Available online at www.sciencedirect.com
ScienceDirect
European Journal of Operational Research 186 (2008) 965–971
EUROPEAN JOURNAL OF OPERATIONAL RESEARCH
www.elsevier.com/locate/ejor

Discrete Optimization

A branch-and-bound algorithm for the linear ordering problem with cumulative costs

Giovanni Righini

Dipartimento di Tecnologie dell'Informazione, Università degli Studi di Milano, Via Bramante 65, 26013 Crema, Italy

Received 14 July 2006; accepted 15 February 2007
Available online 2 April 2007

Abstract

The linear ordering problem with cumulative costs is an $\mathcal{NP}$-hard combinatorial optimization problem arising from an application in UMTS mobile-phone communication systems. This paper presents a polynomially computable lower bound that is particularly effective when embedded in a branch-and-bound algorithm. The same idea can be further exploited to sort the children nodes at each node of the search tree, in order to find the optimal solution earlier. A suitable truncation of the resulting branch-and-bound algorithm results in a fast constructive heuristic.

Keywords: Branch-and-bound; Combinatorial optimization; Linear ordering problem

1. Introduction

The linear ordering problem (LOP in the remainder) is an $\mathcal{NP}$-hard combinatorial optimization problem that consists in finding an optimal permutation of a given set $\mathcal{N}$ of items; the objective function depends on whether item i precedes or follows item j for each (i, j) pair of items in $\mathcal{N}$. The LOP has many applications, ranging from marketing to psychology, from scheduling to archaeology: a polyhedral study of the LOP is due to (Grötschel et al., 1984, 1985a,b). More references to applications can be found in (Reinelt, 1985). In graph theoretical terms the LOP is defined as follows: a complete digraph $\mathcal{D} = (\mathcal{N}, \mathcal{A})$ is given and the goal is to find an optimal spanning acyclic tournament in $\mathcal{D}$. An acyclic tournament is an acyclic digraph $\mathcal{T} = (\mathcal{N}, \mathcal{A}')$ such that for every two distinct vertices i and j in $\mathcal{N}$ the arc set $\mathcal{A}'$ contains exactly one arc with endpoints i and j. Acyclic tournaments are in one-to-one correspondence with permutations of the vertices; hereafter a generic permutation is indicated by $\Pi : \{1, \ldots, N\} \mapsto \mathcal{N}$, where $N = |\mathcal{N}|$; so, $\Pi(k)$ indicates the vertex placed in position k and $\Pi^{-1}(i)$ indicates the position in which vertex i is placed.

The linear ordering problem with cumulative costs (LOP-CC in the remainder) arises from an application in the sector of UMTS mobile-phone telecommunication systems. The detailed motivation of the LOP-CC was recently described by (Bertacco et al., 2004), who also gave a proof of $\mathcal{NP}$-hardness for the problem. In the LOP-CC each vertex $i \in \mathcal{N}$ has a non-negative weight p_i; for each ordered pair of distinct vertices i and j in $\mathcal{N}$ a non-negative cost c_{ij} is given; in general $c_{ij} \neq c_{ji}$. The

E-mail address: righini@dti.unimi.it

0377-2217/$ - see front matter
doi:10.1016/j.ejor.2007.02.044

objective function to be minimized is the sum of cost terms α associated with the vertices and recursively defined in this way:

$$\alpha_i = p_i + \sum_{j|\Pi^{-1}(j)>\Pi^{-1}(i)} c_{ij}\alpha_j \quad \forall i \in \mathcal{N}.$$

In (Bertacco et al., 2004) the authors proposed a depth-first-search branch-and-bound algorithm which is much faster than a general purpose solver (ILOG CPLEX 9.0.2) and a heuristic algorithm based on dynamic programming.

The main contribution of this paper is a new lower bound, which is used in a branch-and-bound algorithm for the exact optimization of the LOP-CC. A policy for sorting the branches at each node of the search tree is also illustrated: it allows to obtain an effective heuristic from the truncation of the branch-and-bound algorithm.

2. The model

The mathematical programming model of the LOP-CC introduced by (Bertacco et al., 2004) is the following:

$$\begin{aligned}
\min \quad & z = \sum_{i\in\mathcal{N}} \alpha_i, \\
\text{s.t.} \quad & \alpha_i = p_i + \sum_{j=1}^{N} c_{ij}x_{ij}\alpha_j \quad \forall i \in \mathcal{N}, && (1)\\
& x_{ij} + x_{ji} = 1 \quad \forall i,j \in \mathcal{N}, && (2)\\
& x_{ij} + x_{jk} + x_{ki} \leqslant 2 \quad \forall i,j,k \in \mathcal{N}, && (3)\\
& \alpha_i \in \Re_+ \quad \forall i \in \mathcal{N}, \\
& x_{ij} \in \{0,1\} \quad \forall i,j \in \mathcal{N}.
\end{aligned}$$

The objective is the minimization of the overall value of variables α that represent the costs accumulated at the vertices. The value of each term α_i is given by constraints (1) and depends on the cumulative costs α of the vertices that follow i in the acyclic tournament defined by the x variables. Constraints (2) and (3) impose that the values of the x variables define an acyclic tournament. As indicated in Bertacco et al., (2004) this non-linear model can be linearized with the introduction of additional variables; this is useful to solve the LOP-CC with a general-purpose mixed integer linear programming solver.

3. Branch-and-bound

The algorithm devised by (Bertacco et al., 2004) is a depth-first-search branch-and-bound algorithm, in which each vertex is assigned to a position, starting from position N and going back to position 1. In this way all values of α variables are recursively computed, starting from $\alpha_{\Pi(N)}$ down to $\alpha_{\Pi(1)}$. In order not to enumerate all the 2^N possible permutations, the algorithm exploits a lower bounding technique.

Consider the generic node at level l in the search tree, where l vertices have been assigned to the last l positions in the acyclic tournament, while the others are still to be ordered. Call $\mathcal{L}$ the set of the unordered vertices and $\mathcal{R}$ the set of the vertices already ordered. The value of the objective function can be split into two terms:

$$z = \sum_{i\in\mathcal{L}} \alpha_i + \sum_{i\in\mathcal{R}} \alpha_i. \quad (4)$$

The second term in (4) can be computed exactly, since the vertices in $\mathcal{R}$ have already been ordered and hence the value of α is known for them. The first term in (4) can be split into three terms:

$$\sum\nolimits_{i\in\mathcal{L}}\alpha_i = \sum_{i\in\mathcal{L}}\left(\sum_{j\in\mathcal{L}} c_{ij}x_{ij}\alpha_j\right) + \sum_{i\in\mathcal{L}} p_i + \sum_{i\in\mathcal{L}}\sum_{j\in\mathcal{R}} c_{ij}\alpha_j. \quad (5)$$

For each vertex $i \in \mathcal{L}$ the first term in (5) includes the contributions to α_i due to the vertices in $\mathcal{L}$, the second term in (5) represents the fixed cost and the third term in (5) includes the contributions to α_i due to the vertices in $\mathcal{R}$. The third term does not depend on the x variables since all vertices in $\mathcal{R}$ follow all vertices in $\mathcal{L}$ in the acyclic tournament under construction (i.e. $x_{ij} = 1\ \forall i \in \mathcal{L}, j \in \mathcal{R}$). Hence the values of the second and third terms in (5) can be computed exactly. An illustration is given in Fig. 1.

The lower bound used by (Bertacco et al., 2004) is given by all costs that can be computed exactly, that is

$$\mathrm{LB}_0 = \sum_{i\in\mathcal{L}} p_i + \sum_{i\in\mathcal{L}}\sum_{j\in\mathcal{R}} c_{ij}\alpha_j + \sum_{i\in\mathcal{R}} \alpha_i.$$

To prove that $\mathrm{LB}_0 \leqslant z$, observe that $z - \mathrm{LB}_0 = \sum_{i\in\mathcal{L}}\sum_{j\in\mathcal{L}} c_{ij}x_{ij}\alpha_j \geqslant 0$, because all p and c data are non-negative and hence α variables are also non-negative.

The computational experiments reported in (Bertacco et al., 2004) show that this lower bound is effective enough to win the comparison versus CPLEX 9.0.2 solving the linearized version of the mathematical formulation given above. However, this lower bound disregards the unknown term

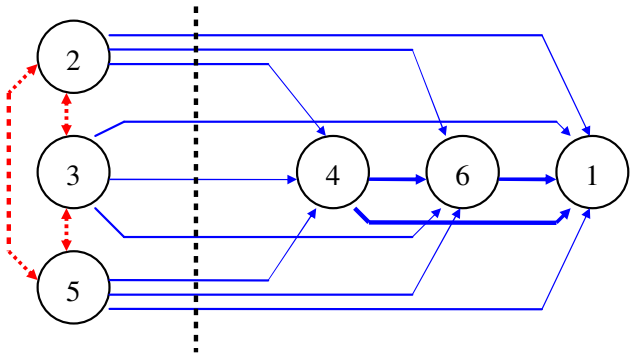

Fig. 1. A small example with six vertices, three fixed and three to be decided. The bolded arcs correspond to precedences between vertices in $\mathscr{R}$, that determine the second term in (4); the arcs crossing the vertical line correspond to precedences between vertices in $\mathscr{L}$ and vertices in $\mathscr{R}$, that determine the third term in (5). The bi-directional vertical arrows on the left correspond to the undecided precedences, that determine the first term in (5).

$H = \sum_{i\in\mathscr{L}}\sum_{j\in\mathscr{L}} c_{ij}x_{ij}\alpha_j$. Therefore, any lower bound to the value of H can be used to strengthen LB_0, thus possibly reducing the size of the search tree and the computing time.

4. A lower bound

For each vertex $i \in \mathscr{L}$ define $\beta_i = p_i + \sum_{j\in\mathscr{R}} c_{ij}\alpha_j$. In any complete solution the final value of α_i is certainly not less than the value of β_i at any intermediate node in the search tree, owing to the non-negativity of the data. Therefore, β_i is a valid lower bound to the final value of α_i. In addition for each pair of distinct vertices i and j in $\mathscr{L}$, we know that in a complete solution either i precedes j or vice versa: therefore, either $c_{ij}\alpha_j$ or $c_{ji}\alpha_i$ will contribute to the overall cost. Hence the following inequalities hold:

$$H \geqslant \sum_{i,j\in\mathscr{L}} \min\{c_{ij}\alpha_j, c_{ji}\alpha_i\} \geqslant \sum_{i,j\in\mathscr{L}} \min\{c_{ij}\beta_j, c_{ji}\beta_i\}.$$

Hence the value

$$LB_1 = LB_0 + \sum_{i,j\in\mathscr{L}} \min\{c_{ij}\beta_j, c_{ji}\beta_i\},$$

which is computable in $O((N-l)^2)$ at each node at level l in the search tree, is a valid lower bound.

5. Computational results

The algorithm of (Bertacco et al., 2004) was re-implemented to test the effectiveness of the lower bounding technique described above. In the re-implementation, the C programming language was used, instead of C++, without any change to the algorithm. All tests reported in the remainder have been done on a 2.0 GHz PC AMD Sempron 3000+ with 512MB RAM, Windows XP operating system and the DevC++ 4.9.9.0 compiler.

The computational tests were done on four datasets, each made of 500 randomly generated instances kindly provided by Livio Bertacco. All instances have size equal to 16 (this is due to technological reasons related to the UMTS application from which the problem arises). Experiments were also made on subgraphs of size $N = 14$ and $N = 12$ by considering only the first N rows and columns of the cost matrices.

Table 1 presents the computational results. For each set of 500 instances and for each size the table reports the average and the maximum computing time and the average and the maximum number of nodes explored in the search tree. These values are reported for the branch-and-bound algorithm with lower bound LB_0 and for the branch-and-bound algorithm with lower bound LB_1.

Although the re-implementation could not exactly reproduce the outcome of (Bertacco et al., 2004), owing to the differences in hardware, operating system, programming language and compiler, the results with lower bound LB_0 scale with the size of the instances in a similar way as those reported in the above mentioned paper. The comparison shows that LB_1 allows to reduce the computing time of one order of magnitude and the number of nodes evaluated by two to three orders of magnitude.

The two lower bounds were also compared by computing their values at the root node for all the problem instances. The outcome is reported in Table 2: each column reports the average value of the gap between the optimal value z^* and the lower bound LB computed at the root node ($\frac{z^*-LB}{LB}$). From the results reported in Table 2 it is clear that LB_1 is significantly tighter on all classes of instances and for any size.

6. Branch sorting policy

This section illustrates another idea concerning the order in which children nodes are explored at each node in the search tree.

In Bertacco et al. (2004) the authors described a depth-first-search policy that allows an effective recursive implementation of the code: the vertices are initially sorted according to their fixed cost p_i

Table 1
Comparison between branch-and-bound algorithms using lower bounds LB_0 and LB_1

Set	Size	Avg. time		Max. time		Avg. no. of nodes		Max. no. of nodes	
		LB_0	LB_1	LB_0	LB_1	LB_0	LB_1	LB_0	LB_1
A	12	0.03	0.00	1.00	0.00	138,249.65	1815.98	3,419,066	46,409
	14	0.50	0.01	31.00	1.00	2,745,671.10	11,862.01	166,581,704	529,724
	16	12.58	0.14	973.00	5.00	60,974,332.11	100,422.75	4,729,842,076	4,963,008
B	12	0.05	0.01	2.00	1.00	270,668.24	6806.23	12,487,041	260,226
	14	1.30	0.07	105.00	3.00	7,037,065.99	19,438.65	562,082,006	4,077,536
	16	38.85	1.50	2022.00	178.00	188,009,341.31	137,804.72	9,804,359,196	204,379,303
C	12	0.02	0.00	1.00	0.00	151,073.13	5005.30	3,820,123	111,448
	14	0.53	0.06	29.00	3.00	2,890,538.03	52,734.75	160,998,680	327,3061
	16	14.94	1.00	1069.00	44.00	72,449,144.17	740,194.33	5,178,947,760	37,629,119
D	12	0.03	0.00	1.00	1.00	170,845.18	5907.80	3,338,684	140,596
	14	0.64	0.07	16.00	2.00	3,488,891.31	67,846.74	89,150,125	1,930,878
	16	16.78	1.20	542.00	33.00	81,387,534.69	861,567.37	2,623,680,651	26,285,589
Avg.	12	0.03	0.00	1.25	0.05	148,734.07	4883.83	5,766,228.50	139,669.75
	14	0.74	0.05	45.50	2.25	4,040,541.61	37,970.54	244,703,128.75	2,452,799.75
	16	20.79	0.96	1151.50	65.00	100,705,088.07	459,997.29	5,584,207,420.75	68,314,254.75

Table 2
Average percentage gaps at the root node

Data-set	Size	LB_0 (%)	LB_1 (%)
A	12	44.64	15.23
	14	51.39	20.42
	16	57.22	25.75
B	12	69.73	37.76
	14	76.39	47.09
	16	82.30	56.49
C	12	69.67	40.54
	14	77.03	50.62
	16	83.09	60.24
D	12	73.90	45.23
	14	81.02	56.27
	16	86.52	65.92
Avg.	12	64.49	34.69
	14	71.46	43.60
	16	77.28	52.10

and this sequence determines the order in which children nodes are visited: the unordered vertices (those in $\mathscr{L}$) are kept in a FIFO queue and each step down in the search tree corresponds to a *Pop* operation, while each backtrack step corresponds to a *Push* operation on the queue. In this way the first leaf that the algorithm examines corresponds to the solution in which the vertex with the largest fixed cost is in the last position (i.e., it is fixed first) and the one with the smallest fixed cost is in the first position (i.e., it is fixed last). However, the leaves of the search tree are not visited according to the lexicographic order defined by the initial sequence: for instance, if the starting sequence is $\langle 1,2,3,4\rangle$, with $p_1 < p_2 < p_3 < p_4$, the leaves of the search tree are generated in this order: first $\langle 1,2,3,4\rangle$, then $\langle 2,1,3,4\rangle$ and now $\langle 3,1,2,4\rangle$ precedes $\langle 1,3,2,4\rangle$ even if $p_1 < p_3$.

Even if this technique is very effective from a software implementation viewpoint, a suitable sorting of the children nodes, repeated at each node of the search tree, may allow the algorithm to discover the optimal solution earlier.

To have a quantitative figure of merit of the effectiveness of the branching policy and in particular of the ordering in which the children nodes are explored, it is necessary to measure "how early" the optimal solution is discovered. For this purpose let us consider a generic instance of the LOP-CC and the corresponding search tree as it is explored by a depth-first-search branch-and-bound algorithm and let us indicate by P^* the optimal path to be followed along the search tree to reach the first leaf corresponding to an optimal solution. Let us define a vector b_l with one component for each level l of the search tree. Each component b_l may range in $[0,\ldots,N-l-1]$, that is it may take on a number of different values equal to the number of children nodes at that level. The value of b_l indicates the number of "wrong" branches that have been explored from the node of P^* at level l before the branch that leads to the optimal solution. A vector

$b = (0, 0, \ldots, 0)$ represents the ideal case, in which the algorithm finds the optimal solution as the first leaf of the search tree. Dividing b_l by the maximum number of children nodes at level l, we get a fraction $\eta_l = \frac{b_l}{N-l-1}$, whose value is between 0 and 1, representing the effectiveness of the branch sorting policy at level l.

We measured the average values of the components of vector η over all the instances of the benchmark, and we observed rather high values, often greater than 0.5. This motivated the design of a different policy to sort the branches, inspired by lower bound LB_1.

As before we indicate with $\mathscr{L}$ the set of vertices still to be ordered in a node at level l in the search tree. For each pair of vertices $i, j \in \mathscr{L}$, a choice is made between arcs (i, j) and (j, i), looking at the minimum among the two costs $c_{ij}\beta_j$ and $c_{ji}\beta_i$. If $c_{ij}\beta_j < c_{ji}\beta_i$ we orient the arc from i to j or vice versa. Depending on these decisions the in-degree of each vertex is determined. Then all the vertices in $\mathscr{L}$ are sorted by increasing values of their in-degree so that the one with maximum in-degree is the last one in the ordering and then it is fixed first.

Table 3 reports the average values of η_l over the 500 instances of the first data-set with $N = 16$, using the policy of (Bertacco et al., 2004), with the FIFO queue, and using the policy in which the branches are sorted at each node of the search tree as explained above. In the bottom lines of the table the overall results of the branch-and-bound algorithm are also reported for the two cases. Even if the branch sorting policy does not yield any advantage in the overall performances of the branch-and-bound algorithm, it significantly reduces the average values of η: this means that it allows the algorithm to find an optimal solution earlier. This suggests a criterion to truncate the exploration of the search tree, as shown in the next section.

Table 3
Effects of branch sorting

Level	FIFO queue η	Sorted branches η
1	0.034	0.034
2	0.267	0.045
3	0.375	0.049
4	0.370	0.046
5	0.408	0.040
6	0.373	0.037
7	0.379	0.025
8	0.330	0.027
9	0.313	0.029
10	0.321	0.031
11	0.317	0.019
12	0.254	0.020
13	0.242	0.017
14	0.179	0.010
15	0.180	0.000
16	0.000	0.000
Avg. no. nodes	100,422.75	96,265.056
Max. no. nodes	4,963,008	5,177,634
Avg. time	0.134	0.122
Max. time	4	5

7. Truncated branch-and-bound

Time is very critical for the UMTS application from which the problem arises. Therefore, it is important to develop very fast heuristics still providing good solutions. The two terms of comparison available in the literature are a paper by (Benvenuto et al., 2005), who developed a GRASP heuristic, and the paper by (Bertacco et al., 2004), who presented a heuristic algorithm based on dynamic programming and proved it over-performs the aforementioned GRASP.

Remarkably the computing time of the exact branch-and-bound algorithm with lower bound LB_1 is comparable with that reported for the dynamic programming heuristic algorithm presented in (Bertacco et al., 2004). However, better results can be obtained by truncating the branch-and-bound in order to reduce the computing time at the expense of the optimality guarantee. For this purpose the fraction of branches to be explored at each level of the search tree is limited by a given threshold $0 \leqslant \theta \leqslant 1$: at each node of the search tree the children nodes are sorted as explained in the previous section and the algorithm explores only the first $\lfloor \theta(N - l - 1) \rfloor + 1$ children nodes. Setting $\theta = 1$ the whole search tree is (implicitly) explored and the best solution found is guaranteed to be optimal. Setting $\theta = 0$ only the first branch is explored at each node and the search tree reduces to a path from the root to the first leaf. Choosing intermediate values one can tune the trade-off between speed and solution cost.

Table 4 shows the average computing time, the maximum computing time, the average percentage gap, the maximum percentage gap and the number of optimal solutions found for the whole data-set of 2000 instances, for five different values of θ including the two limit cases $\theta = 0$ and $\theta = 1$.

These results show that for $\theta = 0.25$ the truncated branch-and-bound algorithm finds optimal

Table 4
Truncated branch-and-bound

Size	θ	Avg. time	Max. time	Avg. gap (%)	Max. gap (%)	Opt. sol.
12	0.00	0.000	0.000	8.764	147.793	232
	0.25	0.000	0.000	1.018	37.446	1303
	0.50	0.002	1.000	0.053	7.627	1913
	0.75	0.002	1.000	0.007	3.306	1991
	1.00	0.005	1.000	0.000	0.000	2000
14	0.00	0.001	1.000	12.539	140.787	95
	0.25	0.003	1.000	1.303	37.770	1155
	0.50	0.012	1.000	0.097	10.062	1903
	0.75	0.025	1.000	0.011	2.978	1990
	1.00	0.053	3.000	0.000	0.000	2000
16	0.00	0.000	0.000	20.355	369.291	34
	0.25	0.014	1.000	1.697	37.544	947
	0.50	0.129	14.000	0.378	10.387	1846
	0.75	0.358	56.000	0.005	1.897	1990
	1.00	0.903	173.000	0.000	0.000	2000

solutions for about half of the instances and for $\theta = 0.50$ it finds optimal solutions for about 95% of the instances. The correspondent reduction in computing time can be very significant: up to a factor of 3 for $\theta = 0.75$ and a factor of 10 for $\theta = 0.50$.

8. Large instances

To exploit the ideas explained above the algorithms were also tested on larger instances with the purpose of studying which is the maximum size for which an approach based on truncated branch-and-bound is viable. This was done in the same spirit of an analogous investigation reported by (Bertacco et al., 2004); it must not be viewed as a trial to present an accurate heuristic for large instances of the LOP-CC: for this purpose local search algorithms are obviously more appropriate.

The benchmark used is the same of (Bertacco et al., 2004) and it consists of sub-matrices of different size taken from a randomly generated (32×32) cost matrix. (Bertacco et al., 2004) reported to have solved these instances with up to $N = 20$ in 3 hours. The leftmost part of Table 5 reports the computing time (hh:mm:ss) needed by the branch-and-bound algorithm with lower bound LB_1 to solve the instances up to $N = 26$ within one hour and a half. A trial with $N = 28$ was stopped after 24 hours. Heuristic values computed with the truncated branch-and-bound with $\theta = 0.5$ and $\theta = 0.25$ are also reported in the rightmost part of the table.

Table 5
Results obtained on larger instances

Size	Exact		Truncated		
	Time	Cost	Time	Cost	θ
18	0:00:02	5.636060	0:00:01	5.636060	0.50
20	0:01:31	8.115764	0:00:38	8.115764	0.50
22	0:03:10	8.178600	0:01:17	8.178600	0.50
24	0:16:59	11.962475	0:06:17	11.962475	0.50
26	1:37:33	15.208841	0:02:31	15.208841	0.25
28	>24:00:00		0:53:54	17.428847	0.25

The results show that the truncated branch-and-bound algorithm was able to find the optimal solution for all instances for which the optimum is known with a reduction in computing time of a factor of about 3 with $\theta = 0.5$ and about 40 with $\theta = 0.25$.

It should be remarked that the computing time of the truncated branch-and-bound approach tends to explode when the size of the instance grows. Therefore, for large instances, like those of the LOLIB, one has to choose whether to develop fast or accurate heuristics.

If enough computing time is available, it is possible to develop accurate heuristics, based on local search. For instance, while this paper was under review, a tabu search algorithm for the LOP-CC was presented by (Duarte et al., 2006). After a careful calibration of a number of parameters, it could compute within a reasonable time heuristic solutions for instances larger than those considered here above, some taken from the LOLIB library and others generated at random with up to 150 vertices.

On the other hand, if computing time is a very scarce resource, as in the case of the telecommunication systems that gave origin to the papers by (Benvenuto et al., 2005) and (Bertacco et al., 2004), fast constructive heuristics are required. The truncated branch-and-bound illustrated above can be easily turned into a fast constructive heuristic by setting $\theta = 0$. Table 6 reports the results obtained in this way (column "Constr. heur.") on the random instances of (Duarte et al., 2006) (available on-line

Table 6
Comparison of heuristics on random large instances

Size	GRASP		Constr. heur.	TSLOPCC	
	Cost	Time	Cost	Cost	Time
35	0.935	1.68	0.588	0.344	0.39
100	8.48E+05	24.75	1.43E+04	1.20E+03	30.75
150	2.90E+11	78.31	8.84E+07	2.25E+06	180.43

as reported in their paper), compared with the results of the GRASP heuristic of (Benvenuto et al., 2005) (column "GRASP") and those of the tabu search of (Duarte et al., 2006) (column "TSLOPCC"). The values in both these columns are taken from (Duarte et al., 2006). The computing time of our constructive heuristic is not reported, because all data-sets, made of 25 instances each, were solved in less than one second on a 2.0 GHz processor; computing times for GRASP and TSLOPCC are taken from (Duarte et al., 2006) and were measured on a 3.2 GHz processor.

From the results reported it is clear that the constructive heuristic obtained from the truncated branch-and-bound algorithm dominates the GRASP in both time and cost. No domination exists between truncated branch-and-bound and tabu search, since the former is faster and the latter is more accurate.

Acknowledgements

The author is grateful to Matteo Fischetti, whose presentation at AIRO Winter 2005 workshop inspired this work, to Livio Bertacco for kindly providing his computer code and the input data, and to three anonymous referees for their helpful comments.

References

Benvenuto, N., Carnevale, G., Tomasin, S., 2005. Optimum power control and ordering in SIC receivers for uplink CDMA systems. IEEE International Conference on Communications, ICC 2005 4, 2333–2337.

Bertacco, L., Brunetta, L., Fischetti, M., 2004. The linear ordering problem with cumulative costs. Technical report, Department of Information Engineering, University of Padova, October 13, 2004.

Duarte, A., Laguna, M., Martì R., 2006. Tabu search for the linear ordering problem with cumulative costs. Technical report, Universidad de Valencia.

Grötschel, M., Jünger, M., Reinelt, G., 1984. A cutting plane algorithm for the linear ordering problem. Operations Research 32, 1195–1220.

Grötschel, M., Jünger, M., Reinelt, G., 1985a. On the acyclic subgraph polytope. Mathematical Programming 33, 28–42.

Grötschel, M., Jünger, M., Reinelt, G., 1985b. Facets of the linear ordering polytope. Mathematical Programming 33, 43–60.

LOLIB. Linear Ordering LIBrary. <http://www.iwr.uni-heidelberg.de/groups/comopt/software/LOLIB/>.

Reinelt, G., 1985. The linear ordering problem: algorithms and applicationsResearch and Exposition in Mathematics, vol. 8. Heldermann Verlag, Berlin.