Author's Accepted Manuscript

www.elsevier.com/locate/cor

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

Cite this article as: R. Montemanni, D.H. Smith and L.M. Gambardella, A heuristic manipulation technique for the sequential ordering problem, *Computers & Operations Research* (2007), doi:10.1016/j.cor.2007.05.003

A Heuristic Manipulation Technique for the Sequential Ordering Problem

R. Montemanni[a,*], 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 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

*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 $A = \{(i, j)|i, j \in V\}$ is the set of arcs. A cost $c_{ij} \in \mathcal{N}$ 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 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 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 : the number of solutions generated by the underlying heuristic method before the first artificial precedence constraints are added to the problem;

v : the number of solutions generated by the underlying heuristic method between two consecutive updates to the set of active artificial precedence constraints;

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

z : the (approximate) number of artificial precedence constraints substituted after every v new solutions have been generated by the underlying heuristic algorithm (after the first u 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 $OptPath_k$, with cost L_k, is generated by the underlying heuristic algorithm, matrix $m = [m_{ij}]$ is updated as follows:

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

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

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

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:

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

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:

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

where τ_0 is the initial value of trails. It was found that good values for the algorithm's parameters are $\tau_0 = (FirstSolution \cdot |V|\ )^{-1}$ 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.

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

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

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

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-

ered. 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 250 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.

Problem			Results		Improvement
n	r	p	HAS-SOP	APC + HAS-SOP	(%)
200	100	1	90.3	88.4	2.10
300	100	1	76.4	71.9	5.89
400	100	1	64.1	61.7	3.74
500	100	1	55.0	55.1	-0.18
600	100	1	49.8	47.6	4.42
700	100	1	42.6	39.5	7.28
200	1000	1	1549.5	1551.5	-0.13
300	1000	1	1586.7	1584.9	0.11
400	1000	1	1811.1	1773.1	2.10
500	1000	1	1877.4	1838.4	2.08
600	1000	1	1986.7	1957.6	1.46
700	1000	1	1969.2	1943.1	1.33
200	100	15	2066.0	2025.1	1.98
300	100	15	3738.6	3648.2	2.42
400	100	15	5087.1	4917.7	3.33
500	100	15	6931.5	6826.4	1.52
600	100	15	7806.6	7733.0	0.94
700	100	15	9573.0	9453.7	1.25
200	1000	15	22602.9	22419.2	0.81
300	1000	15	34447.9	34215.0	0.68
400	1000	15	46638.6	46183.2	0.98
500	1000	15	62693.7	61675.7	1.62
600	1000	15	72701.1	70136.9	3.53
700	1000	15	85177.7	82697.0	2.91
200	100	30	4254.6	4236.0	0.44
300	100	30	6228.2	6180.2	0.77
400	100	30	8476.5	8395.3	0.96
500	100	30	10333.2	10108.6	2.17
600	100	30	13001.8	12858.7	1.10
700	100	30	15905.8	15654.8	1.58
200	1000	30	41371.6	41315.2	0.14
300	1000	30	55013.4	54448.5	1.03
400	1000	30	85979.6	85922.9	0.07
500	1000	30	101751.8	100973.3	0.77
600	1000	30	132314.3	131733.7	0.44
700	1000	30	141557.9	141012.6	0.39
200	100	60	71749.0	71749.0	0.00
300	100	60	9726.0	9726.0	0.00
400	100	60	15232.4	15230.2	0.01
500	100	60	18260.4	18251.3	0.05
600	100	60	23357.4	23336.7	0.09
700	100	60	24192.3	24181.6	0.04
200	1000	60	71556.0	71556.0	0.00
300	1000	60	109530.5	109494.8	0.03
400	1000	60	140994.9	140936.9	0.04
500	1000	60	178478.1	178500.4	-0.01
600	1000	60	214970.2	214875.1	0.04
700	1000	60	246489.6	246458.9	0.01
				Average improvement	1.30

Table 2: Best results over ten runs.

Problem			Results		Improvement
n	r	p	HAS-SOP	APC + HAS-SOP	(%)
200	100	1	88	83	5.68
300	100	1	74	65	12.16
400	100	1	59	54	8.47
500	100	1	51	51	0.00
600	100	1	44	42	4.55
700	100	1	41	32	21.95
200	1000	1	1532	1525	0.46
300	1000	1	1536	1518	1.17
400	1000	1	1783	1714	3.87
500	1000	1	1840	1770	3.80
600	1000	1	1936	1922	0.72
700	1000	1	1912	1873	2.04
200	100	15	2002	1926	3.80
300	100	15	3520	3383	3.89
400	100	15	4838	4735	2.13
500	100	15	6584	6536	0.73
600	100	15	7610	7217	5.16
700	100	15	9383	9124	2.76
200	1000	15	21775	21654	0.56
300	1000	15	33533	33148	1.15
400	1000	15	45055	44749	0.68
500	1000	15	60175	58316	3.09
600	1000	15	70454	68219	3.17
700	1000	15	81439	80887	0.68
200	100	30	4247	4216	0.73
300	100	30	6151	6148	0.05
400	100	30	8289	8276	0.16
500	100	30	10047	9974	0.73
600	100	30	12810	12689	0.94
700	100	30	15733	15180	3.51
200	1000	30	41278	41196	0.20
300	1000	30	54367	54278	0.16
400	1000	30	85579	85288	0.34
500	1000	30	100453	100112	0.34
600	1000	30	130244	130541	-0.23
700	1000	30	139769	138695	0.77
200	100	60	71749	71749	0.00
300	100	60	9726	9726	0.00
400	100	60	15228	15228	0.00
500	100	60	18246	18240	0.03
600	100	60	23342	23259	0.36
700	100	60	24151	24149	0.01
200	1000	60	71556	71556	0.00
300	1000	60	109471	109471	0.00
400	1000	60	140862	140816	0.03
500	1000	60	178323	178212	0.06
600	1000	60	214724	214608	0.05
700	1000	60	246128	245753	0.15
Average improvement					2.11

Table 3: Worst results over ten runs.

Problem			Results		Improvement
n	r	p	HAS-SOP	APC + HAS-SOP	(%)
200	100	1	93	94	-1.08
300	100	1	79	76	3.80
400	100	1	67	67	0.00
500	100	1	62	58	6.45
600	100	1	54	53	1.85
700	100	1	44	44	0.00
200	1000	1	1572	1576	-0.25
300	1000	1	1604	1633	-1.81
400	1000	1	1864	1816	2.58
500	1000	1	1921	1888	1.72
600	1000	1	2025	2001	1.19
700	1000	1	2013	2013	0.00
200	100	15	2186	2182	0.18
300	100	15	4013	3883	3.24
400	100	15	5292	5135	2.97
500	100	15	7146	7013	1.86
600	100	15	7992	8030	-0.48
700	100	15	9949	9762	1.88
200	1000	15	24211	22931	5.29
300	1000	15	36808	35687	3.05
400	1000	15	51051	49220	3.59
500	1000	15	65177	64765	0.63
600	1000	15	75082	72542	3.38
700	1000	15	90808	84321	7.14
200	100	30	4269	4256	0.30
300	100	30	6338	6257	1.28
400	100	30	8697	8639	0.67
500	100	30	10845	10362	4.45
600	100	30	13311	13084	1.71
700	100	30	16135	15968	1.04
200	1000	30	41544	41516	0.07
300	1000	30	56606	54725	3.32
400	1000	30	86918	86487	0.50
500	1000	30	104420	102118	2.20
600	1000	30	135642	133236	1.77
700	1000	30	148310	143460	3.27
200	100	60	71749	71749	0.00
300	100	60	9726	9726	0.00
400	100	60	15250	15250	0.00
500	100	60	18303	18278	0.14
600	100	60	23370	23440	-0.30
700	100	60	24249	24238	0.05
200	1000	60	71556	71556	0.00
300	1000	60	109590	109590	0.00
400	1000	60	141118	141118	0.00
500	1000	60	178679	178840	-0.09
600	1000	60	215389	215348	0.02
700	1000	60	247045	247045	0.00
				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.

Problem			Results		Improvement
n	r	p	HAS-SOP	APC + HAS-SOP	(%)
200	100	1	244.9	308.2	-25.85
300	100	1	338.1	279.9	17.21
400	100	1	291.4	330.1	-13.28
500	100	1	286.0	300.3	-5.00
600	100	1	368.0	284.3	22.74
700	100	1	295.4	297.2	-0.61
200	1000	1	312.1	422.5	-35.37
300	1000	1	357.5	349.0	2.38
400	1000	1	395.4	325.8	17.60
500	1000	1	417.0	352.9	15.37
600	1000	1	386.0	438.3	-13.55
700	1000	1	421.9	457.3	-8.39
200	100	15	260.9	391.4	-50.02
300	100	15	276.2	364.5	-31.97
400	100	15	159.4	195.6	-22.71
500	100	15	237.7	292.0	-22.84
600	100	15	167.4	252.3	-50.72
700	100	15	152.9	220.3	-44.08
200	1000	15	425.7	347.2	18.44
300	1000	15	315.0	370.7	-17.68
400	1000	15	389.8	463.9	-19.01
500	1000	15	407.1	376.4	7.54
600	1000	15	387.2	401.4	-3.67
700	1000	15	312.4	335.8	-7.49
200	100	30	167.1	263.6	-57.75
300	100	30	318.0	364.3	-14.56
400	100	30	275.8	407.4	-47.72
500	100	30	350.2	360.2	-2.86
600	100	30	440.4	404.8	8.08
700	100	30	105.1	185.2	-76.21
200	1000	30	314.4	379.9	-20.83
300	1000	30	374.5	430.4	-14.93
400	1000	30	524.0	523.1	0.17
500	1000	30	446.6	444.8	0.40
600	1000	30	427.1	374.2	12.39
700	1000	30	426.1	465.4	-9.22
200	100	60	2.2	3.1	-40.91
300	100	60	32.3	34.2	-5.88
400	100	60	270.0	175.6	34.96
500	100	60	369.1	292.3	20.81
600	100	60	406.3	423.2	-4.16
700	100	60	500.0	471.6	5.68
200	1000	60	92.4	106.6	-15.37
300	1000	60	49.8	27.8	44.18
400	1000	60	217.8	326.4	-49.86
500	1000	60	337.5	369.9	-9.60
600	1000	60	398.8	411.8	-3.26
700	1000	60	410.8	392.7	4.41
				Average improvement	-10.69

$= 200, 100, 60;\ 300, 100, 60;\ 200, 1000, 60)$. For these problems the extra time required by *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

[1] N. Ascheuer. *Hamiltonian path problems in the on-line 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] R. Montemanni, D.H. Smith, and L.M. Gambardella. Ant Colony Systems for large Sequential Ordering Problems. In *Proceedings of IEEE Swarm Intelligence Symposium*, 2007.

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

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

[16] 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.

```
For each pair (r, s)
    m_rs := 0
EndFor
counter := 1
R_A := ∅
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 ∈ V such that π_k(r) < π_k(s) ≤ π_k(r) + 5 and (r, s) ∉ R
    /* π_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 (1) */
        m_sr := m_sr − L_1/L_k /* This is equation (2) */
    EndFor
    counter := counter + 1
    If (counter mod v) == u)
        If (counter ≠ u)
            For i:=1 to z
                (r, s) = argmin_{(j,k)∈A,(j,k)∈R_A}{m_jk}
                If (m_rs ≥ 0)
                    R := R\{(r, s)}
                    R_A := R_A\{(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) = argmax_{(j,k)∈A,(j,k)∉R}{m_jk}
            If (m_rs ≥ 0)
                R := R ∪ {(r, s)}
                R_A := R_A ∪ {(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.*

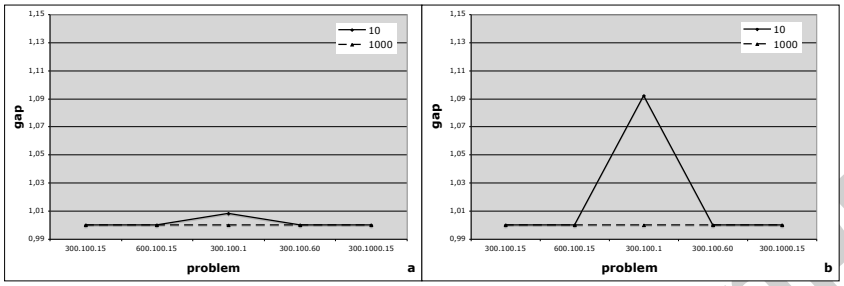

Figure 2: Tuning of parameter u, with default value 100. Average (a) and best (b) results over five runs are presented. The vertical axis shows the ratio of the solution cost to the solution cost obtained with the default value of parameter u.

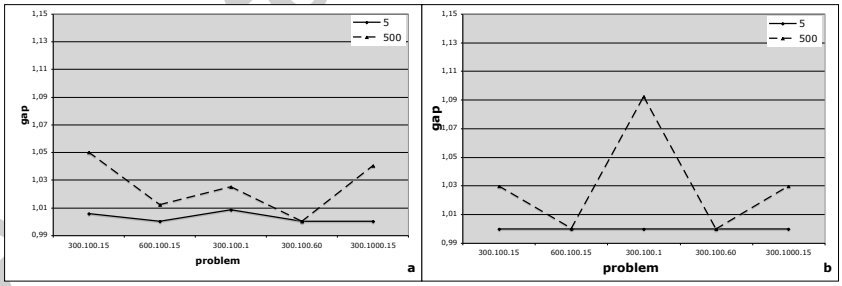

Figure 3: Tuning of parameter v, with default value 50. Average (a) and best (b) results over five runs are presented. The vertical axis shows the ratio of the solution cost to the solution cost obtained with the default value of parameter v.

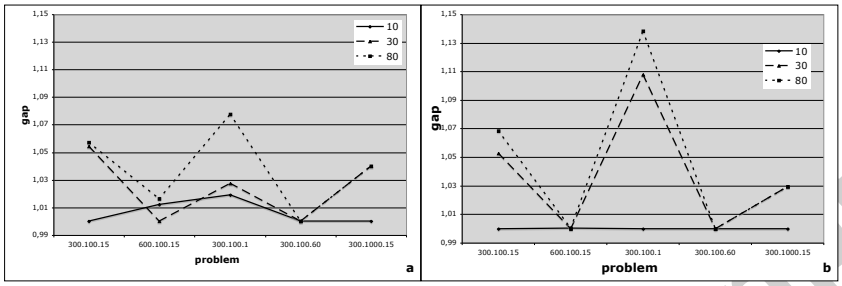

Figure 4: Tuning of parameter w, with default value 20. Average (a) and best (b) results over five runs are reported. The vertical axis shows the ratio of the solution cost to the solution cost obtained with the default value of parameter w.

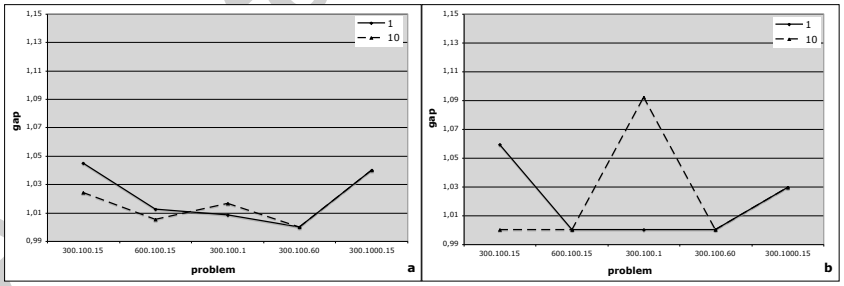

Figure 5: Tuning of parameter z, with default value 5. Average (a) and best (b) results over five runs are reported. The vertical axis shows the ratio of the solution cost to the solution cost obtained with the default value of parameter z.

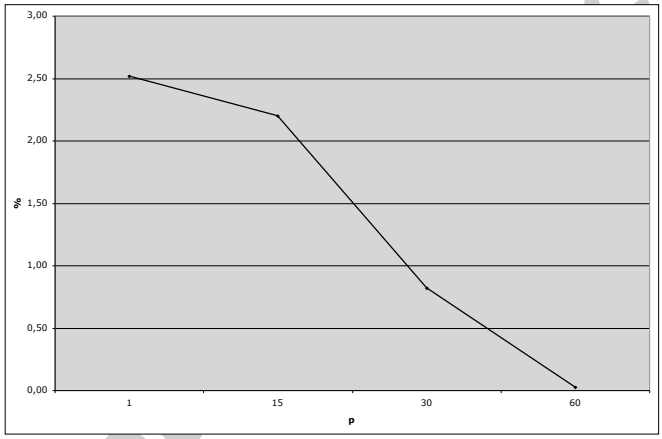

Figure 6: Percentage improvements for different values of p. Average results over ten runs are presented.