Journal of Applied Operational Research (2013) **5**(1), 2–13

ISSN 1735-8523 (Print), ISSN 1927-0089 (Online)

A decomposition-based exact approach for the sequential ordering problem

Roberto Montemanni *, Marco Mojana, Gianni A. Di Caro and Luca Maria Gambardella

Dalle Molle Institute for Artificial Intelligence (IDSIA)
Università Della Svizzera Italiana (USI)
Lugano, Switzerland

Abstract. Many different real world 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 novel branch and bound method that manipulates and exploits the structure of the precedence constraints to decompose the problem in smaller suproblems is introduced. Computational experiments on benchmark instances commonly adopted in the literature are presented. Improvements to best-known results are found by the new approach.

Keywords: sequential ordering problem; branch-and-bound; integer linear programming; scheduling; traveling salesman problem

Received September 2012. Accepted January 2013

Introduction

The Sequential Ordering Problem (SOP) is a combinatorial optimization problem. Given a directed graph, a set of precedence constraints between pairs of vertices and a strarting vertex, it requires to find the minimum cost Hamiltonian tour. When a tour visits the vertices fulfilling 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, (Ascheuer 1996) focuses on removing the bottlenecks created by a naïve management of a stacker crane in a manufacturing systems. The abstract formulation naturally adapts also to many transportation problems, as in (Pulleyblank and Timlin 1991), 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 (Stahlbock and Voß 2008).

It is possible to derive some simple relations between the SOP and other combinatorial problems. For example, a SOP without precedences is equivalent to an Asymmetric Travelling Salesman Problem (ATSP) and when the

* *Correspondence*: Roberto Montemanni, IDSIA/SUPSI, Galleria 2, CH-6928 Manno, Switzerland. E-mail: roberto@idsia.ch

latter has symmetric costs we can refer to it as Travelling Salesman Problem (TSP). Looking at the reductions in the opposite way, we can say that if an algorithm is able to solve any SOP, then it will be able to tackle any ATSP or TSP instance. These similarities allow us to state that the SOP must be at least as hard as the TSP: in fact, both are in the NP-hard class. The problem has been initially formulated in (Escudero 1988) as the underlying model for a production planning system. The algorithm proposed expolits promerties of the ATSP polytope, and the mathematical model is reinforced with inequalities able to exploit the presence of a partial order to provide tighter bounds and to remove infeasible sequences. The same direction is taken by (Ascheuer *et al*. 1993), 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 (Escudero *et al*. 1994). The work presented in (Ascheuer 1996) 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 later work, presented in (Hernadvolgyi 2003) and (Hernadvolgyi 2004) aims at generating good lower bounds by means of a technique, called "State Space Lattice Abstraction", that reduces the instance size. From a more abstract point of view, all these publications deal with the definition of the SOP polytope facets. A more in depth analysis of the influence of the precedences on the ATSP convex hull is reported in (Balas *et al*. 1995). Also the metaheuristic field has followed a similar path: the existing ATSP algorithms have been adapted to enforce the requested vertex order. A naïve approach was used in (Ascheuer 1996): a solution for the relaxed ATSP instance was computed and checked: all the unfeasible ones were simply rejected. In the area of genetic algorithms, (Chen and Smith 1996) and (Moon *et al*. 2002) introduced a series of crossover operators that preserve the tour feasibility. In (Pulleyblank and Timlin 1991) improvements have been recorded thanks to the use of the Voronoi quantized crossover that adopts complete graph representation. The opportunity given by the spread of parallel execution environments has been taken by [9], that describes a multithread implementation of a rollout algorithm. The ant-based metaheuristics have been modified such that every agent is aware of the moves forbidden by the precedences: it is the case of ACS-SOP. In (Gambardella and Dorigo 2000) the authors described HAS-SOP: an algorithm that couples ACS-SOP with a 3-opt local search. The LS has been further optimized by means of a stack that keeps track of the most promising regions to rescan. The effectiveness of this algorithm is affected by the density of the precedences: (Montemanni *et al*. 2007) and (Montemanni *et al*. 2008a) has shown how to exploit this property by artificially enforcing those precedences that are more likely to be fulfilled by the optimal solution. The effectiveness of the proposed local searches, the ant colony matheuristics and the heuristic manipulation technique has been evaluated in (Montemanni *et al*. 2008b) and (Montemanni *et al*. 2009) taking the quay crane assignment problem as a benchmark. Particle Swarm Optimization (PSO) has also been applied with success to the problem. At first glance it seems unnatural to code the solution of a combinatorial optimization problem by means of position and speed vectors: an intermediate step was necessary to find an abstraction of these concepts, that are normally related to the continuous world, that has permitted to encode and operate on permutations. This gave rise to the hybrid discrete PSO described in (Anghinolfi *et al*. 2009) and (Anghinolfi *et al*. 2011). A matheuristic approach (Maniezzo *et al*. 2009) combining a Mixed Integer Linear Programming solver and an Ant System in a simple but effective way has been described in (Mojana *et al*. 2011) and (Mojana 2011).

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 subproblems. Such an approach is tailored for instances dominated by precedences, like those of the SOPLIB2006 dataset (Montemanni *et al*. 2008a). The innovation of the method resides in the idea that adding artificial precedence constraints can lead to some very favorable structures of the precedence graph, that can in turn lead to a decomposition of the original problem. A preliminary version of this work has appeared in (Mojana *et al*. 2012).

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 $A = \{(i,j) | i,j \in V\}$ is the set of arcs. A cost $cij \in N_0^+$ 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. vertex i has to precede vertex 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 problem can alternatively be represented as a scheduling problem with precedences, sometimes referred to as an *open* ATSP with precedences. It is sufficient to duplicate the starting vertex *1* into a ending vertex $|V|+1$ (with a precedence constraints between each vertex and the ending vertex) and impose that each feasible solution (that is not a tour anymore) starts in *1* and ends in $|V|+1$ after having visited all the vertices, while fulfilling precedence constraints. Such a structure can be sometimes useful to make reasoning simpler (see Section 4.1).

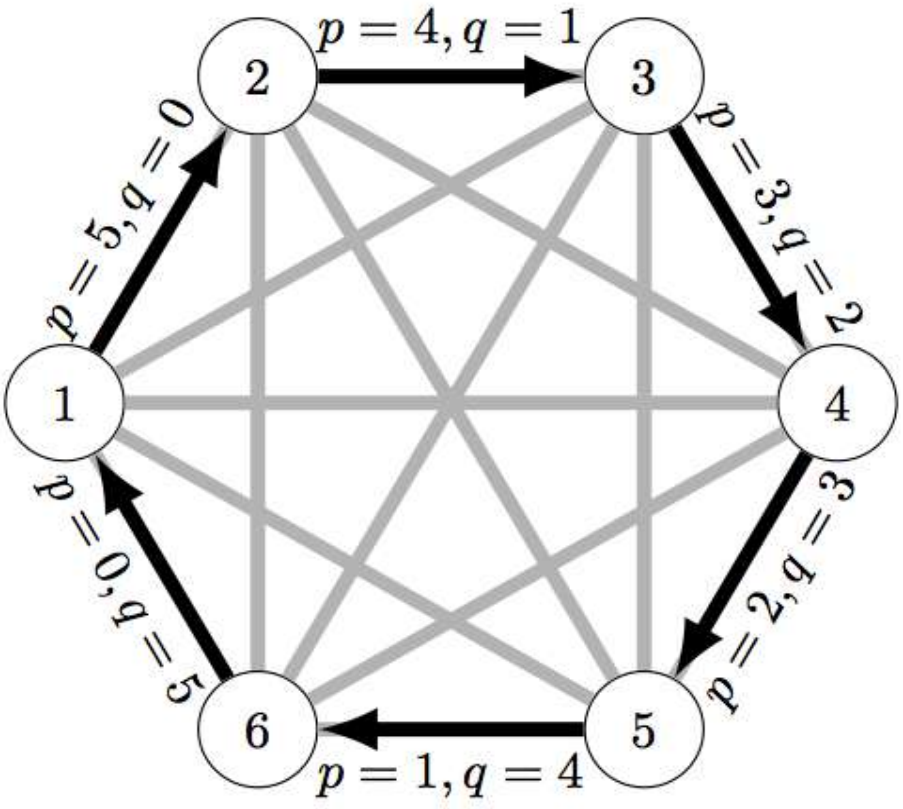


Fig. 1. Example of integer feasible solution for the flow model.

A mixed integer linear programming formulation

The mathematical model we adopt is the two-commodity network flow formulation discussed in (Moon *et al.* 2002), where however no experimental study was presented. To have a better understanding of the meaning of the formulation, we will briefly describe the underlying scenario, taking Figure 1 as a reference. 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. The resulting mathematical formulation is the following one:

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

s.t.

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

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

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

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

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

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

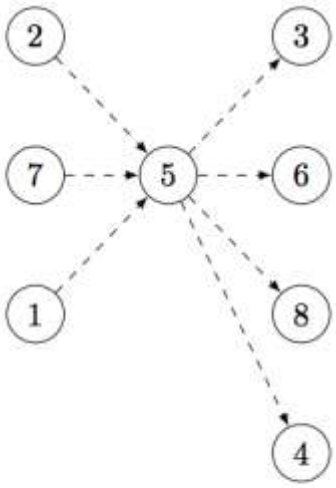
$$y_{ij}^{q} \geq 0 \quad \forall i,j \in V \tag{8}$$

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

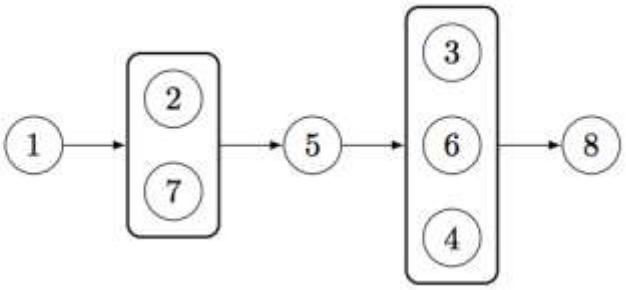
Variables y^{p}_{ij} (y^{q}_{ij}) correspond to the quantity of commodity p (q) carried from i to j. The binary variables y_{ij}, together with constraint (5), prevent the vehicle from "splitting" and following two different paths. The constraints (2) and (3) represent the flow balance equations. Every customer produces one unit of commodity q and consumes one unit of commodity p, so during the tour the vehicle load is always $|V|-1$ (imposed by 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.

A branch-and-bound algorithm

The SOP shares with all the other NP-hard problems the characteristic that the time for solving an instance grows exponentially with the size of the instance (intended as number of vertices, i.e. $|V|$). It is also easy to experimentally see that the effort necessary to close an instance is heavily 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 almost every pair of vertices, it is trivial to find what is sometimes the only feasible solution. The same happens in the opposite situation: when the precedence graph does not contain any edge, the problem reduces to ATSP. The worst case is 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 be exploited by a solving procedure. We have focused our interest on a particular structure that will be described in details in Section 4.1. When such a structure 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 small 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 framework. The exact procedure to build the search tree will be explained in the following sections. In the description we will use the name node for a search-tree node of the branch and bound tree, while we will stick to the name vertex while referring to an element of set V.



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



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

Fig. 2. Fixed vertex decomposition.

The decomposition idea

We present a decomposition technique that can be integrated in a MILP 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. Such 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 they can be applied recursively. This can be done for example when the precedence relation R contains a structure like the one in Figure 2(a) for the case with $k = 2$. Notice that in the figure the open ATSP representation of the problem introduced near the end of Section 2 is used for simplifying the reasoning. 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 2(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 the ending vertex *8*. 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 ending vertex in the first subproblem and of the starting 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. We integrate the discussed idea in a branch-and-bound approach with the aim of introducing artificial precedence constraints in such a way to create new fixed vertices in the SOP instance.

The algorithm

In section 4.1 we have described a precedence structure that can allow us to greatly reduce 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.

Let us assume that in problem *Pr* there is a vertex i that has a precedence relation with almost all the other vertices, but not with vertex j. This would prevent us from applying the decomposition seen in Section 4.1. What we could do to reach the structure in Figure 2(a) is to artificially add the precedence $i \rightarrow j$. 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: *Pr'*, which is a clone of *Pr* augmented with $i \rightarrow j$, and *Pr"*, which is derived from *Pr* and completed with $j \rightarrow i$. Since the two added precedences are symmetric, the two subproblems *Pr'* and *Pr"* form a partition of the search space of *Pr*. At this point we know two operations that can be applied to a problem *Pr* and that generate two subproblems *Pr'* and *Pr"*:

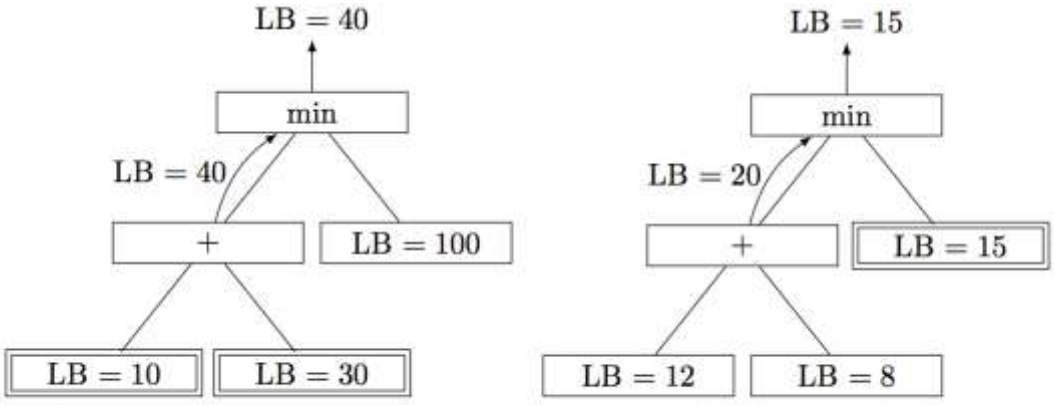
- **Split:** *Pr* is divided in two problems of smaller size. An optimal solution for *Pr* can be rebuild by concatenating those of *Pr'* and *Pr"*. It can be applied only if a fixed vertex is present.
- **Add an artificial precedence:** in this case *Pr'* and *Pr"* are of the same size. To obtain the maximum effect, the precedence added must involve a vertex that is already related with a high number of vertices. The optimal solution for *Pr* is the best between that of *Pr'* and *Pr"*.

Using these two simple operations we can implement a 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. In the next subsections we will describe the strategies we have chosen to identify the nodes to expand at each iteration and to select the branching to apply on the identified node.

Selection of the search tree node to expand

Assuming a minimization problem, in a traditional branch and bound the global lower bound is typically given by the open node with the smallest linear relaxation cost. This suggests that the only way to improve the lower bound is to work on this node. In our case, the situation is slightly different: in the same tree we must deal with many different subproblems and it makes no sense to compare their solutions. In order to compare the bounds, we have to keep track of the applied decompositions to correctly recombine the subproblems. Once the nodes to compare have been generated by a classic branching operation (adding an artificial precedence), all the classical branch and bound relations hold. Since the beginning of the design phase, we have understood that this requirement was of utmost importance.

The goal of the search strategy is to choose in which direction to continue the search. Concretely, given a tree, we have to understand which nodes need our attention, i.e. which ones are currently determining the global lower bound. In Figure 3 we can see an example of how the global LB is computed. The root children are created by adding a precedence and its reverse to the original problem: in this case the two LB are combined with the *min* function because they refer to the same subproblem. On the other hand, the node on the left at the second level is split: the operator to put together the children LBs is the sum. The two parts of the figure illustrate some numerical examples of what we have explained before, i.e. the differences between a normal branch and bound and one that supports problem decompositions. Following the reasoning used to design the tool, we want to find the leaves that form the global LB and that are worth to be *searched*. A random leaf among the selected one is expanded by the algorithm in our implementation.



(a) Example of tree where the global LB is influenced by more than one leaf

(b) Example of tree where the leaf with minimum LB does not determine the global LB

Fig. 3. LB propagation in the tree. The inner nodes contain the function to combine the LB of the children. The leaves that determine the global LB are highlighted by a double border.

Branching strategy

The role of the branching strategy is to choose how to attempt to improve the bound of a given search tree node. To implement this, we have to consider multiple objectives:

- Complete the search as fast as possible;
- The search could take a considerable amount of time: it would be good to provide good lower bounds already during the search. This helps also in focusing only on the most promising part of the tree;
- Do not put too much effort on easy subproblems.
- Putting all these consideration together, we have implemented the Algorithm 1.

Algorithm 1 Branch strategy pseudocode

```
if Problem is easy then
    Solve to optimality with the MILP solver
else if Problem relaxation not available then
    Solve the relaxation
else if Problem contains some fixed vertices then
    Decompose the problem
else
    Create a new branch by adding a precedence
end if
```

At the beginning the problem is inspected to judge what is the probability that it can be solved in a short time. A simple criterion (currently adopted in our implementation) is to check if the problem contains less than μ vertices (in out tests we used $\mu = 20$). If the problem is considered too difficult to solve at once, we would like to prove that the expansion is not necessary. To avoid this expensive operation we solve the network formulation relaxation. In this way, reasoning on the bounds, we can possibly delay any further operation on the node, until it becomes an obstacle to the global lower bound improvement.

If a relaxation solution is already available, it means that the selected node must be improved. Creating a branch is expensive, because the number of open nodes increases and the two new leaves have a difficulty level comparable to that of the parent. So, before doing that, it is the right moment to analyze the precedence structure to detect fixed vertices.

Given that all the previous tests failed, we are forced to create a branch. To do that we have to choose which precedence to add. There is no exact way to follow, but some choices are smarter than others. Once again we have to fight against a multi-objective problem:

- For every vertex, we know the range of its feasible positions inside the optimal sequence. The effort to fix a vertex position is exponential in the size of the range, so we should try to keep it as small as possible;
- We have also to consider the centrality of the range: if it is near the ends of the sequence, it means that, when the vertex has been fixed, we will split the problem in a part that will have a length similar to the one of the complete problem and another part containing just a couple of vertices. In that case, the small problem will be solved almost instantaneously, but the other will require an amount of work comparable to that of solving the original instance. Considering that the time required to solve a problem is exponential in its size, the total time to solve the two sub-problems, given the fixed vertex position p, is:

$$t^{dim} = t^{p} + t^{\mathtt{dim}-p} \tag{10}$$

that — due to the exponential nature — has a minimum for $p = (dim - 1)/2$, i.e. when the position of the fixed vertex is in the center of the sequence.

Taking into account both requirements is not always easy: it happens that there are vertices with a small number of feasible solutions, but very close to the ends. Before going into the details, we need to summarize some definitions:

The **predecessors** of a vertex are the vertices that must precede it:

$$\mathrm{pred}(i) = \{j \in V \,|\, (j,i) \in R\} \tag{11}$$

The **successors** of a vertex are the vertices that must follow it:

$$\mathrm{succ}(i) = \{j \in V | \, (i,j) \in R\} \tag{12}$$

The **free vertices** of a vertex are the vertices that do not have a precedence relation with it:

$$\mathrm{free}(i) = V \setminus \{i\} \setminus \mathrm{pred}(i) \setminus \mathrm{succ}(i) \tag{13}$$

The **centrality** of a vertex is:

$$\mathrm{centr}(i) = \min\left(|\mathrm{pred}\,(i)\,|, |\mathrm{succ}\,(i)\,|\right) \tag{14}$$

Let us see how our implementation chooses the best vertex. We consider only the vertices that reside in the central part of the sequence, i.e. those that have a centrality at least equal to:

$$\mathrm{min}_{\mathrm{centr}} = \min\left(\frac{1-w}{2} \cdot \mathrm{dim}, \max_{i \in V} \mathrm{centr}\,(i)\right) \tag{15}$$

The meaning of the formula (15) is easier to understand with an example. Let *w = 0.3*: we start by considering the vertices that occupy the central 30% of the sequence; if none of them satisfies the requirement, we enlarge the window until the most central vertices fit. When we have built the set of candidates, we select the one that minimizes *free(n)*. The rationale behind this procedure is that when the number of free vertices increases by one, the leaves doubles, instead, when the vertex is not exactly in the center the penalty is much smaller.

Now that we know which vertex we want to fix, we have to choose another vertex, among *free(·)*, to finally build the precedence. It is worth mentioning that there is no need to choose the "direction" of the precedence: the branch is created exactly by considering opposite directions. To choose the free vertex we will rely on the heuristic estimates provided by the flow intensities *y* in the solution of the linear relaxation of the MILP associated with the problem under investigation. We know that the new lower bound of the node will be equal to the minimum between that of the children, so ideally, we want that the relaxation solution value of both children rises. This translates in finding a precedence *(u,v)* that is violated by the current relaxation, even if reversed, i.e. such that:

$$\left| \sum_{j=1}^{|V|} \overline{y}_{uj}^{p} - \sum_{j=1}^{|V|} \overline{y}_{vj}^{p} \right| < 1$$

Our algorithm chooses the free vertex that minimizes the left hand side of the inequality. In our experiments we have used *w = 0.3* (that was suggested by the analysis of preliminary tests).

Computational experiments

The experimental results concentrate on SOPLIB2006 instances: they look promising for the approach we propose since they are dominated by precedence constraints (see Section 1). The MILP solver adopted in the experiments is IBM ILOG CPLEX 12.3 (http://www.cplex.com). After having introduced the instances, we investigate the effectiveness of the branch and bound frame- work we propose in terms of lower bounds progression and finally we present the detailed experimental results of the method.

Benchmark instances

SOPLIB2006 has been presented in (Montemanni *et al*. 2008a), where the reader can find a detailed description of the 48 instances of this datasat. The characteristics of each instance are encoded in its (file) name, according to the schema *R.n.r.p.sop*, where the meaning of the placeholders is the following one:

- **n**: The instance size, i.e. the number of vertices |V|. The library defines problems with n ∈ {200; 300; 400; 500; 600; 700}.
- **r**: The maximum cost of an edge: $0 \leq c_{ij} \leq r \; \forall i, j \in V$. All the costs are integers produced using a uniform pseudorandom number generator. The library defines problems with r ∈ {100; 1000}.
- **p**: The approximate precedence ratio. The number of random active precedences (not closed by transitivity) is roughly proportional to: 2·p/(100·n(n-1)). The library defines problems with p ∈ {1; 15; 30; 60}.

Lower bounds progression

Implementing such an elaborate framework like the one described in Section 4 on top of a solver makes sense only if there is a clear advantage over the solver itself. We have therefore compared the lower bounds progression of the CPLEX MILP solver with that of the branch and bound framework we propose run on top of CPLEX on a same instance. The instance that has been selected for the test is *R.200.100.30*: it is a small instance (this makes computations easier) but at the same time is a good representative of the whole dataset. The comparison has been executed on a MacBook Pro equipped with an Intel Core 2 Duo, that runs at 2.53 GHz, and 4GB of RAM. Figure 4 shows the variation of the lower bound in a run of 12 hours.

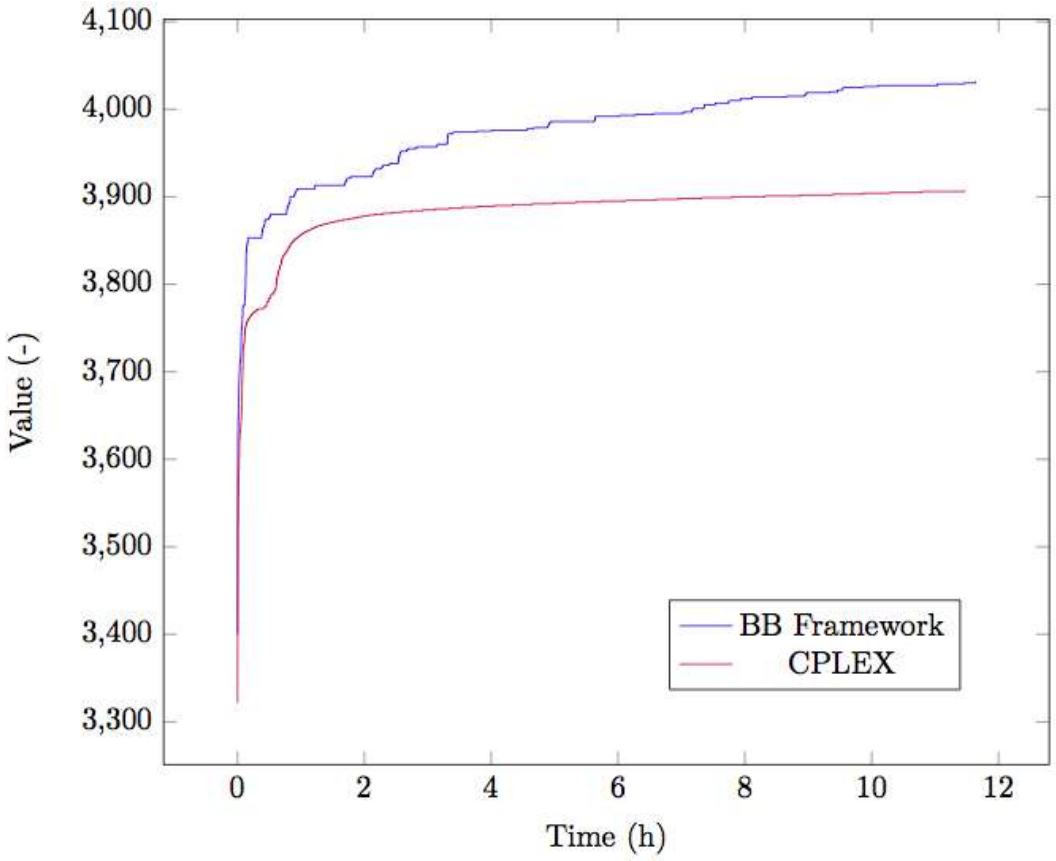


Fig. 4. Lower bound trend comparison against CPLEX.

During the first minutes, the two curves are superimposed: the reason is that initially the two algorithms have to solve the relaxation of the root and execute all the separation routines. After that point, the two plots separate. CPLEX goes on branching with its custom logic, that has two problems: the first is that at the beginning it succeeds to improve the bound but only very slowly, and the second is the apparent presence of a plateau. The cause can be found in the fact that branching decisions do not allow any simplification of the problem. Moreover we know that to improve the lower bound the most effective search strategy is "best bound", i.e. we choose every time one of the best nodes. This usually leads to a huge number of open nodes that are an obstacle to the improvements because, in general, to raise the global lower bound of some units, we have to raise the lower bound of all the open nodes by the same amount.

What distinguishes our framework is that the more we go deeper in the search tree, the more the nodes will represent smaller problems. This decompositions compensate the explosion of the number of nodes. This explains the regular progression shown in the plot.

Experimental results

Over the 48 instances of the dataset, 16 have been previously solved to optimality. The results for the remaining 32 instances, for which optimality has never been proven before are summarized in the first block of rows of Table 1.

Table 1: Results on the open SOPLIB2006 instances.

Instances	Best known			Branch and bound			
	LB	UB	Gap %	LB (Impr %)	UB (Impr %)	Gap %	Time (s)
R.200.100.15	1091	1792	39.12	1257(13.21)	1792 (0.00)	29.85	-
R.200.100.30	3693	4216	12.41	4185(11.76)	4216 (0.00)	0.74	-
R.200.1000.15	13364	20481	34.75	14565 (8.25)	20481 (0.00)	28.89	-
R.200.1000.30	37103	41196	9.94	40170 (7.64)	41196 (0.00)	2.49	-
R.300.100.15	1839	3162	41.84	2166(15.10)	3161 (0.03)	31.48	-
R.300.100.30	5259	6120	14.07	5839 (9.93)	6120 (0.00)	4.59	-
R.300.1000.1	1292	1331	2.93	1294 (0.15)	1294 (2.86)	**0.00**	19864.0
R.300.1000.15	19265	29248	34.13	21096 (8.68)	29183 (0.22)	27.71	-
R.300.1000.30	46665	54147	13.82	51495 (9.38)	54147 (0.00)	4.90	-
R.400.100.15	2494	3925	36.46	2747 (9.21)	3906 (0.49)	29.67	-
R.400.100.30	7117	8165	12.84	7755 (8.23)	8165 (0.00)	5.02	-
R.400.1000.1	1342	1456	7.83	1343 (0.07)	1343 (8.41)	**0.00**	3004.3
R.400.1000.15	25518	39612	35.58	28159 (9.38)	29685 (33.44)	5.14	-
R.400.1000.30	7228 0	85192	15.16	79868 (9.50)	85132 (0.07)	6.18	-
R.500.100.1	4	11	63.64	4 (0.00)	4(175.00)	**0.00**	9760.0
R.500.100.15	3111	5431	42.72	3543(12.19)	5361 (1.31)	33.91	-
R.500.100.30	7792	9665	19.38	8600 (9.40)	9665 (0.00)	11.02	-
R.500.1000.1	1315	1501	12.39	1316 (0.08)	1316 (14.06)	**0.00**	9383.3
R.500.1000.15	29304	51091	42.64	32950(11.07)	50725 (0.72)	35.04	-
R.500.1000.30	83568	99018	15.60	91272 (8.44)	98987 (0.03)	7.79	-
R.600.100.1	1	6	83.33	1 (0.00)	1(500.00)	**0.00**	6652.1
R.600.100.15	3071	5798	47.03	3656(16.00)	5684 (2.01)	35.68	-
R.600.100.30	10400	12465	16.57	11841(12.17)	12465 (0.00)	5.01	-
R.600.1000.1	1337	1534	12.84	1337 (0.00)	1337 (14.73)	**0.00**	23004.9
R.600.1000.15	32411	57812	43.94	36546(11.31)	57237 (1.00)	36.15	-
R.600.1000.30	105986	126789	16.41	116037 (8.66)	126789 (0.00)	8.48	-
R.700.100.1	1	5	80.00	1 (0.00)	1(400.00)	**0.00**	13781.6
R.700.100.15	3872	7380	47.53	4494(13.84)	7311 (0.94)	38.53	-
R.700.100.30	12237	14513	15.68	13663(10.44)	14510 (0.02)	5.84	-
R.700.1000.1	1229	1579	22.17	1231 (0.16)	1231 (28.27)	**0.00**	56712.0
R.700.1000.15	35270	67510	47.76	40662(13.26)	66837 (1.01)	39.16	-
R.700.1000.30	108184	134474	19.55	118718 (8.87)	134474 (0.00)	11.72	-
R.200.100.1	61	61	0.00	61 (0.00)	61 (0.00)	0.00	425.81
R.200.100.60	71749	71749	0.00	71749 (0.00)	71749 (0.00)	0.00	0.30
R.200.1000.1	1404	1404	0.00	1404 (0.00)	1404 (0.00)	0.00	169.32
R.200.1000.60	71556	71556	0.00	71556 (0.00)	71556 (0.00)	0.00	1.90
R.300.100.1	26	26	0.00	26 (0.00)	26 (0.00)	0.00	2239.56
R.300.100.60	9726	9726	0.00	9726 (0.00)	9726 (0.00)	0.00	0.58
R.300.1000.60	109471	109471	0.00	109471 (0.00)	109471 (0.00)	0.00	2.14
R.400.100.1	13	13	0.00	13 (0.00)	13 (0.00)	0.00	4821.69
R.400.100.60	15228	15228	0.00	15228 (0.00)	15228 (0.00)	0.00	49.25
R.400.1000.60	140816	140816	0.00	140816 (0.00)	140816 (0.00)	0.00	41.85
R.500.100.60	18240	18240	0.00	18240 (0.00)	18240 (0.00)	0.00	10.80
R.500.1000.60	178212	178212	0.00	178212 (0.00)	178212 (0.00)	0.00	25.58
R.600.100.60	23293	23293	0.00	23293 (0.00)	23293 (0.00)	0.00	8.44
R.600.1000.60	214608	214608	0.00	214608 (0.00)	214608 (0.00)	0.00	8.67
R.700.100.60	24102	24102	0.00	24102 (0.00)	24102 (0.00)	0.00	45.89
R.700.1000.60	245589	245589	0.00	245589 (0.00)	245589 (0.00)	0.00	75.14

For each of the instances we report the results obtained by our new branch and bound approach together with the best known lower and upper bounds currently available in the literature and the corresponding optimality gap, calculated as *(UB-LB)/UB*. More specifically, we report the lower and upper bounds obtained together with the optimality gap and the computation time (in seconds) required to close the instances. A maximum time limit of 48 hours is imposed on a computer equipped with a dual AMD Opteron 250 2.4GHz processor and 4GB of RAM. The improvements of the new lower and upper bounds over the previously best known values, calculated as *(LB – BestknownLB) / max{LB; BestknownLB}* for the lower bounds and *(BestknownLB - UB) / min{UB; BestknownUB}* for the upper bounds, where *BestknownXB* refers to the results available before the present study, and are enclosed in brackets. In the second block of rows of Table 1 we report the results for the instances for which optimality had been proven before. In this case, the main data of interest is the time required by the new branch and bound method to close them (last column).

The results in the first part of Table 1 indicate that the branch and bound approach proposed in this paper is extremely effective on the open SOPLIB2006 instances. In particular, it was able to close 8 new instances of the 32 which were still open. For these instances, 26 lower bounds and 21 upper bounds have been improved with respect to the best known results, with an average improvement of 8.0% and 37.0% on average. The average optimality gap for the 48 instances of the dataset is now 9.3%, which indicates the research is not far from closing all the instances. When analyzing the second part of Table 1 we can observe how the new branch and bound method is able to close the 16 instances for which optimality had been proven before in reasonable computation times (very short for most of the cases), proving to be an efficient method.

Conclusions

An exact method based on a decomposition approach and leading to a branch and bound framework has been proposed for the Sequential Ordering Problem. The new method has been specifically designed for instances with a substantial number of precedence constraints. Experimental results have shown the effectiveness of the new method on the target instances.

References

Anghinolfi D, Montemanni R, Paolucci M and Gambardella LM (2009). A particle swarm optimization approach for the sequential ordering problem. In Proceedings of the VIII Metaheuristic International Conference (MIC 2009), Hamburg, Germany.

Anghinolfi D, Montemanni R, Paolucci M and Gambardella LM (2011). A hybrid particle swarm optimization approach for the sequential ordering problem. Computers & Operations Research, 38(7):1076–1085.

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

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

Balas E, Fischetti M and Pulleyblank WR (1995). The precedence-constrained asymmetric traveling salesman polytope. Mathematical Programming, 68(1):241–265.

Chen S and Smith S (1996). Commonality and genetic algorithms. Technical Report CMU-RI- TR-96-27, Robotics Institute, Carnegie Mellon University, Pittsburgh, USA.

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

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

Gambardella LM and Dorigo M (2000). An ant colony system hybridized with a new lo- cal search for the sequential ordering problem. INFORMS Journal on Computing, 12(3):237–255.

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

Hernadvolgyi IT (2003). Solving the sequential ordering problem with automatically generated lower bounds. In Proceedings of Operations Research 2003, pages 355–362. Heidelberg, Springer Verlag.

Hernadvolgyi IT (2004). Automatically generated lower bounds for search. PhD thesis, School of Information Technology and Engineering, University of Ottawa, Ontario, Canada.

Maniezzo V, Stuetzle T and Voß S, editors (2009). Hybridizing Metaheuristics and Mathematical Programming. Heidelberg, Springer.

Mojana M (2011). Matheuristic techniques for the sequential ordering problem. Master thesis, Faculty of Informatics, Università della Svizzera italiana, Lugano, Switzerland.

Mojana M, Montemanni R, Di Caro GA and Gambardella LM (2011). An algorithm combining linear programming 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, Singapore.

Mojana M, Montemanni R, Di Caro GA and Gambardella LM (2012). A branch and bound approach for the sequential oridering problem. In P. Luangpaiboon *et al*. eds., editor, Proceedings of ICAOR 2012 – The 4th International Conference on Applied Operational Research, Lecture Notes in Management Science 4, pages 266–273.

Montemanni R, Smith DH and Gambardella LM (2007). Ant colony systems for large se- quential ordering problems. In Proceedings of the IEEE Swarm Intelligence Symposium, pp 60–67, Honolulu, USA.

Montemanni R, Smith DH and Gambardella LM (2008a). A heuristic manipulation tech- nique for the sequential ordering problem. Computers and Operations Research, 35(12):3931–3944.

Montemanni R, Smith DH, Rizzoli AE and Gambardella LM (2008b). Sequential ordering problems for crane scheduling in port terminals. In Bruzzone *et al*., editors, Proceedings of HMS 2008 - The 11th Intermodal Workshop on Harbor, Maritime and Multimodal Logistic Modeling and Simulation, pages 180–189, Campora San Giovanni, Italy.

Montemanni R, Smith DH, Rizzoli AE and Gambardella LM (2009). Sequential ordering problems for crane scheduling in port terminals. International Journal of Simulation and Process Modelling, 5(4):348–361.

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

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

Seo D-I and Moon B-R (2003). A hybrid genetic algorithm based on complete graph repre- sentation for the sequential ordering problem. In Proceedings of the international conference on Genetic and evolutionary computation: Part I, GECCO'03, pp 669–680, Heidelberg, Springer-Verlag.

Stahlbock R and Voß S (2008). Operations research at container terminals: a literature update. OR Spectrum, 30(1):1–52.