A New Version of Ant System for Subset Problems

Guillermo Leguizamón
Interest Group on Computer Systems
Universidad Nacional de San Luis, Argentina
legui@unsl.edu.ar

Zbigniew Michalewicz
University of North Carolina
Charlotte, NC 28223, USA
zbyszek@uncc.edu *and*
Institute of Computer Science
Polish Academy of Sciences

Abstract- Early applications of Ant Colony Optimization (ACO) have been mainly concerned with solving ordering problems (e.g., Traveling Salesman Problem). In this paper we introduce a new version of Ant System — an ACO algorithm for solving subset problems. The computational study involves the Multiple Knapsack Problem (MKP); the reported results show the potential power of the ACO approach for solving this type of subset problems.

1 Introduction

The Ant Colony Optimization (ACO) technique has emerged recently [3, 4] as a new meta-heuristic for hard combinatorial optimization problems. This meta-heuristic belongs to the class of problem-solving strategies derived from nature (other categories include evolutionary algorithms, neural networks, simulated annealing). The ACO algorithm is basically a multi-agent system where low level interactions between single agents (i.e., artificial ants) result in a complex behaviour of the whole ant colony.

ACO algorithms have been inspired by colonies of real ants [3], which deposit a chemical substance (called *pheromone*) on the ground. This substance influences the choices they make: the larger amount of pheromone is on a particular path, the larger probability is that an ant selects the path. Artificial ants in ACO algorithms bahave in similar way.

Early experiments with the ACO algorithm were connected with ordering problems such as the Traveling Salesman Problem or the Quadratic Assignment Problem. In section 2 we illustrate the basic concepts of the original Ant System algorithm, the first ACO algorithm introduced by Dorigo, Maniezzo, and Colorni [5], using as example the Traveling Salesman Problem (TSP). Further sections of this paper investigate the applicability of the ACO algorithm for solving subset problems. The proposed application is a particular implementation of the ACO meta-heuristic in which pheromone trail is put on the problem's components instead of the problem's connections.

2 Ant System for the TSP

Given a set of n cities and a set of distances between them, the Traveling Salesman Problem (TSP) is the problem of finding a minimum length closed path (a *tour*), which visits every city exactly once. Thus we have to minimize

$$COST(i_1, \ldots, i_n) = \sum_{j=1}^{n-1} d(C_{i_j}, C_{i_{j+1}}) + d(C_{i_n}, C_{i_1}), \quad (1)$$

where $d(C_x, C_y)$ is the distance between cities x and y.

Let $b_i(t)$ $(i = 1, ..., n)$ be the number of ants in city i at time t and let $a = \sum_{i=1}^{n} b_i(t)$ be the total number of ants. Let $\tau_{ij}(t+n)$ be the intensity of pheromone trail on connection (i, j) at time $t+n$, given by

$$\tau_{ij}(t+n) = (1-\rho)\tau_{ij}(t) + \Delta\tau_{ij}(t, t+n), \quad (2)$$

where $0 < \rho \leq 1$ is a coefficient which represents pheromone evaporation. $\Delta\tau_{ij}(t, t+n) = \sum_{k=1}^{a} \Delta\tau_{ij}^k(t, t+n)$, where $\Delta\tau_{ij}^k(t, t+n)$ is the quantity per unit of length of trail substance (pheromone in real ants) laid on connection (i, j) by the k^{th} ant at time $t+n$ and is given by the following formula:

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

where Q is a constant and L_k is the tour length found by the k^{th} ant. For each edge, the intensity of trail at time 0 ($\tau_{ij}(0)$) is set to a very small value.

While building a tour, the probability that ant k in city i visits city j is

$$P_{ij}^k(t) = \begin{cases} \frac{[\tau_{ij}(t)]^\alpha [\eta_{ij}]^\beta}{\sum_{h \in allowed_k(t)} [\tau_{ih}(t)]^\alpha [\eta_{ih}]^\beta}, & \textit{if } j \in allowed_k(t) \\ 0, & \textit{otherwise,} \end{cases} \quad (3)$$

where $allowed_k(t)$ is the set of cities not visited by ant k at time t, and η_{ij} represents a local heuristic. For the TSP, $\eta_{ij} = \frac{1}{d(C_i, C_j)}$ (and it is called 'visibility').

The parameters α and β control the relative importance of pheromone trail versus visibility. Hence, the transition probability is a trade-off between visibility, which says that closer cities should be chosen with a higher probability, and trail intensity, which says that if the connection (i, j) enjoys a lot of traffic then is it highly profitable to follow it.

A data structure, called a *tabu list*, is associated to each ant in order to avoid that ants visit a city more than once. This list $tabu_k(t)$ maintains a set of visited cities up to time t by the k^{th} ant. Therefore, the set $allowed_k(t)$ can be defined as follows: $allowed_k(t) = \{j | j \notin tabu_k(t)\}$. When a tour is completed, the $tabu_k(t)$ list $(k = 1, \ldots, a)$ is emptied and every ant is free again to choose an alternative tour for the next cycle.

By using the above definitions, we can describe the Ant System algorithm as follows:

Initialize
for t=1 to number of cycles do
  for k=1 to a do
    Repeat until k has completed a tour
      Select city j to be visited next with probability P_{ij}^k
        given by Eq. (3)
    end
    Calculate the length L_k *of the tour generated by ant k*
  end
  Save the best solution found so far
  Update the trail levels τ_{ij} *on all paths according to Eq. (2)*
end
Print the best solution found

3 Ant System for Subset Problems

Subset problems are quite different from ordering problems. Out of a set S of n items we have to select the best subset of s items, possibly satisfying some additional constraints. There is no concept of a path here, so it is difficult to apply the concepts described in the previous section directly to subset problems. The main difference is the following. In ordering problems, the sequence $\tilde{S} =< i_1, i_2, ..., i_j >$ and the set $R = \{i_{j+1}, i_{j+2}, ..., i_n\}$ represent a partial solution to the problem and the set of remaining cities to be considered in order to complete the ordering of n items from the set S, respectively. The selection processes of the next item from the set R involves probabilities $P_{i_j i_p}^k(t)$ (e.g., Eq. (3)) $(p \in \{j+1, j+2, ..., n\})$, which depend on the trail $\tau_{i_j i_p}$ on the edge (i_j, i_p) and the local heuristic measure $\eta_{i_j i_p}$ (Figure 1).

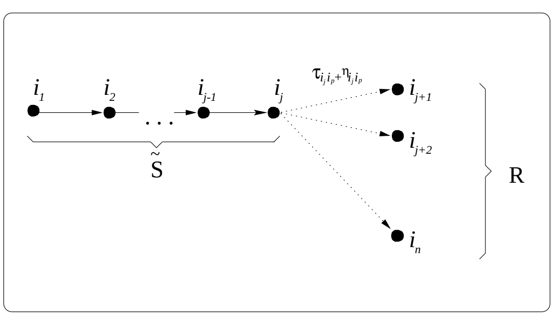

Figure 1: A sequence representing a partial solution $\tilde{S}$ at step j during a particular cycle

On the other hand, in subset problems we are not interested in solutions giving a particular order (e.g., a tour in the TSP). Therefore, a partial solution is represented by $\tilde{S} = \{i_1, i_2, ..., i_j\}$ and the most recent element incorporated to $\tilde{S}$, i_j, need not be involved in the process for selecting the next element (Figure 2).

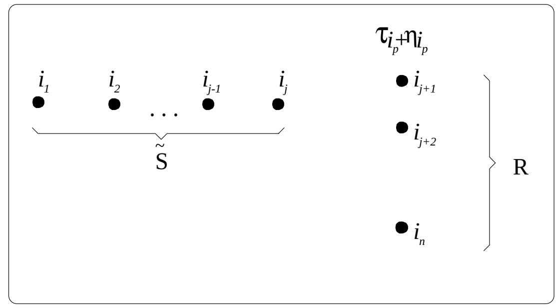

Figure 2: A set representing a partial solution $\tilde{S}$ at step j during a particular cycle

Moreover, solutions for ordering problems have a fixed length, as we search for a permutation of a known number of elements. Thus, for example, the update of the pheromone trails could be done once every n steps, that is once *all* the ants have completed an ordering (which happens, of course, at the same time). Solutions for subset problems, however, do not have a fixed length. Thus it is necessary to establish a number, N_{max}, which will be used to determine the end of the construction cycle for all the ants. The original Ant System must therefore be modified accordingly.

First of all, the pheromone trail is now laid on each element from set S, with the intended meaning that elements with a higher trail level are more profitable. Therefore, the *intensity of pheromone trail* on item i at time $t + N_{max}$ is given by:

$$\tau_i(t + N_{max}) = (1 - \rho)\tau_i(t) + \Delta\tau_i(t, t + N_{max}), \quad (4)$$

where $N_{max} < n$ is the maximum number of items allowed to be added to some solution by some ant. The constant N_{max} was introduced to achieve consistency with the definitions of Ant System for ordering problems, where n items must be considered to obtain a permutation and where the updating process on the trail values is done every n "units of time" (that is, after the ants have simultaneously completed a permutation). On the other hand, for subset problems, the length of a cycle varies as the ants start simultaneously to build solutions, but they finish at different times, depending on the number of items satisfying the problem constraints. The ants always finish before the time $(t + n)$, that is at time $(t + N_{max})$. Let us denote

$$\Delta\tau_i(t, t + N_{max}) = \sum_{i=1}^{a} \Delta\tau_i^k(t, t + N_{max}),$$

which is obtained by summing the contribution $\Delta\tau_i^k(t, t + N_{max})$ of each ant k. In other words, this is the quantity of pheromone trail laid on item i by the k^{th} ant at time $t + N_{max}$.

This quantity is given by the following formula:

$$\Delta\tau_i^k(t, t+N_{max}) = \begin{cases} G(L_k), \ \textit{if } k^{th} \textit{ ant} \\ \qquad \textit{incorporates item } i \\ 0, \ \textit{otherwise} \end{cases} \quad (5)$$

In Eq. (5) the function G depends on the problem and gives the amount of trail being added to item i. Usually, $G(L_k) = Q/L_k$ or $G(L_k) = QL_k$ for minimization or maximization problems, respectively (Q is a parameter of the method). L_k is the value of the *objective function* obtained by the k^{th} ant. Further, the local heuristic should assign a value to each element without any considerations about possible connections between them (ordering is not important any longer). Although not present in the original formulation of Ant System, we consider two types of heuristics for our new version,[1] static and dynamic.

- Static: η_i is set at the beginning of the run to a fixed value $\forall_i \in S$
- Dynamic: $\eta_i(\tilde{S}_k(t))$ depends on the partial solution, $i \in S - \tilde{S}_k(t)$ and $\tilde{S}_k(t)$ is the k^{th} partial solution at time t

Then, for a partial solution $\tilde{S}_k(t) = \{i_1, ..., i_j\}$ being built by ant k, the probability $P_{i_p}^k(t)$ of selecting i_p as the next item ($p \in \{j+1, j+2, ..., n\}$) is given as

$$P_{i_p}^k(t) = \begin{cases} \frac{[\tau_{i_p}(t)]^\alpha [\eta_{i_p}(\tilde{S}_k(t))]^\beta}{\sum_{j \in allowed_k(t)} [\tau_j(t)]^\alpha [\eta_j(\tilde{S}_k(t))]^\beta} \\ \qquad \textit{if } i_p \in allowed_k(t) \\ 0, \ \textit{otherwise,} \end{cases} \quad (6)$$

where $allowed_k(t) \subseteq S - \tilde{S}_k(t)$ is the set of remaining feasible items. Thus, the higher the value of τ_{i_p} and/or $\eta_{i_p}(\tilde{S}_k(t))$, the more profitable it is to include item i_p in the partial solution.

The outline of the new version of the Ant System algorithm for subset problems is as follows:

Initialize
for t=1 to number of cycles do
  $N_{max} = 0$
  for k=1 to a do
    $N_{items} = 0$
    Repeat Until $allowed_k$ *is empty*
      Select item i to be incorporated with probability P_i^k
        given by Eq. (6)
      $N_{items} \leftarrow N_{items} + 1$
    end
    Calculate L_k*, the objective function of the generated*
      solution
    Save the best solution so far
    $N_{max} = \max\{N_{item}, N_{max}\}$
  end
  Update the trail levels τ_i *on all items according to Eq. (4)*
end
Print the best solution found

[1] In this work we use only a dynamic local heuristic. See [7] for a definition of a static local heuristic.

The subset-based and permutation-based Ant Systems have many features in common. However, in the permutation-based Ant System the pheromone is laid on *paths* while for subset problems no path exists connecting the items. A subset-based Ant System takes advantage of one of the central ideas involved in the selection process of a permutation-based ant system: "the more amount of trail on a particular *path*, the more profitable is that *path*". This idea was adapted here in the following way: "the more pheromone trail on a particular *item*, the more profitable that *item* is". In other words, we move the pheromone from paths to items. At the same time, a local heuristic is also used in the new version, but now it considers *items* only instead of *connections* between them.

4 Formulation of Ant System for the MKP

The Multiple Knapsack Problem (MKP)[2] can be formulated as follows [2]:

$$\begin{array}{l} maximize \sum_{j=1}^n p_j x_j \\ subject\ to \ \sum_{j=1}^n r_{ij} x_j \le c_i \quad i = 1, ..., m, \\ x_j \in \{0, 1\} \quad j = 1, ...n. \end{array} \quad (7)$$

There are m constraints in this problem, so the MKP is also called the *m-dimensional Knapsack Problem*. Let $I = \{1, ..., m\}$ and $J = \{1, ..., n\}$, with $c_i \ge 0$ for all $i \in I$. A *well-stated* MKP assumes that $p_j > 0$ and $r_{ij} \le c_i \le \sum_{j=1}^n r_{ij}$ for all $i \in I$, $j \in J$, since any violation of these conditions will result in some x_j being fixed to zero or some constraints being eliminated. Note that the $[r_{ij}]_{m \times n}$ matrix and $[c_i]_m$ vector are both non-negative, which distinguishes this problem from the general 0-1 linear integer programming problem. Many practical problems can be formulated as a MKP, for example, the capital budgeting problem, where a project j has profit p_j and consumes r_{ij} units of resource i. The goal is to find a subset of projects $J^* \subseteq J$ such that the total profit is maximized and all resource constraints are satisfied. For solving MKP, the ants look for a subset of n items (see the MKP formulation) such that the total profit is maximized and all resource constraints are satisfied.

In order to define the local heuristic for MKP, let us consider the following example. Let $n = 4$ and $m = 3$ be the number of items and constraints, respectively. The following vectors and matrix represent the values for p_j, c_i, and r_{ij}: $(p_1, p_2, p_3, p_4) = (4, 10, 2, 6)$, $(c_1, c_2, c_3) = (8, 12, 10)$, and

$$\begin{pmatrix} r_{11} & r_{12} & r_{13} & r_{14} \\ r_{21} & r_{22} & r_{23} & r_{24} \\ r_{31} & r_{32} & r_{33} & r_{34} \end{pmatrix} = \begin{pmatrix} 4 & 4 & 2 & 1 \\ 6 & 6 & 3 & 3 \\ 4 & 4 & 2 & 2 \end{pmatrix}$$

Recall, that $\tilde{S}_k(t)$ denotes the set of items that have been selected by ant k at time t. Suppose that $\tilde{S}_k(1) = \{4\}$. Some definitions are given before we define the local heuristic. Let $u_i(k, t) = \sum_{l \in \tilde{S}_k(t)} r_{il}$ be the amount of resource i consumed at the time t, with respect to the solution being built by ant k. Following our example, we have:

[2] MKP belongs to the family of *NP-hard* problems.

$$u_1(k,t) = \sum_{l \in \{4\}} r_{1l} = r_{14} = 1$$
$$u_2(k,t) = \sum_{l \in \{4\}} r_{2l} = r_{24} = 3$$
$$u_3(k,t) = \sum_{l \in \{4\}} r_{3l} = r_{34} = 2$$

Let $\gamma_i(k,t) = c_i - u_i(k,t)$ be the remaining amount to reach the boundary of the constraint i:

$$\gamma_1(k,t) = c_1 - u_1(k,t) = 8 - 1 = 7$$
$$\gamma_2(k,t) = c_2 - u_2(k,t) = 12 - 3 = 9$$
$$\gamma_3(k,t) = c_3 - u_3(k,t) = 10 - 2 = 8$$

The following formula gives the $tightness$ of item j on constraint i according to the $\tilde{S}_k(t)$, i.e., the ratio between r_{ij}, the amount of resource i consumed by project j, and $\gamma_i(k,t)$. So, the lower the ratio, the better the item:

$$\delta_{ij}(k,t) = \frac{r_{ij}}{\gamma_i(k,t)} \quad (8)$$

Each $\delta_{ij}(k,t)$ is computed as follows:

$$\delta_{11}(k,t) = \tfrac{4}{7} \quad \delta_{12}(k,t) = \tfrac{4}{7} \quad \delta_{13}(k,t) = \tfrac{2}{7}$$

$$\delta_{21}(k,t) = \tfrac{6}{9} \quad \delta_{22}(k,t) = \tfrac{6}{9} \quad \delta_{23}(k,t) = \tfrac{3}{9}$$

$$\delta_{31}(k,t) = \tfrac{4}{8} \quad \delta_{32}(k,t) = \tfrac{4}{8} \quad \delta_{33}(k,t) = \tfrac{2}{8}$$

Finally, we define the average tightness on all constraints i in case of item j being chosen to be included in $\tilde{S}_k(t)$:

$$\bar{\delta}_j(k,t) = \frac{\sum_{i=1}^{m} \delta_{ij}(k,t)}{m} \quad (9)$$

For our example, larger values are obtained for variables $j = 1$ and $j = 2$:

$$\bar{\delta}_1(k,t) = 0.579; \; \bar{\delta}_2(k,t) = 0.579; \; \bar{\delta}_3(k,t) = 0.290$$

However, we need to take in account the profits p_j in order to obtain a *pseudo-utility* measure for each candidate item. Therefore, the local heuristic for the MKP, $\eta_j(\tilde{S}_k(t))$, is defined as follows:

$$\eta_j(\tilde{S}_k(t)) = \frac{p_j}{\bar{\delta}_j(k,t)} \quad (10)$$

Referring to our example, the most profitable item is $j = 2$, since its profit p_2 is larger than p_1. These values of η_j:

$$\eta_1(\{4\}) = \tfrac{4}{0.579} = 6.908, \eta_2(\{4\}) = \tfrac{10}{0.579} = 17.271,$$
$$\eta_3(\{4\}) = \tfrac{2}{0.290} = 6.897$$

will affect the probability values for item selection (Eq. 6). Hence, the transition probability values represent a trade-off between *pseudo-utility* (which says, that more profitable items which use less resources should be chosen with a high probability) and trail intensity (which says, that if an item j is included in many solutions, then it is highly desirable).

A data structure, called *tabu list*, is also associated with each ant in order to prevent an ant from chosing an item more than once, i.e., $tabu_k(t)$ maintains a set of items included in the solution up to time t by the k^{th} ant. This list also maintains $u_j(k,t)$ $(j = 1, \ldots, m)$ in order to reduce the computational time. Thus the set $allowed_k(t)$ can be defined as follows:

$allowed_k(t) = \{j | j \notin tabu_k(t)$ *and* $\tilde{S}_k(t)$*, in case item j is added, satisfies all the constraints*$\}$,

and $tabu_k(t)$ list represents $\tilde{S}_k(t)$ according to our definition of section 3. Since MKP is a maximization problem, then $G(L_k) = QL_k$, where $Q = \frac{1}{\sum_{j=1}^{n} p_j}$.

5 Experiments and results

The Ant System was coded in C language and was run in the Parallel Virtual Machine environment to take advantage of the distributed features of the algorithm [10]. The parallel version of the Ant System run on a Parsytec based on PowerPC processors. In experiments reported in this section, different parameters values were considered: $\alpha = 1$; $\beta = 1, 5, 9$; $\rho = 0.3$, and the total number of ants in the system a is equal to n, where $n = |S|$, i.e., the cardinality of the MKP. The maximum number of cycles was set to 100 for all experiments. The results are expressed in terms of averages out of ten runs (with different seeds).

The Ant System was tested on 11 MKP instances taken from [1]. Tables 1 and 2 show, for each instance, the known optimum, the average best result, and the number of *hits* — runs (out of 10) in which the system found the optimum solution.

Table 1: *The results of AS for the 11 test cases of MKP*

		(α, β)					
Instance	Opt	(1,1)		(1,5)		(1,9)	
mkp1	7772	7772	10	7772	10	7772	10
mkp2	8722	8717	4	8722	10	8719	9
mkp3	141278	141078	6	140778	0	140778	0
mkp4	130883	130645	5	130819	6	130883	10
mkp5	95677	95667	8	95667	8	95667	8
mkp6	119337	119337	10	119337	10	119337	10
mkp7	98796	98796	10	98796	10	98796	10
mkp8	130623	130389	4	130623	10	130311	1
mkp9	1095445	1095253	0	1095382	0	1095382	0
mkp10	624319	622821	6	624319	10	622238	1
mkp11	4554	4554	10	4554	10	4554	10

Table 2: *The results of AS for the 11 test cases of MKP*

		(α, β)					
Instance	Opt	(5,1)		(5,5)		(5,9)	
mkp1	7772	7635	0	7768	7	7770	8
mkp2	8722	8655	0	8720	7	8716	0
mkp3	141278	139583	0	140778	0	140778	0
mkp4	130883	127558	0	130439	0	130787	4
mkp5	95677	94351	0	95657	6	95637	1
mkp6	119337	116690	1	119337	10	119337	10
mkp7	98796	97460	4	98796	10	98796	10
mkp8	130623	125742	0	130311	2	130233	0
mkp9	1095445	1092659	0	1095376	0	1095382	0
mkp10	624319	615414	0	624178	8	622066	0
mkp11	4554	4491	0	4554	10	4554	10

The results reported in Tables 1 and 2 indicate that the best performance (i.e., largest number of hits) is obtained for cases where $\alpha = 1$ (in accordance with the earlier results reported in [6, 7]). The test case mkp9 was the hardest; the Ant System failed to find the optimum for all parameters combinations. For this test case, the best obtained result is equal to the suboptimal value (1095382) reported in [9]. Note also, that the largest instances in this test set were mkp9 and mkp10 with $n = 105$ variables and $m = 5$ constraints; all other test cases were of smaller size.

Additional MKP instances (taken also from [1]) were tested with $(\alpha, \beta) = (1, 5)$; one of the combinations that showed the best performance on the earlier instances. All these instances have $n = 100$ variables and $m = 5$ or $m = 10$ constraints.[3] The AS found the optimum solution for 13 (out of 20) test cases with 100 variables. In 13 test cases the best solution was found within 40 cycles; only 3 test cases required more than 70 cycles (Table 3).

Table 3: *The results of AS for additional test cases of MKP*

Instance	Best known	Best Found	Average	#cycle
5.100-00	24381	**24381**	24331.2	35
5.100-01	24274	**24274**	24245.6	23
5.100-02	23551	**23551**	23527.6	11
5.100-03	23534	23527	23463.0	78
5.100-04	23991	**23991**	23949.8	34
5.100-05	24613	**24613**	24563.0	22
5.100-06	25591	**25591**	25504.8	35
5.100-07	23410	**23410**	23361.8	22
5.100-08	24216	24204	24173.4	43
5.100-09	24411	**24411**	24326.0	17
10.100-00	23064	23057	22996.4	59
10.100-01	22801	**22801**	22672.2	55
10.100-02	22131	**22131**	21980.0	24
10.100-03	22772	**22772**	22631.0	72
10.100-04	22751	22654	22578.4	42
10.100-05	22777	22652	22565.2	77
10.100-06	21875	**21875**	21758.2	21
10.100-07	22635	22551	22519.4	11
10.100-08	22511	22418	22292.4	62
10.100-09	22702	**22702**	22588.0	24

It is interesting to compare this performance of AS with some other meta-heuristic technique. We have developed an evolutionary algorithm (EA) where an individual is represented as permutation vector of n numbers (for n variables of the MKP) and a decoder uses the solution vector to build a (unique) feasible solution. The operators used were order crossover and "swap two numbers" mutation. The parameter values were $p_c = 0.65$, $p_m = 0.2$, generation gap equal 50%, population size of 100, maximum number of generations: 10000. All discussed results report averages of 10 runs.

On the set of 11 test cases (mkp1 – mkp11) the performance of the EA was quite similar to the performance of AS. EA scored 10 hits (out of 10) for cases mkp3, mkp4, mkp5, mkp6, and mkp11. It failed (i.e., 0 hits) on cases mkp2, mkp9, and mkp10. For the remaining test cases (e.g., mkp1, mkp7, and mkp8) the number of hits were 4, 6, and 8, respectively (Table 4).

It was also interesting to note that an increase in population size (from 100 to 200) improved only slightly the performance of the EA: for test cases mkp1, mkp2, mkp7, and mkp8, the number of hits increased to 6, 2, 10, and 10, respectively. These values are similar to those reported in column $(\alpha, \beta) = (1, 5)$ of Table 1. However, on the harder set of 20 test cases the performance of the EA was much worse than the performance of AS.

[3] The name *m.n-q* of an instance indicates: the number of constraints m, variables n, and a sequence number q.

Table 4: *The results of EA for 11 test cases of MKP*

Instance	Best known	Best Found	#hits	#gen(avg)
mkp1	7772	7772	4	1736.2
mkp2	8722	8695	0	2069.3
mkp3	141278	141278	10	365.4
mkp4	130833	130833	10	2069.3
mkp5	95677	95677	10	430.5
mkp6	119337	119337	10	850.5
mkp7	98796	98796	6	40.3
mkp8	130623	130623	8	341.1
mkp9	1095445	1092626	0	4032.0
mkp10	624319	603801	0	2385.2
mkp11	4554	4554	10	585.3

Table 5 displays the results of EA on 20 harder instances of MKP. It shows for each instance the known optimum [2], the best values found by AS and EA (out of 10 runs), the (rounded) number of cycles for AS required for finding the best solution, the number of hits (which is always zero) and the average generation number for EA required for finding the best solution.

Table 5: *The results of EA for additional test cases of MKP*

Instance	Best known	Best Found	#hits	#gen(avg)
5.100-00	24381	23626	0	5203.7
5.100-01	24274	23504	0	4307.3
5.100-02	23551	22628	0	3381.3
5.100-03	23534	23223	0	3381.3
5.100-04	23991	23427	0	5038.1
5.100-05	24613	23593	0	6183.1
5.100-06	25591	24506	0	5903.0
5.100-07	23410	22727	0	2836.2
5.100-08	24216	23262	0	4703.3
5.100-09	24411	23539	0	3908.2
10.100-00	23064	22747	0	3428.0
10.100-01	22801	21755	0	3117.5
10.100-02	22131	21114	0	3674.3
10.100-03	22772	21867	0	3410.0
10.100-04	22751	21784	0	3465.6
10.100-05	22777	22101	0	2879.1
10.100-06	21875	21481	0	4381.2
10.100-07	22635	21916	0	3952.0
10.100-08	22511	21726	0	2353.2
10.100-09	22702	21737	0	2522.3

Comparing the results given in Tables 3 and 5 it is clear that the AS performs better than EA on selected instances of the MKP. It should be pointed out, however, that a parameter tuning was done for the Ant System (selection of the best parameters α and β), whereas EA was run with a standard set up. Additionally, EA incorporated a particular constraint-handling technique based on a decoder. It would be interesting to check other constraint-handling techniques for EA, like penalty methods or repair algorithms, as well as different representations (binary) and different operators.

It is difficult to compare running times of both algorithms as the Ant System was designed to run in the Parallel Virtual Machine environment. However, running both algorithms in a serial environment (on Sun Ultra 1) we observed that 100 cycles of AS took approximately one third of a time required by 10,000 generations of EA.

6 Conclusions

In this paper we presented a new version of Ant System extended to handle subset problems. In the proposed version of the system, pheromone trail is put on the problem's components instead of the problem's connections. The AS performed very well on several instances of multiple knapsack problem. It outperformed a standard evolutionary algorithm on harder instances of the problem. The results indicate the potential of the ACO approach for solving constrained subset problems.

In the full version of the paper [8] we report the results of this new version of Ant System on other subset problems: the set covering problem and maximum independent set problem. We also look at various local heuristics which can be used for constructing solutions and examine their influence on the performance of Ant System.

Acknowledgments

The authors would like to thank Marco Dorigo for his valuable comments on the earlier draft of this paper.

Bibliography

[1] Beasley, J., "OR-Library: Distributing Test Problems by Electronic Mail"; e-mail: o.rlibrary@ic.ac.uk.

[2] Chu, P. and Beasley, J. (1998). "A Genetic Algorithm for the Multi-constraint Knapsack Problem", Journal of Heuristics, Vol.4, pp.63-86.

[3] Dorigo M. and G. Di Caro (1999). "The Ant Colony Optimization Meta-Heuristic". In D. Corne, M. Dorigo and F. Glover (eds), New Ideas in Optimization. McGraw-Hill, 1999. (Also available as: Tech. Rep. IRIDIA/99-1, Université Libre de Bruxelles, Belgium.)

[4] Dorigo M. and L.M. Gambardella (1997). "Ant Colony System: A Cooperative Learning Approach to the Traveling Salesman Problem". IEEE Transactions on Evolutionary Computation, Vol.1, No.1, pp.53-66.

[5] Dorigo M., V. Maniezzo, and A. Colorni (1991). "Positive feedback as a search strategy". Tech. Rep. No. 91-016, Politecnico di Milano, Italy.

[6] Leguizamón, G.; Crespo, M.L.; Kavka, C, and Cena, M. (1997). "The Ant Colony Metaphor for Multiple Knapsack Problem". Proceedings of the 3th Congreso Argentino en Ciencias de la Computacion. pp. 1080-1090. La Plata, Argentina. October 1997.

[7] Leguizamón, G.; Crespo, M.L.; Kavka, C. and Cena, M. (1998). "A Study of Performance of an Ant Colony System applied to Multiple Knapsack Problem". Proceedings of EIS-98, E. Apayd (ed), pp. 567-573. ICSC Academic.

[8] Leguizamón, G. and Michalewicz, Z. (1999). "Ant Systems for Subset Problems", in preparation.

[9] Khuri, S.; Bäck, T. and Heitkötter, J. (1994). "The Zero/One Multiple Knapsack Problem and Genetic Algorithms". Proceedings of the 1994 ACM Symposium on Applied Computing, E.Deaton and D.Oppenheim and J.Urban and H.Berghel (eds), pp.188-193. ACM Press, New York.

[10] Wilkinson, B. and Allen, M. (1997). "Parallel Programming Techniques and Application Using Networked Workstations". Preliminary Draft. Prentice Hall.