Comput Optim Appl (2012) 51:1297–1317
DOI 10.1007/s10589-010-9384-9

A benchmark library and a comparison of heuristic methods for the linear ordering problem

Rafael Martí · Gerhard Reinelt · Abraham Duarte

Received: 11 September 2009 / Published online: 5 January 2011
© Springer Science+Business Media, LLC 2010

Abstract The linear ordering problem consists of finding an acyclic tournament in a complete weighted digraph of maximum weight. It is one of the classical NP-hard combinatorial optimization problems. This paper surveys a collection of heuristics and metaheuristic algorithms for finding near-optimal solutions and reports about extensive computational experiments with them. We also present the new benchmark library `LOLIB` which includes all instances previously used for this problem as well as new ones.

Keywords Metaheuristics · Empirical comparison · Library

1 Introduction

Let $D_n = (V_n, A_n)$ denote the complete digraph on n nodes, where V_n is the set of nodes and A_n the set of arcs. A *tournament* T in A_n consists of a subset of arcs containing for every pair of nodes i and j either arc (i, j) or arc (j, i), but not

Electronic supplementary material The online version of this article (doi:10.1007/s10589-010-9384-9) contains supplementary material, which is available to authorized users.

R. Martí (✉)
Dept. de Estadística e Investigación Operativa, University of Valencia, Valencia, Spain
e-mail: rafael.marti@uv.es

G. Reinelt
Institute for Computer Science, University of Heidelberg, Heidelberg, Germany
e-mail: gerhard.reinelt@informatik.uni-heidelberg.de

A. Duarte
Dept. Ciencias de la Computación, University Rey Juan Carlos, Madrid, Spain
e-mail: abraham.duarte@urjc.es

both. T is an *acyclic tournament* if it does not contain any directed cycle. Obviously, an acyclic tournament induces an ordering $\langle v_{i_1}, v_{i_2}, \ldots, v_{i_n}\rangle$ of the nodes (and vice versa): node v_{i_1} is the one with no entering arcs in T, v_{i_2} has exactly one entering arc, etc., and v_{i_n} is the node with no outgoing arc. Given arc weights w_{ij} for every pair $i, j \in V_n$, the *linear ordering problem* (LOP) consists of finding an acyclic tournament T in A_n such that $\sum_{(i,j)\in T} w_{ij}$ is maximal, or in other words, of finding an ordering of the nodes such that the sum of the weights of the arcs compatible with this ordering is maximal.

Alternatively, the problem can be defined as a matrix problem. Given an (n, n) matrix $C = (c_{ij})$ the *triangulation problem* is to determine a simultaneous permutation of the rows and columns of C such that the sum of superdiagonal entries becomes as large as possible (or equivalently, the sum of subdiagonal entries is as small as possible). Note, that it does not matter if diagonal entries are taken into account or not. Obviously, by setting arc weights $w_{ij} = c_{ij}$ for the complete digraph D_n, the triangulation problem for C can be solved as linear ordering problem in D_n. Conversely, a linear ordering problem for D_n can be transformed to a triangulation problem for an (n, n)-matrix C by setting $c_{ij} = w_{ij}$ and the diagonal entries $c_{ii} = 0$ (or to arbitrary values).

The LOP can also be seen as a problem of ranking objects. Suppose there are n objects which are to be ranked where there is a benefit c_{ij} if object i is ranked before object j. The task of finding a linear ranking maximizing the sum of benefits w.r.t. this ranking amounts to solving a LOP or triangulation problem.

In the following we do not distinguish between the graph and the matrix or the ranking problem and denote the objective function coefficients by c_{ij}.

Note, that if a constant is added to both entries c_{ij} and c_{ji} or if we take diagonal entries into account, the optimality of an ordering is not affected by this transformation. However, the quality of bounds does change. If we add large constants, then every feasible solution is close to optimal and no real comparison of qualities is possible (which is very relevant when we are comparing heuristics). Therefore we transform every problem matrix C to its *normal form* satisfying the following conditions:

i. All entries of C are integral and nonnegative,
ii. $c_{ii} = 0$, for all $i = 1, \ldots, n$,
iii. $\min\{c_{ij}, c_{ji}\} = 0$, for all $1 \le i < j \le n$.

Note that this normal form is unique.

The LOP can be formulated as a 0/1 linear integer programming (IP) problem as follows. We use 0/1 variables x_{ij}, for $(i, j) \in A_n$, stating whether arc (i, j) is present in the tournament or not. Taking into account that a tournament is acyclic if and only if it does not contain any dicycle of length 3, it is easily seen that the LOP can be formulated as the 0/1-IP

$$\max \sum_{(i,j)\in A_n} c_{ij}x_{ij}$$

$$\begin{aligned} x_{ij} + x_{ji} &= 1, \quad \text{for all } i, j \in V_n,\ i < j,\\ x_{ij} + x_{jk} + x_{ki} &\le 2, \quad \text{for all } i, j, k \in V_n,\ i < j,\ i < k,\ j \ne k,\\ x_{ij} &\in \{0, 1\}, \quad \text{for all } i, j \in V_n. \end{aligned}$$

This IP is the basis for approaches to solve the LOP to optimality. If we replace the constraints "$x_{ij} \in \{0, 1\}$" by "$0 \le x_{ij} \le 1$" then we obtain a linear programming problem, whose optimum objective function value provides an upper bound on the optimum value of the LOP. In cases where we do not know optimum solutions to the benchmark problems, we will usually take this upper bound for assessing the quality of heuristics. We will not address the computation of optimum solutions in this paper.

The LOP has been the subject of study for a long time. Applications mentioned in the literature are, for instance, aggregation of individual preferences [11], triangulation of input-output tables [8, 26], determination of ancestry relationships [15], scheduling with preferences [4], assessment of corruption perception [1] and minimizing crossing numbers in graph drawing [21].

This paper is organized as follows. In Sect. 2 we describe simple heuristic approaches for the LOP. Section 3 discusses metaheuristics summarizing the most relevant work in approximate optimization for the LOP. The benchmark problems are described in Sect. 4. In Sect. 5 we report extensive experimental results targeting both short and medium computation time. Some final remarks and online Appendix with the best known solution values conclude the paper.

2 Heuristics

In this section we review construction and improvement heuristics. *Construction methods* obtain a solution, i.e., a linear ordering from scratch adding iteratively one node at each step. *Improvement heuristics*, also called *local search* or *ascent methods* (in the case of maximization problems) start from the solution obtained with a construction method and iteratively improve it by performing local changes usually referred to as *moves*. The different types of moves and selection strategies characterize the various heuristics. In this section we consider the two typical moves: *insertion* and *exchange*.

2.1 Construction method of Chenery and Watanabe

The earliest heuristic for the LOP was introduced in 1958 by Chenery and Watanabe [8]. It concerns an application in economy and it is a simple method to rank the sectors of input-output tables. Sectors having a large share of outputs to other sectors should be ranked first. We can view this method as a greedy algorithm in which the attractiveness a_i of sector i is the sum of the elements in its corresponding row, i.e.,

$$a_i = \sum_{j=1}^{n} c_{ij}.$$

The method (CW) successively constructs an ordering by selecting in each step the most attractive sector among the sectors not ranked so far.

2.2 Construction methods of Aujac and Masson

Aujac and Masson [2] also rank sectors based on coefficients. The *output coefficient* b_{ij} of a sector i with respect to another sector j is defined as

$$b_{ij} = \frac{c_{ij}}{\sum_{k \neq i} c_{ik}}.$$

The first method (AM-O) intends to rank sector i before sector j whenever $b_{ij} > b_{ji}$. Since this is impossible in general, it heuristically tries to find a linear ordering with few contradictions to this principle. Similarly, the second method (AM-I) defines *input coefficients*, where c_{ij} is divided by the sum of the entries in the corresponding column, and ranks sectors according to them.

2.3 Construction methods of Becker

Becker [3] proposed a construction method (Bci) related to the previous ones in that it calculates special quotients to rank the sectors. For each sector i the number

$$q_i = \frac{\sum_{k \neq i} c_{ik}}{\sum_{k \neq i} c_{ki}}$$

is computed. The sector with the largest quotient q_i is then ranked highest. Its corresponding rows and columns are deleted from the matrix, and the procedure is applied to the remaining sectors.

Becker also proposed a second construction method (Bcr) based on rotations, which seems a local search method. Starting from a random ordering, at each iteration this method tries to improve the current ordering $O = \langle i_1, i_2, \ldots, i_n \rangle$ by evaluating all orderings $\langle i_{m+1}, i_{m+2}, \ldots, i_n, 1, 2, \ldots, i_m \rangle$, for $m = 1, 2, \ldots, n-1$. If the best one among them improves O, the method takes it as the new current ordering and continues, otherwise it stops.

2.4 Construction methods based on insertions

We have included two further constructive methods, BI1 and BI2, based on insertions. They basically select an arbitrary unassigned object and insert it into the partial solution at the currently best possible position. Let $\langle i_1, \ldots, i_k \rangle$ be the current partial ordering. BI1 computes for every object l not ranked so far coefficients

$$q_t = \sum_{j=1}^{t-1} c_{i_j l} + \sum_{j=t}^{k} c_{l i_j},$$

for $1 \leq t \leq k$, and inserts l at the position with maximum coefficient. Alternatively, BI2 uses coefficients

$$q_t = \sum_{j=1}^{t-1} c_{i_j l} + \sum_{j=t}^{k} c_{l i_j} - \sum_{j=1}^{t-1} c_{l i_j} - \sum_{j=t}^{k} c_{i_j l}.$$

2.5 Improvement methods based on insertions

Removing a node from the current ordering and inserting it at a different position is a simple possibility for searching for an improvement. Laguna et al. [24] define

$move(O_j, i)$ as the modification which deletes O_j from its current position j in permutation O and inserts it at position i (i.e., between the objects currently in positions $i-1$ and i). The *move value* is the difference between the objective function values after and before the move. The method `LSi` scans the list of nodes (in the order given by the current permutation) in search for the first node whose movement results in an strictly positive move value.

A different approach based on insertions is the method known as k-opt, which basically selects k elements of a solution and locally optimizes with respect to these elements (i.e., considers all subsets of k objects $O_{i_1}, \dots, O_{i_k}$ in the current permutation and finds the best assignment of these objects to the positions $i_1, \dots, i_k$). Since the number of possible new assignments grows exponentially with k, we have only implemented 2-opt and 3-opt.

2.6 Improvement based on exchanges

This heuristic checks whether the objective function can be improved if the positions of two objects in the current ordering are exchanged. All such possibilities are checked and the method stops when no further improvement is possible this way.

In [24] a limited version of this improvement method (`LSe`) is considered. The authors tested a method in which only contiguous sectors are considered for exchange (swap) and concluded that better solutions can be obtained with general insertions. However, as far as we know, general exchanges (between any pair of sectors) have never been tested. We will consider them in our computational comparison.

2.7 Kernighan and Lin improvement

Kernighan and Lin [22] introduced compound moves as a series of simple moves. In contrast to pure improvement heuristics, they allow that some of the simple moves (such as insertions or exchanges) are not improving. However, it is required that the composition produces an improvement. Algorithm `KL1` considers sequences of up to n exchange moves. Each of the simple moves is the best available one, but it can be a non-improving. If the composition of k moves, for some $1 \leq k \leq n$, results in an improvement, then it is performed. Otherwise the moves are discarded and the original solution is restored. Variant `KL2` works in the same way composing sequences of insertion moves.

2.8 Local enumeration

This heuristic chooses windows $\langle i_k, i_{k+1}, \dots, i_{k+L-1} \rangle$ of a given length L of the current ordering $\langle i_1, i_2, \dots, i_n \rangle$ and determines the optimum subsequence of the respective objects by enumerating all possible orderings. The window is moved along the complete sequence until no more improvements can be found. We implemented this method, denoted `LE`, with length $L = 8$.

3 Metaheuristics

In recent years, a series of methods have appeared under the name *metaheuristics*, a term coined by Glover [14] in 1986. Basically, we consider a metaheuristic as combination of simple heuristics with some scheme of randomization and additional features, which can be interpreted as learning mechanism and systematic exploration of search spaces. The following approaches fall into this category. We give only short descriptions of these procedures here because they are discussed in depth in the recent book by Martí and Reinelt [27].

3.1 KLM—Kernighan and Lin multi-start method

Multi-start procedures exploit a local or neighborhood search procedure by applying it from multiple random initial solutions. It is well known that search methods based on local optimization that aspire to find global optima usually require some type of diversification to overcome local optimality. Without diversification, such methods can reduce to tracing paths that are confined to a small area of the solution space, making it almost impossible to find a global optimum. We apply the Kernighan and Lin method from different initial random solutions until a pre-specified time limit is reached.

3.2 CK—Chanas and Kobilansky multi-start method

Chanas and Kobylanski [8] proposed a multi-start method, called CK, based on the following symmetry property. If the permutation $O = \langle O_1, O_2, \dots, O_n \rangle$ is an optimum solution to the maximization problem, then an optimum solution to the minimization problem is $O^* = \langle O_n, O_{n-1}, \dots, O_1 \rangle$. The method utilizes this property to escape local optimality: once a local optimum solution O is found, the process is re-started from the permutation O^* (REVERSE operation).

In a global iteration, CK performs insertions as long as the solution improves. Given a solution, the algorithm explores $move(O_j, i)$ for all O_j and i in O, and performs the best one. When no further improvement is possible, it generates a new solution by applying the REVERSE operation from the last solution obtained, and performs a new global iteration. The method halts when the best solution found cannot be improved further in the current global iteration.

3.3 GRASP—greedy randomized adaptive search procedure

Each GRASP iteration [12] consists of constructing a trial solution and then applying an improvement procedure to find a local optimum (i.e., the final solution for that iteration). In [5] we can find several GRASP algorithms embedded in a scatter search procedure for the LOP. The best one combines a randomized adaptive construction based on the greedy evaluation e, with a local search based on a best insertion (i.e., move to the best solution in the neighborhood defined by insertion moves). The evaluation $e(i)$ of object i is the sum of the elements in its corresponding matrix row:

$$e(i) = \sum_{j=1}^{n} c_{ij}.$$

3.4 TS—tabu search

Tabu search [16] is a metaheuristic that guides a local search procedure to explore the solution space beyond local optimality by allowing non-improving moves in the local search. The basic tabu search algorithm proposed in [24] implements a short-term memory structure alternating two phases: intensification and diversification. It is based on insertions as the improvement phase of the GRASP algorithm described above. However, instead of scanning the objects in search for a move in their original order, they are randomly selected in the intensification phase based on a measure of influence.

The basic method is complemented with a long-term intensification based on the path relinking methodology [25], and with a long-term diversification based on the REVERSE operation proposed in `CK`. Both long-term strategies incorporate frequency information (memory structures) recorded during the application of the short-term phase.

3.5 SS—scatter search

Scatter search is an evolutionary or population based method [25] that operates on a relatively small set of solutions, called reference set, combining them to obtain new and hopefully better solutions. In [5] an SS algorithm is described in which solutions are generated with a diversification method, combined with a min-max method based on a voting scheme, and improved with the local search method employed in the GRASP method referenced above.

Some preliminary tests [5] disclose the best strategies to implement these three methods. We can highlight the use of a frequency-based procedure as the diversification generator method and the combination of multiple solutions.

3.6 VNS—variable neighborhood search

Variable neighborhood search [20] is based on a simple and effective idea: a systematic change of the neighborhood within a local search algorithm. In [13] a VNS algorithm is proposed based on neighborhoods $\mathcal{N}_k$, $k = 1, \dots, n$, where the neighborhood of the solution p, $\mathcal{N}_k(p)$, is the set of solutions that are obtained when we apply $k - 1$ insertion moves to p.

The VNS algorithm applies three steps: *shaking*, in which the current solution is perturbed, *improving*, in which a local optimum with respect to the current neighborhood is obtained, and *updating*, in which a change in the neighborhood is performed. The authors proposed in [13] a hybrid algorithm in which VNS is combined with memory structures for improved outcomes.

3.7 GA and MA—genetic and memetic algorithms

In [31] a genetic algorithm coupled with a local search procedure, called *memetic algorithm*, is developed. As in SS above, it basically consists of generating, improving, and combining solutions. In the initialization, a population of individuals is obtained by first generating a set of random permutations (solutions) and then applying a local search procedure, based on insertions, to each of them.

In each iteration of the algorithm, called *generation*, new solutions are generated by applying crossover and mutation to randomly selected solutions in the population (according to a uniform distribution). Local search is applied again to improve each new solution. The new population is created by merging the best solutions in the population and the new improved solutions. It is worth mentioning that the authors consider four different crossover operators: DPX (similar distance from parents), CX (cycle crossover), OB (order based crossover) and RANK (computing the average ranking of the elements). In computational experiments CX and OB performed best.

In [19] a similar method based on combining a classical GA with a local search is presented. It is called *hybrid genetic algorithm* (HGA) and it is very similar to the method in [31]. The local search is also based on exchanges. Additionally, this method also applies the CX and OB crossover operators. However, instead of DPX and RANK, it applies PMX (partially matched crossover).

3.8 SA—simulated annealing

Simulated annealing proceeds in the same way as ordinary local search, but incorporates some randomization in move selection to avoid getting trapped in a local optimum by means of non-improving moves. The moves are accepted according to probabilities taken from the analogy with the *annealing process*. As far as we know there is no previous implementation of SA for the LOP although some methods, such as the noising algorithm by Charon and Hudry [7] are based on the same principle. We implemented an SA method based on insertion moves like the other local search based metaheuristics in this section.

3.9 ILS—iterated local search

In [31] an iterated local search algorithm is proposed. This method iterates between local search phases by applying three main steps: perturb a locally optimal solution with exchange moves, apply the local search based on insertions to the perturbed solution, and determine the new solution to be perturbed based on the search history. The authors perform a comparison in this paper which favors the memetic algorithm.

4 The library of benchmark problems

We have compiled a comprehensive set of benchmark instances including all problem instances that have so far been used for conducting computational experiments. Furthermore we have included new instances. In their original definition, some problem instances are not in normal form. For the computations documented here, all instances have been transformed to normal form. We give a brief description of the origin and the characteristics of the groups of instances.

4.1 Input/output matrices

This is a well-known set of instances, first used in [18]. It contains 50 real-world linear ordering problems taken from input-output tables from various sources. They are comparatively easy and are thus more of interest for economists than for the assessment of approximate methods for hard problems. The original entries in these tables were not necessarily integral, but for the Linear Ordering Library `LOLIB` they were scaled to integral values.

4.2 SGB instances

These instances were used in [24] and are taken from the *Stanford GraphBase* [23]. They are random instances with entries drawn uniformly distributed from $[0, 25000]$. The set has a total of 25 instances with $n = 75$.

4.3 Random instances of type A

This is a set with 175 random instances that has been widely used for experiments. Problems of type I (called *RandomAI*) are generated from a $[0, 100]$ uniform distribution. This type of problems was proposed in [29] and generated in [5]. Problems were originally generated from a $[0, 25000]$ uniform distribution in [24] and modified afterwards, sampling from a significatively more narrow range (i.e., $[0, 100]$) to make them harder to solve. Sizes are $n =$ 100, 150, and 200, and there are 25 instances in each set giving a total of 75. We have extended this set including 25 additional instances with size $n = 500$.

Instances of type II (called *RandomAII*) are generated by counting the number of times a sector appears in a higher position than another in a set of randomly generated permutations. This type of instances was proposed in [6] and generated in [5]. For a problem of size n, $\frac{n}{2}$ permutations are generated. There are 25 instances for each n, where $n =$ 100, 150, and 200.

4.4 Random instances of type B

For these random instances, the superdiagonal entries are drawn uniformly distributed from $[0, U_1]$ and the subdiagonal entries from $[0, U_2]$, where $U_1 \geq U_2$. For the problems `p40-i` with $n = 40$, we set $U_1 = 100$ and $U_2 = 100 + 4(i - 1)$. For $n = 44$ and 50 we set $U_1 = 100$ and $U_2 = 100 + 2(i - 1)$ for the problems `p44-i` and `p50-i`, respectively.

4.5 Instances of Mitchell and Borchers

These instances have been used in [28]. They are random matrices where the subdiagonal entries are uniformly distributed in $[0, 99]$ and the superdiagonal entries are drawn uniformly from $[0, 39]$. Finally, a certain percentage of the entries was set to zero.

4.6 Instances of Schiavinotto and Stützle

Some further benchmark instances have been used in [31]. These instances were generated from the real-world input-output tables of Sect. 4.1 by replicating them to obtain larger problems. Thus, the distribution of numbers in these instances somehow reflects real input-output tables, but otherwise they behave more like random problems. That data set has been called `XLOLIB`; instances with $n = 150$ and 250 are available. For each original input-output instance, two instances, one of size $n = 150$ and another one of size $n = 250$ were generated. The original set contains 98 instances (49 with size 150 and 49 with size 250). We have removed 20 of these instances because there entries were so large that the sum of entries was not representable as a four byte integer. Therefore, this set finally has 78 instances.

4.7 Further special instances

We included some further problem instances that were used for experiments in some publications.

- `EX` instances
  These instances were used in particular in [9] and [10].
- `econ` instances
  The problems instances `econ36` through `econ77` were generated from the matrix `usa79`. They turned out not to be solvable as linear program using only 3-dicycle inequalities.
- `atp` instances
  These instances were created from the results of ATP tennis tournaments in 1993/1994. Nodes correspond to a selection of players and the weight of an arc (i, j) is the number of victories of player i against player j.
- Paley graphs
  Paley graphs have been used in [17] to prove results about the acyclic subdigraph polytope. They are a special class of tournaments where adjacency comes from an algebraic definition. They are constructed from the members of a suitable finite field by connecting pairs of elements that differ in a quadratic residue.

Table 1 summarizes the number of instances (#Instances) in each set described above. Moreover, it specifies the number of instances where the optimum is known (#Optima) or where only an upper bound is known (#Upper Bounds). In the computational experiments we call OPT-I to the set of 229 instances where the optimum is known, and UB-I to the set of 255 instances where only an upper bound is known.

`LOLIB` is available at the web sites

- http://comopt.ifi.uni-heidelberg.de/software/LOLIB
- http://heur.uv.es/optsicom/LOLIB

Also the constants eliminated by the transformation to normal form can be found there. In online Appendix we list all problems with their optimum solution values or currently known best lower and upper bounds.

Table 1 Number of instances in each set

Set	#Instances	#Optima	#Upper bounds
IO	50	50	–
SGB	25	25	–
RandomAI	100	–	100
RandomAII	75	25	50
RandomB	90	70	20
MB	30	30	–
XLOLIB	78	–	78
Special	36	29	6
Total	484	229	255

5 Computational comparison

We divide our experimentation into three parts according to the classification of the instances and methods introduced in previous sections.

In the first experiment we consider the 229 instances of OPT-I and the simple heuristics described in Sect. 2. In this experiment for each instance and each method we compute the relative deviation *Dev* (in percent) between the best solution value *Value* obtained with the method and the optimal value for that instance. For each method, we also report the number of instances *#Opt* for which an optimum solution could be found. In addition, we calculate the so-called *score statistic* [30] associated with each method. For each instance, the *nrank* of method *M* is defined as the number of methods that found a better solution than the one found by *M*. In the event of ties, the methods receive the same *nrank*, equal to the number of methods strictly better than all of them. The value of *Score* is the sum of the *nrank* values for all the instances in the experiment, thus, the lower the *Score* the better the method.

Tables 2 and 3 report about our results for 7 constructive and 7 improving heuristics respectively on the OPT-I set. We do not report running times in these tables because these methods are very fast and their running times are extremely short (below 1 millisecond).

Table 2 shows results for:

- `CW`: Chenery and Watanabe algorithm
- `AM-O`: Aujac and Masson algorithm (output coefficients)
- `AM-I`: Aujac and Masson algorithm (input coefficients)
- `Bcq`: Becker algorithm (based on quotients)
- `Bcr`: Becker algorithm (based on rotations)
- `BI1`: Best Insertion algorithm (variant 1)
- `BI2`: Best Insertion algorithm (variant 2)

In Table 3 the results obtained with the following improvement methods (started with a random initial solution) are given:

- `LSi`: Local Search based on insertions
- `2opt`: Local Search based on 2-opt

Table 2 Constructive methods on OPT-I instances

	CW	AM-O	AM-I	Bcq	Bcr	BI1	BI2
IO							
Dev(%)	19.07	32.94	31.45	4.07	30.19	3.24	4.18
Score	231	291	266	101	289	89	104
#Opt	0	0	0	0	0	0	0
SGB							
Dev(%)	12.83	26.15	26.15	3.57	31.56	3.89	3.03
Score	100	125	125	54	175	56	40
#Opt	0	0	0	0	0	0	0
RandomAII							
Dev(%)	2.60	36.50	36.55	1.57	37.75	1.09	1.26
Score	100	135	136	68	162	34	48
#Opt	0	0	0	0	0	0	0
RandomB							
Dev(%)	10.13	24.69	24.69	7.04	26.41	5.24	4.87
Score	276	368	368	194	454	124	106
#Opt	0	0	0	0	0	0	0
MB							
Dev(%)	8.40	43.37	43.37	2.90	40.30	2.49	2.27
Score	120	178	178	80	154	52	48
#Opt	0	0	0	0	0	0	0
Special							
Dev(%)	12.05	34.16	33.11	4.58	27.86	5.01	5.41
Score	110	161	156	56	153	64	65
#Opt	2	1	2	2	2	1	2
OPT-I							
Avg. Dev(%)	10.85	32.97	32.55	3.95	32.35	3.49	3.50
Sum #Opt	2	1	2	2	2	1	2

– `3opt`: Local Search based on 3-opt
– `LSe`: Local Search based on exchanges
– `KL1`: Kernighan-Lin based on exchanges
– `KL2`: Kernighan-Lin based on insertions
– `LE`: Local enumeration

Results in Table 2 clearly indicate that OPT-I instances pose a challenge for the simple heuristics with average percentage deviations ranging from 3.49% to 32.97%. On the other hand, the improvement methods are able to obtain better solutions with average percentage deviations (shown in Table 3) ranging from 0.57% to 2.30%. We have not observed significant differences when applying the improvement method from different initial solutions. For example, as shown in Table 3 the `LSi` method

Table 3 Improvement methods on OPT-I instances

	`LSi`	`2opt`	`3opt`	`LSe`	`KL1`	`KL2`	`LE`
IO							
Dev(%)	1.08	0.64	0.23	1.73	1.35	4.24	0.01
Score	243	181	125	295	239	232	49
#Opt	0	1	4	0	1	0	43
SGB							
Dev(%)	0.16	0.81	0.53	1.35	0.63	0.28	1.09
Score	42	122	84	154	100	63	135
#Opt	1	0	0	0	0	1	0
RandomAII							
Dev(%)	0.16	0.77	0.38	0.62	0.61	0.09	0.54
Score	46	161	81	134	134	29	112
#Opt	0	0	0	0	0	0	0
RandomB							
Dev(%)	0.79	4.04	2.13	3.78	3.51	0.61	3.56
Score	124	400	232	387	359	95	362
#Opt	1	0	0	0	0	1	0
MB							
Dev(%)	0.02	0.57	0.14	3.10	0.40	0.01	0.17
Score	64	178	113	210	149	41	83
#Opt	0	0	0	0	0	4	3
Special							
Dev(%)	1.19	3.30	2.05	3.21	2.40	0.89	3.52
Score	69	144	82	138	120	49	156
#Opt	4	2	2	2	3	3	3
OPT-I							
Avg. Dev(%)	0.57	1.69	0.91	2.30	1.49	1.02	1.48
Sum #Opt	5	3	6	2	4	8	49

exhibits a *Dev* value of 0.16% on the RandomAII instances when it is started from random solutions. When it is run from the `CW` or the `Bcr` solutions, it obtains a *Dev* value of 0.17% and 0.18% respectively.

We applied the *non-parametric Friedman test* for multiple correlated samples to the best solutions obtained by each of the 7 constructive methods. This test computes, for each instance, the rank value of each method according to solution quality (where rank 7 is assigned to the best method and rank 1 to the worst one). Then, it calculates the average rank values of each method across all the instances solved. If the averages differ greatly, the associated p-value or significance will be small. The resulting p-value of 0.000 obtained in this experiment clearly indicates that there are statistically significant differences among the 7 methods tested. Specifically, the rank values produced by this test are 6.2 (`BI2`), 6.2 (`BI1`), 5.6 (`Bcq`), 3.9 (`CW`), 2.2 (AM-I), 2.1 (AM-O) and 1.9 (Bcr). Heuristics BI1, BI2 and Bcq consistently provide the best solutions among the constructive heuristics. Considering that BI1 and BI2 obtain very similar rank values, we compared both with two well-known nonparametric tests for pairwise comparisons: the *Wilcoxon test* and the *Sign test*. The former one answers the question: Do the two samples (solutions obtained with BI1 and BI2 in our case) represent two different populations? The resulting p-value of 0.857 indicates that the values compared could come from the same method. On the other hand, the Sign test computes the number of instances on which an algorithm supersedes another one. The resulting p-value of 0.792 indicates that there is no clear winner between BI1 and BI2 when we consider all the instances in the OPT-I set. If we apply these two pairwise tests to compare BI1 and Bcq we obtain a p-value of 0.000 (in both tests) indicating that there are significant differences between these constructive methods.

Regarding the improvement methods, LSi and KL2 seem the best ones, although the differences among methods are smaller than in the constructive procedures. The significance level of the Friedman test is 0.000 indicating that there are statistically significant differences among the 7 methods tested, and the rank values in this test are: 5.7 (KL2), 5.4 (LSi), 4.8 (3opt), 4.1 (LE), 3.1 (KL1), 2.8 (2opt) and 2.2 (LSe). Considering that KL2 and LSi obtain very similar rank values, we compared both with the *Wilcoxon test* and the *Sign test*. The resulting p-value of 0.005 obtained in the former one indicates that the values compared come from different methods (using the typical significance level of $\alpha = 0.05$ as the threshold between rejecting or not rejecting the null hypothesis). On the other hand, the resulting p-value of 0.000 obtained in the Sign test indicates that KL2 consistently beats LSi when we consider all the instances in the OPT-I set.

In our second experiment we consider the metaheuristics described in Sect. 3 to solve the OPT-I instances. Table 4 reports the values *Dev*, *#Opt* and *Score* obtained with the following 10 methods executed for 10 seconds on each instance:

- TS: Tabu Search
- MA: Memetic Algorithm
- VNS: Variable Neighborhood Search
- SA: Simulated Annealing
- SS: Scatter Search
- GRASP: Greedy randomized adaptive search procedure
- ILS: Iterated Local Search
- GA: Genetic Algorithm
- KLM: Kernighan-Lin multi-start
- CKM: Chanas and Kobilansky multi-start

Table 4 shows that most of the metaheuristic algorithms considered are able to obtain all the optimal solutions within the time limit of 10 seconds considered (they actually obtain it in around 1 second). The Friedman test indicates that there are statistically significant difference among the methods although some rank values are very similar: 7.15 (ILS), 7.15 (MA), 7.00 (TS), 6.80 (VNS), 6.38 (GRASP), 6.00 (SS), 5.92 (CKM), 4.30 (KLM), 2.80 (SA) and 1.51 (GA). It must be noted that ILS and MA obtain the optimum value in all the instances but one of OPT-I and TS obtains 215

Table 4 Metaheuristic algorithms on OPT-I instances running for 10 seconds

	TS	MA	VNS	SA	SS	GRASP	ILS	GA	CKM	KLM
IO										
Dev(%)	0.00	0.00	0.00	0.03	0.01	0.00	0.00	0.38	0.23	0.00
Score	50	50	50	304	97	58	50	382	373	50
#Opt	50	50	50	16	42	49	50	9	10	49
SGB										
Dev(%)	0.00	0.00	0.00	0.03	0.00	0.00	0.00	0.76	0.01	0.00
Score	25	25	25	222	52	33	25	245	159	77
#Opt	25	25	25	0	20	23	25	0	7	16
RandomAII										
Dev(%)	0.00	0.00	0.00	0.08	0.01	0.01	.00	21.44	0.01	0.02
Score	25	25	53	222	92	136	25	250	102	136
#Opt	25	25	19	0	13	5	25	0	11	6
RandomB 70										
Dev(%)	0.00	0.00	0.02	0.25	0.01	0.00	0.00	0.75	0.13	0.00
Score	70	70	109	546	123	70	70	676	381	82
#Opt	70	70	64	10	62	70	70	1	28	68
MB										
Dev(%)	0.00	0.00	0.00	1.33	0.00	0.00	0.00	35.53	0.00	0.00
Score	30	30	30	270	112	80	30	300	41	169
#Opt	30	30	30	0	16	21	30	0	28	10
Special										
Dev(%)	0.02	0.00	0.11	0.39	0.09	0.05	0.00	1.38	0.14	0.01
Score	77	30	75	189	126	98	30	251	158	107
#Opt	15	28	20	6	12	14	28	4	9	15
OPT-I										
Dev(%)	0.00	0.00	0.02	0.35	0.02	0.01	0.00	10.04	0.09	0.01
#Opt	215	228	208	32	165	182	228	14	93	124

optima out of 229 instances. As expected, the associated p-value of the Wilcoxon and Sign tests when comparing `ILS` and `MA` is 1, indicating that we cannot differentiate between both methods in this case. On the other hand, both pairwise tests provide a p-value lower than 0.05 when comparing `ILS` and `TS`, indicating that there are significant differences between the results of these two methods (although they present the same average percentage deviation to optima: 0.00%). We conclude that instances in OPT-I are easy for the best metaheuristics and therefore not adequate to compare them.

We have also computed the upper bound obtained with the linear programming formulation (LP relaxation) described in Sect. 1 on the OPT-I instances. The percentage gap between the value of this relaxation and the optimal solution value provides a measure of its quality. Specifically, the average gap on each subset of instances is: 0.00% (IO), 0.00% (SGB), 0.08% (RandomAII), 1.47% (15 RandomB instances with $n \leq 150$), 0.00% (MB) and 0.88% (Special). We can conclude that the value of the upper bound is close to the optimal value on the OPT-I instances.

In our third experiment we target the 255 instances in UB-I where we only have an upper bound for comparison. We do not consider the simple heuristics anymore (since they already had difficulties to solve the easier problems in OPT-I) and limit this comparison to the metaheuristic algorithms considered above. In this experiment, we first determine the best known value for all the instances in UB-I. According to the previous experiment (Table 4) ILS, MA and TS, seem to be the best methods. We therefore run them for one hour on each instance to determine the best known value *BestValue* for the 255 instances in UB-I.

We give for each instance and each method the relative deviation *D.Best* (in percent) between the best solution value *Value* obtained with the method and the best known value *BestValue* as well as the relative deviation *D.UB* (in percent) between *Value* and the upper bound. For each method, we also report the number of instances *#Best* for which the value of the solution is equal to *BestValue*. As in the previous experiment we calculate the score statistic. Tables 5 and 6 report the values of these four statistics on the UB-I instances when running the 10 metaheuristic algorithms for 10 and 600 seconds respectively.

According to the differences among methods observed in Table 5, where the deviations w.r.t. the best solution known range from 0.05% to 16.15%, we can conclude that the instances in set UB-I are more difficult to solve than those in OPT-I (where the deviations range from 0.00% to 10.04%).

Results in Table 5 show that MA is able to obtain the largest number of best solutions (97 of a total of 255 instances) in short runs (10 seconds). No other method is able to obtain more than 55 best solutions, which clearly indicates the superiority of MA. On the other hand, considering average percentage deviations with respect to the best solutions, the differences among the methods appear to be very small. MA and ILS present on average a deviation of 0.05% and 0.06% respectively, while TS, SS and VNS present averages deviations of 0.24% 0.27% and 0.29%, respectively. This indicates that although these methods are not able to match most of the best solution values, they obtain solutions with values very close to the best.

Considering the deviations with respect to the upper bound, most of the methods present similar values ranging from 6.66% to 7.40% (with the exception of GA (21.98%)). The associated p-value of the Friedman test is 0.000 indicating that there are differences among the 10 metaheuristic procedures observed in this table. The rank values obtained with this non-parametric test are: 9.4 (MA), 8.9 (ILS) 7.3 (TS), 6.3 (VNS), 6.2 (SS), 4.9 (GRASP), 4.1 (SA), 3.6 (KLM), 3.2 (CKM) and 1.0 (GA). We perform now pairwise comparison between MA and ILS. The resulting p-value of 0.000 obtained in both the Wilcoxon and the Sign tests clearly indicates that there are significant differences between the results of both methods. Similarly, we obtain a p-value of 0.000 on both tests when comparing ILS and TS, or MA and TS.

Results in Table 6 are in line with those of Table 5 (although as expected, the longer the run time the lower the deviations). The average percentage deviations with

Table 5 Metaheuristic algorithms on UB-I instances running for 10 seconds

	TS	MA	VNS	SA	SS	GRASP	ILS	GA	CKM	KLM
RandomAI										
D.Best(%)	0.12	0.05	0.47	1.77	0.27	0.42	0.09	10.59	0.76	0.98
D.UB(%)	17.81	17.75	18.10	18.88	17.92	18.05	17.79	26.28	18.34	18.45
Score	269	115	581	741	423	561	200	1000	818	792
#Best	5	33	0	0	1	0	17	0	0	0
RandomAII										
D.Best(%)	0.01	0.00	0.01	0.07	0.02	0.04	0.00	35.97	0.06	0.02
D.UB(%)	0.38	0.38	0.39	0.44	0.40	0.41	0.38	36.21	0.44	0.40
Score	84	26	93	200	133	176	53	216	216	128
#Best	3	39	8	0	0	0	14	0	0	0
RandomB										
D.Best(%)	0.00	0.00	0.00	0.31	0.02	0.00	0.00	0.91	0.01	0.14
D.UB(%)	3.20	3.20	3.26	3.51	3.22	3.20	3.21	4.08	3.22	3.34
Score	20	20	76	180	57	20	24	195	60	104
#Best	20	20	11	0	13	20	19	0	12	7
XLOLIB										
D.Best(%)	0.64	0.14	0.44	0.56	0.71	1.17	0.15	24.01	1.76	1.95
D.UB(%)	3.21	2.72	3.01	3.13	3.27	3.72	2.73	25.96	4.30	4.48
Score	385	126	269	343	398	538	123	780	656	668
#Best	0	1	0	0	0	0	1	0	0	0
Special										
D.Best(%)	0.45	0.05	0.52	2.07	0.35	0.68	0.06	9.29	0.40	0.45
D.UB(%)	9.61	9.26	9.67	11.04	9.52	9.81	9.27	17.35	9.57	9.61
Score	25	8	32	64	21	32	8	69	21	23
#Best	3	4	2	0	3	3	4	0	3	3
UB-I										
D.Best(%)	0.24	0.05	0.29	0.95	0.27	0.46	0.06	16.15	0.60	0.71
D.UB(%)	6.84	6.66	6.89	7.40	6.87	7.04	6.68	21.98	7.17	7.26
#Best	31	97	21	0	17	23	55	0	15	10

respect to the best solutions of 0.05% achieved with MA in 10 seconds runs is reduced to 0.01% in the 600 seconds runs. Similarly, the average percentage deviations with respect to the best solutions of 0.06% and 0.24% achieved with ILS and TS in 10 seconds runs, drops to 0.03% and 0.16% respectively in the 600 seconds runs. On the other hand, MA is able to match 205 best known solutions, while none of the other methods obtains more than 108 best known solutions (which is the case of the ILS method followed by TS with 65). These results confirm that MA is the best method followed by ILS and TS. As in the previous experiment, we applied the Friedman test, obtaining a p-value of 0.000 and the following rank values: 9.25 (MA),

Table 6 Metaheuristic algorithms on UB-I instances running for 600 seconds

	TS	MA	VNS	SA	SS	GRASP	ILS	GA	CKM	KLM
RandomAI										
D.Best(%)	0.10	0.00	0.41	0.40	0.18	0.28	0.09	1.14	0.44	0.27
D.UB(%)	17.78	17.72	18.06	18.04	17.87	17.93	17.79	18.65	18.06	17.90
Score	253	103	640	679	418	492	254	895	703	488
#Best	20	97	0	0	2	2	23	0	0	15
RandomAII										
D.Best(%)	0.00	0.00	0.01	0.11	0.01	0.02	0.00	0.10	0.03	0.01
D.UB(%)	0.38	0.38	0.38	0.49	0.39	0.39	0.38	0.47	0.40	0.38
Score	102	50	181	485	272	324	139	379	379	190
#Best	21	50	16	0	2	0	14	0	0	9
RandomB										
D.Best(%)	0.00	0.00	0.03	0.01	0.04	0.00	0.00	0.87	0.00	0.00
D.UB(%)	3.20	3.20	3.24	3.21	3.24	3.20	3.20	4.05	3.20	3.20
Score	20	20	80	79	86	20	20	200	20	20
#Best	20	20	12	12	11	20	20	0	20	20
XLOLIB										
D.Best(%)	0.42	0.02	0.37	0.63	0.84	0.59	0.04	2.18	1.15	0.59
D.UB(%)	2.99	2.61	2.94	3.20	3.40	3.16	2.62	4.70	3.70	3.15
Score	298	121	289	426	529	407	113	702	622	438
#Best	0	33	0	0	0	0	44	0	0	0
Special										
D.Best(%)	0.26	0.02	0.23	1.20	0.24	0.55	0.00	2.25	0.18	0.15
D.UB(%)	9.44	9.24	9.42	10.27	9.43	9.70	9.22	11.21	9.38	9.35
Score	24	9	23	47	24	35	7	70	22	16
#Best	4	5	3	2	3	3	7	0	3	4
UB-I										
D.Best(%)	0.16	0.01	0.21	0.47	0.26	0.29	0.03	1.31	0.36	0.20
D.UB(%)	6.76	6.63	6.81	7.04	6.87	6.88	6.64	7.82	6.95	6.80
#Best	65	205	31	14	18	25	108	0	23	48

8.30 (ILS), 7.67 (TS), 5.99 (KLM), 5.57 (VNS), 5.24 (GRASP), 5.02 (SS), 3.58 (SA), 3.22 (CKM) and 1.16 (GA). Pairwise comparison between best methods (MA and ILS, MA and TS, ILS and TS) result in a p-value of 0.000 with the Wilcoxon and Sign tests in all the cases.

Considering all the runs performed with the metaheuristic procedures in the three previous experiments, we applied the Friedman test, obtaining a p-value of 0.000 and the following rank values overall: 8.65 (MA), 8.14 (ILS), 7.34 (TS), 6.22 (VNS), 5.75 (SS), 5.48 (GRASP), 5.13 (KLM), 3.54 (CKM), 3.53 (SA) and 1.22 (GA). We can establish three ordered groups of methods according to this rank-

Table 7 10 replications of 10 seconds on UB-I instances

	TS			MA			ILS		
	MAX	MIN	AVG	MAX	MIN	AVG	MAX	MIN	AVG
RandomAI									
D.Best(%)	0.31	0.49	0.40	0.01	0.12	0.06	0.14	0.25	0.20
D. UB(%)	37.50	37.61	37.56	37.32	37.38	37.35	37.39	37.47	37.43
RandomAII									
D.Best(%)	0.01	0.02	0.01	0.00	0.00	0.00	0.00	0.01	0.00
D. UB(%)	0.41	0.42	0.42	0.40	0.40	0.40	0.40	0.41	0.40
XLOLIB									
D.Best(%)	0.76	1.13	0.96	0.09	0.32	0.19	0.10	0.30	0.20
D. UB(%)	3.58	3.95	3.79	2.94	3.16	3.03	2.95	3.14	3.04

ing. The best methods are MA, ILS and TS. Acceptable performance is shown by VNS, SS, GRASP and KLM, while SA, CKM and GA would be classified as poor.

The predominant performance of MA is particularly intriguing because most of the elements in MA are present in some of the other methods. In particular SS and GA are also based on generating, improving and combining solutions as MA, and they share the same local search procedure based on insertions as the improving method. However, as described in [31], the implementation of the local search in MA is performed with a *pivoting rule* mechanism that explores the neighborhood of a solution in constant time. This reduces the CPU time of the local search by several orders of magnitude, thus permitting MA to explore a significantly larger number of solutions than the other competing methods. It seems then that the implementation details, specially the incremental computation of the move value, make an important difference in local search based methods.

In our final experiment we test the robustness of the three best methods identified so far: MA, ILS and TS. Specifically, we study in this experiment the behavior of the algorithms when they are replicated (executed several times from different random initial solutions). Table 7 shows the deviations (with respect to best solutions and upper bounds) of the maximum value (worst result) MAX, minimum value (best result) MIN and average value AVG of the three methods under comparison. We perform for each method on each problem instance 10 independent runs of 10 seconds each. We limit here the comparison to the most challenging instances: RandomAI, RandomAII and XLOLIB.

Results in Table 7 indicate that the three methods are quite robust. We observe very small variations between the MAX and MIN values in this table. Moreover, average values, AVG, are very similar (slightly worse) to those reported on Table 5, which correspond to a single run.

6 Conclusions

A computational comparison of 24 methods for the LOP has been presented. Overall, experiments with 484 instances were performed to compare the procedures. This extensive experimentation allows us to confirm that classical heuristics, mostly based on simple ordering rules, only obtain good solutions to relatively easy instances (although we can find large instances in this set). Within these heuristics, it turns out that the improvement methods, even when applied to random solutions, clearly outperform the constructive procedures. When we target much more difficult instances, we need to apply complex metaheuristics to obtain high quality solutions. Our experimentation reveals that the memetic algorithm implementation `MA` seems best suited for this problem closely followed by iterated local search `ILS`, and with the tabu search `TS` ranked in third place.

Acknowledgements The authors thank T. Schiavinotto and T. Stützle for providing their implementation of the memetic and ILS algorithms [31]. The authors also thank J.M. Gonzalez for his implementation of a genetic algorithm.

This research has been partially supported by the Ministerio de Ciencia e Innovación of Spain (TIN2009-07516).

References

1. Achatz, H., Kleinschmidt, P., Lambsdorff, J.: Der Corruption Perceptions Index und das Linear Ordering Problem. ORNews **26**, 10–12 (2006)
2. Aujac, H.: La hiérarchie des industries dans un tableau des echanges industriels. Rev. Econ. **2**, 169–238 (1960)
3. Becker, O.: Das Helmstädtersche Reihenfolgeproblem – die Effizienz verschiedener Näherungsverfahren. In: Computer Uses in the Social Sciences, Bericht einer Working Conference des Inst. f. höh. Studien u. wiss. Forsch. Wien (1967)
4. Boenchendorf, K.: Reihenfolgenprobleme/Mean-Flow-Time Sequencing. Mathematical Systems in Economics, vol. 74. Verlagsgruppe Athenäum, Hain, Scriptor, Königstein (1982)
5. Campos, V., Glover, F., Laguna, M., Martí, R.: An experimental evaluation of a scatter search for the linear ordering problem. J. Glob. Optim. **21**, 397–414 (2001)
6. Chanas, S., Kobylanski, P.: A new heuristic algorithm solving the linear ordering problem. Comput. Optim. Appl. **6**, 191–205 (1996)
7. Charon, I., Hudry, O.: A survey on the linear ordering problem for weighted or unweighted tournaments. 4OR **5**, 5–60 (2007)
8. Chenery, H.B., Watanabe, T.: International comparisons of the structure of production. Econometrica **26**, 487–521 (1958)
9. Christof, T.: Low-dimensional 0/1-Polytopes and Branch-and-Cut in Combinatorial Optimization. Shaker, Aachen (1997)
10. Christof, T., Reinelt, G.: Combinatorial optimization and small polytopes. Top **4**, 1–64 (1996)
11. Condorcet, M.J.A.N.: Essai sur l'application de l'analyse à la probabilité des décisions rendues à la pluralité des voix (1785), Paris
12. Feo, T., Resende, M.G.C.: Greedy randomized adaptive search procedures. J. Glob. Optim. **2**, 1–27 (1995)
13. Garcia, C.G., Pérez-Brito, D., Campos, V., Martí, R.: Variable neighborhood search for the linear ordering problem. Comput. Oper. Res. **33**, 3549–3565 (2006)
14. Glover, F.: Future paths for integer programming and links to artificial intelligence. Comput. Oper. Res. **13**, 533–549 (1986)
15. Glover, F., Klastorin, T., Klingman, D.: Optimal weighted ancestry relationships. Manag. Sci. **20**, 1190–1193 (1974)
16. Glover, F., Laguna, M.: Tabu Search. Kluwer Academic, Dordrecht (1997)

17. Goemans, M.X., Hall, L.A.: The strongest facets of the acyclic subgraph polytope are unknown. In: Proc. of the 5th Int. IPCO Conference. LNCS, vol. 1084, pp. 415–429. Springer, Berlin (1996)
18. Grötschel, M., Jünger, M., Reinelt, G.: A cutting plane algorithm for the linear ordering problem. Oper. Res. **32**, 1195–1220 (1984)
19. Huang, G., Lim, A.: Designing a hybrid genetic algorithm for the linear ordering problem. In: Cantu-Paz, E., et al. (eds.) Proc. of Genetic and Evolutionary Computation—GECCO 2003. LNCS, vol. 2723, pp. 1053–1064. Springer, Berlin (2003)
20. Hansen, P., Mladenovic, N.: Variable neighborhood search. In: Glover, F., Kochenberger, G. (eds.) Handbook of Metaheuristics, pp. 145–184 (2003)
21. Jünger, M., Mutzel, P.: 2-layer straightline crossing minimization: performance of exact and heuristic algorithms. J. Graph Algorithms Appl. **1**, 1–25 (1997)
22. Kernighan, B.W., Lin, S.: An efficient heuristic procedure for partitioning graphs. Bell Syst. Tech. J. **49**, 291–308 (1979)
23. Knuth, D.E.: The Stanford GraphBase: A Platform for Combinatorial Computing. Addison-Wesley, Reading (1993)
24. Laguna, M., Martí, R., Campos, V.: Intensification and diversification with elite tabu search solutions for the linear ordering problem. Comput. Oper. Res. **26**, 1217–1230 (1999)
25. Laguna, M., Martí, R.: Scatter Search: Methodology and Implementations in C. Kluwer Academic, Dordrecht (2003)
26. Leontief, W.: Quantitative input-output relations in the economic system of the United States. Rev. Econ. Stat. **18**, 105–125 (1936)
27. Martí, R., Reinelt, G.: The Linear Ordering Problem: Exact and Heuristics Methods in Combinatorial Optimization. Applied Mathematical Sciences, vol. 175. Springer, Berlin (2010)
28. Mitchell, J.E., Borchers, B.: Solving linear ordering problems. In: Frenk, H., Roos, K., Terlaky, T., Zhang, S. (eds.) High Performance Optimization. Applied Optimization, vol. 33, pp. 340–366. Kluwer Academic, Dordrecht (2000)
29. Reinelt, G.: The Linear Ordering Problem: Algorithms and Applications. Research and Exposition in Mathematics, vol. 8. Heldermann, Berlin (1985)
30. Ribeiro, C.C., Uchoa, E., Werneck, R.F.: A hybrid GRASP with perturbations for the Steiner problem in graphs. INFORMS J. Comput. **14**, 228–246 (2002)
31. Schiavinotto, T., Stützle, T.: The linear ordering problem: instances, search space analysis and algorithms. J. Math. Model. Algorithms **3**, 367–402 (2004)