A Comparison between ACO Algorithms for the Set Covering Problem

Lucas Lessing¹, Irina Dumitrescu², and Thomas Stützle¹

¹ Intellectics Group, Darmstadt University of Technology
² Canada Research Chair in Distribution Management, HEC Montreal

Abstract. In this paper we present a study of several Ant Colony Optimization (ACO) algorithms for the Set Covering Problem. In our computational study we emphasize the influence of different ways of defining the heuristic information on the performance of the ACO algorithms. Finally, we show that the best performing ACO algorithms we implemented, when combined with a fine-tuned local search procedure, reach excellent performance on a set of well known benchmark instances.

1 Introduction

The Set Covering Problem (SCP) is an $\mathcal{NP}$-hard combinatorial optimization problem that arises in a large variety of practical applications, for example in airline crew scheduling or vehicle routing and facility placement problems [9, 12, 18]. Two Ant Colony Optimization (ACO) approaches for the SCP based on Ant System have been proposed so far [8, 10]; however, their computational performance is relatively poor compared to the state-of-the-art in SCP solving.

In this paper we present the results of a computational study comprising several ACO algorithms including $\text{MAX-MIN}$ Ant System (MMAS) [16], Ant Colony System (ACS) [3], a hybrid between MMAS and ACS [17], as well as Approximate Nondeterministic Tree-Search (ANTS) [14]. Many of the best ACO computational results have been achieved using these algorithms and therefore they serve as good candidates for tackling the SCP. (For details on the full study we refer to [11].) We study the influence of different ways of defining the heuristic information on the performance of the algorithms. We distinguish between the usage of static and dynamic heuristic information. In the static case, the heuristic information is computed when initializing the algorithm and it remains the same throughout the whole run of the algorithm. In the dynamic case, the heuristic information depends on the partial solution available, hence it has to be computed at each step of an ant’s walk. Therefore, it results in a higher computational cost that may be compensated by the higher accuracy of the heuristic values.

The paper is structured as follows. In the next section we introduce the SCP and lower bounds to it. In Section 3, we present some details on the ACO algorithms we applied, while in Sections 4 and 5 we give details on the local search and the heuristic information. We present experimental results in Section 6 and conclude in Section 7.
2 Set Covering Problem

The SCP can be formulated as follows. Let $A = (a_{ij})$ be an $m \times n$ 0-1 matrix and $c = (c_j)$ a positive integer $n$-dimensional vector, where each element $c_j$ of $c$ gives the cost of selecting column $j$ of matrix $A$. We say that row $i$ is covered by column $j$, if $a_{ij}$ is equal to 1. The SCP consists of finding a subset of columns of minimal total cost such that all the rows are covered. The SCP is usually formulated as an integer program. Let $N = \{1, \ldots, n\}$ be the index set of the columns and $M = \{1, \ldots, m\}$ be the index set of the rows. The integer programming (IP) formulation of the SCP can be given as follows:

$$z_{SCP} = \min \sum_{j \in N} c_j x_j$$

s.t. $\sum_{i \in M} a_{ij} x_j \geq 1, \forall i \in M$  

(1)  

$$x_j \in \{0,1\}, \quad \forall j \in N.$$  

(2)

We now introduce the Lagrangean relaxation of this problem. For every vector $u = (u_1, \ldots, u_m) \in \mathbb{R}^m_+$ we define the Lagrangean relaxation to be

$$z_{LR}(u) = \min_{x \in \{0,1\}^m} \left[ \sum_{j \in N} c_j x_j + \sum_{i \in M} u_i (1 - \sum_{j \in N} a_{ij} x_j) \right]$$

(3)

$$= \min_{x \in \{0,1\}^m} \left[ \sum_{j \in N} c_j (u) x_j + \sum_{i \in M} u_i \right],$$

(4)

where $c_j(u) = c_j - \sum_{i \in M} a_{ij} u_i$ are called Lagrangean costs. The Lagrangean relaxation provides a lower bound on the optimal solution of the SCP, i.e. $z_{LR}(u) \leq z_{SCP}, \forall u \geq 0$. The best such lower bound on the optimal solution of the SCP is obtained by solving the Lagrangean dual problem, $z_{LD} = \max_{u \geq 0} z_{LR}(u)$.

In our experiments, the Lagrangean dual was solved by the subgradient method [7], stopped after at most 100 $\cdot$ m iterations as proposed by Yagiura et al. [19].

When we consider the IP formulation, a solution to the SCP is a 0-1 vector. However, when we refer to a solution, we sometimes refer to the set of indices of the variables fixed to one. This should be clear from the context.

3 Ant Colony Optimization for the SCP

Artificial ants in ACO algorithms can be seen as probabilistic construction heuristics that generate solutions iteratively, taking into account accumulated past search experience: pheromone trails and heuristic information on the instance under solution. In the SCP case, each column $j$ has associated a pheromone trail $\tau_j$ that indicates the learned desirability of including column $j$ into an ants’ solution; $\eta_j$ indicates the heuristic desirability of choosing column $j$. 

procedure \texttt{ACOforSCP}
\begin{verbatim}
initializeParameters;
\textbf{while} termination condition is not true do
\textbf{for} \( k := 1 \) to \( m_a \) do
  \textbf{while} solution not complete do
    applyConstructionStep\((k)\);
  endwhile
  eliminateRedundantColumns\((k)\);
  applyLocalSearch\((k)\);
endfor
updateStatistics;
updatePheromones;
endwhile
return best solution found
endprocedure \texttt{ACOforSCP}
\end{verbatim}

Fig. 1. High-level view of the applied ACO algorithms. (For details, see text.)

In all ACO algorithms an ant starts with an empty solution and constructs a complete solution by iteratively adding columns until all rows are covered. It does so by probabilistically preferring solution components (columns in the SCP case) with high associated pheromone trail and/or heuristic value. Once \( m_a \) solutions are constructed and improved by local search, the pheromone trails are updated.

The application of ACO algorithms to the SCP differs from applications to other problems, such as the TSP. First, the solution construction of the individual ants does not necessarily end after the same number of steps for each ant, but only when a cover is completed. Second, the order in which columns are added to a solution does not matter, while in many other applications the order in which solution components are added to a partial solution may be important. Third, in the case of the SCP the solution constructed by the ants may contain redundant solution components which are eliminated before fine-tuning by a local search procedure. This latter feature is also present in two earlier applications of ACO algorithms to the SCP [8, 10]. However, the computational results obtained in [8, 10] are relatively poor, which may be due to the type of ACO algorithm chosen (i.e., Ant System, which is known to perform poorly compared to more recent variants of ACO algorithms). Here, we present the application of more recent ACO variants that exploit a large variety of different types of heuristic information. All these algorithms follow the algorithmic outline given in Figure 1.

3.1 \texttt{MAX-MIN} Ant System

\texttt{MAX-MIN} Ant System (\texttt{MMAS}) [16] is at the core of many successful ACO applications. In \texttt{MMAS} solutions are constructed as follows: an ant \( k \) \((k = 1, \ldots, m_a)\) chooses column \( j \) with probability

\[
p_j^k = \begin{cases} \frac{\tau_j^k}{\sum_{S_k \in S} \tau_j^k}, & \text{if } j \notin S_k, \\ 0, & \text{otherwise}, \end{cases}
\]

\[(5)\]
where the parameter $\beta \geq 0$ determines the relative influence of the heuristic information with respect to the pheromone and $S_k$ is the partial solution of ant $k$ (note that here we assume that $S_k \subseteq N$). Once all solutions are completed, the pheromone values are updated by first evaporating all pheromone trails, i.e., by setting $\tau_j := (1 - \rho) \cdot \tau_j$, $\forall j \in N$, then by adding an amount of $\Delta \tau = 1/z$ to the columns contained in the best-so-far (i.e., the best since the start of the algorithm) or the iteration-best (i.e., the best found in the current iteration) solution, where $z$ is the cost of the solution used in the pheromone update.

As usual in $\mathcal{M}\mathcal{M}\mathcal{A}\mathcal{S}$, the range of feasible pheromone values is limited to an interval $[\tau_{\text{min}}, \tau_{\text{max}}]$ to avoid premature stagnation of the search and pheromones are initialized to $\tau_{\text{max}}$. To further increase the exploration of solutions with columns that have only a small probability of being chosen, the pheromone trails are occasionally re-initialized. The re-initialization is triggered when no improved solution is found for a given number of iterations.

### 3.2 Ant Colony System

The second ACO algorithm we tested is Ant Colony Systems (ACS) [3]. ACS exploits the so-called pseudo-random proportional action choice rule in the solution construction: ant $k$ chooses the next column to be

$$ j = \begin{cases} 
\arg\max_{q \in S_k} \{\eta[q]^\beta\}, & \text{if } q \leq q_0 \\
\text{draw}(J), & \text{otherwise}
\end{cases} $$

(6)

where $q$ is a random number uniformly distributed in $[0, 1]$, $q_0$ ($0 \leq q_0 \leq 1$) is a parameter that controls how strongly the ants exploit deterministically the combined past search experience and heuristic information, and $\text{draw}(J)$ is a random number generated according to the probability distribution defined in Equation 5.

In ACS, ants modify the pheromone trails also while constructing a solution. Immediately after ant $k$’s adding of a column $j$ to its partial solution $S_k$, the ant modifies $\tau_j$ according to: $\tau_j := (1 - \xi) \tau_j + \xi \cdot \bar{\tau}_j$, where $\xi$, $0 \leq \xi \leq 1$, is a parameter and $\bar{\tau}_j = 1/(n \cdot z_{\text{GR}})$, (where $z_{\text{GR}}$ is the cost of a greedy solution), is the initial value of the pheromone trails. The local pheromone update has the effect that each time a column is chosen, it is made less attractive for the other ants, thus increasing exploration.

Once all ants have constructed their solutions, pheromones are deposited according to $\tau_j := (1 - \rho) \tau_j + \rho \cdot \Delta \tau^*_j$, $\forall j \in S^*$, where $S^*$ is the best-so-far solution and $\Delta \tau^*_j = 1/z^*$, where $z^*$ is the cost of $S^*$.

### 3.3 $\mathcal{M}\mathcal{M}\mathcal{A}\mathcal{S}$–ACS–Hybrid

$\mathcal{M}\mathcal{M}\mathcal{A}\mathcal{S}$–ACS–Hybrid is an ACO algorithm that follows the rules of $\mathcal{M}\mathcal{M}\mathcal{A}\mathcal{S}$ but uses the pseudo-random proportional rule of ACS to determine the next column to be added to an ants’ partial solution. Because of this aggressive construction rule, the hybrid uses generally much tighter pheromone trail limits than $\mathcal{M}\mathcal{M}\mathcal{A}\mathcal{S}$. $\mathcal{M}\mathcal{M}\mathcal{A}\mathcal{S}$–ACS–Hybrid has shown promising performance in [13, 17].
3.4 Approximate Nondeterministic Tree Search

Approximate Nondeterministic Tree Search (ANTS) is a recent ACO algorithm developed by Maniezzo [14]. The main innovative feature of ANTS is the use of lower bounds on the completion of a solution as the heuristic information used in the solution construction (see also Section 5.2). The probability of choosing the next column \( j \) to be added by ant \( k \) is given by:

\[
p_j^k = \frac{\zeta r_j + (1 - \zeta)\eta_j}{\sum_{i \in S_k} [\zeta r_i + (1 - \zeta)\eta_i]}, \quad \text{if } j \notin S_k,
\]

where \( 0 \leq \zeta \leq 1 \) is a parameter that determines the influence of the pheromone trails versus the heuristic information. The contribution of all ants is considered in the pheromone trail update; the amount \( \Delta r_j^k \) is given by:

\[
\Delta r_j^k := \begin{cases} 
\vartheta \left( 1 - \frac{\zeta(S_h)-LB}{L_{avg} - LB} \right), & \text{if } j \in S_k \\
0, & \text{otherwise},
\end{cases}
\]

where \( \vartheta \geq 0, LB \) is a lower bound on the optimal solution value, and \( L_{avg} \) is a moving average of the solution qualities of recently constructed solutions; care is taken to avoid that pheromones become negative. If an ant’s solution is worse than the current average, the pheromone trail of the columns in the solution is decreased; if the ant’s solution is better, the pheromone trail is increased.

4 Local Search: r-Flip

In most applications of ACO to combinatorial optimization problems improving the solutions constructed by the ants in a local search phase [4, 5, 16] was worth doing. As our computational results will show, this is also the case for many SCP instances. We use an efficient local search, based on the r-flip neighborhood. The r-flip neighborhood of a solution \( x = (x_1, \ldots, x_n) \) consists of all the solutions that can be obtained by flipping (i.e., removing a column from or adding a new column to the solution) at most \( r \) variables in the solution; this means that a solution \( x' = (x'_1, \ldots, x'_n) \) is a neighbor of \( x \) if the Hamming-Distance \( d(x, x') = \sum_{i \in \{1, \ldots, n\}} |x_i \neq x'_i| \) between \( x \) and \( x' \) is at most \( r \). An efficient implementation of a local search algorithm based on the 3-flip is given by Yagiura et al., [19]. The code of this local search algorithm has been provided to us by M. Yagiura; we adapted and used it to the ACO algorithms described above.

5 Heuristic Information

It is widely acknowledged that the heuristic information can play a significant role in the performance of ACO algorithms. However, there are several possible ways of defining heuristic information. The types of heuristic information used are static or dynamic. In the static case, the heuristic information can be
computed only once, when initializing the algorithm. Then it remains the same throughout the whole run of the algorithm. In the dynamic case, the heuristic information depends on the partial solution constructed and it is computed at each construction step of each ant (the partial solutions between the ants differ). The use of dynamic heuristic information usually results in higher computational costs but typically also in higher accuracy of the computed heuristic values. Next we explain the types of heuristic information we considered in our study.

5.1 Static heuristic information

**Column Costs:** A very simple heuristic is to use the column costs for the heuristic information and to set \( \eta_j = 1/c_j \).

**Lagrangian Costs:** Instead of the column costs, the Lagrangian costs associated to each column may be used to define the heuristic information. In this case, since the probability of selecting a column cannot be negative, the Lagrangian costs need to be normalized. Therefore, let the normalized Lagrangian costs be: \( C_j = c_j(u) - \min_{h \in N} c_h(u) + \epsilon \), where \( \epsilon \) is a small positive number, \( c_j(u) \) are the Lagrangian costs, and the vector \( u \) considered is the one obtained at the end of the subgradient procedure. The heuristic information in this case is: \( \eta_j = 1/C_j \).

5.2 Dynamic heuristic information

**Cover Costs:** This is the most commonly used heuristic information in the previously defined ACOs for the SCP. The cover cost of a column \( j \) are defined as \( c_j/\text{card}_j(S) \), where \( \text{card}_j(S) \) is the number of rows covered by column \( j \), but not covered by any column in the partial solution \( S \). Hence, the heuristic information can be defined as:

\[
\eta_j = \frac{\text{card}_j(S)}{c_j} \quad \forall j \notin S.
\]  

**Lagrangian Cover Costs:** In the definition of the cover costs it is possible to use the normalized Lagrangian costs \( C_j \) instead of the column costs \( c_j \); the resulting cost values are called Lagrangian cover costs. The heuristic information is then obtained by replacing \( c_j \) in Equation 9 by \( C_j \).

**Marchiori & Steenbeek Cover Costs:** In their Iterated Greedy algorithm [15] Marchiori and Steenbeek propose a variant of the cover costs. Let \( S \) be a current (partial) solution and \( \text{cov}(S) \) the set of rows covered by the columns in \( S \); \( \text{cov}(j, S) \) is the set of rows covered by column \( j \), but not covered by any column in \( S \setminus \{j\} \). Let \( c_{min}(i) \) be the minimum cost of all columns that cover row \( i \). The cover value \( cv \) of a column \( j \) with respect to a (partial) solution \( S \) is

\[
\text{cv}(j, S) = \sum_{i \in \text{cov}(j, S)} c_{min}(i).
\]  

Note that if \( \text{cv}(j, S) = 0 \), then column \( j \) is redundant with respect to \( S \). This cover value is used to define the modified cover costs \( \text{cov.val}(j, S) \) as

\[
\text{cov.val}(j, S) = \begin{cases} 
\infty, & \text{if } \text{cv}(j, S) = 0 \\
\frac{c_j}{\text{cv}(j, S)}, & \text{otherwise}.
\end{cases}
\]
The heuristic information is then again given by \( \eta_j = 1/\text{conv. rank}(j, S) \).

**Marchiori & Steenbeck Lagrangean Cover Costs with Normalized Costs:** The column costs \( c_j \) used to define the cover costs of Marchiori & Steenbeck can be replaced by the normalized Lagrangean costs \( C_j \); the resulting cost values are called Marchiori & Steenbeck Lagrangean cover costs and the heuristic information is obtained like in the previous case.

**Lower Bounds:** In this case we consider that we have a partial solution available. Let \( N_1, N_0 \subseteq N \), where \( N_1 \) is the index set of the variables already fixed to 1 and \( N_0 \) the index set of the variables fixed to 0, \( N_0 \cap N_1 = \emptyset \); the set \( N_0 \) is obtained, e.g., in a preprocessing stage that eliminates redundant columns. Let 
\[
N_{\text{free}} = N \setminus (N_0 \cup N_1),
\]
the index set of the variables that are still free, and let
\[
M' = \{ i \in M : \sum_{j \in N_i} a_{ij} = 0 \},
\]
the set of rows that are not yet covered. Since our goal is to complete the partial solution to a full one as well as we can, we need to solve a reduced SCP, considering only the variables that are still free.

However, the reduced problem might still be difficult to solve, especially if the number of fixed variables is not very large. Therefore, we will not attempt to solve it directly; instead we will calculate lower bounds on the solution of the reduced problem. The Lagrangean relaxation of the reduced SCP is
\[
\bar{z}_{\text{SCP}(N_{\text{free}})}(u) = \min_{x \in \{0,1\}^{N_{\text{free}}}} \sum_{j \in N_{\text{free}}} \bar{c}_j(u)x_j + \sum_{i \in M'} u_i,
\]
where \( \bar{c}_j(u) = c_j - \sum_{i \in M', a_{ij}u_i} \) are the Lagrangean costs of the reduced problem.

The Lagrangean relaxation of the reduced problem gives a lower bound on the optimal solution of the reduced problem, i.e.,
\[
\bar{z}_{\text{SCP}(N_{\text{free}})}(u) \leq \bar{z}_{\text{SCP}(N_{\text{free}})},
\]
for any \( u \geq 0 \). Since our aim is to use information that is already available, we calculate the lower bound using the value of \( u \) output by the subgradient procedure already run for the full SCP, thus, avoiding to run the subgradient procedure many times.

The heuristic information for column \( j \) is inversely proportional to the lower bound obtained by tentatively adding this column.

6 Tests and Results

We performed an experimental analysis of the ACO algorithms presented, with emphasis on the study of the influence of the heuristic information on the final performance. We ran each of the ACO algorithms (M-MAS, ACS, M-MAS-ACS-Hybrid, ANTS) together with each of the seven different versions of obtaining heuristic information. The algorithms were implemented in C++ and compiled with the GNU-C-Compiler using -O2 optimization settings. All the experiments were executed on a Intel 2.4 GHz Xeon processor with 2 GB of RAM.

The algorithms were tested on a set of benchmark instances available from ORLIB. These instances are randomly generated instances, divided into 12 subsets identified as SCP4, SCP5, SCP6, SCPA, SCPB, SCPC, SCPD, SCPE, SCPNRE, SCPRF, SCPNRG, SCPNRH. SCP4 and SCP5 have 10 instances each; all other classes have 5 instances. The instances differ in size and density of the matrices. In addition, we also tested the algorithms on the instances of the FASTER-com petition,
which stem from a real-world application of the SCP. The instance identifiers are RAIL507, RAIL516, RAIL592, RAIL2536, RAIL2586, RAIL4284, and RAIL4872, where the number in the identifier gives the number of rows; the number of columns in these instances ranges from 47,311 to 1,092,610.

In preliminary experiments we tried to find reasonable parameter settings for each of the algorithm–heuristic information combinations. This was done by considering a set of parameters for each algorithm and then modifying one, while keeping the others fixed. The parameters tested include $m_a \in \{1, 5, 10, 20\}$, $\beta \in \{1, 3, 5\}$, $q_0 \in \{0.9, 0.95, 0.98, 0.99\}$, $\rho \in \{0.2, 0.5, 0.8\}$ (for ACS $\rho \in \{0.1, 0.2, 0.3\}$), $\xi \in \{0.1, 0.2, 0.3\}$, $\zeta \in \{0.0, 0.3, 0.5\}$, and $\theta \in \{0.1, 0.2, 0.3\}$ (the parameters apply only to the algorithms where they are actually used). We found that for $m_a = 5$, $\beta = 5$, and $\rho = 0.2$ the algorithms performed best; $q_0$ is set to 0.98, $\xi$ to 0.2 in the algorithms that use these parameters; for ANTS the parameters are set $\zeta = 0.3$ and $\theta = 0.5$. In addition, each of the algorithms was tested with and without $r$-flip local search. The $r$-flip local search was stopped after a maximum of 200 iterations. Each algorithm was run 10 times with a computation time limit of 100 seconds for the ORLIB instances; for the large FASTER instances (RAIL2536 and larger) 500 seconds were allowed.

Figure 2 plots the development of the objective function values over computation times for ACS with the seven different types of heuristic information, without local search (without local search, ACS showed best performance with
respect to the number of best known solutions found — see also Table 1 — and therefore we chose it here). It can be observed that dynamic heuristic information based on cover costs generally yields the best time-quality tradeoff with the cover costs defined by Marchiori & Steenbeck typically performing best. (In the latter case, it does not really matter whether the column costs or the Lagrangian costs are used; however, this makes a difference when using the standard cover costs). Static heuristic information typically shows quite poor performance with the only exception being the column cost on instance SCP-NRR2. Hence, the higher computational cost of the dynamic heuristic information appears to be well spent. A notable exception is the usage of the lower bound information while constructing solutions, which typically results in rather poor performance. However, the ranking of the heuristic information is completely reversed when using local search, as shown in Figure 3. Here, the heuristic information based on the lower bounds (used in the ANTS algorithm) gives by far the best performance. Additionally, it is quite visible that on most of the instances the performance is rather similar, with the notable exception of the version using lower bounds as the heuristic information. Interestingly, on one of the largest instances, SCP-NRR2, the heuristic information based on column costs is second best. Given that without local search these latter two kinds of heuristic information showed rather poor performance, one conclusion from these results may be that once a powerful local search such as the r-flip algorithm is available, it may be better that the starting solutions for the local search are not too good to leave the local search some room for improvement.

Table 1 shows for each possible ACO algorithm–heuristic information combination the total number of optimal solutions found and the sum of the ratios between the optimal (or best known) solution divided by the solution quality returned in each trial for the ORLIB-instances (proven optimal solutions are known for instances SCP4 through SCPE). The observations made above are confirmed. The combination of ANTS plus lower bound based heuristic information obtained in all 700 trials the best known or optimal solutions. The average computation times for classes SCP4 through SCPE are on average always less than one CPU second, while the hardest instance took on average 23.3 seconds. We note that this result slightly improves over that of the iterated r-flip heuristic of Yagiura et al. [19], which finds the best known solutions in 694 trials. Table 2 shows the same information for the FASTER instances. Here, occasionally, even the versions without local search perform better than the variants without r-flip. However, in part, this may be due to an inappropriate termination condition for the local search. In any case, on these instances the iterated r-flip algorithm of Yagiura et al. performs significantly better than the ants.

We tested ANTS plus LB heuristic information on a set of 16 benchmark instances taken from [1]; except for one instance, it found the optimal solutions for all instances, in every trial. The average computation times were typically below two seconds and only two instances took slightly longer (five and eight seconds, on average). Compared to other state-of-the-art algorithms than the iterated r-flip, our best performing variant appears to be competitive with the
Fig. 3. Development of the average objective function value found by the best combination between ACO algorithm, local search, and heuristic information. Top left for instance SCPWE2, top right SCPWE5, bottom left SCPWE2, and bottom right SCPWE2.

iterated greedy algorithm of Marchiori and Steenbeek [15]. The CFT heuristic of Caprara, Fischetti and Toth appears to have similar performance on the ORLIB instances, but it performs much better on the FASTER instances. Our best variant also outperforms earlier evolutionary algorithms like those presented in [2,6] and two ACO algorithms based on Ant System that were proposed earlier in the literature [8,10].

7 Conclusions

In this paper we studied the behavior of various ACO algorithms for the SCP and, in particular, the influence of various ways of defining the heuristic information on the final performance. When applying the ACO algorithms without local search, we noted that (i) for a fixed ACO algorithm, the heuristic information has significant influence on the final performance and that typically the best performance is reached by using dynamic heuristic information based on cover costs or extensions thereof proposed by Marchiori and Steenbeek; (ii) the overall best performance with respect to the number of best known solutions found was obtained by ACS, which suggests that a rather aggressive construction policy, as implemented by ACS, is needed to achieve good performance. However, if local search is used, the heuristic information based on lower bound information,
Table 1. Summary statistics for the final computational results for 70 randomly generated ORLIB instances across 10 trials per instance. Given is $n_{opt}$, the number of best known (or optimal) solutions found in the 700 ($= 70 \cdot 10$) trials and rel, the sum of the ratios of best known solution value divided by the solution quality returned by each trial.

\[
\begin{array}{cccccccc}
 & c_j & n_{opt} & rel & cc & n_{opt} & rel & LR-c & n_{opt} & rel & LR-cc & n_{opt} & rel & MS-cc & n_{opt} & rel & MS-ccLR & n_{opt} & rel & LB & n_{opt} & rel \\
\hline
M-MAS & 309.691.3 & 492.696.4 & 301.685.3 & 253.680.4 & 204.696.5 & 492.696.4 & 210.669.9 & \\
ACS & 311.698.7 & 511.696.3 & 328.688.2 & 331.687.9 & 556.696.4 & 498.695.6 & 419.699.9 & \\
M-MAS-H & 192.681.0 & 496.695.4 & 230.680.4 & 238.681.9 & 515.695.9 & 475.694.8 & 194.675.8 & \\
ANTS & 224.687.1 & 390.693.0 & 272.683.0 & 242.681.7 & 399.693.2 & 379.692.3 & 191.674.2 & \\
\hline
\end{array}
\]

which performs as one of the 'worst' without local search, gives the overall best results in combination with the ANTS algorithm.

Comparisons with other algorithms have shown that on various instance classes, the best variants we tested reach state-of-the-art performance. However, this is not true for the FASTER benchmark set that stems from real-world SCP applications. However, experiments with optimized parameters for this instance class indicate that there is room for considerable improvement.

Acknowledgments This work was supported by a European Community Marie Curie Fellowship, contract HPMF-CT-2001.

References

Table 2. Summary statistics for the final computational results for the seven FASTER instances. For details see caption of Table 1.

<table>
<thead>
<tr>
<th></th>
<th>$c_j$</th>
<th>$\eta_{\text{opt}}$</th>
<th>$\eta_{\text{opt}}$</th>
<th>$\eta_{\text{opt}}$</th>
<th>$\eta_{\text{opt}}$</th>
<th>$\eta_{\text{opt}}$</th>
<th>$\eta_{\text{opt}}$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$\alpha$</td>
<td>rel</td>
<td>rel</td>
<td>rel</td>
<td>rel</td>
<td>rel</td>
<td>rel</td>
</tr>
<tr>
<td></td>
<td>Without local search</td>
<td>LR-c</td>
<td>LR-cc</td>
<td>MS-cc</td>
<td>MS-ccLR</td>
<td>LB</td>
<td></td>
</tr>
<tr>
<td>MMAS</td>
<td>0</td>
<td>0.4</td>
<td>0.6</td>
<td>0.6</td>
<td>0.4</td>
<td>0.6</td>
<td>0.7</td>
</tr>
<tr>
<td>ACS</td>
<td>0</td>
<td>0.9</td>
<td>0.6</td>
<td>0.6</td>
<td>0.6</td>
<td>0.6</td>
<td>0.6</td>
</tr>
<tr>
<td>MMAS-H</td>
<td>0</td>
<td>0.9</td>
<td>0.6</td>
<td>0.6</td>
<td>0.6</td>
<td>0.6</td>
<td>0.6</td>
</tr>
<tr>
<td>ANTS</td>
<td>0</td>
<td>3.8</td>
<td>0.6</td>
<td>0.6</td>
<td>0.6</td>
<td>0.6</td>
<td>0.6</td>
</tr>
</tbody>
</table>

With 7-Flip

<table>
<thead>
<tr>
<th></th>
<th>$c_j$</th>
<th>$\eta_{\text{opt}}$</th>
<th>$\eta_{\text{opt}}$</th>
<th>$\eta_{\text{opt}}$</th>
<th>$\eta_{\text{opt}}$</th>
<th>$\eta_{\text{opt}}$</th>
<th>$\eta_{\text{opt}}$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$\alpha$</td>
<td>rel</td>
<td>rel</td>
<td>rel</td>
<td>rel</td>
<td>rel</td>
<td>rel</td>
</tr>
<tr>
<td></td>
<td>Without local search</td>
<td>LR-c</td>
<td>LR-cc</td>
<td>MS-cc</td>
<td>MS-ccLR</td>
<td>LB</td>
<td></td>
</tr>
<tr>
<td>MMAS</td>
<td>3</td>
<td>3.7</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>ACS</td>
<td>0</td>
<td>3.8</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>MMAS-H</td>
<td>0</td>
<td>3.8</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>ANTS</td>
<td>5</td>
<td>3.9</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>