A heuristic manipulation technique for the sequential ordering problem

R. Montemanni, D.H. Smith, L.M. Gambardella

PII: S0305-0548(07)00109-8
DOI: doi:10.1016/j.cor.2007.05.003
Reference: CAOR 2045

To appear in: Computers & Operations Research

Received date: 20 February 2007
Revised date: 3 May 2007
Accepted date: 17 May 2007


This is a PDF file of an unedited manuscript that has been accepted for publication. As a service to our customers we are providing this early version of the manuscript. The manuscript will undergo copyediting, typesetting, and review of the resulting galley proof before it is published in its final citable form. Please note that during the production process errors may be discovered which could affect the content, and all legal disclaimers that apply to the journal pertain.
A Heuristic Manipulation Technique for the Sequential Ordering Problem

R. Montemanni\textsuperscript{a,}\textsuperscript{*}, D.H. Smith\textsuperscript{b}, L.M. Gambardella\textsuperscript{a}

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

\textsuperscript{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 travelling 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 many real world applications, mainly in the fields of transportation and production planning.

A problem manipulation technique to be used in conjunction with heuristic algorithms is discussed. The aim of the technique is to make the search space associated with each problem more attractive for the underlying heuristic algorithms.

This novel methodology is tested in combination with the state-of-the-art method for the sequential ordering problem. Improved results are

\textsuperscript{*}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.
obtained, particularly for the largest problems considered.

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

1 Introduction

The *Sequential Ordering Problem* (SOP), also referred to as the *Asymmetric Travelling Salesman Problem with Precedence Constraints*, can be modelled in graph theoretical terms as follows. A complete directed graph $D = (V,A)$ is given, where $V$ is the set of nodes and $A = \{(i,j) \mid i,j \in V\}$ is the set of arcs. A cost $c_{ij} \in \mathbb{N}$ is associated with each arc $(i,j) \in A$. Without loss of generality it can be assumed 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 ($c_{i1} = 0 \ \forall \ i \in V$ by definition). This artifact creates an analogy with the asymmetric travelling salesman problem. Such an analogy is exploited by many known algorithms. Furthermore an additional precedence digraph $P = (V,R)$ is given, 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. Such a relation will be denoted 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. It is also assumed to be transitively closed, since $i \prec k$ can be inferred from $i \prec j$ and $j \prec k$. Note 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 travelling 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 (Escud-
ero [7] and Seo and Moon [16]), single vehicle routing problems with pick-up and delivery constraints (Pulleyblank and Timlin [14], Savelsbergh [15]) 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 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 of computing and improving 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 Hernández [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 schemata 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 [16]. 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.

The contribution of the present article is a problem manipulation technique to be used with existing heuristic algorithms, in order to enhance their performance. The rationale behind the approach is that for a given algorithm, the shape of the search space of the problem can be modified by adding artificial precedence constraints, in order to make it more attractive for the underlying heuristic algorithm. In particular, the problem manipulation technique proposed for the SOP will be experimentally shown to give better results than for the underlying algorithm without use of the technique. The technique presented combines both addition and retraction of precedence constraints. Embryonic versions of the method discussed in this paper, without constraint retraction, have been presented in Montemanni et al. [13].

The paper is organized as follows: Section 2, which is the core of the paper, will describe in detail the problem manipulation technique proposed. Section 3 will briefly introduce the Ant Colony Optimization paradigm and its application to the SOP. This section has been introduced since the proposed manipulation technique will be tested in conjunction with the ACO algorithm for the SOP originally described in Gambardella and Dorigo [9]. Computational experiments will be presented in Section 4, while conclusions will be drawn in Section 5.

2 A problem manipulation approach

It is easy 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, the method described in the remainder of this section has been developed.

Having selected an underlying heuristic method, the idea is to monitor the solutions generated by this method, and to identify precedence patterns common to solutions with a low objective value. Once such precedence patterns are identified, they can be added to the original problem as artificial precedence constraints. The manipulated problem is likely to be easier than the original
one, as it has a reduced solution space.

Notice that any heuristic method that produces a sequence of feasible solutions to the problem (most of the known methods work in this way) can be used as the underlying method for the proposed manipulation approach.

Of course such a heuristic method may cut out all the optimal solutions of the original problem, leading to suboptimal solutions even when the best solution of the modified problem is retrieved. To overcome this side effect, during the execution of the algorithm artificial precedence constraints will not be added permanently, but will also be retracted (and substituted by other constraints).

Formally, the proposed methodology is built on top of an existing algorithm and makes use of an additional set of variables \(\{m_{ij}\}\). Variable \(m_{ij}\) will be an indicator for the “quality” of the solutions in which node \(i\) is visited before node \(j\). The following parameters also need to be defined:

\[ u : \text{the number of solutions generated by the underlying heuristic method before the first artificial precedence constraints are added to the problem;} \]

\[ v : \text{the number of solutions generated by the underlying heuristic method between two consecutive updates to the set of active artificial precedence constraints;} \]

\[ w : \text{the (approximate) number of artificial precedence constraints active at each moment in time (after the first } u \text{ solutions have been generated by the underlying heuristic approach);} \]

\[ z : \text{the (approximate) number of artificial precedence constraints substituted after every } v \text{ new solutions have been generated by the underlying heuristic algorithm (after the first } u \text{ solutions have been generated);} \]

After having selected an underlying heuristic algorithm, the manipulation approach, which runs on top of such an algorithm, can be summarized as follows.

Initialize \(m_{ij} = 0 \ \forall (i,j) \in A\).
Each time a new solution \( \text{OptPath}_k \), with cost \( L_k \), is generated by the underlying heuristic algorithm, matrix \( m = [m_{ij}] \) is updated as follows:

\[
m_{ij} = m_{ij} + \frac{L_1}{L_k} \quad \forall i, j \in V, \pi_k(i) < \pi_k(j) \leq \pi_k(i) + t, (i, j) \notin R \tag{1}
\]

\[
m_{ji} = m_{ji} - \frac{L_1}{L_k} \quad \forall i, j \in V, \pi_k(i) < \pi_k(j) \leq \pi_k(i) + t, (i, j) \notin R \tag{2}
\]

where \( L_1 \) is the cost of the very first solution generated by the underlying heuristic algorithm and \( \pi_k(i) \) is the index of the position occupied by node \( i \) in solution \( \text{OptPath}_k \). Notice that \( L_1 \) plays here the role of a normalization factor, and is used to avoid numerical problems. The value of \( t \) regulates the width of the window considered for updates.

The first update (equation (1)) reinforces the entry corresponding to a sequence which is in solution \( \text{OptPath}_k \). The update is proportional to the inverse of the cost of the solution itself. Equation (2) 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.

Notice that entries of the memory matrix \( m \) corresponding to active artificial precedence constraints are not updated. This will make a rotation of the active constraints more likely (see the remainder of this section).

Now consider possible values of \( t \). Values 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. On the other hand, values of \( t \) 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, it was decided not to list \( t \) as a key parameter of the algorithm. Preliminary results suggest that the method is not sensitive at all to changes to (reasonable values of) \( t \). In particular, values within the interval \([4, 10]\) seem to guarantee the best performance. Here \( t = 5 \) is used.
Now that it has been clarified how the memory matrix $m$ is handled, it remains to clarify how artificial precedence constraints are managed. After the first $u$ solutions are created by the underlying heuristic algorithm, a first set of (approximately) $w$ artificial precedence constraints are added to the set $R$. The new constraints are selected as the ones not yet present in the precedence digraph $P$ with the highest entries in matrix $m$. 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 $P$.

After the first artificial precedence constraints have been added to the problem, every time $v$ new solutions are available, artificial precedence constraints are updated by dropping $z$ constraints, that are substituted by (approximately) $z$ new constraints. The artificial precedence constraints to be dropped are selected as those with the smallest entries in the memory matrix $m$. Conversely, the ones added are those with the highest entries in the same matrix $m$. Notice that, since entries of matrix $m$ corresponding to active constraints are not reinforced (see equation (1)), it is likely that during the time they were active, entries corresponding to other (non active) constraints have reached higher values. Such a strategy leads to a mechanism where artificial precedence constraints are activated in turn. The mechanism should also prevent optimal solutions of the original problem from being permanently hidden by the active artificial precedence constraints.

The pseudo-code in Figure 1 summarizes the manipulation technique. In the pseudo-code, the set $R_A$ contains the active artificial precedence constraints. It is assumed that $u < v$ and $z \leq w$.

3 Ant Colony Optimization for the SOP

In this section the basic concepts of the HAS-SOP algorithm, originally presented in Gambardella and Dorigo [9], are discussed. In Section 4 the proposed problem manipulation technique will be tested on top of the HAS-SOP algorithm, which is nowadays considered the state-of-the-art heuristic method for
the SOP. 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.

Application of an ACO algorithm to a combinatorial optimization problem requires definition of a constructive algorithm (called ACS-SOP here) and possibly a local search. The resulting algorithm is a Hybrid Ant System for the SOP called HAS-SOP, which is described in detail in Gambardella and Dorigo [9].

3.1 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. Constructive computational agents called ants (simulating real ants) are sent out sequentially. 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 \( \eta_{ij} \) here defined as \( 1/c_{ij} \), and the pheromone trail \( \tau_{ij} \), that contains a measure of how good it has been in the past to include arc \((i, j)\) into a solution. 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

\[
p_{ij\in F(i)} = \frac{\tau_{ij}\eta_{ij}}{\sum_{l\in F(i)}(\tau_{il}\eta_{il})}.
\]

The value \( q_0 \) is given by \( q_0 = 1 - s/|V| \). The parameter \( s \) represents the
number of nodes it would be desirable to choose using the probabilistic transition rule, independently of the number of nodes of the problem, and is set to 10 in the experiments reported here.

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. If the shortest path generated since the beginning of the computation is referred to as $OptPath_{Best}$, and its cost as $L_{Best}$, $\forall \{i,j\} \in OptPath_{Best}$, the following formula for pheromone update is used:

$$\tau_{ij} = (1 - \rho) \cdot \tau_{ij} + \frac{\rho}{L_{best}}$$  \hspace{1cm} (3)$$

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. This assures variety in the solutions generated. The rule is:

$$\tau_{ij} = (1 - \psi) \cdot \tau_{ij} + \psi \cdot \tau_0$$  \hspace{1cm} (4)$$

where $\tau_0$ is the initial value of trails. It was found that good values for the algorithm’s parameters are $\tau_0 = (FirstSolution \cdot |V|)^{-1}$ and $\rho = \psi = 0.1$, where $FirstSolution$ is the length of the shortest solution generated by the ant colony following the ACS-SOP algorithm without using the pheromone trails. The number of ants in the population was set to 10. Experience has shown the chosen parameter settings to be robust.

### 3.2 Complete algorithm (HAS-SOP)

The HAS-SOP algorithm is the ACS-SOP algorithm augmented by local search.

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 travelling salesman problem (Savelsbergh [15]). 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 beyond the scope of this paper (although the local search routine is used by the HAS-SOP algorithm) the interested reader is referred 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 (3).

4 Computational results

The aim of this section is to provide an experimental evaluation of the improvements given by the problem manipulation technique described in Section 2, when it runs on top of the HAS-SOP algorithm, described in Section 3.

All the methods considered have been coded in C++ (starting from the original implementation of HAS-SOP, see [9]).

4.1 Benchmark problems

The benchmark problems available at TSPLIB\(^1\) have been used initially for testing the effectiveness of the proposed manipulation technique. Unfortunately it was impossible to observe any significant difference in performance between the original HAS-SOP method and the one enhanced with the manipulation technique, since the problems tend to be rather easy for modern heuristics (for most of the problems the best solutions have been proved 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, it was decided to generate new bigger random problems, which are harder to solve than those contained in the (dated) TSPLIB.

The problems generated, are publicly available\(^2\), and are named \(n-r-p\), where the meaning of each element is as follows:

\(^{1}\)http://www.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95/.

\(^{2}\)http://www.idsia.ch/~roberto/SOPLIB06.zip.
The following values for the parameters above were used, and problems were generated for all 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 different sizes, different granularity for costs, and with radically different percentages of precedence constraints. The set appears to provide a good testbed for modern SOP heuristic algorithms.

### 4.2 Parameter tuning

The following quantities have been selected during some preliminary tests, and are used as reference default values: $u = 100$, $v = 50$, $w = 20$ and $z = 5$.

The values of the parameters will be changed one at a time, keeping the others at the default values, and measuring how the method performs. A selection of five problems (namely $n, r, p = 300, 100, 15; 600, 100, 15; 300, 100, 1; 300, 100, 60; 300, 100, 15$) is used for parameter tuning.

In Figure 2 the study concerning parameter $u$ (measuring the number of solutions generated before the first artificial precedence constraint is created) is reported. The performances when $u = 10$ and $u = 1000$ are compared with those achieved with the default setting $u = 100$. Average and best results over five runs are considered. The experiments have been run on an Intel Pentium 4 1.5GHz / 256MB machine with a maximum computation time set to 600
seconds. The results reported suggest that the manipulation technique is not very sensitive to changes in the value of parameter $u$: values ranging between 10 and 1000 do not make the approach perform significantly differently. The only significantly worse performance is obtained for the problem $300 - 100 - 1$, with $u = 10$. In the remainder of the tests $u = 100$ will be used.

In Figure 3 the study on parameter $v$ (measuring the number of solutions generated between two consecutive updates to the set of artificial precedence constraints) is summarized. The performances when $v = 5$ and $v = 500$ are compared with those achieved with the default setting $v = 50$. Average and best results over five runs are considered. Also in this case the changes in parameter $v$ do not lead to very different results. It is possible to notice a partial performance degradation with $v = 500$. This is reasonable since intuitively there is not enough rotation of artificial precedence constraints, and the algorithm will tend to focus on a sub-optimal search sub-space. In the remainder of the tests $v = 50$ will be used.

In Figure 4 the study on parameter $w$ (measuring the number of active artificial precedence constraints at each moment in time) is summarized. The performances when $w = 10$, $w = 30$ and $w = 80$ are compared with those achieved with the default setting $w = 20$. Average and best results over five runs are considered. The results indicate that also in the case of $w$, the method is not particularly sensitive to parameter tuning. The interesting observation is that small values produce better results (especially in terms of best results), while values in the order of 80, which are still small in terms of the total number of possible precedence constraints, already produce a performance degradation. It appears that this property is strongly connected with the characteristics of the problem under investigation. In the remainder of the tests $w = 20$ will be used.

In Figure 5 the study on parameter $z$ (measuring the number of artificial precedence substituted every $z$ iterations of the algorithm) is reported. The performance when $z = 1$ and $z = 10$ are compared with those achieved with the default setting $z = 5$. Average and best results over five runs are consid-
Figure 5 suggests that again parameter $z$ is not very sensitive to tuning, but small values have to be chosen. This is in accordance with the previous observation that values of $w$ that guarantee good results have to be small.

4.3 Comparison with HAS-SOP

The experiments have been run on a Dual AMD Opteron 2.4GHz / 4GB computer. 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. Ten runs are considered for each possible problem/method combination. The results of the experiments are reported in Tables 1-3. The first three columns are parameters of the problems, while the remaining columns are devoted, depending on the table, to the presentation of the average, best and worst results obtained by the original HAS-SOP method and by enhancing HAS-SOP with the manipulation technique. The enhanced method is referred to as APC + HAS-SOP, where APC stands for Artificial Precedence Constraints. Percentage improvements over the standard HAS-SOP method are finally reported in the tables.

The results of Table 1 confirm that adding artificial precedence constraints is convenient since, first of all, it virtually never leads to worse average results, and even when the average is worse, it is by a negligible quantity. This fact is particularly important because it indicates that the use of artificial precedence constraints can be recommended: in the worst case the quality of the solutions is (almost) the same, but often the additional implementation effort is justified by improved results.

A deeper analysis of Table 1 leads to the observation that the average improvements guaranteed by the problem manipulation technique decrease as the value of $p$ (percentage of precedence constraints present in the original problem) increases. Figure 6 depicts this phenomenon; it can be explained by observing that large values of $p$ imply already reduced search spaces. This in turn implies
Table 1: Average results over ten runs.

<table>
<thead>
<tr>
<th>$n$</th>
<th>$r$</th>
<th>$p$</th>
<th>HAS-SOP</th>
<th>APC + HAS-SOP</th>
<th>Improvement (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>200</td>
<td>100</td>
<td>1</td>
<td>90.3</td>
<td>88.4</td>
<td>2.10</td>
</tr>
<tr>
<td>300</td>
<td>100</td>
<td>1</td>
<td>76.4</td>
<td>71.9</td>
<td>5.89</td>
</tr>
<tr>
<td>400</td>
<td>100</td>
<td>1</td>
<td>64.1</td>
<td>61.7</td>
<td>3.74</td>
</tr>
<tr>
<td>500</td>
<td>100</td>
<td>1</td>
<td>55.0</td>
<td>55.1</td>
<td>-0.18</td>
</tr>
<tr>
<td>600</td>
<td>100</td>
<td>1</td>
<td>49.8</td>
<td>47.6</td>
<td>4.42</td>
</tr>
<tr>
<td>700</td>
<td>100</td>
<td>1</td>
<td>42.6</td>
<td>39.5</td>
<td>7.28</td>
</tr>
<tr>
<td>200</td>
<td>1000</td>
<td>1</td>
<td>1549.5</td>
<td>1551.5</td>
<td>-0.13</td>
</tr>
<tr>
<td>300</td>
<td>1000</td>
<td>1</td>
<td>1586.7</td>
<td>1584.9</td>
<td>0.11</td>
</tr>
<tr>
<td>400</td>
<td>1000</td>
<td>1</td>
<td>1811.1</td>
<td>1773.1</td>
<td>2.10</td>
</tr>
<tr>
<td>500</td>
<td>1000</td>
<td>1</td>
<td>1877.4</td>
<td>1838.4</td>
<td>2.08</td>
</tr>
<tr>
<td>600</td>
<td>1000</td>
<td>1</td>
<td>1986.7</td>
<td>1957.6</td>
<td>1.46</td>
</tr>
<tr>
<td>700</td>
<td>1000</td>
<td>1</td>
<td>1969.2</td>
<td>1943.1</td>
<td>1.33</td>
</tr>
<tr>
<td>200</td>
<td>100</td>
<td>15</td>
<td>2066.0</td>
<td>2025.1</td>
<td>1.98</td>
</tr>
<tr>
<td>300</td>
<td>100</td>
<td>15</td>
<td>3738.6</td>
<td>3648.2</td>
<td>2.42</td>
</tr>
<tr>
<td>400</td>
<td>100</td>
<td>15</td>
<td>5087.1</td>
<td>4917.7</td>
<td>3.33</td>
</tr>
<tr>
<td>500</td>
<td>100</td>
<td>15</td>
<td>6931.5</td>
<td>6826.4</td>
<td>1.52</td>
</tr>
<tr>
<td>600</td>
<td>100</td>
<td>15</td>
<td>7806.6</td>
<td>7733.0</td>
<td>0.94</td>
</tr>
<tr>
<td>700</td>
<td>100</td>
<td>15</td>
<td>9573.0</td>
<td>9453.7</td>
<td>1.25</td>
</tr>
<tr>
<td>200</td>
<td>1000</td>
<td>15</td>
<td>22602.9</td>
<td>22419.2</td>
<td>0.81</td>
</tr>
<tr>
<td>300</td>
<td>1000</td>
<td>15</td>
<td>34447.9</td>
<td>34215.0</td>
<td>0.68</td>
</tr>
<tr>
<td>400</td>
<td>1000</td>
<td>15</td>
<td>46638.6</td>
<td>46183.2</td>
<td>0.98</td>
</tr>
<tr>
<td>500</td>
<td>1000</td>
<td>15</td>
<td>62693.7</td>
<td>61675.7</td>
<td>1.62</td>
</tr>
<tr>
<td>600</td>
<td>1000</td>
<td>15</td>
<td>72701.1</td>
<td>70369.9</td>
<td>3.53</td>
</tr>
<tr>
<td>700</td>
<td>1000</td>
<td>15</td>
<td>85177.7</td>
<td>82697.0</td>
<td>2.91</td>
</tr>
<tr>
<td>200</td>
<td>100</td>
<td>30</td>
<td>4254.6</td>
<td>4236.0</td>
<td>0.44</td>
</tr>
<tr>
<td>300</td>
<td>100</td>
<td>30</td>
<td>6228.2</td>
<td>6180.2</td>
<td>0.77</td>
</tr>
<tr>
<td>400</td>
<td>100</td>
<td>30</td>
<td>8476.5</td>
<td>8395.3</td>
<td>0.96</td>
</tr>
<tr>
<td>500</td>
<td>100</td>
<td>30</td>
<td>10333.2</td>
<td>10108.6</td>
<td>2.17</td>
</tr>
<tr>
<td>600</td>
<td>100</td>
<td>30</td>
<td>13001.8</td>
<td>12858.7</td>
<td>1.10</td>
</tr>
<tr>
<td>700</td>
<td>100</td>
<td>30</td>
<td>15905.8</td>
<td>15654.8</td>
<td>1.58</td>
</tr>
<tr>
<td>200</td>
<td>1000</td>
<td>30</td>
<td>41371.6</td>
<td>41315.2</td>
<td>0.14</td>
</tr>
<tr>
<td>300</td>
<td>1000</td>
<td>30</td>
<td>55013.4</td>
<td>54448.5</td>
<td>1.03</td>
</tr>
<tr>
<td>400</td>
<td>1000</td>
<td>30</td>
<td>85979.6</td>
<td>85922.9</td>
<td>0.07</td>
</tr>
<tr>
<td>500</td>
<td>1000</td>
<td>30</td>
<td>101751.8</td>
<td>100773.3</td>
<td>0.77</td>
</tr>
<tr>
<td>600</td>
<td>1000</td>
<td>30</td>
<td>132314.3</td>
<td>131733.7</td>
<td>0.44</td>
</tr>
<tr>
<td>700</td>
<td>1000</td>
<td>30</td>
<td>141557.9</td>
<td>141012.6</td>
<td>0.39</td>
</tr>
<tr>
<td>200</td>
<td>100</td>
<td>60</td>
<td>71749.0</td>
<td>71749.0</td>
<td>0.00</td>
</tr>
<tr>
<td>300</td>
<td>100</td>
<td>60</td>
<td>9726.0</td>
<td>9726.0</td>
<td>0.00</td>
</tr>
<tr>
<td>400</td>
<td>100</td>
<td>60</td>
<td>15232.4</td>
<td>15230.2</td>
<td>0.01</td>
</tr>
<tr>
<td>500</td>
<td>100</td>
<td>60</td>
<td>18260.4</td>
<td>18251.3</td>
<td>0.05</td>
</tr>
<tr>
<td>600</td>
<td>100</td>
<td>60</td>
<td>23357.4</td>
<td>23336.7</td>
<td>0.09</td>
</tr>
<tr>
<td>700</td>
<td>100</td>
<td>60</td>
<td>24192.3</td>
<td>24181.6</td>
<td>0.04</td>
</tr>
<tr>
<td>200</td>
<td>1000</td>
<td>60</td>
<td>71556.9</td>
<td>71556.9</td>
<td>0.00</td>
</tr>
<tr>
<td>300</td>
<td>1000</td>
<td>60</td>
<td>109530.5</td>
<td>109494.8</td>
<td>0.03</td>
</tr>
<tr>
<td>400</td>
<td>1000</td>
<td>60</td>
<td>140994.9</td>
<td>140936.9</td>
<td>0.04</td>
</tr>
<tr>
<td>500</td>
<td>1000</td>
<td>60</td>
<td>178478.1</td>
<td>178500.4</td>
<td>-0.01</td>
</tr>
<tr>
<td>600</td>
<td>1000</td>
<td>60</td>
<td>214970.2</td>
<td>214875.1</td>
<td>0.04</td>
</tr>
<tr>
<td>700</td>
<td>1000</td>
<td>60</td>
<td>246489.6</td>
<td>246458.9</td>
<td>0.01</td>
</tr>
</tbody>
</table>

Average improvement 1.30
Table 2: Best results over ten runs.

<table>
<thead>
<tr>
<th>Problem</th>
<th>Results</th>
<th>Improvement (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>HAS-SOP</td>
<td>APC + HAS-SOP</td>
</tr>
<tr>
<td>n</td>
<td>r</td>
<td>p</td>
</tr>
<tr>
<td>200</td>
<td>100</td>
<td>1</td>
</tr>
<tr>
<td>300</td>
<td>100</td>
<td>1</td>
</tr>
<tr>
<td>400</td>
<td>100</td>
<td>1</td>
</tr>
<tr>
<td>500</td>
<td>100</td>
<td>1</td>
</tr>
<tr>
<td>600</td>
<td>100</td>
<td>1</td>
</tr>
<tr>
<td>700</td>
<td>100</td>
<td>1</td>
</tr>
<tr>
<td>200</td>
<td>1000</td>
<td>1</td>
</tr>
<tr>
<td>300</td>
<td>1000</td>
<td>1</td>
</tr>
<tr>
<td>400</td>
<td>1000</td>
<td>1</td>
</tr>
<tr>
<td>500</td>
<td>1000</td>
<td>1</td>
</tr>
<tr>
<td>600</td>
<td>1000</td>
<td>1</td>
</tr>
<tr>
<td>700</td>
<td>1000</td>
<td>1</td>
</tr>
<tr>
<td>200</td>
<td>100</td>
<td>15</td>
</tr>
<tr>
<td>300</td>
<td>100</td>
<td>15</td>
</tr>
<tr>
<td>400</td>
<td>100</td>
<td>15</td>
</tr>
<tr>
<td>500</td>
<td>100</td>
<td>15</td>
</tr>
<tr>
<td>600</td>
<td>100</td>
<td>15</td>
</tr>
<tr>
<td>700</td>
<td>100</td>
<td>15</td>
</tr>
<tr>
<td>200</td>
<td>1000</td>
<td>15</td>
</tr>
<tr>
<td>300</td>
<td>1000</td>
<td>15</td>
</tr>
<tr>
<td>400</td>
<td>1000</td>
<td>15</td>
</tr>
<tr>
<td>500</td>
<td>1000</td>
<td>15</td>
</tr>
<tr>
<td>600</td>
<td>1000</td>
<td>15</td>
</tr>
<tr>
<td>700</td>
<td>1000</td>
<td>15</td>
</tr>
<tr>
<td>200</td>
<td>100</td>
<td>30</td>
</tr>
<tr>
<td>300</td>
<td>100</td>
<td>30</td>
</tr>
<tr>
<td>400</td>
<td>100</td>
<td>30</td>
</tr>
<tr>
<td>500</td>
<td>100</td>
<td>30</td>
</tr>
<tr>
<td>600</td>
<td>100</td>
<td>30</td>
</tr>
<tr>
<td>700</td>
<td>100</td>
<td>30</td>
</tr>
<tr>
<td>200</td>
<td>1000</td>
<td>30</td>
</tr>
<tr>
<td>300</td>
<td>1000</td>
<td>30</td>
</tr>
<tr>
<td>400</td>
<td>1000</td>
<td>30</td>
</tr>
<tr>
<td>500</td>
<td>1000</td>
<td>30</td>
</tr>
<tr>
<td>600</td>
<td>1000</td>
<td>30</td>
</tr>
<tr>
<td>700</td>
<td>1000</td>
<td>30</td>
</tr>
<tr>
<td>200</td>
<td>100</td>
<td>60</td>
</tr>
<tr>
<td>300</td>
<td>100</td>
<td>60</td>
</tr>
<tr>
<td>400</td>
<td>100</td>
<td>60</td>
</tr>
<tr>
<td>500</td>
<td>100</td>
<td>60</td>
</tr>
<tr>
<td>600</td>
<td>100</td>
<td>60</td>
</tr>
<tr>
<td>700</td>
<td>100</td>
<td>60</td>
</tr>
<tr>
<td>200</td>
<td>1000</td>
<td>60</td>
</tr>
<tr>
<td>300</td>
<td>1000</td>
<td>60</td>
</tr>
<tr>
<td>400</td>
<td>1000</td>
<td>60</td>
</tr>
<tr>
<td>500</td>
<td>1000</td>
<td>60</td>
</tr>
<tr>
<td>600</td>
<td>1000</td>
<td>60</td>
</tr>
<tr>
<td>700</td>
<td>1000</td>
<td>60</td>
</tr>
</tbody>
</table>

Average improvement 2.11
Table 3: Worst results over ten runs.

<table>
<thead>
<tr>
<th>Problem</th>
<th>Results</th>
<th>Improvement</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>n r p</td>
<td>HAS-SOP</td>
<td>APC + HAS-SOP (%)</td>
<td></td>
</tr>
<tr>
<td>200 100 1</td>
<td>93 94</td>
<td>-1.08</td>
<td></td>
</tr>
<tr>
<td>300 100 1</td>
<td>79 76</td>
<td>3.80</td>
<td></td>
</tr>
<tr>
<td>400 100 1</td>
<td>67 67</td>
<td>0.00</td>
<td></td>
</tr>
<tr>
<td>500 100 1</td>
<td>62 58</td>
<td>6.45</td>
<td></td>
</tr>
<tr>
<td>600 100 1</td>
<td>54 53</td>
<td>1.85</td>
<td></td>
</tr>
<tr>
<td>700 100 1</td>
<td>44 44</td>
<td>0.00</td>
<td></td>
</tr>
<tr>
<td>200 1000 1</td>
<td>1572 1576</td>
<td>-0.25</td>
<td></td>
</tr>
<tr>
<td>300 1000 1</td>
<td>1604 1633</td>
<td>-1.81</td>
<td></td>
</tr>
<tr>
<td>400 1000 1</td>
<td>1864 1816</td>
<td>2.58</td>
<td></td>
</tr>
<tr>
<td>500 1000 1</td>
<td>1921 1888</td>
<td>1.72</td>
<td></td>
</tr>
<tr>
<td>600 1000 1</td>
<td>2025 2001</td>
<td>1.19</td>
<td></td>
</tr>
<tr>
<td>700 1000 1</td>
<td>2013 2013</td>
<td>0.00</td>
<td></td>
</tr>
<tr>
<td>200 100 15</td>
<td>2186 2182</td>
<td>0.18</td>
<td></td>
</tr>
<tr>
<td>300 100 15</td>
<td>4013 3883</td>
<td>3.24</td>
<td></td>
</tr>
<tr>
<td>400 100 15</td>
<td>5292 5135</td>
<td>2.97</td>
<td></td>
</tr>
<tr>
<td>500 100 15</td>
<td>7146 7031</td>
<td>1.86</td>
<td></td>
</tr>
<tr>
<td>600 100 15</td>
<td>7992 8030</td>
<td>-0.48</td>
<td></td>
</tr>
<tr>
<td>700 100 15</td>
<td>9949 9762</td>
<td>1.88</td>
<td></td>
</tr>
<tr>
<td>200 1000 15</td>
<td>24211 24531</td>
<td>5.29</td>
<td></td>
</tr>
<tr>
<td>300 1000 15</td>
<td>36808 35687</td>
<td>3.05</td>
<td></td>
</tr>
<tr>
<td>400 1000 15</td>
<td>51051 49220</td>
<td>3.59</td>
<td></td>
</tr>
<tr>
<td>500 1000 15</td>
<td>65177 64765</td>
<td>0.63</td>
<td></td>
</tr>
<tr>
<td>600 1000 15</td>
<td>75082 72542</td>
<td>3.38</td>
<td></td>
</tr>
<tr>
<td>700 1000 15</td>
<td>90808 84321</td>
<td>7.14</td>
<td></td>
</tr>
<tr>
<td>200 100 30</td>
<td>4289 4256</td>
<td>0.30</td>
<td></td>
</tr>
<tr>
<td>300 100 30</td>
<td>6383 6257</td>
<td>1.28</td>
<td></td>
</tr>
<tr>
<td>400 100 30</td>
<td>8607 8639</td>
<td>0.67</td>
<td></td>
</tr>
<tr>
<td>500 100 30</td>
<td>10845 10362</td>
<td>4.45</td>
<td></td>
</tr>
<tr>
<td>600 100 30</td>
<td>13311 13084</td>
<td>1.71</td>
<td></td>
</tr>
<tr>
<td>700 100 30</td>
<td>16135 15968</td>
<td>1.04</td>
<td></td>
</tr>
<tr>
<td>200 1000 30</td>
<td>41544 41516</td>
<td>0.07</td>
<td></td>
</tr>
<tr>
<td>300 1000 30</td>
<td>56606 54725</td>
<td>3.32</td>
<td></td>
</tr>
<tr>
<td>400 1000 30</td>
<td>86018 86487</td>
<td>0.50</td>
<td></td>
</tr>
<tr>
<td>500 1000 30</td>
<td>104420 102118</td>
<td>2.20</td>
<td></td>
</tr>
<tr>
<td>600 1000 30</td>
<td>135642 133236</td>
<td>1.77</td>
<td></td>
</tr>
<tr>
<td>700 1000 30</td>
<td>148310 143460</td>
<td>3.27</td>
<td></td>
</tr>
<tr>
<td>200 100 60</td>
<td>71749 71749</td>
<td>0.00</td>
<td></td>
</tr>
<tr>
<td>300 100 60</td>
<td>9726 9726</td>
<td>0.00</td>
<td></td>
</tr>
<tr>
<td>400 100 60</td>
<td>15250 15250</td>
<td>0.00</td>
<td></td>
</tr>
<tr>
<td>500 100 60</td>
<td>18303 18278</td>
<td>0.14</td>
<td></td>
</tr>
<tr>
<td>600 100 60</td>
<td>23370 23440</td>
<td>-0.30</td>
<td></td>
</tr>
<tr>
<td>700 1000 60</td>
<td>24249 24288</td>
<td>0.05</td>
<td></td>
</tr>
<tr>
<td>200 1000 60</td>
<td>71556 71556</td>
<td>0.00</td>
<td></td>
</tr>
<tr>
<td>300 1000 60</td>
<td>109590 109590</td>
<td>0.00</td>
<td></td>
</tr>
<tr>
<td>400 1000 60</td>
<td>141118 141118</td>
<td>0.00</td>
<td></td>
</tr>
<tr>
<td>500 1000 60</td>
<td>178679 178040</td>
<td>-0.09</td>
<td></td>
</tr>
<tr>
<td>600 1000 60</td>
<td>215389 215348</td>
<td>0.02</td>
<td></td>
</tr>
<tr>
<td>700 1000 60</td>
<td>247045 247045</td>
<td>0.00</td>
<td></td>
</tr>
</tbody>
</table>

Average improvement 1.41
no obvious advantage in further reducing the search spaces themselves.

The study of the results for different values of $n$ (number of nodes of the graph) seems to indicate that the improvements obtained are independent of the size of the problems.

Finally, it can be observed that no significant differences seem to exist in the improvements guaranteed by artificial precedence constraints for problems created with different values of parameter $r$ (range of the costs). This is an indication that the performances of the manipulation technique, and in turn of HAS-SOP, tend to be independent with respect to parameter $r$.

Table 2 is the analogue of Table 1, where the best results over the ten runs considered are reported instead of the average results. The same conclusions drawn on Table 1 also apply to Table 2. The only difference is that in the latter case the improvement seems to be amplified (e.g., the average improvement is 2.11% vs 1.30%). This confirms that the use of the manipulation technique really leads to better results when the search space is properly modified.

Table 3 suggests that also when the worst results over the ten runs considered are taken into account, the improvements guaranteed by artificial precedence constraints are in line with those obtained for the average results. This aspect is important because it indicates that artificial precedence constraints do not tend to hide “good” solutions.

Statistical tests on the results provided by the two algorithms considered have also been performed. A paired $t$-test indicates that the difference between the average results of the two methods is extremely significant ($p = 0.000067$), and is clearly in favour of $APC + HAS-SOP$.

Finally, in Table 4 the average computation times required by HAS-SOP and APC + HAS-SOP to retrieve their best solutions are reported. It might appear that artificial precedence constraints make convergence slower, but it is also important to remember that better solutions are retrieved by APC + HAS-SOP. This provides a good justification for the increased computation times. It is finally interesting to compare the computation times on those problems for which the two methods have exactly the same performance (e.g., $n, r, p$
Table 4: Average computation times over ten runs.

<table>
<thead>
<tr>
<th>Problem</th>
<th>Results</th>
<th>Improvement (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>HAS-SOP</td>
<td>APC + HAS-SOP</td>
</tr>
<tr>
<td>200</td>
<td>100</td>
<td>1</td>
</tr>
<tr>
<td>300</td>
<td>100</td>
<td>1</td>
</tr>
<tr>
<td>400</td>
<td>100</td>
<td>1</td>
</tr>
<tr>
<td>500</td>
<td>100</td>
<td>1</td>
</tr>
<tr>
<td>600</td>
<td>100</td>
<td>1</td>
</tr>
<tr>
<td>700</td>
<td>100</td>
<td>1</td>
</tr>
<tr>
<td>200</td>
<td>1000</td>
<td>1</td>
</tr>
<tr>
<td>300</td>
<td>1000</td>
<td>1</td>
</tr>
<tr>
<td>400</td>
<td>1000</td>
<td>1</td>
</tr>
<tr>
<td>500</td>
<td>1000</td>
<td>1</td>
</tr>
<tr>
<td>600</td>
<td>1000</td>
<td>1</td>
</tr>
<tr>
<td>700</td>
<td>1000</td>
<td>1</td>
</tr>
<tr>
<td>200</td>
<td>100</td>
<td>15</td>
</tr>
<tr>
<td>300</td>
<td>100</td>
<td>15</td>
</tr>
<tr>
<td>400</td>
<td>100</td>
<td>15</td>
</tr>
<tr>
<td>500</td>
<td>100</td>
<td>15</td>
</tr>
<tr>
<td>600</td>
<td>100</td>
<td>15</td>
</tr>
<tr>
<td>700</td>
<td>100</td>
<td>15</td>
</tr>
<tr>
<td>200</td>
<td>100</td>
<td>30</td>
</tr>
<tr>
<td>300</td>
<td>100</td>
<td>30</td>
</tr>
<tr>
<td>400</td>
<td>100</td>
<td>30</td>
</tr>
<tr>
<td>500</td>
<td>100</td>
<td>30</td>
</tr>
<tr>
<td>600</td>
<td>100</td>
<td>30</td>
</tr>
<tr>
<td>700</td>
<td>100</td>
<td>30</td>
</tr>
<tr>
<td>200</td>
<td>100</td>
<td>60</td>
</tr>
<tr>
<td>300</td>
<td>100</td>
<td>60</td>
</tr>
<tr>
<td>400</td>
<td>100</td>
<td>60</td>
</tr>
<tr>
<td>500</td>
<td>100</td>
<td>60</td>
</tr>
<tr>
<td>600</td>
<td>100</td>
<td>60</td>
</tr>
<tr>
<td>700</td>
<td>100</td>
<td>60</td>
</tr>
<tr>
<td>200</td>
<td>1000</td>
<td>1</td>
</tr>
<tr>
<td>300</td>
<td>1000</td>
<td>1</td>
</tr>
<tr>
<td>400</td>
<td>1000</td>
<td>1</td>
</tr>
<tr>
<td>500</td>
<td>1000</td>
<td>1</td>
</tr>
<tr>
<td>600</td>
<td>1000</td>
<td>1</td>
</tr>
<tr>
<td>700</td>
<td>1000</td>
<td>1</td>
</tr>
<tr>
<td>200</td>
<td>1000</td>
<td>15</td>
</tr>
<tr>
<td>300</td>
<td>1000</td>
<td>15</td>
</tr>
<tr>
<td>400</td>
<td>1000</td>
<td>15</td>
</tr>
<tr>
<td>500</td>
<td>1000</td>
<td>15</td>
</tr>
<tr>
<td>600</td>
<td>1000</td>
<td>15</td>
</tr>
<tr>
<td>700</td>
<td>1000</td>
<td>15</td>
</tr>
<tr>
<td>200</td>
<td>1000</td>
<td>30</td>
</tr>
<tr>
<td>300</td>
<td>1000</td>
<td>30</td>
</tr>
<tr>
<td>400</td>
<td>1000</td>
<td>30</td>
</tr>
<tr>
<td>500</td>
<td>1000</td>
<td>30</td>
</tr>
<tr>
<td>600</td>
<td>1000</td>
<td>30</td>
</tr>
<tr>
<td>700</td>
<td>1000</td>
<td>30</td>
</tr>
<tr>
<td>200</td>
<td>1000</td>
<td>60</td>
</tr>
<tr>
<td>300</td>
<td>1000</td>
<td>60</td>
</tr>
<tr>
<td>400</td>
<td>1000</td>
<td>60</td>
</tr>
<tr>
<td>500</td>
<td>1000</td>
<td>60</td>
</tr>
<tr>
<td>600</td>
<td>1000</td>
<td>60</td>
</tr>
<tr>
<td>700</td>
<td>1000</td>
<td>60</td>
</tr>
</tbody>
</table>

Average improvement: -10.69%
= 200, 100, 60; 300, 100, 60; 200, 1000, 60). For these problems the extra time required by \textit{APC + HAS-SOP} to find the best solution (a few seconds) is almost certainly due to the computational overhead derived by the management of artificial precedence constraints.

5 Conclusions

A problem manipulation method, which creates and adds artificial precedence constraints to the original problem, has been tested on top of a well-known ant colony system algorithm for the sequential ordering problem. The manipulation technique induces improvements in the performance of the underlying algorithm, leading to better results, especially on large and difficult problems.

It is important to observe that the proposed problem manipulation method is not only applicable to ant colony systems for SOP. It can also be adapted to many other combinatorial optimization methods. The key issue is to identify families of artificial constraints (connected to the structure of the problem under investigation) that can be used to intelligently modify the solution space.

References


For each pair \((r, s)\)
\[ m_{rs} := 0 \]
EndFor

counter := 1

\(R_A := \emptyset\)

While (exit criterion not met) do

\(OptPath_k := \) Solution returned by the underlying heuristic method

Compute \(L_k\) /* \(L_k\) is the length of the path \(OptPath_k\) */

For each node \(r, s \in V\) such that \(\pi_k(r) < \pi_k(s) \leq \pi_k(r) + 5\) and \((r, s) \notin R\) /* \(\pi_k(i)\) is the index of the position of node \(i\) in solution \(OptPath_k\) */

\[ m_{rs} := m_{rs} + L_i/L_k \] /* This is equation (1) */

\[ m_{sr} := m_{sr} - L_i/L_k \] /* This is equation (2) */

EndFor

counter := counter + 1

If (counter \mod v) == u)

If (counter \neq u)

For \(i:=1\) to \(z\)

\((r, s) = \min_{(j,k) \in A, (j,k) \in RA} \{m_{jk}\}\)

If \((m_{rs} \geq 0)\)

\(R := R \setminus \{(r,s)\}\)

\(R_A := R_A \setminus \{(r,s)\}\)

Else

\(i := w\) /* Forcing the exit from the For loop */

EndIf

EndFor

\(j := z\)

Else

\(j := w\)

EndIf

For \(i:=1\) to \(j\)

\((r, s) = \max_{(j,k) \in A, (j,k) \in R} \{m_{jk}\}\)

If \((m_{rs} \geq 0)\)

\(R := R \cup \{(r,s)\}\)

\(R_A := R_A \cup \{(r,s)\}\)

Else

\(i := j\) /* Forcing the exit from the For loop */

EndIf

EndFor

EndIf

EndWhile

Let \(L_{best}\) be the shortest \(L_k\) from beginning and \(OptPath_{best}\) the corresponding path

Print \(L_{best}\) and \(OptPath_{best}\)

Figure 1: Pseudo-code for the problem manipulation approach for the SOP.
Figure 2: Tuning of parameter $u$, with default value 100. Average (a) and best (b) results over five runs are presented. The vertical axis shows the ratio of the solution cost to the solution cost obtained with the default value of parameter $u$.

Figure 3: Tuning of parameter $v$, with default value 50. Average (a) and best (b) results over five runs are presented. The vertical axis shows the ratio of the solution cost to the solution cost obtained with the default value of parameter $v$. 

23
Figure 4: Tuning of parameter $w$, with default value 20. Average (a) and best (b) results over five runs are reported. The vertical axis shows the ratio of the solution cost to the solution cost obtained with the default value of parameter $w$.

Figure 5: Tuning of parameter $z$, with default value 5. Average (a) and best (b) results over five runs are reported. The vertical axis shows the ratio of the solution cost to the solution cost obtained with the default value of parameter $z$. 
Figure 6: Percentage improvements for different values of $p$. Average results over ten runs are presented.