Tabu Search for the Linear Ordering Problem with Cumulative Costs

ABRAHAM DUARTE
Departamento de Computación, Universidad Rey Juan Carlos, Spain.
Abraham.Duarte@urjc.es

MANUEL LAGUNA
Leeds School of Business, University of Colorado at Boulder, USA
laguna@colorado.edu

RAFAEL MARTÍ
Departamento de Estadística e Investigación Operativa, Universidad de Valencia, Spain
Rafael.Marti@uv.es

Abstract

Given a matrix of weights, the Linear Ordering Problem (LOP) consists of finding a permutation of the columns and rows in order to maximize the sum of the weights in the upper triangle. This well known NP-complete problem can also be formulated on a complete weighted graph, where the objective is to find an acyclic tournament that maximizes the sum of arc weights. The variant of the LOP that we target here was recently introduced and adds a cumulative non-linear propagation of the costs to the sum of the arc weights. We first review the previous methods for the LOP and for this variant with cumulative costs (LOPCC) and then propose a heuristic algorithm for the LOPCC, which is based on the Tabu Search (TS) methodology. Our method achieves search intensification and diversification through the implementation of both short and long term memory structures. Our extensive experimentation with 224 instances shows that the proposed procedure outperforms existing methods in terms of solution quality and has reasonable computing-time requirements.

Keywords: Combinatorial Optimization, Metaheuristics, Linear Ordering Problem

Original version: February 25, 2008
Revised version: May 19, 2009

1. Introduction

Given a matrix of weights $C = \{c_{ij}\}_{n\times n}$, the LOP consists of maximizing the expression:

$$C_{LOP}(p) = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} c_{p_i p_j} .$$

where p_i is the index of the column (and row) in position i in the permutation. In the LOP, the permutation p provides the ordering of both the columns and the rows. The equivalent problem in graphs is that of finding, in a complete weighted graph, an acyclic tournament with a maximal sum of arc weights (Reinelt, 1985).

Bertacco, et al. (2008) introduced a variant of this problem referred to as the *Linear Ordering Problem with Cumulative Costs* (LOPCC). Given a complete digraph with nonnegative node weights d_i and nonnegative arc costs c_{ij}, the objective of the LOPCC is to find a Hamiltonian path $p = (p_1, p_2, .., p_n)$ and the corresponding node values α_i that minimize the expression:

$$C_{LOPCC}(p) = \sum_{i=1}^{n} \alpha_i$$

where

$$\alpha_{p_i} = d_{p_i} + \sum_{j=i+1}^{n} c_{p_i p_j} \alpha_{p_j} \qquad \text{for } i = n, n\text{-}1, \ldots, 1.$$

The cumulative backward computation of the α-values, from n to 1, makes the objective function nonlinear. The bounded version of this problem, in which $\alpha_i \leq U$ for all i, is referred to as *Bounded Linear Ordering Problem with Cumulative Costs* (BLOPCC).

The optimization of the universal mobile telecommunication standard (UMTS) in mobile-phone telecommunication systems is an application of the LOPCC (Proakis, 2004). In this context, mobile terminals (MTs) communicate simultaneously with a common base station. In order to distinguish among the signals of different MTs, UMTS adopts the so-called code division multiple access technique, where each terminal is identified by a specific code. In real situations, the MTs partially interfere with each other due to distortions introduced by radio propagation. The successive interference cancellation (SIC) is a very effective technique for interference reduction. SIC sequentially detects MTs following a predetermined order and removes the associated interference, improving the detection capability for the next users. If we define α_i as the power level at which user i must transmit his/her data and c_{ij} as the power of interference generated from user i to user j, the optimal solution to the corresponding LOPCC minimizes the overall transmission power (i.e. maximizes the duration of the MT batteries) while ensuring a proper reception for all users.

As stated in Bertacco, et al. (2008), a practical problem that arises in telecommunications and motivates the LOPCC is the joint power-control and receiver optimization (JOPCO). Given a set of users $\{1, 2, ..., n\}$, the interference factors c_{ij} for each pair of users (i, j), the noise power N_0, the spreading factor N_S, and the target ratio T, the problem consists of simultaneously optimizing the SIC detection order and the transmission power levels α_i, considering that a proper reception is ensured when the average signal-to-noise plus interference is equal to the target ratio *T*. A solution to the JOPCO problem is a permutation $p = (p_1, p_2, .., p_n)$ specifying the order in which users $\{1, 2, ..., n\}$ are examined to determine their transmission power levels α_i and cancel the associated interference. In mathematical terms:

$$Min \sum_{i=1}^{n} \alpha_{p_i}$$

$$\text{subject to:} \qquad T = \frac{c_{p_i p_i}\alpha_{p_i}}{N_0\sqrt{c_{p_i p_i}} + \sum_{l \in U_i} \alpha_l N_S c_{l p_i}} \quad \text{for } i = 1, \dots, n$$

$$\text{where:} \qquad U_i = \{p_{i+1}, p_{i+2}, \dots, p_n\} \text{ for } i = 1, \dots, n-1$$

Figure 1 shows an example of a *C* matrix in which we have drawn a triangle above the main diagonal to single out the coefficients that are added to calculate the objective function value of the associated LOP.

$$C = \begin{pmatrix} 0 & 2 & 1 & 3 \\ 4 & 0 & 1 & 5 \\ 2 & 3 & 0 & 5 \\ 1 & 2 & 1 & 0 \end{pmatrix}$$

Figure 1. Coefficient matrix

Considering the coefficient matrix in Figure 1, the objective function value of a solution $p = (1,2,3,4)$ in the LOP is calculated as follows:

$$C_{LOP}(p) = c_{12} + c_{13} + c_{14} + c_{23} + c_{24} + c_{34} = 2+1+3+1+5+5 = 17$$

Figure 2 shows the associated acyclic tournament when formulating the problem in a graph.

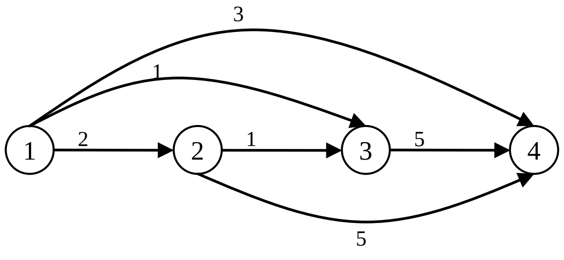

Figure 2. Acyclic tournament

Consider now the instance of the LOPCC given by the coefficient matrix in Figure 1 and the node weights $d = (3,2,4,1)$. In this case, the objective function value of solution $p = (1,2,3,4)$ is computed as:

$$C_{LOPCC}(p) = \alpha_1 + \alpha_2 + \alpha_3 + \alpha_4$$

where

$$\alpha_4 = d_4 = 1$$
$$\alpha_3 = d_3 + c_{34}\alpha_4 = 4 + 5*1 = 9$$
$$\alpha_2 = d_2 + c_{23}\alpha_3 + c_{24}\alpha_4 = 2 + 1*9 + 5*1 = 16$$
$$\alpha_1 = d_1 + c_{12}\alpha_2 + c_{13}\alpha_3 + c_{14}\alpha_4 = 3 + 2*16 + 1*9 + 3*1 = 47$$

and

$$C_{LOPCC}(p) = 47 + 16 + 9 + 1 = 73.$$

The LOP optimal solution to this illustrative example is $p^1 = (3,2,1,4)$ with a maximum objective function value of $C_{LOP}(p^1) = 22$. The objective function value corresponding to the LOPCC is $C_{LOPCC}(p^1) = 152$, which does not reach the maximum value of 261 associated with the solution (3,4,2,1). Symmetrically, the LOPCC optimal solution is $p^2 = (4,1,3,2)$ with a minimum value of $C_{LOPCC}(p^2) = 61$. The LOP objective function value of this solution is $C_{LOP}(p^2) = 10$, which does not match the minimum value of 8 associated with the solution (4,1,2,3). In other words, a solution p that maximizes the value of $C_{LOP}(p)$ does not necessarily maximize the value of $C_{LOPCC}(p)$, in the same way that a solution p that minimizes the value of $C_{LOPCC}(p)$ does not result in a minimum value of $C_{LOP}(p)$. In fact, we generated 25 random small instances ($n = 5$) and computed the correlation coefficient between the C_{LOP} and the C_{LOPCC} values in the universe of solutions. The average value of this coefficient across the 25 instances is 0.34, confirming that the two objective functions are not as tightly related as the casual observation of their mathematical expressions may seem to indicate.

These examples illustrate that although both the LOP and the LOPCC share some common characteristics, the calculation of their respective objective functions is such that it makes their associated solution space structurally different. This is why it cannot be expected that a solution procedure for the LOP will perform well when applied to the LOPCC. However, since the LOP has been thoroughly studied for a number of years, it wouldn't be wise to ignore the search strategies devised in this context when developing a customized solution for the LOPCC.

Although the interest for the LOPCC was triggered by the UMTS application, our interest in the problem goes beyond this context. This is parallel to development of solution methods for the LOP, which was introduced in the context of input-output tables. Therefore, we study the LOPCC in order to design effective search strategies and then apply the resulting method to several problem instances, including those related to the UMTS application. In particular, we introduce additional instances and also modify LOP instances found in the LOLIB library.

The linear ordering problem has generated a considerable amount of research interest since 1958, when Chenery and Watanabe outlined some ideas on how to obtain solutions for this problem and Becker (1967) proposed the first heuristic based on calculating quotients to rank each node in the graph. The proposed procedure is quite fast and produces reasonable results considering its simplicity. After this seminal work, several heuristics and metaheuristics have been proposed. We review a relevant subset of these procedures.

The multi-start method by Chanas and Kobylanski (1996) is based on a mechanism that searches for the best position for inserting a node in the partial ordering under construction. Nodes are scanned according to the order established by the current solution. When no further improvement is possible, the order given by the current local optimum is reversed and the process is re-started.

Recently, metaheuristic optimization has been applied to the LOP. Laguna, et al. (1999) proposed two procedures based on the tabu search methodology, one limited to short term memory structures and one incorporating long term memory components. Starting from a randomly generated permutation *p*, the short-term procedure alternates between intensification and diversification phases. Intensification iterations start with the random selection of a node. The probability of selecting a given node is proportional to an influence measure. The method scans the neighborhood in search of the first move with a strictly positive value (i.e., a move such that $C_{LOP}(p') > C_{LOP}(p)$, where p is the current permutation of nodes and p' is the permutation that results after the execution of the chosen move). The first improving, if available, or the best non-improving move is selected. The node that is moved becomes tabu-active for a pre-established number of iterations, and therefore it cannot be selected for insertions during this time. In the diversification phase, the probability of selecting a node is inversely proportional to the number of times that the node has been selected during the intensification phase. The basic procedure stops when a number of global iterations (intensification phase followed by the diversification phase) are performed without improving the best solution found so far. Laguna, et al. (1999) also proposed an extended tabu search method in which the basic procedure is coupled with both long-term intensification and diversification. The intensification incorporates elements from the path relinking methodology, while the diversification is inspired by a symmetry property previously introduced by Chanas and Kobylanski (1996).

Campos, et al. (2001) adapt the evolutionary method known as scatter search to the LOP. The adaptation follows the so-called *template* introduced by Glover (1998). Diversification generation is based on modifying a measure of attractiveness proposed by Becker (1967) with a frequency measure that discourages nodes from occupying positions that they have frequently occupied in previous solution generations. Local improvement consists of a "hill climbing" heuristic that chooses the best position where to insert a node. The *reference set* is updated with the "best" solutions found (where the meaning of best includes not only solution quality but also diversity). Subsets of reference solutions are generated and then subjected to a combination method. The solution combination method is based on a voting scheme that creates a new trial solution from a subset of reference solutions. (See Laguna and Martí (2003) for a detailed description of the scatter search methodology and its applications.) Computational experiments were performed to compare the scatter search implementation with the multi-start method developed by Chanas and Kobylanski and the tabu search procedures by Laguna, et al. (1999). The experiments showed that both the tabu search and the scatter search variants have very small average deviations from optimality for the well-known LOLIB instances.

García et al. (2006) applied the variable neighborhood search (VNS) to the LOP. Based on a systematic change of neighborhood in a local search procedure, VNS uses both deterministic and random strategies in search for the global optimum. The method combines several neighborhoods for a thorough exploration of the search space. The authors explore different search strategies and propose a hybrid method in which the VNS is coupled with a short term tabu search for improved outcomes. Extensive experimentation with both real and synthetic instances shows that the proposed procedure is highly competitive when compared to the tabu and scatter search implementations.

Bertacco, et al. (2008) show that the LOPCC is NP-hard. They also propose a mixed integer linear programming formulation of the BLOPCC, which is derived from the LOP model of Grötschel, et al. (1984). The authors solve small instances with up to 16 nodes with *Cplex* as well as with a customized branch and bound algorithm. Both methods are able to solve optimally all the instances with up to 12 nodes in a few seconds on a personal computer. For problems with 14 and 16 nodes, *Cplex* failed in most cases while the branch-and-bound procedure still succeeded. Experiments with problems of size 20 resulted in solution times of about 4 hours.

It is interesting to point out that the mixed-integer programming formulation for the BLOPCC may be modified to solve the more general LOPCC. The modification consists of changing the values of U with a sufficiently large M value. It is well-known, however, that big-M formulations produce weak linear programming relaxations, rendering the model useless for the purpose of obtaining optimal solutions to the LOPCC.

Benvenuto, et al. (2005) tackled the LOPCC in the context of mobile-phone telecommunication systems, referring to the problem as the *Joint Optimization of Power Control and Ordering* (JOPCO). The authors propose a greedy heuristic as well as a greedy randomized (GR) procedure in which the node inserted in the path under construction is randomly selected from a reduced candidate list of the best available nodes. GR is iterated a pre-established number of times and the solution achieving the lowest system transmit power is selected. The authors also consider solving the mixed integer programming formulation presented above and reported computational experiments with small instances ($n \leq 16$). The greedy method obtains solutions with an optimality gap in the neighborhood of 40%.

2. A Tabu Search Approach to the LOPCC

Our adaptation of tabu search for the LOPCC has three phases: construction, intensification and diversification. Instead of starting from a totally random solution, we developed a semi-greedy construction heuristic. The construction heuristic is used to generate a pre-specified number of solutions (*maxConstructions*) from which we select the one with the best objective function value to start the search. Figure 3 shows a description in form of pseudo-code of our construction heuristic.

We point out that the reduced candidate list in step 6 is built considering the increase in the objective function value caused by placing each unselected node in the current empty position k. However, once a node is chosen from this list, we search for the best position where to place the selected node. This entails calculating the ΔC values associated with placing the selected node in positions k+1 to n. (The ΔC value associated with placing the selected node in position k is already known from the calculation in step 4.) The ΔC values are computed efficiently as an incremental update from the previous step. The overall constructive method has a computational complexity of O(n^2).

```
1. Initialize the position pointer and the set of unselected nodes. Since
   the construction starts from the last element, the position pointer $k = n$
   and the set of unselected nodes UN = $\{1, \ldots, n\}$
2. Randomly choose node $i \in$ UN and place it in position $k$ and delete the
   chosen node from the unselected set. That is, $p_k = i$ and UN = UN \ $\{i\}$.
while ( UN $\neq \emptyset$)
{
    3. Move to the next empty position in the permutation. That is, make
       $k = k - 1$.
    4. Create a candidate list of nodes with the elements in UN.
    5. Calculate the increase in the objective function value ($\Delta C$) caused
       by placing an unselected node in position $k$. Since the LOPCC is a
       minimization problem, the most attractive node in the candidate
       list is the one that causes the least increase in the objective function
       value ($\Delta C_{\min}$) and the least attractive node is the one that causes the
       largest increase ($\Delta C_{\max}$).
    6. The candidate list is reduced by considering only the nodes for
       which $\Delta C \leq \Delta C_{\max} - \beta(\Delta C_{\max} - \Delta C_{\min})$. A node is then randomly
       chosen from this reduced candidate list.
    7. The chosen node, say $i$, is placed in the position, from $k$ to $n$, that
       minimizes the increase in the objective function. UN = UN \ $\{i\}$.
}
```

Figure 3. Greedy randomized adaptive construction

Once a solution has been chosen as starting point, a tabu search is launched. The two phases of the search interact as shown in Figure 4. Each phase begins from the current solution and after termination they return the overall best solution and a new current solution. The search terminates after *maxGlobal* iterations have elapsed without improving the overall best solution. A global iteration consists of executing steps 3 and 4 in Figure 4.

```
1. Find initial solution $p$ with the construction heuristic
2. Update the overall best solution found ($p_{\text{overall\_best}} = p$)
while (global iterations without improving overall best < $maxGlobal$)
{
    3. Execute the intensification phase starting from the current solution
       $p$. Return the best overall solution $p_{\text{overall\_best}}$ and the new current
       solution $p$.
    4. Execute the diversification phase starting from the current solution
       $p$. Return the best overall solution $p_{\text{overall\_best}}$ and the new current
       solution $p$.
}
```

Figure 4. Global iterations of the tabu search procedure

The intensification and diversification phases, steps 3 and 4 in Figure 4, were adapted from the tabu search procedure for the LOP described in Section 1. The adaptation consists of: 1) modifying the measure of influence for node selection in the intensification phase and 2) deriving a new move value computation.

An iteration of the intensification phase begins by randomly selecting a node. We consider the score value as a measure of influence to discriminate among the different nodes. The score value for node *i* in the context of the LOPCC is calculated as follows:

$$score(i) = \sum_{j} c_{ij} d_j + \sum_{j} c_{ji} d_i$$

In our intensification phase, we give priority to the low score nodes in an attempt to finding positions that will decrease the objective function value of the current solution. We insert the selected node in the first position that improves the objective value, if one exists, or alternatively in the best non-improving position. Note that this rule may result in the selection of a non-improving move. The move is executed even when the move value is not positive, resulting in a deterioration of the current objective function value. The displaced node becomes tabu-active for *tabuTenure* iterations, and therefore it cannot be selected for insertions during this number of iterations. The intensification phase terminates after *maxIntensify* consecutive iterations without improvement.

The diversification phase is performed for *maxDiversify* iterations. A node is randomly selected iteratively, where the probability of selecting a node is inversely proportional to its frequency count (which records the number of times that it has been chosen to be moved in the intensification phase). The chosen node is placed in the best position, as determined by the move values, allowing for non-improving moves to be executed.

The most computationally intensive element of our procedure is the search for a position where to move a chosen node. The search for the best position is performed in both phases and requires the calculation of the difference between the current objective function value and the value after the move. A brute-force approach would create a trial solution by temporarily executing a move on the current solution and then evaluating the objective function. The calculated value would then be compared to the current objective function value. Instead of this approach, we followed one that is more computationally efficient and that reduces the number of calculations.

The simplistic approach of evaluating move values by calculating the difference between the current objective function value and the objective function value after the move is inefficient at best and totally impractical when tackling all but fairly small problem instances. We use a 6-node example to show how we save computational time when evaluating trial moves within our neighborhood search. We then present the general formulas that we employ within our code.

Suppose that the current solution consists of the permutation p = (1, 2, 3, 4, 5, 6), then the corresponding objective function value is calculated as follows:

$$C_{LOPCC}(p) = \alpha_1 + \alpha_2 + \alpha_3 + \alpha_4 + \alpha_5 + \alpha_6$$

$$\begin{aligned}
\alpha_1 &= d_1 + c_{12}\alpha_2 + c_{13}\alpha_3 + c_{14}\alpha_4 + c_{15}\alpha_5 + c_{16}\alpha_6 \\
\alpha_2 &= d_2 + c_{23}\alpha_3 + c_{24}\alpha_4 + c_{25}\alpha_5 + c_{26}\alpha_6 \\
\alpha_3 &= d_3 + c_{34}\alpha_4 + c_{35}\alpha_5 + c_{36}\alpha_6 \\
\alpha_4 &= d_4 + c_{45}\alpha_5 + c_{46}\alpha_6 \\
\alpha_5 &= d_5 + c_{56}\alpha_6 \\
\alpha_6 &= d_6
\end{aligned}$$

The calculation is done in reversed order. That is, α_6 is calculated first, then α_5 and so on and so forth. Now, let us suppose that we would like to know the move value associated with inserting node 4 between nodes 1 and 2 to transform the current permutation into $p' = (1, 4, 2, 3, 5, 6)$. The objective function value of this neighbor solution is:

$$C_{LOPCC}(p') = \alpha'_1 + \alpha'_2 + \alpha'_3 + \alpha'_4 + \alpha_5 + \alpha_6$$

$$\begin{array}{ll|r|l}
\cline{3-3}
\alpha'_1 = & d_1 + & c_{14}\alpha'_4 + c_{12}\alpha'_2 + c_{13}\alpha'_3 & + c_{15}\alpha_5 + c_{16}\alpha_6 \\
\alpha'_4 = & d_4 + & c_{42}\alpha'_2 + c_{43}\alpha'_3 & + c_{45}\alpha_5 + c_{46}\alpha_6 \\
\alpha'_2 = & d_2 + & c_{23}\alpha'_3 & + c_{25}\alpha_5 + c_{26}\alpha_6 \\
\cline{3-3}
\alpha'_3 = & d_3 + & \multicolumn{1}{r}{} & c_{35}\alpha_5 + c_{46}\alpha_6 \\
\alpha_5 = & d_5 + & \multicolumn{1}{r}{} & c_{56}\alpha_6 \\
\alpha_6 = & d_6 & \multicolumn{1}{r}{} &
\end{array}$$

Clearly, the difference between $C_{LOPCC}(p)$ and $C_{LOPCC}(p')$ is given by the terms inside the box above. In general, moves that change a node j currently in position p_j to a position p_i for which $i < j$ cause 4 types of changes in the alpha values:

$$\begin{array}{c|c|c|c|c|c}
(& p_1, \ldots, p_{i-1}, & p_i, & p_{i+1}, \ldots, p_j, & p_{j+1}, \ldots, p_n &) \\
\multicolumn{1}{c}{} & \multicolumn{1}{c}{\text{Type 3}} & \multicolumn{1}{c}{\text{Type 2}} & \multicolumn{1}{c}{\text{Type 1}} & \multicolumn{1}{c}{\text{Type 0}} & \multicolumn{1}{c}{}
\end{array}$$

Mathematically, the difference between the new alpha value and the previous one (i.e., $\Delta\alpha = \alpha' - \alpha$) is given by the following expressions:

Type 0: $\Delta\alpha_{p_s} = 0 \quad \text{for} \quad s > j$

Type 1: $\Delta\alpha_{p_s} = -c_{p_s p_j}\alpha_{p_j} + \sum_{k=s+1}^{j-1} c_{p_s p_k}\Delta\alpha_{p_k} \quad \text{for} \quad s = j-1, j-2, \ldots, i$

Type 2: $\Delta\alpha_{p_j} = \sum_{k=i}^{j} c_{p_j p_k}(\alpha_{p_k} + \Delta\alpha_{p_k})$

Type 3: $\Delta\alpha_{p_s} = \sum_{k=s+1}^{j} c_{p_s p_k}\Delta\alpha_{p_k} \quad \text{for} \quad s = i-1, i-2, \ldots, 1$

For the opposite move direction, that is, for $i > j$, the differences in the alpha values are calculated as follows:

Type 0: $\Delta\alpha_{p_s} = 0 \quad \text{for} \quad s > i$

Type 1: $\Delta\alpha_{p_j} = -\sum_{k=j+1}^{i} c_{p_j p_k}\alpha_{p_k}$

Type 2: $\Delta\alpha_{p_s} = c_{p_s p_j}(\alpha_{p_j} + \Delta\alpha_{p_j}) + \sum_{k=s+1}^{i} c_{p_s p_k}\Delta\alpha_{p_k} \quad \text{for} \quad s = i, i-1, \ldots, j$

$$\text{Type 3:} \qquad \Delta\alpha_{p_s} = \sum_{k=s+1}^{i} c_{p_s p_k} \Delta\alpha_{p_k} \qquad \text{for} \quad s = j-1, j-2, \ldots, 1$$

We have empirically estimated that this procedure to compute the move value reduces the computational time by 50%.

3. Computational Experiments

The tabu search procedure described in the previous section was implemented in C and all experiments were performed on a personal computer with a 3.2 GHz Intel Xenon processor and 2.0 GB of RAM. We have tested the procedure on three sets of instances:

(1) *UMTS Instances* from the telecommunications group of the Engineering School of the University of Padova, related to detection-order optimization in UMTS networks, and previously reported in Bertacco, et al. (2008). We have considered the four communication scenarios considered in previous works, i.e., synchronous and asynchronous transmissions, with and without scrambling. There are 2000 instances in the original set, we selected the first 25 instances of size n = 16 from each group for a total of 100 instances in this set.

(2) *LOLIB Instances* from the public-domain library, consisting of input-output tables of sectors in the European economy. This is the well-known library for the LOP with 49 instances.

(3) *Random Instances* generated from a (0, 100) uniform distribution. Reinelt (1985) proposed the construction of these problem instances. We generate instances of sizes ranging from 35 to 150 nodes. There are 25 instances in each set for a total of 75.

Bertacco, et al. (2008) describe a process to obtain instances of the LOPCC from matrices associated with the LOP in the context of power transmission. We follow their procedure to obtain LOPCC instances (i.e., to generate the d-vector) for the matrices in the three sets described above. The resulting LOPCC instances are available for downloading at http://www.uv.es/rmarti.

The performance of our tabu search implementation depends on the judicious selection of the values for the six input parameters (*maxConstructions*, *maxGlobal*, *maxIntensify*, *maxDiversify*, *tabuTenure* and β). We start by setting *maxConstructions* = 100 and running an experiment to find the value of β that results in solutions with the highest quality. For this experiment, we employ the 25 random instances of size n = 100. Table 1 reports the average objective function value, the average percent deviation from the best-known solution, the number of best solutions out of 100, and the average CPU time in seconds for six different values of the β parameter.

Table 1. Fine-tuning of the construction heuristic

β	0.5	0.6	0.7	0.8	0.9	1.0
Avg. Obj. Fun.	3.55E+03	3.25E+03	2.84E+03	2.76E+03	2.71E+03	3.40E+03
Avg. Deviation	16.55%	13.18%	7.61%	5.14%	1.11%	31.39%
No. of best	0	2	5	13	22	1
CPU Time	0.55	0.54	0.54	0.53	0.53	0.52

The results summarized in Table 1 indicate that setting β to 0.9 gives, on average, the best outcome. These results compare favorably with the 97.47 % average deviation obtained by executing GR (Benvenuto, et al. 2005) for 100 constructions (which requires 0.05 seconds on average). If we run GR for 1000 constructions the quality of the solutions exhibits a moderate improvement with an average deviation still larger than 90%.

Since the search does not consist of only running the construction heuristic, we need to verify that the setting for β works well when the rest of the search is performed. We use the fine-tuning system known as Calibra[1] (Adenso-Diaz and Laguna, 2006) to find the best settings for *maxIntensify*, *maxDiversify*, *tabuTenure* and β. Calibra utilizes design of experiments and local search to determine the best values for up to 5 input parameters. We setup the experiment to instruct Calibra to search for the best parameter values within the following set of values:

$$maxIntensify = \{ 0.5n, 0.6n, \ldots, 2.0n \}$$
$$maxDiversify = \{ 0.5n, 0.6n, \ldots, 2.0n \}$$
$$tabuTenure = \{ 0.5\sqrt{n}, 0.6\sqrt{n}, \ldots, 2.0\sqrt{n} \}$$
$$\beta = \{ 0.5, 0.6, \ldots, 1.0 \}$$

The quality of the solutions generated by our tabu search procedure can only increase with the value of *maxGlobal*. Calibra searches for the best parameter values considering the quality of the solutions produced by the procedure being subjected to the fine-tuning process. Calibra would always choose the largest allowed value for *maxGlobal* if we let Calibra adjust this parameter. Hence, we set *maxGlobal* to 10 and let Calibra adjust the other 4 parameters. The Calibra run confirmed that the best value for β is 0.9 and suggested the following values as the best for the rest of the parameters:

$$maxIntensify = 0.8n$$
$$maxDiversify = 1.7n$$
$$tabuTenure = 1.7\sqrt{n}$$

With the search parameters set as indicated above, we proceed to compare the relative merit of our tabu search approach, referred to as TS_LOPCC. In the following set of experiments we compare the performance of TS_LOPCC with the Enumerative Method (EM), i.e. the branch and bound of Bertacco, et al. (2008), and the Greedy Randomized procedure (GR) of Benvenuto, et al. (2005). We employed the EM implementation that the authors sent to us, and for which we are grateful. On the other hand, we implemented the GR method according to the description published in Benvenuto, et al. (2005) and set the number of iterations to 10,000 to obtain an execution time close to the TS_LOPCC method.

[1] The latest version of Calibra is available for download at http://coruxa.epsig.uniovi.es/~adenso/wincalibra_download.exe

As mentioned in the introduction, the LOPCC shares some common features with the LOP and therefore one might wonder whether a procedure designed for the LOP, when properly modified, would perform well when applied to the LOPCC. In order to test this, we modified the tabu search method (TS_LOP) originally developed by Laguna, et al. (1999) for the LOP. The modification is straightforward and was done to obtain a baseline for comparison and to confirm the need for a specialized method for the LOPCC. We made two basic changes to TS_LOP. First, we changed the search from maximization to minimization. And second, given a solution *p*, instead of computing its value as $C_{LOP}(p)$, we compute it as $C'_{LOP}(p,d)$, which roughly approximates the $C_{LOPCC}(p)$ value, where:

$$C'_{LOP}(p,d) = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} d_{p_j} c_{p_i p_j}$$

Tables 2 to 9 show, for each problem set and method, the average objective function value (Obj. Function), the average percent deviation from the optimal or best-known solution (Avg. Deviation), its standard deviation (Std. Deviation), the number of optimal or best-known solutions (Num. of Opt. or Best), and the average computational time (CPU seconds). Since optimal solutions are not known for the random instances, the deviation values reported in Tables 7 to 10 are against the best solution found during the experiment. Also for these tables, we report the number of best solutions found instead of the number of optimal solutions.

Table 2. 25 *UMTS* instances (size 16) synchronous without scrambling.

	EM	**GR**	**TS_LOP**	**TS_LOPCC**
Obj. Function	6.238	7.389	7.018	6.238
Avg. Deviation	0.0%	15.1%	8.8%	0.0%
Std. Deviation	0.0%	8.8%	8.3%	0.0%
Num. of Opt.	25	0	0	25
CPU seconds	18.39	0.11	0.07	0.06

Table 3. 25 *UMTS* instances (size 16) synchronous with scrambling.

	EM	**GR**	**TS_LOP**	**TS_LOPCC**
Obj. Function	14.000	23.052	21.818	14.00
Avg. Deviation	0.0%	36.4%	24.1%	0.03%
Std. Deviation	0.0%	15.2%	17.2%	0.2%
Num. of Opt.	25	0	1	24
CPU seconds	7.72	0.11	0.07	0.03

Table 4. 25 *UMTS* instances (size 16) asynchronous without scrambling.

	EM	**GR**	**TS_LOP**	**TS_LOPCC**
Obj. Function	12.427	19.732	21.598	12.43
Avg. Deviation	0.0%	34.7%	30.7%	0.03%
Std. Deviation	0.0%	11.4%	16.6%	0.1%
Num. of Opt.	25	0	0	24
CPU seconds	44.77	0.12	0.08	0.03

Table 5. 25 *UMTS* instances (size 16) asynchronous with scrambling.

	EM	**GR**	**TS_LOP**	**TS_LOPCC**
Obj. Function	19.030	33.014	37.121	19.05
Avg. Deviation	0.0%	40.5%	35.2%	0.06%
Std. Deviation	0.0%	11.0%	20.4%	0.3%
Num. of Opt.	25	0	0	24
CPU seconds	22.28	0.12	0.07	0.03

Tables 2 to 5 show that for the UMTS instances, the GR procedure is clearly inferior in terms of solution quality, although its performance is quite acceptable if we consider that GR employs very simplistic search strategies. Surprisingly, the TS_LOP, originally designed for the linear ordering problem without cumulative costs, performs slightly better than the GR method that was specifically designed for this problem. The results show that TS_LOPCC is capable of finding optimal solutions with high frequency (97 out of 100) and has very modest computational requirements (0.03 seconds). The performance of EM, averaging 17.72 seconds for finding optimal solutions, is in line with the results reported by Bertacco, et al. (2008). The minimum and maximum deviations in the 100 UMTS instances reported in Tables 2 to 5 are 12.11% and 59.10% for the GR method, 2.06% and 60.98% for the TS_LOP and 0.00% and 0.73% for the TS_LOPCC method. These results give initial evidence of the robustness of the proposed method, a conjecture that we test with additional experiments, whose results are presented in Tables 6 to 9.

Table 6. 49 *LOLIB* instances (size 50)

	GR	**TS_LOP**	**TS_LOPCC**
Obj. Function	5.82E+12	8.564E+11	7.24E+08
Avg. Deviation	82.77%	71.70%	1.61%
Std. Deviation	32.7%	27.0%	15.8%
Num. of Best	1	0	48
CPU seconds	1.91	0.56	2.31

Table 7. 25 *Random* instances (size 35)

	GR	**TS_LOP**	**TS_LOPCC**
Obj. Function	0.935	3486712454	0.344
Avg. Deviation	48.8%	100.0%	0.86%
Std. Deviation	20.0%	0.0%	1.6%
Num. of Best	1	0	24
CPU seconds	1.68	10.23	0.39

Table 8. 25 *Random* instances (size 100)

	GR	**TS_LOP**	**TS_LOPCC**
Obj. Function	416417.87	8.5035E+30	1197.52
Avg. Deviation	98.56%	100.0%	0.0%
Std. Deviation	1.0%	0.0%	0.0%
Num. of Best	0	0	25
CPU seconds	200.01	86.21	30.75

Table 9. 25 *Random* instances (size 150)

	GR	**TS_LOP**	**TS_LOPCC**
Obj. Function	46.165E+08	2.2411E+50	2252617.61
Avg. Deviation	99.91%	100.0%	0.0%
Std. Deviation	0.2%	0.0%	0.0%
Num. of Best	0	0	25
CPU seconds	200.03	281.14	180.43

The results for medium and large size instances ($35 \leq n \leq 150$) are shown in Tables 6 to 9. These experiments confirm the conclusions drawn from the small size instances. TS_LOPCC clearly outperforms the other two methods. For instance, the number of best solutions obtained with GR, TS_LOP and TS_LOPCC is 2, 0 and 116 (out of 119 large instances), respectively. We attempted to solve one instance of size 35 with EM and the procedure did not terminate after 4 days of computing time. The best-known solutions to these problems, therefore, are the ones found with the heuristics that we tested. The deterioration of the solution quality obtained by TS_LOP from the problems in Tables 2-5 to the problems in Tables 6-9 has an intuitive explanation. As the size of the instances increases, the nonlinear terms in C_{LOPCC} (p) become more important causing a more severe underestimation of the $C'_{LOP}(p,d)$ approximation. The minimum and maximum deviations in the 118 instances reported in Tables 6 to 9 are 66.35% and 94.83% for the GR method, 80.86% and 100.00% for the TS_LOP and 0.00% and 24.86% for the TS_LOPCC method. Tables 10, 11 and 12 in the Appendix show the best solution value found for each problem instance in the *UMTS*, *LOLIB* and *Random* sets, respectively. Our method is able to match the best value in 219 out of the 224 instances considered (with an asterisk between parentheses indicating an instance in which TS_LPOCC does not match the best known objective function value).

To provide additional performance insights, we construct a time-to-target (TTT) plot for TS_LOPCC when applied to a given problem instance. TTT plots have been developed with the goal of analyzing the probability (abscissa axis) that a stochastic search procedure finds a solution with an objective function value that is at least as good as a target within a specified computational time (ordinate axis). TTT plots were used by Feo, Resende and Smith (1994) and have been advocated by Hoos and Stützle (1998) and assume that computational times follow an exponential distribution.

We consider the t1d100.1 instance from the *Random* set, which has a best-known objective function value of 294.7. TS_LOPCC is run m times and the resulting computational times employed to reach the target value are stored and sorted in increasing order. Associate with the i-th sorted running time t_i is a probability value $p_i = (i - 1/2)/m$. The (t_i, p_i) points, for $i = 1, .., m$, are plotted to construct the TTT plot shown in Figure 5 (see empirical). This figure was obtained with the *perl* program by Aiex, et al. (2008) and m=200. The theoretical curve is constructed by assuming the computational times grow exponentially with the probability of finding a solution whose objective function value matches the specified target.

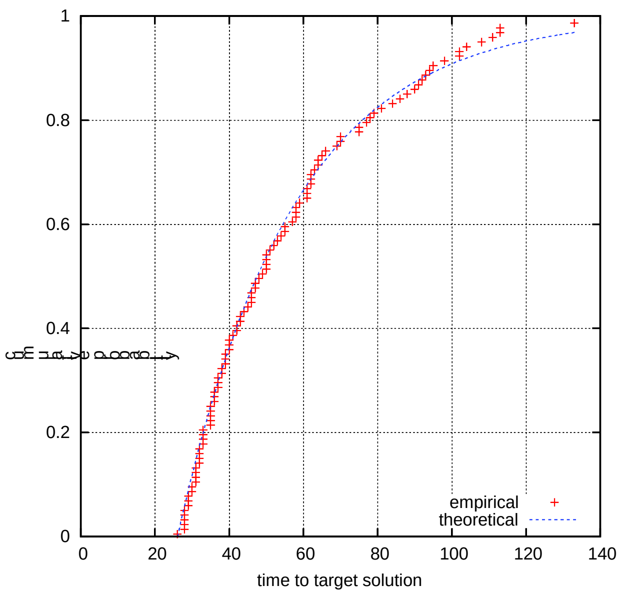

Figure 5. Time-to- target plot for t1d100.1

Figure 5 shows that the probability that TS_LOPCC finds a solution that is at least as good as the target value of 294.7 in at most 60 seconds is about 60%, in at most 80 is about 80% and in at most 100 seconds is about 90%. The empirical probabilities match well the theoretical curve in this example and we have verified (with a large sample) that similar plots are obtained for hard problem instances in our test sets.

4. Conclusions

We described the development of a tabu search for the linear ordering problem with cumulative costs. The problem is of practical significance and, given its complexity, the application of metaheuristic technology is well justified. In our research, we exploit the similarities of the LOPCC and the LOP, and, at the same time, we acknowledge their differences. In particular, we showed that the objective function in the LOPCC is such that a naïve modification of a solution method for the LOP is not capable of finding high-quality solutions to LOPCC instances as the size of the problems increases. Our tabu search adaptation for the LOPCC borrows from our previous experiences with the LOP while incorporating strategic search elements that proved effective in the current context.

Acknowledgments

This research has been partially supported by the Ministerio de Ciencia e Innovación (TIN2006-02696, SEJ2005-08923/ECON), by the Comunidad de Madrid (URJC/TIC-3731) and by the Government of Castilla y León (BU008A06).

References

Adenso-Díaz, B. and M. Laguna (2006) “Fine-tuning of Algorithms Using Partial Experimental Designs and Local Search,” *Operations Research*, vol. 54, no. 1, pp. 99-114.

Aiex, R.M., M. G. C. Resende and C. C. Ribeiro (2007) “TTTPLOTS: A perl Program to Create Time-to-Target Plots” *Optimization Letters*, vol. 1, pp. 355-366.

Becker, O. (1967) “Das Helmstädtersche Reihenfolgeproblem — die Effizienz verschiedener Näherungsverfahren” in *Computer uses in the Social Sciences*, Berichteiner Working Conference, Wien, January 1967.

Benvenuto, N., G. Carnevale and S. Tomasin (2005) “Optimum Power Control and Ordering in SIC Receivers for Uplink CDMA Systems,” IEEE-ICC 2005, Seoul, Korea.

Bertacco, L., L. Brunetta and M. Fischetti (2008) “The Linear Ordering Problem with Cumulative Costs”, *European Journal of Operational Research*, vol. 189, no. 3, pp. 1345-1357.

Campos, V., F. Glover, M. Laguna and R. Martí (2001) “An Experimental Evaluation of a Scatter Search for the Linear Ordering Problem”, *Journal of Global Optimization*, vol. 21, pp. 397-414.

Chanas, S. and P. Kobylanski (1996) “A New Heuristic Algorithm Solving the Linear Ordering Problem”, *Computational Optimization and Applications*, vol. 6, pp. 191-205.

Feo, T. A., M. G. C. Resende and S. H. Smith (1994) “A Greedy Randomized Search Procedure for Maximum Independence Set,” *Operations Research*, vol. 42, pp. 860-878.

Grotschel, M., M. Junger and G. Reinelt (1984) “A Cutting Plane Algorithm for the Linear Ordering Problem”, *Operations Research*, vol. 32, no. 6, pp. 1195-1220.

Hoos, H. H. and T. Stützle (1998) “Evaluating Las Vegas Algorithms — Pitfalls and Remedies” in *Proc. of the 14th Conf. on Uncertainty in Artificial Intelligence*, pp. 238-245.

Laguna, M., R. Martí and V. Campos (1999) “Intensification and Diversification with Elite Tabu Search Solutions for the Linear Ordering Problem,” *Computers and Operations Research*, vol. 26, pp. 1217-1230.

LOLIB (1997) http://www.iwr.uni-heildelberg.de/groups/comopt/software/LOLIB/

Proakis, J.G. (2004) *Digital Communications 4th edition*, Mc Graw Hill, New York.

Reinelt, G. (1985) *The Linear Ordering Problem: Algorithms and Applications*, Research and Exposition in Mathematics, vol. 8, H. H. Hofmann and R. Wille (Eds.), Heldermann Verlag Berlin.

Appendix: Best-Known Solutions to all Test Problems

Table 10. Best known objective function values to *UMTS* instances

Instance	Value	Instance	Value	Instance	Value	Instance	Value
mrho1_00	12.675	mrho1_01	18.756	mrho1_10	17.105	mrho1_11	25.610
mrho2_00	2.953	mrho2_01(*)	9.897	mrho2_10	7.435	mrho2_11	19.289
mrho3_00	3.774	mrho3_01	7.543	mrho3_10	7.172	mrho3_11	5.087
mrho4_00	5.002	mrho4_01	7.430	mrho4_10	6.777	mrho4_11	8.999
mrho5_00	4.021	mrho5_01	6.161	mrho5_10(*)	6.161	mrho5_11	13.171
mrho6_00	9.856	mrho6_01	17.084	mrho6_10	18.363	mrho6_11	20.803
mrho7_00	3.372	mrho7_01	8.547	mrho7_10	7.447	mrho7_11	13.110
mrho8_00	5.633	mrho8_01	7.563	mrho8_10	12.614	mrho8_11	10.060
mrho9_00	4.627	mrho9_01	12.827	mrho9_10	13.253	mrho9_11	11.592
mrho10_00	6.741	mrho10_01	33.836	mrho10_10	18.139	mrho10_11(*)	38.887
mrho11_00	3.963	mrho11_01	8.506	mrho11_10	6.902	mrho11_11	9.281
mrho12_00	2.751	mrho12_01	3.452	mrho12_10	2.449	mrho12_11	3.216
mrho13_00	2.411	mrho13_01	3.228	mrho13_10	4.068	mrho13_11	4.496
mrho14_00	14.717	mrho14_01	35.729	mrho14_10	23.614	mrho14_11	44.275
mrho15_00	6.705	mrho15_01	16.581	mrho15_10	10.982	mrho15_11	35.022
mrho16_00	1.729	mrho16_01	2.175	mrho16_10	2.123	mrho16_11	4.565
mrho17_00	4.519	mrho17_01	13.720	mrho17_10	18.221	mrho17_11	9.099
mrho18_00	8.141	mrho18_01	22.326	mrho18_10	22.924	mrho18_11	40.619
mrho19_00	5.991	mrho19_01	5.922	mrho19_10	7.825	mrho19_11	8.037
mrho20_00	12.290	mrho20_01	21.502	mrho20_10	21.594	mrho20_11	23.450
mrho21_00	8.841	mrho21_01	28.360	mrho21_10	18.769	mrho21_11	29.041
mrho22_00	4.573	mrho22_01	10.727	mrho22_10	8.828	mrho22_11	12.379
mrho23_00	4.744	mrho23_01	8.888	mrho23_10	7.073	mrho23_11	10.597
mrho24_00	13.563	mrho24_01	32.150	mrho24_10	32.682	mrho24_11	66.905
mrho25_00	2.368	mrho25_01	7.090	mrho25_10	7.636	mrho25_11	8.758

(*) TS_LOPCC does not match the best-known solution to this problem instance

Table 11. Best known objective function values to *LOLIB* instances

Instance	Value	Instance	Value	Instance	Value
be75eec	6.61	t65n11xx	2.09	t75i11xx	4455.05
be75np	30386616.48	t65w11xx	19.26	t75k11xx	1.32
be75oi	3.07	t69r11xx	14.04	t75n11xx	9.90
be75tot	297138.99	t70b11xx	93.67	t75u11xx(*)	0.068427
stabu70	15.01	t70d11xx	79.74	tiw56n54	2.65
stabu74	14.03	t70d11xxb	4.44	tiw56n58	3.63
stabu75	9.42	t70f11xx	1.27	tiw56n62	3.02
t59b11xx	76396.22	t70i11xx	121146.03	tiw56n66	2.69
t59d11xx	4395.50	t70k11xx	0.50	tiw56n67	1.88
t59f11xx	24.14	t70l11xx	798.92	tiw56n72	1.57
t59i11xx	621682.25	t70n11xx	0.05	tiw56r54	2.66
t59n11xx	1618.90	t70u11xx	35490330260.19	tiw56r58	3.60
t65b11xx	29362.84	t70w11xx	0.50	tiw56r66	2.19
t65d11xx	3898.61	t70x11xx	0.23	tiw56r67	1.54
t65f11xx	1.25	t74d11xx	4.76	tiw56r72	1.35
t65i11xx	826127.40	t75d11xx	5.19		
t65l11xx	2663.86	t75e11xx	2338.28		

(*) TS_LOPCC does not match the best-known solution to this problem instance

Table 12: Best known objective function values to *Random* instances

Instance	Value	Instance	Value	Instance	Value
t1d35.1	0.931	t1d100.1	294.70	t1d150.1	11010.20
t1d35.2	0.167	t1d100.2	297.27	t1d150.2	216751.90
t1d35.3	0.155	t1d100.3	1431.21	t1d150.3	772872.52
t1d35.4	0.196	t1d100.4	8864.08	t1d150.4	88416.64
t1d35.5	1.394	t1d100.5	172.11	t1d150.5	100095.91
t1d35.6	0.200	t1d100.6	479.30	t1d150.6	58579.73
t1d35.7	0.120	t1d100.7	6638.95	t1d150.7	172627.38
t1d35.8	0.226	t1d100.8	3095.19	t1d150.8	353334.04
t1d35.9	0.436	t1d100.9	75.83	t1d150.9	434222.49
t1d35.10(*)	0.205	t1d100.10	174.44	t1d150.10	146736.66
t1d35.11	0.372	t1d100.11	259.80	t1d150.11	13900.04
t1d35.12	0.235	t1d100.12	246.92	t1d150.12	98680.52
t1d35.13	0.196	t1d100.13	841.47	t1d150.13	133838.81
t1d35.14	0.138	t1d100.14	279.38	t1d150.14	99741.17
t1d35.15	1.376	t1d100.15	458.02	t1d150.15	352880.29
t1d35.16	0.287	t1d100.16	829.98	t1d150.16	24497326.54
t1d35.17	0.200	t1d100.17	783.01	t1d150.17	98564.17
t1d35.18	0.384	t1d100.18	724.49	t1d150.18	767813.83
t1d35.19	0.238	t1d100.19	249.29	t1d150.19	96627.16
t1d35.20	0.068	t1d100.20	272.82	t1d150.20	2279010.02
t1d35.21	0.202	t1d100.21	232.05	t1d150.21	50249.96
t1d35.22	0.177	t1d100.22	176.64	t1d150.22	866394.89
t1d35.23	0.346	t1d100.23	1796.08	t1d150.23	23972543.85
t1d35.24	0.138	t1d100.24	559.05	t1d150.24	119426.11
t1d35.25	0.146	t1d100.25	698.71	t1d150.25	462316.51

(*) TS_LOPCC does not match the best-known solution to this problem instance