

A Benchmark Library and a Comparison of Heuristic Methods for the Linear Ordering Problem

Rafael Martí · Gerhard Reinelt ·
Abraham Duarte

August 3, 2009

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 metaheuristics 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.

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 both. T is an *acyclic tournament* if it does not contain any directed cycle. Obviously, an acyclic tournament induces an ordering $\langle v_{i_1}, v_{i_2}, \dots, v_{i_n} \rangle$ 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.

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@rjc.es

Alternatively, the problem can be defined as 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 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 $i = 1, \dots, n$,
- iii. $\min\{c_{ij}, c_{ji}\} = 0$, for all $1 \leq i < j \leq n$.

Note that this normal form is unique.

The LOP can be formulated as 0/1 linear integer programming 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

$$\begin{aligned} \max \quad & \sum_{(i,j) \in A_n} c_{ij} x_{ij} \\ & x_{ij} + x_{ji} = 1, \text{ for all } i, j \in V_n, i < j, \\ & x_{ij} + x_{jk} + x_{ki} \leq 2, \text{ for all } i, j, k \in V_n, i < j, i < k, j \neq k, \\ & x_{ij} \in \{0, 1\}, \text{ for all } i, j \in V_n. \end{aligned}$$

This IP is the basis for approaches to solve the LOP to optimality. If we replace the constraints " $x_{ij} \in \{0, 1\}$ " by " $0 \leq x_{ij} \leq 1$ " 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 since long time. Applications mentioned in the literature are e.g., aggregation of individual preferences [11], triangulation of input-output tables [26,8], determination of ancestry relationships [15], scheduling with preferences [4], assessment of corruption perception [1], minimizing crossing numbers in graph drawing [21].

This paper is organized as follows. In Section 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 Section 4. In Section 5 we report about extensive computational tests, addressing results obtainable as well as in short as in medium computation time. Some final remarks and the 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 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 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-0**) 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. Starting from a random ordering, at each iteration this method tries to improve the current ordering $O = \langle i_1, i_2, \dots, i_n \rangle$ by evaluating all orderings $\langle i_{m+1}, i_{m+2}, \dots, i_n, 1, 2, \dots, i_m \rangle$, for $m = 1, 2, \dots, n - 1$. 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 $\langle i_1, \dots, i_k \rangle$ 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

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

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 LS1 scans the list of nodes (in the order given by the current permutation) in search for the first node whose movement results in a 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 is considered. The authors tested a method in which only contiguous sectors are considered for exchange 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 the recent years a series of methods have appeared under the title *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.

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 this diversification, such methods can become reduced to tracing paths that are confined to a small area of the solution space, making 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 the insertion move $move(O_j, i)$ of each element O_j for all the positions 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 finishes 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 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 $\mathcal{N}_k(p)$ is the set of solutions that are obtained when we apply an insertion move $k - 1$ times from 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 [30] 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 (classical 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 [30]. The local search is also based on exchanges and it 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 the move selection to avoid getting trapped in a local optimum by means of non-improving moves. These 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 as the other local search based metaheuristics in this section.

3.9 ILS - Iterated Local Search

In [30] an iterated local search algorithm is proposed. This method iterates over 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 problems including all problem instances which 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 problems have been transformed to normal form. We give a brief description of the origin and the characteristics of the groups of problems.

4.1 Input/Output matrices

This is a well-known set of instances, first used for computation in [18]. It contains 50 real-world linear ordering problems generated 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 LOLIB they were scaled to integral values.

4.2 SGB instances

These instances were used in [24] and are taken from the *Stanford Graph-Base* [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 problems 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 [28] and generated in [5]. Problems were originally generated from a $[0, 25000]$ uniform distribution in [24] and modified afterwards, sampling from a significantly narrow range ($[0, 100]$) 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 $n = 500$.

Problems 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 problems was proposed in [6] and generated in [5]. For a problem of size n , $\frac{n}{2}$ permutations are generated. There are 25 instances with sizes 100, 150 and 200, respectively.

4.4 Random instances of type B

For these random problems, the superdiagonal entries are drawn uniformly distributed from the interval $[0, U_1]$ 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 $n = 50$ we set $U_1 = 100$ and $U_2 = 100 + 2(i - 1)$ for problems **p44-i**, **p50-i**, respectively.

4.5 Instances of Mitchell and Borchers

These instances have been used in [27]. They are random matrices where the subdiagonal entries are uniformly distributed in $[0, 99]$ and the superdiagonal entries are drawn uniformly from $[0, 39]$. Furthermore a certain percentage of the entries was zeroed out.

4.6 Instances of Schiavinotto and Stütze

Some further benchmark instances have been used in [30]. These instances were generated from the real-world input-output tables of 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. The data set has been called **XLOLIB**, instances with $n = 150$ and $n = 250$ 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 their entries were so large that the sum of entries was not representable as 4-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 in each set described above. Moreover, it specifies the number of instances for which the optimum or for which only an upper bound is known. In the computational experiments we call the set of 229 instances for which the optimum is known **OPT-I**, and the set of 255 instances for which an upper bound is known **UB-I**.

LOLIB is available at the web sites

- <http://comopt.ifi.uni-heidelberg.de/software/LOLIB>,
- <http://heur.uv.es/opticom/LOLIB>.

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

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

Table 1 Number of instances in each set

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 Section 2. In this experiment we compute for each instance and each method 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* [29] 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 milisecond).

Table 2 shows results for:

- CW: Chenery and Watanabe algorithm
- AM-0: 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

	CW	AM-0	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(%)	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
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

Table 2 Constructive methods on OPT-I instances

- **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 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.

	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	0	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

Table 3 Improvement methods on OPT-I instances

Heuristics BI1, BI2 and Bcq consistently provide the best solutions among the constructive heuristics. 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 6.3 (BI2), 6.2 (BI1), 5.5 (Bcq), 3.8 (CW), 2.2 (AM-I), 2.1 (AM-O) and 1.8 (Bcr).

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), 4.3 (LE),

3.1 (KL1), 2.8 (2opt) and 2.1 (LSe). Considering that KL2 and LSi 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 KL2 and LSi in our case) represent two different populations? The resulting p -value of 0.024 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 Sign test computes the number of instances on which an algorithm supersedes another one. The resulting p -value of 0.000 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 Section 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 Neighbourhood 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 metaheuristics 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: 7.55 (TS), 7.55 (MA), 7.35 (VNS), 7.03 (GRASP), 6.75 (SS), 6.60 (CKM), 5.10 (KLM), 3.62 (SA), 2.25 (GA) and 1.21 (ILS). It must be noted that TS and MA obtain the optimum value in all the instances of OPT-I. As expected, the associated p -value of the Wilcoxon test when comparing both methods is 1, indicating that we cannot differentiate between TS and MA in this case. We therefore conclude that instances in OPT-I are easy for the best metaheuristics and therefore not adequate to compare them.

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 metaheuristics 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) TS and MA, seem to be the best methods. We therefore run both methods for one hour on each instance to determine the best known value *BestValue* for the 255

	TS	MA	VNS	SA	SS	GRASP	ILS	GA	CKM	KLM
IO										
Dev(%)	0.00	0.00	0.00	0.03	0.01	0.00	10.18	0.38	0.23	0.00
Score	50	50	50	270	90	57	490	343	334	50
#Opt	50	50	50	16	42	49	0	9	10	9
SGB										
Dev(%)	0.00	0.00	0.00	0.03	0.00	0.00	7.83	0.76	0.01	0.00
Score	25	25	25	197	47	31	250	220	141	68
#Opt	25	25	25	0	20	23	0	0	7	16
RandAII										
Dev(%)	0.00	0.00	0.00	0.08	0.01	0.01	19.77	21.44	0.01	0.02
Score	25	25	47	197	80	116	233	242	88	117
#Opt	25	25	19	0	13	5	0	0	11	6
RandB 70										
Dev(%)	0.00	0.00	0.02	0.25	0.01	0.00	13.29	0.75	0.13	0.00
Score	70	70	103	486	115	70	700	607	339	80
#Opt	70	70	64	10	62	70	0	1	2	68
MB										
Dev(%)	0.00	0.00	0.00	1.33	0.00	0.00	29.12	35.53	0.00	0.00
Score	30	30	30	240	98	71	275	295	39	149
#Opt	30	30	30	0	16	21	0	0	28	10
Special										
Dev(%)	0.02	0.00	0.11	0.39	0.09	0.05	11.68	1.38	0.14	0.01
Score	64	30	66	166	109	84	281	226	138	930
#Opt	15	28	20	6	12	14	1	4	9	15
OPT-I										
Dev(%)	0.00	0.00	0.02	0.35	0.02	0.01	15.31	10.04	0.09	0.01
#Opt	215	228	208	32	165	182	1	14	93	124

Table 4 Metaheuristics on OPT-I instances running for 10 seconds

instances in UB-I. We did not try to measure the merit of these methods with these runs, but only wanted to obtain a value for future comparisons.

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 metaheuristics 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 0.04% to 17.72%, 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 15.31%).

Results in Table 5 show that MA is able to obtain the largest number of best solutions (98 of a total of 255 instances) in short runs (10 seconds). No other method is able to obtain more than 31 best solutions, which clearly indicates the superiority of MA. On the other hand, considering average percentage de-

viations with respect to the best solutions, the differences among the methods appear to be very small. MA presents on average a deviation of 0.04% while TS, SS and VNS present averages deviations of 0.23% 0.26% and 0.28%, respectively. This indicates that although these methods are not able to match the best solution values, they obtain solutions with values very close to the best.

	TS	MA	VNS	SA	SS	GRASP	ILS	GA	CKM	KLM
RandAI										
D.Best	0.12	0.05	0.47	1.77	0.26	0.42	13.42	10.59	0.77	0.98
D.UB	17.81	17.75	18.10	18.88	17.92	18.05	28.91	26.28	17.04	18.45
Score	201	105	482	641	326	461	964	936	677	692
#Best	5	33	0	0	1	0	0	0	0	0
RandAII										
D.Best	0.01	0.00	0.01	0.07	0.02	0.04	23.42	35.97	0.06	0.02
D.UB	0.38	0.38	0.39	0.44	0.40	0.41	23.71	36.21	0.44	0.40
Score	63	25	74	175	109	151	225	191	191	104
#Best	3	39	8	0	0	0	0	0	0	0
RandB										
D.Best	0.00	0.00	0.00	0.31	0.04	0.00	13.44	0.91	0.01	0.14
D.UB	3.20	3.20	3.26	3.51	3.24	3.20	16.22	4.08	3.22	3.34
Score	20	20	67	160	50	20	200	175	52	91
#Best	20	20	11	0	11	20	0	0	12	7
XLOLIB										
D.Best	0.62	0.12	0.42	0.53	0.68	1.14	22.37	23.99	1.74	1.92
D.UB	3.21	2.72	3.01	3.13	3.27	3.72	24.39	25.96	4.30	4.48
Score	307	87	200	266	320	460	724	758	578	590
#Best	0	2	0	0	0	0	0	0	0	0
Special										
D.Best	0.43	0.03	0.50	2.05	0.32	0.65	15.97	9.27	0.38	0.43
D.UB	9.61	9.26	9.67	11.04	9.52	9.81	23.58	17.35	9.57	9.61
Score	21	7	27	57	17	28	69	63	17	19
#Best	3	4	2	0	3	3	0	0	3	3
UB-I										
D.Best	0.23	0.04	0.28	0.95	0.26	0.45	17.72	16.15	0.59	0.70
D.UB	6.84	6.66	6.89	7.40	6.87	7.04	23.36	21.98	6.91	7.26
#Best	31	98	21	0	17	23	0	0	15	10

Table 5 Metaheuristics on UB-I instances running for 10 seconds

Moreover, considering the deviations with respect to the upper bound, most of the methods present similar values ranging from 6.66% to 7.26% (with the exception of ILS (23.36%) and GA (21.98%)). The associated p -value of the Friedman test is 0.000 indicating that there are differences among the 10 metaheuristics observed in this table. The rank values obtained with this non-parametric test are: 9.7 (MA), 8.2 (TS), 7.3 (VNS), 7.2 (SS), 5.9 (GRASP), 5.2 (SA), 4.5 (KLM), 4.1 (CKM), 1.6 (ILS) and 1.4 (GA).

	TS	MA	VNS	SA	SS	GRASP	ILS	GA	CKM	KLM
RandAI										
D.Best	0.10	0.00	0.41	0.40	0.18	0.28	11.84	1.14	0.43	0.27
D.UB	17.78	17.72	18.06	18.04	17.87	17.93	27.66	18.65	16.74	17.90
Score	206	100	544	581	336	403	900	795	591	421
#Best	20	100	0	0	2	2	0	0	0	15
RandAII										
D.Best	0.00	0.00	0.01	0.11	0.01	0.02	18.97	0.10	0.03	0.01
D.UB	0.38	0.38	0.38	0.49	0.39	0.39	19.28	0.47	0.40	0.38
Score	97	50	156	435	231	275	500	329	329	160
#Best	21	50	16	0	2	0	0	0	0	9
RandB										
D.Best	0.00	0.00	0.03	0.01	0.02	0.00	11.37	0.87	0.00	0.00
D.UB	3.20	3.20	3.24	3.21	3.22	3.20	14.21	4.05	3.20	3.20
Score	20	20	72	71	77	20	200	180	20	20
#Best	20	20	12	12	13	20	0	0	20	20
XLOLIB										
D.Best	0.39	0.00	0.34	0.61	0.82	0.57	19.47	2.15	1.12	0.56
D.UB	2.99	2.61	2.94	3.20	3.40	3.16	21.56	25.96	3.70	3.15
Score	221	78	213	348	451	329	702	624	544	360
#Best	0	78	0	0	0	0	0	0	0	0
Special										
D.Best	0.24	0.00	0.21	1.18	0.21	0.53	13.12	2.22	0.16	0.13
D.UB	9.44	9.24	9.42	10.27	9.43	9.70	21.05	11.21	9.38	9.35
Score	21	7	19	42	20	31	70	63	18	13
#Best	4	7	3	2	3	3	0	0	3	4
UB-I										
D.Best	0.15	0.00	0.20	0.46	0.25	0.28	14.96	1.30	0.35	0.19
D.UB	6.76	6.63	6.81	7.04	6.87	6.88	20.75	7.82	6.68	6.80
#Best	65	255	31	14	18	25	0	0	23	48

Table 6 Metaheuristics on UB-I instances running for 600 seconds

Results in Table 6 are in line with those of Table 5 (although as expected, the longer the run the lower the deviations). The average percentage deviations with respect to the best solutions of 0.04% achieved with MA in 10 seconds runs is reduced to 0.00% in the 600 seconds runs. Similarly, the average percentage deviations with respect to the best solutions of 0.23% and 0.28% achieved with TS and VNS in 10 seconds runs, drops to 0.15% and 0.20% respectively in the 600 seconds runs. On the other hand, MA is able to match the 255 best known solutions, while none of the other methods obtains more than 65 best known solutions (which is the case of the TS method). These results confirm that MA is the best method followed by TS. Considering all the runs performed with the metaheuristic procedures in the previous experiments, we applied the Friedman test, obtaining a p -value of 0.000 and the following rank values: 9.05 (MA), 8.04 (TS), 6.98 (VNS), 6.60 (SS), 6.31 (GRASP), 5.49 (KLM), 4.82 (CKM), 4.51 (SA), 1.93 (GA) and 1.26 (ILS). We can establish

three ordered groups of methods according to this ranking. The best methods are **MA**, **TS**, **VNS**, **SS** and **GRASP**. Acceptable performance is shown by **SA**, **KLM** and **CKM**, while **ILS** 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 [30], 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.

	TS			MA			VNS		
	MAX	MIN	AVG	MAX	MIN	AVG	MAX	MIN	AVG
RandAI									
D.Best	0.31	0.48	0.40	0.01	0.12	0.06	0.54	0.55	0.54
D. UB	37.50	37.61	37.56	37.32	37.38	37.35	37.65	37.65	37.65
RandAII									
D.Best	0.01	0.02	0.01	0.00	0.00	0.00	0.02	0.02	0.02
D. UB	0.41	0.42	0.42	0.40	0.40	0.40	0.42	0.42	0.42
XLOLIB									
D.Best	0.73	1.10	0.94	0.06	0.29	0.16	0.67	0.67	0.67
D. UB	3.58	3.95	3.79	2.94	3.16	3.03	3.52	3.52	3.52

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

In our final experiment we test the robustness of the three best methods identified so far: **MA**, **TS** and **VNS**. 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 10 independent runs of 10 seconds each. We limit here the comparison to the most challenging instances: **RandAI**, **RandAII** 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 tabu search TS. A deeper analysis permits to group the 10 metaheuristics reviewed into three categories: the first one includes the best methods: MA, TS, VNS, SS and GRASP; the second one includes the methods with an acceptable performance: SA KLM and CKM, while the third one includes those metaheuristics performing worst: ILS and GA.

7 Acknowledgement

The authors thank T. Schiavinotto and T. Stützle for providing their implementation of the memetic algorithm [30]. The authors also thank J.M. Gonzalez and A.T. Navarro for their implementations of the hybrid genetic algorithm [19] and iterated local search method [30], respectively.

This research has been partially supported by the Ministerio de Ciencia e Innovación of Spain (TIN2006-02696, TIN2009-07516) and by the Comunidad de Madrid-Universidad Rey Juan Carlos project (CCG08-URJC/TIC-3731).

References

1. ACHATZ, H., KLEINSCHMIDT P. AND LAMBSDORFF, J. *Der Corruption Perceptions Index und das Linear Ordering Problem*, ORNews **26** (2006), 10–12.
2. AUJAC, H.: *La hiérarchie des industries dans un tableau des échanges industriels*, Rev. Economique **2** (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.: *Reihenfolgeprobleme / Mean-flow-time sequencing*, Mathematical Systems in Economics 74, Verlagsgruppe Athenäum, Hain, Scriptor, 1982.
5. CAMPOS, V., GLOVER, F. LAGUNA, M. AND MARTÍ, R.: *An experimental evaluation of a scatter search for the linear ordering problem*, J. of Global Optimization **21** (2001), 397–414.
6. CHANAS, S. AND KOBYLANSKI, P.: *A new heuristic algorithm solving the linear ordering problem*, Computational Optimization and Applications **6** (1996), 191–205.
7. CHARON, I. AND HUDRY, O.: *A survey on the linear ordering problem for weighted or unweighted tournaments*, 4OR **5** (2007), 5–60.
8. CHENERY, H.B. AND WATANABE, T.: *International comparisons of the structure of production*, Econometrica **26** (1958), 487–521.

9. CHRISTOF, T.: *Low-dimensional 0/1-polytopes and branch-and-cut in combinatorial optimization*, Shaker, 1997.
10. CHRISTOF, T. AND REINELT, G.: *Combinatorial optimization and small polytopes*, Top **4** (1996), 1–64.
11. CONDORCET, M.J.A.N.: *Essai sur l'application de l'analyse à la probabilité des décisions rendues à la pluralité des voix*, Paris, 1785.
12. FEO, T. AND RESENDE, M.G.C.: *Greedy randomized adaptive search procedures*, J. of Global Optimization **2** (1995), 1–27.
13. GARCIA, C.G., PÉREZ-BRITO, D., CAMPOS, V. AND MARTÍ, R.: *Variable neighborhood search for the linear ordering problem*, Computers and Operations Research **33** (2006), 3549–3565.
14. GLOVER, F.: *Future paths for integer programming and links to artificial intelligence*, Computers and Operations Research **13** (1986), 533–549.
15. GLOVER, F., KLASTORIN, T., AND KLINGMAN, D.: *Optimal weighted ancestry relationships*, Management Science **20** (1974), 1190–1193.
16. GLOVER, F. AND LAGUNA, M.: *Tabu search*, Kluwer Academic Publishers, 1997.
17. GORMANS, M.X. AND HALL, L.A.: *The strongest facets of the acyclic subgraph polytope are unknown*, in: *Proc. of the 5th Int. IPCO Conference*, LNCS 1084, Springer, 1996, 415–429.
18. GRÖTSCHEL, M., JÜNGER, M. AND REINELT, G.: *A cutting plane algorithm for the linear ordering problem*, Operations Research **32** (1984), 1195–1220.
19. HUANG, G. AND 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 2723, Springer, 2003, 1053–1064.
20. HANSEN, P. AND MLADENOVIC, N.: *Variable neighborhood search*, in: Glover, F. and Kochenberger, G. (eds.) *Handbook of Metaheuristics* (2003), 145–184.
21. JÜNGER, M. AND MUTZEL, P.: *2-layer straightline crossing minimization: performance of exact and heuristic algorithms*, J. of Graph Algorithms and Applications **1** (1997), 1–25.
22. KERNIGHAN, B.W. AND LIN, S.: *An efficient heuristic procedure for partitioning graphs*, Bell Systems Technical Journal **49** (1979) 291–308.
23. KNUTH, D.E.: *The Stanford GraphBase: a platform for combinatorial computing*, Addison-Wesley, 1993.
24. LAGUNA, M., MARTÍ R. AND CAMPOS, V.: *Intensification and diversification with elite tabu search solutions for the linear ordering problem*, Computers and Operations Research **26** (1999), 1217–1230.
25. LAGUNA, M. AND MARTÍ R.: *Scatter search: methodology and implementations in C*, Kluwer Academic Publishers, 2003.
26. LEONTIEF, W.: *Quantitative input-output relations in the economic system of the United States*, The Review of Economics and Statistics **18** (1936).
27. MITCHELL, J.E. AND BORCHERS, B.: *Solving linear ordering problems*, in: Frenk, H., Roos, K., Terlaky, T. and Zhang, S. (eds.), *High Performance Optimization*, Applied Optimization Vol. 33) Kluwer, 2000, 340–366
www.rpi.edu/~mitchj/generators/linord.
28. REINELT, G.: *The linear ordering problem: algorithms and applications*, research and exposition in mathematics 8, Heldermann, 1985.
29. RIBEIRO, C.C., UCHOA, E. AND WERNECK, R.F.: *A Hybrid GRASP with Perturbations for the Steiner Problem in Graphs*, INFORMS Journal on Computing **14** (2002), 228–246.
30. SCHIAVINOTTO, T. AND STÜTZLE, T.: *The linear ordering problem: Instances, search space analysis and algorithms*, J. of Mathematical Modelling and algorithms **3** 2004, 367–402.

Appendix

We list all problem instances with optimum solution values (printed in bold) or best known lower and upper bounds. Note that after having conducted the experiments for this paper a small number of additional optima have been proved. But this does not affect the conclusions drawn from our computations.

Problem	Bounds	Problem	Bounds
be75eec	236464	be75np	716994
be75oi	111171	be75tot	980516
stabu70	362512	stabu74	541393
stabu75	553303	t59b11xx	209320
t59d11xx	147354	t59f11xx	122520
t59i11xx	8261545	t59n11xx	20928
t65b11xx	356758	t65d11xx	237739
t65f11xx	217295	t65i11xx	14469163
t65l11xx	16719	t65n11xx	32157
t65w11xx	138181029	t69r11xx	771149
t70b11xx	528419	t70d11xx	376725
t70d11xxb	366469	t70f11xx	360336
t70i11xx	24785782	t70k11xx	60659200
t70l11xx	25253	t70n11xx	52704
t70u11xx	21716400	t70w11xx	224319954
t70x11xx	283808865	t74d11xx	566089
t75d11xx	578304	t75e11xx	2739219
t75i11xx	63567735	t75k11xx	108844
t75n11xx	93777	t75u11xx	52708323
tiw56n54	91554	tiw56n58	125224
tiw56n62	176715	tiw56n66	226547
tiw56n67	226033	tiw56n72	365146
tiw56r54	102948	tiw56r58	129568
tiw56r66	209491	tiw56r67	222810
tiw56r72	270663	usa79	1813986

Table 8 Bounds for IO problems

Problem	Bounds	Problem	Bounds
sgb75.01	2724126	sgb75.02	2616392
sgb75.03	2747384	sgb75.04	2734169
sgb75.05	2707863	sgb75.06	2707280
sgb75.07	2727928	sgb75.08	2712837
sgb75.09	2687364	sgb75.10	2733387
sgb75.11	2732686	sgb75.12	2692548
sgb75.13	2714591	sgb75.14	2733926
sgb75.15	2732810	sgb75.16	2747797
sgb75.17	2747864	sgb75.18	2579344
sgb75.19	2736221	sgb75.20	2732021
sgb75.21	2740289	sgb75.22	2710122
sgb75.23	2720981	sgb75.24	2743879
sgb75.25	2731542		

Table 9 Bounds for SGB problems

Problem	Bounds	Problem	Bounds
t1d100.01	[106849,114468]	t1d100.02	[105916,114077]
t1d100.03	[109819,117843]	t1d100.04	[109120,117639]
t1d100.05	[108859,117538]	t1d100.06	[108181,117057]
t1d100.07	[108763,117118]	t1d100.08	[107377,115756]
t1d100.09	[108549,116527]	t1d100.10	[108755,117518]
t1d100.11	[107812,116637]	t1d100.12	[108352,116617]
t1d100.13	[108689,116816]	t1d100.14	[105510,114199]
t1d100.15	[108667,117221]	t1d100.16	[107374,115781]
t1d100.17	[105596,113860]	t1d100.18	[107804,115959]
t1d100.19	[107921,116987]	t1d100.20	[109785,119008]
t1d100.21	[107255,116326]	t1d100.22	[108107,116987]
t1d100.23	[106146,113264]	t1d100.24	[108728,116959]
t1d100.25	[106933,115311]	t1d150.01	[234918,261413]
t1d150.02	[234247,259050]	t1d150.03	[235980,262076]
t1d150.04	[234291,259619]	t1d150.05	[234379,259862]
t1d150.06	[233919,260035]	t1d150.07	[234863,260922]
t1d150.08	[237215,260956]	t1d150.09	[236893,261816]
t1d150.10	[234716,260359]	t1d150.11	[233935,259316]
t1d150.12	[235904,262105]	t1d150.13	[236893,262727]
t1d150.14	[234279,259552]	t1d150.15	[231941,257699]
t1d150.16	[232703,258626]	t1d150.17	[236503,262929]
t1d150.18	[233943,259865]	t1d150.19	[234560,260286]
t1d150.20	[234939,260652]	t1d150.21	[233606,259087]
t1d150.22	[234960,260134]	t1d150.23	[233225,258077]
t1d150.24	[236016,261151]	t1d150.25	[236072,261299]
t1d200.01	[410249,464286]	t1d200.02	[407161,460090]
t1d200.03	[406724,459566]	t1d200.04	[409262,462950]
t1d200.05	[410570,465427]	t1d200.06	[405801,457852]
t1d200.07	[411393,465758]	t1d200.08	[408572,461301]
t1d200.09	[407602,461319]	t1d200.10	[405802,458731]
t1d200.11	[410042,463460]	t1d200.12	[412494,464490]
t1d200.13	[407830,463572]	t1d200.14	[407759,461072]
t1d200.15	[408324,462516]	t1d200.16	[407349,461195]
t1d200.17	[409870,462673]	t1d200.18	[407369,460830]
t1d200.19	[411846,465780]	t1d200.20	[405019,458746]
t1d200.21	[406908,459819]	t1d200.22	[407060,461211]
t1d200.23	[407950,460547]	t1d200.24	[409922,462154]
t1d200.25	[404891,458197]	t1d500.01	[2402357,2888540]
t1d500.02	[2411570,2895970]	t1d500.03	[2404784,2893183]
t1d500.04	[2413600,2901834]	t1d500.05	[2391486,2876304]
t1d500.06	[2399394,2885012]	t1d500.07	[2400739,2888525]
t1d500.08	[2413108,2897720]	t1d500.09	[2406223,2892796]
t1d500.10	[2404420,2888237]	t1d500.11	[2416286,2902095]
t1d500.12	[2402581,2886004]	t1d500.13	[2405118,2892015]
t1d500.14	[2410693,2893998]	t1d500.15	[2411718,2898081]
t1d500.16	[2416067,2899513]	t1d500.17	[2401800,2889687]
t1d500.18	[2421159,2906476]	t1d500.19	[2404029,2888547]
t1d500.20	[2414713,2899907]	t1d500.21	[2405615,2892266]
t1d500.22	[2408164,2894690]	t1d500.23	[2408689,2893340]
t1d500.24	[2402712,2886781]	t1d500.25	[2405718,2890451]

Table 10 Bounds for RandA1 problems

Problem	Bounds	Problem	Bounds
t2d100.01	25362	t2d100.02	28326
t2d100.03	25886	t2d100.04	26076
t2d100.05	25118	t2d100.06	25380
t2d100.07	27144	t2d100.08	23784
t2d100.09	27752	t2d100.10	26690
t2d100.11	25106	t2d100.12	26782
t2d100.13	27878	t2d100.14	25878
t2d100.15	24232	t2d100.16	28206
t2d100.17	26704	t2d100.18	26928
t2d100.19	28760	t2d100.20	25220
t2d100.21	24452	t2d100.22	27230
t2d100.23	25588	t2d100.24	24800
t2d100.25	23742	t2d150.01	[76041,76276]
t2d150.02	[73622,73811]	t2d150.03	[69705,69894]
t2d150.04	[73962,74136]	t2d150.05	[79723,79847]
t2d150.06	[75440,75604]	t2d150.07	[73858,74067]
t2d150.08	[67463,67803]	t2d150.09	[70739,70915]
t2d150.10	[69027,69302]	t2d150.11	[72800,72987]
t2d150.12	[72180,72429]	t2d150.13	[74579,74739]
t2d150.14	[68132,68331]	t2d150.15	[76828,77007]
t2d150.16	[72018,72238]	t2d150.17	[70185,70468]
t2d150.18	[73191,73424]	t2d150.19	[75958,76133]
t2d150.20	[67369,67651]	t2d150.21	[70292,70571]
t2d150.22	[69286,69515]	t2d150.23	[74799,74958]
t2d150.24	[70063,70323]	t2d150.25	[73853,74026]
t2d200.01	[147736,148294]	t2d200.02	[144212,144903]
t2d200.03	[141368,142105]	t2d200.04	[150876,151471]
t2d200.05	[150236,150854]	t2d200.06	[141254,142009]
t2d200.07	[149752,150379]	t2d200.08	[149908,150415]
t2d200.09	[141954,142747]	t2d200.10	[149620,150232]
t2d200.11	[147542,148262]	t2d200.12	[152468,153064]
t2d200.13	[137612,138514]	t2d200.14	[144380,145083]
t2d200.15	[140442,141227]	t2d200.16	[147444,148114]
t2d200.17	[131844,132728]	t2d200.18	[151188,151663]
t2d200.19	[137314,138182]	t2d200.20	[146504,147051]
t2d200.21	[143564,144221]	t2d200.22	[146920,147524]
t2d200.23	[145024,145731]	t2d200.24	[151258,151769]
t2d200.25	[149128,149723]		

Table 11 Bounds for RandA2 problems

Problem	Bounds	Problem	Bounds
p40-01	29457	p40-02	27482
p40-03	28061	p40-04	28740
p40-05	27450	p40-06	29164
p40-07	28379	p40-08	28267
p40-09	30578	p40-10	31737
p40-11	30658	p40-12	30986
p40-13	33903	p40-14	34078
p40-15	34659	p40-16	36044
p40-17	38201	p40-18	37562
p40-19	38956	p40-20	39658
p44-01	35948	p44-02	35314
p44-03	34335	p44-04	33551
p44-05	34827	p44-06	33962
p44-07	33171	p44-08	34127
p44-09	33403	p44-10	33778
p44-11	34016	p44-12	33850
p44-13	35385	p44-14	35801
p44-15	33827	p44-16	36188
p44-17	35454	p44-18	36669
p44-19	36436	p44-20	37438
p44-21	37786	p44-22	36722
p44-23	36605	p44-24	38286
p44-25	38129	p44-26	39107
p44-27	39170	p44-28	40264
p44-29	41819	p44-30	40387
p44-31	43817	p44-32	42545
p44-33	42355	p44-34	44988
p44-35	44114	p44-36	45575
p44-37	45297	p44-38	47414
p44-39	48979	p44-40	47774
p44-41	48137	p44-42	49511
p44-43	51014	p44-44	51949
p44-45	52857	p44-46	52776
p44-47	54122	p44-48	54355
p44-49	57279	p44-50	56444
p50-01	44667	p50-02	43835
p50-03	44256	p50-04	43928
p50-05	[42907,44196]	p50-06	[42325,43765]
p50-07	[42640,43977]	p50-08	[42666,44655]
p50-09	[43711,45183]	p50-10	[43575,45346]
p50-11	[43527,45132]	p50-12	[42808,44671]
p50-13	[43169,44872]	p50-14	[44519,46272]
p50-15	[44866,46479]	p50-16	[45310,46693]
p50-17	[46011,47751]	p50-18	[46897,48152]
p50-19	[47212,49162]	p50-20	[46779,48155]

Table 12 Bounds for RandB problems

Problem	Bounds	Problem	Bounds
r100a2	145270	r100b2	143271
r100c2	141702	r100d2	142630
r100e2	147416	r150a0	360978
r150a1	349251	r150b0	367635
r150b1	347627	r150c0	363895
r150c1	346492	r150d0	363180
r150d1	348902	r150e0	367181
r150e1	349910	r200a0	654604
r200a1	616399	r200b0	651237
r200b1	622112	r200c0	657441
r200c1	611956	r200d0	654375
r200d1	616617	r200e0	645207
r200e1	611306	r250a0	1019120
r250b0	1013737	r250c0	1010961
r250d0	1015041	r250e0	1008267

Table 13 Bounds for MB problems

Problem	Bounds	Problem	Bounds
atp24	172	atp48	483
atp66	761	atp76	934
atp111	[1322,1474]	atp134	[1605,1834]
atp163	[1870,2237]	atp452	[2506,2950]
econ36	541580	econ43	659558
econ47	828816	econ58	1205330
econ59	1168566	econ61	1170998
econ62	1195365	econ64	1216375
econ67	1339163	econ68	1395785
econ71	1480366	econ72	1738510
econ73	1945719	econ76	2516430
econ77	2550957	ex1	449
ex2	441	ex3	438
ex4	390	ex5	405
ex6	395	pal11	35
pal19	107	pal23	161
pal27	252	pal31	[285,300]
pal43	[543,597]	pal55	[1045,1084]

Table 14 Bounds for Spec problems

Problem	Bounds	Problem	Bounds
be75eec_150	[3482828,3527035]	be75np_150	[7174325,7317546]
be75oi_150	[2246534,2259482]	be75tot_150	[12287707,12509023]
stabu1_150	[2875732,2923697]	stabu2_150	[4327538,4398662]
stabu3_150	[4510445,4582377]	t59b11xx_150	[3239550,3298634]
t59d11xx_150	[1462697,1500324]	t59f11xx_150	[1543733,1578719]
t59n11xx_150	[318951,323636]	t65b11xx_150	[6453836,6547627]
t65d11xx_150	[3559347,3646934]	t65f11xx_150	[3159326,3231148]
t65l11xx_150	[253396,255390]	t65n11xx_150	[550733,558953]
t69r11xx_150	[11855957,12021166]	t70b11xx_150	[9647229,9802850]
t70d11xx_150	[5825147,5956793]	t70d11xx_150	[6174178,6305710]
t70f11xx_150	[5150097,5285191]	t70l11xx_150	[436862,438087]
t70n11xx_150	[948913,959803]	t74d11xx_150	[9396044,9601749]
t75d11xx_150	[9642140,9850135]	t75e11xx_150	[41570193,42053463]
t75k11xx_150	[1541596,1569847]	t75n11xx_150	[1743094,1767716]
tiw56n54_150	[837945,857599]	tiw56n58_150	[1155392,1183448]
tiw56n62_150	[1626921,1668859]	tiw56n66_150	[2107619,2160391]
tiw56n67_150	[2372926,2422085]	tiw56n72_150	[4135907,4222250]
tiw56r54_150	[958139,979551]	tiw56r58_150	[1219295,1248461]
tiw56r66_150	[1940755,1988549]	tiw56r67_150	[2056347,2107042]
tiw56r72_150	[2823758,2888328]	be75eec_250	[8893533,9150239]
be75np_250	[17814072,18473322]	be75oi_250	[5910096,5978555]
be75tot_250	[30958609,32055676]	stabu1_250	[7742444,8012535]
stabu2_250	[11500448,11898812]	stabu3_250	[11900315,12298263]
t59b11xx_250	[8411910,8708488]	t59d11xx_250	[3841167,4015773]
t59f11xx_250	[3993006,4162626]	t59n11xx_250	[825608,854352]
t65b11xx_250	[17273996,17751127]	t65d11xx_250	[9351696,9745769]
t65f11xx_250	[8409567,8739670]	t65l11xx_250	[666254,679527]
t65n11xx_250	[1429362,1475116]	t69r11xx_250	[31815927,32688871]
t70b11xx_250	[25403045,26133557]	t70d11xx_250	[15207063,15833337]
t70d11xx_250	[16040574,16658483]	t70f11xx_250	[13586135,14178278]
t70l11xx_250	[1113128,1132769]	t70n11xx_250	[2444390,2515339]
t74d11xx_250	[24430250,25350257]	t75d11xx_250	[25017059,25972723]
t75e11xx_250	[106873878,110318580]	t75k11xx_250	[4093849,4249417]
t75n11xx_250	[4524942,4658704]	tiw56n54_250	[2098726,2182012]
tiw56n58_250	[2906657,3026780]	tiw56n62_250	[4142745,4317513]
tiw56n66_250	[5370404,5582715]	tiw56n67_250	[6324858,6536019]
tiw56n72_250	[11149706,11520848]	tiw56r54_250	[2387349,2484343]
tiw56r58_250	[3060323,3185528]	tiw56r66_250	[4948414,5147594]
tiw56r67_250	[5289658,5491577]	tiw56r72_250	[7451579,7727048]

Table 15 Bounds for XLOLIB problems