Lecture Notes in Management Science (2012) Vol. **4**: 266–273
4th International Conference on Applied Operational Research, Proceedings
© Tadbir Operational Research Group Ltd. All rights reserved. www.tadbir.ca

ISSN 2008-0050 (Print), ISSN 1927-0097 (Online)

A branch and bound approach for the sequential ordering problem

Marco Mojana, Roberto Montemanni, Gianni Di Caro and Luca M. Gambardella

Dalle Molle Institute for Artificial Intelligence (IDSIA), Manno, Switzerland
Università della Svizzera Italiana, Lugano, Switzerland
marco.mojana@usi.ch; roberto@idsia.ch

Abstract. Many different real problems can be modeled in mathematical terms as a Sequential Ordering Problem. This combinatorial optimization problem can be seen as a scheduling problem with precedence constraints among the jobs. In the present paper a branch and bound method that exploits the structure of the precedence constraints is introduced and tested on some benchmark instances commonly adopted in the literature.

Keywords: sequential ordering problem; mixed integer linear programming; branch and bound; traveling salesman problem with precedence constraints

Introduction

The Sequential Ordering Problem (SOP) is a combinatorial optimization problem. Given a directed graph and a set of precedence constraints between pairs of vertices, it requires to find the minimum cost Hamiltonian tour. Without loss of generality, we can assume that a fixed starting vertex 1 is given. When a tour visits the vertices following the given precedences, it is declared feasible. Practical problems that can be modeled as SOPs are those that ask to find a permutation of a set of jobs that minimizes the total time or cost required to complete them. This sort of problems commonly arises in all the stages of the life cycle of a commercial product and the ability to solve them efficiently represents a substantial edge over the competitors. For example [1] focuses on removing the bottlenecks created by a naïve management of a stacker crane in a manufacturing system. The abstract formulation naturally adapts also to many transportation problems, as in [10], where the authors illustrates how to minimize the distance flown by a helicopter that must connect some offshore platforms. Another field where the knowledge of good SOP algorithms makes the difference is the scheduling of the load/unload operations in container terminals, as described in [8]. The problem has been initially formulated in [3] as the underlying model for a production planning system. At the beginning, the precedences among the vertices were simply ignored: this way the problem could be solved with the assignment problem formulation for ATSP. If the solution was not feasible, a distinct procedure took care of the fixing. At that point was clear the importance of refining the mathematical models with inequalities able to exploit the presence of a partial order to provide tighter bounds and to remove infeasible sequences. This was the direction taken by [2], that illustrated how to generate valid cuts by means of a polynomial time separation algorithm. The cutting plane approach was used, in combination with a Lagrangian relaxation, in [4]. The work presented in [1] contains a huge collection of known and new valid cuts for the ATSP and SOP. It reports also the performances of a branch and cut solver applied on real problems faced in the management of manufacturing systems. A matheuristic approach ([5]) combining a Mixed Integer Linear Programming solver and an Ant System in a simple but effective way has been described in [7]. We refer the interested reader to [6] for a more complete and detailed literature review.

In this paper we present a branch and bound approach for the SOP which exploits the structure of the precedences to decompose the problem in smaller sub-problems. Such an approach is tailored for instances dominated by precedences, like those of the SOPLIB2006 dataset (available at http://www.idsia.ch/~roberto/SOPLIB06.zip).

Problem description

The SOP can be modeled in graph theoretical terms as follows. A complete directed graph $D = (V, A)$ is given, where V is the set of vertices and $A = \{(i, j) \mid i, j \in V\}$ is the set of arcs. A cost $c_{ij} \in N$ is associated with each arc $(i, j) \in A$. A fixed starting vertex $1 \in V$, that by definition is the first and last vertex of the solution-tour. Furthermore an additional precedence digraph $P = (V, R)$ is given, defined on the same vertex 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. The precedence digraph P must be acyclic in order for a feasible solution to exist, and is also assumed to be transitively closed. By definition, for the last arc traversed by a tour (entering vertex 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. Notice that the SOP reduces to the classical Asymmetric Travelling Salesman problem when no precedence constraint is given. The problem can be represented as the following Mixed Integer Linear Program (MILP):

$$\text{Minimize} \sum_{i=1}^{|V|}\sum_{j=1}^{|V|} \frac{1}{|V|-1} c_{ij} \left(y_{ij}^{p} + y_{ij}^{q}\right) \qquad (1)$$

s.t.

$$\sum_{j=1}^{|V|} y_{ij}^{p} - \sum_{j=1}^{|V|} y_{ji}^{p} = \begin{cases} |V|-1 & \text{for } i=1, \\ -1 & \text{otherwise} \end{cases} \quad \forall i \in V \qquad (2)$$

$$\sum_{j=1}^{|V|} y_{ij}^{q} - \sum_{j=1}^{|V|} y_{ji}^{q} = \begin{cases} 1-|V| & \text{for } i=1, \\ 1 & \text{otherwise} \end{cases} \quad \forall i \in V \qquad (3)$$

$$\sum_{j=1}^{|V|} y_{ij}^{p} + y_{ij}^{q} = |V|-1 \quad \forall i \in V \qquad (4)$$

$$y_{ij}^{p} + y_{ij}^{q} = (|V|-1)\, y_{ij} \quad \forall i,j \in V \qquad (5)$$

$$\sum_{j=1}^{|V|} y_{uj}^{p} - \sum_{j=1}^{|V|} y_{vj}^{p} \geq 1 \quad \forall (u,v) \in R \qquad (6)$$

$$y_{ij}^{p} \geq 0 \quad \forall i,j \in V \qquad (7)$$

$$y_{ij}^{q} \geq 0 \quad \forall i,j \in V \qquad (8)$$

$$y_{ij} \in \{0,1\} \quad \forall i,j \in V \qquad (9)$$

The model is based on the two-commodity network flow formulation discussed in [9], without any experimental study. The SOP can be seen as the problem of optimizing the tour of a pickup and delivery truck that must satisfy the needs of a set of customers. The starting vertex *1* produces $|V|-1$ units of commodity p and the other vertices consume one unit each. For the second commodity q, it is the other way round: the starting vertex *1* requires $|V|-1$ units and the others provide one unit each. To fulfill all the requests, a hypothetical vehicle starts from vertex *1* filled with $|V|-1$ units of commodity p, visits all the vertices in a certain sequence and for each one of them it delivers one unit of p and picks up a unit of q. When the vehicle returns to the depot, the collected quantity of q will perfectly satisfy vertex *1* needs. In the resulting mathematical formulation the variables y^{p}_{ij} (y^{q}_{ij}) correspond to the quantity of commodity p (q) carried from i to j. The binary variables y_{ij}, together with constraint (5), prevent the vehicle from “splitting” and following two different paths. The constraints (2) and (3) represent the flow balance equations. The vehicle total load remains constant during the tour, because when it stops at a vertex, it delivers one unit of p and picks up a unit of q. When it starts, it is loaded only with the $|V|-1$ units of commodity p produced by vertex *1* and no units of q (because it has not collected any from the vertices $2 \div n$ yet). Combining these two considerations, we impose a constant load with equation (4). Precedences are enforced in constraints (6), exploiting the fact that the quantity of commodity p decreases along the tour. Finally, the objective function simply does the summation of the cost of the traversed arcs. In our approach we use a set of inequalities (see [2]) that can be added to the formulation to strengthen its linear relaxation.

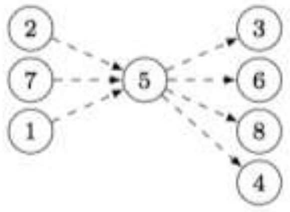
A branch and bound approach

The SOP shares with all the other NP-hard problems the characteristic that the complexity of solving an instance grows exponentially with its size (intended as number of vertices, i.e. $|V|$). In comparison to other similar combinatorial problems— like the asymmetric TSP — this problem requires an additional graph that defines the precedences between the vertices. It turns out that the effort necessary to close an instance is also affected by the number and the structure of the precedence graph edges. When the graph P is very dense, i.e. a precedence relation is defined for each pair of vertices, it is trivial to find what is usually the only feasible solution. The same happens in the opposite situation: when the precedence graph does not contain any edges, the problem reduces to ATSP. The worst case verifies when the number of precedence is midway from the maximum and zero. This reasoning cannot be limited to the raw quantity of edges in P, it is in fact interesting to investigate peculiar (sub)structures that could lead to an increase or reduction of complexity. We have focused our interest on a particular structure that will be described in details in the reminder of the section. When such a condition is detected, the instance can be split in two independent problems. The strategy that we would like to present in this paper comprises two steps: in the first we artificially create the aforementioned condition and in the second we split the problem. The execution speed increases dramatically because solving the two subproblems requires less effort w.r.t. solving the originating one. These two steps can be applied recursively also to subproblems giving rise to a branch and bound algorithm. The exact procedure to build the search tree will be explained in the following sections.

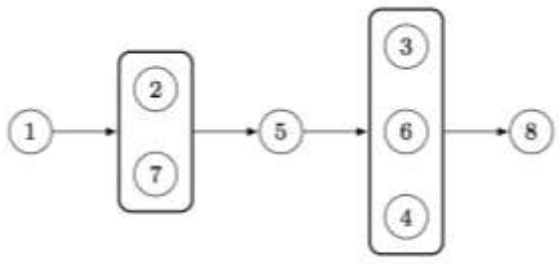
The underlying decomposition idea. We would like to address a decomposition technique that can be integrated both in the preprocessing phase and in the solver. The goal is to split the original problem in two or more parts, solve them separately and recombine the solutions. Ideally we would like to find k subsets of vertices such that a solution for the original instance is given by the concatenation of the solutions of the k subproblems. A decomposition is particularly interesting when the subproblems are of the same type of the parent, so we can exploit the same algorithms and moreover it can be applied recursively. This can be done for example when the precedence relation R contains a structure like the one in Figure 1(a) for the case with $k = 2$.

Vertex *5* is connected to all the other vertices, so we know in advance which of them will precede or follow it. Such a vertex will be referred to as *fixed vertex*. Following the schema in Figure 1(b), each feasible solution starts as usual with *1*, then we proceed with a permutation of the vertices *{2,7}*, going on we find the fixed vertex *5*, after that there will be a permutation of the vertices *{3, 6, 4}* and the sequence closes with vertex *8*, before going back to *1* (omitted in the figure).

In this case we can split the problem in two parts: the first composed of vertices *{1,2,7,5}* and the second of the vertices *{5,3,6,4,8}*, with vertex *5* playing the role of the starting and ending vertex in the second subproblem. All the precedences between vertices in the same subproblem are still valid and those between different subproblems are already enforced by the split. The idea will be integrated in a branch and bound approach where the aim will be to introduce artificial precedence constraints in such a way to create new fixed vertices in the SOP instance.



(a) Precedence relation involving the fixed node 5. The dashed arrows represent a subset of the precedence relations between the nodes.



(b) Structure of each feasible solution. The two subproblems are needed to find the optimal permutation of the nodes in the two boxes. Fixed node 5 becomes the end node and the start node of the first and the second subproblem, respectively.

Fig. 1. Fixed node decomposition

The algorithm. In the previous subsection we have described a profitable precedence structure that allows us to greatly reduce the solver effort. It can be applied as is in a preprocessing step, but it is often the case that these structures are incomplete, so it is impossible to decompose the problem without manipulating it. Every problem modification must be applied paying much attention at preserving the completeness of the search.

In the following description, we will continue to use the term *vertices* for the jobs to schedule and *nodes* for the subproblems that compose the B&B tree. Let us assume that in problem *P* there is a vertex *a* that has a precedence relation with almost all the other vertices, but not with vertex *b*: that prevent us from applying the decomposition seen in the previous subsection. What we could do to reach the structure in Figure 1(a) is to artificially add the precedence *a*→*b*. This is however not possible, because it would bias the search by removing a priori a set of solutions that could contain the optimal one. This issue is easily avoidable by creating two subproblems: *P*′ will be a clone of *P* augmented with *a*→*b* and *P*″ will be derived from *P* and completed with *b*→*a*. Since the two added precedences are symmetric, the two subproblems *P*′ and *P*″ form a partition of the search space of *P*. At this point we know two operations that can be applied to a problem *P* and that generate two subproblems *P*′ and *P*″: (i) *Split*. *P* is divided in two problems of smaller size. An optimal solution for *P* can be rebuild by concatenating those of *P*′ and *P*″. Can be applied only if a fixed vertex is present; (ii) *Add an artificial precedence*. In this case *P*′ and *P*″ are of the same size. To obtain the maximum effect, the precedence added must involve a vertex that is already related with a high number of vertices. The optimal solution for *P* is the best between that of *P*′ and *P*″. Using these two simple operations we can implement a powerful branch and bound algorithm. We start from creating the search tree root using the instance given by the user, then, at every iteration, we select a node: if it contains a fixed vertex, we apply a split, otherwise we add an artificial precedence.

Our main goal was to improve the lower bound on the known instances: for this reason we have chosen to always select the vertex that determined the lower bound. In a branch and bound algorithm every node represents a different partition of the same problem: our strategy works on different partitions of different subproblems. For this reason when we want to increase the lower bound we cannot directly compare the node lower bounds, otherwise we will bias the search towards the smaller problems. To select the candidate nodes to expand, we start from the root and every time we find a problem to which a precedence was added, we continue the search from the child with the lowest lower bound; when we encounter a problem that has been split, we proceed from all the children. The result is a set of nodes that are currently determining the lower bound. Among the candidates we choose the node that has the highest potential to improve the result, i.e. the nodes are ranked lexicographically based on two parameters: the maximum possible relative improvement they can give and their gap. Subproblems with at most 20 vertices are passed directly to the MILP solver we adopted (CPLEX 12.3), because our empirical studies revealed they can easily be solved to optimality in seconds. If the subproblem has more than 20 vertices, then we try to estimate a lower bound by solving the linear relaxation of the formulation introduced in Section 2 applied to the subproblem itself. The LP solver of CPLEX 12.3 is adopted in our implementation. If, after this computation, the expansion still looks attractive, we search for a fixed vertex: in the case of a positive result we immediately split the node and reconsider the whole tree, otherwise we must choose a precedence to add. The evaluation must combine different and sometime interfering objectives: the first is that one of the two vertices to connect must already have many precedence relations with the other vertices and the second is that we want to split the problem in the most balanced way (considering the number of vertices). Also in this case we have followed a hybrid approach: we firstly consider as candidates the vertices that can give a well-balanced split; among them we choose the one that is the most constrained.

Computational experiments

The experimental results concentrate on SOPLIB2006 instances since—as discussed in Section 1—they are promising for the approach we propose, because they are dominated by precedence constraints. SOPLIB2006 has been presented in [8], here we will see a brief description of the 48 instances of this dataset. The nomenclature follows the schema: *R.n.r.p.sop*. The meaning of the placeholders is the following: *n* is the instance size, i.e. the number of vertices $|V|$; *r* is the maximum cost of an edge; *p* is an approximation of the percentage of precedence constraints in the problem.

Over the 48 instances of the dataset, 16 had been previously solved to optimality. The branch and bound approach was able to solve to optimality these instances with an average computation time of 53.57 seconds.

The branch and bound approach proposed in this paper was extremely effective on the 32 open SOPLIB2006 instances. In particular, with a maximum computation time of 48 hours on a Dual AMD Opteron 250 2.4GHz/4GB RAM for each instance, it was able to close 8 new instances of the 32 that were still open. For these instances, 26 lower bounds (LBs) and 21 upper bounds (UBs) were improved with respect to the best known results in the literature [7], with an average improvement of 8.01% and 37.02%, respectively. The average optimality gap ((UB-LB)/UB) for the 48 instances of the dataset is now 9.27%, which indicates the research is not far from closing all the instances.

References

N. Ascheuer. Hamiltonian path problems in the on-line optimization of flexible manufacturing systems. Technical Report 3, Konrad-Zuse-Zentrum fur Informationstechnik Berlin, Heilbronner Str. 10, D-10711 Berlin - Wilmersdorf, February 1996.

N. Ascheuer, L. F. Escudero, M. Groetschel, and M. Stoer. A cutting plane approach to the sequential ordering problem (with applications to job scheduling in manufacturing). SIAM Journal on Optimization, 3(1):25–42, 1993.

L.F. Escudero. An inexact algorithm for the sequential ordering problem. European Journal of Operational Research, 37(2):236–249, 1988.

L.F. Escudero, M. Guignard, and K. Malik. A lagrangian relax-and-cut approach for the sequential ordering problem with precedence relationships. Annals of Operations Research, 50(1):219–237, 1994.

V. Maniezzo, T. Stuetzle, and S. Voß, editors. Hybridizing Metaheuristics and Mathematical Programming, volume 10. Springer, 2009.

M. Mojana. Matheuristic techniques for the sequential ordering problem. Master's thesis, Faculty of Informatics, Università della Svizzera italiana, 2011.

M. Mojana, R. Montemanni, G. Di Caro, and L.M. Gambardella. An algorithm combining linear program- ming and an ant system for the sequential ordering problem. In Proceedings of ATAI 2011 - The Second Annual International Conference on Advanced Topics in Artificial Intelligence, pages 80–85, 2011.

R. Montemanni, D.H. Smith, and L.M. Gambardella. A heuristic manipulation technique for the sequential ordering problem. Computers and Operations Research, 35(12):3931–3944, December 2008.

C. Moon, J. Kim, G. Choi, and Y. Seo. An efficient genetic algorithm for the traveling salesman problem with precedence constraints. European Journal of Operational Research, 140(3):606–617, 2002.

Q. Pulleyblank and M. Timlin. Precedence constrained routing and helicopter scheduling: Heuristic design. Technical Report RC17154 (#76032), IBM T.J. Watson Research Center, Yorktown Heights, New York 10598, USA, 1991.