( S \in F \). A cost function \( z \) is defined over the solution domain, \( z : 2^C \rightarrow \mathbb{R} \), the objective of the algorithm is to find a minimum cost feasible solution \( S^* \), i.e., to find \( S^* \in F \) such that \( z(S^*) \leq z(S), \forall S \in F \). Failing this, the algorithm anyway returns the best feasible solution found, \( \hat{S} \in F \).

As a running example of CO problem, we can use the Traveling Salesman Problem (TSP), defined over a weighted undirected complete graph \( G=(V,E,D) \), where \( V \) is the set of vertices, \( E \) is the set of edges and \( D \) is the set of edge weights, corresponding to distances between vertices. For the TSP we have that the component
set corresponds to $E$, that $F$ corresponds to the set of Hamiltonian cycles in $G$ and that $z$ is defined to be, for each feasible solution $S$, the sum of the weights associated to the edges belonging to $S$.

2.1. Type 1 heuristics: focus on solution structure

Type 1 heuristics are methods that exploit structural properties of a feasible solution in order to quickly come to a good feasible solution. Generally they belong to one of two main classes: constructive heuristics or local search heuristics.

Constructive heuristics are based on the following general framework.

1. Sort the components in $C$ by increasing costs.
2. Set $\bar{S} = \emptyset$ and $i = 1$.
3. Repeat
   
   If $(\bar{S} \cup \{c_i\}$ is a partial feasible solution) then $\bar{S} = \bar{S} \cup \{c_i\}$.
   
   Until $\bar{S} \in F$.

Notice that such a simple approach can yield optimal solutions for certain problems, eg. the minimum spanning tree of a graph, while in other cases it could be unable even to construct a feasible solution.

In the case of the TSP this approach would suggest to order all edges by increasing edge length, take the least cost one and add to it increasing cost edges, provided they do not close subtours, until a Hamiltonian circuit is completed. More involved constructive strategies give rise to well-known TSP heuristics, like the Farthest Insertion, the Nearest Neighbor or the Sweep algorithms.

Local Search heuristics preliminarily need the definition of a solution neighborhood. The neighborhood of a solution $S$, $N(S)$, is a subset of $2^C$ defined by a specific neighborhood definition function $N : 2^C \rightarrow 2^{2^C}$. It is common practice, though not mandatory, to restrict the attention to feasible solutions only, thus to neighborhood definition functions in the form $N : F \rightarrow 2^F$. The specific neighborhood function used has a deep impact on the algorithm performance and its choice is left to the algorithm designer.

Given $N$, a local search algorithm is an algorithm with the following structure.

1. Generate an initial feasible solution $S$.
2. Find $S'$, such that $S' \in N(S)$ and $z(S') = \min\{z(\bar{S}), \forall \bar{S} \in N(S)\}$.
3. If $z(S') < z(S)$
   
   $S = S'$, go to step 2.

4. $\bar{S} = S$.

The update of a solution, in step 3, which transforms solution $S$ into solution $S'$ is called a move from $S$ to $S'$.
Despite its apparent triviality, the design of a good local search algorithm can be very intricate. While we refer the reader to [37] and [2] for a thorough discussion, we just want to point out that a deep impact decision is, at step 2, whether to explore the whole of \( N(S) \), find the best solution \( S' \) in it and check if this improves over \( S \), or to update \( S \) as soon as an improving \( S' \) solution is found, even if it not guaranteed to be the best in \( N(S) \).

Several mathematical properties can be studied on the structure of the neighborhood sets, and it is possible to prove that there are problems for which a local search approach guarantees to find an optimal solution: suffice it to say that the simplex algorithm essentially follows the above framework.

In the case of the TSP, two very simple and widely used local search methods are 2-opt and 3-opt, which take a solution in the form of a list of \( n \) vertices and exhaustively try to swap the elements of all pairs or triplets of vertices in it, respectively. More complicated neighborhoods give rise to more effective heuristics, among which the method of Lin and Kernighan [30] deserves a special citation.

### 2.2. Type 2 heuristics: focus on heuristic guidance

Heuristics of type 1 can perform very well, at times optimally, but can also get stuck with very poor quality solutions. As mentioned, one of the major problems to face is the possibility to get trapped in local optima, with no chance to escape from them. To improve the quality of the solutions they can produce, thus the size of the problems that can be satisfactorily solved, a new wave of approaches has been presented starting from about the mid '70s. They consist in algorithms which explicitly or implicitly manage the trade-off between search \textit{diversification}, when evidence exists that search is going on in bad regions of the search space, and \textit{intensification}, aimed at finding the best solutions within the region being analyzed.

These algorithms have been called \textit{metaheuristics}. Among the best known ones, we can list:

1. Simulated Annealing;
2. Tabu Search;
3. GRASP;

The list could be much longer than this, since this field of research is still very active as this volume testifies. We concentrate on these just to show how the diversification/intensification issue was tackled in each of them and to set the stage for an analogous discussion in the case of ACO.

#### 2.2.1. Simulated Annealing

The essential idea motivating Simulated Annealing (SA) [1] is to modify local search in order to accept, in probability, worsening moves. The framework of SA is:

1. Generate an initial feasible solution \( S \), initialize the best so far solution \( \bar{S} = S \) and initialize parameter \( T \).
2. Generate $S'$, where $S' \in N(S)$.
3. If $z(S') < z(S)$ then $S = S'$, if $(z(\tilde{S}) > z(S)) \tilde{S} = S$
   else accept to set $S = S'$ with probability $p = e^{-\frac{z(S') - z(S)}{T}}$.
4. If (annealing condition) decrease $T$.
5. If not (end condition) go to step 2.

SA, with respect to local search, only accepts partial neighborhood explorations
and implements the intensification/diversification strategy by means of the
annealing (decrease) of parameter $T$. The end condition can be on the CPU time elapsed,
on the number of iterations or on other parameters specific of the run.

2.2.2. Tabu Search

The idea behind Tabu Search (TS) [27] is again to modify local search in order
to escape from local minima. In the TS case this is accomplished by moving onto
the best solution of the neighborhood at each iteration, even though it is worse
than the current one. A special memory structure called tabu list, TL, forbids to
return to already explored solutions. The general structure of the TS algorithm is
as follows.

1. Generate an initial feasible solution $S$, set $\tilde{S} = S$ and initialize $TL = \emptyset$.
2. Find $S'$, such that $S' \in N(S)$ and $z(S') = \min \{z(\tilde{S}), \forall \tilde{S} \in N(S), \tilde{S} \notin TL\}$.
3. $S = S'$, $TL = TL \cup S$, if $(z(\tilde{S}) > z(S))$ set $\tilde{S} = S$.
4. If not (end condition) go to step 2.

In the case of TS the intensification/diversification strategy must be explicitly
implemented with reference to the actual memory structures used to store the tabu
list. Several strategies are possible, we refer the interested reader to [27] for a
discussion.

2.2.3. GRASP

GRASP (standing for Greedy Randomized Adaptive Search Procedure) [18]
departs from the SA and TS line of trying to escape from local minima, in that the
idea here is to restart search from another promising region of the search space as
soon as a local optimum is reached. GRASP essentially consists in a multistart
approach, i.e., an approach which simply iterates local search starting from different
initial solutions, with a suggestion on how to construct the initial solutions. GRASP
combines the two algorithms of section 2.1 in the following common framework.

1. Build a solution $S$ (= $\tilde{S}$ in the first iteration) by a constructive greedy randomized
   procedure.
2. Apply a local search procedure starting from $S$ and producing $S'$.
3. If $z(S') < z(S)$ then $\tilde{S} = S'$.
4. If not (end condition) go to step 2.
The greedy randomized procedure of step 2 is the constructive heuristic of section 2.1 where, at step 3, the component to append is not necessarily the first feasible one but a feasible one chosen in probability among the most promising ones (i.e., at the top of the list). The diversification/intensification issue is handled by controlling the randomization of the constructive procedure.

2.2.4. Genetic Algorithms

Genetic Algorithms (GA)\cite{28} follow a completely different approach from those so far presented, in that they do not rely on construction or local search but on the parallel update of a set $R$ of solutions. This is achieved by recombining subsets of 2 elements in $R$, the parent sets, to obtain new solutions. Actually, GA should not be properly called a metaheuristic algorithm, since they do not guide a lower level heuristic. However, to be effective on CO problems they are often hybridized with SA, TS or local search. The general framework is as follows.

1. Initialize a set $R$ of solutions.
2. Construct a set $R_c$ of solutions by recombining pairs of randomly chosen elements in $R$.
3. Construct a set $R_m$ of solutions by randomly modifying elements in $R_c$.
4. Construct the new set $R$ by extracting elements from $R_m$, following a montecarlo strategy with repetitions.
5. If not(end condition) go to step 2.

Different, often problem-dependent solutions have been proposed for the recombination, the modification and the extraction operators at steps 2, 3 and 4, respectively. The diversification/intensification strategy here depends on the details of the operators above, which could lead to sets of similar solution, thus fostering intensification, or to sets of solution widely scattered over the search space, thus promoting diversification.

2.2.5. Ant System

Finally, a first framing of an ACO algorithm. By Ant System (AS) we mean the first ACO algorithm presented (\cite{10}, \cite{14}, \cite{13}). The idea underlying was to modify a constructive heuristic, in the spirit of the first step of GRASP, so that the ordering of the components could be recalculated at each iteration taking into account not only the a priori expectation, $\eta_k$, of the usefulness of a particular component $c_i$, as in standard constructive approaches, but also an a posteriori measure, $\tau_i$, of the goodness of solutions constructed using that particular component. The general framework is as follows.

1. Initialize a set $A$ of partial solutions: $a_i = \emptyset, i = 1, \ldots, m$.
2. For $i=1$ to $m$
   choose a component $c_j$ to append to solution $a_i$ with probability given as a function of $a_i, \eta_j, \tau_j$.
3. If the solutions in $A$ are not complete solutions, go to step 2.
4. Evaluate \( z(a_i), i = 1, \ldots, m \) and update \( \tau_j, j = 1, \ldots, n \) accordingly.

5. If not (end condition) go to step 1.

Step 4 can be conveniently integrated with a local search starting from each \( a_i \) solution in \( A \). Diversification and intensification are essentially controlled by the \( \tau \) management policy, which will be detailed in sections 3 and 4.

The general ACO framework can be specialized in different ways, we mention here the possibility of including mathematical programming results, which lead the resulting algorithms to be more suitably included in the third class of heuristics, see for example subsection 2.3.3.

2.3. Type 3 heuristics: focus on mathematical programming contribution

It can be noticed that all algorithms above rely on simple general ideas. Obviously, when they are better detailed and adapted to a particular problem, mathematical considerations become an issue. However, the design of a metaheuristic for a problem can be done neglecting much of the mathematical programming results obtained on that same problem. A straightforward consideration is that results derived from the study of exact approaches could be exploited also in the framework of heuristic design. This is an issue common to some recently presented techniques. Actually, an hybridization of lower bounding techniques like linear relaxation or subgradient optimization of Lagrangean relaxations with feasibility recovering techniques is standard practice in exact algorithms design; the main difference between these established techniques and the approaches introduced in this subsection is in the focus on primal feasible solution update that characterizes heuristic methods.

In the following we will outline three algorithms: bionomic algorithms, scatter search techniques and ANTS procedures.

2.3.1. Bionomic Algorithms

Bionomic Algorithms (BA), proposed by N.Christofides [9], follow the GA framework, but replace much of the randomness of the GA operators with more normative procedures. Moreover, BA accept multiple parent combinations and variable cardinality solution sets, which are both elements that enhance the algorithm performance on CO problems. While the actual structure of a BA cannot be described in few steps, the general framework is the following:

1. Initialize the set \( R \) of solutions.
2. Local optimization of each solution in \( R \).
3. Define a suitable adjacency graph over \( R \) and use it to define parent sets.
4. Use a randomized, many to one mapping to generate solutions from parent sets.
5. Update \( R \) with the solutions found at step 4.
6. If not (end condition) go to step 2.

The essential elements, with respect to GA, are the adjacency graph for the definition of the parent sets, the structural use of local search and the room provided
for tailored recombination operators. Thus, besides local search, we have variable cardinality solution sets and multiple parent recombination.

2.3.2. Scatter Search Techniques

Scatter Search (SS) is a technique originally presented by Glover in [23], but which has only recently been applied to CO problems. Two essential papers present it: a more normative one in [25] and a more general one in [26]. By making reference to [25], SS is a technique which combines polyhedral cutting planes approaches with primal solutions recombination and update. The general framework for an integer problem, IP, whose linear relaxation is LP, is the following.

1. Initialize the Reference Set $R$ with feasible IP solutions.
2. Generate a set $H$ of extreme points adjacent to the optimal LP solution.
3. Define parent sets as subsets containing both elements of $H$ and of $R$.
4. Construct a set $Q$ of seed solutions with points obtained from linear combination of points in the parent sets.
5. Carry the solutions in $Q$ to their corresponding local optima.
6. Update $R$ with the best solutions in $Q$ obtained at step 5.
7. Find a violated valid inequality for problem IP. If no violated inequality can be identified or the maximum number of iterations is reached then STOP, otherwise add the inequality to LP and go to step 2.

Once again, we find variable cardinality solution sets and multiple parent recombination. The difference with BA is that SS is much more oriented to linear recombinations of parent sets to generate new solutions and it is deeply interleaved with the cutting plane process. The algorithms thus evolves at the same time a lower and an upper bound of the problem to solve: a characteristic shared only with ANTS but of great computational impact.

2.3.3. ANTS Procedures

Finally, ANTS ([31]) is an extension of AS presented in subsection 2.2.5, pre-scribing the use of lower bounds in order to determine the a priori effectiveness of a move and a modified update procedure for the determination of the a posteriori effectiveness. Specifically, to compute $\eta_j, j = 1, \ldots, n$, it is suggested to use a lower bound to the cost of completing the current partial solution and to update $\tau_j, j = 1, \ldots, n$, it is suggested a moving average on the last solutions produced. These modifications do not affect the general structure of the algorithm of subsection 2.2.5, but have a significant impact on its effectiveness on CO problems.

These issues will be described in detail in Section 4.

3. Ant Colony Optimization

Ant Colony Optimization was initially proposed by Coloni, Dorigo and Maniezzo [10], [14]. The main underlying idea was that of parallelizing search over several
constructive computational threads, all based on a dynamic memory structure incorporating information on the effectiveness of previously obtained results and in which the behavior of each single agent is inspired by the behavior of real ants. The importance of this original AntSystem, AS, resides mainly in being the prototype of a number of ant algorithms which have found interesting and successful applications.

An ant is defined to be a simple computational agent, which iteratively constructs a solution for the problem to solve. Partial problem solutions are seen as states; each ant moves from a state \( i \) to another one \( \psi \), corresponding to a more complete partial solution. At each step \( \sigma \), each ant \( k \) computes a set \( \mathcal{A}_k^\sigma(i) \) of feasible expansions to its current state, and moves to one of these in probability, according to a probability distribution specified as follows.

For ant \( k \), the probability \( p^k_{i\psi} \) of moving from state \( i \) to state \( \psi \) depends on the combination of two values:

1. the attractiveness \( \eta \) of the move, as computed by some heuristic indicating the \textit{a priori} desirability of that move;
2. the trail level \( \tau \) of the move, indicating how proficient it has been in the past to make that particular move: it represents therefore an \textit{a posteriori} indication of the desirability of that move.

Trails are updated at each iteration, increasing the level of those that facilitate moves that were part of "good" solutions, while decreasing all others. The specific formula for defining the probability distribution at each move makes use of a set \( \text{tabu}_k \) which indicates a problem-dependent set of infeasible moves for ant \( k \). Different authors use different formulae, according to [31] probabilities are computed as follows: \( p^k_{i\psi} \) is equal to 0 for all moves which are infeasible (i.e., they are in the tabu list), otherwise it is computed by means of the following formula (1), where \( \alpha \) is a user-defined parameter (\( 0 \leq \alpha \leq 1 \)).

\[
p^k_{i\psi} = \alpha \cdot \tau_{i\psi} + (1 - \alpha) \cdot \eta_{k\psi} \sum_{(i\psi) \notin \text{tabu}_k} \alpha \cdot \tau_{i\psi} + (1 - \alpha) \cdot \eta_{k\psi}
\]

Parameter \( \alpha \) defines the relative importance of trail with respect to attractiveness. After each iteration \( t \) of the algorithm, that is when all ants have completed a solution, trails are updated following formula (2).

\[
\tau_{i\psi}(t) = \rho \tau_{i\psi}(t-1) + \Delta \tau_{i\psi}
\]

where \( \rho \) is a user-defined coefficient and \( \Delta \tau_{i\psi} \) represents the sum of the contributions of all ants that used move \( (i\psi) \) to construct their solution. The ants' contributions are proportional to the quality of the achieved solutions , i.e., the better an ant solution, the higher will be the trail contribution added to the moves it used. The general structure of an ACO algorithm is as follows.

**ACO framework**

\[\text{Step 1. (Initialization)}\]
Initialize \( \tau_{i\psi}, \forall (i, \psi) \).

\[\text{Step 2. (Construction)}\]
For each ant \( k \) do
repeat
\[\text{compute } \eta_{k\psi}, \forall (i, \psi).\]
choose the state to move into, with probability given by (1). 
append the chosen move to the $k$-th ant’s set $tabu_k$.

until ant $k$ has completed its solution.
carry the solution to its local optimum.

enddo.

Step 3. (Trail update)
For each ant move $(\nu \psi)$ do
compute $\Delta T_{\nu \psi}$.
update the trail matrix by means of (2).
enddo.

Step 4. (Terminating condition)
If not (end condition) go to step 2.

Figure 1. Pseudo code of ACO general structure

This general framework has been specified in different ways by the authors working on the ACO approach. This variety was well represented in first conference entirely devoted to algorithms inspired by the observation of ants behavior (ANTS’98: From Ant Colonies to Artificial Ants, First International Workshop on ANT Colony Optimization, Brussels, Belgium, October 1998). This event was attended by research groups from several European countries (Germany, Italy, Switzerland, UK, France, Austria, The Netherlands, Slovenia, Spain and Belgium) besides other groups from Japan, Russia, Brazil, Mexico, Israel and USA. Different applications have been presented: from plan merging to routing problems, from driver scheduling to search space sharing, from set covering to nurse scheduling, from graph coloring to dynamic multiple criteria balancing problems. Large part of the relevant literature can be accessed on line from http://iridia.ulb.ac.be/ants98/ants98.html, maintained by M. Dorigo.

These applications add to those already presented, which will be briefly overviewed in the following. Table 1 presents a summary of the main ACO metaheuristics so far published. The first column of the table shows the name given to the metaheuristic, when available, the second columns the authors who followed that approach and the third the problems it has been applied to (where TSP stands for Traveling Salesman Problem, QAP for Quadratic Assignment Problem, JSP for Job Shop Scheduling Problem, VRP for Vehicle Routing Problem, SOP for Sequential Ordering Problem, FAP for Frequency Assignment Problem, GCP for Graph Coloring Problem, SCS for Shortest Common Supersequence).

Table 1. ACO applications (adapted from [15])

<table>
<thead>
<tr>
<th>Application</th>
<th>Authors</th>
<th>Year</th>
<th>Problems</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABC</td>
<td>Bonabeau et al.</td>
<td>1991</td>
<td>network routing</td>
</tr>
<tr>
<td></td>
<td>van der Put,</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Rothkrantz</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ACS</td>
<td>Dorigo, Gamba</td>
<td>1996</td>
<td>TSP, VRP</td>
</tr>
<tr>
<td></td>
<td>deldella</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AntNet</td>
<td>Di Caro, Dorigo</td>
<td>1996</td>
<td>network routing</td>
</tr>
<tr>
<td></td>
<td>[6], [7]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ANTS</td>
<td>Maniezzo [31]</td>
<td></td>
<td>QAP, FAP</td>
</tr>
<tr>
<td></td>
<td>, Maniezzo,</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Carbonaro [32]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AS</td>
<td>Colomi, Dorigo,</td>
<td>1997</td>
<td>TSP, QAP, JSP</td>
</tr>
<tr>
<td></td>
<td>Maniezzo [10],</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>[11], [17]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ASrank</td>
<td>Bullheimer,</td>
<td>2002</td>
<td>TSP, VRP</td>
</tr>
<tr>
<td></td>
<td>Hartl, Strauss</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>[3]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>HAS</td>
<td>Gambardella,</td>
<td>1997</td>
<td>QAP, VRP, SOP</td>
</tr>
<tr>
<td></td>
<td>Taillard, Dorigo</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>[20], [19]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MMAS</td>
<td>Stuetze, Hoos</td>
<td>1995</td>
<td>TSP, QAP</td>
</tr>
<tr>
<td></td>
<td>(1998) [42]</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>-</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Costa, Hertz</td>
<td>1996</td>
<td>GCP</td>
</tr>
<tr>
<td></td>
<td>(1997) [12]</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>-</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Michel, Middendorf (1998) [36]</td>
<td></td>
<td>SCS</td>
</tr>
</tbody>
</table>
The variety of the contributions testifies both the flexibility of the approach and the infancy of the field, where no evidence of the superiority of a particular technique over the other has so far emerged. In Section 4 we will concentrate on a particular one, ANTS, to provide more insight into the computational elements of a specific implementation, while in the following of this section we will present more elements about the different ACO contributions.

3.1. ACO approaches to TSP

The first application of ACO used the traveling salesman problem (TSP) as a benchmark problem. This was because TSP is one of the most studied NP-hard problem, and the ant metaphor is easily adapted to it. Several authors built upon this initial contribution.

Stuetzle and Hoos [41] introduced Max-Min Ant System (MMAS), a modification of Ant System applied to the TSP. These authors explicitly introduced in the algorithm two parameters, a maximum and minimum trail levels, whose values are chosen in a problem-dependent way in order to restrict possible trail values to the interval $[\tau_{\text{min}}, \tau_{\text{max}}]$. Moreover, MMAS controls the trail levels (initialized to their maximum value $\tau_{\text{max}}$), only allowing the best ant at each iteration to update trails, thus providing a feedback on its results. Trails that do not receive any or very rare reinforcements will continuously lower their strength and will be selected more and more rarely by the ants, until they reach the $\tau_{\text{min}}$ value. The $\tau_{\text{min}}$ and $\tau_{\text{max}}$ parameters are used to counteract premature stagnation of search, maintaining at the same time some kind of elitist strategy. When applied to TSP, MMAS performs better than AS.

Bullnheimer, Hartl and Strauss [5] proposed yet another modification of AS, called $\text{AS}_{\text{rank}}$, introducing a rank-based version of the probability distribution to limit the danger of over-emphasized trails caused by many ants using sub-optimal solutions. The idea is the following. At each iteration, when all solutions are completed the ants are sorted by solution quality (that is, tour lengths in the case of the TSP) and the contribution of an ant to the trail level update is weighted according to the rank of the ant, considering only the $\omega$ best ants.

In Table 2, taken from [5], we compare against each other a simulated annealing (SA), a simulated annealing with the nearest neighbour heuristic (SA$_N$), a genetic algorithm (GA), an Ant System (AS), an Ant System with elitist strategy (AS$_{\text{elite}}$) and an ant system with elitist strategy and ranking (AS$_{\text{rank}}$). The table reports the percentage deviation of the average results obtained on five different TSP instances (considering also real-life problems from an industrial application), the percentage deviation of the best results and the percentage deviation of the worst results.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>SA</td>
<td>2,784</td>
<td>0,550</td>
<td>7,416</td>
</tr>
<tr>
<td>SA$_N$</td>
<td>2,036</td>
<td>0,148</td>
<td>4,902</td>
</tr>
<tr>
<td>GA</td>
<td>2,112</td>
<td>0,614</td>
<td>4,252</td>
</tr>
<tr>
<td>AS</td>
<td>1,886</td>
<td>0,652</td>
<td>2,802</td>
</tr>
<tr>
<td>AS$_{\text{elite}}$</td>
<td>1,112</td>
<td>0,200</td>
<td>2,866</td>
</tr>
<tr>
<td>AS$_{\text{rank}}$</td>
<td>1,012</td>
<td>0,100</td>
<td>2,212</td>
</tr>
</tbody>
</table>
In general, the Ant System can compete with the other two meta-heuristics; for large problems $A_{S_{rank}}$ outperforms the other methods regarding average and especially worst case behaviour.

Gambardella and Dorigo [21] merged AS and Q-learning, a well known reinforcement learning algorithm from Artificial Intelligence, into an algorithm called Ant-Q. The idea was to update trails with values which predicted the quality of solution using the edges to which the trails were associated. Even though showing a good performance, Ant-Q was abandoned for the equally good but simpler Ant Colony System (ACS) algorithm [16], that uses a constant value instead of the mentioned prediction term. In this algorithm the trail values are added offline, at the end of each iteration of the algorithm, only to the arcs belonging to the best tour from the beginning of the search process, while ants perform online step-by-step trail updates to favor the emergence of solutions other than the best so far. Each ant uses a pseudo-random proportional rule to choose the next node to move to. This is a decision rule based on a $q_0 \in [0, 1]$ parameter that permits to modulate the exploration behaviour, concentrating the system activity either on the best solutions or on the entire search space. ACS also uses a data structure associated to vertices called candidate list which provides additional local heuristic information. The candidate list associated with a vertex contains only the $d$ vertices nearest to the one in subject, and the ants choose the next move scanning the candidate list instead of examining all the unvisited neighbouring vertices.

3.2. ACO approaches to QAP

The quadratic assignment problem (QAP) is the problem of assigning $n$ facilities to $n$ locations so that a quadratic assignment cost is minimized. QAP can be considered one of the hardest CO problems, and can be solved to optimality only for comparatively small instances. For this reason, QAP was chosen as a second benchmark for AS, resulting in code AS-QAP [34]. AS-QAP was of limited effectiveness, but was the first evidence of the robustness of AS.

Recent extensions make use of a well-tuned local optimizer, obtaining very good results [33]. In fact, while the process of an individual ant will almost always converge very quickly to a possibly mediocre solution, the interaction of many feedback processes can instead lead to convergence towards a region of the space containing good solutions, so that a very good solution can be found (without however being stuck on it). In other words, the ant population does not converge on a single solution, but on a set of (good) solutions; the ants continue their search to further improve the best solution found. The results obtained showed the Ant System’s competitive performance on several test problems. Further developments lead to the design of ANTS, which will be detailed in section 4.

Also several other systems previously introduced were adapted to the QAP. For example, two efficient techniques are the MMAS-QAP algorithm [42] and HAS-QAP [20]. The latter is an algorithm which interleaves an ant colony algorithm with a simple local search. Comparisons with some of the best heuristics for the QAP have shown that HAS-QAP is among the best as far as real world, irregular, and structured problems are concerned. On the other hand, on random, regular and unstructured problems the performance resulted to be less competitive.
3.3. ACO approaches to VRP

Vehicle routing problems (VRPs) are a CO problems in which a set of vehicles stationed at a depot has to serve a set of customers before returning to the depot, minimizing the number of vehicles used and the total distance traveled by the vehicles. Capacity constraints are imposed on vehicle trips, plus possibly a number other constraints coming from real-world application, such as time windows, back-hauling, rear loading, vehicle objections, maximum tour length, etc. Also the VRP can be considered as a generalization of the TSP, in fact the VRP reduces to the TSP when only one vehicle is available. Some of the most successful applications of ACO metaheuristics to VRP are the following.

A direct extension of AS based on the AS$_{rank}$ algorithm is AS-VRP, an algorithm designed by Bullheimer, Hartl and Strauss [5] [7]. They used various standard heuristics to improve the quality of VRP solutions and modified the construction of the tabu list considering constraints on the maximum total tour length of a vehicle and its capacity. The results obtained on some problem instances were sufficiently interesting to justify a more detailed study.

Also Gambardella, Taillard and Agazzi [22] faced the VRP, adapting the ACS approach to define MACS-VRPTW, and considering the time window extension to VRP which introduces a time range within which a customer must be serviced. This approach has proved to be competitive with the best known approaches in the literature, as testified by Table 3, in which six different problem types (R1 and R2 with randomly distributed vertices, C1 and C2 with clustered vertices and RC1 and RC2 with random-clustered vertices, from [40]) have been used to compare MACS-VRPTW with five other heuristics: the adaptive memory programming method of Rochat and Taillard (RT) and Taillard et al. (TD) ([39], [44]), the method of Chiang and Russell (CR) ([8]), the GA of Potvin and Bengio (PB) ([38]) and the method of Thangiah et al. (TH) ([45]).

<table>
<thead>
<tr>
<th></th>
<th>R1 Vei Dist</th>
<th>C1 Vei Dist</th>
<th>RC1 Vei Dist</th>
<th>R2 Vei Dist</th>
<th>C2 Vei Dist</th>
<th>RC2 Vei Dist</th>
</tr>
</thead>
<tbody>
<tr>
<td>MACS</td>
<td>12 1217.73</td>
<td>10 828.38</td>
<td>11.63 1382.42</td>
<td>2.73 967.75</td>
<td>3.00 589.86</td>
<td>3.25 1129.19</td>
</tr>
<tr>
<td>RT</td>
<td>12.25 1208.50</td>
<td>10 828.38</td>
<td>11.88 1377.39</td>
<td>2.91 961.72</td>
<td>3.00 589.86</td>
<td>3.38 1119.59</td>
</tr>
<tr>
<td>TB</td>
<td>12.17 1209.35</td>
<td>10 828.38</td>
<td>11.50 1389.22</td>
<td>2.82 980.27</td>
<td>3.00 589.86</td>
<td>3.38 1117.44</td>
</tr>
<tr>
<td>CR</td>
<td>12.42 1289.95</td>
<td>10 885.86</td>
<td>12.38 1455.82</td>
<td>2.91 1135.14</td>
<td>3.00 698.88</td>
<td>3.38 1361.14</td>
</tr>
<tr>
<td>PB</td>
<td>12.58 1296.80</td>
<td>10 838.01</td>
<td>12.13 1446.20</td>
<td>3.00 1117.70</td>
<td>3.00 589.93</td>
<td>3.38 1360.57</td>
</tr>
<tr>
<td>TH</td>
<td>12.33 1238.00</td>
<td>10 832.00</td>
<td>12.00 1284.00</td>
<td>3.00 1005.00</td>
<td>3.00 650.00</td>
<td>3.38 1229.00</td>
</tr>
</tbody>
</table>

4. The ANTS algorithm

ANTS is an extension of the Ant System proposed in [10], which specifies some underdefined elements of the general algorithm, such as the attractiveness function to use or the initialization of the trail distribution. This turns out to be variations of the general ACO framework that make the resulting algorithm similar in structure to tree search algorithms. In fact, the essential trait which distinguishes ANTS from a tree search algorithm is the lack of a complete backtracking mechanism,
which is substituted by a probabilistic (Non-deterministic) choice of the state to move to and by an incomplete (Approximate) exploration of the search tree; this is the rationale behind the name ANTS, which is an acronym of Approximated Non-deterministic Tree Search. In the following we will outline two distinctive elements of the ANTS algorithm within the ACO framework, namely the attractiveness function and the trail updating mechanism.

4.1. Attractiveness

The attractiveness of a move can be effectively estimated by means of lower bounds (upper bounds in case of maximization problems) to the cost of the completion of a partial solution. In fact, if a state $i$ corresponds to a partial problem solution it is possible to compute a lower bound to the cost of a complete solution containing $i$. Therefore, for each feasible move $(i, \psi)$, it is possible to compute the lower bound to the cost of a complete solution containing $\psi$: the lower the bound the better the move. Since large part of research in CO is devoted to the identification of tight lower bounds for the different problems of interest, good lower bounds are usually available. Their use has several advantages, some of which are listed in the following ([31]).

\begin{itemize}
  \item A tight bound gives strong indications on the opportunity of a move.
  \item When the bound value becomes greater than the current upper bound, it is obvious that the considered move leads to a partial solution which cannot possibly be completed in a solution better than the current best one. The move can therefore be discarded from further analysis.
  \item If the bound is derived from linear programming (LP) and dual cost information is therefore available, it is possible to compute reduced costs for the problem decision variables, which in turn - when compared to an upper bound to the optimal problem solution cost - permit to a priori eliminate some variables. This results in a reduction to the number of possible moves, therefore to a reduction of the search space.
  \item A further advantage of LP lower bound is that the primal values of the decision variables, as appearing in the bound solution, can be used as an estimate of the probability with which each variable will appear in good solutions. This provides an effective way for initializing the trail values, thus eliminating the need for the user-defined parameters.
\end{itemize}

The use LP bounds is a very effective and straightforward general policy, whenever tight such bounds have been identified for the problem to solve.

4.2. Trail update

A good trail updating mechanism avoids stagnation, the undesirable situation in which all ants repeatedly construct the same solutions, making impossible any further exploration in the search process. This derives from an excessive trail level on the moves of one solution, and can be observed in advanced phases of the search process, if parameters are not well tuned to the problem.

The procedure evaluates each solution against the last $k$ ones globally constructed by ANTS. As soon as $k$ solutions are available, we compute their moving average $\overline{z}$; each new solution $z_{\text{curr}}$ is compared with $\overline{z}$ (and then used to compute the new moving average value). If $z_{\text{curr}}$ is lower than $\overline{z}$, the trail level of the last
solution’s moves is increased, otherwise it is decreased. Formula (3) specifies how this is implemented:

\[ \Delta \tau_{k} = \tau_{0} \cdot \left(1 - \frac{z_{\text{curr}} - LB}{z - LB}\right) \]  

(3)

where \( z \) is the average of the last \( k \) solutions and \( LB \) is a lower bound to the optimal problem solution cost. The use of a dynamic scaling procedure permits to discriminate small achievement in the latest stage of search, while avoiding to focusize search only around good achievement in the earliest stages.

Based on the described elements, the ANTS metaheuristic is the following.

**ANTS algorithm**

**Step 1.** (Initialization)

Compute a (linear) lower bound \( LB \) to the problem to solve.

Initialization \( \tau_{k}, \forall (i, \psi) \) with the primal variable values.

**Step 2.** (Construction)

For each ant \( k \) do

repeat

compute \( \eta_{\psi}, \forall (i, \psi) \), as a lower bound to the cost of a complete solution containing \( \psi \).

choose the state to move into, with probability given by (1).

append the chosen move to the \( k \)-th ant’s set \( tabu_{k} \).

until \( k \) ant has completed its solution.

carry the solution to its local optimum.

**Step 3.** (Trail update)

For each ant move \( \psi \) do

compute \( \Delta \tau_{\psi} \).

update the trail matrix by means of (2) and (3).

**Step 4.** (Terminating condition)

If not(end-test) go to step 2.

**Figure 2. Pseudo code for the ANTS algorithm**

It can be noticed that the general structure of the ANTS algorithm is closely akin to that of a standard tree-search procedure. At each stage we have in fact a partial solution which is expanded by branching on all possible offspring; a bound is then computed for each offspring, possibly fathoming dominated ones, and the current partial solution is selected among that associated to the surviving offspring on the basis of lower bound considerations. By simply adding backtracking and eliminating the Montecarlo choice of the node to move to, we revert to a standard branch and bound procedure. An ANTS code can therefore be easily turned into an exact procedure.
5. Computational results

The above elements have been implemented and tested on two well known CO problems: the Quadratic Assignment and the Frequency Assignment problems (QAP and FAP, respectively). It is important to notice that the reason behind the choice of these problems is in the attempt to empirically evaluate the robustness of the ANTS approach. While in fact preliminary non-optimized codes testified the validity of the issues reported in Section 4, that is the effectiveness of the ANTS procedure when tight bounds are available, both QAP and FAP were chosen because of the ineffectiveness of the bounds so far presented in the literature. A good performance on these problems a fortiori suggests the efficiency of the approach in the general case.

5.1. QAP

The Quadratic Assignment Problem (QAP) is one of the best known and most difficult CO problems, as it is testified by the comparatively small gap existing between the dimension of the problems that can be solved to optimality by means of complete enumeration and the dimension of the problems that can be solved by means of the most advanced exact methods proposed in the literature.

The ANTS algorithm makes use of a lower bound derived form the well-known Gilmore and Lawler bound. Details are provided in [31], where both the ANTS and the derived exact procedure are described. Computational results for the heuristic part were given on all QAPLIB problem instances of dimension up to n = 40 and presented for ANTS and for two state-of-the-art heuristic procedures: Li et al.’s GRASP [29] and Taillard’s robust tabu search (TS) [43]. Table 4 shows the obtained results.

Table 4. ANTS on QAP

<table>
<thead>
<tr>
<th></th>
<th>TS</th>
<th>GRASP</th>
<th>ANTS</th>
<th></th>
<th>TS</th>
<th>GRASP</th>
<th>ANTS</th>
</tr>
</thead>
<tbody>
<tr>
<td>CHR20A</td>
<td>0.06</td>
<td>1.48</td>
<td>0.00</td>
<td>NUG30</td>
<td>0.00</td>
<td>0.39</td>
<td>0.00</td>
</tr>
<tr>
<td>CHR20B</td>
<td>2.06</td>
<td>4.65</td>
<td>0.00</td>
<td>TAB20A</td>
<td>0.16</td>
<td>1.53</td>
<td>0.13</td>
</tr>
<tr>
<td>ROU20</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>TAB20B</td>
<td>0.00</td>
<td>0.11</td>
<td>0.00</td>
</tr>
<tr>
<td>TAI20A</td>
<td>0.00</td>
<td>0.19</td>
<td>0.00</td>
<td>THO30</td>
<td>0.00</td>
<td>0.15</td>
<td>0.00</td>
</tr>
<tr>
<td>CHR22A</td>
<td>0.00</td>
<td>1.15</td>
<td>0.00</td>
<td>ESC32A</td>
<td>0.00</td>
<td>2.77</td>
<td>0.00</td>
</tr>
<tr>
<td>CHR22B</td>
<td>0.71</td>
<td>2.03</td>
<td>0.00</td>
<td>ESC32B</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>CHR25A</td>
<td>4.48</td>
<td>4.64</td>
<td>0.76</td>
<td>ESC32H</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>TAI25A</td>
<td>0.12</td>
<td>0.79</td>
<td>0.00</td>
<td>MC33</td>
<td>0.12</td>
<td>0.07</td>
<td>0.00</td>
</tr>
<tr>
<td>TAI26B</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>TAB35A</td>
<td>0.65</td>
<td>1.80</td>
<td>0.32</td>
</tr>
<tr>
<td>BUR26A</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>TAB35A</td>
<td>0.00</td>
<td>0.23</td>
<td>0.00</td>
</tr>
<tr>
<td>BUR26B</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>STE36A</td>
<td>0.00</td>
<td>1.76</td>
<td>0.00</td>
</tr>
<tr>
<td>BUR26E</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>STE36B</td>
<td>0.00</td>
<td>1.44</td>
<td>0.00</td>
</tr>
<tr>
<td>BUR26F</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>STE36C</td>
<td>0.00</td>
<td>0.67</td>
<td>0.00</td>
</tr>
<tr>
<td>KRA30A</td>
<td>0.00</td>
<td>0.34</td>
<td>0.00</td>
<td>LIPA40A</td>
<td>0.00</td>
<td>1.11</td>
<td>0.00</td>
</tr>
<tr>
<td>KRA30B</td>
<td>0.00</td>
<td>0.15</td>
<td>0.00</td>
<td>TAI40A</td>
<td>0.93</td>
<td>2.06</td>
<td>0.47</td>
</tr>
<tr>
<td>LIPA30A</td>
<td>0.00</td>
<td>0.19</td>
<td>0.00</td>
<td>TAI40B</td>
<td>0.00</td>
<td>0.03</td>
<td>0.00</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>THO40</td>
<td>0.06</td>
<td>0.91</td>
<td>0.03</td>
</tr>
</tbody>
</table>
The good performance that all algorithms have lead to result worse than the best known ones only on a comparatively small number of problems. The best performing among the algorithms is ANTS, both considering the best and the average quality of the solutions proposed. It is interesting to see how, even in the presence of a bad bound at the root node, the nondeterministic strategy followed by ANTS permits to quickly identify good solutions.

5.2. FAP

The Frequency Assignment Problem is the problem that arises when a region is covered, for wireless communications, by cells centered on base stations and transmitters scattered around the region want to connect with the antennas of the base stations. Each connection, or link, between a transmitter and a base station can be made on a frequency supported by the antenna. However, the frequency concurrently operated by overlapping cells must be separated in order to minimize the interference on the communications taking place in the cells.

The current state of development of the research on FAP does not provide efficient lower bounds. We developed one ([35]), which is not very tight but is efficient to compute, and included it in the ANTS algorithm. Computational results were obtained on three well-known problem datasets from the literature, obtaining the results presented in Table 5. More details can be found in [32], in these same proceedings. In this subsection we report the computational results obtained on a number of different test problems drawn from literature: the CELAR, GRAPH and PHILADELPHIA problems. The CELAR dataset consists of 11 problems proposed within the framework of EUCLID (European Cooperation for the Long term in Defence) CALMA (Combinatorial ALgorithms for Military Applications) project [46]. The GRAPH test problems [47] are 14 problems patterned after the CELAR problems which exhibit the same structure. The Philadelphia (Phil) problems, originally presented by Anderson [3], are among the most studied FAP instances. The problems are based on the area around Philadelphia and consist of cells located in a hexagonal grid. All results have been obtained implementing the algorithms in C and running the codes on a PentiumII 233 MHz machine equipped with 64 Mb of RAM for 1200 CPU seconds.

Table 5. ANTS on FAP

<table>
<thead>
<tr>
<th>Cesar</th>
<th>DS</th>
<th>TS</th>
<th>ANTS</th>
<th>SA1</th>
<th>SA2</th>
<th>Graph</th>
<th>DS</th>
<th>TS</th>
<th>ANTS</th>
<th>SA1</th>
<th>SA2</th>
<th>Phil</th>
<th>DS</th>
<th>TS</th>
<th>ANTS</th>
<th>SA1</th>
<th>SA2</th>
</tr>
</thead>
<tbody>
<tr>
<td>625</td>
<td>238</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>160</td>
<td>15</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>7</td>
<td>4</td>
<td>0</td>
<td>51</td>
<td>263</td>
<td></td>
</tr>
<tr>
<td>428</td>
<td>122</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>299</td>
<td>15</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>17</td>
<td>251</td>
<td></td>
</tr>
<tr>
<td>738</td>
<td>196</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>264</td>
<td>35</td>
<td>14</td>
<td>14</td>
<td>96</td>
<td>8</td>
<td>3</td>
<td>8</td>
<td>0</td>
<td>31</td>
<td>288</td>
<td></td>
</tr>
<tr>
<td>1474</td>
<td>012</td>
<td>8</td>
<td>0</td>
<td>1</td>
<td>519</td>
<td>33</td>
<td>42</td>
<td>64</td>
<td>213</td>
<td>6</td>
<td>6</td>
<td>3</td>
<td>6</td>
<td>0</td>
<td>36</td>
<td>353</td>
<td></td>
</tr>
<tr>
<td>1823</td>
<td>688</td>
<td>32</td>
<td>11</td>
<td>54</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>33</td>
<td>3</td>
<td>30</td>
<td>31</td>
<td>252</td>
<td></td>
</tr>
<tr>
<td>2138667</td>
<td>149607</td>
<td>5319</td>
<td>6994</td>
<td>30160</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>18</td>
<td>17</td>
<td>36</td>
<td>30</td>
<td>288</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>&gt; 10^8</td>
<td>10^8803903</td>
<td>1100022964698997</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>17</td>
<td>14</td>
<td>31</td>
<td>22</td>
<td>249</td>
<td></td>
</tr>
<tr>
<td>2418</td>
<td>1364</td>
<td>709</td>
<td>306</td>
<td>457</td>
<td>566</td>
<td>18</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>17</td>
<td>14</td>
<td>31</td>
<td>22</td>
<td>249</td>
<td></td>
</tr>
<tr>
<td>79406</td>
<td>4988</td>
<td>16732</td>
<td>30024</td>
<td>23634</td>
<td>665</td>
<td>14</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>403</td>
<td>32</td>
<td>256</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>107310</td>
<td>58554</td>
<td>31516</td>
<td>31518</td>
<td>33557</td>
<td>856</td>
<td>37</td>
<td>127</td>
<td>91</td>
<td>81</td>
<td>1</td>
<td>1</td>
<td>51</td>
<td>18</td>
<td>254</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1364</td>
<td>848</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
The computational results show that the ANTS heuristic is competitive with the best approaches so far presented. In particular, Table 3 shows that, apart from ANTS, different approaches are the best performing ones. The CELAR problems are best solved by the ANTS and the SA1 algorithms, the GRAPH have the same structure as the CELAR problems, and in fact the results obtained are similar among the two groups, while the PHIL problems are more suited for the DSATUR and TS approaches. On all test problems, under the imposed computational constraints (1200 secs CPU time), it found a good solution and exhibited more stable results among those produced by the tested algorithms.

6. Conclusions

There is a natural evolution in the life of algorithms that attract the attention of the research community. At the beginning the need to face previously unsolvable problems, or the attractiveness of the motivating idea justifies interest in the approach, despite possible second-rate results. If the core idea is worthwhile, enduring research improves the results obtained up to the limit of the state-of-the-art, and possibly beyond. This is what we witnessed for local search procedures, for simulated annealing, for genetic algorithms, etc. It seems that ACO algorithms are following the same track. After the early results on the robustness of the idea, but with little effectiveness to show, now we have moved into a maturity where ant-based algorithms should not be dismissed when high quality solutions are needed for difficult CO problems.

There is still no way to tell which among the different ACO approaches currently studied is the one that will yield better results. The area is still (already?) actively investigated, new ideas are poured into the general framework and effectiveness and robustness are under improvement. We have no means to forecast how ACO will compare with the most established metaheuristic approaches so far. What we can say is that ACO is not sterile, that ants have not dried out on the first maybe intriguing promises.

References


Science dell'Informazione, University of Bologna, Via Saccie 3, 47023 Cesena, Italy. E-mail address: {maniezzo, carbonar}@csr.unibo.it