

Metaheuristics for the Linear Ordering Problem with Cumulative Costs

ABRAHAM DUARTE

Departamento de Ciencias de la Computación, Universidad Rey Juan Carlos, Spain.
Abraham.Duarte@urjc.es

RAFAEL MARTÍ

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

ADA ÁLVAREZ

Programa de Postgrado en Ingeniería de Sistemas, Universidad Autónoma de Nuevo León, México. Adalvarez@mail.uanl.mx

FRANCISCO ÁNGEL-BELLO

Departamento de Ingeniería Industrial y de Sistemas, Tecnológico de Monterrey, México.
fangel@itesm.mx

Original version: December 29, 2010.

Revised version: July 14, 2011

Abstract

The linear ordering problem with cumulative costs (LOPCC) is a variant of the well-known linear ordering problem, in which a cumulative propagation makes the objective function highly non-linear. The LOPCC has been recently introduced in the context of mobile-phone telecommunications. In this paper we propose two metaheuristic methods for this NP-hard problem. The first one is based on the GRASP methodology, while the second one implements an Iterated Greedy-Strategic Oscillation procedure. We also propose a post-processing based on Path Relinking to obtain improved outcomes. We compare our methods with the state-of-the-art procedures on a set of 218 previously reported instances. The comparison favors the Iterated Greedy – Strategic Oscillation with the Path Relinking post-processing, which is able to identify 87 new best objective function values.

Keywords: Combinatorial Optimization, Metaheuristics, Linear Ordering Problem

1. Introduction

The Linear Ordering Problem with Cumulative Costs (LOPCC) is an NP-hard problem arising in the context of UMTS mobile-phone communication systems. In a broadband uplink code-division multiple-access transmission (CDMA), interference among users is a key point. Successive interference cancellation (SIC) for code division multiple access systems is an attractive technique to reduce multi-user access interference in uplink transmissions (Benvenuto, 2005). SIC sequentially detects mobile terminals following a predetermined order and removes the associated interference, improving the detection capability for the next users. The LOPCC appears in this context as the SIC optimization to ensure a proper signal reception.

The LOPCC was originally introduced in Bertacco et al. (2005). It can be described in mathematical terms as follows: Let $G = (V, A)$ be a weighted directed graph, where V is the set of vertices and A the set of arcs ($n = |V|$, $m = |A|$). Let d_i be the nonnegative weight of vertex i and let c_{ij} be the nonnegative cost of arc (i, j) . The LOPCC consists of finding a Hamiltonian path $p = (p_1, p_2, \dots, p_n)$ or permutation and the corresponding vertex values α_j that minimize the expression:

$$C(p) = \sum_{j=1}^n \alpha_j, \text{ where } \alpha_{p_i} = d_{p_i} + \sum_{j=i+1}^n c_{p_i p_j} \alpha_{p_j} \text{ for } i = n, n-1, \dots, 1.$$

The cumulative backward computation of the α -values, from n to 1, makes the objective function nonlinear and, as stated in Duarte et al. (2011), makes its solution space structurally different than the solution space of the classical linear ordering problem (LOP), and therefore solution procedures for the LOP do not perform well on the LOPCC.

Bertacco et al. (2005) introduced LOPCC and its bounded variant BLOPCC, showing that both problems are NP-hard. They also presented a Mixed-Integer linear Programming model (MIP) and an ad-hoc enumerative method. The authors solved small instances with up to 16 nodes with Cplex as well as with a customized branch and bound algorithm. Both methods are able to optimally solve 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 their branch-and-bound procedure still success. Experiments with instances of size 20 resulted in solution times of about 4 hours.

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 proposed 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 target the MIP formulation (Bertacco et al., 2005) and reported computational experiments with small instances ($n \leq 16$). The greedy method obtains solutions with an optimality gap of about 40%.

Righini (2008) proposed a new lower bound applied in a branch-and-bound algorithm for the provably optimization of the LOPCC. The algorithm implements a strategy for sorting the branches at each node of the search tree. Specifically, the first leaf that the branch and bound examines corresponds to the vertex with the largest cost. As shown in their computational experiments, this technique allows the algorithm to find the optimal solution earlier. The authors also presented an effective heuristic based on the truncation of the branch-and-bound algorithm (TB&B). The truncation reduces the total running time in a fraction of 10.

Duarte et al. (2011) proposed an adaptation of the tabu search methodology to the LOPCC with three phases: construction, intensification and diversification. The construction heuristic

is used to generate a pre-specified number of solutions from which select the one with the best objective function value to launch a tabu search. Then, the two phases of the search, intensification and diversification, alternate until a maximum number of global iterations have elapsed without improving the overall best solution. Each phase begins with the current solution and after termination both return the overall best solution. They are adapted from the respective phases of the tabu search procedure for the classical LOP (Laguna et al., 1999). The adaptation consists of modifying the measure of influence for move selection and the introduction of a new move value computation. The extensive experimentation shows that long-term diversification, based on frequency memory, coupled with the intensification strategy based on local search, is able to enhance the performance of a basic tabu search procedure and outperforms previous competing methods.

In this paper, we propose two metaheuristic procedures for the LOPCC. The first one is based on the GRASP methodology, while the second one implements an Iterated Greedy-Strategic Oscillation procedure. We also propose a post-processing based on Path Relinking to obtain improved outcomes. We compare our best procedure with the greedy randomized by Benvenuto et al. (2005), the tabu search by Duarte et al. (2011), and the truncated branch and bound by Righini (2008). Our extensive experimentation with 218 instances shows that the proposed procedure outperforms the state of the art methods in terms of solution quality with a reasonable computing-time requirement.

2. GRASP

The GRASP methodology was developed in the late 1980s (Feo and Resende, 1989) and the acronym was coined in Feo et al. (1994). Basically, each GRASP iteration consists in constructing a trial solution, combining randomization and greediness in a specific way, and then applying local search from the constructed solution to reach a local optimum. We refer the reader to Resende and Ribeiro (2003; 2010) for recent surveys of this metaheuristic methodology.

```

begin Constructive
1.   $U \leftarrow \{1, \dots, n\}$  %Set of unselected vertices
2.   $sol \leftarrow \emptyset$  %Solution under construction
3.  Select  $i$  randomly form  $U$ 
4.   $k \leftarrow n$ 
5.  Set  $i$  in the position  $k$  of  $sol$ 
6.   $U \leftarrow U \setminus \{i\}$ 
7.  while  $U \neq \emptyset$  do
8.       $k \leftarrow k - 1.$ 
9.       $\forall j \in U$  Compute  $\Delta C(j)$  in position  $k.$ 
10.      $C_{\max} \leftarrow \max\{\Delta C(j) / j \in V\}$ 
11.      $C_{\min} \leftarrow \min\{\Delta C(j) / j \in V\}$ 
12.      $RCL \leftarrow \{j \in U / \Delta C(j) \leq C_{\min} + \beta(C_{\max} - C_{\min})\}$ 
13.     Select  $i$  randomly from RCL
14.     Set  $i$  in the position  $k$  of  $sol$ 
15.      $U = U \setminus \{i\}$ 
16. end while
17. return  $sol$ 
end

```

Figure 1. Constructive procedure

The constructive procedure for the LOPCC randomly selects a vertex in the first iteration and places it in the last position, n , of the permutation (solution) under construction. In the next iterations, a vertex is selected according to the increase of the objective function, ΔC , caused

by placing an unselected node in the last available position, say k . Instead of selecting the “best” vertex with respect to ΔC , as a greedy algorithm would do, the procedure creates a restricted candidate list, RCL, of “good” vertices and randomly selects one of them. As it is customary in GRASP, the RCL contains those unselected vertices with a ΔC value below a threshold. Figure 1 shows the implementation details of this method.

Duarte et al. (2011) introduced a vertex score value (measure of influence) to discriminate among different vertices in their tabu search procedure. Specifically, the score value for vertex i is computed as:

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

Instead of selecting vertices according to this score, our local search procedure considers the contribution of each vertex, α -values, to the objective function. Specifically, the LS procedure selects the element with the largest contribution to the value of the current solution. In mathematical terms, we select the element i^* with the minimum α value.

$$i^* = \arg \min_{i \in V} \{\alpha_i\}$$

This new strategy takes into account that the vertices placed at the end of the permutation (close to position n) are “more critical” than the vertices placed at the beginning of the permutation (close to position 1) because their contribution to the objective function is much larger than the others due to the cumulative backward propagation of the objective function. In other words, our LS procedure tries to move some vertices from the last positions to the first positions. If there is no improving move associated with i^* , we resort to the next element with the largest α_{i^*} value and so on. This local search method performs iterations until no further improvement is possible.

```

begin LocalSearch(sol)
1.   Let  $\alpha = \{\alpha_1, \alpha_2, \dots, \alpha_n\}$  be the contribution of
     each vertex to the objective function
2.   imp  $\leftarrow$  true
3.   while imp do
4.     imp  $\leftarrow$  false
5.     for  $j = 1$  to  $n$  do
6.       Sort  $\alpha$  in ascending order
7.       Let  $i$  be the element with  $\alpha[j]$  contrib
8.       Let  $pos\_i$  the position of  $i$  in  $sol$ 
9.       for  $k = pos\_i - 1$  to  $1$  do
10.         $sol \leftarrow swap(sol, k)$ 
11.         $sBest \leftarrow UpdateBestSol(sol, imp)$ 
12.      end for
13.    end for
14.  end while
15.  return  $sBest$ 
end

```

Figure 2. Local search procedure

The neighborhood of our local search method is defined by an *insert* operation. In other words, an element is removed from its current position i and inserted in another position j (with $j < i$ to move it to a previous position). However, instead of directly considering a target position j , the method scans the intermediate swap moves of consecutive positions in a search for the best improving move. Specifically, given a vertex in a position i , we first try to exchange the vertices in positions i and $i - 1$; and record the associated move value (change in the objective function); then we try to exchange the vertices in positions $i - 1$ and $i - 2$

record the associated move value and so on until we reach position 1. In this way we compute the move values in an incremental way, thus accelerating this time-consuming computation. This decomposition of a general insert move into its “associated” swap moves was successfully implemented by Schiavinotto and Stützle (2004) in the context of the classical LOP. Figure 2 shows the implementation details of our local search procedure for the LOPCC.

Exploring the neighborhood of a solution is computationally expensive given the backward computation of the α -values. We have however considered an incremental computation of these values when a swap of consecutive vertices is performed. In particular, suppose that the current solution consists of the permutation $p = (1, 2, \dots, p_{i-1}, p_i, \dots, p_n)$ and that we exchange vertices p_{i-1} and p_i , thus obtaining $p' = (1, 2, \dots, p_{i-2}, p_i, p_{i-1}, \dots, p_n)$. Then, the difference between the new alpha value, α' , and the previous one, α , ($\Delta\alpha = \alpha' - \alpha$) is given by:

$$\begin{aligned}\Delta\alpha_{p_s} &= 0 && \text{for } i < s \leq n \\ \Delta\alpha_{p_{i-1}} &= -c_{p_{i-1}p_i} \alpha_{p_i} \\ \Delta\alpha_{p_i} &= c_{p_i p_{i-1}} (\alpha_{p_{i-1}} + \Delta\alpha_{p_{i-1}}) \\ \Delta\alpha_{p_s} &= \sum_{k=s+1}^i c_{p_s p_k} \Delta\alpha_{p_k} && \text{for } 1 \leq s \leq i - 2\end{aligned}$$

The combination of the composition of swap moves to obtain a general insert move with the incremental computation of the α -values above, makes the local search very efficient, as it will be shown in our computational experience.

3. Iterated Greedy – Strategic Oscillation for the LOPCC

Iterated Greedy (IG) is a recent methodology that has been successfully applied to some hard optimization problems. See for example, Marchiori and Steenbeek (2000), for an implementation on the set covering problem, or Ruiz and Stützle (2007) for an application on a scheduling problem. It is based on a simple and effective principle: generate a sequence of solutions by iterating over a greedy constructive heuristic applying two phases, destruction and reconstruction (Jacobs and Brusco 1995, Culberson and Luo. 1996). Figure 3 shows the general IG outline.

```

begin IteratedGreedy
1.  s0 ← GreedyConstruction()
2.  s  ← ImproveSolution(s0)
3.  while not TerminationCondition do
4.      sd ← Destruction(s)
5.      sc ← Construction(sd)
6.      s' ← ImproveSolution(sc)
7.      s  ← AcceptanceCriterion(s', s)
8.      sBest ← UpdateBestSolution(s)
9.  end while
10. return sBest
end

```

Figure 3. Iterated greedy procedure.

As shown in Figure 3, IG starts from an initial solution (step 1) obtained with a greedy constructive heuristic. Then it generates a sequence of solutions by partially destructing and reconstructing this solution (steps 3 to 9) with the greedy algorithm. This is why we say that IG

iterates over a greedy algorithm. Specifically, during the destruction phase (step 4), some elements of the current solution are eliminated, obtaining a partial solution. In the construction phase (step 5), the greedy constructive heuristic is applied to add elements into the partial solution until a complete solution is obtained. Once a newly reconstructed solution has been obtained, an acceptance criterion is applied to decide whether it will replace the incumbent solution or not (step 8). The process iterates through these two phases until some termination condition is met (e.g. maximum number of iterations, or maximum computation time allowed). Additionally, an optional local search phase for improving the initial and reconstructed solutions can be applied (step 6).

The approach described above can also be considered an implementation of the strategic oscillation methodology (Glover and Laguna, 1997). Strategic oscillation (SO) is closely linked to the origins of tabu search, and operates by orienting moves in relation to a critical level, as identified by a stage of construction. In particular, we consider a constructive/destructive type of strategic oscillation, where constructive steps “add” elements (as in the step 5 of Figure 3) and destructive steps “drop” elements (as in the step 4). As described in Chapter 4 of the tabu search monograph (Glover and Laguna, 1997), the alternation of constructive with destructive processes, which strategically dismantle and then rebuild successive trial solutions, affords an enhancement of such traditional constructive procedures.

In our Iterated Greedy-Strategic Oscillation implementation for the LOPCC we apply the constructive algorithm described in Section 2 with β set to 1 (see Figure 1) as the greedy procedure. Additionally, we apply the local search described in Figure 2 as the improvement procedure. As mentioned, the main phases in an IG algorithm are the destruction and the construction. The destruction procedure in our method is applied by randomly selecting $\%k$ vertices in the solution. These elements (vertices) are then removed from the solution. Since the solutions are ordered sets (permutations), when we remove a vertex or a set of vertices from a solution, we consider shifting the remaining vertices to occupy the “blank spaces” left by the removed vertices. We apply two different strategies that we call *left pack* and *right pack*. In the former, vertices are shifted to the left while in the later vertices are shifted to the right. The following example illustrates both strategies.

Let $s = (2, 4, 3, 1, 5, 7, 6)$ be a solution with $n=7$. If we set $\%k = 30$, it means that the destruction procedure removes 2 elements from s at random. Consider that these two elements are vertices 3 and 7. Then, the destruction procedure returns the partial solution $ps = (2, 4, -, 1, 5, -, 6)$. The *left pack* strategy packs the elements at the beginning of the permutation, leaving the “empty positions” at the end; thus obtaining the partial solution $ps_1 = (2, 4, 1, 5, 6, -, -)$. On the other hand, the *right pack* strategy packs the elements at the end of the permutation and the empty positions are moved at the beginning, resulting in $ps_2 = (-, -, 2, 4, 1, 5, 6)$. These two strategies lead us to two different IG-SO procedures: IG-SO1 (*left pack*) and IG-SO2 (*right pack*). We study the performance of these strategies in Section 5 where we report our experimental results.

The construction phase focuses on the set of removed elements in the previous application of the destructive phase. These vertices are sorted according to the increasing of the objective function caused by placing the corresponding element in an unselected position. The construction procedure selects the “best” vertex; i.e., the one that causes the least increasing in the objective function value.

The most common acceptance criterion in this method (step 7 in the algorithm of Figure 3) specifies accepting new solutions only if they improve the best solution so far. However, we have empirically found that this can lead to a premature convergence of the method (or

getting trapped in a local optimum relatively far from the global one). As a consequence, we have implemented in IG-SO1 and IG-SO2 a more aggressive acceptance criterion for diversification purposes, based on accepting always the re-constructed solution.

4. Path Relinking post-optimization

Path relinking (PR) was suggested as an approach to integrate intensification and diversification strategies in the context of tabu search (Glover, 1996; Glover and Laguna, 1997). PR generates new solutions constructing a path from an *initiating* solution to a *guiding* solution. This process is accomplished by introducing attributes contained in the guiding solutions, and incorporating them in an intermediate solution originated in the initiating solution. In this section we adapt PR to the LOPCC as a post-processing of the GRASP or Iterated Greedy – Strategic Oscillation. Specifically, we explore different designs in which greedy, randomized, and evolutionary elements are considered in the PR implementation as a form of search intensification.

The PR algorithm operates on a set of solutions, called elite set (ES), constructed with the application of a previous method, which can be either GRASP or IG - SO in our case. Based on previous studies (see for example Resende and Werneck, 2004) this set should be created considering both quality and diversity. We therefore apply a method to generate b different solutions (with relatively good quality and diversity) to populate it. Given the pair of solutions (U, V) in ES, we consider the path from U to V (where U is the initiating solution and V the guiding one). In this path, we move one by one the elements in U to their corresponding position in the guiding solution. Specifically, given element (vertex) k , let k_u and k_v be its corresponding position in U and V respectively. To generate an intermediate solution, the procedure swaps the elements k and k' that are placed in positions k_u and k_v in U . Therefore, after a sequence of swaps, a series of intermediate solutions are generated and the guiding solution is reached. Figure 4 illustrates this method with $U = (2,1,4,3)$ and $V = (3,4,2,1)$. This figure only shows a small fraction of the solutions examined to generate a path from U to V . For $k = 1$, we have that $k_u = 2, k_v = 4$ and $k' = 3$ and we would swap vertex 1 with 3 in U obtaining solution $(2,3,4,1)$. Similarly, in the first step PR generates the following four intermediate solutions (depicted in the figure in the same column and with the swapped elements in white font): $(3, 1,4,2), (2,4, 1,3), (4,1,2, 3), (2,3,4,1)$. Consider for example that $(4,1,2, 3)$ is the best of the four in terms of the objective function; then we will apply the next swaps to it to generate intermediate solutions closer to $V = (3,4,2,1)$. Specifically, we generate the following three intermediate solutions: $(3, 1,2, 4), (1,4, 2, 3), (4,3,2, 1)$. In this way, the guiding solution is finally reached.

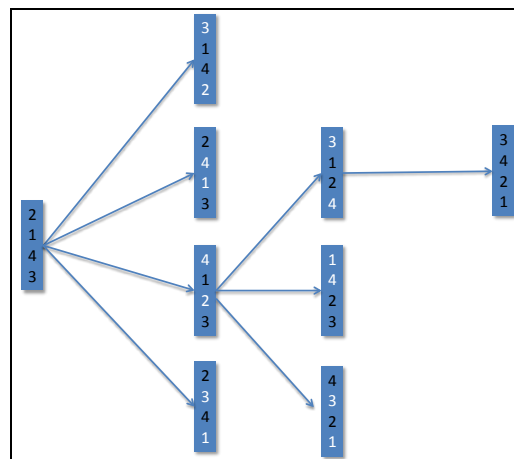


Figure 4. Path relinking.

Note that the PR shown in Figure 4 implements a greedy selection; i.e., at each step in the path from U to V , the selection of the elements to be swapped is made to minimize the objective function value. Other alternatives, not tested in this paper, are the greedy randomized selection and the truncated path relinking (see Resende et al., 2010).

We first consider a straightforward implementation of PR called **Static PR**. In this variant, we first apply the generation procedure (GRASP or IG - SO) to construct the elite set ES and then we apply PR to generate new solutions between all the pairs of solutions in ES. Given two solutions in ES, U and V , we apply path relinking from U to V . If the best solution in the path improves upon the best solution generated, x_{best} , we update it; otherwise we discard it. The algorithm terminates when PR is applied to all the pairs in ES and the best overall solution x_{best} is returned as the output of the algorithm. When we apply PR between two solutions, we also apply the local search method (described in Section 2) to some of the solutions generated in the path (every $prMax$ steps). Note that two consecutive solutions after a *relinking* step only differ in the positions of two vertices and therefore, it does not seem efficient to apply the local search exploration at every step of the relinking process.

An alternative design is given by the **Dynamic PR** in which, after creating the initial ES, each solution U generated with the corresponding method (GRASP or IG - SO) is directly subjected to the PR algorithm, which is applied between U and a solution V selected from ES. The selection is made probabilistically according to the quality of the solutions in ES. The local search method is applied as in the static version, but now, the resulting solution is directly tested for inclusion in ES. If successful, it can be used as the guiding solution in subsequent iterations of PR, which makes a difference with the static PR described above.

Solutions in ES are sorted from the best x_1 to the worst x_b . If a solution generated with PR x is better than x_1 , it is directly admitted to ES. Additionally, if it is better than x_b and “sufficiently different” than the other solutions in ES, it is also admitted to ES. Since solutions are represented as permutations in the LOPCC, we consider the *Manhattan distance* to test the term “sufficiently different”. Specifically, given two solution, $U = (u_1, u_2, \dots, u_n)$ and $V = (v_1, v_2, \dots, v_n)$, their distance is computed as:

$$d(U, V) = \sum_{i=1}^n |u_i - v_i|.$$

Therefore, the distance between a solution and the set of solutions ES,

$$d(U, ES) = \min_{V \in ES} \{d(U, V)\},$$

is considered “sufficiently different” if $d(U, ES) \geq dth$, where the parameter dth is a distance threshold value empirically adjusted. To keep the size of ES constant and equal to b , whenever we add a solution to this set, we remove another one (its closest solution in ES among those worse than it in value).

We finally consider a third design based on PR called **Evolutionary Path Relinking** (Resende et al., 2010). In particular, the solutions obtained with the application of PR are considered to be candidates to enter ES, as in the dynamic design described above, but now PR is again applied to them as long as new solutions enter to ES. In this way, we can say that ES evolves. When none of the new solutions obtained with combinations are admitted to enter the ES, GRASP or IG - SO is then applied again using the same rules to include the generated solutions in the ES. The whole method stops after a specified number of global iterations. We compare in our computational study in Section 5 these three designs, static, dynamic and evolutionary.

5. Computational Experiments

All the metaheuristics described in the previous sections were implemented in Java and experiments were conducted on a MacBook Pro computer with a 2.4 GHz Intel I3 processor and 4 GB of RAM.

5.1 Sets of instances

We have tested the procedure on three sets of instances previously reported (Bertacco et al., 2005; Duarte et al., 2009):

- (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. (2005). 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 (LOLIB, 1997). This is the well-known library for the LOP. We consider the 43 instances of size $n = 50$.
- (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.

5.2 Constructive Methods and Local Search

In our preliminary experimentation, to determine the values of the key search parameters and to compare the different search strategies, we employ the 25 random instances of size 100. In our first experiment we measure the contribution of the local search in the GRASP algorithm described in Section 2 and compare it with the previous constructive heuristic, CH, proposed in Duarte et al. (2011) with and without its associated local search. Table 1 reports the average objective function value, the average percent deviation from the best-known solution, the number of best solutions found (out of 25), and the average CPU time in seconds obtained with these four methods run for 100 iterations. In this experiment we set to 1 the value of the β parameter. In the next experiment we will discuss different values for β . We run the CH heuristic with the parameters' values recommended by their authors.

Table 1. Constructive variants on 25 *Random instances* (size 100)

	GRASP (without LS)	GRASP (with LS)	CH (without LS)	CH (with LS)
Obj. Function	4270886.39	1149.83777	9157900.33	1195.13279
Avg. Deviation	93339.7%	0.9%	176688.2%	5.3%
Num. of Best	0	18	0	7
CPU seconds	0.14	165.35	0.18	179.68

Table 1 shows that the best solution quality is obtained with the GRASP (in the version with the local search), which is able to match 18 best solutions (of this experiment) out of 25 instances. Comparing the constructive methods with and without the local search, the results clearly indicate the advantage of using an improvement method (with a reduction on the average deviation of several orders of magnitude). However, the variants with local search

employ more computational time than the other methods as expected (165.35 and 179.68 seconds on average, compared to less than one second for the other two methods).

To complement the analysis of the results shown in Table 1, we now compare GRASP (with LS) and CH (with LS) with two well-known nonparametric tests for pair-wise comparisons: the Wilcoxon test and the Sign test. The former one answers the question: Do the two samples (solutions obtained with both methods in our case) represent two different populations? The resulting p -value of 0.001 indicates that the values compared come from different methods. Table 2 shows the ranks computation of this test. On the other hand, the Sign test computes the number of instances on which an algorithm supersedes another one. The resulting p -value of 0.001 indicates that the GRASP (with LS) is the clear winner between both methods.

Table 2. Wilcoxon test on 25 *Random instances* (size 100)

	N	Average rank	Sum of ranks
Negative ranks (CH<GRASP)	4	8.50	34.00
Positive ranks (GRASP<CH)	21	13.86	291.00
Ties (CH=GRASP)	0		
Total	25		

Our second preliminary experiment has the goal of finding appropriate values for the critical search parameter of the GRASP procedure. With the same set of 25 instances used in the previous experiment, we compute the average percent deviation from the best-known solution of the solutions obtained with the GRASP method with the parameter β set to 0.0, 0.1, 0.2, ... 0.9 and 1.0. Figure 5 shows the average percentage deviation with respect to the best known solutions obtained with each variant.

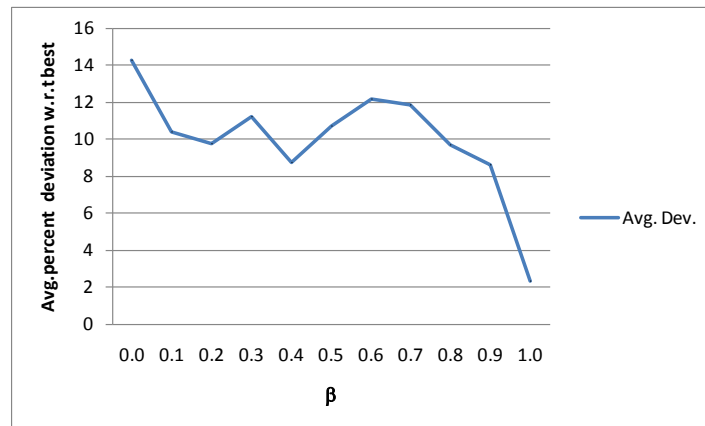


Figure 5. GRASP Avg. deviations.

The results in Figure 5 show that the best outcomes are obtained when the GRASP is run with a value of $\beta = 1.0$. This is an interesting result since this parameter manages the “proportion” of randomization versus greediness in the construction process. Therefore we can conclude that diversity is more important than quality when constructing a solution for the LOPCC (to be submitted to an improvement phase). This behavior is different than that one observed by Martí et al. (2011) in the classical LOP where the best solutions are obtained with a value of 0.4 in the parameter of a proposed GRASP.

5.3 Iterated Greedy – Strategic Oscillation

In our third preliminary experiment we undertake to compare the two iterated greedy-strategic oscillation methods described in Section 3, IG-SO1 and IG-SO2, to study how the k

parameter (percentage of removed elements in the destruction phase) affects their performance, and to compare them with GRASP. Note that with low values of k (10% or 20%) the iterated greedy only removes a small fraction of the elements in the current solution and therefore it creates an intensification pattern since similar solutions are generated, exploring their neighborhood (by means of the associated local search). Alternatively, large values of k (80% or 90%) make the IG-SO to destroy most of the current solution, producing a diversification effect over the long term of the search. Figure 6 shows in a diagram the results of this experiment.

The diagram depicted in Figure 6 shows that the best results are found with IG-SO1 with $k=70\%$ (that exhibits a 4.35% average percent deviation with respect to the best known solution) and lower quality results are obtained with extreme k -values (especially with values lower than 20%). Moreover, if we compute the average percent deviation of GRASP with respect to the best known values in this experiment, we obtain a 6.34%, which indicates the superiority of IG-SO over GRASP. Moreover, the number of best solutions that each method is able to match is 9 for IG-SO1 ($k=70\%$), 9 for IG-SO2 ($k=50\%$) and 7 for GRASP. Note that the best solutions are computed with respect to the best values achieved in each particular experiment (in this way we can better discriminate among methods). This is why we cannot directly compare these values across different tables. As a consequence of the previous experiments we will consider IG-SO1 ($k=70\%$) as our best method to be coupled with PR in the next experiments.

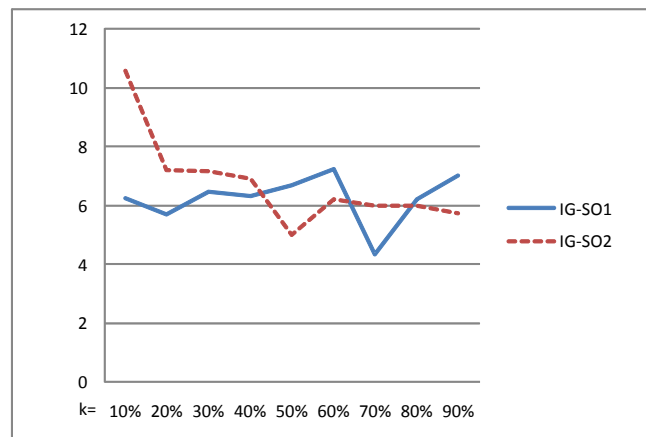


Figure 6. Iterated greedy variants.

5.4 Path Relinking

In our four and final preliminary experiment, we compare the path relinking variants described in Section 4. Table 3 shows the results of the iterated greedy-strategic oscillation by itself, IG-SO1, and coupled with the three PR methods, Static PR, Dynamic PR and EvPR. The parameter dth is empirically adjusted to $0.05 * MaxDistance$ where this maximum distance is computed as:

$$MaxDistance = \sum_{i=1}^n |i - (n - i + 1)|.$$

On the other hand, regarding the $prMax$ parameter controlling the application of the local search to intermediate solutions, experimental results show that as it increases the algorithm is able to marginally improve the quality of the final solution. However, running times also increase since the application of the local search is time-consuming. We therefore set $prMax$ to 10 as a good compromise between solution quality and speed.

Table 3. Path relinking variants on 25 *Random instances* (size 100)

	IG-SO1	Static PR	Dynamic PR	EvPR
Obj. Function	1069.65	1089.82	1039.96	1058.78
Avg. Deviation	3.08%	3.49%	2.04%	1.44%
Num. of Best	5	3	7	11
CPU seconds	296.48	338.04	353.11	351.37

Results in Table 3 report that the best outcomes are obtained with the iterated greedy-strategic oscillation method coupled with evolutionary path relinking, EvPR. This variant is able to match 11 best solutions (in this experiment) out of 25 instances, which compares favorably with the 5, 3 and 7 best solutions obtained with the iterated greedy by itself, and coupled with the static and dynamic variants respectively. The average percentage deviations reported in this table are in line with the best solutions found with each method. Specifically, IG-SO1, Static PR, Dynamic PR and EvPR obtain 3.08%, 3.49%, 2.04% and 1.44% respectively. It must be noted that the static PR variant performs worse than the dynamic one and even worse than the iterated greedy-strategic oscillation by itself.

We applied a statistical test to the data used to generate Table 3. Specifically, we applied the Friedman test for paired samples to the best solutions obtained by each method. This test computes, for each instance, the rank-value of each method according to solution quality (where rank 4 is assigned to the worst method and rank 1 to the best one). Then, it calculates the average rank values of each method across all the instances solved. If the averages are very different, the associated p -value or significance will be small. The resulting significance level of 0.018 obtained in this experiment clearly indicates that there are statistically significant differences among the four methods tested (it is lower than the typical threshold of 0.05). Specifically, the rank values produced by this test are 1.92, 2.32, 2.82 and 2.94 for the EvPR, Dynamic PR, IG-SO1 and Static PR, respectively. This indicates that among the procedures that we tested, EvPR is the best at obtaining solutions with minimum values.

5.5 Comparison with best methods

In our final experiment, we compare our best algorithm, EvPR, with the best previous methods: the GR by Benvenuto et al. (2005), the truncated branch and bound, TB&B, by Righini (2008), and the tabu search, TS, by Duarte et al. (2011). Tables 4 and 5 show the results of these four methods on the 100 UMTS, and the 43 LOLIB instances respectively. Tables 6, 7 and 8 show the results on the Random instances with 35, 100 and 150 nodes respectively.

Table 4. Best methods on 100 *UMTS instances*

	GR	TB&B	TS	EvPR
Obj. Function	79.05	12.92	12.92	12.92
Avg. Deviation	574.20%	0.00%	0.00%	0.00%
Num. of Opt.	0	100	100	100
CPU seconds	3.90	0.21	1.63	1.62

Table 5. Best methods on 43 *LOLIB instances*

	GR	TB&B	TS	EvPR
Obj. Function	5.53E+10	7.24E+10	8.26E+08	1.35
Avg. Deviation	350347.9%	287638.8%	1.40%	0.00%
Num. of Opt.	0	0	28	36
CPU seconds	33.46	30.00	37.74	32.34

Table 6. Best methods on 25 *Random instances* (n=35)

	GR	TB&B	TS	EvPR
Obj. Function	0.91	0.63	0.34	0.34
Avg. Deviation	126.43%	0.45%	0.51%	0.45%
Num. of Opt.	0	0	21	24
CPU seconds	8.84	8.00	3.40	3.75

Table 7. Best methods on 25 *Random instances* (n=100)

	GR	TB&B	TS	EvPR
Obj. Function	288137.55	9614.23	1161.46	1058.78
Avg. Deviation	13001.93%	430.00%	5.79%	0.74%
Num. of Best	0	0	12	13
CPU seconds	511.19	540.00	406.79	351.38

Table 8. Best methods on 25 *Random instances* (n=150)

	GR	TB&B	TS	EvPR
Obj. Function	1.12E+11	1.14E+08	2.27E+06	1.85E+06
Avg. Deviation	993996.3%	1736.7%	9.19%	3.37%
Num. of Best	0	0	7	15
CPU seconds	2249.80	2000.00	2074.31	1127.24

The results in Tables 4 to 8 clearly indicate that the GR method obtains low quality solutions, several orders of magnitude larger than its competing methods. On the other hand, the UMTS instances, reported in Table 4, are not adequate to compare state-of-the-art methods, such as TB&B, TS and EvPR, since all of them are able to obtain the optimal value in all these instances within very short computational times.

Considering the LOLIB (Table 5) and the small Random instances (Table 6) we observe a similar pattern; the truncated branch and bound (TB&B) is not able to match any optimal value, while TS and EvPR present good results. In particular TS obtains 28 and 21 optimal values (out of 43 and 25 instances) and EvPR obtains 36 and 24 respectively. Similarly, the average percentage deviation confirms a slight improvement of EvPR over TS. Specifically, TS presents a 1.4% and 0.51% while EvPR achieves a 0.00% and 0.45% on LOLIB and small random instances respectively.

Considering now the most challenging instances (Random instances with $n=100$ and $n=150$) we can confirm the superiority of EvPR over the other three methods considered and, in particular, over the recent tabu search method (TS). Specifically, TS exhibits a 5.79 and a 9.19 percentage deviation with respect to the best known solutions over the medium and large random instances respectively, while EvPR obtains a remarkable 0.74% and 3.37%. It is important to note that EvPR consumes shorter computational times than TS.

We applied a Friedman test for paired samples to the best solutions obtained by each method in Tables 5 to 8 and the resulting significance level of 0.000 clearly indicates that there are statistically significant differences among the four methods tested. Moreover, the rank values produced by this test are 1.33, 1.67, 3.10 and 3.90 for the EvPR, TS, TB&B and GR respectively. This indicates that among the procedures that we tested, EvPR is the best at obtaining solutions with minimum values. We finally compare EvPR and TS with the Wilcoxon and the Sign tests. The resulting p -value of 0.000 obtained in both tests, together with the previous

results, indicates that the EvPR is the clear winner in the comparison. Table 9 shows the ranks computed in the Wilcoxon test.

Table 9. Wilcoxon test on best methods

	N	Average rank	Sum of ranks
Negative ranks (EvPR<TS)	79	63.01	4978.00
Positive ranks (EvPR>TS)	39	52.38	2043.00
Ties (EvPR=TS)	0		
Total	118		

To complement the analysis above, Figure 7 shows the typical search profile for EvPR and TS. This run corresponds to the set of instances used in our preliminary testing with a time limit of 400 seconds. Tables in the Appendix report the best known values identified in our study.

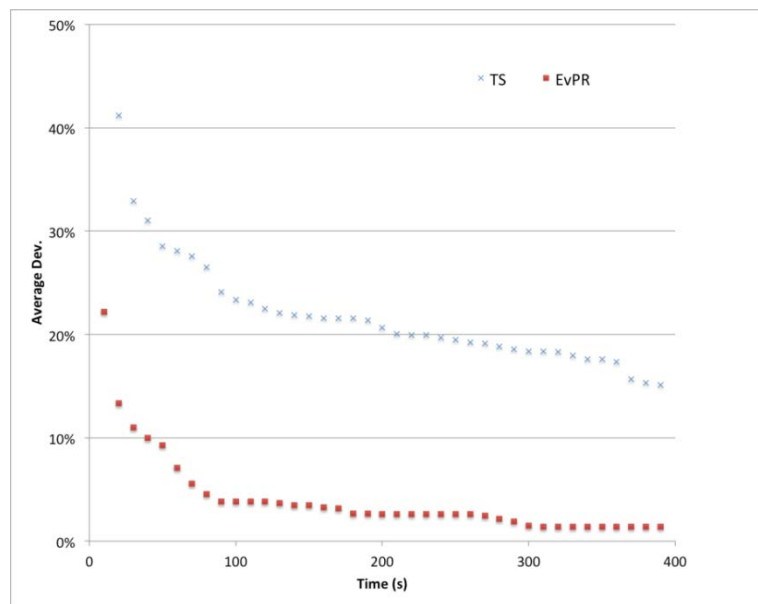


Figure 7. Search profile of best methods.

6. Conclusions

We have described the development and implementation of six different methods for the solution of the linear ordering problem with cumulative costs. Specifically we have proposed a GRASP, two iterated greedy-strategic oscillation and three path relinking algorithms. We arrived to our final design by way of performing a series of preliminary experiments. The final design is then compared to state-of-the-art methods and the outcome of our experiments seems quite conclusive in regard to the merit of the procedure that we propose, which is able to identify 87 new best solutions out of 218 instances. We believe that the performance boost that we achieved by the combination of a memory-less method, such as iterated greedy-strategic oscillation, mainly based on randomization with an adaptive memory procedure, such as path relinking, is a valuable lesson for future implementations.

Acknowledgments

This work is partially supported by the *Ministerio de Educación y Ciencia* under reference code TIN2009-07516. The authors thank Prof. Giovanni Righini for providing them with their truncated branch and bound for the LOPCC.

References

- 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, 2005. The Linear Ordering Problem with Cumulative Costs. *European Journal of Operational Research* 189 (3), 1345-1357.
- Culberson, J. and F. Luo (1996). Exploring the k-colorable landscape with iterated greedy. D.S. Johnson, M.A. Trick, eds. Cliques, coloring, and satisfiability: second DIMACS implementation challenge 26. American Mathematical Society, 245-84.
- Duarte, A., M. Laguna, R. Martí, 2011. Tabu Search for the Linear Ordering Problem with Cumulative Costs. *Computational Optimization and Applications* 48, 697 - 715.
- Feo, T.A., and M.G.C. Resende, 1989. A probabilistic heuristic for a computationally difficult set covering problem. *Operations Research Letters* 8, 67–71.
- Feo, T.A., M.G.C. Resende, and S.H. Smith, 1994. A greedy randomized adaptive search procedure for maximum independent set. *Operations Research* 42, 860–878.
- Glover, F., 1996. Tabu search and adaptive memory programming—Advances, applications and challenges, *Interfaces in Computer Science and Operations Research*, R.S. Barr, R.V. Helgason, and J.L. Kennington (Editors), Kluwer Academic Publishers, Dordrecht, The Netherlands, 1–75.
- Glover, F., and M. Laguna, 1997. *Tabu search*, Kluwer Academic Publishers, Norwell, MA, USA, 1997.
- Jacobs, L.W. and Brusco, M.J., 1995. A local-search heuristic for large set-covering problems. *Naval Research Logistics* 42, 1129-1140.
- 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* 26, 1217-1230.
- LOLIB, 1997. <http://www.iwr.uni-heidelberg.de/groups/comopt/software/LOLIB/>
- Marchiori, E. and A. Steenbeek, 2000. An evolutionary algorithm for large set covering problems with applications to airline crew scheduling. In: Cagnoni, S. et al. (Eds.), *Real-World Applications of Evolutionary Computing*, Lecture Notes in Computer Science, vol. 1803. Springer-Verlag, Berlin, pp. 367–381.
- Martí, R., G. Reinelt and A. Duarte, 2011. A benchmark library and a comparison of heuristic methods for the linear ordering problem. *Computational Optimization and Applications*, Forthcoming.
- 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.
- Resende, M.G.C., and C.C. Ribeiro, 2003. Greedy randomized adaptive search procedures. *Handbook of metaheuristics*, F. Glover and G. Kochenberger (Editors), Kluwer Academic Publishers, Norwell, MA, USA, pp. 219–250.
- Resende, M.G.C., and R.F. Werneck, 2004. A hybrid heuristic for the p-median problem”, *Journal of Heuristics* 10, 59–88.
- Resende, M.G.C., R. Martí, M. Gallego and A. Duarte, 2010. GRASP and Path Relinking for the Max-Min Diversity Problem. *Computers and Operations Research* 37, 498-508.

Resende, M.G.C., and C.C. Ribeiro, 2010. Greedy randomized adaptive search procedures: Advances and applications", *Handbook of metaheuristics*, 2nd edition, M. Gendreau and J.-Y. Potvin (Editors), Springer, pp. 281–317.

Righini, G., 2008. A branch-and-bound algorithm for the linear ordering problem with cumulative costs. *European Journal of Operational Research* 186 (3), 965-971.

Ruiz, R., and T. Stützle 2007. A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem. *European Journal of Operational Research* 177, 2033–2049.

Schiavinotto, T. and T. Stützle, 2004. The linear ordering problem: Instances, search space analysis and algorithms. *Journal of Mathematical Modelling and Algorithms* 3, 367–402.

Appendix – Best Known solutions

Numbers in bold have been identified in this study by our method (IG-SO1+EvPR).

Instance	Value	Instance	Value	Instance	Value
be75eec	5.085	t69r11xx	14.036	t75n11xx	9.897
be75np	16543433.910	t70b11xx	93.671	t75u11xx	0.326
be75oi	2.788	t70d11xn	79.742	tiw56n54	2.645
be75tot	297138.273	t70d11xx	4.435	tiw56n58	3.620
stabu1	13.284	t70f11xx	1.267	tiw56n62	3.024
stabu2	14.029	t70i11xx	121146.029	tiw56n66	2.687
stabu3	9.412	t70k11xx	0.492	tiw56n67	1.877
t59b11xx	76261.813	t70l11xx	798.919	tiw56n72	1.567
t59d11xx	4086.303	t70u11xx	35490330260.185	tiw56r54	2.626
t59n11xx	1618.897	t70x11xx	0.231	tiw56r58	3.602
t65b11xx	28230.444	t74d11xx	4.756	tiw56r66	2.189
t65d11xx	3898.568	t75d11xx	5.059	tiw56r67	1.541
t65i11xx	826108.100	t75e11xx	2062.289	tiw56r72	1.349
t65l11xx	2657.735	t75i11xx	4454.931		
t65w11xx	19.258	t75k11xx	1.323		

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

Instance	Value	Instance	Value	Instance	Value
t1d35.1	0.923	t1d100.1	253.988	t1d150.1	8588.289
t1d35.2	0.167	t1d100.2	288.372	t1d150.2	184853.686
t1d35.3	0.154	t1d100.3	1307.432	t1d150.3	574943.633
t1d35.4	0.196	t1d100.4	7539.979	t1d150.4	75510.287
t1d35.5	1.394	t1d100.5	169.336	t1d150.5	79069.363
t1d35.6	0.200	t1d100.6	395.035	t1d150.6	46829.985
t1d35.7	0.120	t1d100.7	5936.281	t1d150.7	161149.153
t1d35.8	0.226	t1d100.8	2760.619	t1d150.8	251940.422
t1d35.9	0.436	t1d100.9	62.942	t1d150.9	364320.250
t1d35.10	0.205	t1d100.10	162.942	t1d150.10	122217.421
t1d35.11	0.369	t1d100.11	233.586	t1d150.11	13900.039
t1d35.12	0.234	t1d100.12	236.696	t1d150.12	65717.265
t1d35.13	0.196	t1d100.13	593.319	t1d150.13	109460.320
t1d35.14	0.138	t1d100.14	249.162	t1d150.14	74854.867
t1d35.15	1.376	t1d100.15	406.478	t1d150.15	352880.286
t1d35.16	0.286	t1d100.16	707.413	t1d150.16	16950196.691
t1d35.17	0.199	t1d100.17	725.790	t1d150.17	77828.419
t1d35.18	0.381	t1d100.18	622.942	t1d150.18	711286.599
t1d35.19	0.236	t1d100.19	228.486	t1d150.19	67840.414
t1d35.20	0.068	t1d100.20	255.151	t1d150.20	1886041.875
t1d35.21	0.202	t1d100.21	228.590	t1d150.21	41453.911
t1d35.22	0.177	t1d100.22	159.336	t1d150.22	695751.688
t1d35.23	0.345	t1d100.23	1658.168	t1d150.23	22203891.826
t1d35.24	0.132	t1d100.24	469.658	t1d150.24	105162.367
t1d35.25	0.143	t1d100.25	644.782	t1d150.25	462316.511

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