An Ant Colony Optimization Meta-Heuristic for Subset Selection Problems

Christine Solnon, Derek Bridge

Chapter I

An Ant Colony Optimization Meta-Heuristic for Subset Selection Problems

Christine Solnon [I.1] Derek Bridge [I.2]

Subset selection problems involve finding an optimal feasible subset of an initial set of objects with respect to an objective function and/or some constraints. Many well-known combinatorial problems are members of this class, e.g., maximum clique problems, knapsack problems, boolean satisfiability problems, constraint satisfaction problems, and graph matching problems.

In this chapter we define a generic ant colony optimization (ACO) algorithm for this class of problems. Basically, this algorithm successively generates subsets through the repeated selection of objects, and uses "pheromone trails" as a greedy heuristic to choose, at each step, the next object to be selected. The algorithm is parameterized by a pheromonal strategy and we propose and compare two different instantiations of it: a first one where pheromone is laid on objects and a second one where pheromone is laid on cliques of objects. The proposed algorithm is also parameterized by problem-specific features, and we present and evaluate instantiations of it for solving maximum clique problems, knapsack problems and constraint satisfaction problems.

[I.1] LIRIS CNRS UMR 5205, University Lyon 1, France, christine.solnon@liris.cnrs.fr, http://www710.univ-lyon1.fr/~csolnon

[I.2] Department of Computer Science, University College Cork, Ireland, d.bridge@cs.ucc.ie, www.cs.ucc.ie/dbridge.html

I. AN ACO META-HEURISTIC FOR SS PROBLEMS

I.1 Introduction

The Ant Colony Optimization (ACO) meta-heuristic is one of the best-known examples of swarm intelligence systems. This meta-heuristic has recently joined the arsenal of methods for finding solutions and near-solutions to intractable discrete optimization problems [9, 10, 11]. The problem to be solved is modelled as a search for a minimum cost path in a graph. Artificial ants walk the graph, each path corresponding to a potential solution to the problem. The behavior of the ants is inspired by that of real ants: they deposit pheromone on the path in a quantity proportional to the quality of the solution represented by that path; they resolve choices between competing destinations probabilistically, where the probabilities are proportional to the pheromone accumulated on previous iterations. This indirect form of communication, known as stigmergy, *intensifies* the search around the most promising parts of the search space. On the other hand, there is also a degree of pheromone evaporation, which allows some past history to be forgotten, to *diversify* the search to new, and hopefully more successful, areas of the search space. The trade-off between intensification and diversification is influenced by modifying the values of parameters.

Ant algorithms in general, and instantiations of the ACO meta-heuristic in particular, have been applied to many discrete optimization problems. Many of these problems are ordering or sequencing problems, which are easily modelled as best hamiltonian path-finding problems, where ants must visit every vertex in a graph. Examples include traveling salesperson problems [8], quadratic assignment problems [27], vehicle routing problems [6], and permutation constraint satisfaction problems [22]. To solve such best hamiltonian path-finding problems with the ACO meta-heuristic, a pheromone trail τ_{ij} is associated with each edge (i, j). This pheromone trail represents the learned desirability of visiting vertex j immediately after vertex i, and it is used to guide ants during their path construction step.

Many combinatorial problems, however, involve selection rather than ordering or sequencing. Given a set S, some subset $S' \subseteq S$ that satisfies certain properties and/or that optimizes some objective function is to be selected. We will refer to the class of such problems as Subset Selection problems —abbreviated by SS problems. Some well-known examples of SS problems are, e.g., the maximum clique problem, the multidimensional knapsack problem, the boolean satisfiablility problem, and the constraint satisfaction problem. For these problems, the *order* in which objects are selected is not significant. Therefore, it is meaningless to model them as path-finding problems and lay pheromone trails on consecutively visited vertices. Two main pheromonal strategies may be considered for solving SS problems with ACO:

- One may associate a pheromone trail τ_i with each object $i \in S$, so that τ_i represents the learned desirability of selecting object i. This pheromonal strategy has been proposed in [17] for the general subset selection problem, where it has been experimentally validated on the multiple knapsack problem. It also has been proposed for constraint satisfaction problems [30], maximum clique problems [25], and edge-weighted k-cardinality tree problems [5].

- One may associate a pheromone trail τ_{ij} with each pair of different objects $(i, j) \in S \times S$, so that τ_{ij} represents the learned desirability that objects i and j belong to the same subset. This pheromonal strategy has been proposed for different specific SS problems: maximum clique problems [12], multiple knapsack problems [2], constraint satisfaction problems [23], edge-weighted k-cardinality tree problems [5], and graph matching problems [21].

A goal of this chapter is to present these different algorithms in a unified framework. Hence, we introduce a generic ACO algorithm for SS problems. This algorithm is parameterized by problem-specific features and by a pheromonal strategy which determines whether ants lay pheromone on objects or on pairs of objects.

We introduce the class of SS problems and give example members of it in Section I.2. We define the generic Ant Colony Optimization algorithm for this class of problems in Section I.3. We describe instantiations of this algorithm with respect to two different pheromonal strategies in Section I.4, and we describe instantiations of this algorithm to solve well-known SS problems in Section I.5. We discuss parameter settings and their influence on the solution process in Section I.6. Finally in Section I.7, we present some experimental results.

I.2 Subset Selection Problems

Subset Selection problems (SS problems) involve finding an optimal feasible subset of objects within an initial set of objects. More formally, an SS problem may be defined by a triple $(S, S_{feasible}, f)$ where

- S is a set of objects;
- $S_{feasible} \subseteq \mathcal{P}(S)$ is a set that contains all feasible subsets of S;
- $f : S_{feasible} \rightarrow I\!R$ is an objective function that associates a real-valued cost $f(S')$ with every feasible subset of objects $S' \in S_{feasible}$.

The goal of an SS problem $(S, S_{feasible}, f)$ is to find $S^* \subseteq S$ such that $S^* \in S_{feasible}$ and $f(S^*)$ is maximal. Here are a few examples of SS problems.

Maximum Clique. The goal is to find a largest set of pairwise adjacent vertices in a graph [4], i.e.,

- S contains all the vertices of the graph;
- $S_{feasible}$ contains all the cliques of G, i.e., all the sets $S' \subseteq S$ such that every pair of distinct vertices in S' is connected by an edge in the graph;
- f is the cardinality function.

Multidimensional Knapsack. The goal is to find a subset of objects that maximizes a total profit while satisfying some resource constraints, i.e.,

- S is the set of objects;
- $S_{feasible}$ contains every subset that satisfies all the resource constraints, i.e., $S_{feasible} = \{S' \subseteq S \mid \forall i \in 1..m, \sum_{j \in S'} r_{ij} \leq b_i\}$ where m is the number of resources, r_{ij} is the consumption of resource i by object j, and b_i is the available quantity of resource i;
- f returns the total profit, i.e., $\forall S' \in S_{feasible}, f(S') = \sum_{j \in S'} p_j$ where p_j is the profit associated with object j.

Maximum Boolean Satisfiability. The goal is to find a truth assignment of boolean variables that satisfies a maximum number of boolean formulas, i.e.,

- S contains all boolean variables;
- $S_{feasible}$ contains all the subsets of S, i.e., $S_{feasible} = \mathcal{P}(S)$;
- f returns the number of satisfied formulas, i.e., $\forall S' \in S_{feasible}, f(S')$ is the number of boolean formulas that are satisfied when all variables of S' are set to True and all others to False.

The classical boolean satisfiability (SAT) problem is a special case of the Maximum Boolean Satisfiability where the goal is to find a truth assignment of boolean variables that satisfies *all* boolean formulas. This problem may be modelled as a subset selection problem by defining f as the function that returns 1 if all formulas are satisfied and 0 otherwise.

Maximum Constraint Satisfaction. The goal is to find an assignment of values to variables that satisfies a maximum number of constraints [31], i.e.,

- S contains every label $\langle X_i, v_i \rangle$ pairing a variable X_i with a value v_i from that variable's domain $D(X_i)$;
- $S_{feasible}$ contains all the subsets of S that do not contain two different labels for the same variable, i.e.,

$$S_{feasible} = \{S' \subseteq S \mid \forall (\langle X_i, v_i \rangle, \langle X_j, v_j \rangle) \in S' \times S', X_i = X_j \Rightarrow v_i = v_j\}$$

- f returns the number of satisfied constraints, i.e., $\forall S' \in S_{feasible}, f(S')$ is the number of constraints such that every variable involved in the constraint is assigned a value by a label of S' and the constraint is satisfied by this assignment of values to variables.

As with SAT problems, constraint satisfaction problems (CSPs) are special cases of Maximum Constraint Satisfaction and can easily be modelled as SS problems.

Minimum Vertex Cover. The goal is to find the smallest subset of the vertices of a graph G that contains at least one vertex of every edge of G. As this is a minimization problem, we may consider the dual problem that involves finding the largest set of vertices such that no edge has its two vertices in this set, i.e.,

- S contains all the vertices of the graph;
- $S_{feasible}$ contains all the subsets $S' \subseteq S$ such that for all edges (i, j) of the graph, $i \notin S'$ or $j \notin S'$;
- f is the cardinality function.

Graph Matching. Given two labelled graphs, the goal is to find a multivalent mapping of their vertices that maximizes their similarity [7], i.e.,

- S contains all pairs of vertices (i, j) such that i is a vertex of the first graph and j is a vertex of the second graph;
- $S_{feasible}$ contains all the subsets of S, i.e., $S_{feasible} = \mathcal{P}(S)$;
- f evaluates the similarity of the two graphs with respect to the mapping defined by a subset of S: for each subset $S' \subseteq S$ that defines a multivalent mapping —associating each vertex of one graph to a set of vertices of the other graph— $f(S')$ depends on (1) the number of features of each graph that are mapped to at least one similar feature of the other graph and (2) the number of vertices that are mapped to more than one vertex of the other graph.

Edge-Weighted k-Cardinality Tree. This problem is a generalization of the minimum weight spanning tree problem. Given a weighted graph, the goal is to find a minimum weight subtree with exactly k edges, i.e.,

- S contains all the vertices of the graph;
- $S_{feasible}$ contains all the subsets $S' \subseteq S$ such that $|S'| \leq k$ and S' is a tree;
- $f(S') = 0$ if $|S'| \neq k$, and $f(S') = -\sum_{(i,j)\in S'} weight(i, j)$ otherwise.

Numerous other problems are naturally modelled as SS problems. We should point out that, of course, many problems can be modelled in multiple ways. In particular, a problem might be modelled as either a SS problem or as the kind of ordering problem that we mentioned in Section I.1. It is well-known, for example, that traveling salesperson problems can be modelled as constraint satisfaction problems (e.g. variables represent cities, values are integers that represent the order in which the cities are visited) and hence can be alternatively solved as SS problems.

Algorithm I.1 The ACO meta-heuristic for SS problems

Input: A SS problem $(S, S_{feasible}, f)$
A pheromonal strategy $\phi \in \{Vertex, Clique\}$
A set of parameters $\{\alpha, \beta, \rho, \tau_{min}, \tau_{max}, nbAnts\}$
Output: A feasible subset of objects $S' \in S_{feasible}$

01. Let C be the set of pheromonal components w.r.t. the strategy ϕ
02. Initialize the pheromone trail $\tau(c)$ associated with each $c \in C$ to τ_{max}
03. **repeat**
04. **for** each ant $k \in [1; nbAnts]$, construct a solution S_k as follows:
05. Randomly choose a first object $o_i \in S$
06. $S_k \leftarrow \{o_i\}$
07. $Candidates \leftarrow \{o_j \in S \mid S_k \cup \{o_j\} \in S_{feasible}\}$
08. **while** $Candidates \neq \emptyset$ **do**
09. Choose an object $o_i \in Candidates$ with probability
10. $p(o_i, S_k) = \frac{[\tau_{factor}(o_i, S_k)]^{\alpha} \cdot [\eta_{factor}(o_i, S_k)]^{\beta}}{\sum_{o_j \in Candidates} [\tau_{factor}(o_j, S_k)]^{\alpha} \cdot [\eta_{factor}(o_j, S_k)]^{\beta}}$
11. $S_k \leftarrow S_k \cup \{o_i\}$
12. Remove o_i from $Candidates$
13. Remove from $Candidates$ every object o_j such that $S_k \cup \{o_j\} \notin S_{feasible}$
14. **end while**
15. **end for**
16. Optionally, apply local search to one or more solutions of $\{S_1, \ldots, S_{nbAnts}\}$
17. **for** each $c \in C$, update the pheromone trail $\tau(c)$ as follows:
18. $\tau(c) \leftarrow \tau(c) \cdot \rho + \delta_{\tau}(c, \{S_1, \ldots, S_{nbAnts}\})$
19. **if** $\tau(c) < \tau_{min}$ **then** $\tau(c) \leftarrow \tau_{min}$
20. **if** $\tau(c) > \tau_{max}$ **then** $\tau(c) \leftarrow \tau_{max}$
21. **end for**
22. **until** maximum number of cycles reached **or** acceptable solution found
23. **return** the best solution found since the beginning

I.3 A Generic ACO Algorithm for SS Problems

Algorithm I.1 describes a generic ACO algorithmic scheme for solving SS problems. We refer to this algorithmic scheme as Ant-SS. This algorithm is parameterized by:

- the actual SS problem to be solved, described by a triple $(S, S_{feasible}, f)$; instantiations of this algorithm for solving maximum clique, knapsack and constraint satisfaction problems are described in Section I.5.
- a pheromonal strategy ϕ which defines the set of pheromonal components on which ants lay pheromone trails; two different pheromonal strategies are described in Section I.4.
- a set of numeric parameters, the role and setting of which are discussed in Section I.6.

This algorithm first initializes all pheromone trails to τ_{max} and then iterates on a repeat-until loop (lines 03 to 22). Each iteration is composed of a solution construction step —where ants exploit pheromone trails to construct solutions— and a pheromone updating step —where pheromone trails are intensified with respect to these constructed solutions.

I.3.1 Solution construction step

Lines 05 to 14 of algorithm I.1 describe the procedure used by ants to construct subsets. The first object is chosen randomly; subsequent objects are chosen within the set *Candidates* —that, for a given ant, contains all feasible objects with respect to the objects the ant has chosen so far— using a probabilistic *state transition rule*. More precisely, the probability $p(o_i, S_k)$ of selecting $o_i \in Candidates$ when ant k has already selected the subset of objects S_k depends on two factors:

- The *pheromone factor* $\tau_{factor}(o_i, S_k)$ evaluates the learned desirability of adding object o_i to subset S_k based on the pheromone trails that have been deposited previously on pheromonal components. The definition of this factor depends on the pheromonal strategy ϕ as discussed in Section I.4.

- The *heuristic factor* $\eta_{factor}(o_i, S_k)$ evaluates the promise of object o_i based on information local to the ant, i.e., the solution it has built so far S_k. The definition of this factor is dependent on the problem $(S, S_{feasible}, f)$ as discussed in Section I.5.

As usual in ACO algorithms, α and β are two parameters that determine the relative importance of these two factors.

Note that such an incremental construction of solutions —where feasible objects are iteratively added to a solution that is initially empty— supposes that $S_{feasible}$ is defined in such a way that every feasible subsets can be incrementally constructed, i.e., for each non empty feasible subset $S' \in S_{feasible}$, there must exist at least one object $o_i \in S'$ such that $S' - \{o_i\}$ is also feasible. Let us consider for example the edge-weighted k-cardinality tree problem defined in Section I.2. For this problem, we have defined $S_{feasible}$ as the set of all trees having a number of vertices that is smaller or equal to k, so that we can incrementally build a tree with k vertices starting from the empty tree. Had we defined $S_{feasible}$ as the set of all trees having exactly k vertices, this would no longer have been possible.

Once each ant has constructed a feasible subset, one or more constructed solutions may be improved by using a problem-dependent form of local search (line 16): in some cases local search is applied to all constructed solutions; in other cases it is applied to the best solution of the cycle.

I.3.2 Pheromone updating step

Lines 17 to 21 of algorithm I.1 describe the updating of pheromone trails at the end of each cycle. Evaporation is simulated by multiplying the quantity of pheromone on each pheromonal component by a pheromone persistence rate ρ such that $0 \leq \rho \leq 1$.

Then a quantity δ_τ of pheromone is added. In the standard ACO meta-heuristic, an ant can deposit pheromone while constructing its solution, i.e., while walking, or after a solution has been built, or both [10]. However, we avail only of the option of depositing pheromone *after* ants have constructed their solutions. This is because, for SS problems, the quality of partial solutions is not necessarily a good estimate of the quality of complete solutions, which depends on problem-specific criteria some of which can only be evaluated on complete solutions (e.g. the size of S_k).

Also, in some of the very earliest ant algorithms (e.g. [8]), it was realised that not every ant should get to deposit pheromone. Accordingly, we allow for an *elitist strategy*, where only the best ants in each cycle, i.e., one or more of those whose solutions are no worse than any other ant's solution, deposit pheromone. The amount of pheromone deposited by these best ants depends on the quality of the constructed solution. In many cases, it is inversely proportional to the gap in quality between the constructed solution and the best constructed solution since the beginning of the run S_{best}, i.e., the amount of pheromone deposited on component c with respect to the constructed solutions $\{S_1, \ldots, S_{nbAnts}\}$ is defined as follows:

$$\begin{array}{lcll} \delta_\tau(c, \{S_1, \ldots, S_{nbAnts}\}) & = & \frac{1}{1+f(S_{\text{best}})-f(S_k)} & \text{if } \exists k \in [1; nbAnts] \text{ such that} \\ & & & c \text{ is a pheromonal component of } S_k \\ & & & \text{and } \forall i \in [1; nbAnts], f(S_k) \geq f(S_i) \\ \delta_\tau(c, \{S_1, \ldots, S_{nbAnts}\}) & = & 0 & \text{otherwise} \end{array}$$

The algorithm follows the $\mathcal{MAX}$-$\mathcal{MIN}$ Ant System [28]: we explicitly impose lower and upper bounds τ_{min} and τ_{max} on pheromone trails (with $0 < \tau_{min} < \tau_{max}$) (lines 19-20). These bounds restrict differences between the pheromone on components, which encourages wider exploration. All components are initialized to the maximum allowed τ_{max} (line 02). This makes all choices quite attractive in early cycles, keeping exploration high in these cycles. However, in Section I.6.2, we discuss an alternative way of initializing the trails based on a preprocessing step.

I.4 Instantiations of Ant-SS w.r.t. Pheromonal Strategies

The generic algorithm Ant-SS is parameterized by a pheromonal strategy ϕ and we now describe two instantiations of this algorithm: Ant-SS(Vertex) —where pheromone is laid on objects— and Ant-SS(Clique) —where pheromone is laid on pairs of objects. For each pheromonal strategy $\phi \in \{Vertex, Clique\}$, we have to define:

- the set C of pheromonal components associated with an SS problem $(S, S_{feasible}, f)$, i.e., the set of components on which ants may lay pheromone trails;
- the set of pheromonal components associated with a solution $S_k \in S_{feasible}$, i.e., the set of pheromonal components on which some pheromone is actually laid when solution S_k is rewarded;
- the pheromone factor $\tau_{factor}(o_i, S_k)$ associated with an object o_i and a partial solution S_k used in the probabilistic transition rule.

I.4. INSTANTIATIONS OF ANT-SS W.R.T. PHEROMONAL STRATEGIES

I.4.1 The "vertex" pheromonal strategy

In ACO algorithms, ants lay pheromone on components of the best constructed solutions in order to attract other ants towards the corresponding areas of the search space. For SS problems, solutions constructed by ants are subsets of objects, and the order in which objects are selected is not significant. Hence, a first pheromonal strategy consists in laying pheromone on objects of the best constructed solutions.

- The set C of pheromonal components contains all objects, i.e., $C = S$. Intuitively, the quantity of pheromone on every pheromonal component $o_i \in C$ —$\tau(o_i)$— represents the learned desirability of selecting o_i when constructing a solution.
- The set of pheromonal components associated with a solution S_k is the set of objects $o_i \in S_k$.
- The pheromone factor in the probabilistic transition rule corresponds to the quantity of pheromone on the considered object, i.e., $\tau_{factor}(o_i, S_k) = \tau(o_i)$.

I.4.2 The "clique" pheromonal strategy

The "vertex" pheromonal strategy implicitly supposes that an object's desirability is rather independent of the other selected objects. However, in some cases the desirability of an object may depend on the subset of already selected objects. Let us consider for example a constraint satisfaction problem that contains two variables x and y that can be assigned either 0 or 1, so that the initial set of objects S contains the labels $\langle x, 0\rangle$, $\langle x, 1\rangle$, $\langle y, 0\rangle$, and $\langle y, 1\rangle$. Let us now suppose that this problem contains the constraint $x \neq y$. In this case, the desirability of selecting either label $\langle y, 0\rangle$ or label $\langle y, 1\rangle$ for y depends on the label selected for x (and vice-versa). Hence, pheromone trails could be used to learn that $(\langle x, 0\rangle, \langle y, 1\rangle)$ and $(\langle x, 1\rangle, \langle y, 0\rangle)$ are pairs of labels that fit well together whereas $(\langle x, 0\rangle, \langle y, 0\rangle)$ and $(\langle x, 1\rangle, \langle y, 1\rangle)$ are less interesting pairs of labels.

Hence, a second pheromonal strategy for subset selection problems consists in laying pheromone on pairs of objects.

- The set C of pheromonal components contains every pair of objects $(o_i, o_j) \in S \times S$. Intuitively, the quantity of pheromone on every component $(o_i, o_j) \in C$ —$\tau(o_i, o_j)$— represents the learned desirability of selecting both o_i and o_j within the same solution.
- The set of pheromonal components associated with a solution S_k is the set of pairs of objects $(o_i, o_j) \in S_k \times S_k$.
- The pheromone factor in the probabilistic transition rule depends on the quantity of pheromone on every pair of objects (o_i, o_j) such that o_j is an object that has already been selected in S_k, i.e.,

$$\tau_{factor}(o_i, S_k) = \sum_{o_j \in S_k} \tau(o_i, o_j)$$

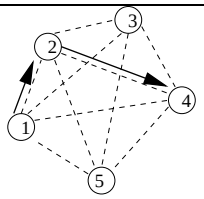
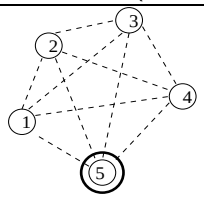
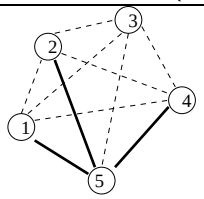
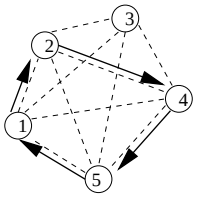
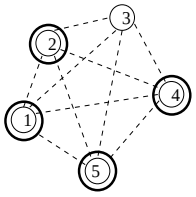
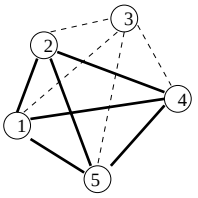
	The ant's walk	Ant-SS(Vertex)	Ant-SS(Clique)
Pheromone factor of a vertex w.r.t. a partial walk	 $S_k = \{1, 2, 4\}$ **Fig. I.1a.**	 $\tau_{factor}(5, S_k) = \tau(5)$ **Fig. I.1b.**	 $\tau_{factor}(5, S_k) = \tau(1,5) + \tau(2,5) + \tau(4,5)$ **Fig. I.1c.**
Pheromonal components that are rewarded w.r.t. a complete walk	 **Fig. I.1d.**	 **Fig. I.1e.**	 **Fig. I.1f.**

Figure I.1. Pheromonal strategies

Note that this pheromone factor can be computed in an incremental way: once the first object o_i has been randomly chosen, for each candidate object o_j, the pheromone factor $\tau_{factor}(o_j, S_k)$ is initialized to $\tau(o_i, o_j)$; then, each time a new object o_l is added to the subset, for each candidate object o_j, the pheromone factor $\tau_{factor}(o_j, S_k)$ is incremented by $\tau(o_l, o_j)$.

I.4.3 Comparison of Ant-SS(Vertex) and Ant-SS(Clique)

Taking a graph-theoretic view, Figure I.1 illustrates the two pheromonal strategies. Fig. I.1a depicts the current set of objects selected by an ant as vertices visited in a walk of a complete graph. In Fig. I.1b and I.1c, we highlight in black the pheromonal components respectively considered by Ant-SS(Vertex) and Ant-SS(Clique) to define the probability of selecting object 5: whereas the pheromonal factor in Ant-SS(Vertex) only depends on the pheromone trail laying on the candidate vertex, the pheromonal factor in Ant-SS(Clique) depends on all pheromone trails laying between the candidate vertex and the current set of objects.

Then, Fig. I.1d depicts the final set of objects selected by an ant. In Fig. I.1e and I.1f, we highlight in black the pheromonal components onto which pheromone will be respectively deposited by Ant-SS(Vertex) and Ant-SS(Clique). In the latter case, note that all edges belonging to the clique of visited vertices and not just the edges that were traversed are updated.

This figure illustrates differences in the time complexities of Ant-SS(Vertex) and Ant-SS(Clique). Indeed, to construct subsets (lines 04 to 16 of Algorithm I.1), Ant-SS(Vertex) and Ant-SS(Clique) perform nearly the same number of operations: the

only difference is in the computation of pheromone factors for the probabilistic transition rule and, as pointed out in Section I.4.2, pheromone factors in Ant-SS(Clique) may be computed in an incremental way, i.e., each time a new object is selected, the pheromone factor of each candidate object can be updated by a simple addition, which further reduces the differences between the time complexities of the two instantiations of the algorithm.

However, to *update* the pheromone trails of pheromonal components (lines 17 to 21 of Algorithm I.1), Ant-SS(Vertex) and Ant-SS(Clique) perform a different number of operations: in Ant-SS(Vertex), the evaporation step requires $\mathcal{O}(\mid S \mid)$ operations and the reward of a subset S_k requires $\mathcal{O}(\mid S_k \mid)$ operations, whereas in Ant-SS(Clique), the evaporation step requires $\mathcal{O}(\mid S \mid^2)$ operations and the reward of a subset S_k requires $\mathcal{O}(\mid S_k \mid^2)$ operations.

I.5 Instantiations of Ant-SS w.r.t. Different Problems

To solve a specific SS problem with Ant-SS, one has to define (1) the heuristic factor $\eta_{factor}(o_i, S_k)$, which evaluates the promise of object o_i with respect to the current solution S_k, and which is used in the probabilistic transition rule; (2) optionally, a local search procedure that may be applied to one or more of the constructed solutions. In some cases, one may modify the definition of δ_τ, that specifies the quantity of pheromone to be added onto pheromonal components.

I.5.1 The maximum clique problem

Heuristic factor. Interestingly, for maximum clique problems it has been found that it is better, in general, for there to be no heuristic factor, i.e., $\eta_{factor}(o_i, S_k) = 1$ [12]. Hence, ants choose objects based on the pheromone factor alone.

The idea in typical heuristics for this problem is to favor vertices with the largest degrees in the 'residual graph', i.e., the subgraph induced by the set *Candidates*. The underlying motivation is that if the selected vertex has a large degree then a larger number of candidates will remain for addition to the clique.

When using no pheromone, or at the beginning of the search when all pheromone trails have the same value, this heuristic does allow ants to find larger cliques than a random choice. But, when combined with learned pheromone, after a hundred cycles or so, larger cliques are obtained without using the heuristic than when using it [12]. This is because the heuristic causes a significant increase in the re-sampling ratio, i.e., the percentage of solutions that are re-computed. This shows that the search gets trapped around a set of locally optimal cliques and is not diversified enough to find large cliques.

Local search. Many local search procedures are possible. The $(2, 1)$-exchange procedure used in GRASP [1] has been successfully applied to an ant algorithm for this problem [25]. Given a solution (clique) S_k, this local search procedure looks for three objects (vertices) o_i, o_j and o_k such that

- o_i belongs to S_k;

- o_j and o_l do not belong to S_k; and
- o_j and o_l are adjacent to every vertex of $S_k - \{o_i\}$.

It then replaces o_i by o_j and o_l, thus increasing the clique size by one. This is applied repeatedly to the largest clique of the cycle until it becomes locally optimal, i.e., it cannot be improved by a further $(2,1)$-exchange. Experiments show that applying local search to more cliques than just the largest does not significantly improve solution quality but is much more time-consuming.

I.5.2 The multidimensional knapsack problem

Heuristic factor. The following heuristic factor has been used in ant algorithms for the multidimensional knapsack problem [17, 2]. Let $c_{S_k}(i) = \sum_{j \in S_k} r_{i,j}$ be the total quantity of resource i consumed by the objects in ant k's solution, S_k. And let $d_{S_k}(i) = b_i - c_{S_k}(i)$ be the remaining capacity of resource i. The following ratio

$$h_{S_k}(j) = \sum_{i=1}^{m} \frac{r_{ij}}{d_{S_k}(i)}$$

represents the tightness of object j: the ratio of its consumption of each of the m resources to their remaining capacity. The lower this ratio, the better is object j.

Finally, we define the heuristic factor for object j by incorporating j's profit to obtain a pseudo-utility factor:

$$\eta_{factor}(j, S_k) = \frac{p_j}{h_{S_k}(j)}$$

Local search. We do not have any experimental results for the performance of different local search procedures on solutions constructed by ants for this problem. Most of the available local search procedures are inspired by the one given in [14]. Space precludes a more detailed description of their approach. In short, search is split into iterations of two alternating phases: *constructive*, when objects are added to the solution, and *destructive*, when objects are removed from the solution. "Critical events" occur when adding objects makes a feasible solution infeasible or when deleting objects makes an infeasible solution feasible. The last solution encountered prior to a critical event is subjected to local optimization; any better solution found is remembered, prior to taking the move.

I.5.3 Maximum constraint satisfaction problems

Heuristic factor. The following heuristic factor has been successfully used in our ant algorithms for constraint satisfaction [23, 30]. It is inversely proportional to the number of *additional* constraints that would be violated. Formally, the heuristic factor for adding object (label) $\langle X, v \rangle$ to set of labels S_k is given by:

$$\eta_{factor}(\langle X, v \rangle, S_k) = \frac{1}{1 + cost(\langle X, v \rangle \cup S_k) - cost(S_k)}$$

where $cost(S)$ is the number of constraints violated by the set of labels S. (Recall that it is already part of our formulation of constraint satisfaction as a SS problem —Section I.2— that the set *Candidates* w.r.t. S_k will not contain any labels that would result in more than one value being assigned to the same variable; so this does not need to be catered for in the definition of the heuristic factor.)

This heuristic factor is not cheap to compute. Furthermore, it may need to be computed for large numbers of candidates: if there are n uninstantiated variables then there are nm candidates, assuming a uniform domain size of m. We have found that, for even quite small problems, the total cost of computing the heuristic factors is unacceptably high.

To reduce the number of candidates for which the heuristic factor is computed, we use what we have called a *cascaded decision*. We first select a variable X_j from among those that are uninstantiated, i.e., $\{X_i \mid \langle X_i, v_i \rangle \in Candidates\}$. This is done using a separate variable ordering heuristic (discussed below). Once the variable has been selected, the heuristic factors of labels associated with all other variables are set to 0 (i.e., the probability of selecting these labels is set to 0 and pheromone factors need not be computed for these labels), whereas the heuristic factors of labels associated with the selected variable are actually computed.

There is a good reason for choosing the next variable independently of the pheromone in a cascaded fashion, quite apart from the time savings that result by computing the heuristic factor fewer times. In any ant walk, there is no question of *whether* or not a variable should be chosen (unlike the case of the values): each variable must be chosen exactly once. Hence, the only decision an ant must make about variables is the *order* in which they are chosen. The pheromone gives no information about preferred orderings. It makes sense then to make this decision separately using information that is predictive of good orderings.

In [30], seven common variable ordering heuristics are compared in an ant algorithm for solving constraint satisfaction problems. The most competitive is the *smallest-domain-first* where the uninstantiated variable with the smallest number of values in its domain that are consistent with the existing labels S_k is chosen.

Local search. Local search using the min-conflicts heuristic [20] has been used in ant algorithms for constraint satisfaction [23]. At each step in the local search, we randomly choose a variable that is involved in one or more violated constraints and then we choose a value for this variable which minimizes the number of conflicts. This repair procedure stops when the number of violated constraints is not improved for a certain number of successive iterations.

I.6 Influence of the parameters on the solution process

When solving a combinatorial optimization problem with a heuristic approach such as evolutionary computation or ACO, one usually has to find a compromise between two dual goals. On the one hand, one has to *intensify* the search around the most "promising" areas, that are usually close to the best solutions found so far. On the other hand, one has to *diversify* the search and favor exploration in order to

discover new, and hopefully more successful, areas of the search space. The behavior of ants with respect to this intensification/diversification duality can be influenced by modifying parameter values.

I.6.1 Influence of τ_{min} and τ_{max}

As pointed out in [29], the goal of bounding pheromone trails within an interval $[\tau_{min}, \tau_{max}]$ is to avoid premature stagnation of search, i.e., a situation where all ants construct the same solution over and over again so that no better solutions can be found anymore. Indeed, by imposing explicit limits τ_{min} and τ_{max} on the minimum and maximum pheromone trails, one ensures that relative differences between pheromone trails cannot become too extreme. Therefore, the probability of choosing a vertex cannot become too small and stagnation situations are avoided. Furthermore, by initializing pheromone trails to τ_{max} at the beginning of the search, one ensures that during early cycles the relative difference between pheromone trails is rather small (after i cycles, it is bounded by a ratio of ρ^i). Hence, exploration is emphasized at the beginning of the search.

In all the considered SS problems in [23, 12, 2, 30], τ_{min} has been set to 0.01. However, the setting of τ_{max} is problem-dependent: [29] has shown that τ_{max} should be set to an estimate of the asymptotically maximum pheromone trail value, which is defined by $\delta_{avg}/(1-\rho)$ where δ_{avg} is the average quantity of pheromone that is laid on pheromonal components at each cycle. This quantity varies from one problem to another. For example, τ_{max} has been set to 6 for maximum clique problems [12] and multiple knapsack problems [2] whereas it has been set to 4 for constraint satisfaction problems [23, 30].

I.6.2 Influence of α and ρ

The two pheromonal parameters α and ρ have a great influence on the solution process. Indeed, diversification can be emphasized either by decreasing the value of the pheromone factor weight α —so that ants become less sensitive to pheromone trails— or by increasing the value of the pheromone persistence rate ρ —so that pheromone evaporates more slowly. When increasing the exploratory ability of ants in this way, one usually finds better solutions, but as a counterpart it takes longer to find them. Hence, one has to set these parameters depending on the availability of time for solving: if CPU-time is limited, it is better to choose parameter values that favor rapid convergence, such as $\alpha \in \{3, 4\}$ and $\rho < 0.99$; if CPU-time is not limited, it is better to choose parameter values that favor exploration, such as $\alpha \in \{1, 2\}$ and $\rho \geq 0.99$. This duality has been observed on many different SS problems: maximum clique problems [12], multiple knapsack problems [2], constraint satisfaction problems [23], and graph matching problems [21].

To speed-up the convergence process of algorithms such as Ant-SS, we have proposed in [24] to introduce a preprocessing step. Indeed, with parameter values that favor exploration, such as $\alpha \in \{1, 2\}$ and $\rho \geq 0.99$, pheromone actually improves the ants' collective behaviour only after some 100 or so cycles, whereas pheromone

is rather expensive to manage —especially for Ant-SS(Clique). The idea is to collect a significant number of solutions in a greedy way —using just the heuristic factor, without using pheromone— and repair them using local search. These solutions constitute a kind of sampling of the search space. Then, we select from this sample set the best ones and use them to initialize pheromone trails. Finally, we continue the search, now guided by pheromone trails as well as the heuristic factor. This preprocessing step has been shown to be effective —allowing us to find better solutions more quickly— on constraint satisfaction problems in [23] and maximum clique problems in [12].

I.6.3 Influence of *nbAnts*

To emphasize diversification and avoid premature stagnation, one can also increase the number of ants so that more states are explored at each cycle. This parameter is usually set experimentally. We will discuss it in the context of maximum clique problems, but we observed very similar results for other SS problems.

We ran Ant-SS(Vertex) on maximum clique problems with different values for *nbAnts* between 10 and 50 [25]. On average, the best results were obtained with $nbAnts = 30$. With lower values, solution quality is often lower because the best clique constructed in each cycle is usually significantly smaller. With greater values, running time often increases while solution quality is not significantly improved because the best clique constructed in each cycle is not significantly better than those found with 30 ants. Increasing the number of ants also generally decreases the number of cycles needed because the quality of the solutions constructed in each cycle is improved. However, with more ants the time needed to compute one cycle increases.

We also found that the preprocessing step that we described in Section I.6.2 allows us to use fewer ants and more often rival the performance of using, for example, four times as more ants without preprocessing.

Finally, note that the best setting for *nbAnts* clearly varies from a SS problem to another, and also from an instance to another within a same SS problem. It might be preferable to take a more analytic approach to setting this parameter, resulting in a problem-dependent heuristic (similar to the case of τ_{max} in Section I.6.1). In the $\mathcal{MAX}$-$\mathcal{MIN}$ Ant System for solving the traveling salesperson problem, for example, the number of ants ordinarily equals the number of vertices [28]. However, an analytic argument for this heuristic is not presented.

I.6.4 Influence of β

This parameter determines the sensitivity of ants to the heuristic factor in the state transition rule. This heuristic factor is problem-dependant. Its relevancy depends on the considered problem, and therefore the setting of β is different from one problem to another.

For example, β has been set to 0 for maximum clique problems in [12, 25] (as no significant heuristic factor has been found); it has been set to 1 for edge-weighted k-cardinality tree problems in [5]; it has been set to 5 for multiple knapsack problems

in both [2] and [17]; and it has been set to 10 for constraint satisfaction problems in both [23] and [30]. For the graph matching problem [21], two different heuristic factors have been combined: a first one that evaluates the immediate benefit of adding an object, and a second one that anticipates the potential benefit of adding this object. These two heuristic factors are weighted by two different parameters: $\beta_1 = 8$ for the immediate benefit and $\beta_2 = 3$ for the anticipated potential benefit.

I.7 Experimental Results

For each of the SS problems described in the previous section, we have published extensive experimental results in [12, 25, 2, 22, 30]. Here we display results that exemplify the effects of the two pheromonal strategies on maximum clique, and constraint satisfaction problems.

Test Suites. For maximum clique problems, we display results obtained on the six $Cn.9$ graphs of the DIMACS challenge on clique coloring and satisfiability[I.3]. These graphs respectively have 125, 250, 500, 1000 and 2000 vertices and maximum known cliques of 34, 44, 57, 68 and 78 vertices respectively. This first test suite allows us to exemplify the scale-up properties of Ant-SS.

For constraint satisfaction problems, we display results obtained on four sets of randomly-generated solvable instances of the model-A class [19]. Each set contains twenty different instances, each of them having 100 variables, a uniform domain size of 8, and a connectivity of 0.14. The four sets of instances have been generated with different tightness ratio p_t in order to exemplify what happens around the phase transition region. This phase transition region —that contains the more difficult instances— occurs when the constrainedness κ is equal to 1 [13]. Hence, the four sets of instances have been generated by respectively setting the tightness ratio p_t to 0.2, 0.23, 0.26, and 0.29, so that the constrainedness κ respectively is equal to 0.74, 0.87, 1.00, and 1.14.

Experimental setup. For all problems, we have set *nbAnts* to 30, α to 1, ρ to 0.99 and τ_{min} to 0.01. For maximum clique problems, we have set τ_{max} to 6 and β to 0; for constraint satisfaction problems, we have set τ_{max} to 4 and β to 10.

For maximum clique problems, we display average results over 50 runs for each of the 6 graphs ; for constraint satisfaction problems, we display average results over 100 runs for each of the 4 sets of instances (5 runs for each of the 20 instances).

All runs have been performed on a 2GHz Pentium 4 processor.

Comparison of solution quality. The first two columns of Table I.1 show the quality of the solutions found by Ant-SS(Vertex) and Ant-SS(Clique) when local search is not applied to improve the solutions constructed by ants. On instances considered in this table (and others not reported here), Ant-SS(Clique) generally outperforms Ant-SS(Vertex): both variants are able to find optimal solutions to "easy" instances (i.e., smaller graph C125.9 for the maximum clique problem, and instances that are

[I.3] This benchmark is available at `http://dimacs.rutgers.edu/`.

Table I.1. Each row displays solution quality of the "vertex" and "clique" pheromonal strategies, without and with local search. For maximum clique problems, it displays the average size of the best clique found; for constraint satisfaction problems, it displays the percentage of runs that succeeded in finding a solution (we only consider solvable instances).

Maximum Clique Problems

		Without local search		With local search	
Graph	$\omega(G)$	Ant-SS(Vertex)	Ant-SS(Clique)	Ant-SS(Vertex)	Ant-SS(Clique)
C125.9	34	34.0	34.0	34.0	34.0
C250.9	44	43.9	44.0	44.0	44.0
C500.9	≥ 57	55.2	55.6	55.3	55.9
C1000.9	≥ 68	65.3	66.0	65.7	66.2
C2000.9	≥ 78	73.4	74.1	74.5	74.3

Constraint Satisfaction Problems

		Without local search		With local search	
p_t	κ	Ant-SS(Vertex)	Ant-SS(Clique)	Ant-SS(Vertex)	Ant-SS(Clique)
0.20	0.74	100%	100%	100%	100%
0.23	0.87	45%	93%	91%	100%
0.26	1.00	89%	97%	99%	100%
0.29	1.14	100%	100%	100%	100%

far enough from the phase transition region for constraint satisfaction problems); however, on harder instances (i.e., when increasing the size of the graph for maximum clique problems, or when getting closer to the phase transition region for constraint satisfaction problems), Ant-SS(Clique) always obtains better results than Ant-SS(Vertex).

The last two columns of Table I.1 show the quality of the solutions found when local search is used to improve solutions built by ants. It shows that integrating local search into Ant-SS actually improves solution quality, for both pheromonal strategies.

Comparison of CPU time. Table I.2 shows that the number of cycles —and therefore the CPU time— needed to find the best solution depends on the problem size and constrainedness. For example, Ant-SS(Vertex) requires respectively 60, 359, 722, 1219 and 1770 cycles on average to solve the six maximum clique instances that have 125, 250, 500, 1000 and 2000 vertices respectively.

This table also shows that Ant-SS(Vertex) needs fewer cycles than Ant-SS(Clique) on maximum clique problems, whereas it nearly always perform more cycles on constraint satisfaction problems. However, as Ant-SS(Clique) needs much more time to perform one cycle than Ant-SS(Vertex), Ant-SS(Vertex) always converges sooner than Ant-SS(Clique).

Table I.2. Each row displays the average number of cycles and the CPU-time (in seconds) spent to find the best solution.

Maximum Clique Problems

	Without local search				With local search			
	Ant-SS(Vertex)		Ant-SS(Clique)		Ant-SS(Vertex)		Ant-SS(Clique)	
Graph	Cycles	Time	Cycles	Time	Cycles	Time	Cycles	Time
C125.9	60	0.1	126	0.2	14	0.0	23	0.0
C250.9	359	0.8	473	1.7	172	0.5	239	1.0
C500.9	722	3.8	923	8.9	477	4.6	671	8.6
C1000.9	1219	13.2	2359	55.0	832	23.4	1242	49.8
C2000.9	1770	41.3	3268	214.4	1427	112.4	2067	238.7

Constraint Satisfaction Problems

	Without local search				With local search			
	Ant-SS(Vertex)		Ant-SS(Clique)		Ant-SS(Vertex)		Ant-SS(Clique)	
p_t	Cycles	Time	Cycles	Time	Cycles	Time	Cycles	Time
0.20	90	1.2	63	4.8	2	0.0	2	0.0
0.23	1084	19.5	849	78.2	472	30.8	230	31.6
0.26	1019	18.1	737	66.1	365	26.9	260	38.7
0.29	449	7.8	516	45.2	27	1.8	38	5.4

Note also that the number of cycles is always decreased when local search is used to improve solutions built by ants. However, as this local seach step is time consuming, CPU-times are not always decreased.

CPU time vs solution quality. Tables I.1 and I.2 show us that to choose between Ant-SS(Vertex) and Ant-SS(Clique), one has to consider the CPU time available for the solution process. Indeed, Ant-SS(Clique) usually finds better solutions than Ant-SS(Vertex) at the end of its solution process, but it is also more time consuming. As a consequence, if one has to find a solution within a short time limit, one has better use Ant-SS(Vertex), whereas for larger time limits, or when there is no time limit, one has better use Ant-SS(Clique). This is illustrated in Figure I.2 on the maximum clique problem with graph `C500.9`. This figure plots the evolution of the size of largest clique with respect to CPU-time. It shows us that for time limits smaller than 8 seconds, Ant-SS(Vertex) finds larger cliques than Ant-SS(Clique), whereas for larger time limits Ant-SS(Clique) finds larger cliques than Ant-SS(Vertex).

Figure I.2 also compares Ant-SS with a multi-start local search procedure called `multi-start LS`. This multi-start local search procedure iterates on the two following steps: (1) randomly build a maximal clique, and (2) apply the local search procedure described in Section I.5.1 on this maximal clique. Figure I.2 shows us that during the first second of CPU time `multi-start LS` finds better solutions than Ant-SS. Indeed, Ant-SS spends time to manage pheromone whereas this pheromone starts influencing ants only after a few hundreds or so cycles. Hence, Ant-SS(Vertex)

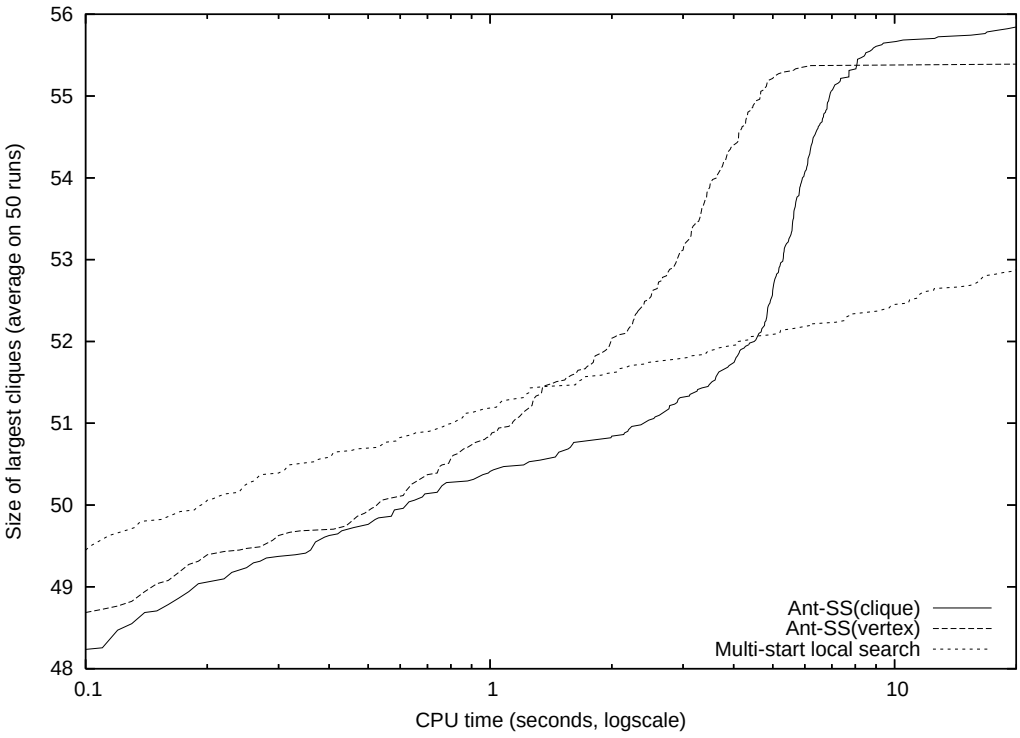


Figure I.2. Convergence results for the different strategies.

(resp. Ant-SS(Clique)) finds better solutions than `multi-start LS` only after one second (resp. five seconds) of CPU time.

Comparison with other approaches. Table I.1 shows that, for constraint satisfaction problems, all runs of Ant-SS(Clique) with local search have been successfull, even for the more difficult instances. In [16], we report more experimental results, and we compare Ant-SS(Clique) performances with other state of the art evolutionary approaches and with a constraint programming approach based on a complete tree search. We show that Ant-SS(Clique) clearly outperforms evolutionary approaches —as it nearly always succeeds in finding a solution whereas evolutionary approaches much more often fail when getting closer to the phase transition region. We also show that, if for small instances constraint programming is faster than Ant-SS(Clique), run-times of constraint programming grow exponentially when increasing problem size so that its efficiency becomes significantly lower than Ant-SS(Clique) on large instances.

For maximum clique problems, we compare Ant-SS performances with three other state of the art approaches in [25]: a reactive tabu search approach [3], an adaptive greedy approach [15] and a genetic approach combined with local search [18]. We show that Ant-SS(Clique) obtains very competitive results: it clearly outperforms the genetic approach; it obtains results that are comparable with the

adaptive greedy approach —outperforming it on some instances, and being outperformed on others; it is slightly outperformed by the reactive tabu search approach —which is the best algorithm for the maximum clique problem we are aware of.

Also, in [21], we have compared Ant-SS(Clique) with the reactive tabu search approach of [26] for solving graph matching problems, and we have shown that the two approaches obtain very complementary results, each approach being able to solve instances that the other one cannot solve.

I.8 Summary

We have defined the class of subset selection problems, in which the task is to find a feasible and optimal subset of an initial set of objects S. We have shown that problems such as maximum clique, multidimensional knapsack and maximum constraint satisfaction are examples of this class of problems.

We have given a generic ACO algorithm for solving subset selection problems. The algorithm is instantiated with respect to two main parameters: the subset selection problem $(S, S_{feasible}, f)$ and the pheromonal strategy ϕ. The pheromonal strategy determines the components on which pheromone is placed, and we have defined and investigated two strategies that are suited to subset selection problems. In one, the "vertex" strategy, pheromone is placed on each object of S to represent the learned desirability of selecting that object; in the other, the "clique" strategy, pheromone is placed on pairs of objects of S to represent the learned desirability that the two objects belong to the same subset.

Then, we discussed the effect of the algorithm's numeric parameters (α, β, ρ, τ_{min}, τ_{max} and $nbAnts$). In particular, we described how to set these parameters and the influence each has on the degree of exploration in runs of the algorithm.

We showed how to instantiate the algorithm for each of maximum clique, multidimensional knapsack and maximum constraint satisfaction. This required us to define possible heuristic factors and local search procedures.

Finally, we included some selected experimental results for maximum clique and constraint satisfaction problems. In particular, we compared the two pheromonal strategies. We showed that the "clique" strategy typically finds better quality solutions but the "vertex" strategy generally requires less CPU-time.

Bibliography

[1] J. Abello, P.M. Pardalos and M.G.C. Resende: On maximum clique problems in very large graphs. In DIMACS Series on Discrete Mathematics and Theoretical Computer Science, vol.50, pp.119–130, American Mathematical Society, 2000

[2] I. Alaya, C. Solnon, and K. Ghdira: Ant algorithm for the multi-dimensional knapsack problem, in International Conference on Bioinspired Optimization Methods and their Applications (BIOMA 2004), pp. 63–72, 2004

[3] R. Battiti and M. Protasi: Reactive Local Search for the Maximum Clique Problem. In Algorithmica, 29(4), pp.610–637, 2001

[4] I. Bomze, M. Budinich, P. Pardalos, and M. Pelillo: The maximum clique problem. In D.-Z. Du and P. M. Pardalos (eds.), Handbook of Combinatorial Optimization, vol. 4, pp. 1–74, Kluwer Academic Publishers, 1999

[5] C. Blum: Ant Colony Optimization for the Edge-Weighted k-Cardinality Tree Problem. In GECCO 2002, pp. 27-34, 2002

[6] B. Bullenheimer, R.F. Hartl and C. Strauss: Applying the Ant System to the Vehicle Routing Problem. In S. Voß, S. Martello, I.H. Osman and C. Roucairol (eds.), Meta-Heuristics: Advances and Trends in Local Search Paradigms for Optimization, pp.285–296, Kluwer, 1999

[7] P.-A. Champin and C. Solnon: Measuring the similarity of labeled graphs, in 5th International Conference on Case-Based Reasoning (ICCBR 2003), LNCS 2689, Springer-Verlag, pp. 80-95, 2003

[8] M. Dorigo, V. Maniezzo and A. Colorni: The Ant System: Optimization by a Colony of Cooperating Agents. IEEE Transactions on Systems, Man, and Cybernetics – Part B, 26(1), pp.1–13, 1996

[9] M. Dorigo and G. Di Caro: The Ant Colony Optimization Meta-Heuristic. In D. Corne, M. Dorigo and F. Glover (eds.), New Ideas in Optimization, pp.11–32, McGraw Hill, 1999

[10] M. Dorigo, G. Di Caro and L.M. Gambardella: Ant Algorithms for Discrete Optimization. Artificial Life, 5(2), pp.137–172, 1999

[11] M. Dorigo and T. Stützle: Ant Colony Optimization. MIT Press, 2004

[12] S. Fenet and C. Solnon: Searching for Maximum Cliques with Ant Colony Optimization. In "Applications of evolutionary computing (EvoCOP 2003)", LNCS 2611, Springer Verlag, pp. 236–245, 2003

[13] I.P. Gent, E. MacIntyre, P. Prosser, and T. Walsh: The constrainedness of search. Proceedings of AAAI-96, AAAI Press, Menlo Park, California, 1996

[14] F. Glover and G.A. Kochenberger: Critical Event Tabu Search for Multidimensional Zero-One Knapsack Problems. In I.H. Osman and J.P. Kelly (eds.), Metaheuristics: The Theory and Applications, pp.407–427, Kluwer, 1996

[15] A. Grosso, M. Locatelli, and F. Della Croce: Combining Swaps and Node Weights in an Adaptive Greedy Approach for the Maximum Clique Problem. In Journal of Heuristics, 10(2), Kluwer Academic Publishers, pp.135–152, 2004

[16] J. van Hemert and C. Solnon: A Study into Ant Colony Optimization, Evolutionary Computation and Constraint Programming on Binary Constraint Satisfaction Problems. In 4th European Conference on Evolutionary Computation in Combinatorial Optimization (EvoCOP 2004), LNCS 3004, Springer Verlag, pp.114–123, 2004

[17] G. Leguizamon and Z. Michalewicz: A new version of Ant System for Subset Problem. In Congress on Evolutionary Computation, pp. 1459–1464, 1999

[18] E. Marchiori: Genetic, Iterated and Multistart Local Search for the Maximum Clique Problem. In "Applications of Evolutionary Computing, Proceedings of EvoWorkshops 2002: EvoCOP, EvoIASP, EvoSTim", LNCS 2279, Springer Verlag, pp.112–121, 2002

[19] E. MacIntyre, P. Prosser, B. Smith, and T. Walsh: Random Constraints Satisfaction: theory meets practice. CP98, LNCS 1520, Springer Verlag, pp.325–339, 1998

[20] S. Minton, S.D. Johnston, A.B. Philips and P. Laird: Minimizing Conflicts: A Heuristic Repair Method for Constraint Satisfaction and Scheduling Problems. In Artificial Intelligence, 58, pp.161–205, 1992

[21] O. Sammoud, C. Solnon, and K. Ghdira: Ant Algorithm for the Graph Matching Problem. In 5th European Conference on Evolutionary Computation in Combinatorial Optimization (EvoCOP 2005), LNCS 3448, Springer-Verlag, pp. 213–223, 2005

[22] C. Solnon: Solving Permutation Constraint Satisfaction Problems with Artificial Ants. In Proceedings of ECAI'2000, IOS Press, Amsterdam, The Netherlands, pp. 118–122, 2000

[23] C. Solnon: Ants Can Solve Constraint Satisfaction Problems. In IEEE Transactions on Evolutionary Computation, 6(4), pp. 347–357, 2002

[24] C. Solnon: Boosting ACO with a Preprocessing Step. In "Applications of Evolutionary Computing, Proceedings of EvoWorkshops2002: EvoCOP, EvoIASP, EvoSTim", LNCS 2279, Springer-Verlag, pp. 161–170, 2002

[25] C. Solnon and S. Fenet: A Study of ACO Capabilities for Solving the Maximum Clique Problem, Journal of Heuristics, forthcoming, 2005

[26] S. Sorlin and C. Solnon: Reactive Tabu Search for Measuring Graph Similarity. In the 5th IAPR Workshop on Graph-based Representations in Pattern Recognition (GbR 2005), LNCS 3434, Springer Verlag, pp.172–182, 2005

[27] T. Stützle and M. Dorigo: ACO Algorithms for the Quadratic Assignment Problem. In D. Corne, M. Dorigo and F. Glover (eds.), New Ideas in Optimization, pp.33–50, McGraw Hill, 1999

[28] T. Stützle and H. Hoos: Improvements on the Ant System: Introducing the $\mathcal{MAX}$-$\mathcal{MIN}$ Ant System. In G. Smith, N. Steele and R. Albrecht (eds.), Procs. of Artificial Neural Nets and Genetic Algorithms, pp.245–249, 1997

[29] T. Stützle and H. Hoos: $\mathcal{MAX} - \mathcal{MIN}$ Ant System. In Journal of Future Generation Computer Systems, vol. 10, pp.889–914, 2000

[30] F. Tarrant and D. Bridge: When Ants Attack: Ant Algorithms for Constraint Satisfaction Problems. Artificial Intelligence Review, forthcoming.

[31] E.P.K. Tsang: Foundations of Constraint Satisfaction, Academic Press, 1993