VERY STRONGLY CONSTRAINED PROBLEMS: AN ANT COLONY OPTIMIZATION APPROACH

Vittorio Maniezzo, Matteo Roffilli

Dept. Computer Science, University of Bologna,
Mura Anteo Zamboni 7, Bologna, Italy
e-mail: vittorio.maniezzo@unibo.it, roffilli@csr.unibo.it

Abstract: Ant Colony Optimization (ACO) is a class of metaheuristic 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 paper is composed of three parts. The first one frames ACO in current trends of research on metaheuristics for combinatorial optimization. The second outlines current research within the ACO community, reporting recent results obtained on different problems, while the third part focuses on a particular research line, named ANTS, providing some details on the algorithm and presenting results recently obtained on a prototypical strongly constrained problem: the set partitioning problem.

1. Introduction

Nowadays companies are facing many real world problems whose nature is intrinsically combinatorial. For example, trucks have to be routed, depots or sale points have to be located, containers have to be filled, wood or leather masters have to be cut, communication networks have to be designed, radio links must have an associated frequency, CPU time has to be scheduled, etc. Complexity theory tells us that the combinatorial problem underlying these applications is 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. In addition, several operational constraints are often imposed to ensure feasibility of the solutions, constraints which further complicate the solution process. Nevertheless, these instances must be solved and the constraints must be taken into account. Thus, the need to bow down to suboptimal

solutions emerges, provided that they are both 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*. Alternatively, one can 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*. Indeed, 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 approximate algorithms, loosely called *heuristic algorithms*. There is a huge amount of literature about how to construct a heuristic algorithm (henceforth heuristic), which lately concentrated on the design of so-called metaheuristic algorithms, or metaheuristics. A metaheuristic is essentially a basic heuristic with a superimposed mechanism to ensure the possibility to escape from local optima. A short list of the most representative metaheuristics includes: simulated annealing (Cerny, 1985), tabu search (Glover, 1989, 1990), guided local search (Voudouris and Tsang, 1995), greedy randomized adaptive search procedures (GRASP, Feo and Resende, 1995), iterated local search (Lourenco et al., 2002), evolutionary computation (EC, Fogel et al., 1966), and scatter search (Glover, 1977).

In this paper, we concentrate on the ACO metaheuristic framework showing how it can be applied successfully to many theoretical problems, which are studied as prototypes of real world problems. In particular, we will present a detailed result on a very constrained Set Partitioning problem (SPP). This is chosen as severely constrained problems usually are very hard to deal with metaheuristics, as they limit the effectiveness of the underlying local search or constructive basic heuristic, which is at the heart of the metaheuristic. Very often on these problems, and most evidently on the SPP, adapted exact techniques provide better computational results than metaheuristics.

The paper is structured as follows. Section 2 and 3 describe the common elements of the heuristics belonging to the Ant Colony Optimization class and the results obtained by current approaches on different problems. Section 4 concentrates on the ANTS approach, one method of the ACO class, describing its essential ingredients. Section 5 presents the Set Partitioning problem as one of the more constrained combinatorial optimization (CO) problems. Section 6 shows how it is possible to adapt the ANTS approach to solve the SP problem. Finally, Section 7 presents a brief discussion on the computational results and Section 8 a brief discussion.

2. Ant Colony Optimization

Colorni, Dorigo and Maniezzo (Colorni et al. (1991) and Dorigo (1992)) initially proposed the Ant System method, then structured by Dorigo and di Caro (1999) in the Ant Colony Optimization (ACO) framework, taking inspiration from the way ants collect food and coordinate their movements. The main underlying idea they extracted from biological studies about ants, was that of parallelizing the search of the optimal solution of CO problems by exploiting several constructive computational threads, called *ants*.

An ant is defined as a simple computational agent endowed with memory, which iteratively and independently constructs from scratch a solution for the problem to solve.

The ant works in a constructive way progressively stacking small pieces of solution. Partial solutions are treated as states: each ant moves from a state φ to another one ψ, producing a more complete partial solution. This process is iterated until the ant ends in a final state corresponding to a complete solution of the problem.

In order to ensure the feasibility of the final solution, at each step σ, the ant computes a set $A(\varphi)$ of feasible expansions to its current state φ (i.e. a collection of admissible new states), and *moves* to one of these according to a probability distribution specified as follows.

The probability $p_{\varphi\psi}$ of moving from state φ to state ψ depends on the combination of two values:

1. The *attractiveness* η of the move, as computed by some heuristic indicating the *a priori* desirability of that move;
2. The *trail level* τ of the move, indicating how proficient it has been in the past to make that particular move: it represents therefore an *a posteriori* indication of the desirability of that move.

The attractiveness represents the inclination of the ant, which selects a new state (i.e. a new direction toward the food) according to its internal evaluations.

The trail level, also called *pheromome* in accordance to the biological deposit of this chemical, codifies the memory of the ant or better of all the ants which have been in the same situation. Loosely speaking, the trail level is a way, biologically proven in the case of real ants, to enable the coordination of a colony of ants without a direct communication. For a detailed description of the biological experiments about trail level see the book by Dorigo and Stützle (2004).

From an operational point of view, 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 problem-dependent knowledge about feasible and infeasible moves.

Let us describe how to apply the ACO framework to common problems.

A CO problem is defined over a set $\mathbf{C} = \{c_1, \ldots, c_n\}$ of basic components and a subset S of C represents a solution of the problem. $\mathbf{F} \subseteq \mathbf{C}$ is the subset of *feasible solutions*, thus a solution S is feasible if and only if $S \in \mathbf{F}$. A *cost function* z is defined over the solution domain, $z : \mathbf{C} \rightarrow R$. The goal of the algorithm is to find a *minimum cost feasible solution* S^*, that is to find S^*: $S^* \in F$ and $z(S^*) \leq z(S)$, $\forall S \in F$. If the last requirement is not match, the algorithm anyway returns the best feasible solution found $\overline{S}$.

2.1 Ant System

The Ant System (AS) (Dorigo *et al.*, 1991) was the first algorithm implementing the ACO framework. The idea underlying it was to modify a constructive heuristic so that the ordering of the components could be recalculated at each iteration. The procedure takes into account not only the *a priori* expectation (attractiveness), η_i, of the usefulness of a particular component c_i, as in standard constructive approaches, but also an *a posteriori* measure (trail level), τ_i, of the goodness of solutions constructed using that particular component. The general framework can be schematized as follows.

AS algorithm

Step 1. (**Initialization**)
Initialize $\tau_{\varphi\psi}$, $\forall (\varphi,\psi)$

Step 2. (**Construction**)
for each ant k *do*
repeat
compute $\eta_{\varphi\psi}$, $\forall (\varphi,\psi)$
choose the state to move into, with probability $p_{\varphi\psi}$
until ant *k* has completed its solution.
(optional) carry the solution to its local optimum.
enddo

Step 3. (**Trail update**)
for each ant move (φ,ψ) *do*
compute $\Delta\tau_{\varphi\psi}$
update the trail matrix (evaporation, deposit)
enddo

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

Figure 1. Pseudo code for the AS algorithm.

Essentially, diversification and intensification are controlled by the τ management policy. A wide-accepted strategy for the choice of Step 2 is the follow:

- $p_{\varphi\psi}$ is equal to 0 for all moves which are infeasible, otherwise it is computed by means of the following formula , where α is a user-defined parameter ($0 \leq \alpha \leq 1$):

$$p_{\varphi\psi} \leftarrow \frac{\alpha\tau_{\varphi\psi} + (1-\alpha)\eta_{\varphi\psi}}{\sum_{feas} \alpha\tau_{\varphi\psi} + (1-\alpha)\eta_{\varphi\psi}} \qquad (1)$$

Parameter α 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, each ant *k* deposits a quantity of pheromone over its trail, fully backtracking the followed path from destination to source. Summing the contribution of all the *m* ants the trail levels are updated following the formula:

$$\tau_{\varphi\psi} \leftarrow (1-\rho)\tau_{\varphi\psi} + \sum_{k=1}^{m} \Delta^{k}_{\varphi\psi} \qquad (2)$$

where $\Delta\tau_{\varphi\psi}$ represents the sum of the contributions of all ants that used move ($\varphi\psi$) to construct their solution. The ants' contributions are proportional to the quality of the achieved solutions, that is the better an ant solution, the higher will be the trail contribution added to the moves it used. In order to avoid an unlimited growth of the trail level, a normalization procedure, called *evaporation* (as it resembles the biological evaporation of the pheromone), is executed before the trail update where ρ ($0 \leq \rho \leq 1$) is the pheromone evaporation rate. The evaporation manages the trade-off between the importance of the historical paths found (with high trail level) and the need to explore new paths (with a low trail level).

The importance of this original Ant System resides mainly in being the prototype of a number of ants-inspired algorithms, which have found interesting and successful applications.

3 An Outline of Current Research

The first application of AS 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 can be easily applied to it. Several authors built upon this initial contribution.

The TSP is the problem faceted by a salesman, who starts from his home and needs to visit all his customer's cities before returning at home. Obviously, he wants to minimize the cost of his trip. Exploiting the TSP problem, the basic AS has been improved during years taking into account both problem-dependent and performance issues. The main variations regard the trail update strategy and the computation of the attractiveness. In the following, we will first show a short collection of state-of-the-art ACO algorithms, then a list of problems which ants were successfully applied on.

3.1 Ants variants

Elitist Ant System

Dorigo (1992) and Dorigo *et al.* (1991, 1996) made a first improvement of the basic AS, called *elitist strategy*. The idea was to give more strength to the reinforcement of the trail level of the best ant, *i.e.* the ant that provided the best solution. This is accomplished defining a best ant whose contribution in the trail level update is:

$$\tau_{\varphi\psi} \leftarrow \tau_{\varphi\psi} + \sum_{k=1}^{m} \Delta_{\varphi\psi}^{k} + \gamma\Delta_{\varphi\psi}^{best}$$

Where γ is a parameter that defines the weight given to the best-ant.

Max-Min Ant System

Stützle and Hoos (1997, 2000) and Stützle (1999) introduced *Max-Min Ant System* (*MM*AS), a modification of Elitist Ant System. The authors explicitly introduced in the algorithm two parameters, a maximum and minimum trail level, whose values are chosen in a problem-dependent way in order to restrict possible trail values to the interval [T*min*, T*max*]. Moreover, *MM*AS controls the trail levels (initialized to their maximum value T*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 T*min* value. The T*min* and T*max* parameters are used to counteract premature stagnation of search, maintaining at the same time some kind of elitist strategy. When applied to TSP, *MM*AS performs better than AS.

Rank-Based Ant System

Bullnheimer *et al.* (1999) proposed yet another modification of AS, called *ASRank*, 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 *w* best ants.

Ant Colony System

Gambardella and Dorigo (1995) 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 that 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 (Dorigo and Gambardella, 1997a,b), 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, 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 visit. This is a decision rule based on a $q_0 \in [0,1]$ parameter that permits to modulate the exploration behavior, 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 *cl* vertices nearest to the one in subject, and the ants choose the next move scanning the candidate list instead of examining all the unvisited neighboring vertices.

Hyper-cube Framework for ACO

An alternative strategy to manage and update the pheromone values, called Hyper-Cube Framework (HCF) for ACO has been proposed by (Blum, 2004; Blum and Dorigo, 2003; Blum *et al.*, 2001). The main characteristic of this approach is that pheromone values are bounded between 0 and 1, with associated changes to the standard equations used to deal with pheromone. This idea is similar to that of *MM*AS with the difference that in this later case the maximal pheromone value is not bounded to one, but derived from the problem being solved. The idea underlying HFC is to model a CO problem by means of binary (*i.e.* $\{0,1\}$) decision variables. While in the standard AS pheromone information is typically used with problems where the decision variables appear to have several possible values, in this case each individual pheromone value actually relates to a single solution feature which a solution may or may not exhibit (from which the $\{0,1\}$ value). In this sense, a

solution is one corner in a hyper-cube of dimension *n*, where *n* is the number of components. In order to generate lower bounds for the CO problem, the HCF admits that the pheromone can assume values in the range [0,1]. In this case, a feasible solution corresponds to a convex combination of components.

Blum (2004) suggests that the HCF approach is more robust across problems with different objective function values, and that it allows for improved implementation of *intensification* and *diversification*. These techniques allow to intensify a search around particular solution features and to diversify a search into new spaces of solutions respectively.

3.2 Problems

In the past years, since ACO has been applied to a broad variety of problems and it is difficult to track every application. However, in the following we will present some of the most relevant problems from both an historical perspective and current trends. As reported in Table 1, current works on ACO are devoted to exploit the implicit parallelism of the algorithm in order to speed up the computation on modern multi-core processors. While these studies are focused on particular problems, we can recognize common strategies aimed at splitting the problem (Ouyang and Yan, 2004), exchanging information among colonies (Ellabib *et al.*, 2007), and deploying the best implementation (Delisle *et al.*, 2005). In the following, we will outline some of the most relevant applications.

3.2.1 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 is a NP-hard optimization problem (Sahni and Gonzales, 1976), it can be considered one of the hardest CO problems, and it can be solved to optimality only for comparatively small instances (n=36). For this reason, historically QAP was chosen as a second benchmark for ACO.

Basic AS was of limited effectiveness when applied to QAP but it was the first evidence of the robustness of AS-QAP.

An extension by Maniezzo and Colorni (1999) makes use of a well-tuned local optimizer, obtaining good results. 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 improve further the best solution found. The results obtained showed the Ant System's competitive performance on several test problems.

Table 1. An overview of ACO applications.

Problem type	**Problem name**	**Main references**
Routing	Traveling salesman	Dorigo, Maniezzo, and Colorni (1991)
		Dorigo (1992)
		Gambardella and Dorigo (1995)
		Dorigo and Gambardella (1997a,b)
		Stützle and Hoos (1997, 2000)
		Ouyang and Yan (2004)
	Vehicle routing	Bullnheimer, Hartl, and Strauss (1999a,b)
		Gambardella Taillard, and Agazzi (1999)
		Reimann, Stummer, and Doerner (2002)
		Ellabib et al., (2007)
	Sequential ordering	Gambardella and Dorigo (1997, 2000)
Assignment	Quadratic assignment	Maniezzo, Colorni, and Dorigo (1994)
		Stützle (1997b)
		Maniezzo and Colorni (1999)
		Maniezzo (1999)
		Stützle and Hoos (2000)
		Wiesemann and Stützle (2006)
	Graph colouring	Costa and Hertz (1997)
	Generalized assignment	Lourenco and Serra (1998)
	Frequency assignment	Maniezzo and Carbonaro (2000)
	University course	Socha, Knowles, and Sampels (2002)
	timetabling	Socha, Sampels, and Manfrin (2003)
Scheduling	Job shop	Colorni, Dorigo, Maniezzo, and Trubian (1994)
	Open shop	Pfahringer (1996)
	Flow shop	Stützle (1998)
	Total tardiness	Bauer, Bullnheimer, Hartl, and Strauss (2000)
	Total weighted tardiness	den Besten, Stützle, and Dorigo (2000)
		Merkle and Middendorf (2003)
	Project scheduling	Merkle, Middendorf, and Schmeck (2002)
	Group shop	Blum (2003)
Subset	Multiple knapsack	Leguizamon and Michalewicz (1999)
	Max independent set	Leguizamon, Michalewicz and Schutz (2001)
	Redundancy allocation	Liang and Smith (1999)
	Set covering	Hadji, Rahoual, Talbi, and Bachelet (2000)
	Weight constrained graph tree partition	Cordone and Maffioli (2001)
	Arc-weighted k-cardinality tree	Blum and Blesa (2003)
	Maximum clique	Fenet and Solnon (2003)

In addition, several other systems previously introduced were adapted to the QAP. For example, two efficient techniques are the MMAS-QAP algorithm (Stützle (1997b), Stützle and Hoos (2000)) and Hybrid AS (HAS-QAP) (Gambardella et al. 1999). Both of them consider the problem instances classified in two categories: randomly generated problems without any structure and structured real-life problems. As the performance of heuristic approaches is strongly dependent on

the type of considered problem, they analyse the obtained results respect to the best performing methods in the corresponding category.
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.
MMAS-QAP seems to be one of the most promising approaches for the solution of structured real life QAPs.

3.2.2 ACO approaches to VRP

Vehicle Routing Problems (VRPs) are 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 (Toth and Vigo, 2001). 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. The goal in the VRP is to find a collection of routes that minimizes the total travel time such that each costumer is server by exactly on vehicle, the route of each vehicle starts and ends at the depot, and the total demand covered by each vehicle does not exceed its capacity.
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 heuristics to VRP are the following.
A direct extension of AS based on the algorithm is AS-VRP, an algorithm designed by Bullnheimer *et al.* (1999b). 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.
Gambardella *et al.* (1999b) also 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. Recently, some authors (Ellabib *et al.*, 2007) proposed a novel strategy to exchange information among ants suitable for optimized parallel implementations.

3.3 ACO approaches to 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. Maniezzo and Carbonaro (2000) developed one, which is not very tight but is efficient to compute, and included it in the ANTS algorithm (see next Section). Computational results were obtained on three well-known problem datasets from the literature. They show that the ANTS heuristic is competitive with the best approaches so far presented.

4 Approximate Nondeterministic Tree Search (ANTS) algorithm

Approximate Nondeterministic Tree Search (ANTS) is an extension of the Ant System proposed in (Maniezzo, 1999), that exploits ideas from mathematical programming. ANTS algorithm specifies some underdefined elements of the general ants 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. 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 estimated effectively 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 t corresponds to a partial problem solution it is possible to compute a lower bound (LB) to the cost of a complete solution containing t. Therefore, for each feasible move ($\varphi \rightarrow \psi$), it is possible

to compute the lower bound to the cost of a complete solution containing it: 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.

- A tight bound gives strong indications on the opportunity of a move.
- When the bound value becomes greater than the current upper bound, it is obvious that the considered move leads to a partial solution which conclusion can not be better than the current best one. Thus, the move can therefore be discarded from further analysis.
- 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. These permits to a priori eliminate some variables, when compared to an upper bound to the optimal problem solution cost. The result is a reduction of the number of possible moves, therefore a reduction of the search space.
- 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.
- The use of LP bounds is a very effective and straightforward general policy, every time 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 algorithm evaluates each solution against the last n ones globally constructed by ANTS. As soon as n solutions are available, the algorithm computes the moving average $\bar{z}$ of their cost. Now, the cost of each new solution z_{new} is compared with $\bar{z}$ and then used to compute the new moving average value. In the case z_{new} is lower than $\bar{z}$, the corresponding trail levels are increased, otherwise decreased, following the formula:

$$\Delta\tau_{\varphi\psi} \leftarrow \vartheta\left(1 - \frac{z_{new} - LB}{\bar{z} - LB}\right) = \vartheta\left(\frac{\bar{z} - z_{new}}{\bar{z} - LB}\right) \qquad (4)$$

where LB is a lower bound to the optimal problem solution cost and ϑ a prefixed parameter. In particular, if the new solution is better of the moving average (*i.e.* $\bar{z} > z_{new}$), the trail level is increased by a quantity $\bar{z} - z_{new}$ normalized by a value that is proportional to the phase of the optimization. If the algorithm is far from the estimated lower bound the improvement $\Delta\tau_{\varphi\psi}$ of the solution is diminished (*i.e.* $\bar{z} - LB$ is high), otherwise if the algorithm is approaching the lower bound the improvement is amplified.

In the case of a new solution worse than the moving average, the trail level is decreased following the same dynamic normalization.

This use of a dynamic scaling procedure permits to discriminate small achievement in the latest stage of search, while avoiding focalizing search only around good achievement in the earliest stages. This mechanism implicitly performs the pheromone evaporation, which in fact is not explicitly formulated. Learning method based on this idea are called *reinforcement comparison* methods; they are able to compare the solution values respect to a reference measure varying during the construction of the solution. One of the more difficult aspects to be considered in reinforcement learning algorithms is the trade-off between *exploration* and *exploitation*. To obtain good results, an agent must prefer actions that it has tried in the past and found to be effective in producing desirable solutions (exploitation): but to discover such actions, it has to try actions that it has not selected before (exploration). Neither exploration nor exploitation can be pursued exclusively without failing at the task: for this reason, ANTS system uses this stagnation avoidance procedure to facilitate exploration and attractiveness mechanism to determine the desirability of known moves.

Based on the described elements, the ANTS algorithm is reported in Figure 2. It can be noticed that the general structure of the ANTS algorithm is closely similar to that of a standard tree-search procedure. At each stage the algorithm obtains in fact a partial solution which is expanded by branching on all possible children. A bound is then computed for each offspring 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.

ANTS algorithm

Step 1. (**Initialization**)
Compute a (LP) lower bound to the problem to solve
Initialize $\tau_{\varphi\psi}$, $\forall$ (φ,ψ) as a function of the primal variable values

Step 2. (**Construction**)
for each ant k *do*
repeat
Compute $\eta_{\varphi\psi}$, $\forall$ (φ,ψ) as a lower bound to the cost of a complete solution containing ψ
Choose the state to move into, with probability given by (1)
until ant *k* has completed its solution
carry the solution to its local optimum
enddo

Step 3. (**Trail update**)
for each ant's move (φ,ψ) *do*
Compute $\Delta\tau_{\varphi\psi}$
Update the trail matrix by means of (4)
enddo

Step 4. (**Terminating condition**)
if not (end-test) go to step 2

Figure 2. Pseudo code for the ANTS algorithm.

5 The Set Partitioning problem

The Set Partitioning problem (SP) is the problem of partitioning a given set of elements into mutually independent subsets, chosen from a given collection, minimizing a cost function defined as the sum of the costs associated to each of the eligible subsets. Its importance derives from the fact that many actual situations can be modeled as SP. For example, many combinatorial optimization problems (crew scheduling, vehicle routing, project scheduling, warehouse location, etc) can be modeled as SP with maybe some additional constraints.

A specific problem that needs to solve SP instances arises in the context of logical design of Data Warehouses (DW), namely the Vertical Fragmentation Problem (VFP). DWs are foremost systems for improving the support given to company internal decision processes and data analysis procedures. A DW enables executives to retrieve summary data, derived by those present in operational information systems. An essential feature of a successful DW is a fast query response. DW design follows three main phases: i) conceptual, ii) logical, and iii) physical design. VFP is a

part of logical design and has the objective of minimizing the query response time by reducing the number of disk pages to be accessed. VFP problem details are rather intricate, and we refer the interested reader to Golfarelli *et al.* (2002) and Maniezzo *et al.* (2001) for problem details. While the specific formulation elements are not important for this paper, we point out that a substantial number of constraints are actually SP constraints; thus it is possible to solve problem VFP only if an effective means for solving SP is available.

The Set Partitioning problem can be modeled as follows. Let x_j, $j = 1,...,n$ be a binary variable denoting whether or not the j^{th} subset is part of the solution. A cost c_j is associated to each subset j. Let $A = [a_{ij}]$, with $i = 1,...,m$ and $j = 1,...,n$, be a $\{0,1\}$ coefficient matrix whose columns correspond to the subset and whose rows to the set elements. If $a_{ij} = 1$ the i^{th} element is a member of the j^{th} subset.

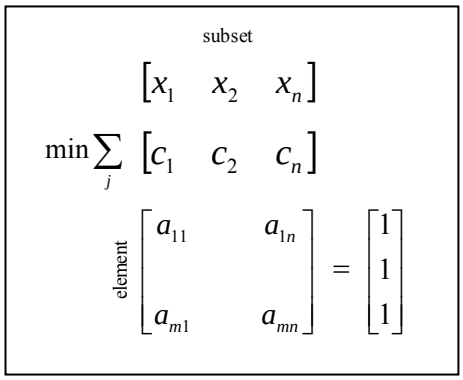

Figure 3. Matrix representation of the SP problem.

A mathematical formulation of SP is the following:

$$\text{(SP)} \qquad min \quad \sum_{j=1}^{n} c_j x_j \qquad (5.1)$$

$$s.t. \quad \sum_{j=1}^{n} a_{ij} x_j = 1; \qquad \forall i = 1,\ldots,m \qquad (5.2)$$

$$x_j \in \{0,1\}; \qquad \forall j = 1,\ldots,n \qquad (5.3)$$

By applying a Lagrangean relaxation of the constraints (5.2) we obtain the following problem:

$$\text{(LSP)} \qquad min \quad \sum_{j=1}^{n} \left(c_j x_j - \sum_{i=1}^{m} \lambda_i \alpha_{ij} x_j \right) + \sum_{i=1}^{m} \lambda_i \qquad (6)$$

$$s.t. \quad x_j \in \{0,1\}; \qquad j = 1,\ldots,n$$

Now, the bound to the original SP problem can be obtained by posing:

$$x_j = \begin{cases} 1 & if\ c_j - \sum_{i=1}^{m} \lambda_i \alpha_{ij} \leq 0 \\ 0 & otherwise \end{cases} \qquad (7)$$

The value of the solution found is a lower bound for the original SP. By associating dual variables α_i to each constraint (5.2) and relaxing constraints (5.3), the dual problem DSP becomes:

$$\text{(DSP)} \qquad max \quad \sum_i \alpha_i \qquad (8)$$

$$s.t. \quad \sum_i a_{ij} \alpha_i \leq c_j \qquad j = 1,\ldots,n$$

$$\alpha_i unbounded \qquad i = 1,\ldots,m$$

The Linear programming relaxation of the SP formulation usually gives a good lower bound to the problem. It is worth recalling that for the SP problem, efficient polynomial neighborhoods contain few solutions, and local search is of very limited use. Indeed, even finding a feasible solution is extremely difficult.

In the last years, the SP problem has been studied extensively because of its important applications. A detailed survey of SP applications and solution methods can be found in Balas and Padberg (1976). For instance, Marsten and Shepardson (1981) and Hoffman and Padberg (1993) have applied very successfully SP-based algorithms to airline crew scheduling problems. In both cases, an LP based branch and bound algorithm has been used to solve large-scale instances of the SP problem. To the same aim, Fisher and Kedia (1990) presented a dual ascent algorithm, Chu and Beasley (1995) introduced a genetic algorithm while Wedelin (1995) exploited a Lagrangean dual approach with cost perturbations. Atamturk *et al.* (1995) proposed other effective heuristics.

5.1 ACO approaches to SP

The first implementation of an ACO algorithm for the SP was proposed by Maniezzo and Milandri (2002), called BE-ANTS. They noticed that a direct implementation of the basic ACO framework is incapable of obtaining feasible solutions for many standard test set instances. The approach advocated by that paper included in a standard ANTS algorithm elements taken from bounded

enumeration procedures. They are tree search procedures where the number of nodes, which can be expanded at each tree level, is limited from above by a parametric constraint *k*. Greater value for *k* entails a bigger search space to explore, where *k* unlimited makes the algorithm become a complete enumeration procedure. In the case of SP, each level of the search tree is associated with one constraint. Level 0 corresponds to empty solution; level 1 considers the first constraint, and so on. The order according to which constraints are considered in successive tree expansions greatly affects the algorithm performance. As the presented paper follows this approach, it will be explained in details in the next section.

Recently, an enhancement of the original AS and ACO algorithms has been proposed by Crawford and Castro (2005) for solving the SP problem. The authors presented a hybrid technique that merges AS and ACO with Forward Checking (FC). Instead of performing arc consistency to the instantiated variables, FC performs a restricted form of it to the not yet instantiated variables. This reduces the search tree and the overall amount of work done. However, they noted that FC does more work when each assignment is added to the current partial solution. Forward Checking allows selecting columns if they do not produce any conflict with respect to the next column to be chosen. However, FC checks only the constraints between the current variable and the future variables. To overcome this problem, the authors introduced a full arc consistency, called Full LookAhead (FLA), which further reduces the domains and removes possible conflicts. The main advantage of FLA in respect of FC is that it detects also the conflicts among future variables and therefore allows branches of the search tree that will lead to failure to be pruned earlier. Regarding the SP problem, ACO with FLA techniques experimentally showed interesting performances demonstrating that the hybridization improves the search process, mostly with respect to success costs instead running times.

6 ANTS approaches to SP

The algorithm presented in this paper, called SP-ANTS, builds on the experience of the BE-ANTS heuristic for SP presented in Maniezzo and Milandri (2002).

The SP-ANTS algorithm and the standard ACO algorithm, differs mainly for the following issues:

i) the trail laying policy;

ii) the synchronization step among ants after each expansion.

Trail is not leaved directly on the components, which build up a solution (the subsets), but on the couplings (component/element), that is equivalent to the couple (variable/constraint). A high trail

value τ_{ij} indicates that a particular variable j demonstrated to be a good choice for covering constraint i. Thus the same variable j may have different attractiveness, depending on the particular choice (which variable to use for covering constraint i) an ant has to take.

A *synchronization step* is implemented after each ant expands its partial solution, in order to define the different possible further expansions.

Essentially, the number of ants is equal to the value of parameter *k*. At each level, one ant ant_i is assigned to each of the *k* branches to be expanded and computes its possible expansion set E_i. Then all expansion sets are united, ordered by non-decreasing cost, and the *k* of them, if so many exists, are chosen for further expansion.

Three cases make a branch non-eligible for further expansion:

i. The partial solution associated with the branch is *infeasible*.
ii. The partial solution associated with the branch has a *cost already greater* than that of the best one already found, thus it cannot lead to an improvement of the best solution found.
iii. The partial solution associated with the branch is *not among the k ones* chosen for the expansion of the level.

We recall that a branch, which can be expanded, is a valid branch. The SP-ANTS algorithm for SP problem is as follows:

SP-ANTS algorithm

Step (**Initialization**)

1. i=1. The first constraint to be covered is chosen. The set E of feasible expansions is initialized with all feasible expansions of the empty solution.

Step (**Construction**)

2. The value $\alpha\tau_{ij} + (1-\alpha)\eta_i$ is computed for each $j \in E$, where α, τ and η have the usual ACO interpretation. The partial solutions are accordingly ordered.

3. *k* valid branches in E are selected, by means of the usual ant probability selection formula (1), and one ant is assigned to each of them.

4. i=i+1. Each ant *j* decides the constraint (element) to cover at level *i* and accordingly computes the expansions E_j of its current partial solution. $E = \bigcup_j E_j$
5. *if* i < m goto Step 2.

Step (**Trail update** and **Terminating condition**)

6. *if* (end condition) *then* Stop *else* update trails, goto Step 1.

6b. *if* (end condition) *then* Stop *else* update trails, perform one subgradient iteration, goto Step 1.

Figure 4. Pseudo code for the SP-ANTS algorithm.

In order to specify completely the algorithm, it is necessary to define how to compute the η and the τ values. These are problem-specific elements, which in the case of the SP we implemented as follows.

The desirability η_i of a column *j* can be set equal either to the sum of the dual variable values of the still uncovered constraints which are covered by column *j* or simply to the number of still uncovered constraints which are covered by column *j* (we tested both possibilities). Thus, columns covering the most difficult (or the greater number) of uncovered constraints are preferred. Trails are updated by means of the formula introduced in Maniezzo (1999). Notice however that trails are laid on the coupling (*i*,*j*), that is, we explicitly increase or decrease trails stating how proficient it was to cover constraint *i* with variable *j*.

As a last comment to the implementation of SP-ANTS it is worth saying that the lower bound to the SP was computed by means of the Lagrangean relaxation of the problem and that SP-ANTS was intertwined with a subgradient optimization routine, so that Step 6b is chosen.

6.1 SP-ANTS at work

In Table 2, we propose as an example a simple SP instance while Figure 5 shows a trace of the execution of our code on this case. Figure 6 depicts a graphic representation of the tree search on the same instance.

Table 2. Example SP instance (first line: costs; first column: indices, other cells: constraint matrix).

row index/ cost	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
0	1	1	1	0	0	0	0	0	0	0	0
1	0	1	1	1	0	0	0	0	0	0	0
2	0	1	1	0	0	0	0	0	0	0	0
3	0	0	1	1	0	0	1	0	0	0	0
4	0	0	0	0	1	1	1	0	0	0	0
5	0	0	0	0	1	0	0	1	0	0	1
6	0	0	0	0	0	0	0	1	1	1	0

Table 3. Step 1 of the SP-ANTS algorithm.

row index/ cost	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
0	1	1	1	0	0	0	0	0	0	0	0
1	0	1	1	1	0	0	0	0	0	0	0
2	0	1	1	0	0	0	0	0	0	0	0
3	0	0	1	1	0	0	1	0	0	0	0
4	0	0	0	0	1	1	1	0	0	0	0
5	0	0	0	0	1	0	0	1	0	0	1
6	0	0	0	0	0	0	0	1	1	1	0

SP-ANTS starts choosing first the row 2 (see box in Table 3), as it has the fewest 1s (*i.e.* 2). We see that columns 1 and 2 can cover it (see grey marks in Table 3) constituting the set E of feasible expansions. According to the maximal number of branches *k* (fixed to 2), the attractiveness function controls the choice of the branches in a probabilistic way.

Table 4. Step 2 of the SP-ANTS algorithm.

row index/ cost	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
0	1	1	1	0	0	0	0	0	0	0	0
1	0	1	1	1	0	0	0	0	0	0	0
2	0	1	1	0	0	0	0	0	0	0	0
3	0	0	1	1	0	0	1	0	0	0	0
4	0	0	0	0	1	1	1	0	0	0	0
5	0	0	0	0	1	0	0	1	0	0	1
6	0	0	0	0	0	0	0	1	1	1	0

If the ant chooses column 1, the next row to face is row 3, as it can be covered by the smallest number of columns. In fact, columns 2, 3 and 6 can cover row 3 but columns 2 and 3 intersect (see cells (1,2) and (1,3) in Table 4) with column 1 thus resulting non-eligible for further expansion. Therefore, only column 6 is available.

If the ant chooses column 2, the next row to cover is row 4 (see box in Table 4), which can be covered by two columns (*i.e.* columns 4 and 5).

This collectively gives rise to three possible expansions ($1 \rightarrow 6, 2 \rightarrow 4, 2 \rightarrow 5$), but only two are further explored since *k* is fixed to 2.

Table 5. Step 3 of the SP-ANTS algorithm.

row index/ cost	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
0	1	1	1	0	0	0	0	0	0	0	0
1	0	1	1	1	0	0	0	0	0	0	0
2	0	1	1	0	0	0	0	0	0	0	0
3	0	0	1	1	0	0	1	0	0	0	0
4	0	0	0	0	1	1	1	0	0	0	0
5	0	0	0	0	1	0	0	1	0	0	1
6	0	0	0	0	0	0	0	1	1	1	0

Assuming that the least-cost ones are chosen (*i.e.* $2 \to 4, 2 \to 5$), being the stochastic case straightforward but cumbersome to describe, the remaining of the process is clear. As soon as a branch covers all rows, it is not further expanded (see Table 5 the final result).

```
NumCols:11 NumRows:7 NumElem:20

First row: 2
Elem:1 Cost:1
Elem:2 Cost:2
*Branch:0 NextRow:3 TotCost:1 \----> 1
*Branch:1 NextRow:4 TotCost:2 \----> 2

Next level
Elem:4 Cost:6 Branch:1
Elem:6 Cost:7 Branch:0
Elem:5 Cost:7 Branch:1
*Branch:0 Next Row:6 TotCost:6 \----> 4 2
*Branch:1 Next Row:5 TotCost:7 \----> 5 2

Next level
Elem:7 Cost:14 Branch:1
Elem:8 Cost:14 Branch:0
Elem:9 Cost:15 Branch:0
Elem:10 Cost:17 Branch:1
*Branch:0 Next Row:null TotCost:14 \----> 8 4 2
*Branch:1 Next Row:null TotCost:14 \----> 7 5 2
```

Figure 5. Trace of the execution of our code on the instance.

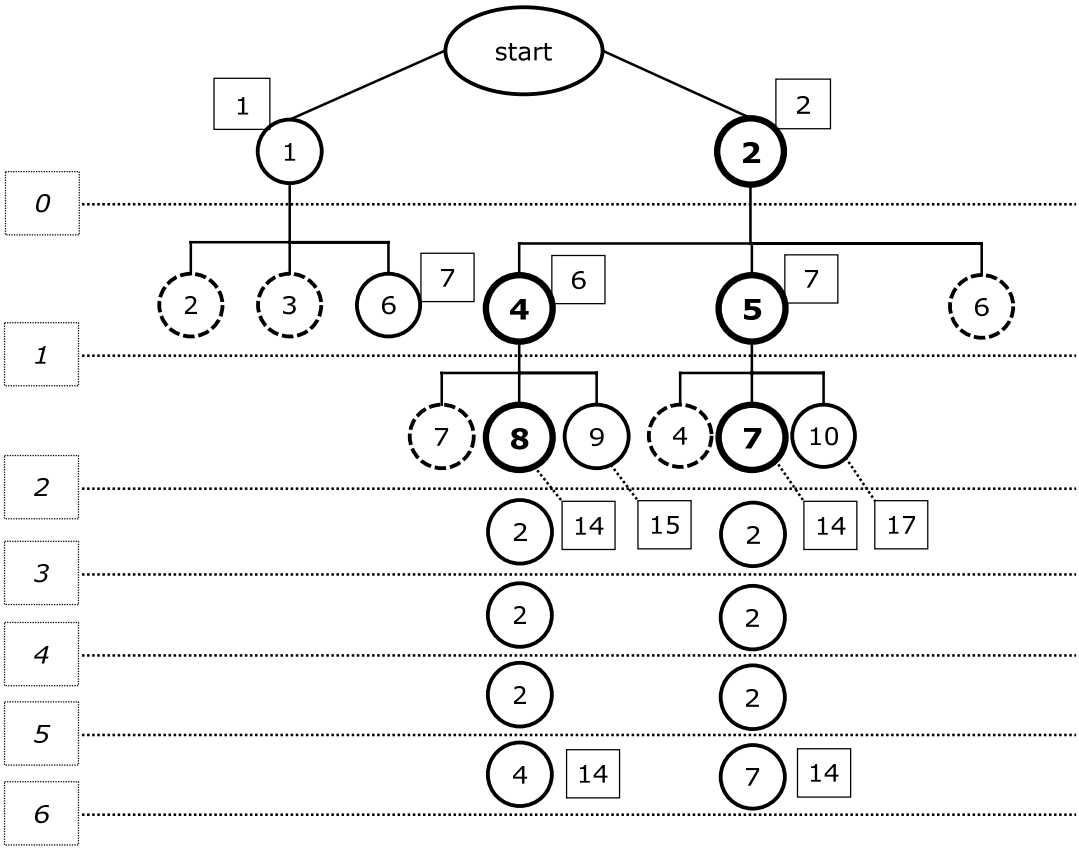

Figure 6. Graphic representation of the tree search. Dashed circles are not admissible nodes. Bold circles are expanded nodes. Numbers in the rectangles are the cost for the relative node. Rectangles on the left count the iterations.

7 Results

The algorithms described in the previous sections were implemented in Visual C++ .NET2005 and run on a 1.86 Pentium Centrino machine with 1 GB of memory. Tests were made on a common benchmark set, downloaded from OR-LIB, which was used in Chu and Beasley (1998) for validating the genetic algorithm. We selected a representative subset of difficult problems.

The results obtained on SP instances are presented in Table 6. Columns represent problem identifier (Problem), optimal solution by relaxation cost of Linear Programming (LP opt), optimal solution by Integer Programming (IP opt), ACO best solution (out of at most three runs) obtained by ANTS (ANTS), mean ANTS CPU time (seconds) to find the solution (CPU sec), and best solution obtained by the GA of Beasley and Chu (1999) (BCGA).

Table 6. Computational results on SP instances.

Problem	LP opt	IP opt	ANTS	CPU sec.	BCGA
NW01	114852.0	114852	n.f.	-	n.f.
NW02	105444.0	105444	115929	1747,81	108816
NW03	24447.0	24492	24492	271,58	24492
NW04	16310.7	16862	16978	295,32	16862
NW06	7640.0	7810	7810	7,30	7810
NW17	10875.7	11115	11133	682,86	11115
NW18	338864.3	340160	349958	558,66	345130
AA02	30494.0	30494	36812	981,13	30500
AA04	25877.6	26402	33231	1167,47	28261
KL01	1084.0	1086	1096	187,88	1086
KL02	215.3	219	229	410,62	219

As a comment to Table 6, on the bad side we may notice that the performance of ANTS is still inferior to that of BCGA, though not much so. On the good side, we must notice that standard ACO (which also we implemented) is not capable of finding a feasible solution for any of listed instances, whereas ANTS already has a performance comparable with that of state of the art solution methods. Moreover, as mentioned, the computational results are still preliminary and we expect that, with a correct parameter setting and with a reasonable number of test repetitions, ANTS results will further improve.

8 Conclusions

This paper presented a new algorithm, named ANTS, designed for solving any combinatorial optimization problem in general, and very hard, tightly constrained instances in particular. The methodology includes in a standard ACO framework ideas taken from bounded enumeration search. Ants loose their identity, as at each step a partial solution is assigned to each ant, but the resulting framework can be applied also to problems for which the standard ACO framework is very ineffective. The computational results on instance of the very constrained Set Partitioning problem, albeit very incomplete, testify the viability of the approach.

9 Bibliography

1. Atamturk A., Nemhauser G.L., and Savelsbergh M.W.P.. *A combined lagrangian, linear programming and implication heuristic for large scale set partitioning problems*. Journal of Heuristics, 1:247–259, 1995.
2. Balas E. and Padberg M.. *Set partitioning: a survey*, SIAM Review, 18:710-760, 1976.
3. Bauer, A., Bullnheimer, B., Hartl, R. F. and Strauss, C. Minimizing total tardiness on a single machine using ant colony optimization. Central European Journal for Operations Research and Economics, 8(2), 125-141, 2000.
4. Blum, C. An ant colony optimization algorithm to tackle shop scheduling problems. Technical report TR/IRIDIA/2003-1. IRID1A, Universite Libre de Bruxelles, Brussels, 2003.
5. Blum, C. *Theoretical and practical aspects of ant colony optimization*, Vol. 282 of Dissertations in Artificial Intelligence, IOS Press, Nieuwe Hemweg, The Netherlands, 2004.
6. Blum, C. and Dorigo, M. The hyper-cube framework for ant colony optimization, IEEE Transactions on Systems, Man and Cybernetics–Part B 34(2): 1161–1172, 2003.
7. Blum, C., Roli, A. and Dorigo, M. HC-ACO: The hyper-cube framework for ant colony optimization, 4th Metaheuristics International Conference, MIC'2001, Porto, Portugal, pp. 399–403, 2001.
8. Blum, C., and Blesa, M. J. Metaheuristics for the edge-weighted k-cardinality tree problem. Technical report LSI-03-1-R, Departament de Llenguatges i Sistemes Informatics, Universitat Politecnica de Catalunya, Barcelona, Spain, 2003.
9. Bullnheimer, B., Hartl, R. F., and Strauss, C. A new rank-based version of the Ant System: A computational study. Central European Journal for Operations Research and Economics, 7(1), 25-38, 1999a.
10. Bullnheimer, B., Hartl, R. F., and Strauss, C. Applying the Ant System to the vehicle routing problem. In S. Voss, S. Martell°, I. H. Osman. and C. Roucairol (Eds.), Meta-Heuristics: Advances and Trends in Local Search Pamdifons for Optimization (pp. 285-296). Dordrecht, Netherlands, Kluwer Academic Publishers, 1999b.
11. Cerny, V. *A thermodynamical approach to the traveling salesman problem*. Journal of Optimization Theory and Applications, 45(1), 41-51, 1985.
12. Chu P.C. and Beasley J.E. *A genetic algorithm for the set partitioning problem*. Technical report, The Management School, Imperial College, London, England, April, 1995

13. Chu P.C. and Beasley J.E. *Constraint handling in genetic algorithms: the set partitoning problem*. Journal of Heuristics, 4:323–357, 1998.
14. Colorni A., Dorigo M., and Maniezzo V., Distributed optimization by ant colonies, Proceedings of ECAL'91, European Conference on Artificial Life, Elsevier Publishing, 1991.
15. Colorni, A., Dorigo, M., Maniezzo, V., and Trubian, M. Ant System for job-shop schedullin. JORBEL-Belgian Journal of Operations Research, Statistics and Computer Science, 34(1), 39 53, 1994.
16. Cordone, R., and Maffioli, F. Coloured Ant System and local search to design local telecommuni¬cation networks. In E. J. W. Boers, J. Gottlieb, P. L. Lanzi, R. E. Smith. S. Cagnoni, E. Hart, G. R. Raidl, and H. Tijink (Eds.), Applications of Evolutionary Computing: Proceedings of Eva Workshops 2001, vol. 2037 of Lecture Notes in Computer Science (pp. 60-69). Berlin, Springer-Verlag, 2001.
17. Costa, D., and Hertz, A. Ants can colour graphs. Journal of the Operational Research Society, 48, 295-305, 1997.
18. Crawford B. and Castro C. Combination of metaheuristic and local search with Constraint Programming techniques conference, University of Nantes, Nantes – France November 28-29, 2005
19. den Besten, M. L., Stützle, T., and Dorigo, M. (2000). Ant colony optimization for the total weighted tardiness problem. In M. Schoenauer, K. Deb, G. Rudolph, X. Yao, E. Limon, J. J. Merelo, and H.-P. Schwefel (Eds.), Proceedings of PPSN-VI, Sixth International Conference on Parallel Problem Solving from Nature, vol. 1917 of Lecture Notes in Computer Science (pp. 611-620). Berlin, Springer-Verlag.
20. Davis D., Poli R., Balakrishnan K, Honavar V., Rudolph G, Wegener J., Bull L., Potter M., Schultz A., Miller J., Burke E., and Jonoska N. (Eds.), Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2002) (pp. 1317-1325). San Francisco, Morgan Kaufmann, 2002.
21. Delisle P., Gravel M., Krajecki M., Gagné C.,and Price W.L. Comparing Parallelization of an ACO: Message Passing vs. Shared Memory. Lecture Notes in Computer Science, Vol 3636/2005,1-11, 2005
22. Dorigo M., Optimization, learning and natural algorithms, Ph.D. Thesis, Politecnico di Milano, Milano, 1992.
23. Dorigo M., Colorni A., and Maniezzo V., Positive feedback as a search strategy, Technical Report TR91-016, Politecnico di Milano, 1991.

24. Dorigo M., G. Di Caro. The Ant Colony Optimization Meta-Heuristic. In D. Corne, M. Dorigo and F. Glover, editors, New Ideas in Optimization, McGraw-Hill, 11-32, 1999.
25. Dorigo M., Maniezzo V., and Colorni, A. Ant System: Optimization by a colony of cooperating agents. IEEE Transactions on Systems, Man, and Cybernetics-Part 8, 26(1), 29-41, 1996.
26. Dorigo M., and Gambardella L. M. Ant colonies for the traveling salesman problem. BioSystems, 43(2), 73-81, 1997a.
27. Dorigo M., and Gambardella L. M. Ant Colony System: A cooperative learning approach to the traveling salesman problem. IEEE Transactions on Evolutionary Computation, 53-66, 1997b.
28. Dorigo M. and Stützle T. Ant Colony Optimization. The MIT Press, 2004.
29. Ellabib I., Calamai P.,and Basir O. Exchange strategies for multiple Ant Colony System. Information Sciences, Vol. 177, Issue 5, 2007.
30. Fenet, S., and Solnon, C. Searching for maximum cliques with ant colony optimization. In G. R. Raidl, J.-A. Meyer, M. Middendorf, S. Cagnoni, J. J. R. Cardalda, D. W. Come, J. Gottlieb, A. Guillot, E. Hart, C. G. Johnson, and E. Marchiori (Eds.), Applications of Evolutionary Computing, Proceedings of EvoWorkshops 2003, vol. 2611 of Lecture Notes in Computer Science (pp. 236-245). Berlin, Springer-Verlag, 2003.
31. Feo, T. A., and Resende, M. G. C. Greedy randomized adaptive search procedures. Journal of Global Optimization, 6, 109-133, 1995.
32. Fogel, L. J., Owens, A. J., and Walsh, M. J. Artificial Intelligence through Simulated Evolution. New York, John Wiley and Sons, 1966.
33. Fisher M.L., Kedia P., *Optimal solution for set covering/partitioning problems using dual heuristics*. Man.Sci., 36, 674-688, 1990
34. Gambardella L.M. and Dorigo M., Ant-q: a reinforcement learning approach to the traveling salesman problem, Proceedings of the Twelfth International Conference on Machine Learning, ML-95, Palo Alto, CA, Morgan Kaufmann, 1995.
35. Gambardella, L. M., and Dorigo, M. HAS-SOP: An hybrid Ant System for the sequential ordering problem. Technical report IDSIA-11-97, IDSIA, Lugano, Switzerland, 1997.
36. Gambardella, L. M., and Dorigo, M. Ant Colony System hybridized with a new local search for the sequential ordering problem. INFORMS Journal on Computing, 12(3), 237-255, 2000.
37. Gambardella, L. M., Taillard, E. D., and Dorigo, M. Ant colonies for the quadratic assignment problem. Journal of the Operational Research Society, 50(2), 167-176, 1999.

38. Gambardella, L. M., Taillard, E. D., and Agazzi, G. MACS-VRPTW: A multiple ant colony system for vehicle routing problems with time windows. In D. Come, M. Dorigo, and F. Glover (Eds.), New Ideas in Optimization (pp. 63-76). London, McGraw Hill, 1999.
39. Glover, F. Heuristics for integer programming using surrogate constraints. Decision Sciences, 8, 156-166, 1977.
40. Glover, F.. Tabu search-Part I. ORSA Journal on Computing, 1(3), 190-206, 1989.
41. Glover, F. Tabu search-Part II. ORSA Journal on Computing, 2(1), 4-32, 1990.
42. Golfarelli M., Maniezzo V., and Rizzi S. *Materialization of Fragmented Views in Multidimensional Databases.* Technical Report TR-001-02, Scienze dell'Informazione, University of Bologna, 2002.
43. Hadji, R.. Rahoual, M., Talbi, E., and Bachelet, V. Ant colonies for the set covering problem. In M. Dorigo, M. Middendorf, and T. Stfitzle (Eds.), Abstract Proceedings of ANTS 2000-From Ant Colonies to Artificial Ants: Second International Workshop on Ant Algorithms (pp. 63-66). Brussels, Universite Libre de Bruxelles, 2000
44. Hoffman K. and Padberg M., *Solving airline crew scheduling problems by branch and cut*, Management Science 39(6):667-682, 1993.
45. Leguizamon, G., and Michalewicz, Z. A new version of Ant System for subset problems. In P. J. Angeline, Z. Michalewicz, M. Schoenauer, X. Yao, and A. Zalzala (Eds.), Proceedings of the 1999 Congress on Evolutionary Computation (CEC'99) (pp. 1459-1464). Piscataway, NJ, IEEE Press, 1999.
46. Leguizameon, G., Michalewicz, Z., and Schutz, M. (2001). A ant system for the maximum independent set problem. In Proceedings of the VII Argentinian Congress on Computer Science, El Calafate, Santa Cruz, Argentina, vol. 2 (pp. 1027-1040), 2001
47. Lourenco, H. R., Martin, O., and Stützle, T. Iterated local search. In F. Glover and G. Kochcnberger (Eds.), Handbook of Metaheuristics, vol. 57 of International Series in Operations Research and Management Science (pp. 321-353). Norwell, MA, Kluwer Academic Publishers, 2002.
48. Lourenco, H., and Serra, D. Adaptive approach heuristics for the generalized assignment problem. Technical report No. 304, Universitat Pompeu Fabra, Department of Economics and Management, Barcelona, Spain, 1998.
49. Maniezzo V.. *Exact and approximate nondeterministic tree-search procedures for the quadratic assignment problem*. INFORMS Journal on Computing, 11(4):358–369, 1999.
50. Maniezzo V., Carbonaro A., Golfarelli M., and Rizzi S.. *An ANTS Algorithm for Optimizing the Materialization of Fragmented Views in Data Warehouses: Preliminary Results.* In

Applications of Evolutionary Computing, volume 2037 of Lecture Notes in Computer Science, pages 80–89. Springer Verlag, 2001.

51. Maniezzo V. and Milandri M.. *An Ant-Based Framework for Very Strongly Constrained Problems*. In M. D. et al., editor, ANTS 2002, vol 2463 of LNCS, pp 222–227. SV, 2002.
52. Maniezzo, V., and Carbonaro, A. An ANTS heuristic for the frequency assignment problem. Future Generation Computer Systems, 16(8), 927-935, 2000.
53. Maniezzo, V., Colorni, A., and Dorigo, M. (1994). The Ant System applied to the quadratic assignment problem. Technical report IRIDIA/94-28, IRIDIA, Universite Libre de Bruxelles, Brussels.
54. Maniezzo, V., and Colorni, A. The Ant System applied to the quadratic assignment problem. IEEE Transactions on Data and Knowledge Engineering, 11(5), 769-778, 1999.
55. Marsten R.E. and Shepardson F., *Exact solution of crew problems using the set partitioning mode*, Recent successful applications, Networks, 11:165-177, 1981.
56. Merkle, D., and Middendorf, M.. Ant colony optimization with global pheromone evaluation for scheduling a single machine. Applied Intelligence, 18(l), 105-111, 2003.
57. Merkle, D., Middendorf, M., and Schmeck, H. Ant colony optimization for resource-constrained project scheduling. In D. Whitley, D. Goldberg, E. Cantu-Paz, L. Spector, I. Parmee, and H.-G. Beyer (Eds.), Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2000) (pp. 893¬900). San Francisco, Morgan Kaufmann, 2002.
58. Ouyang J, Yan G., A multi-group ant colony system algorithm for TSP. Proceedings of International Conference on Machine Learning and Cybernetics, Vol. 1, pp. 117-121, 2004.
59. Reimann, M., Stummer, M., and Doerner. K. A savings based Ant System for the vehicle routing problem. In W. B. Langdon, E. Canto-Paz, K. Mathias, R. Roy
60. Pfahringer, B. Multi-agent search for open shop scheduling: Adapting the Ant-Q formalism. Technical report TR-96-09, Austrian Research Institute for Artificial Intelligence, Vienna, 1996.
61. Sahni, S., and Gonzalez, T. P-complete approximation problems. Journal of the ACM, 23(3), 555-565, 1976.
62. Socha, K., Knowles, J., and Sampels, M. A MAX-.A41.1\1 Ant System for the university course timetabling problem. In M. Dorigo, G. Di Caro, and M. Sampels (Eds.), Proceedings of ANTS 2002-From Ant Colonies to Artificial Ants: Third International Workshop on Ant Algorithms, vol. 2463 of Lecture Notes in Computer Science (pp. 1-13). Berlin, Springer-Verlag, 2003.

63. Socha, K., Sampels, M., and Manfrin, M. Ant algorithms for the university course timetabling problem with regard to the state-of-the-art. In G. R. Raidl, J.-A. Meyer, M. Middendorf, S. Cagnoni, J. J. R. Cardalda, D. W. Come, J. Gottlieb, A. Guillot, E. Hart, C. G. Johnson, and E. Marchiori (Eds.), Applications of Evolutionary Computing, Proceedings of Eva Workshops 2003, vol. 2611 of Lecture Notes in Computer Science (pp. 334-345). Berlin, Springer-Verlag, 2003.
64. Stützle, T. Local Search Algorithms for Combinatorial Problems: Analysis, Improvements, and New Applications, vol. 220 of DISKI. Sankt Augustin, Germany, Infix, 1999.
65. Stützle, T. MAX-MIN. Ant System for the quadratic assignment problem. Technical report AIDA-97-4, FG Intellektik, FB Informatik, TU Darmstadt, Germany, 1997b.
66. Stützle, T. An ant approach to the flow shop problem. In Proceedings of the Sixth European Congress on Intelligent Techniques and Soli Computing (EUFIT'98), vol. 3 (pp. 1560-1564). Aachen, Ger¬many, Verlag Mainz, Wissenschaftsverlag, 1998.
67. Stützle, T. and Hoos H. The MAX-MIN Ant System and local search for the traveling salesman problem. In T. Back, Z. Michidewicz, and X. Yao (Eds.), Proceedings q. the 1997 IEEE International Conference on Evolutionary Computation (ICEC'97) (pp. 309-314). Piscataway, NJ, IEEE Press, 1997.
68. Stützle, T. and Hoos, H. H. MAX-.MLAT Ant System. Future Generation Computer Systems, 16(8), 889-914, 2000.
69. Toth, P., and Vigo, D. (Eds.). The Vehicle Routing Problem. SIAM Monographs on Discrete Mathematics and Applications. Philadelphia, Society for Industrial and Applied Mathematics, 2001.
70. Voudouris, C., and Tsang, E. Guided local search. Technical report CSM-247, Department of Computer Science, University of Essex, Colchester, UK, 1995.
71. Wedelin D., *An algorithm for large scale 0-1 integer programming with application to airline crew scheduling*. Annals of Operations Research 57:283-301, 1995.
72. Wiesemann W., and Stützle T. Iterated Ants: An Experimental Study for the Quadratic Assignment Problem. ANTS Workshop 2006: 179-190, 2006.