Proceedings of the 2007 IEEE Swarm Intelligence Symposium (SIS 2007)

ANT COLONY SYSTEMS FOR LARGE SEQUENTIAL ORDERING PROBLEMS

R. Montemanni [a,c], *D.H. Smith* [b], *L.M. Gambardella* [a]

[a] Istituto Dalle Molle di Studi sull'Intelligenza Artificiale (IDSIA)
Galleria 2, CH-6928 Manno-Lugano, Switzerland
{roberto, luca}@idsia.ch

[b] Division of Mathematics and Statistics, University of Glamorgan
Pontypridd CF37 1DL, United Kingdom
dhsmith@glam.ac.uk

ABSTRACT

The sequential ordering problem is a version of the asymmetric traveling salesman problem where precedence constraints on vertices are imposed. A tour is feasible if these constraints are respected, and the objective is to find a feasible solution with minimum cost.

The sequential ordering problem models a lot of real world applications, mainly in the fields of transportation and production planning.

In this paper we propose an extension of a well known ant colony system for the problem, aiming at making the approach more efficient on large problems. The extension is based on a problem manipulation technique that heuristically reduces the search space.

Computational results, where the extended ant colony system is compared to the original one, are presented.

KEYWORDS: Problem manipulation techniques, ant colony optimization, asymmetric traveling salesman, scheduling.

1. INTRODUCTION

The *Sequential Ordering Problem* (*SOP*), also referred to as *Asymmetric Traveling Salesman Problem with Precedence Constraints*, can be modeled in graph theoretical terms as follows. A complete directed graph $D = (V, A)$, where V is the set of nodes and $A = \{(i, j)|i, j \in V\}$ is the set of arcs, is given. A cost $c_{ij} \in \mathcal{N}$ is associated with each arc $(i, j) \in A$. Without loss of generality we assume that a fixed starting node $1 \in V$ is given. It has to precede all the other nodes. The tour is also closed at node 1, after all the other nodes have been visited. This to create an analogy with the asymmetric traveling salesman problem ($c_{i1} = 0 \; \forall i \in V$). Furthermore we are given an additional precedence digraph $P = (V, R)$, defined on the same node set V as D. An arc $(i, j) \in R$, represents a precedence relationship, i.e. i has to precede j in every feasible tour. We will denote such a relation as $i \prec j$ in the remainder of the paper. The precedence digraph P must be acyclic in order for a feasible solution to exist. We also assume it is transitively closed, since $i \prec k$ can be inferred from $i \prec j$ and $j \prec k$. Notice that for the last arc traversed by a tour (entering node 1), precedence constraints do not apply. A tour that satisfies precedence relationships is called *feasible*. The objective of the *SOP* is to find a feasible tour with the minimal total cost.

It is interesting to observe that *SOP* reduces to the classical asymmetric traveling salesman problem (*ATSP*) in the case where no precedence constraint is given. This observation implies that *SOP* is $\mathcal{NP}$-hard, being a generalization of the *ATSP*.

The *SOP* models real-world problems such as production planning (Escudero [7]), single vehicle routing problems with pick-up and delivery constraints (Pulleyblank and Timlin [13], Savelsbergh [14]) and transportation problems in flexible manufacturing systems (Ascheuer [1]).

Sequential ordering problems were initially solved as constrained versions of the *ATSP*, especially for the development of exact algorithms. The main effort has been put into extending the mathematical definition of the *ATSP* by introducing new classes of valid inequalities to model the additional constraints. The first mathematical model for the *SOP* was introduced in Ascheuer et al. [2] where a cutting plane approach was proposed to compute lower bounds on the optimal solution. In Escudero et al [8], a Lagrangean relaxation method was described and embedded into a branch and cut algorithm. Ascheuer [1] has proposed a new class of valid inequalities and has described a new

[c] Corresponding author, tel +41 58 666 666 7, fax +41 58 666 666 1.
Partially supported by the Swiss National Science Foundation through project 200020-109854/1.

branch-and-cut method for a broad class of *SOP* instances. This is based on the polyhedral investigation carried out on *ATSP* problems with precedence constraints by Balas et al. [3]. The approach in [1] also investigates the possibility to compute and improve sub-optimal feasible solutions starting from the upper bound provided by the polyhedral investigation. The upper bound is the initial solution of a heuristic phase based on well-known *ATSP* heuristics that are iteratively applied in order to improve feasible solutions. These heuristics do not handle constraints directly: infeasible solutions are simply rejected. A branch and bound algorithm with lower bounds obtained from homomorphic abstractions of the original search space has been presented in Hernàdvölgyi [11] (see also [12]). A genetic algorithm has been proposed in Chen and Smith [4]. The method works in the space of feasible solutions by introducing a sophisticated crossover operator that preserves the common schema of two parents by identifying their maximum partial order through matrix operations. The new solution is completed using constructive heuristics. A hybrid genetic algorithm based on complete graph representation has been discussed in Seo and Moon [15]. A parallelized roll-out algorithm has been described in Guerriero and Mancini [10]. Gambardella and Dorigo [9] presented an approach based on Ant Colony Optimization enriched with sophisticated local search procedures. This last method can be classified as state-of-the-art for the sequential ordering problem and will be described in detail in Section 2.

The contribution of the present article will be an extension of the method described in [9], aiming at improving the performance of the method on large problems. The novel idea at the basis of the extension is a problem manipulation technique.

2. ANT COLONY OPTIMIZATION

The *Ant Colony System* (*ACS*) algorithm is an element of the *Ant Colony Optimization* (*ACO*) family of methods (Dorigo et al. [5]). These algorithms are based on a computational paradigm inspired by real ant colonies and the way they function. The main underlying idea was to use several constructive computational agents (simulating real ants). A dynamic memory structure, which incorporates information on the effectiveness of previous choices, based on the obtained results, guides the construction process of each agent. The behavior of each single agent is therefore inspired by the behavior of real ants.

The paradigm is based on the observation, made by ethologists, that the medium used by ants to communicate information regarding shortest paths to food, consists of *pheromone trails*. A moving ant lays some pheromone on the ground, thus making a path by a trail of this substance. While an isolated ant moves practically at random (exploration), an ant encountering a previously laid trail can detect it and decide, with high probability, to follow it, thus reinforcing the trail with its own pheromone (exploitation). What emerges is a form of *autocatalytic* process where the more the ants follow a trail, the more attractive that trail becomes to be followed. The process is thus characterized by a positive feedback loop, where the probability with which an ant chooses a path increases with the number of ants that previously chose the same path. The mechanism above is the inspiration for the algorithms of the *ACO* family.

2.1. Ant Colony Optimization for the *SOP*

As said in the previous section, application of an *ACO* algorithm to a combinatorial optimization problem requires definition of a constructive algorithm and possibly a local search. Accordingly, a constructive algorithm called *ACS-SOP* in which a set of artificial ants builds feasible solutions to the *SOP* has been designed, together with a local search specialized for the *SOP* that takes these solutions to their local optimum. The resulting algorithm is a Hybrid Ant System for the *SOP* called *HAS-SOP*, which is described in detail in Gambardella and Dorigo [9].

*2.1.1. Construction phase (*ACS-SOP*)*

ACS-SOP is strongly based on the Ant Colony System algorithm (Dorigo and Gambardella [6]). *ACS-SOP* implements the constructive phase of *HAS-SOP*, and its goal is to build feasible solutions for the *SOP*. It generates feasible solutions with a computational cost of order O($|V|^2$).

Informally, *ACS-SOP* works as follows. Ants are sent out sequentially (not in parallel). Each ant iteratively starts from node 1 and adds new nodes until all nodes have been visited. When in node i, an ant applies a so-called transition rule, that is, it probabilistically chooses the next node j from the set $F(i)$ of feasible nodes. $F(i)$ contains all the nodes j still to be visited and such that all nodes that have to precede j, according to precedence constraints, have already been inserted in the sequence.

The ant in node i chooses the next node j to visit on the basis of two factors: the heuristic *desirability* η_{ij} here defined as $1/c_{ij}$, and the *pheromone trail* τ_{ij}, that contains a measure of how good it has been in the past to include arc (i, j) into a solution (it is the "memory" of the colony). The next node to visit is chosen with probability q_0 as the node j, $j \in F(i)$, for which the product $\tau_{ij} \cdot \eta_{ij}$ is highest (deterministic rule), while with probability $1 - q_0$ the node j is chosen with a probability given by

$$p_{ij_{j\in F(i)}} = \frac{\tau_{ij} \cdot \eta_{ij}}{\sum_{l\in F(i)}(\tau_{il} \cdot \eta_{il})}$$

(i.e., nodes connected by arcs with higher values of $\tau_{ij} \cdot \eta_{ij}$, $j \in F(i)$, have higher probability of being chosen).

The value q_0 is given by $q_0 = 1 - s/|V|$. The parameter s represents the number of nodes we would like to choose using the probabilistic transition rule, independently of the number of nodes of the problem.

In *ACS-SOP* only the best ant, that is the ant that built the shortest tour since the beginning of the computation, is allowed to deposit pheromone trail. The rationale is that in this way a "preferred route" is memorized in the pheromone trail matrix, and future ants will use this information to generate new solutions in a neighbourhood of this preferred route. If we refer to the shortest path generated since the beginning of the computation as $OptPath_{Best}$, and to its cost as L_{Best}, $\forall\{i,j\} \in OptPath_{Best}$, we have the following formula for pheromone update:

$$\tau_{ij} = (1-\rho)\cdot\tau_{ij} + \frac{\rho}{L_{best}} \quad (1)$$

Pheromone is also updated during solution building. In this case, however, it is removed from visited arcs. In other words, each ant, when moving from node i to node j, applies a pheromone updating rule that causes the amount of pheromone trail on arc (i,j) to decrease.

The rule is:

$$\tau_{ij} = (1-\psi)\cdot\tau_{ij} + \psi\cdot\tau_0 \quad (2)$$

where τ_0 is the initial value of trails. It was found that good values for the algorithm's parameters are $\tau_0 = ($*FirstSolution* $\cdot n)^{-1}$, $\rho = \psi = 0.1$, $s = 10$, where *FirstSolution* is the length of the shortest solution generated by the ant colony following the *ACS-SOP* algorithm without using the pheromone trails. Experience has shown these values to be robust. The number of ants in the population was set to 10. The rationale for using formula (2) is that it causes ants to eat away pheromone trail while they build solutions so that a certain variety in generated solutions is assured (if pheromone trail was not consumed by ants, they would tend to generate very similar tours).

2.1.2. *Complete algorithm (*HAS-SOP*)*

The *HAS-SOP* algorithm is the *ACS-SOP* algorithm augmented by local search. Local search is an optional component of *ACO* algorithms, although it has been shown since early implementations that it can greatly improve the overall performance of the *ACO* metaheuristic when static combinatorial optimization problems are considered (Dorigo and Gambardella [6]).

In *HAS-SOP*, local search is applied once each ant has built its solution: the solution is carried to its local optimum by an application of the extremely efficient *SOP-3-exchange* local search routine. This local search routine is a specialization to the sequential ordering problem of a known local search method for the asymmetric traveling salesman problem (Savelsbergh [14]). It is able to directly handle multiple constraints without increasing the computational complexity of the original local search. Since the description of such a local search method is out of the scope of this paper (although the local search routine is used by the algorithm we propose) we refer the interested reader to Gambardella and Dorigo [9] for its detailed description. Locally optimal solutions are then used to update pheromone trails on arcs, according to the pheromone trail update rule (1).

The algorithm stops when a fixed CPU time has elapsed.

3. ARTIFICIAL PRECEDENCE CONSTRAINTS

It is trivial to observe that adding precedence constraints to a given problem reduces its search space, making the problem potentially easier to solve. Starting from this observation, we developed the method described in the remainder of this section.

The underlying idea is to monitor the solutions generated by *HAS-SOP*, and to identify precedence patterns common to solutions with a low cost. Once such precedence patterns are identified, they can be added to the original problem as *artificial precedence constraints*. The resulting problem is likely to be easier than the original one, as it has a reduced solution space. Of course such a heuristic method may cut out all the optimal solutions of the original problem, leading to suboptimal solutions even in the case that the best solution of the modified problem is retrieved.

Formally, the methodology we propose is integrated into the *HAS-SOP* method and makes use of an additional set of variables m. Variable m_{ij} will be an indicator for the "quality" of the solutions in which node i is visited before node j. We also need the following additional parameters:

- u : number of solutions generated (ants sent out) before the first artificial precedence constraints are added to the problem;
- v : number of solutions generated (ants sent out) between two consecutive creations of artificial precedence constraints;
- w : (approximate) number of artificial precedence constraints added each time group of new artificial precedence constraints is generated;

The method we propose is integrated into the classical *HAS-SOP* algorithm as follows. We initialize $m_{ij} = 0 \ \forall(i,j) \in A$.

Each time a new solution $OptPath_k$, with cost L_k, is generated by an ant of the colony (and taken down to its local optimum), matrix m is updated as follows:

$$m_{ij} = m_{ij} + \frac{L_1}{L_k} \ \forall i,j \in V, \pi_k(i) < \pi_k(j) \le \pi_k(i) + z \quad (3)$$

$$m_{ji} = m_{ji} - \frac{L_1}{L_k} \; \forall i,j \in V, \pi_k(i) < \pi_k(j) \leq \pi_k(i) + z \quad (4)$$

where L_1 is the cost of the solution generated by the very first ant and $\pi_k(i)$ is the index of the position occupied by node i in solution $OptPath_k$. z is not a listed parameter (see next page). Its value regulates the width of the window considered for updates.

The first update (equation (3)) reinforces the entry corresponding to a sequence which is in solution $OptPath_k$. The update is proportional to the inverse of the cost of the solution itself. Equation (4) decreases the value on arcs that are traversed in the opposite direction in the current solution. This second update has been inserted to make those pairs of nodes that do not seem to have a clear ordering relationship less attractive. Notice that only pairs with a positive entry in matrix m will be potentially transformed into artificial precedence constraints.

We need now to briefly comment on z . Values of z that are too small might lead to a method where only arcs common to many solutions are identified, and not pairs of nodes that are in the same order (but not necessarily contiguous) in many solutions, as we would like. On the other hand, values of z that are too large might make the method too sensitive, and lead to a large number of negative entries in matrix m. Notwithstanding these considerations, we decided not to list z among the parameters because preliminary results suggest that the method is not sensitive at all to changes to (reasonable values of) z. In particular, values within the interval $[4, 10]$ seem to guarantee the best performance. In our method we set $z = 5$.

After the first u solutions are created by ants (and taken down to their local optimum), a first set of (approximately) w artificial precedence constraints are added to matrix P. The new constraints are selected as the ones not yet present in the precedence digraph P with the highest entries in matrix m, plus those implied by them by transitivity. If there are less than w entries of m with a positive value, then only the precedence constraints corresponding to them will be added to P (together with those implied by transitivity). The next set of artificial constraints will be added every time v new solutions have been generated, following the same logic.

The new hybrid ant colony system that makes use of *Artificial Precedence Constraints* will be referred to as *HAS-SOP*$_{APC}$. It is summarized by the pseudo-code of Figure 1.

After some preliminary tests, we identified two prominent (and promising) settings of the parameters of *HAS-SOP*$_{APC}$. These settings suggest the following two algorithms arising from method *HAS-SOP*$_{APC}$:

- ***HAS-SOP*$^{P}_{APC}$:** parameters are set as follows: $u = 20, v = \infty, w = 10$. This parameter setting leads to the configuration that we will refer to as method *HAS-SOP*$^{P}_{APC}$, where P stands for *preprocessing*.

  The artificial precedence constraints are added all at once at the beginning of the execution of the conventional *HAS-SOP* algorithms, which then runs on a steady problem, that has more precedence constraints than the original one.

  In detail, 20 solutions (i.e. 2 for each ant) are generated (and taken down to their local optima). The 10 most promising artificial constraints (according to the values in matrix m) are added to the precedence digraph P (together with those implied by transitivity). The *HAS-SOP* algorithm runs on the modified problem for the available computational time remaining.

- ***HAS-SOP*$^{C}_{APC}$:** Parameters are set up as follows: $u = 100, v = 1000, w = 1$. This parameter setting leads to the configuration that we will identify as *HAS-SOP*$^{C}_{APC}$ in the remainder of the paper, where C stands for *cumulative*. In this case artificial precedence constraints are added in a cumulative fashion during the whole execution of the conventional *HAS-SOP* algorithm. In particular, every time 1000 new solutions are generated, matrix m is examined and the precedence constraints associated with the entry of the matrix with the highest positive value (if any) is added to the precedence digraph P (together with constraints implied by transitivity). The precedence digraph P evolves therefore during the whole computation, getting more and more restrictive.

The two methods listed above will be considered for the computational experiments described in Section 4.

4. COMPUTATIONAL RESULTS

The aim of this section is to compare the original *HAS-SOP* algorithm with the modified methods *HAS-SOP*$^{P}_{APC}$ and *HAS-SOP*$^{C}_{APC}$, described in Section 3.

All the methods have been coded in C++ (starting from the original implementation of *HAS-SOP*, see [9]) and all the experiments have been run on a Intel Pentium 4 1.5GHz / 256MB machine. The maximum computation time was set to 600 seconds for all the problems. This computation time should be long enough to let all the methods reach a steady state, where further improvements are unlikely to be found.

4.1. Benchmark problems

The benchmark problems available at TSPLIB[1] have been initially used for testing the new algorithms we propose.

[1] http://www.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95/.

Unfortunately it was impossible to observe any significant difference in performance between *HAS-SOP* and *HAS-SOP*$_{APC}$ methods, since the problems tend to be rather easy for modern heuristics (for most of the problems the best solutions have been proven to be optimal, and for the remaining ones no improvement has been registered in the last ten years, and very good lower bounds are available). For this reason, we decided to generate new random problems, bigger and harder to solve than those contained in the (dated) TSPLIB.

The problems we generated, that are publicly available[2], were named *n-r-p*, where the meaning of each element is as follows:

n: the number of nodes of the problem, i.e. $V = \{1, 2, \dots, n\}$;

r: the cost range, i.e. $0 \leq c_{ij} \leq r \ \ \forall i, j \in V$;

p: the approximate percentage of precedence constraints, i.e. the number of precedence constraints of the problem will be about $\frac{p}{100} \cdot \frac{n(n-1)}{2}$.

We considered the following values for the parameters above, generating problems for all the possible combinations of them:

- $n \in \{200, 300, 400, 500, 600, 700\}$;
- $r \in \{100, 1000\}$;
- $p \in \{1, 15, 30, 60\}$.

The resulting set of problems covers a wide range of situations, with problems of different size, with different granularity for costs, and with radically different percentages of precedence constraints. The set should provide a good testbed for modern *SOP* heuristic algorithms.

4.2. Experiments

Five runs are considered for each possible problem/method combination. The results of the experiments are reported in Table 1. The first three columns are parameters of the problems, while the remaining columns are devoted to the presentation of the average and best results obtained by the methods considered. Percentage improvements over the standard *HAS-SOP* method are reported for *HAS-SOP*$^P_{APC}$ and *HAS-SOP*$^C_{APC}$ (both for average and best cases). Extra lines have finally been inserted into the table to present averages for percentage improvements on different subsets of the testbed.

Table 1 confirms the intuition that adding artificial precedence constraints, together with the associated search-space reduction, can help ant colony system methods for the sequential ordering problem.

A deeper analysis of the results reported in Table 1 leads to the observation that method *HAS-SOP*$^P_{APC}$ works better than *HAS-SOP*$^C_{APC}$. This result, which might be somehow surprising, can be explained by observing that method *HAS-SOP*$^C_{APC}$, contrary to what happens for the simpler method *HAS-SOP*$^P_{APC}$, seems to have two main drawbacks (revealed by other experiments not reported here). In some situations (typically when just a few constraints are specified in the original problem) solutions with very different characteristics are likely to be retrieved by the *HAS-SOP* method, leading to many negative entries in matrix m. In this case the algorithm is able to add just a few artificial constraints. On the other hand, for problems with many (original) precedence constraints, the strategy of adding constraints during the whole computation tend to quickly overrestrict the search space around a small set of solutions, preventing the algorithm from exploring other (possibly promising) areas of the search-space.

Another interesting phenomenon emerging from Table 1 is that method *HAS-SOP*$^P_{APC}$ very rarely obtains worse results than the classic *HAS-SOP* algorithm. This observation suggests that it is convenient to use *HAS-SOP*$^P_{APC}$ because even if it does not provide improvements, it does not “hide” good solutions that the classical method would be able to retrieve.

A further analysis of Table 1 suggests that the improved methods work better for the largest problems, for which the standard method is likely to have more difficulties. This proves once again that artificial precedence constraints help in making the problem easier to handle. Another confirmation of this intuition comes from the observation that the improvements guaranteed by the new methods tend to vanish when p (percentage of precedence constraints in the original problem) increases, and consequently the search-space is already small and there is not an obvious convenience in reducing it further.

5. CONCLUSIONS AND FUTURE WORK

A problem manipulation method, which creates and adds artificial precedence constraints to the original problem, has been embedded into a well-known ant colony system for the sequential ordering problem.

The extended method induces small improvements in the performance of the classical ant colony system, leading to better results, especially on large and difficult problems.

It is important to observe that the problem manipulation method we propose can be adapted to many other combinatorial optimization methods, and that it is not applicable to ant colony systems only. In our future research we will then try to generalize the manipulation method. We will apply it

[2] http://www.idsia.ch/~roberto/SOPLIB06.zip.

to different problems, with different algorithms driving the optimization.

Another stream of research will be dedicated to the development of improved versions of *HAS-SOP*$^{C}_{APC}$, in which artificial constraints can be not only added, but also retracted during the computation.

6. ACKNOWLEDGEMENTS

The authors would like to thank Diego Frei and Filip Klisic for their contributions in the implementation of the new extensions into the original *HAS-SOP* code.

7. REFERENCES

[1] N. Ascheuer. *Hamiltonian path problems in the online optimization of flexible manufacturing systems*. PhD thesis, Technische Universität Berlin, 1995.

[2] N. Ascheuer, L.F. Escudero, M. Grötschel, and M. Stoer. A cutting plane approach to the sequential ordering problem (with applications to job scheduling in manufacturing). *SIAM Journal on Optimization*, 3:25–42, 1993.

[3] E. Balas, M. Fischetti, and W.R. Pulleyblank. The precedence-constrained asymmetric traveling salesman polytope. *Mathematical Programming*, 65:241–265, 1995.

[4] S. Chen and S. Smith. Commonality and genetic algorithms. Technical Report CMU-RI-TR-96-27, The Robotic Institute, Carnagie Mellon University, 1996.

[5] M. Dorigo, G. Di Caro, and L.M. Gambardella. Ant algorithms for discrete optimization. *Artificial Life*, 5:137–172, 1999.

[6] M. Dorigo and L.M. Gambardella. Ant Colony System: a cooperative learning approach to the traveling salesman problem. *IEEE Transactions on Evolutionary Computation*, 1:53–66, 1997.

[7] L.F. Escudero. An inexact algorithm for the sequential ordering problem. *European Journal of Operational Research*, 37:232–253, 1988.

[8] L.F. Escudero, M. Guignard, and K. Malik. A Lagrangean relax-and-cut approach for the sequential ordering problem with precedence relationships. *Annals of Operations Research*, 50:219–237, 1994.

[9] L.M. Gambardella and M. Dorigo. An ant colony system hybridized with a new local search for the sequential ordering problem. *INFORMS Journal on Computing*, 12(3):237–255, 2000.

[10] F. Guerriero and M. Mancini. A cooperative parallel rollout algorithm for the sequential ordering problem. *Parallel Computing*, 29(5):663–677, 2003.

[11] I.T. Hernàdvölgyi. Solving the sequential ordering problem with automatically generated lower bounds. In *Proceedings of Operations Research 2003*, pages 355–362, 2003.

[12] I.T. Hernàdvölgyi. *Automatically Generated Lower Bounds for Search*. PhD thesis, University of Ottawa, 2004.

[13] W. Pullyblank and M. Timlin. Precedence constrained routing and helicopter scheduling: heuristic design. Technical Report RC17154 (#76032), IBM T.J. Watson Research Center, 1991.

[14] M.W.P. Savelsbergh. An efficient implementation of local search algorithms for constrained routing problems. *European Journal of Operational Research*, 47:75–85, 1990.

[15] D.-I. Seo and B.R. Moon. A hybrid genetic algorithm based on complete graph representation for the sequential ordering problem. In *Proceedings of GECCO 2003*, pages 69–680, 2003.

```
1.  For each pair $(r,s)$
        $\tau_{rs} := \tau_0$
#       $m_{rs} := 0$
    EndFor
#   counter $:= 1$
2.  For $k$:=1 to $m$ do
        Let $r_k$ be the node where ant $k$ is located
        $r_k$ := 1 /* All ants start from node 1 */
    EndFor
    /* The path of agent $k$ is stored in $Path_k$ */
    For $k$:=1 to $m$ do
        For $i$:=1 to $n-1$ do
            Starting from $r_k$ compute the set $F(r_k)$ of feasible nodes
            /* $F(r_k)$ contains all the nodes $j$ still to be visited and such that all
            nodes that have to precede $j$ have already been inserted in the sequence */
            Choose the next node $s_k$ according to the transition rule (see Section 2.1.1)
            $Path_k := (r_k, s_k)$
            $\tau_{r_k s_k} := (1-\psi) \cdot \tau_{rs} + \psi \cdot \tau_0$ /* This is equation (2) */
            $r_k := s_k$
        EndFor
        $OptPath_k := LocalSearchRoutine(Path_k)$
        Compute $L_k$ /* $L_k$ is the length of the path $OptPath_k$ */
#       For each nodes $r, s \in V$ such that $\pi_k(i) < \pi_k(j) \leq \pi_k(i) + 5$
#       /* $\pi_k(i)$ is the index of the position of node $i$ in solution $OptPath_k$ */
#           $m_{rs} := m_{rs} + L_1/L_k$ /* This is equation (3) */
#           $m_{sr} := m_{sr} - L_1/L_k$ /* This is equation (4) */
#       EndFor
#       counter := counter + 1
#       If (counter mod $v$) == $u$)
#           For $i$:=1 to $w$
#               $(r,s) = argmax_{(j,k) \in A, (j,k) \notin P} \{m_{jk}\}$
#               If $(m_{rs} \geq 0)$
#                   $R$ := $R \cup (r,s)$
#               Else
#                   $i$ := $w$ /* Forcing the exit from the For loop */
#               EndIf
#           EndFor
#       EndIf
#   EndFor
    Let $L_{best}$ be the shortest $L_k$ from beginning and $OptPath_{best}$ the corresponding path
    For each arc $(r,s) \in OptPath_{best}$
        $\tau_{rs} := (1-\rho) \cdot \tau_{rs} + \rho / L_{best}$ /* This is equation (1) */
    EndFor
    If (Time > MaxTime)
        then
            Print $L_{best}$ and $OptPath_{best}$
        else
            repeat Step 2
    EndIf
```

Figure 1. The *HAS-SOP*$_{APC}$ algorithm. The omission of the steps marked with # leads to the conventional *HAS-SOP* algorithm.

Table 1. Experimental results. Averages and best results over five runs. Maximum computation time 600 seconds.

Problem			Average results					Best results				
			Results			Improvement over *HAS-SOP* (%)		Results			Improvement over *HAS-SOP* (%)	
n	*r*	*p*	*HAS-SOP*	*HAS-SOP*$^{P}_{APC}$	*HAS-SOP*$^{C}_{APC}$	*HAS-SOP*$^{P}_{APC}$	*HAS-SOP*$^{C}_{APC}$	*HAS-SOP*	*HAS-SOP*$^{P}_{APC}$	*HAS-SOP*$^{C}_{APC}$	*HAS-SOP*$^{P}_{APC}$	*HAS-SOP*$^{C}_{APC}$
200	100	1	91,8	88,8	90,0	3,27	1,96	88	87	87	1,14	1,14
300	100	1	77,6	72,6	72,6	6,44	6,44	74	65	65	12,16	12,16
400	100	1	63,8	63,8	66,0	0,00	-3,45	59	59	64	0,00	-8,47
500	100	1	55,2	55,6	56,4	-0,72	-2,17	51	51	54	0,00	-5,88
600	100	1	50,6	48,4	48,4	4,35	4,35	44	42	42	4,55	4,55
700	100	1	44,4	40,4	42,6	9,01	4,05	41	32	38	21,95	7,32
Averages for r = 100 and p = 1						**3,72**	**1,86**				**6,63**	**1,80**
200	1000	1	1568,6	1547,2	1580,4	1,36	-0,75	1542	1529	1546	0,84	-0,26
300	1000	1	1595,6	1591,2	1604,2	0,28	-0,54	1568	1533	1574	2,23	-0,38
400	1000	1	1812,6	1778,6	1814,0	1,88	-0,08	1783	1757	1798	1,46	-0,84
500	1000	1	1880,2	1864,6	1886,8	0,83	-0,35	1842	1786	1838	3,04	0,22
600	1000	1	1987,4	1962,4	1987,0	1,26	0,02	1948	1933	1934	0,77	0,72
700	1000	1	1956,4	1947,6	1963,2	0,45	-0,35	1912	1873	1873	2,04	2,04
Averages for r = 1000 and p = 1						**1,01**	**-0,34**				**1,73**	**0,25**
Averages for p = 1						**2,37**	**0,76**				**4,18**	**1,02**
200	100	15	2140,2	2107,8	2121,6	1,51	0,87	2002	1984	2004	0,90	-0,10
300	100	15	3835,6	3669,2	3830,2	4,34	0,14	3673	3455	3724	5,94	-1,39
400	100	15	5104,8	5070,0	5090,0	0,68	0,29	4852	4735	4735	2,41	2,41
500	100	15	7021,0	6857,4	7005,4	2,33	0,22	6610	6610	6749	0,00	-2,10
600	100	15	7885,0	7758,8	7854,8	1,60	0,38	7701	7702	7702	-0,01	-0,01
700	100	15	9633,0	9531,6	9670,0	1,05	-0,38	9324	9124	9124	2,15	2,15
Averages for r = 100 and p = 15						**1,92**	**0,25**				**1,90**	**0,16**
200	1000	15	22562,4	22275,2	22466,8	1,27	0,42	22220	21857	22224	1,63	-0,02
300	1000	15	35633,0	34276,4	36163,2	3,81	-1,49	34455	33148	34428	3,79	0,08
400	1000	15	48115,2	46541,0	47649,2	3,27	0,97	47076	45115	45615	4,17	3,10
500	1000	15	63895,8	63787,4	64022,4	0,17	-0,20	63104	62001	61138	1,75	3,12
600	1000	15	76766,2	71308,6	74246,0	7,11	3,28	76011	68219	71612	10,25	5,79
700	1000	15	88765,4	85522,0	88375,4	3,65	0,44	86689	83995	84195	3,11	2,88
Averages for r = 1000 and p = 15						**3,21**	**0,57**				**4,12**	**2,49**
Averages for p = 15						**2,57**	**0,41**				**3,01**	**1,32**
200	100	30	4262,8	4246,8	4251,4	0,38	0,27	4256	4216	4216	0,94	0,94
300	100	30	6275,4	6222,6	6234,4	0,84	0,65	6191	6186	6176	0,08	0,24
400	100	30	8506,4	8506,4	8448,4	0,00	0,68	8289	8289	8385	0,00	-1,16
500	100	30	10511,8	10261,0	10332,4	2,39	1,71	10098	10067	10098	0,31	0,00
600	100	30	13276,0	13125,4	13126,0	1,13	1,13	13152	12833	13058	2,43	0,71
700	100	30	15898,4	15850,2	15789,4	0,30	0,69	15756	15672	15647	0,53	0,69
Averages for r = 100 and p = 30						**0,84**	**0,85**				**0,71**	**0,24**
200	1000	30	41453,8	41387,0	41453,4	0,16	0,00	41372	41283	41367	0,22	0,01
300	1000	30	55747,8	54762,0	54752,4	1,77	1,79	55289	54452	54523	1,51	1,39
400	1000	30	86946,0	86476,2	86946,0	0,54	0,00	86311	85934	86311	0,44	0,00
500	1000	30	103318,4	103049,6	104190,8	0,26	-0,84	101365	101365	101485	0,00	-0,12
600	1000	30	135192,6	133749,0	133148,2	1,07	1,51	133984	132216	132141	1,32	1,38
700	1000	30	144310,4	143611,6	143904,2	0,48	0,28	141263	141263	140582	0,00	0,48
Averages for r = 1000 and p = 30						**0,71**	**0,46**				**0,58**	**0,52**
Averages for p = 30						**0,78**	**0,66**				**0,65**	**0,38**
200	100	60	71749,0	71749,0	71749,0	0,00	0,00	71749	71749	71749	0,00	0,00
300	100	60	9726,0	9726,0	9726,0	0,00	0,00	9726	9726	9726	0,00	0,00
400	100	60	15240,4	15235,6	15234,4	0,03	0,04	15228	15228	15228	0,00	0,00
500	100	60	18309,4	18291,6	18291,6	0,10	0,10	18279	18240	18240	0,21	0,21
600	100	60	23400,6	23387,8	23387,8	0,05	0,05	23371	23333	23362	0,16	0,04
700	100	60	24294,6	24264,2	24268,8	0,13	0,11	24247	24233	24189	0,06	0,24
Averages for r = 100 and p = 60						**0,05**	**0,05**				**0,07**	**0,08**
200	1000	60	71585,0	71585,0	71614,0	0,00	-0,04	71556	71556	71556	0,00	0,00
300	1000	60	109542,4	109494,8	109494,8	0,04	0,04	109471	109471	109471	0,00	0,00
400	1000	60	141121,0	141121,0	141161,0	0,00	-0,03	140920	140920	140963	0,00	-0,03
500	1000	60	178781,4	178781,4	178850,8	0,00	-0,04	178408	178408	178464	0,00	-0,03
600	1000	60	215642,2	215467,8	215467,8	0,08	0,08	214956	214850	214850	0,05	0,05
700	1000	60	247321,4	246785,2	247137,4	0,22	0,07	246759	246142	246142	0,25	0,25
Averages for r = 1000 and p = 60						**0,06**	**0,02**				**0,05**	**0,04**
Averages for p = 60						**0,05**	**0,03**				**0,06**	**0,06**
Overall averages						**1,44**	**0,47**				**1,97**	**0,70**