Metaheuristics for large-scale instances of the linear ordering problem

Celso S. Sakuraba[a], Débora P. Ronconi[a], Ernesto G. Birgin[b] and Mutsunori Yagiura[c]

[a]*Production Engineering Department, Polytechnic School, University of São Paulo*
Av. Prof. Almeida Prado, Trav. 2, No. 128 - Cidade Universitária, São Paulo 05508-070 Brazil
sakuraba@usp.br, dronconi@usp.br
[b]*Department of Computer Science, Institute of Mathematics and Statistics, University of São Paulo*
Rua do Matão, 1010, Cidade Universitária, São Paulo 05508-090 Brazil egbirgin@ime.usp.br
[c]*Department of Computer Science and Mathematical Informatics, Graduate School of Information Science, Nagoya University, Furocho, Chikusaku, Nagoya 464-8603, Japan*
yagiura@nagoya-u.jp

Abstract. This paper presents iterated local search and great deluge trajectory metaheuristics for the linear ordering problem (LOP). Both metaheuristics are based on the TREE local search method introduced in Sakuraba and Yagiura, 2010 (Efficient local search algorithms for the linear ordering problem, *International Transactions in Operational Research* 17, pp. 711–737) that is the only method ever applied to a set of large-sized instances that are in line with the scale of nowadays real applications. By providing diversification and intensification features, the introduced methods improve all best known solutions of the large-sized instances set. Extensive numerical experiments show that the introduced methods are capable of tackling sparse and dense large-scale instances with up to 8,000 vertices and 31,996,000 edges in a reasonable amount of time; while they also performs well in practice when compared with other state-of-the-art methods in a benchmark with small and medium-scale instances.

Keywords: Metaheuristics, iterated local search, great deluge, linear ordering problem, large-scale instances.

1 Introduction

The linear ordering problem (LOP) was first described by Chenery and Watanabe (1958) in the context of the economic input-output studies proposed by Leontief (1966). By using input-output matrices that represent the flow among production sectors of a certain region, it is possible to analyze the stability of the regions' economy. In order to make this analysis, the order of the rows and columns of the matrix must be rearranged in a way that maximizes the sum of the values above the diagonal of the matrix, where an identical permutation must be used to rearrange the rows and the columns. Then the sectors that correspond to bigger suppliers and consumers can be identified. The LOP can also be seen as a graph problem, as shown in the example of Figure 1. In the graph of Figure 1(b), each vertex v_i corresponds to the i-th row and column of the matrix of Figure 1(a), and the forward edges, represented above the vertices, correspond to the values in the upper triangle of the matrix. The equivalence of the two representations is immediate, e.g. by regarding the matrix as the adjacency matrix of the graph.

We formally define the LOP in its graph representation as follows: given a directed graph $G = (V, E)$ with vertices set V ($|V| = n$), edges set $E \subseteq V \times V$ ($|E| = m$), and a cost c_{uv} for each edge $(u, v) \in E$, find a permutation of the vertices that maximizes the total cost of the direct edges, i.e. edges directed from a vertex u to a vertex v with v being a vertex in a position after u in the permutation. The sum of the direct edges corresponds to the sum of the

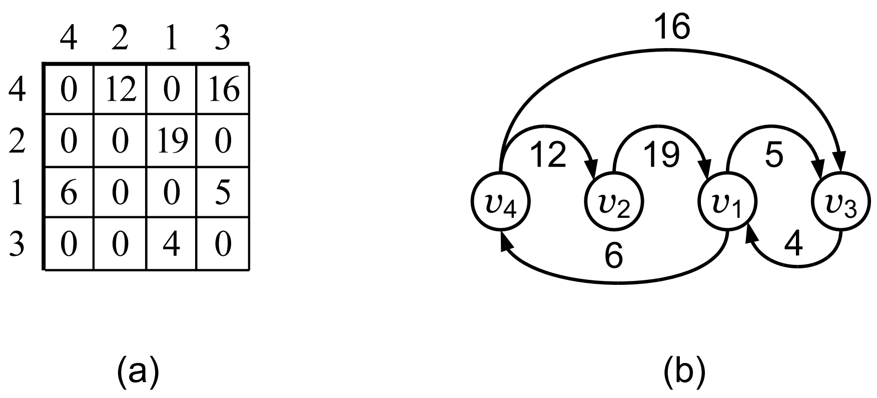

Figure 1: (a) Matrix and (b) graph representations of the same solution to an instance of the LOP.

values above the diagonal of the matrix. We assume without loss of generality that $c_{uv} > 0$ holds for all $(u, v) \in E$ and that if we regard G as an undirected graph, it is connected (which implies $m \geq n - 1$). For convenience, we also assume $c_{uv} = 0$ for all $(u, v) \notin E$. Denoting a permutation by $\pi : \{1, \ldots, n\} \to V$, where $\pi(i) = v$ means that v is the i-th element of π, the total cost of the direct edges is formally defined as follows:

$$cost(\pi) = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} c_{\pi(i)\pi(j)}. \tag{1}$$

Maximizing this sum is equivalent to minimizing the sum of the reverse edges (edges of the form (u, v) such that v appears before u in the permutation) or minimizing the sum of the elements below the matrix diagonal, which would give rise to the “min” formulation of the LOP. Throughout the remainder of this paper, we deal with the LOP formulation that maximizes the cost of the direct edges unless otherwise stated.

The LOP can be formulated by the following linear integer programming problem:

$$\text{Maximize} \quad \sum_{i \in V} \sum_{\substack{j \in V \\ j \neq i}} c_{ij} x_{ij} \tag{2}$$

$$\text{subject to} \quad x_{ij} + x_{ji} = 1 \qquad \forall\, i, j \in V, i \neq j \tag{3}$$

$$x_{ij} + x_{jk} + x_{ki} \leq 2 \quad \forall\, i, j, k \in V, i \neq j \neq k \neq i \tag{4}$$

$$x_{ij} \in \{0, 1\} \qquad \forall\, i, j \in V. \tag{5}$$

In the model (2–5), $x_{ij} = 1$ if edge (i, j) is a direct one and $x_{ij} = 0$ otherwise. The objective function (2) is equivalent to the sum of the cost of the direct edges (1). Constraints (3) and (4) assure that the set of direct edges does not have 2 and 3-dicycles, which also prevents k-dicycles for $k \geq 4$ (Grötschel et al., 1984, 1985).

Besides the economic context, the LOP has various real world applications in many different fields, such as the aggregation of individual preferences in marketing, and the most probable chronological ordering of potteries in archeology (Grötschel et al., 1984), among others. In the sports domain, instances of the LOP corresponding to point differential matrices of 347 teams in NCAA college basketball are described in Sukegawara et al. (2011), while data corresponding to the results of ATP tennis tournaments with up to 452 players are available at http://www.optsicom.es/lolib/#instances. The increasing amount of information available nowadays and the need for solutions that integrate more levels of the production chain demand solutions for problems of a different scale. In fact, very large-scale instances of several classic problems, as for example instances of the set covering problem with more than a million columns and several thousand rows, are available at the OR-Library. Analysing instances of the LOP containing information of hundreds of sectors or classifying data from a pool with thousands of alternatives are far from being unrealistic situations. Therefore there is an imminent need for algorithms capable of dealing with large-scale instances.

Known to be an NP-hard problem (Karp, 1972; Garey and Johnson, 1979) (the proof of NP-completeness was given for the feedback arc set problem, which is equivalent to the LOP), the LOP has been extensively studied in the literature and many exact and heuristic methods have been proposed to solve it. Some results, conjectures and open problems dealing with the combinatorial and algorithmic aspects of the LOP are analyzed in Charon and Hudry (2007). It is possible to solve instances of the LOP with up to seventy-five vertices using its MIP formulation and the commercial solver CPLEX. However, instances with hundreds of vertices or more were handled in the literature so far by using heuristic and metaheuristic approaches. Next we provide a brief description of some relevant elements of these methods.

Chanas and Kobylanski (1996) suggested a multi-start algorithm (CK) that utilizes the symmetric property of LOP, i.e. if the permutation $\pi = (\pi(1), \pi(2), \ldots, \pi(n))$ is an optimal solution to the maximization problem (2–5) then $(\pi(n), \pi(n-1), \ldots, \pi(1))$ minimizes the objective function (2). The authors applied a local search based on an insertion mechanism and, once a local optimal solution is found, the process is re-started from its reverse permutation. In Laguna et al. (1999) the insertion move is also considered within four different variants of the tabu search method: (i) a basic procedure that alternates between an intensification and a diversification phase (TS); (ii) TS associated with path relinking; (iii) TS and a long-term diversification, and (iv) TS with path relinking and a long-term diversification. Computational tests suggested that version (iv) outperforms the CK algorithm. It should be pointed out that the long-term diversification was inspired by the reverse operation developed by Chanas and Kobylanski (1996). A scatter search algorithm was proposed in Campos et al. (2001). It uses a frequency-based memory to guide the construction of a diversified initial reference set improved by the best neighborhood search developed in Laguna et al. (1999). Furthermore, a specific solutions combination method was developed for the LOP problem through the use of a min-max rule. A study about variants of the variable neighborhood search (VNS) to solve the LOP problem was carried in Garcia et al. (2006). A hybrid version of VNS, where the local search is replaced with the short-term tabu search proposed in Laguna et al. (1999), is also analyzed in Garcia et al. (2006). In Huang and Lim (2003) and Schiavinotto and Stützle (2004) genetic algorithms are coupled with local searches. For the generation of new solutions Huang and Lim (2003) applied classical crossover and mutation operators associated with a local search that uses the insertion move and a first-fit search strategy. In the memetic algorithm (MA) proposed by Schiavinotto and Stützle (2004), where some additional crossover operators were evaluated, numerical results pointed out that the classical operators (cycle and order-based) performed best. Schiavinotto and Stützle (2004) also presented an iterated local-search (ILS) algorithm whose main components are: the perturbation scheme (based on interchange moves), the local search (ls_{f}), and the acceptance criterion (automatic tuning process). A common component of both proposed methods is the use of the ls_{f} procedure that uses the insertion move with the best search strategy and is able to evaluate the insert neighborhood in $O(n^2)$.

More recently, Kröemer et at. (2013) proposed a bio-inspired metaheuristic based on artificial immune systems for the linear ordering problem. The proposed method consists in a modified

B-cell algorithm based on clonal selection and it is quite similar to a GA method. A population of candidate solutions evolves by cloning, hypermutation, and selection. Numerical experiments, considering a library of small instances with known optimal solutions named LOLIB and comparing the introduced method with a GA and a differential evolution (DE) metaheuristic, are presented. In Ceberio et al. (2013), the authors investigate instances' features that can provide useful insights into the difficulties of tackling the problem by applying local procedures associated with the insert neighbourhood. Ye et al. (2014) presents a multi-parent memetic algorithm that integrates a multi-parent recombination operator for generating offspring solutions and a distance-and-quality based criterion for the pool updating.

A comprehensive survey about existing metaheuristic approaches for the LOP problem was presented in Martí et al. (2012), where computational experiments are reported. A comparison among the existing methodologies indicates that MA is the best method followed by ILS. The authors suggest that this good behavior may be due to the efficient implementation of the MA local search that, by calculating each neighbour solution in constant time, reduces the total cost of evaluating the insert neighborhood to $O(n^2)$. Moreover, the authors highlight that this local-search move complexity issue makes a significant difference in the overall performance of local-search methods and on metaheuristics that use the local-search strategies in their composition.

In this paper, we develop metaheuristic algorithms based on the TREE local-search strategy recently introduced in Sakuraba and Yagiura (2010), whose time complexity for considering the whole neighbourhood of a given solution is $O(n + \Delta \log \Delta)$, where Δ is the maximum degree of the graph. Computational experiments presented in Sakuraba and Yagiura (2010) show that the TREE local search outperforms the ls_{f} local-search strategy introduced in Schiavinotto and Stützle (2004) on large-scale instances. Since the metaheuristic identified in Martí et al. (2012) as the one that presents the best performance over a set of ten evaluated metaheuristics is based on the ls_{f} local-search strategy, embedding the TREE local-search strategy within a metaheuristic framework sounds as a promising method to tackle large-scale instances of the LOP.

We consider two metaheuristic algorithms to deal with large-scale instances of the LOP: (a) an iterated local-search algorithm (ILS) and (b) a great deluge algorithm (GDA). Iterated local search is a metaheuristic framework that obtained good results for a variety of problems, and, according to Schiavinotto and Stützle (2003), it is a good candidate to handle the LOP. Johnson and McGeoch (1997) suggested that the great deluge algorithm is also a good candidate among variants of simulated annealing, despite its lack of popularity if compared to other metaheuristics. This recommendation is based on the results of computational experiments on the traveling salesman problem, for which the great deluge algorithm obtained better results than the threshold accepting method (Dueck and Scheuer, 1990). ILS and GDA metaheuristics share the characteristic of handling only one solution at a time. This type of metaheuristics is classified as *trajectory metaheuristics*, in contrast to the *populational-based* ones (Blum and Roli, 2003). The solution is iteratively modified and two successive solutions are always neighbors one to the other. This feature allows us to combine the aforementioned metaheuristics with the TREE local-search algorithm in an efficient way, since TREE is based on neighborhood search and its data structure would require a considerable amount of memory space if many solutions have to be kept.

The rest of this work is organized as follows. The TREE local-search strategy, the iterated local search and the great deluge metaheuristics are briefly described in Section 2. Section 3 is devoted to numerical experiments and the discussion of the numerical results. The last section summarizes the outcomes of this work.

2 Proposed Algorithms

In this section, the TREE local search, the iterated local-search metaheuristic, and the great deluge metaheuristic are described. To simplify the presentation, and to be in accordance with the vast optimization literature, methods will be described as minimization methods. Should the reader find it convenient, he/she may consider the LOP formulation that minimizes the sum of the costs of the reverse edges, or, equivalently, minimizes the sum of the cost of elements below the diagonal of the matrix.

2.1 The TREE Algorithm

The TREE algorithm is an algorithm designed to efficiently perform the local search through the insert neighborhood of the LOP. The procedure starts from an initial solution π_{init} and repeatedly replaces it with the best solution in its neighborhood until no better solution is found. The insert neighborhood of the LOP consists of solutions obtainable by taking one vertex from a position i and inserting it after (resp., before) the vertex in position j for $i < j$ (resp., $i > j$). It is the most common neighborhood used for the LOP and it is also the one that presented the best results so far (Huang and Lim, 2003; Schiavinotto and Stützle, 2004). TREE is an algorithm that can perform the local search of large-scale instances of the LOP in reasonable time. This result is achieved by implementing the following operations efficiently:

- Find a vertex v that, when inserted in a given position, generates a solution π' with a cost smaller than the cost of the current solution π, or conclude that no such vertex exists, in $O(n)$ time.
- Find the position where v should be inserted to generate π' in $O(\log d_v)$ time, where d_v is the degree of v.
- Update the data structure related to the new solution π' in $O(d_v \log \Delta)$ time, where Δ is the maximum degree of the graph.

To process these operations, a tree with $O(d_v)$ leaves like the one illustrated in Figure 2 is built for each vertex v. Each leaf of the tree corresponds to a position where v could be inserted with a different set of reverse edges connected to it. An index v_{name} represented on the left side of each node x of the tree is used to find the position of each leaf. The value $\gamma(x)$ represented in the bottom part of each node, when summed up from a leaf l through the path to the root, corresponds to the cost of vertex v in a solution where v is inserted in a position corresponding to l. We define the cost $cost(v, \pi)$ of v in a permutation π as the sum of the reverse edges in π connected to v. For instance, if we take the bottom value of the leaf between vertices v_4 and v_8 in Figure 2, and sum it up with the bottom values of its ancestors, we obtain the value of $148 + (-53) + 23 = 118$, which is the sum of the reverse edges connected to v_2 if we insert v_2 in the gap between v_4 and v_8. The value in the top part of each node, $\gamma_{\min}(x)$, represents the minimum sum of the values of γ in the nodes among all paths between x and the leaves in the subtree that has x as its root. The value $\gamma_{\min}(r)$, where r is the root of the tree, represents the minimum cost v can have if inserted in an appropriate position of the permutation π. From the definitions above, we can obtain a solution with a cost smaller than the current one (π) if there is a vertex v with $\gamma_{\min}(v) < cost(v, \pi)$. A complete description of the operations performed by the TREE algorithm is given in Sakuraba and Yagiura (2010).

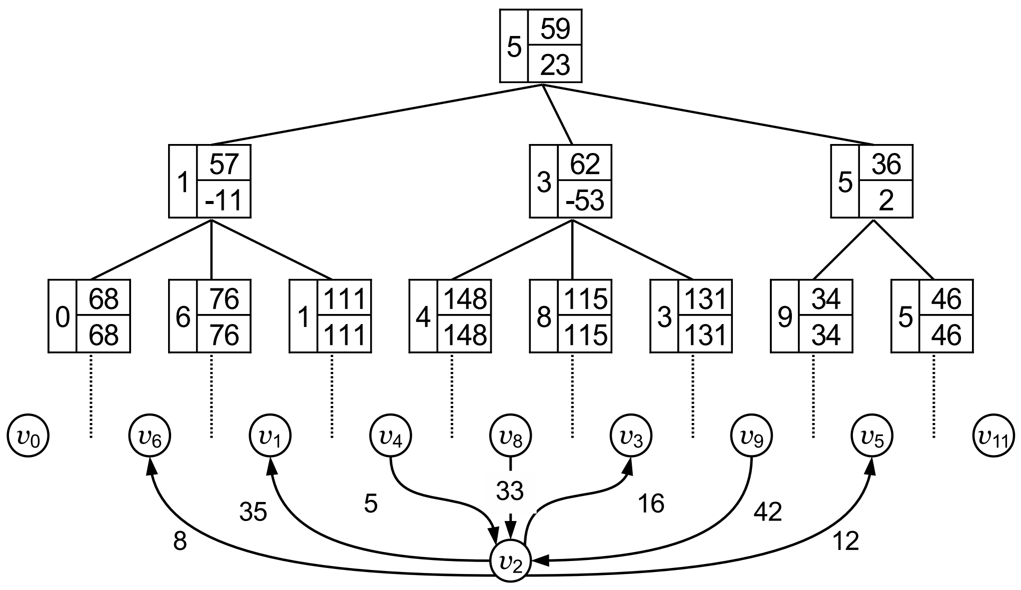

Figure 2: Example of a tree for vertex v_2 and the vertices connected to v_2 with their costs, in a graph with $n = 10$. The value 59 in the root represents the minimum cost v_2 can have if inserted in a certain position of the current permutation.

2.2 Iterated Local Search

Iterated local search is a metaheuristic framework introduced by Baxter (1981). Its general structure can be described in three steps. First, generate an initial solution and apply a local search to obtain a local optimal solution s^*. Then, add a perturbation to s^* and apply a local search again, obtaining $\hat{s}$. If an acceptance criterion is satisfied, set $s^* \leftarrow \hat{s}$. Repeat the last step until a stopping criterion is satisfied. If deterministic perturbations are used, a history of solutions can be saved to avoid short cycles (Lourenço et al., 2003). Algorithm 1 describes the framework of the iterated local search.

Algorithm 1 Framework of the iterated local search

```
1: $s_0$ := GenerateInitialSolution
2: $s^*$ := LocalSearch($s_0$)
3: while stopping criteria are not satisfied do
4:     $s'$ := Perturbation($s^*$, history)
5:     $\hat{s}$ := LocalSearch($s'$)
6:     $s^*$ := AcceptanceCriteria($s^*$, $\hat{s}$, history)
7: end while
```

To apply the framework of ILS to the LOP, we need to define functions *GenerateInitial-Solution*, *Perturbation* and *AcceptanceCriteria*. The *LocalSearch* function corresponds to the iterative use of the TREE algorithm until a local optimal solution is found. To generate the initial solution, we use the constructive heuristic proposed by Becker (1967). This method was selected since it provides a good trade-off between solution quality and execution time. The algorithm calculates a quotient for each vertex, which is given by the sum of the costs of outgoing edges from the vertex divided by that of incoming edges. The solution is obtained by ordering the vertices in non-increasing order of the quotients. For *Perturbation* we use k insert operations choosing vertices v randomly, where k is a parameter. To determine the position in which v is inserted, we walk the tree of v from its root to a leaf looking at the value of $\gamma_{\min}(x)$. In a local search, we walk the tree from its root to a leaf choosing the children that has the minimum value of $\gamma_{\min}$. In a perturbation, we do the opposite and walk the tree from its root to a leaf choosing the children that has the maximum value of $\gamma_{\min}$ in order to add diversity to the obtained solutions. (As an example, in the tree of Figure 2, a perturbation would go from the root to the node with $\gamma_{\min}(x) = 62$ and then to the node with $\gamma_{\min}(x) = 148$; while the local search would go from the root to the node with $\gamma_{\min}(x) = 36$ and then to the node with $\gamma_{\min}(x) = 34$.) As *AcceptanceCriteria*, we choose to always accept the obtained local optimal solution (i.e. $s^* := \hat{s}$ at line 6 of Algorithm 1), to avoid the computational effort that would be required to rebuild or keep the data structure corresponding to s^* in the case of rejection of $\hat{s}$.

2.3 Great Deluge Algorithm

The great deluge algorithm (Dueck, 1993) for maximization problems can be explained as follows: find the highest point in a given surface where it rains continuously, by walking through it without stepping into the water. The equivalent interpretation for minimization problems can be thought as a fish in a lake where it doesn't rain for long periods of time. As the water evaporates, the fish swims through the lake looking for deeper places where it can still survive under the water. To be more precise, the GDA (for minimization problems) works as follows: generate an initial solution s and set an initial value of $waterlevel$ such that $waterlevel > cost(s)$. Then, add a small perturbation to s to obtain s' and, if $cost(s') < waterlevel$, replace s with s' and reduce $waterlevel$ by dry, where $dry > 0$ is the amount of evaporation that is decided at each iteration. Repeat it until a stopping criterion is satisfied. GDA's framework for minimization problems is described below.

Algorithm 2 Framework of the great deluge algorithm

```
1:  $s$ := *GenerateInitialSolution*
2:  set $waterlevel$
3:  **while** stopping criteria are not satisfied **do**
4:      $s'$ := *Perturbation*($s$)
5:      **if** $cost(s') < waterlevel$ **then**
6:          $s := s'$
7:          decide the value of $dry$
8:          $waterlevel := waterlevel - dry$
9:      **end if**
10:     **if** too many iterations without updating $s$ **then**
11:         reset $waterlevel$
12:     **end if**
13: **end while**
```

In the implementation of GDA, we consider the constructive heuristic proposed by Becker (1967), described in Subsection 2.2, to generate the initial solution. The initial $waterlevel$ is set to the value obtained by multiplying the cost of the initial solution by a parameter $iwl > 1$. The value of dry is calculated at each iteration by multiplying the difference between the current $waterlevel$ and the $cost(s)$ by parameter $dryratio$ ($0 < dryratio < 1$).

The perturbation and solution acceptance steps (lines 4 to 6 of Algorithm 2) were implemented in a slightly different way from the one described in the general framework: our procedure always finds a solution that satisfies the decreasing condition at line 5. That is, there is no iteration in which a solution is generated, evaluated and discarded because it does not satisfy the condition at line 5. This makes the algorithm efficient and is one of the merits of incorporating the TREE data structure into the GDA. The procedure that achieves this is now explained. A vertex v is chosen randomly and the tree for v is used to determine the position in which it should be inserted. This position corresponds to the leaf of the tree for v found by the following procedure:

1. Set $e := waterlevel - cost(s)$, $x := r$ and $parcost := \gamma(r)$, where r is the root of the tree for v.

2. Let $C_{wl}(x)$ be the set of children y of x such that the subtree rooted at y has at least one leaf corresponding to a position where v could be inserted so that the obtained solution has a cost smaller than or equal to $waterlevel$, i.e. $C_{wl}(x) = \{y \in C(x) \mid \gamma_{\min}(y) + parcost - cost(v, \pi) \leq e\}$, where $C(x)$ is the set of children of x.

3. Randomly choose a node $y' \in C_{wl}(x)$ and set $x := y'$ and $parcost := parcost + \gamma(y')$.

4. Repeat steps 2 and 3 above until x becomes a leaf and then insert v into the position corresponding to x.

Through this procedure, we can always find a leaf corresponding to a position to insert vertex v so that the resulting solution satisfies the decreasing condition at line 5 of Algorithm 2. Note that the procedure may return a position that keeps v in its current position. Even if this situation happens, $waterlevel$ is updated at line 8 of Algorithm 2.

The resetting of the $waterlevel$ value at lines 10-12 of Algorithm 2 is optional, but essential for the algorithm to have a good performance if it runs for a long time. Otherwise, when the difference between $waterlevel$ and $cost(s)$ becomes too small, GDA behaves like a local search and stops updating the solution after a local optimal solution is reached. In our implementation, the reset operation is applied when n iterations occur without an updating of the current solution s, indicating that a local optimal solution was probably reached. In a reset operation, $waterlevel$ is reset to $iwl \times cost(s)$, where iwl is the same parameter used to set the initial value of $waterlevel$ as iwl times the cost of the initial solution.

3 Numerical Experiments

Numerical experiments are divided into three phases. Initially, preliminary numerical experiments are conducted on a reduced set of instances to calibrate the algorithmic parameters of the proposed metaheuristics. Then, a comparison with state-of-the-art methods from the literature is performed considering a known benchmark with small and medium-scale instances. Finally, the application of the proposed methods to large-scale instances of the LOP is illustrated and its performance analyzed. The proposed ILS and GDA metaheuristics that uses the TREE local-search method will be called ILS-TREE and GDA-TREE from now on, respectively. All codes were written in C language.

3.1 Preliminary experiments

Numerical experiments in this subsection were run on an Intel Xeon 3.0 GHz processor with 8GB of RAM memory. Aiming to select the parameters of the presented algorithms, in this first set of experiments, we considered representative benchmark instances[1] classified by their size: *(i)* small-scale instances (named LOLIB and SGB) and *(ii)* medium-scale instances (named Random I and Random II). A succinct description of the instances sets follows:

Small scale: 74 instances with data related to real world input-output matrices with up to 75 production sectors. Instances are divided into the following two groups:

- **LOLIB:** 49 instances corresponding to European input-output matrices with $44 \leq n \leq 60$ and $285 \leq m \leq 1,556$.
- **SGB:** 25 instances with $n = 75$ and $2,505 \leq m \leq 2,623$ corresponding to American input-output matrices.

Medium scale: 150 instances with $n \in \{100, 150, 200\}$ and $m \approx n^2/2$ described in Campos et al. (2001). Instances are classified into the following two groups:

- **Random I:** 75 instances with costs generated from the discrete uniform distribution in the interval $[0, 100]$.
- **Random II:** 75 instances generated by counting the number of times an element appears in a higher position than another in a set of randomly generated permutations.

Optimal solutions for all small-scale instances are known. For medium-scale instances, the results presented in Garcia et al. (2006) were used as a basis of comparison (henceforth called reference solutions).

In this preliminary test, a CPU time limit of $t = m/(2n)$ seconds was considered, which is approximately the same time used for the state-of-the-art metaheuristics to solve medium-scale instances. Table 1 shows the experimental results of running ILS-TREE varying the number of perturbations $k \in \{5, 10, 20, n/2, m/n\}$. In the table, the first two columns identify the instances' sets and the number of instances per set. Instances in sets Random I and Random II are grouped by their number of vertices n (showed within parenthesis). Each pair of columns named "B/E" and "RD" shows the results of ILS-TREE with a specified number of perturbations k. For a given k, "B/E" contains the number B of solutions found that were better than the corresponding reference solution and the number E of solutions found that were equal to the corresponding optimal or reference solution. Column "RD" contains the average of the percentage relative differences from the optimal or reference solutions, where the average is taken over all the instances in the group. For each instance, the percentage relative difference from the optimal or reference solution is given by

$$\text{RD} = 100\ (f_{MH} - f_{RF})/f_{RF},$$

where f_{MH} and f_{RF} are the objective function value found by the metaheuristic being evaluated and the optimal or reference objective function value, respectively. Note that, since we are dealing with a maximization problem, a positive value of RD means that the proposed algorithm obtained better results. The last row in the table contains, for each value of parameter k, the totalized number of improved or attained solutions $B + E$ and the average RD for the whole set of 224 considered instances.

[1]The instances used in the experiments are available online at http://www.optsicom.es/#lolib/instances.

Table 1: Summary of the performance of ILS-TREE varying the value of parameter k in a set of small and medium-scale instances with a CPU time limit $t = m/(2n)$ seconds.

Instances				ILS-TREE									
				$k=5$		$k=10$		$k=20$		$k=n/2$		$k=m/n$	
	Set name and # Inst			B/E	RD	B/E	RD	B/E	RD	B/E	RD	B/E	RD
Small	LOLIB		49	0/46	-0.0027	0/48	-0.0015	0/49	0.0000	0/49	0.0000	0/49	0.0000
	SGB		25	0/22	-0.0020	0/24	0.0000	0/24	0.0000	0/23	0.0000	0/23	0.0000
Medium	Random I	(100)	25	3/0	-0.1073	4/4	-0.0472	8/3	-0.0104	15/4	0.0118	15/4	0.0112
		(150)	25	0/0	-0.1588	0/0	-0.1224	6/0	-0.0469	19/0	0.0257	21/0	0.0270
		(200)	25	0/0	-0.1755	0/0	-0.1191	4/0	-0.0741	24/0	0.0498	21/0	0.0386
	Random II	(100)	25	0/16	-0.0014	0/19	-0.0005	0/24	-0.0001	0/17	-0.0005	0/19	-0.0004
		(150)	25	1/0	-0.0022	0/1	-0.0017	3/3	-0.0004	0/0	-0.0011	0/3	-0.0010
		(200)	25	0/0	-0.0036	1/0	-0.0023	2/2	-0.0010	1/0	-0.0015	1/0	-0.0011
B+E and average RD				88	-0.0509	101	-0.0331	128	-0.0148	152	0.0094	156	0.0083

We can see from Table 1 that optimal solutions were found for at least 72 out of the 74 small instances for any value of k other than 5. It was also possible to improve the reference solutions in 58 and 57 out of the 75 medium-scale instances in Random I with $k = n/2$ and $k = m/n$, respectively. For the medium-scale instances in Random II, the number of improved solutions was small, but the quality of the obtained solutions is very similar to the quality of the reference solutions (i.e. very small values of RD). In general, it is easy to see that, for the considered instances, the ILS-TREE method with $k = n/2$ and $k = m/n$ presented the best results.

Table 2 shows the results of the preliminary tests with the GDA-TREE for all combinations of $dryratio \in \{0.1, 0.3, 0.5, 0.7, 0.9\}$ and $iwl \in \{1.1, 1.3, 1.5\}$. We can observe from Table 2 that, for the small-scale instances, GDA-TREE with $(iwl, dryratio) = (1.3, 0.7)$ and $(iwl, dryratio) = (1.5, 0.7)$ was able to find the largest number of optimal solutions (70 out of 74 for both combinations of parameters). Ranked in second place in terms of the number of optimal solutions found is GDA-TREE with parameters $(iwl, dryratio) = (1.3, 0.3)$ and $(iwl, dryratio) = (1.5, 0.3)$, that found 69 optimal solution out of 74. Considering set Random I, the combinations of parameters given by $(iwl, dryratio) = (1.3, 0.3)$ and $(iwl, dryratio) = (1.5, 0.3)$ also showed good performances, improving 35 reference solutions out of 75. However, the largest number of improved reference solutions within set Random I was given by combination $(iwl, dryratio) = (1.1, 0.1)$ that improved 39 reference solutions. The performance of GDA-TREE with different combinations of parameters is hard to determine for the Random II instances. However, it is worth noting that the largest number of attained or improved solutions (within set Random II) was obtained by combination $(iwl, dryratio) = (1.1, 0.9)$. The last part of the table shows, for each combination of parameters iwl and $dryratio$, the totalized number of improved or attained solutions $B + E$ and the average RD for the whole set of 224 considered instances. All combinations present a very similar behavior. This appears as a drawback of the GDA-TREE method when compared to the ILS-TREE approach, for which calibrating its solely parameter was a simple task. For the GDA-TREE algorithm, we arbitrarily considered, for the experiments in the following subsection, combination $(iwl, dryratio) = (1.1, 0.3)$, that was arbitrary selected based on a detailed analysis comparing the performance of ILS-TREE and GDA-TREE instance by instance.

Table 2: Summary of the performance of GDA-TREE varying the value of parameters $dryratio$ and iwl in a set of small and medium-scale instances with a CPU time limit $t = m/(2n)$ seconds.

Instances				GDA-TREE									
				$dryratio = 0.1$		$dryratio = 0.3$		$dryratio = 0.5$		$dryratio = 0.7$		$dryratio = 0.9$	
	Set name and # Inst		iwl	B/E	RD	B/E	RD	B/E	RD	B/E	RD	B/E	RD
Small	LOLIB/SGB	74	1.1	0/62	-0.0041	0/65	-0.0036	0/66	-0.0019	0/64	-0.0055	0/63	-0.0064
			1.3	0/66	0.0000	0/69	0.0000	0/68	-0.0017	0/70	0.0000	0/66	-0.0017
			1.5	0/60	0.0000	0/69	0.0000	0/66	-0.0006	0/70	0.0000	0/67	-0.0017
Medium	Random I	75	1.1	39/0	0.0001	33/2	0.0001	31/3	-0.0001	18/1	-0.0003	4/21	-0.0003
			1.3	18/0	-0.0002	35/0	0.0001	31/3	-0.0001	26/2	-0.0002	21/1	-0.0004
			1.5	17/0	-0.0002	35/0	0.0001	32/2	0.0000	26/2	-0.0002	18/3	-0.0004
	Random II	75	1.1	1/12	0.0000	0/22	-0.0001	2/27	0.0000	3/24	0.0000	3/27	0.0000
			1.3	0/5	0.0000	2/13	0.0000	2/15	0.0000	4/22	0.0000	1/24	0.0000
			1.5	0/5	0.0000	2/13	0.0000	1/13	0.0000	0/20	0.0000	3/22	0.0000
B+E and average RD			1.1	114	-0.0013	122	-0.0012	129	-0.0007	110	-0.0019	118	-0.0022
			1.3	89	-0.0001	119	0.0000	119	-0.0006	124	-0.0001	113	-0.0007
			1.5	82	-0.0001	119	0.0000	114	-0.0002	118	-0.0001	113	-0.0007

3.2 Comparative study with other methods of the literature

In the present subsection, we compare the ILS-TREE and GDA-TREE metaheuristics against the ten metaheuristics TS, MA, VNS, SA, SS, GRASP, ILS, GA, CKM, and KLM, whose performances are summarized in Tables 4–6 of Martí et al. (2012, pp.1311–1314). Numerical experiments in this subsection were run on an Intel Xeon X3363 2.83 GHz with 24 GB of RAM memory.

In a first set of experiments we considered set OPT-I composed of 229 instances that are instances with a known optimal solution. We run methods ILS-TREE with $k = m/n$ and GDA-TREE with $(iwl, dryratio) = (1.1, 0.3)$ considering a CPU time limit of $t = 10$ seconds. Table 3 summarizes the results. In the table, “Name” identifies the source subset from which the instances with a known optimal solution were extracted, “# Inst” is the number of instances in the subset, “# Opt” is the number of optimal solutions found by the proposed methods, “RD” is the average percentage relative difference from the optimal solution, and “RD non-opt” is the average percentage relative difference from the optimal solution disregarding the instances in which the optimal solution was found. Figures in the table show that ILS-TREE found an optimal solution in 192 instances, while GDA-TREE found an optimal solution in 175 cases. Moreover, the average percentage relative deviations from the known optimal value were -0.0013% and -0.0095% for ILS-TREE and GDA-TREE, respectively. Since ILS-TREE outperformed GDA-TREE (as already verified in the previous subsection), we compared ILS-TREE versus the ten methods whose performances are summarized in Table 4 of (Martí et al., 2012, p.1311). Figures in the table show that methods MA, ILS, TS, and VNS found 228, 228, 215, and 208 optimal solutions, respectively. ILS-TREE was capable of finding 192 optimal solutions, ranking fifth and outperforming methods GRASP, SS, KLM, CKM, SA, and GA that found 182, 165, 124, 93, 32, and 14 optimal solutions, respectively. When considering the average deviation from the optimal solution, the ILS-TREE presents the same results as the three top-ranked methods MA, ILS, and TS, that is strictly smaller than 0.005% (values are presented with two decimal places in Martí et al. (2012) and all average deviations smaller than 0.005 are reported as 0.00). At this point it is important to notice that numerical results being compared were run on different machines, having a direct effect on the stopping criterion based on CPU time. However, the benchmarks in http://www.spec.org allow us to conclude that both machines are "similar". Therefore, accompanied by every applicable caveat regarding machines, programming languages, compilers, and compiling options, we may roughly say that our analysis shows that the ILS-TREE method performs well when compared to the ten state-of-the-art methods being considered.

Table 3: Performances of ILS-TREE and GDA-TREE in the set of small and medium-scale instances named OPT-I composed of instances with a known optimal solution and considering a CPU time limit of 10 seconds.

Subset of instances		ILS-TREE			GDA-TREE		
Name	# Ins	# Opt	RD	RD non-opt	# Opt	RD	RD non-opt
IO	50	50	0.0000	–	46	-0.0253	-0.3164
SGB	25	23	-0.0000	-0.0005	19	-0.0009	-0.0036
Random A II	25	15	-0.0037	-0.0092	14	-0.0040	-0.0091
Random B	70	70	0.0000	–	70	0.0000	-0.0000
MB	30	18	-0.0006	-0.0015	12	-0.0009	-0.0014
Special	29	16	-0.0063	-0.0140	14	-0.0262	-0.0507
Total # Opt and average RD	229	192	-0.0013	-0.0079	175	-0.0095	-0.0402

In a second set of experiments, we considered set UB-I composed of 255 instances for which no optimal solutions are known. For each instance an upper and a lower bound are known. The upper bound of each instance was obtained by solving an LP-relaxation (remember that we are dealing with a maximization problem). The lower bound of each instance was obtained by running some metaheuristics with a large CPU time limit (see Martí et al. (2012) for details). We run methods ILS-TREE with $k = m/n$ and GDA-TREE with $(iwl, dryratio) = (1.1, 0.3)$ considering a CPU time limit of ten seconds. Table 4 shows the results. In the table, "Name" identifies the source subset from which the instances were extracted, "# Inst" is the number of instances in the subset, "B/E" identifies the number B of solutions that improve the reference solution and the number E of solutions found that are equal to the best known solution (lower bound), "RD" is the average percentage deviation from the best known solution, and "RD non-att" is the average percentage deviation from the best known solution disregarding the instances in which the best known solution was attained. Figures in the table show that ILS-TREE and GDA-TREE attained the best known solution in 29 and 21 instances, respectively. The average percentage deviation from the best known values were -0.2709% and -0.5335%, for the ILS-TREE and GDA-TREE methods, respectively. Once again, the ILS-TREE performed better than GDA-TREE in the considered set of instances. Hence, we analyzed the performance of ILS-TREE in comparison with the ten methods whose performances are summarized in Table 5 of (Martí et al., 2012, p.1313). When considering both, the number of attained best known values and average percentage deviation from the best known values, ILS-TREE ranking fourth (with very similar results to the ones obtained by the TS method that ranking third) over the twelve considered metaheuristics.

In a final set of experiments, ILS-TREE and GDA-TREE were run with an increased CPU time limit $t = 600$ seconds. Table 5 shows the results. Note that, this time, a few reference solutions were improved. These results may be compared to the ones presented in Table 6 of (Martí et al., 2012, p.1314). Once again, similarly to the shorter CPU time limit situation, ILS-TREE ranked in fourth place (close to the third one) when considering the number of attained or improved reference solutions. If the average RD is considered, ILS-TREE ranking

Table 4: Performances of ILS-TREE and GDA-TREE in the set of instances named UB-I composed of instances with unknown optimal solution and considering a CPU time limit of 10 seconds.

Subset of instances		ILS-TREE			GDA-TREE		
Name	# Inst	B/E	RD	RD non-att	B/E	RD	RD non-att
Random A I	100	0/6	-0.1945	-0.2069	0/1	-0.7080	-0.7152
Random A II	50	0/0	-0.0157	-0.0157	0/0	-0.0159	-0.0159
Random B	20	0/20	0.0000	–	0/19	-0.0012	-0.0240
XLOLIB	78	0/0	-0.6117	-0.6117	0/0	-0.8008	-0.8008
Special	7	0/3	-0.1606	-0.2810	0/1	-0.2798	-0.3264
B+E and average RD	255	29	-0.2709	-0.3056	21	-0.5335	-0.5814

third, outperformed by MA and ILS and outperforming TS and the other seven metaheuristics being considered.

Table 5: Performances of ILS-TREE and GDA-TREE in the set of instances named UB-I composed of instances with unknown optimal solution with a CPU time limit of 600 seconds.

Subset of instances		ILS-TREE			GDA-TREE		
Name	# Inst	B/E	RD	RD non-att	B/E	RD	RD non-att
Random A I	100	2/25	-0.0345	-0.0479	0/14	-0.1015	-0.1180
Random A II	50	0/6	-0.0045	-0.0051	0/15	-0.0026	-0.0037
Random B	20	0/20	0.0000	–	0/20	0.0000	–
XLOLIB	78	0/0	-0.3871	-0.3871	0/0	-0.3056	-0.3056
Special	7	1/4	-0.0659	-0.2548	0/3	-0.1271	-0.2224
B+E and average RD	255	58	-0.1346	-0.1748	52	-0.1373	-0.1724

Summing up, the whole set of experiments performed in the present subsection allows us to claim, accompanied by every applicable caveat, that the ILS-TREE metaheuristic performance is comparable to the one reported for the top-ranked state-of-the-art methods considered in Martí et al. (2012).

3.3 Experiments with large-scale instances

Numerical experiments in this subsection were run on an Intel Xeon 3.0 GHz processor with 8GB of RAM memory. In this subsection, we evaluate the performance of ILS-TREE and GDA-TREE in a set of large-scale instances presented in Sakuraba and Yagiura (2010). The set is composed of 150 instances with $500 \leq n \leq 8000$ and $m \approx n(n-1)/2 \times (p/100)$, where $1 \leq p \leq 100$ is a "density" parameter such that each pair of vertices has an edge joining them with probability $p/100$. Edges' costs were randomly generated from the discrete uniform distribution in the interval $[1, 100]$. A full description of the instances is available online at http://www.al.cm.is.nagoya-u.ac.jp/~yagiura/lop/ for benckmark purposes. For each instance, best known solutions reported in Sakuraba and Yagiura (2010) were used as reference values for comparison. It should be stressed that, to the extent of our knowledge, no numerical experiments with other metaheuristics were reported in the literature for large-scale instances like the ones being considered in the present subsection.

Table 6 shows the experimental results of running ILS-TREE varying the number of perturbations $k \in \{5, 10, 20, n/2, m/n\}$ with a CPU time limit $t = m/(2n) \approx 0.0025np$ seconds. In the table, n is the number of nodes and $p/100$ is the density of the instances, while "# Inst" indicates the number of instances with n nodes and density $p/100$. In all cases, we have 5 instances of each type. For each group of 5 instances with the same n and p, the table shows the quantities B/E and RD (already described in the previous subsections). Reference solutions were improved for almost all instances with almost any value of parameter k. Therefore, the comparison must be focused on the average percentage deviations from the reference solutions. In the table, the largest (positive) average percentage deviations for each combination (n, p) are shown in bold. It is easy to see that, once again, the best performance was obtained considering $k = n/2$, followed by $k = m/n$. ILS-TREE with $k = n/2$ improved the reference solutions for all instances, regardless of their size and density, while ILS-TREE with $k = m/n$ improved 149 out of the 150 reference solutions. The larger the instances' size and density, the smaller the average percentage relative improvement, suggesting that solutions found on larger and denser instances may still be improved by increasing the CPU time limit of the metaheuristics. The only groups of instances in which better results were obtained by ILS-TREE with values of k other than $n/2$ and m/n were the ones with $(n, p) = (500, 1)$, $(n, p) = (500, 5)$, $(n, p) = (1000, 1)$, and $(n, p) = (2000, 1)$, that correspond to small (within this set of large-scale instances) and sparse instances. It suggests that, although good results were obtained setting $k = n/2$ or $k = m/n$, even better results for some sparse instances can be obtained by tuning the value of k. Last but not least, note that $k = m/n$ presented the best results for almost all groups of complete graphs (note that $p = 100$ implies that graphs are completely dense). In this case, we have $m/n = (n-1)/2 = n/2 - 1$ (last equality holds because n is even for all instances) and this is why the cases with $k = n/2$ and $k = m/n$ presented almost identical results in the groups of dense-graphs instances. It is worth noting that the best performance of ILS-TREE was obtained by setting $k = n/2$, as it was the case with the small and medium-scale instances. It suggests that ILS-TREE is a robust method with respect to the value of its sole parameter, in the sense that the value selected in the preliminary tests is also the one that presents the best performance in this new set of instances.

Table 7 shows the experimental results of running GDA-TREE for all combinations of $dryratio \in \{0.1, 0.3, 0.5, 0.7, 0.9\}$ (named r in the table due to lack of space) and $iwl \in \{1.1, 1.3, 1.5\}$ with a CPU time limit $t = m/(2n) \approx 0.0025np$ seconds. In the table, instances are grouped by their density $p/100$ and "# Inst" indicates the number of instances in each group. For example, in a line starting with $p = 1$ there are 5 grouped instances with $n = 500$, 5 instances with $n = 1,000$, 5 instances with $n = 2,000$, 5 instances with $n = 3,000$, 5 instances with $n = 4,000$, and 5 instances with $n = 8,000$, totalizing 30 instances. For each combination of parameters iwl and $dryratio$, the table shows the values of "B/E" and "RD" (already described in previous tables). Figures in the table indicate that combination $(iwl, dryratio) = (1.1, 0.9)$ presented the largest average relative deviation, while the largest number of improved solutions was attained with combination $(iwl, dryratio) = (1.5, 0.7)$. Note that, the combinations of parameters with the best overall performance for the large-scale instances do not coincide with the ones obtained in the preliminary numerical experiments of Subsection 3.1 for small and medium-scale instances. Analyzing the average relative deviations, it can be seen that negative values are found for every combination of parameters with $dryratio \in \{0.1, 0.3\}$, while positive average relative deviations are found for every combination of parameters with $dryratio \in \{0.5, 0.7, 0.9\}$. From this observation, it is possible to conclude that the value of parameter $dryratio$ has a larger influence on the GDA-TREE performance.

In a final numerical experiment, GDA-TREE was run with a refined combination of parameters. We considered values of iwl and $dryratio$ around 1.1 and 0.9, respectively, because this

Table 6: Performance of ILS-TREE in the set of large-scale instances considering a CPU time limit $t = m/(2n) \approx 0.0025np$ seconds.

Instances			$k = 5$		$k = 10$		$k = 20$		$k = n/2$		$k = m/n$	
n	p	# Inst	B/E	RD	B/E	RD	B/E	RD	B/E	RD	B/E	RD
	1	5	5/0	3.9036	5/0	**3.9577**	5/0	3.8220	5/0	2.5259	5/0	3.6956
	5	5	5/0	2.2932	5/0	2.4961	5/0	**2.5543**	5/0	2.2769	5/0	2.4816
500	10	5	5/0	1.5625	5/0	1.7363	5/0	1.8163	5/0	**1.9829**	5/0	1.9167
	50	5	4/0	0.1213	5/0	0.1348	5/0	0.2475	5/0	**0.6338**	5/0	0.5309
	100	5	5/0	0.0613	5/0	0.0799	5/0	0.0997	5/0	**0.2316**	5/0	0.2246
	1	5	5/0	3.2116	5/0	3.2335	5/0	**3.3181**	5/0	2.1860	5/0	3.0717
	5	5	5/0	1.0396	5/0	1.2411	5/0	1.3812	5/0	**1.9348**	5/0	1.5158
1000	10	5	5/0	0.4153	5/0	0.5467	5/0	0.6422	5/0	**1.2638**	5/0	0.8056
	50	5	4/0	0.0297	4/0	0.0500	5/0	0.0810	5/0	**0.4353**	5/0	0.2924
	100	5	1/0	-0.0212	1/0	-0.0078	3/0	0.0088	5/0	0.1267	5/0	**0.1314**
	1	5	5/0	1.7134	5/0	1.9627	5/0	**2.1611**	5/0	2.1159	5/0	1.9251
	5	5	5/0	0.4351	5/0	0.5095	5/0	0.5636	5/0	**1.2714**	5/0	0.6682
2000	10	5	5/0	0.1323	5/0	0.1807	5/0	0.2258	5/0	**0.8087**	5/0	0.3639
	50	5	2/0	-0.0209	2/0	-0.0143	2/0	-0.0025	5/0	**0.2310**	5/0	0.1108
	100	5	0/0	-0.0179	1/0	-0.0159	1/0	-0.0102	5/0	0.0611	5/0	**0.0631**
	1	5	5/0	1.1155	5/0	1.1522	5/0	1.2474	5/0	**1.8203**	5/0	1.2086
	5	5	5/0	0.2246	5/0	0.2210	5/0	0.2392	5/0	**0.8720**	5/0	0.3512
3000	10	5	5/0	0.0702	5/0	0.0779	5/0	0.1143	5/0	**0.5450**	5/0	0.1972
	50	5	2/0	-0.0067	2/0	-0.0022	2/0	0.0035	5/0	**0.1615**	5/0	0.0762
	100	5	2/0	-0.0014	3/0	0.0005	2/0	0.0005	5/0	0.0445	5/0	**0.0446**
	1	5	5/0	0.8546	5/0	0.9040	5/0	0.9400	5/0	**1.6238**	5/0	0.9291
	5	5	5/0	0.1766	5/0	0.1672	5/0	0.1856	5/0	**0.6801**	5/0	0.2655
4000	10	5	4/0	0.0615	5/0	0.0589	5/0	0.0802	5/0	**0.4196**	5/0	0.1543
	50	5	3/0	-0.0004	3/0	0.0024	3/0	0.0030	5/0	**0.1200**	5/0	0.0504
	100	5	1/0	-0.0058	1/0	-0.0060	1/0	-0.0057	5/0	0.0283	5/0	**0.0292**
	1	5	5/0	0.3863	5/0	0.3802	5/0	0.3600	5/0	**0.8973**	5/0	0.3607
	5	5	5/0	0.0513	5/0	0.0591	5/0	0.0660	5/0	**0.3028**	5/0	0.0942
8000	10	5	1/0	-0.0068	1/0	-0.0023	3/0	-0.0010	5/0	**0.1404**	5/0	0.0286
	50	5	1/0	-0.0073	1/0	-0.0075	1/0	-0.0062	5/0	**0.0296**	4/0	0.0077
	100	5	3/0	-0.0001	3/0	0.0005	3/0	0.0009	5/0	0.0114	5/0	**0.0122**
B+E and average RD			113	0.5924	117	0.6366	121	0.6712	**150**	**0.8594**	149	0.7202

combination corresponds to the combination with the largest RD (see Table 7). Additionally, several combinations with $dryratio = 0.1$ were included since a detailed analysis showed that GDA-TREE with $dryratio = 0.1$ and different values of iwl found better solutions than the ones found by ILS-TREE in many cases. Table 8 shows the experimental results of running GDA-TREE for all combinations of $dryratio \in \{0.1, 0.9\}$ (named r in the table by lack of space) and $iwl \in \{1.01, 1.03, 1.05, 1.10\}$ with a CPU time limit $t = m/(2n) \approx 0.0025np$ seconds. In the table, n is the number of nodes and $p/100$ is the density of the instances, while "# Inst" indicates the number of instances with n nodes and density $p/100$. In all cases, we have 5 instances of each type. For each group of 5 instances with the same n and p, the table shows the quantities B/E and RD (already described in the previous subsections). It is very clear from the table that GDA-TREE does not have the nice property presented by ILS-TREE, since bold numbers (best RD for each group of 5 instances with same values of n and p) are distributed all over the table corresponding to different combinations of parameters iwl and $dryratio$. Moreover, some of the bold numbers are negative, indicating that for some of the

Table 7: Performance of GDA-TREE in the set of large-scale instances considering a CPU time limit $t = m/(2n) \approx 0.0025np$ seconds.

Instances		iwl	$r = 0.1$		$r = 0.3$		$r = 0.5$		$r = 0.7$		$r = 0.9$	
p	# Inst		B/E	RD	B/E	RD	B/E	RD	B/E	RD	B/E	RD
1	30		25/0	0.6968	25/0	1.7957	25/0	2.0175	25/0	1.8823	25/0	1.8898
5	30		15/0	-1.4959	25/0	0.1300	25/0	0.4550	25/0	0.4429	25/0	0.5335
10	30	1.1	15/0	-2.1626	20/0	-0.7319	20/0	0.0598	20/0	0.0463	24/0	0.1288
50	30		10/0	-1.9080	10/0	-1.0767	15/0	-0.3291	14/0	-0.2914	14/0	-0.1952
100	30		10/0	-0.6431	10/0	-0.3872	15/0	-0.1405	14/0	-0.1200	11/0	-0.0953
B+E and average RD			75	-1.1026	90	-0.0540	100	0.4126	98	0.3920	99	**0.4523**
1	30		25/0	0.7264	25/0	1.7854	25/0	1.8883	25/0	1.8868	25/0	1.8637
5	30		15/0	-1.6417	20/0	0.1386	25/0	0.3285	25/0	0.4338	25/0	0.5478
10	30	1.3	12/0	-2.3698	19/0	-0.7536	20/0	-0.0960	20/0	0.0592	20/0	0.1355
50	30		10/0	-2.0130	10/0	-1.1322	15/0	-0.4512	14/0	-0.3233	13/0	-0.2434
100	30		6/0	-0.6873	10/0	-0.4001	11/0	-0.1858	13/0	-0.1359	12/0	-0.1036
B+E and average RD			68	-1.1971	84	-0.0724	96	0.2968	97	0.3841	95	0.4400
1	30		25/0	0.6608	25/0	1.8025	25/0	1.9862	30/0	1.9271	25/0	1.8485
5	30		15/0	-1.8296	24/0	0.1619	25/0	0.4669	25/0	0.4734	24/0	0.5131
10	30	1.5	10/0	-2.5099	20/0	-0.7900	20/0	0.0139	22/0	-0.0330	19/0	0.1143
50	30		10/0	-2.0174	10/0	-1.1998	15/0	-0.4437	15/0	-0.3021	13/0	-0.2757
100	30		5/0	-0.7066	10/0	-0.4236	10/0	-0.1772	13/0	-0.1299	10/0	-0.1012
B+E and average RD			65	-1.2805	89	-0.0898	95	0.3692	**105**	0.3871	91	0.4198

groups (mostly the ones corresponding to dense instances with large values of n) none of the parameters' combinations make it possible for GDA-TREE to improve, on average, the reference solutions. The positive result is that GDA-TREE with $(iwl, dryratio) = (1.03, 0.9)$ presented an overall positive $RD = 0.4760$. But even in this case, this value is almost half of the best overall $RD = 0.8594$ presented by ILS-TREE with $k = n/2$.

4 Conclusions

In this work, we presented two metaheuristic algorithms for large-scale instances of the linear ordering problem: iterated local search (ILS-TREE) and great deluge algorithm (GDA-TREE). Those are the first metaheuristic methods ever applied to large-sized instances of the LOP problem in line with the scale of current real applications. Combined with the TREE local-search method, both algorithms are capable of dealing with sparse as well as complete graphs with several thousand vertices, obtaining good results in a reasonable amount of time. ILS-TREE was shown to be efficient and robust, obtaining the best overall performance. It is well-known that adjusting the several parameters of a given metaheuristic method is a difficult problem. In this sense, the introduced ILS-TREE metaheuristic presents an attractive feautre: its only parameter besides the CPU time limit is the number of perturbations k, and good results for all sets of instances were obtained by setting it to $k = n/2$, where n is the number of vertices of the graph. Numerical experiments showed that ILS-TREE presents competitive results for the benchmark small and medium-scale instances from the literaure, while it improves all the best known results for the large-scale instances with up to $8,000$ vertices and $31,996,000$ edges.

The TREE local search method introduced in Sakuraba and Yagiura (2010) appears as a promising local search strategy for large-sized instances of the LOP problem due to its low complexity neighborhood search. This state of facts suggests two different directions for future research. On the one hand, in this work, diversification and intensification were provided by the ILS and GDA frameworks, that, by being trajectory metaheuristics, were identified as ade-

Table 8: Performance of GDA-TREE, considering a refined parameter tuning, in the set of large-scale instances considering a CPU time limit $t = m/(2n) \approx 0.0025np$ seconds.

Instances			$iwl = 1.01$		$iwl = 1.03$		$iwl = 1.05$		$iwl = 1.10$	
n	p	# Inst	$r = 0.1$	$r = 0.9$	$r = 0.1$	$r = 0.9$	$r = 0.1$	$r = 0.9$	$r = 0.1$	$r = 0.9$
500	1	5	**4.3384**	3.7733	4.0502	3.7840	4.0170	3.7267	3.9583	3.9273
	5	5	2.8583	2.5195	**2.9532**	2.5192	2.9237	2.5728	2.9055	2.6069
	10	5	2.0200	1.7314	2.0992	1.7314	2.0699	1.7568	**2.1817**	1.7788
	50	5	0.5032	0.2508	0.5284	0.2508	**0.6035**	0.2701	0.5968	0.3495
	100	5	0.2026	0.1064	0.1959	0.1063	**0.2080**	0.1072	0.1971	0.1077
1000	1	5	3.5365	3.1250	3.5311	3.1250	**3.5426**	3.1742	3.4208	3.1647
	5	5	1.9939	1.4489	**2.0835**	1.5014	2.0111	1.5096	2.0469	1.4805
	10	5	1.1483	0.7038	1.2308	0.6802	1.2389	0.5489	**1.2753**	0.5902
	50	5	0.3972	0.1380	**0.4275**	0.1287	**0.4275**	0.1242	0.4273	0.1212
	100	5	-0.1036	0.0257	-0.0634	0.0239	**0.0365**	0.0192	0.0305	0.0236
2000	1	5	2.8301	2.1237	2.8271	2.1308	2.8682	2.1610	**2.8808**	2.0942
	5	5	1.1349	0.5545	**1.1800**	0.5361	1.1789	0.5339	1.1726	0.5165
	10	5	-0.8754	0.3146	0.0829	0.3015	0.0926	0.3022	0.0640	**0.3149**
	50	5	-3.8303	0.0425	-3.6922	0.0384	-3.8592	0.0375	-3.8991	**0.0521**
	100	5	-1.2654	-0.0063	-1.2987	-0.0111	-1.3533	-0.0093	-1.3363	**-0.0004**
3000	1	5	1.9800	1.2059	2.0273	1.1764	**2.0450**	1.2331	2.0376	1.3130
	5	5	-3.4487	**0.2796**	-2.3253	0.2645	-2.3119	0.2616	-2.3151	0.2505
	10	5	-3.9802	**0.1286**	-3.2357	0.1037	-3.2345	0.1245	-3.2356	0.0997
	50	5	-3.1091	-0.0996	-3.2003	-0.1203	-3.1583	-0.1220	-3.1457	**-0.0544**
	100	5	-0.9326	-0.0663	-0.9632	-0.0642	-0.9824	-0.1042	-0.9576	**-0.0542**
4000	1	5	1.4246	0.9514	1.5618	0.9546	**1.5762**	0.9298	1.5757	0.9602
	5	5	-5.6202	0.1826	-4.7608	0.1811	-4.7583	0.1336	-4.7690	**0.1838**
	10	5	-6.9062	-0.0122	-7.2511	0.0257	-7.1498	-0.0580	-7.2713	**0.0417**
	50	5	-2.8468	-0.4730	-2.7993	-0.2938	-2.8641	-0.4687	-2.7668	**-0.2720**
	100	5	-0.9535	-0.2966	-0.9491	**-0.1614**	-0.9476	-0.1963	-0.9465	-0.1856
8000	1	5	-10.8524	**0.1501**	-9.7064	0.1337	-9.6932	0.1204	-9.6922	-0.1203
	5	5	-8.0223	**-1.3047**	-7.9107	-1.3429	-8.1098	-1.3454	-8.0163	-1.8373
	10	5	-5.9331	**-1.5784**	-5.9791	-1.6111	-5.9543	-1.6038	-5.9898	-2.0524
	50	5	-2.6965	-1.2861	-2.6644	-1.3685	-2.6612	**-1.1649**	-2.6608	-1.3680
	100	5	-0.9121	-0.4786	-0.8445	**-0.4454**	-0.8471	-0.4480	-0.8455	-0.4627
Average RD			-1.2640	0.4718	-1.0955	**0.4760**	-1.1015	0.4709	-1.1026	0.4523

quate choices to extend the capabilities of TREE. In the same line of research, the combination of TREE with other trajectory metaheuristics remains to be investigated. On the other hand, since large changes in the current solution imply in costly updates of TREE internal data structure, it appears that populational-based metaheuristics are not an adequate framework to embed the TREE local search algorithm. However, being TREE a promising approach, confirming or refuting this hypothesis is an interesting subject that deserves future research. Moreover, combining TREE with Memetic Algorithms, that are, at the same time, a populational-based method and a trajectory method and have shown outstanding results when applied to the LOP problem (Schiavinotto and Stützle, 2003), may be a promising direction of research. In a second stage, in case the studies confirm that the TREE method is not adequate for being combined with populational metaheuristics, a deep analysis of the updating operations of the TREE data structures would be done. This analysis would point out to the development of a "TREE-based" local search method that, on the one hand, inherits the TREE low complexity neighborhood search (hence being adequate for large-scale instances) and, on the other hand, is suitable for being combined with populational-based metaheuristics. Last but not least, on a different line of research, developing methods capable of dealing with the increasingly large nowadays LOP applications and utilizing them in practice is also an exciting challenge.

Acknowledgements: This research has been financially supported by FAPESP (grants 2010/08434-1, 2010/10133-0, 2013/03447-6, 2013/05475-7, and 2013/07375-0) and CNPq.

References

Baxter, J.: 1981, Local optima avoidance in depot location, *Journal of the Operational Research Society* **32**, 815–819.

Becker, O.: 1967, Das Helmstädtersche Reihenfolgeproblem - die Effizienz verschiedener Näherungsverfahren, in *Computer uses in the Social Science*, Berichteiner Working Conference, Wien.

Blum, C. and Roli, A.: 2003, Metaheuristics in combinatorial optimization: Overview and conceptual comparison, *ACM Comput. Surv.* **35**, 268–308.

Campos, V., Glover, F., Laguna, M. and Martí, R.: 2001, An experimental evaluation of a scatter search for the linear ordering problem, *Journal of Global Optimization* **21**, 397–414.

Ceberio, J., Hernando, L., Mendiburu, A. and Lozano, J.: 2013, Understanding instance complexity in the linear ordering problem, in *Proceedings of the 14th International Conference on Intelligent Data Engineering and Automated Learning – IDEAL 2013*, H. Yin, K. Tang, Y. Gao (eds), *Lecture Notes in Computer Science* **8206**, 479–486.

Chanas, S. and Kobylanski, P.: 1996, A new heuristic algorithm solving the linear ordering problem, *Computational Optimization and Applications* **6**, 191–205.

Charon, I. and Hudry, O.: 2007, A survey on the linear ordering problem for weighted or unweighted tournaments, *4OR: A Quarterly Journal of Operations Research* **5**, 5–60.

Chenery, H. B. and Watanabe, T.: 1958, International comparisons of the structure of production, *Econometrica* **26**, 487–521.

Dueck, G.: 1993, New optimization heuristics - the great deluge algorithm and the record-to-record travel, *Journal of Computational Physics* **104**, 86–92.

Dueck, G. and Scheuer, T.: 1990, Threshold accepting: A general purpose optimization algorithm appearing superior to simulated annealing, *Journal of Computational Physics* **90**, 161–175.

Garcia, C. G., Pérez-Brito, D., Campos, V. and Martí, R.: 2006, Variable neighborhood search for the linear ordering problem, *Computers and Operations Research* **33**, 3549–3565.

Garey, M. R. and Johnson, D. S.: 1979, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W. H. Freeman & Co Ltd, New York, NY, USA.

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

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

Huang, G. and Lim, A.: 2003, Designing a hybrid genetic algorithm for the linear ordering problem, *Genetic and Evolutionary Computation - GECCO 2003, proc.* pp. 1053–1064.

Johnson, D. and McGeoch, L.: 1997, The traveling salesman problem: a case study, *in* E. Aarts and J. Lenstra (eds), *Local search in Combinatorial Optimization*, Interscience Series in Discrete Mathematics and Optimization, Wiley, Chichester, pp. 215–310.

Karp, R. M.: 1972, Reducibility among combinatorial problems, *in* R. E. Miller and J. W. Thatcher (eds), *Complexity of Computer Computations*, Plenum Press, New York, NY, USA, pp. 85–103.

Kröemer, P., Platos, J. and Snasel, V.: 2013, Implementing artificial immune systems for the linear ordering problem, in *Proceedings of the 7th International Conference on Soft Computing Models in Industrial and Environm Applications*, V. Snasel, A. Abraham, and E.S. Corchado (eds), *Advances in Intelligent Systems and Computing* **188**, 53–62.

Laguna, M., Marti, R. and Campos, V.: 1999, Intensification and diversification with elite tabu search solutions for the linear ordering problem, *Computers and Operations Research* **26**, 1217–1230.

Leontief, W.: 1966, *Input-output economics*, Oxford University Press, New York.

Lourenço, H. R., Martin, O. C. and Stützle, T.: 2003, Iterated local search, *in* F. Glover and G. A. Kochenberger (eds), *Handbook of Metaheuristics*, Kluwer Academic Publishers, pp. 321–353.

Martí, R. and Reinelt, G.: 2011, *The linear ordering problem: exact and heuristic methods in combinatorial optimization*, Springer, Berlin.

Martí, R., Reinelt, G. and Duarte, A.: 2012, A benchmark library and a comparison of heuristic methods for the linear ordering problem, *Computational Optimization and Applications* **51**, 1297–1317.

Sakuraba, C. S. and Yagiura, M.: 2010, Efficient local search algorithms for the linear ordering problem, *International Transactions in Operational Research* **17**, 711–737.

Schiavinotto, T. and Stützle, T.: 2003, Search space analysis of the linear ordering problem, *in* F. Rothlauf et al. (eds), *Applications of Evolutionary Computing*, Springer, Heidelberg, Germany, pp. 197–204.

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

Sukegawara, N., Yamamoto, Y. and Zhang, L.: 2011, Lagrangian relaxation and pegging test for linear ordering problems, *Journal of the Operations Research Society of Japan* **54**, 142–160.

Ye, T., Wang, T., Lü, Z. and Hao J.-K.: 2014, A Multi-parent memetic algorithm for the linear ordering roblem, arXiv:1405.4507v1 [cs.NE] (submitted to arXiv on May 18, 2014).