Ant System: Optimization by a Colony of Cooperating Agents

Marco Dorigo, Member, IEEE, Vittorio Maniezzo, and Alberto Colomni

Abstract—An analogy with the way ant colonies function has suggested the definition of a new computational paradigm, which we call Ant System. We propose it as a viable new approach to stochastic combinatorial optimization. The main characteristics of this model are positive feedback, distributed computation, and the use of a constructive greedy heuristic. Positive feedback accounts for rapid discovery of good solutions, distributed computation avoids premature convergence, and the greedy heuristic helps find acceptable solutions in the early stages of the search process. We apply the proposed methodology to the classical Traveling Salesman Problem (TSP), and report simulation results. We also discuss parameter selection and the early setups of the model, and compare it with tabu search and simulated annealing using TSP. To demonstrate the robustness of the approach, we show how the Ant System (AS) can be applied to other optimization problems like the asymmetric traveling salesman, the quadratic assignment and the job-shop scheduling. Finally we discuss the salient characteristics—global data structure revision, distributed communication and probabilistic transitions of the AS.

I. INTRODUCTION

In this paper we define a new general-purpose heuristic algorithm which can be used to solve different combinatorial optimization problems. The new heuristic has the following desirable characteristics:

- It is versatile, in that it can be applied to similar versions of the same problem; for example, there is a straightforward extension from the traveling salesman problem (TSP) to the asymmetric traveling salesman problem (ATSP).
- It is robust. It can be applied with only minimal changes to other combinatorial optimization problems such as the quadratic assignment problem (QAP) and the job-shop scheduling problem (JSP).
- It is a population based approach. This is interesting because it allows the exploitation of positive feedback as a search mechanism, as explained later in the paper. It also makes the system amenable to parallel implementations (though this is not considered in this paper).

These desirable properties are counterbalanced by the fact that, for some applications, the Ant System can be outperformed by more specialized algorithms. This is a problem shared by other popular approaches like simulated annealing (SA), and tabu search (TS), with which we compare the Ant System. Nevertheless, we believe that, as is the case with SA and TS, our approach is meaningful in view of applications to problems which, although very similar to well known and studied basic problems, present peculiarities which make the application of the standard best-performing algorithm impossible. This is the case, for example, with the ATSP.

In the approach discussed in this paper we distribute the search activities over so-called "ants," that is, agents with very simple basic capabilities which, to some extent, mimic the behavior of real ants. In fact, research on the behavior of real ants has greatly inspired our work (see [10], [11], [21]). One of the problems studied by ethologists was to understand how almost blind animals like ants could manage to establish shortest route paths from their colony to feeding sources and back. It was found that the medium used to communicate information among individuals regarding paths, and used to decide where to go, consists of pheromone trails. A moving ant lays some pheromone (in varying quantities) on the ground, thus marking the path by a trail of this substance. While an isolated ant moves essentially at random, an ant encountering a previously laid trail can detect it and decide with high probability to follow it, thus reinforcing the trail with its own pheromone. The collective behavior that emerges is a form of autocatalytic behavior\(^1\) where the more the ants following a trail, the more attractive that trail becomes for being followed. The process is thus characterized by a positive feedback loop, where the probability with which an ant chooses a path increases with the number of ants that previously chose the same path.

Consider for example the experimental setting shown in Fig. 1. There is a path along which ants are walking (for example from food source \(A\) to the nest \(E\), and vice versa, see Fig. 1(a)). Suddenly an obstacle appears and the path is cut off. So at position \(B\) the ants walking from \(A\) to \(E\) (or at position \(D\) those walking in the opposite direction) have to decide whether to turn right or left (Fig. 1(b)). The choice is influenced by the intensity of the pheromone trails left by preceding ants. A higher level of pheromone on the

---

\(^1\)An autocatalytic [12], i.e. positive feedback, process is a process that reinforces itself, in a way that causes very rapid convergence and, if no limitation mechanism exists, leads to explosion.
Fig. 1. An example with real ants. (a) Ants follow a path between points A and E. (b) An obstacle is interposed; ants can choose to go around it following one of the two different paths with equal probability. (c) On the right path gives an ant a stronger stimulus and thus a higher path no previous pheromone on the two alternative paths). Because will reach D before the first ant following path BHD (Fig. 1(c)). The result is that an ant returning from E to D will find a stronger trail on path DCB, caused by the half of all the ants that by chance decided to approach the obstacle via DCBA and by the already arrived ones coming via BCD: they will therefore prefer (in probability) path DCB to path DHB. As a consequence, the number of ants following path BCD per unit of time will be higher than the number of ants following BHD. This causes the quantity of pheromone on the shorter path to grow faster than on the longer one, and therefore the probability with which any single ant chooses the path to follow is quickly biased toward the shorter one. The final result is that very quickly all ants will choose the shorter path.

The algorithms that we are going to define in the next sections are models derived from the study of real ant colonies. Therefore we call our system Ant System (AS) and the algorithms we introduce ant algorithms. As we are not interested in simulation of ant colonies, but in the use of artificial ant colonies as an optimization tool, our system will have some major differences with a real (natural) one:
- artificial ants will have some memory,
- they will not be completely blind,
- they will live in an environment where time is discrete.

Nevertheless, we believe that the ant colony metaphor can be useful to explain our model. Consider the graph of Fig. 2(a), which is a possible AS interpretation of the situation of Fig. 1(b). To fix the ideas, suppose that the distances between D and H, between B and H, and between B and D—via C—are equal to 1, and let C be positioned half the way between D and B (see Fig. 2(a)). Now let us consider what happens at regular discretized intervals of time: \( t = 0, 1, 2, \ldots \). Suppose that 30 new ants come to B from A, and 30 to D from E at each time unit, that each ant walks at a speed of 1 per time unit, and that while walking an ant lays down at time \( t \) a pheromone trail of intensity 1, which, to make the example simpler, evaporates completely and instantaneously in the middle of the successive time interval \( (t + 1, t + 2) \).

At \( t = 0 \) there is no trail yet, but 30 ants are in B and 30 in D. Their choice about which way to go is completely random. Therefore, on the average 15 ants from each node will go toward H and 15 toward C (Fig. 2(b)).

At \( t = 1 \) the 30 new ants that come to B from A find a trail of intensity 15 on the path that leads to H, laid by the 15 ants that went that way from B, and a trail of intensity 30 on the path to C, obtained as the sum of the trail laid by the 15 ants that went that way from B and by the 15 ants that reached B coming from D via C (Fig. 2(c)). The probability of choosing a path is therefore biased, so that the expected number of ants going toward C will be the double of those going toward H: 20 versus 10 respectively. The same is true for the new 30 ants in D which came from E.

This process continues until all of the ants will eventually choose the shortest path.

The idea is that if at a given point an ant has to choose among different paths, those which were heavily chosen by preceding ants (that is, those with a high trail level) are chosen with higher probability. Furthermore, high trail levels are synonymous with short paths.

The paper is organized as follows. Section II contains the description of the AS as it is currently implemented and the definition of the application problem: as the algorithm structure partially reflects the problem structure, we introduce them together. Section III describes three slightly different ways to apply the proposed algorithm. Sections IV and V report on experiments. In Section VI we compare the AS with other heuristics, and in Section VII we substantiate the versatility and robustness of the AS by showing how it can be applied to other optimization problems. In Section VIII we informally discuss why and how the AS paradigm functions. Conclusions are in Section IX.

II. THE ANT SYSTEM

In this section we introduce the AS. We decided to use the well-known traveling salesman problem [26] as benchmark, in order to make the comparison with other heuristic approaches
easier [20]. Although the model definition is influenced by the problem structure, we will show in Section VII that the same approach can be used to solve other optimization problems.

Given a set of \( n \) towns, the TSP can be stated as the problem of finding a minimal length closed tour that visits each town once. We call \( d_{ij} \) the length of the path between towns \( i \) and \( j \); in the case of Euclidean TSP, \( d_{ij} \) is the Euclidean distance between \( i \) and \( j \) (i.e., \( d_{ij} = [(x_i - x_j)^2 + (y_i - y_j)^2]^{1/2} \)). An instance of the TSP is given by a graph \((N, E)\), where \( N \) is the set of towns and \( E \) is the set of edges between towns (a fully connected graph in the Euclidean TSP).

Let \( b_i(t) \) (\( i = 1, \ldots, n \)) be the number of ants in town \( i \) at time \( t \) and let \( m = \sum_{i=1}^{n} b_i(t) \) be the total number of ants. Each ant is a simple agent with the following characteristics:

- It chooses the town to go to with a probability that is a function of the town distance and of the amount of trail present on the connecting edge;
- It forces the ant to make legal tours, transitions to already visited towns are disallowed until a tour is completed (this is controlled by a tabu list);
- When it completes a tour, it lays a substance called trail on each edge \((i, j)\) visited.

Let \( \tau_{ij}(t) \) be the intensity of trail on edge \((i, j)\) at time \( t \). Each ant at time \( t \) chooses the next town, where it will be at time \( t + 1 \). Therefore, if we call an iteration of the AS algorithm the \( m \) moves carried out by the \( m \) ants in the interval \((t, t + 1)\), then every \( n \) iterations of the algorithm (which we call a cycle) each ant has completed a tour. At this point the trail intensity is updated according to the following formula

\[
\tau_{ij}(t + n) = \rho \cdot \tau_{ij}(t) + \Delta \tau_{ij}
\]

where \( \rho \) is a coefficient such that \((1 - \rho)\) represents the evaporation of trail between time \( t \) and \( t + n \),

\[
\Delta \tau_{ij} = \frac{\sum_{k=1}^{m} \Delta \tau_{ij}^k}{n}
\]

where \( \Delta \tau_{ij}^k \) is the quantity per unit of length of trail substance (pheromone in real ants) laid on edge \((i, j)\) by the \( k \)th ant between time \( t \) and \( t + n \); it is given by

\[
\Delta \tau_{ij}^k = \begin{cases} 
\frac{Q}{L_k} & \text{if } k \text{th ant uses edge } (i, j) \text{ in its tour (between time } t \text{ and } t + n) \\
0 & \text{otherwise}
\end{cases}
\]

where \( Q \) is a constant and \( L_k \) is the tour length of the \( k \)th ant.

The coefficient \( \rho \) must be set to a value < 1 to avoid unlimited accumulation of trail (see note 1). In our experiments, we set the intensity of trail at time \( 0, \tau_{ij}(0) \), to a small positive constant \( c \).

In order to satisfy the constraint that an ant visits all the \( n \) different towns, we associate with each ant a data structure called the tabu list, that saves the towns already visited up to time \( t \) and forbids the ant to visit them again before \( n \) iterations (a tour) have been completed. When a tour is completed, the tabu list is then emptied and the ant is free again to choose. We define \( \text{tabu}_k \) the dynamically growing vector which contains the tabu list of the \( k \)th ant, \( \text{tabu}_k \) the set obtained from the elements of \( \text{tabu}_k \) and \( \text{tabu}_k(s) \) the \( s \)th element of the list (i.e., the \( s \)th town visited by the \( k \)th ant in the current tour).

We call \( \text{visibility} \) \( \eta_{ij} \) the quantity \( 1/d_{ij} \). This quantity is not modified during the run of the AS, as opposed to the trail which instead changes according to the previous formula (1).

We define the transition probability from town \( i \) to town \( j \) for the \( k \)th ant as

\[
p_{ij}^k(t) = \begin{cases} 
\frac{[\tau_{ij}(t)]^{\alpha} \cdot [\tau_{ij}(t)]^{\beta}}{\sum_{k\in\text{allowed}_k} [\tau_{ij}(t)]^{\alpha} \cdot [\tau_{ij}(t)]^{\beta}} & \text{if } j \in \text{allowed}_k \\
0 & \text{otherwise}
\end{cases}
\]

where \( \text{allowed}_k = \{N - \text{tabu}_k\} \) and where \( \alpha \) and \( \beta \) are parameters that control the relative importance of trail versus visibility. Therefore the transition probability is a trade-off between visibility (which says that close towns should be chosen with high probability, thus implementing a greedy constructive heuristic) and trail intensity at time \( t \) (that says that if on edge \((i, j)\) there has been a lot of traffic then it is highly desirable, thus implementing the autocatalytic process).

III. THE ALGORITHMS

Given the definitions of the preceding section, the so-called ant-cycle algorithm is simply stated as follows. At time zero an initialization phase takes place during which ants are positioned on different towns and initial values \( \tau_{ij}(0) \) for trail intensity are set on edges. The first element of each ant’s tabu list is set to be equal to its starting town. Thereafter every ant moves from town \( i \) to town \( j \) choosing the town to move to with a probability that is a function (with parameters \( \alpha \) and \( \beta \), see formula (4)) of two desirability measures. The first, the trail \( \tau_{ij}(t) \), gives information about how many ants in the past have chosen that same edge \((i, j)\); the second, the visibility \( \eta_{ij} \), says that the closer a town the more desirable it is. Obviously, setting \( \alpha = 0 \), the trail level is no longer considered, and a stochastic greedy algorithm with multiple starting points is obtained.

After \( n \) iterations all ants have completed a tour, and their tabu lists will be full; at this point for each ant \( k \) the value of \( L_k \) is computed and the values \( \Delta \tau_{ij}^k \) are updated according to formula (3). Also, the shortest path found by the ants (i.e., \( \min_k L_k, k = 1, \ldots, m \)) is saved and all the tabu lists are emptied. This process is iterated until the tour counter reaches the maximum (user-defined) number of cycles \( \text{NCMAX} \), or all ants make the same tour. We call this last case stagnation behavior because it denotes a situation in which the algorithm stops searching for alternative solutions. We investigate this situation in Section IV.
Formally the ant-cycle algorithm is:

1. Initialize:
   - Set \( t := 0 \) \{ \( t \) is the time counter \}
   - Set \( NC := 0 \) \{ \( NC \) is the cycles counter \}
   - For every edge \((i, j)\) set an initial value \( \tau_{ij}(t) = c \) for trail intensity and \( \Delta \tau_{ij} = 0 \)
   - Place the \( m \) ants on the \( n \) nodes

2. Set \( s := 1 \) \{ \( s \) is the tabu list index \}
   - For \( k := 1 \) to \( m \) do
     - Place the starting town of the \( k \)th ant in \( \text{tabu}_k(s) \)

3. Repeat until tabu list is full
   \{ this step will be repeated \( (n-1) \) times \}
   - Set \( s := s + 1 \)
   - For \( k := 1 \) to \( m \) do
     - Choose the town \( j \) to move to, with probability \( p^k_{ij}(t) \) given by Eq. (4) \{ at time \( t \) the \( k \)th ant is on town \( i = \text{tabu}_k(s-1) \) \}
     - Move the \( k \)th ant to the town \( j \)
     - Insert town \( j \) in \( \text{tabu}_k(s) \)

4. For \( k := 1 \) to \( m \) do
   - Move the \( k \)th ant from \( \text{tabu}_k(n) \) to \( \text{tabu}_k(1) \)
   - Compute the length \( L_k \) of the tour described by the \( k \)th ant
   - Update the shortest tour found
   - For every edge \((i, j)\) do
     - For \( k := 1 \) to \( m \) do
       - \( \Delta \tau^k_{ij} = \begin{cases} \frac{Q}{\tau^k_{ij}} & \text{if } (i, j) \in \text{tour described by } \text{tabu}_k \\ 0 & \text{otherwise} \end{cases} \)
     - \( \Delta \tau_{ij} := \Delta \tau_{ij} + \Delta \tau^k_{ij} \)

5. For every edge \((i, j)\) compute \( \tau_{ij}(t + n) \)
   according to equation \( \tau_{ij}(t + n) = \rho \cdot \tau_{ij}(t) + \Delta \tau_{ij} \)
   - Set \( t := t + n \)
   - Set \( NC := NC + 1 \)
   - For every edge \((i, j)\) set \( \Delta \tau_{ij} := 0 \)

6. If \( (NC < NC_{MAX}) \) and (not stagnation behavior) then
   - Empty all tabu lists
   - Goto step 2
else
   - Print shortest tour
   - Stop

The complexity of the ant-cycle algorithm is \( O(NC \cdot n^2 \cdot m) \) if we stop the algorithm after \( NC \) cycles. In fact step 1 is \( O(n^2 + m) \), step 2 is \( O(m) \), step 3 is \( O(n^2 \cdot m) \), step 4 is \( O(n^2 \cdot m) \), step 5 is \( O(n^2) \), step 6 is \( O(n \cdot m) \). Since we have experimentally found a linear relation between the number of towns and the best number of ants (see Section V-A), the complexity of the algorithm is \( O(NC \cdot n^3) \).

We also experimented with two other algorithms of the AS, which we called ant-density and ant-quantity algorithms [6], [12]. They differ in the way the trail is updated. In these two models each ant lays its trail at each step, without waiting for the end of the tour. In the ant-density model a quantity \( Q \) of trail is left on edge \((i, j)\) every time an ant goes from \( i \) to \( j \); in the ant-quantity model an ant going from \( i \) to \( j \) leaves a quantity \( Q \cdot d_{ij} \) of trail on edge \((i, j)\) every time it goes from \( i \) to \( j \). Therefore, in the ant-density model we have

\[
\Delta \tau^k_{ij} = \begin{cases} Q & \text{if the } k \text{th ant goes from } i \text{ to } j \\ 0 & \text{otherwise} \end{cases}
\]

and in the ant-quantity model we have

\[
\Delta \tau^k_{ij} = \begin{cases} Q \cdot d_{ij} & \text{if the } k \text{th ant goes from } i \text{ to } j \\ 0 & \text{otherwise}. \end{cases}
\]

From these definitions it is clear that the increase in trail on edge \((i, j)\) when an ant goes from \( i \) to \( j \) is independent of \( d_{ij} \) in the ant-density model, while it is inversely proportional to \( d_{ij} \) in the ant-quantity model (i.e., shorter edges are made more desirable by ants in the ant-quantity model).

IV. EXPERIMENTAL STUDY 1: PARAMETER SETTING AND BASIC PROPERTIES

We implemented the three algorithms (ant-cycle, ant-density and ant-quantity) of the AS and investigated their relative strengths and weaknesses by experimentation. Since we have not yet developed a mathematical analysis of the models, which would yield the optimal parameter setting in each situation, we ran simulations to collect statistical data for this purpose.

The parameters considered here are those that affect directly or indirectly the computation of the probability in formula (4):

- \( \alpha \): the relative importance of the trail, \( \alpha \geq 0 \);
- \( \beta \): the relative importance of the visibility, \( \beta \geq 0 \);
- \( \rho \): trail persistence, \( 0 \leq \rho < 1 \) (1 - \( \rho \) can be interpreted as trail evaporation);
- \( Q \): a constant related to the quantity of trail laid by ants (see formulas (3), (5), and (6)).

The number \( m \) of ants has always been set equal to the number \( n \) of cities (see Section V-A for the explanation). We tested several values for each parameter while all the others were held constant (over ten simulations for each setting in order to achieve some statistical information about the average evolution). The default value of the parameters was \( \alpha = 1 \), \( \beta = 1 \), \( \rho = 0.5 \), \( Q = 100 \). In each experiment only one of the values was changed, except for \( \alpha \) and \( \beta \), which have been tested over different sets of values, as discussed at the end of this section. The values tested were: \( \alpha \in \{0,0.5,1,2,5\} \), \( \beta \in \{0,1,2,5\} \), \( \rho \in \{0.3,0.5,0.7,0.9,0.999\} \) and \( Q \in \{1,10,100,1000\} \). Preliminary results, obtained on small-scale problems, have been presented in [6], [7], and [12], [13]; all the tests reported in this section are based, where not otherwise stated, on the Oliver30 problem, a 30-cities problem described in [34]. All the tests have been carried out for \( NC_{MAX} = 5000 \) cycles and were averaged over ten trials.

3 In [34] genetic algorithms were applied to solve the Oliver30 problem; they could find a tour of length 424.635. The same result was often obtained by ant-cycle, which also found a tour of length 423.741.
To compare the three models we first experimentally determined the parameters best values for each algorithm, and then we ran each algorithm ten times using the best parameters set. Results are shown in Table I. Parameter \( Q \) is not shown because its influence was found to be negligible.

Both the ant-density and the ant-quantity models have given worse results than those obtained with ant-cycle. The reason is to be found in the kind of feedback information which is used to direct the search process. Ant-cycle uses global information, that is, its ants lay an amount of trail which is proportional to how good the solution produced was. In fact, ants producing shorter paths contribute a higher amount of trail than ants whose tour was poor. On the other side, both ant-quantity and ant-density use local information. Their search is not directed by any measure of the final result achieved. Therefore, it is not surprising that they gave worse performance results (details can be found in [6]).

The optimal value \( \rho = 0.5 \) in ant-cycle can be explained by the fact that the algorithm, after using the greedy heuristic to guide search in the early stages of computation, starts exploiting the global information contained in the values \( \tau_{ij} \) of trail. Ant-cycle needs therefore to have the possibility to forget part of the experience gained in the past in order to better exploit new incoming global information.

Given that we found ant-cycle to be superior to the other two algorithms, we decided to deepen our understanding of ant-cycle alone. Figs. 3–5 present traces of a typical run of the ant-cycle algorithm applied to the Oliver30 problem. In particular, Fig. 3 shows the length of the best found tour at each cycle, and Fig. 4 the standard deviation of the tour lengths of the population at each cycle of the same run. Note how in the early cycles the AS identifies good tours which are subsequently refined in the rest of the run. Since the standard deviation of the population’s tour lengths never drops to zero, we are assured that the algorithm actively searches solutions which differ from the best-so-far found, which gives it the possibility of finding better ones. The search for better solutions is carried on in selected regions of the search space determined by the trail resulting from preceding cycles. This can be observed in Fig. 5, in which the vertical axis shows the average node branching of the problem’s graph. Although the graph is initially fully connected, those arcs whose trail level falls below a (very small) value \( \varepsilon \), which makes their probability of being chosen by ants negligible, are removed. The node branching of node \( i \) is therefore given by the number of edges which exit from node \( i \) and which have a trail level higher than \( \varepsilon \). Note how at the beginning of the run an ant could go from any node to any other (except for tabu list constraints), while at the end the possible choices are significantly reduced.

The same process can be observed in the graphs of Fig. 6, where the AS was applied to a very simple 10-cities problem (CCA0, from [20]), and which depict the effect of ant search on the trail distribution. In the figure the length of the edges is proportional to the distances between the towns; the thickness of the edges is proportional to their trail level. Initially (Fig. 6(a)) trail is uniformly distributed on every edge, and search is only directed by visibilities. Later on in the search process (Fig. 6(b)) trail has been deposited on the edges composing good tours, and is evaporated completely from edges which belonged to bad tours. The edges of the worst tours actually resulted to be deleted from the problem graph, thus causing a reduction of the search space.

Besides the tour length, we also investigated the stagnation behavior, i.e. the situation in which all the ants make the same tour. This indicates that the system has ceased to explore new possibilities and no better tour will arise. With some parameter settings we observed that, after several cycles, all the ants followed the same tour despite the stochastic nature of the algorithms because of a much higher trail level on the edges.
average node branching of a run going to stagnation behavior (Oliver30). Typical run obtained setting $\alpha = 5$ and $\beta = 2$. This high trail level made the probability that an ant chooses an edge not belonging to the tour very low. For an example, see the Oliver30 problem, whose evolution of average branching is presented in Fig. 7. In fact, after 2500 cycles circa, the number of arcs exiting from each node sticks to the value of 2, which—given the symmetry of the problem—means that ants are always following the same cycle.

This led us to also investigate the behavior of the ant-cycle algorithm for different combination of parameters $\alpha$ and $\beta$ (in this experiment we set $NC_{max} = 2500$). The results are summarized in Fig. 8, which was obtained running the algorithm ten times for each couple of parameters, averaging the results and ascribing each averaged result to one of the three following different classes.

- **Bad solutions and stagnation:** For high values of $\alpha$ the algorithm enters the stagnation behavior very quickly without finding very good solutions. This situation is represented by the symbol $\emptyset$ in Fig. 8.
- **Bad solutions and no stagnation:** If enough importance was not given to the trail (i.e., $\alpha$ was set to a low value) then the algorithm did not find very good solutions. This situation is represented by the symbol $\infty$.
- **Good solutions:** Very good solutions are found for $\alpha$ and $\beta$ values in the central area (where the symbol used is $\circ$). In this case we found that different parameter combinations (i.e., $(\alpha = 1, \beta = 1)$, $(\alpha = 1, \beta = 2)$, $(\alpha = 1, \beta = 5)$, $(\alpha = 0.5, \beta = 5)$) resulted in the same performance level; the same result (the shortest tour known on the Oliver30 problem) was obtained in approximately the same number of cycles.

The results obtained in this experiment are consistent with our understanding of the algorithm: a high value for $\alpha$ means that trail is very important and therefore ants tend to choose edges chosen by other ants in the past. On the other hand, low values of $\alpha$ make the algorithm very similar to a stochastic multigreedy algorithm.

In Fig. 9 we present the best tour we found using the experimentally determined optimal set of parameters values for the ant-cycle algorithm, $\alpha = 1$, $\beta = 5$, $\rho = 0.5$, $Q = 100$. This tour is of length 423.741 and presents two inversions, 2–1 and 25–24, with respect to the best tour published in [34].

The major strengths of the ant-cycle algorithm can be summarized as:

- Within the range of parameter optimality the algorithm always finds very good solutions for all the tested problems.
A.

many ants and using the trail communication system; that is, a communicating ants, there is an "optimality point" given by run with among ants than with no communication. In case of communication, the number of evenly spaced points: this is a problem with a priori known optimal solution (160 if each edge has length 10, see Fig. 11).

We ran a set of experiments to assess both the impact of the number of cycles required to reach the optimum, multiplied by the number of ants used in each set of runs, the ordinate shows the so-called one-ant cycles, that is, the number of cycles required to reach the optimum, multiplied by the number of ants used (in order to evaluate the efficiency per ant, and have comparable data). The algorithm has always been able to identify the optimum with any number of cities (m ≈ n); this property was used in the assessment of the computational complexity (Section III).

A second set of tests has been carried out with 16 cities randomly distributed (16 cities random graph). Again we found that the optimal performance was reached with 8–16 ants, a number comparable with the dimension of the problem to be solved.

B. Initialization

This experiment was run in order to study whether the initial ant distribution influences the AS performance. We tested whether there is any difference between the case in which all ants at time t = 0 are in the same city and the case in which they are uniformly distributed. We used ant-cycle the result was that there is a synergistic effect in using

in which the synergistic effects reach a maximum. The results are shown in Figs. 12 and 13.

In Fig. 12 we compare a situation in which ants do not communicate (α = 0), with a situation in which they communicate (α = 1). Results show that communication is indeed exploited by the algorithm. In Fig. 13 we report an experiment in which the 4 × 4 grid problem was solved with m ∈ {4, 8, 16, 32, 64}. The abscissa shows the total number of ants used in each set of runs, the ordinate shows the so-called one-ant cycles, that is, the number of cycles required to reach the optimum, multiplied by the number of ants used (in order to evaluate the efficiency per ant, and have comparable data). The algorithm has always been able to identify the optimum with any number of cells (t = 0), the same integer number of ants on every town (this forces m to be a multiple of n). Uniform distribution was the default starting configuration in the experiments.
applied to the 16 cities random graph, to the $4 \times 4$ grid, and to the Oliver30 problem. In all three cases, distributing ants resulted in better performance.

We also tested whether an initial random distribution of the ants over the cities performed better than a uniform one; results show that there is no significant difference between the two choices, even though the random distribution obtained slightly better results.

C. Elitist Strategy

We use the term "elitist strategy" (because in some way it resembles the elitist strategy used in genetic algorithms) for the modified algorithm in which at every cycle the trail laid on the edges belonging to the best-so-far tour is reinforced more than in the standard version. We added to the trail of each arc of the best tour a quantity $e \cdot Q/L^*$, where $e$ is the number of elitist ants and $L^*$ is the length of the best found tour. The idea is that the trail of the best tour, so reinforced, will direct the search of all the other ants in probability toward a solution composed by some edges of the best tour itself.

The test were carried out again on the Oliver30 problem (the run was stopped after $N_{CMA} = 2500$ cycles) and results indicated that there is an optimal range for the number of elitist ants: below it, increasing their number results in better tours discovered and/or in the best tour being discovered earlier; above it, the elitist ants force the exploration around suboptimal tours in the early phases of the search, so that a decrease in performance results. Fig. 14 shows the outcome of a test on the Oliver30 problem where this behavior is evident.

D. Increasing the Problem Dimensions

The algorithm complexity presented in Section III, $O(NC \cdot n^3)$, says nothing about the actual time required to reach the optimum. The experiment presented in this section is devoted to investigating the efficiency of the algorithm for increasing problem dimensions. Results are reported in Table II for the case of similar problems with increasing dimensions ($r \times r$ grids with the edge length set to 10, as in Fig. 11). It is interesting to note that, up to problems with 64 cities, the algorithm always found the optimal solution.

VI. COMPARISON WITH OTHER HEURISTICS

In this section we compare the efficacy of our algorithm to that of other heuristics, both tailored and general-purpose.

A. Comparison with TSP-Tailored Heuristics

In this section we compare ant-cycle with the heuristics contained in the package "Travel" [4]. This package represents the distances between the cities as an integer matrix and so we implemented an analogous representation in our system. The results of the comparisons on Oliver30 are shown in Table III, where the first column is the length of the best tour identified by each heuristic, and the second column is the improvement on the solution as obtained by the 2-opt heuristic (the 2-opt heuristic is an exhaustive exploration of all the permutations obtainable by exchanging 2 cities). Comparisons have been carried out also with the Lin-Kernighan [27] improvement of

---

**TABLE II**

<table>
<thead>
<tr>
<th>Problem Dimension</th>
<th>Number of Cycles to Reach the Optimum</th>
<th>Time Required to Find the Optimum (seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$4 \times 4$</td>
<td>16</td>
<td>5.6</td>
</tr>
<tr>
<td>$5 \times 5$</td>
<td>25</td>
<td>13.6</td>
</tr>
<tr>
<td>$6 \times 6$</td>
<td>36</td>
<td>60</td>
</tr>
<tr>
<td>$7 \times 7$</td>
<td>49</td>
<td>320</td>
</tr>
<tr>
<td>$8 \times 8$</td>
<td>64</td>
<td>970</td>
</tr>
</tbody>
</table>

Tests were run on an IBM-compatible PC with 8086 based processor.

---

In our case the effect of an ant is to increment the value of the trail on edges belonging to its tour; therefore in our case the equivalent of "saving" an individual is to reinforce its contribution.

\[ n^3, \]
TABLE III
PERFORMANCE OF THE AS CYCLE ALGORITHM COMPARED
WITH OTHER APPROACHES. RESULTS ARE AVERAGED OVER
TEN RUNS, AND ROUNDED TO THE NEAREST INTEGER.6

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Basic</th>
<th>2-opt</th>
<th>L-K</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ant-Cycle</td>
<td>420</td>
<td>420</td>
<td>420</td>
</tr>
<tr>
<td>Near Neighbor</td>
<td>597</td>
<td>437</td>
<td>421</td>
</tr>
<tr>
<td>Far Insert</td>
<td>428</td>
<td>421</td>
<td>420</td>
</tr>
<tr>
<td>Near Insert</td>
<td>510</td>
<td>492</td>
<td>420</td>
</tr>
<tr>
<td>Space Filling Curve</td>
<td>464</td>
<td>431</td>
<td>421</td>
</tr>
<tr>
<td>Sweep</td>
<td>486</td>
<td>426</td>
<td>420</td>
</tr>
<tr>
<td>Random</td>
<td>1212</td>
<td>663</td>
<td>421</td>
</tr>
</tbody>
</table>

TABLE IV
PERFORMANCE OF AS COMPARED TO TS AND SA ON THE OLIVER30 PROBLEM.
RESULTS ARE AVERAGED OVER TEN RUNS USING INTEGER DISTANCES

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Best</th>
<th>Average</th>
<th>Std.dev.</th>
</tr>
</thead>
<tbody>
<tr>
<td>AS</td>
<td>420</td>
<td>420.4</td>
<td>1.3</td>
</tr>
<tr>
<td>TS</td>
<td>420</td>
<td>420.6</td>
<td>1.5</td>
</tr>
<tr>
<td>SA</td>
<td>422</td>
<td>459.8</td>
<td>25.1</td>
</tr>
</tbody>
</table>

the first-column solutions, which has been able to reduce the
length of any tour to 420 (or 421, depending on the starting
solution provided by the basic algorithms).

Note how ant-cycle consistently outperformed 2-opt, while
its efficacy—i.e., the effectiveness it has in finding very good
solutions—can be compared with that of Lin-Kernighan. On
the other hand, our algorithm requires a much longer compu-
tational time than any other tested special-purpose heuristic.

As a general comment of all the tests, we would like to
point out that, given a good parameter setting (for instance
\( \alpha = 1, \beta = 5, \rho = 0.5, Q = 100, \epsilon = 8 \)), our algorithm
consistently found the best known solution for the Oliver30
problem, and converged quickly toward satisfactory solutions.
It always identified for Oliver30 the best-known solution of
length 423.741 in less than 400 cycles, and it took only \( \approx 100 \)
cycles to reach values under 430. The algorithm never fell
into the stagnation behavior. In fact, the average branching
was always greater than 2, and the average length of tours
was never equal to the best tour found but remained somewhat
above it. This indicates that the ants followed different tours.

B. Comparison with General-Purpose Heuristics

We also compare ant-cycle with other general-purpose
heuristics. This comparison is more fair to the AS, which in
fact is a general-purpose heuristic, and not a specialized algo-
rithm for the TSP. To run the comparisons, we implemented
a Simulated Annealing (SA) [1], and a Tabu Search (TS)
[17], [18]; we let each of them run 10 times on the Oliver30
data. SA used the annealing function \( T(t+1) = \alpha T(t) \), with
\( \alpha = 0.99 \): TS was implemented with tabul list length varying
in the interval \([20, 50]\). TS and SA, and the AS as well, were
allowed to run for 1 hour on a IBM-compatible PC with 80386
Intel processor. The results are presented in Table IV.

Results show that the AS for this problem was as effective as
TS and better than SA, when running under the same hardware
and time constraints.

VII. GENERALITY OF THE APPROACH

As we said in Section I, the AS is both versatile and robust.
Versatility is exemplified by the ease with which AS can be
applied to the asymmetric TSP (ATSP), a particular kind of
TSP (Section VII-A). Robustness is exemplified by the possi-
bility of using the same algorithm, although appropriately
adapted, to solve other combinatorial optimization problems
like the quadratic assignment problem (QAP), and the job-shop
scheduling problem (JSP) (Section VII-B).

A. Versatility: The ATSP

The asymmetric traveling salesman problem is a TSP in
which the distance between two nodes is not symmetric (i.e.,
in general \( d_{ij} \neq d_{ji} \)). The ATSP is more difficult than the
TSP; in fact, while symmetric TSP can be solved optimally
even on graphs with several thousand nodes, ATSP instances,
and particularly ATSP instances where the distance matrix
is almost symmetric, can be solved to the optimum only on
graphs with a few dozen nodes [16], [26].

The application of the AS to the ATSP is straightforward,
as no modifications of the basic algorithm are necessary. The
computational complexity of a cycle of the algorithm remains
the same as in the TSP application, as the only differences
are in the distance and trail matrices which are no longer
symmetric. We chose as test problem the RY48P problem [16],
a difficult problem instance with a distance distribution that is
hard to solve even with tailored heuristics and branch and
bound procedures. We ran AS 5 times on it, each time for
4000 cycles. The average length of the best found tour was
14899, that is 3.3% longer than the optimal one. The average
number of cycles to find this result was 1517.

B. Robustness: QAP and JSP

Let’s now consider the robustness of the AS approach. Many
combinatorial problems can be solved by the AS. To apply the
autocatalytic algorithm to a combinatorial problem requires
defining:

1) an appropriate graph representation with search by many
simple agents for the problem;
2) the autocatalytic (i.e. positive) feedback process;
3) the heuristic that allows a constructive definition of the
solutions (which we also call “greedy force”);
4) the constraint satisfaction method (that is, the tabu list).

This has been done for two well-known combinatorial oti-
minization problems—Quadratic Assignment (QAP) and Job-
Shop Scheduling (JSP)—each time obtaining an adapted ver-
Santion of the AS that could effectively handle the relative
problem. The most difficult (and ad hoc) tasks to face when
applying the AS are to find an appropriate graph representation
for the problem to be solved and a greedy force as heuristic.

Quadratic Assignment Problem: A QAP of order \( n \) is the
problem that arises when trying to assign \( n \) facilities to \( n \)
locations.

6The name “basic” means the basic heuristic, with no improvement.
Formally the problem is usually defined using two $n \times n$ (symmetric) matrices:

$$D = \{d_{ij}\},$$

where $d_{ij}$ is the distance between location $i$ and location $j$;

$$F = \{f_{hk}\},$$

where $f_{hk}$ is the flow (of information, products or some other quantity) between facility $h$ and facility $k$.

A permutation $\pi$ is interpreted as an assignment of facility $h = \pi(i)$ to location $i$, for each $i = 1, \ldots, n$. The problem is then to identify a permutation $\pi$ of both row and column indexes of the matrix $F$ that minimizes the total cost:

$$\min z = \sum_{i,j=1}^{n} d_{ij} f_{\pi(i)\pi(j)}.$$

To apply AS to QAP we used the same algorithm as in the case of the TSP, after having studied an approximation of the QAP objective function that allows a problem representation on the basis of a single matrix which is used by the heuristic. The QAP objective function was expressed by a combination of the "potential vectors" of distance and flow matrices. The potential vectors $D$ and $F$, are the row sums of each of the two matrices. Consider the following example:

$$D = \begin{bmatrix}
0 & 1 & 2 & 3 \\
1 & 0 & 4 & 5 \\
2 & 4 & 6 & 0 \\
3 & 5 & 6 & 0 \\
\end{bmatrix}, \quad D = \begin{bmatrix}
6 & 12 \\
10 & 14 \\
\end{bmatrix},$$

$$F = \begin{bmatrix}
0 & 60 & 50 & 10 \\
60 & 0 & 30 & 20 \\
50 & 30 & 0 & 50 \\
10 & 20 & 50 & 0 \\
\end{bmatrix}, \quad F = \begin{bmatrix}
120 & 110 & 130 & 80 \\
\end{bmatrix}.$$

From the two potential vectors, a third matrix $S$ is obtained, where each element is computed as $s_{ih} = d_{i} \cdot f_{ih}$, $d_{i}$ and $f_{ih}$ being elements of the potential vectors.

$$S = \begin{bmatrix}
720 & 660 & 780 & 480 \\
1200 & 1100 & 1300 & 800 \\
1440 & 1320 & 1560 & 960 \\
1680 & 1540 & 1820 & 1200 \\
\end{bmatrix}.$$

The ants choose the node to move to using the inverse of the values of $S$ as visibility data, $\eta_{ih} = 1/s_{ih}$, thus interpreting each element $s_{ih}$ as the heuristic value of the choice of assigning to location $i$ the facility $h$. To show how the heuristic works to build a solution we assume, for simplicity, that the ants act in a deterministic way (and not probabilistically, as it happens in the algorithm), and we don't consider the effect of the trail (i.e., we set all trails $\tau_{ih} = 1$). In constructing the solution we consider the columns of matrix $S$ one by one, starting from that corresponding to the activity with greatest flow potential, and we assign this activity to the location with least distance potential, according to the "min-max" rule. In our example first activity 3 will be assigned to location 1 because the element $o_{13}$ is the smallest of its column; we then pair activity 1 to location 2 (the coupling activity 1—location 1 is inhibited because location 1 already has an assigned activity); continuing in the same way one obtains the couplings 2–3, and 4–4.

We compared AS, and a version of the AS to which was added a non deterministic hill climbing procedure, with many other well known heuristics (see [28] for more details). Experiments were run on IBM-compatible PC's with a 80286 Intel processor, and were stopped after one hour time. The test problems used are those known as Nugent problems [29], Elshafei [15], and Krarup [25]. As can be seen in Table V, the performance of AS was always very good [5]. Ant System always found a result within 5% of the best known, while AS with local optimization always found, except for the Nugent 30 problem, the best known solution. This application is described in details in [28].

**Job-Shop Scheduling Problem:** The JSP can be described as in the following. A set of $M$ machines and a set of $J$ jobs are given. The $j$th job ($j = 1, \ldots, J$) consists of an ordered sequence (chain) of operations from a set $O = \{o_{1}, \ldots, o_{m}\}$. Each operation $o_{jm} \in O$ belongs to job $j$ and has to be processed on machine $m$ for $d_{jm}$ consecutive time instants. $N = |O|$ is the total number of operations. The problem is to assign the operations to time intervals in such a way that no two jobs are processed at the same time on the same machine and the maximum of the completion times of all operations is minimized [22].

To apply the AS to JSP we chose the following representation. A JSP with $M$ machines, $J$ jobs and operation set $O$ is represented as a directed weighted graph $Q = (O', A)$ where $O' = O \cup \{o_{0}\}$, and $A$ is the set of arcs that connect $o_{0}$ with the first operation of each job and that completely connect the nodes of $O$ except for the nodes belonging to a same job. Nodes belonging to a same job are connected in sequence (that is, a node is only connected to its immediate successor, see Fig. 15). Note that graph $Q$ is not the graph with cliques representing tasks that is usually utilized to represent the JSP. Node $o_{0}$ is necessary in order to specify which job will be scheduled first, in case several jobs have their first operation on the same machine. We have therefore $N + 1$ nodes and $N(N-1) + |J|$ arcs, where all the nodes are pairwise connected except $o_{0}$, which is connected only to the first operation of each job. Each arc is weighted by a pair of numbers, $(\tau_{kl}, \eta_{kl})$. The first, $\tau_{kl}$, is the trail level, while the second is the visibility $\eta_{kl}$ and is computed according to a desirability measure derived from a greedy problem specific heuristic like the Longest Processing Time or the Shortest Completion Time. The order in which the nodes are visited by each ant specifies the proposed solution. For instance, consider
A major issue in defining any distributed system is the definition of the communication protocol. In the AS a set of ants communicate by modifications of a global data structure: after each tour the trail left on each ant’s tour will change the probability with which the same decision will be taken in the future. A heuristic also guides ants in the early stages of the computational process, when experience has not yet accumulated into the problem structure. This heuristic automatically loses importance (remember the coefficient \( \rho \) related to evaporation) as the experience gained by ants, and saved in the problem representation, increases.

One way to explain the behavior of AS on the TSP problem is the following. Consider the transition matrix \( p^k(t) \) of ant \( k \): every element \( p^k_{ij}(t) \) is the transition probability from town \( i \) to town \( j \) at time \( t \) as defined by (4). At time \( t = 0 \) each \( p^k_{ij}(0) \) is proportional to \( \eta_{ij} \), i.e., closer towns are chosen with higher probability. As the process evolves, \( p^k(t) \) changes its elements according to (1) and (4). The process can therefore be seen as a space deformation, in which path cost is reduced between towns which are connected by edges with a high amount of traffic, and, conversely, path cost is incremented between towns connected by edges with low traffic levels. From simulations we observed that the matrix \( p^k(t) \), at least in the range of optimality for our parameters, converges to a state\(^9\) that is very close to stationary (i.e., variations in the transition matrix \( p^k(t) \) are very small). When this state is reached the behavior of the ants is dependent on the kind of transition matrix obtained. We observed two situations: in the most rare one, occurring (as we saw in Section IV) for particular parameter settings, only two transition probabilities are significantly higher than zero in every row and therefore all the ants choose the same edge at each step and no new tour is searched. In the most common situations instead, most of the rows have only a few transition probabilities with a significant value. In these cases search never stops, even if the number of significant transitions is highly reduced, with respect to the initial situation. Consider for example Fig. 16, obtained as the steady-state transition matrix for a randomly generated 10-town problem.

\[\begin{array}{cccccccccc}
1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 0 \\
2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
3 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
4 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
5 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
6 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
7 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
8 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
9 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\end{array}\]

\(^9\) The stochastic process that rules the evolution of the matrix \( p^k(t) \) is a Markov process with infinite memory.

VIII. DISCUSSION OF SOME AS CHARACTERISTICS

A major issue in defining any distributed system is the definition of the communication protocol. In the AS a set of ants communicate by modifications of a global data structure: after each tour the trail left on each ant’s tour will change the probability with which the same decision will be taken in the future. A heuristic also guides ants in the early stages of the computational process, when experience has not yet accumulated into the problem structure. This heuristic automatically loses importance (remember the coefficient \( \rho \) related to evaporation) as the experience gained by ants, and saved in the problem representation, increases.
Another way to interpret how the algorithm works is to imagine having some kind of probabilistic superimposition of effects: each ant, if isolated (that is, if \( \alpha = 0 \)), would move with a local, greedy rule. This greedy rule guarantees only locally optimal moves and will practically always lead to bad final results. The reason the rule doesn’t work is that greedy local improvements lead to very bad final steps (an ant is constrained to make a closed tour and therefore choices for local improvements lead to very bad final steps (an ant is followed by an ant ruled by a greedy policy is composed of some (initial) parts that are very good and some (final) parts that are not. If we now consider the effect of the simultaneous presence of many ants, then each one contributes to the trail distribution. Good parts of paths will be followed by many ants and therefore they receive a great amount of trail. On the contrary, bad parts of paths are chosen by ants only when they are obliged by constraint satisfaction (remember the tabu list); these edges will therefore receive trail from only a few ants.

**IX. Conclusion**

This paper introduces a new search methodology based on a distributed autocatalytic process and its application to the solution of a classical optimization problem. The general idea underlying the Ant System paradigm is that of a population of agents each guided by an autocatalytic process directed by a greedy force. Were an agent alone, the autocatalytic process and the greedy force would tend to make the agent converge to a suboptimal tour with exponential speed. When agents interact it appears that the greedy force can give the right suggestions to the autocatalytic process and facilitate quick convergence to very good, often optimal, solutions without getting stuck in local optima. We have speculated that this behavior could be due to the fact that information gained by agents during the search process is used to modify the problem representation and in this way to reduce the region of the space considered by the search process. Even if no tour is completely excluded, bad tours become highly improbable, and the agents search only in the neighborhood of good solutions.

The main contributions of this paper are the following.

1. We employ positive feedback as a search and optimization tool. The idea is that if at a given point an agent (ant) has to choose between different options and the one actually chosen results to be good, then in the future that choice will appear more desirable than it was before\(^ {10} \).
2. We show how synergy can arise and be useful in distributed systems. In the AS the effectiveness of the search carried out by a given number of cooperative ants is greater than that of the search carried out by the same number of ants, each one acting independently from the others.
3. We show how to apply the AS to different combinatorial optimization problems. After introducing the AS by an application to the TSP, we show how to apply it to the ATSP, the QAP, and the JSP.

We believe our approach to be a very promising one because of its generality (it can be applied to many different problems, see Section VII), and because of its effectiveness in finding very good solutions to difficult problems.

Related work can be classified in the following major areas:

- studies of social animal behavior;
- research in “natural heuristic algorithms”;
- stochastic optimization.

As already pointed out the research on behavior of social animals is to be considered as a source of inspiration and as a useful metaphor to explain our ideas. We believe that, especially if we are interested in designing inherently parallel algorithms, observation of natural systems can be an invaluable source of inspiration. Neural networks [32], genetic algorithms [23], evolution strategies [30, 33], immune networks [3], simulated annealing [24] are only some examples of models with a “natural flavor”. The main characteristics, which are at least partially shared by members of this class of algorithms, are the use of a natural metaphor, inherent parallelism, stochastic nature, adaptivity, and the use of positive feedback. Our algorithm can be considered as a new member of this class. All this work in “natural optimization” [12, 9] fits within the more general research area of stochastic optimization, in which the quest for optimality is traded for computational efficiency.

**ACKNOWLEDGMENT**

The authors would like to thank two of the reviewers for the many useful comments on the first version of this paper. We also thank Thomas Bäck, Hughes Bersini, Jean-Louis Deneubourg, Frank Hoffmeister, Mauro Leoncini, Francesco Maffioli, Bernard Manderik, Giovanni Manzini, Daniele Montanari, Hans-Paul Schwefel and Frank Smieja for the discussions and the many useful comments on early versions of this paper.

**REFERENCES**


\(^{10}\) Reinforcement of this nature is used by the reproduction-selection mechanism in evolutionary algorithms [23, 30, 33]. The main difference is that in evolutionary algorithms it is applied to favor (or disfavor) complete solutions, while in AS it is used to build solutions.

Marco Dorigo (S’92-M’93) was born in Milan, Italy, in 1961. He received the Laurea (Master of Technology) in industrial technologies engineering in 1986, and the Ph.D. in information and systems electronic engineering in 1992 from Politecnico di Milano, Milan, Italy.

In 1992 he was a research fellow at the International Computer Science Institute, Berkeley, CA. In 1993 he became a NATO-CNR fellow at the IRIDIA Laboratory, Free University of Brussels, Belgium, and in 1994, he was awarded a Human Capital and Mobility two-year fellowship by the Commission of the European Community to continue his research at the IRIDIA laboratory, where he currently works. His areas of research include evolutionary computation, reinforcement learning, and their application to autonomous robotics and to combinatorial optimization.

Dr. Dorigo is an Associate Editor for the IEEE TRANSACTIONS ON SYSTEMS, MAN, AND CYBERRNETICS, and he was the Guest Editor of a special issue on "Learning autonomous robots". He is a member of the editorial board of the Evolutionary Computation journal and of the Adaptive Behavior journal. He is a member of the Politecnico di Milano Artificial Intelligence and Robotics Project, and of the Italian Association for Artificial Intelligence (AIP). He took part to several CEC ESPRIT Projects, and National research projects.

Vittorio Maniezzo was born in Ferrara, Italy, in 1962. He received the Laurea (Master of Technology) in electronic engineering in 1986 and the Ph.D. degree in automatic control and computer science engineering in 1993, both from Politecnico di Milano, Milan, Italy.

He is currently with the University of Bologna and is a member of the Politecnico di Milano Artificial Intelligence and Robotics Project. He took part to several CEC ESPRIT and National research projects. His current research interests are in the fields of machine learning (evolutionary techniques for sensorimotor coordination, cognitive modeling) and of combinatorial optimization (evolutionary heuristic algorithms, column generation techniques).

Dr. Maniezzo is a member of the Italian Association for Artificial Intelligence (AIP) and of the Italian Association for Operations Research (AIRO).

Alberto Colomini was born in Milan, Italy, in 1946. He received the Laurea (Master of Technology) in electronic engineering in 1970 from Politecnico di Milano, Italy.

He is an Associate Professor of Operations Research at the Politecnico, and Director of the Research center in Decision Support Systems for Environment and Land Use at the MIP (Master Imprese-Politecnico). He currently works in the fields of Combinatorial Optimization (in particular, of heuristics from nature), DSS methodologies (Analytic Hierarchy Process and ELECTRE methods) and DSS applications to environmental impact assessment.

Colomini is Associate Editor of Ricerca Operativa, the Italian journal of Operations Research.