Using Pairwise Precedences for Solving the Linear Ordering Problem

Valentino Santucci[a], Josu Ceberio[b]

[a] *University for Foreigners of Perugia, Piazza Fortebraccio 4, Perugia, Italy*
[b] *University of the Basque Country (UPV/EHU), Manuel Lardizabal Pasealekua, 20018 Donostia, Spain*

Abstract

It is an old claim that, in order to design a (meta)heuristic algorithm for solving a given optimization problem, algorithm designers need first to gain a deep insight into the structure of the problem. Nevertheless, in recent years, we have seen an incredible rise of "new" meta-heuristic paradigms that have been applied to any type of optimization problem without even considering the features of these problems. In this work, we put this initial claim into practice and try to solve a classical permutation problem: the Linear Ordering Problem (LOP). To that end, first, we study the structure of the LOP by focusing on the relation between the pairwise precedences of items in the solution and its objective value. In a second step, we design a new meta-heuristic scheme, namely CD-RVNS, that incorporates critical information about the problem in its three key algorithmic components: a variable neighborhood search algorithm, a construction heuristic, and a destruction procedure. Conducted experiments, on the most challenging LOP instances available in the literature, reveal an outstanding performance when compared to existing algorithms. Moreover, we also demonstrate (experimentally) that the developed heuristic procedures perform individually better than their state-of-the-art counterparts.

Keywords: Linear ordering problem, precedence, meta-heuristic, insert neighbourhood, variable neighborhood search, construction heuristic

1. Introduction

In recent years, we have seen an incredible rise of new meta-heuristic paradigms for solving a large variety of either combinatorial or continuous optimization problems. Many of these algorithms have been proposed on the basis of some natural metaphor. Unfortunately, as stated by Sorensen [42] in the paper *Metaheuristics: the metaphor exposed*, many of these works have followed a dangerous line of research that may lead the area of meta-heuristics away from scientific rigor. In this direction, it is an old claim that, in order to design a (meta)heuristic algorithm for solving a given optimization problem, algorithm designers should first gain a deep insight into the structure of the problem [39].

In this article, we put that classical methodology into practice by approaching the Linear Ordering Problem (LOP) [27, 12].

The first works related to the LOP are by Leontief [26, 25], who tried to model the US economy by triangulating the input-output matrices that describe the dependencies between different economic branches. Nevertheless, nowadays, the LOP has innumerable applications in other areas, anthropology [20], tournament rankings [27], psychology [23], and graph theory [40] to name a few.

Since the LOP is NP-hard [19], researchers have started to focus on using meta-heuristic algorithms in order to obtain good enough solutions (see for instance [37, 12, 4, 41]). A careful revision of the algorithms proposed in the recent literature reveals that researchers have repeatedly focused on what is called either *insert* movement, *insert* neighborhood or variants of both. The reason for this was extensively studied by Ceberio et al. [12] and concluded that the insert operator/neighborhood has exceptional properties for solving the LOP. This is mainly motivated by the following property of the problem:

"Given a permutation σ that describes a solution for a LOP instance, and the item $\sigma(i)$ that is located at position i, then the respective ordering of the previous and posterior items, with respect to i, does not affect its contribution to the objective function."

Another relevant property of a LOP solution is that its objective value is only given by the pairwise precedences among the permuted items in the solution. As we will see later, this allows the LOP objective function to be expressed as a sum of elementary objective contributions.

Both these structural properties are exploited in this work in order to design a novel and effective meta-heuristic for LOP. The algorithm we propose, namely CD-RVNS, is mainly based on iterative applications of the Variable Neighborhood Search (VNS) [32] scheme that are interleaved by a "problem-aware" shaking stage carried out by means of two novel iterative greedy procedures for the destruction and construction of the solutions.

The VNS algorithm is based on Local Search (LS) schemes, i.e., the solutions in the search space are organized according to a neighborhood topology, and the search is carried out by moving from an incumbent solution to a neighboring one according to a given criterion. VNS extends the idea of LS schemes and considers two different neighborhood topologies, by switching from a main neighborhood to a "secondary" one when the search gets trapped in a local optimum. In our work, we have considered the two classical permutation neighborhoods: *insert* and *interchange* [37]. Importantly, computational time is saved by restricting both neighborhoods using the LOP property studied by [12].

After a VNS execution, the resulting solution is partially destroyed and heuristically rebuilt in order to start a new VNS execution. Both the destruction and construction procedures proposed, namely D-LOP and C-LOP, represent a LOP solution as a set of pairwise precedences. In this way, the construction procedure can exploit the structural decomposition of the LOP objective function in order to follow smoother moves in the space, while the destruction stage is able to control the exploration behavior of the algorithm by using the collected information about the elementary objective entities visited throughout the optimization.

Our proposal extends the works previously published in [12] and [5]. Indeed, the idea of the restricted neighborhood introduced in [12] is adopted in the VNS part of the algorithm. It is worth noting that this is the first application of the restriction procedure to the interchange neighborhood, and also inside a VNS scheme. As regards to the C-LOP heuristic, though inspired by the constructive procedure used in [5] in the context of an ant colony optimization algorithm, a totally new and smarter implementation is introduced which allows a quicker execution. Moreover, here we systemize some theoretical definitions in order to provide a complexity analysis of C-LOP both in the worst-case and the average-case scenarios. Other remarkable novelties of this work are the D-LOP procedure and the mechanisms used to interconnect the different algorithmic components.

For the sake of validating the approach presented in this article (CD-RVNS), a thorough experimental study has been carried out and it can be summarized in three blocks: (1) evaluate the contribution of the heuristic construction and destruction procedures, (2) compare the overall performance of CD-RVNS with respect to the state-of-the-art algorithms, and (3) statistically assess the obtained results. Based on the conducted experiments, we clearly conclude that the proposed idea takes a step forward in LOP optimization by outperforming the current state-of-the-art LOP algorithms.

The remainder of the paper is organized as follows: Section 2 provides an analysis of the works related to our article, while in Section 3 a detailed study of the problem structure of the LOP is presented. Then, in Section 4, we introduce our algorithm proposal: CD-RVNS. Afterwards, in Section 5, a thorough experimental analysis of the algorithms and results is presented. Finally, conclusions and lines for future research are exposed in Section 6.

2. Related Work

According to Garey and Johnson [19], the LOP is included in the group of NP-hard problems, which denotes the difficulty of solving it. Furthermore, the decision variant of the LOP has been proven to be strongly NP-complete [18]. Due to the challenge that this problem represents, we find in the literature a wide variety of different optimization algorithms that have approached the LOP from different perspectives. They can be grouped into four categories: (1)

exact methods, (2) approximation algorithms with theoretical guarantees, (3) constructive heuristics, and (4) meta-heuristic algorithms.

As regards the first group, exact methods are able to solve instances of the LOP up to size $n = 80$, however, these instances are considered to be relatively small, and for larger instances, due to the computational requirements needed, these algorithms are not affordable. Some recent references are the Branch and Bound approach by Charon and Hudry [13] and the Cutting Plane algorithm of Mitchell and Borchers [31].

Regarding the second group, due to the NP-hardness of the problem, researchers have tried to look for an approximation scheme with theoretical guarantees. Though a trivial but practically limited 0.5-approximation scheme exists [29]. In fact, in [33] it has been proven that it is NP-hard to approximate the LOP with a factor better than $\frac{65}{66}$ (see also [27, Ch. 7.1]). What is worse is that such a proof is not constructive so, until now, there is no known approximated algorithm with theoretical guarantees that is usable in practice.

In the third group, we consider the algorithms that, by using explicit information of the problem, iteratively build-up a solution. Known as constructive algorithms [3, 6, 14], these start with an empty linear ordering of items and iteratively add a new item by following some criteria. Though these algorithms do not guarantee the optimality of the obtained solution, they are easy to implement and provide good quality solutions in a reasonable amount of computational time. Despite their effectiveness when proposed, nowadays they are usually implemented as part of more general and powerful schemes: meta-heuristic algorithms.

Finally, the algorithms in the fourth group, meta-heuristics, are currently the most competitive schemes for solving the LOP (and also many other combinatorial and continuous domain problems). In the recent literature, Variable Neighbourhood Search [17], Scatter Search [10], Ant Colony Optimization [15, 5], Differential Evolution [4, 36], Memetic Algorithm [41], Iterated Local Search [35] or Great Deluge Algorithm [34] have been applied to the LOP.

According to some relevant works in the literature of LOP [38, 28, 12], the Iterated Local Search (ILS), Memetic Algorithm (MA) and Tabu Search (TS) have resulted as the most competitive paradigms. Among them, ILS has been reported as the algorithm with highest performance rates on the most challenging benchmarks.

In order to measure the performances of heuristic and meta-heuristic algorithms, a number of benchmarks of instances have been proposed throughout decades. The classical benchmarks include IO [21], MB [30] or SGB [24]. Nevertheless, many of the current approaches are able to reach the best known solutions on the previous benchmarks almost systematically (that are, presumably, optimal). Later, other more challenging benchmarks were also proposed, such as Rand [28], xLOLIB [38] or xLOLIB2 [12]. Finally, though less considered in the literature, there exists a set of very large instances used in [34, 35].

3. The Linear Ordering Problem: definition and properties

In the following, after providing a formal definition of the Linear Ordering Problem (LOP), we outline two important properties of the solutions space of the LOP that will be exploited later on in our work.

3.1. LOP definition

LOP can be equivalently defined both as a problem on (tournament) graphs [13] and as a matrix triangulation problem [27]. For the sake of simplicity, in this article we use the matrix-form definition.

Hence, a LOP instance is given as a square matrix H of non-negative numbers, and the objective is to determine a simultaneous permutation of both rows and columns of H such that the sum of the super-diagonal entries is maximized.

More formally, given $H \in \mathbb{R}^{n \times n}$, the goal is to find a permutation π^* of the set $[n] = \{1, 2, \ldots, n\}$ such that $\pi^* = \arg\max_{\pi \in \mathcal{S}_n} f(\pi)$, where $\mathcal{S}_n$ is the set of all the permutations of $[n]$ and the objective function f is defined as

$$f(\pi) = \sum_{i=1}^{n} \sum_{j=i+1}^{n} H_{\pi(i),\pi(j)}. \tag{1}$$

LOP has a variety of applications, and many existing problems can be reformulated easily as the LOP. In the following, we will enumerate briefly some of them.

In graph theory, considering H to be the weight matrix of a graph $\mathcal{G}$, the LOP is equivalent to finding the acyclic subgraph of $\mathcal{G}$ which maximizes the sum of the arc weights [27].

Moreover, the well known Kemeny's problem [23, 2, 1] can be reformulated to LOP: let consider m persons where each one has ranked n objects (e.g., political parties), which is the linear ordering that aggregates the individual orderings in the best possible way? It turns out that it is possible to build a LOP instance by setting H_{ij} to the number of persons who have ranked object i before object j.

One of the first applications of LOP was in economics. Particularly, the LOP was used in the analysis of the so-called *input-output tables* [25]. Every entry H_{ij} represents the monetary flow from the economic sector i to the sector j. Triangulating such a table (i.e., solving the LOP) allows a ranking of sectors to be determined whose linear flow is as large as possible. Such rankings are often used for comparing the industrial structures between different countries.

Finally, a quite exotic application of the LOP is in anthropology [20] where it has been used to determine a consistent ordering of some historical artifacts whose dates are only partially known.

3.2. LOP solutions as sets of precedences

A LOP solution can be straightforwardly represented as a permutation of integers, i.e., a linear ordering of row/column indices (items). Anyway, given $\pi \in \mathcal{S}_n$, and observing equation (1), it is apparent how the terms summed up in the objective function are given by the precedences among the items in π. Indeed, the matrix entry $H_{i,j}$ appears in the (double) summation of equation (1) if and only if $i \prec_\pi j$, i.e., when i precedes j in π. The binary relation $\prec_\pi$, induced by π, is a total strict order on $[n]$. Therefore, for any permutation π, either $H_{i,j}$ or $H_{j,i}$ contributes to the objective value of π depending on, respectively, $i \prec_\pi j$ or $j \prec_\pi i$.

Moving on from this observation, it is possible to (equivalently) represent a permutation as a set of precedences between items. Formally, given $\pi \in \mathcal{S}_n$, the set

$$P = \{(i,j) : i,j \in [n] \text{ and } i \prec_\pi j\} \tag{2}$$

explicitly represents all the precedences encoded by π. For instance, in the permutation $\pi = \langle 312 \rangle$ we have the three precedence relations $3 \prec_\pi 1$, $3 \prec_\pi 2$ and $1 \prec_\pi 2$, thus π is equivalent to the set $P = \{(3,1),(3,2),(1,2)\}$.

Such a set P has the following properties:

- it is consistent, i.e., if $(i,j) \in P$, then $(j,i) \notin P$;
- it is transitively closed, i.e., if $(i,j) \in P$ and $(j,k) \in P$, then $(i,k) \in P$;
- it is complete, i.e., either $(i,j) \in P$ or $(j,i) \in P$.

Note also that, if a set of precedences is consistent, it is also complete if and only if its cardinality is $\binom{n}{2}$. Moreover, a consistent and complete set of precedences is guaranteed to be transitively closed.

Representing a linear ordering as a set of precedences requires more memory than a linear permutation encoding. Anyway, this representation allows a more fine-grained control on the objective value of the solution. Indeed, the LOP objective function can be now restated as a single summation of the most elementary units of objective value: formally, given a consistent and complete set P of precedences over the items in $[n]$, the LOP objective function can be expressed as[1]

$$g(P) = \sum_{(i,j) \in P} H_{i,j}. \tag{3}$$

Hence, P is explicitly formed by *objects* (the precedences) which are connected one-to-one to the atomic objective values of the LOP.

[1] It is straightforward to verify the equivalence between equations (1) and (3) when P is the precedences set representation of π.

Importantly, this representation allows constructive heuristics to be devised that smoothly build-up the LOP solution precedence by precedence, thus avoiding drastic changes in the objective value that are common in all the previously proposed heuristics which work with the usual "linear ordering" representation [27, Ch. 2]. Indeed, classical constructive heuristics iteratively add items to the linear ordering, but every item's insertion accounts for multiple precedences. Therefore, it is apparent how building up a solution precedence by precedence has a smoother impact on its objective value. Since smoothness is often considered to be beneficial in the field of fitness landscape analysis (see for example [22, Ch. 5]), we think that this property can bring to an effective constructive heuristic for the LOP.

For this reason, the construction and destruction procedures proposed in this article (see Section 4) work with the LOP solutions represented as a set of precedences. Interestingly, this is a novelty with respect to the previously proposed LOP construction heuristics, which are all based on the classical linear representation of permutations (see [27, Chap. 2] for a survey on the topic). However, note that, in the field of permutation-based optimization problems, the precedences set representation has been considered by [8] to design a pheromone model for the Ant Colony Optimization (ACO) algorithm and by [5] in the ACO solution construction procedure.

While equation (2) clearly shows how to obtain the precedences set representation of a given permutation, the opposite conversion can be performed as follows. By bearing in mind that permutations in $\mathcal{S}_n$ are bijective functions in $[n]$, any permutation has a corresponding inverse permutation. Hence, given a consistent and complete set of precedences P, we compute an intermediate permutation $\sigma \in \mathcal{S}_n$ such that $\sigma(i) = |\{(k,i) \in P : k \in [n]\}| + 1$, then the permutation $\pi \in \mathcal{S}_n$, equivalent to P, is obtained by inverting σ, i.e., $\pi = \sigma^{-1}$.

3.3. The restricted neighborhood

According to the literature on the LOP, a recurrent option in meta-heuristic design is to implement schemes that include local search algorithms under the insert neighborhood. A recent analysis [12] on the suitability of that neighborhood revealed that given a solution $\pi \in \mathcal{S}_n$ of the LOP (represented as a permutation), for each item in π there exists a set of positions (regardless of the ordering of the rest of the items) in which the item cannot appear and the solution is a local optimum. By definition, this fact is extended to global optima solutions and gives information about the partial appearance of these.

Based on that information, the authors proposed a reduced version of the insert neighborhood, namely *restricted insert neighborhood*. In this manuscript, we will consider it as the central neighborhood of the core algorithm (RVNS). In what follows, the fundamentals of the LOP and the restricted neighborhood are provided for basic intuition[2]:

- Under the insert neighborhood, two solutions σ and π are neighbours if π is obtained by moving an item of σ to a different position.
- Associated to each item $\pi(i)$ are the set of parameters $H_{\pi(i),j}$ and $H_{j,\pi(i)}$ where $j = 1, \ldots, n$.
- The parameters associated to an item $\pi(i)$, remain associated regardless of the position that the item adopts in π after an insert operation.
- The contribution of an item $\pi(i)$ to the objective value consists of the sum of the matrix entries associated to it that appear in super-diagonal locations. Note that (from the previous section) when $i \prec_\pi j$ then $H_{i,j}$ accounts for the objective value of π (and $H_{j,i}$ does not). When an insert operation is performed on an item, then its contribution varies as some $i \prec_\pi j$ precedences no longer hold.

Considering the previous notes, when a solution π is a local optimum, then there is no insert operation in π that increases the contribution of any item to the objective value. Obviously, that contribution depends on the location

[2]For a detailed description on the restricted neighborhood, we recommend the interested reader to address the original work [12].

where it is placed, and also on the ordering of the rest of the items. However, due to the matrix entries associated to each of the items, there are always some positions (at least one) for any item that, when placed there, the solution can be improved (this is critical in order to restrict positions).

With illustrative purposes, let us consider an extreme case in which the parameters of an item in the values in the column are larger than the values in the row. In that particular case, that item must appear necessarily in the last position of the solution in order for this to be locally optimal. As a consequence, when this item is located in any other position, then the solution cannot be a local optimum, and thus cannot be the global optimum. These positions are known as *restricted* positions. Finally, it is worth noting that, when performing a greedy local search, the best insert movement in the insert neighborhood is never given by moving an item to a restricted position. Therefore, by avoiding restricted positions, the neighborhood becomes smaller (less evaluations are carried out) but returns the same result as the classical insert neighborhood.

Finally, it is worthwhile to note that, though the restriction has been originally defined only for the insert neighborhood, it can be safely extended to the interchange neighborhood[3]. Hence, in this work we use the restricted versions of both the insert and interchange neighborhoods in order to devise an efficient variable neighborhood search scheme (see Section 4.1).

4. Designing a metaheuristic for the LOP: CD-RVNS

In this section, we propose a meta-heuristic, namely CD-RVNS, that mainly consists of three algorithmic components:

- RVNS, i.e., a variable-neighborhood search algorithm based on restricted neighborhood definitions specifically designed for the LOP;

[3]Under the interchange neighborhood, two solutions σ and π are neighbors if σ is obtained by swapping the items in positions i and j of π.

- C-LOP, i.e., a randomized heuristic construction procedure that builds-up a LOP solution precedence by precedence;
- D-LOP, i.e., a destruction procedure that removes precedences from a LOP solution with the aim of producing a new starting point for C-LOP.

The three procedures are arranged in an iterative process as follows. An initial solution is built from scratch using C-LOP. Starting from this solution, RVNS climbs up its basin of attraction till a local optimum is met. Then, D-LOP partially destroys the local optimum and the partial solution is fed again to C-LOP for the next iteration of CD-RVNS.

The pseudo-code of CD-RVNS is provided in Algorithm 1. Both C-LOP and D-LOP work with the precedences set representation, while RVNS uses the classical linear representation of permutations. Hence, the procedures described in Section 3.2 are used, in lines 7 and 9, to convert between the two representations. Moreover, in the memory M (lines 3 and 10), we maintain the number of occurrences of any possible precedence in the local optima visited so far by RVNS. M is then used inside D-LOP (line 12) to increase the exploration ability of the search. Finally, it is important to note that, though C-LOP and D-LOP require setting the *greediness factor* α and the *destruction rate* β, their values are automatically and dynamically adapted during the iterations (lines 5 and 11). Therefore, conversely from most of the meta-heuristics in the literature, CD-RVNS has the remarkable property of being a parameter-free algorithm.

We describe the working principles of RVNS, C-LOP, and D-LOP in, respectively, Sections 4.1, 4.2, and 4.3. Finally, Section 4.4 motivates the parameter adaptation scheme that we have designed.

4.1. RVNS: Variable Neighborhood Search with Restricted Neighborhood

Restricted Variable Neighborhood Search (RVNS) is the core paradigm of the proposed approach. As stated in the introduction, a VNS explores solutions alternating between two neighborhoods: a main neighborhood and a secondary neighborhood. Subsequently, the optimization is carried out in RVNS as follows.

Algorithm 1 General scheme of CD-RVNS

function CD-RVNS($H \in \mathbb{R}^{n \times n}$) ▷ H is the LOP instance matrix
  $P \leftarrow \emptyset$ ▷ P is a set of precedences
  $M \leftarrow \mathbf{0}$ ▷ M is a $n \times n$ matrix of counters
  while `n_evals` $<$ `max_evals` **do** ▷ Stopping criterion is not met.
    $\alpha \leftarrow$ get a random number in $[0.9, 1)$
    $P \leftarrow$ C-LOP(P, α, H)
    $\pi \leftarrow$ convert the set of precedences P to a permutation
    $\pi \leftarrow$ RVNS(π, H)
    $P \leftarrow$ convert the permutation π to a set of precedences
    for each $(i, j) \in P$, increase the counter $M_{i,j}$
    $\beta \leftarrow 1 - 0.9 \cdot (\texttt{n\_evals}/\texttt{max\_evals})$
    $P \leftarrow$ D-LOP(P, M, β)
    increase `n_evals` by the function evaluations consumed in RVNS
  return π

First, a greedy local search is carried out on the restricted insert neighborhood until it gets trapped in a locally optimal solution. Then, in order to continue with the optimization, one step of local search is carried out in the secondary neighborhood, the restricted interchange neighborhood. These two phases are iteratively repeated until a local optimum common to both neighborhoods is found.

Afterwards, a procedure to modify the solution called *shake* is applied to the current solution with the aim of moving to another area of the fitness landscape (the best solution found so far is saved). In our proposal, the shake consists of a Destruction-Construction procedure explained in sections 4.3 and 4.2. Finally, the algorithm iterates back to the local search phases with the newly created solution.

4.2. C-LOP: Construction heuristic procedure for LOP

C-LOP is a randomized constructive heuristic which works with the precedences set representation introduced in Section 3.2. Starting from a (possibly empty) partial solution, C-LOP iteratively adds precedences till a complete solution is obtained. The choice of the precedence is guided by the heuristic information contained in the LOP matrix H, while the parameter $\alpha \in [0, 1]$ regulates the greediness of C-LOP. Its pseudo-code is provided in Algorithm 2.

Algorithm 2 The C-LOP heuristic construction procedure

1: **function** C-LOP($P \subset U_n, \alpha \in [0,1], H \in \mathbb{R}^{n \times n}$)
2:     $C \leftarrow U_n \setminus \{(i,j) : (i,j) \in P \text{ or } (j,i) \in P\}$
3:     **while** $|P| < \binom{n}{2}$ **do**
4:         $r \leftarrow$ random number in $[0, 1)$
5:         **if** $r < \alpha$ **then**
6:             $(i,j) \leftarrow \arg\max_{(a,b) \in C} H_{a,b}$
7:         **else**
8:             $(i,j) \leftarrow$ roulette wheel on C basing on H
9:         $P \leftarrow$ TransitiveClosure($P \cup \{(i,j)\}$)
10:        $C \leftarrow C \setminus \{(i,j) : (i,j) \in P \text{ or } (j,i) \in P\}$
11:    **return** P

Let U_n denote the universe set of the possible precedence relations in $[n]$, i.e., $U_n = \{(i,j) : i,j \in [n] \text{ and } i \neq j\}$, then C-LOP requires as input a set $P \subset U_n$ that is both consistent and transitively closed. This allows C-LOP to be used starting from a partial solution. Moreover, since the empty set is consistent and transitively closed, C-LOP can also be used, as any other heuristic, to construct a solution from scratch.

In line 2, the candidate set C of precedences that can be added to P without violating its consistence is computed. Then, the loop in lines 3–10 fills up P by adding new precedences in such a way that the consistence and transitive closure of P are maintained as loop invariant conditions. In every iteration of the loop, the following steps are performed:

- in lines 4–8, a candidate precedence (i,j) is selected from C by following two possible strategies: (i) with probability α, the precedence with the largest heuristic value is selected, or (ii) with probability $1 - \alpha$, a roulette wheel tournament is performed on C in such a way that any $(i,j) \in C$ has probability ${}^{H_{i,j}}/{\sum_{(a,b)\in C} H_{a,b}}$ to be selected;

- in line 9, the selected (i,j) is added to P together with the newly induced precedences;
- in line 10, C is updated by removing the newly introduced precedences and their reverses.

The loop ends as soon as $|P| = \binom{n}{2}$, i.e., when P is consistent and complete. Hence, P is now a valid linear ordering and it is returned in line 11.

As shown in Appendix A, C-LOP has an average time complexity $O(n^2 \log n)$ when invoked starting from an empty solution.

4.3. D-LOP: Destruction procedure for the LOP

D-LOP is a destruction procedure that takes as input a complete LOP solution and iteratively removes precedences from it in such a way that the returned partial solution is a consistent and transitively closed set of precedences that can be safely fed to C-LOP in the next iteration of CD-RVNS.

The number of removals is given by the *destruction rate* $\beta \in [0,1]$, while a preferential ordering on the precedences to be removed is obtained from the occurrences memory M (see lines 3 and 10 of Algorithm 1). The aim is to opt for the removal of precedences that have been met more often in the local optima visited so far by CD-RVNS. This mechanism guides the search of CD-RVNS towards less visited regions of the search space.

The pseudo-code of D-LOP is provided in Algorithm 3. Since the set P in input is complete, $|P| = \binom{n}{2}$. Hence, in line 2, we use β to compute the number m of removals as a percentage of $\binom{n}{2}$. In line 3, we sort, in descending order (with ties randomly broken), the entries of M. This is the ordering by which we consider candidate precedences to be removed from P in the loop in lines 5–14. In line 6, we select a candidate precedence $(i,k) \in P$. In general, by only removing (i,k) we cannot assure that P will be transitively closed. Indeed, we also need to consider all the pairs of precedences (i,j) and (j,k) which are in

Algorithm 3 The D-LOP destruction procedure

1: **function** D-LOP($P \subset U_n, M \in \mathbb{N}^{n\times n}, \beta \in [0,1]$)
2:     $m \leftarrow \lfloor \beta \cdot \binom{n}{2} \rfloor$
3:     sort in descending order the entries of M (ties broken randomly)
4:     $t \leftarrow 0$
5:     **while** $t < m$ and not all the precedences in P have been scanned **do**
6:         $(i,k) \leftarrow$ get next precedence from P based on the ordering in M
7:         $r \leftarrow$ random number in $[0,1)$
8:         **if** $r < 0.5$ **then**
9:             $R \leftarrow \{(i,k)\} \cup \{(i,j) : j \neq k \text{ and } (i,j) \in P \text{ and } (j,k) \in P\}$
10:         **else**
11:             $R \leftarrow \{(i,k)\} \cup \{(j,k) : j \neq i \text{ and } (i,j) \in P \text{ and } (j,k) \in P\}$
12:         **if** $t + |R| \leq m$ **then**
13:             $P \leftarrow P \setminus R$
14:             $t \leftarrow t + |R|$
15:     **return** P

P. Actually, P is guaranteed to be transitively closed if we remove (together with (i,k)) the first precedence of all such pairs or, alternatively, the second precedence of all those pairs. Hence, in lines 7–11, we compute the candidate set R of removals by randomly choosing between the precedences of the first or second type. By noting that t denotes the number of removals so far, in lines 12–14, if the desired number of removals m is not exceeded, the precedences in R are removed from P and the counter is updated. When either the number of desired removals is reached, or all the precedences of P have been scanned, the loop ends and the current (transitively closed) set P is returned in line 15.

It is easy to see that any selected removal induces no more than n additional precedences to be removed (i.e., $|R| \leq n$), hence D-LOP is guaranteed to remove at least $\lfloor \beta \cdot \binom{n}{2} \rfloor - n + 1$ precedences from P. Moreover, though we are not able to theoretically guarantee the removal of exactly $\lfloor \beta \cdot \binom{n}{2} \rfloor$ precedences, in all the conducted experiments, the desired number of removals has been always met.

Finally, the asymptotic complexity of D-LOP is $O(n^2 \log n)$. Indeed, by using the data structure introduced in Appendix A, it is possible to efficiently perform the $O(n^2)$ iterations of the loop in lines 5–14. Hence, the whole complexity is given by the sorting operation in line 3. Since $|M| = \Theta(n^2)$, the sorting step requires $O(n^2 \log n)$ time.

4.4. Parameters Adaptation Scheme

CD-RVNS does not require any parameter to be set. Indeed, the two parameters α and β of, respectively, C-LOP and D-LOP are automatically set during the iterations of CD-RVNS.

The greediness factor α is randomly sampled from the interval $[0.9, 1)$ before any execution of C-LOP (see line 5 of Algorithm 1). This interval guarantees a good trade-off between effectiveness and diversity of the solutions produced by C-LOP. Note also that the interval is open on the right because, otherwise, the setting $\alpha = 1$ results in a deterministic behavior of C-LOP. An experimental analysis, discussed in Section 5.1, shows that C-LOP, using this setting, is preferred to the state-of-the-art LOP construction heuristics in the literature.

The destruction rate β represents the percentage of precedences to remove from the last local optimum visited. Hence, by regulating the value of β, it is possible to trade the exploration and exploitation degrees of CD-RVNS. A small destruction rate guides the search in the nearby area of the recently visited local optimum, while a large value induces a more diverse search that, in principle, can allow CD-RVNS to escape stagnation situations. Using the formula in line 11 of Algorithm 1, the parameter β is linearly shaded from 1 to 0.1 with the passing of the iterations. Therefore, as is common in a lot of the meta-heuristics in the literature, CD-RVNS moves from a diverse search in the earlier phases to a larger intensification behavior in the later iterations.

5. Experimental Study

For the sake of evaluating the performance of CD-RVNS, we carry out a thorough experimental analysis. Firstly, we analyze the performance of the proposed constructive algorithm C-LOP, and compare it with two other state-of-the-art procedures. Then, we evaluate the performance of the overall algorithm, placing special emphasis on the *shaking* method. At that point, we will evaluate the contribution of the constructive-destructive procedure proposed, and we compare it with another common option from the literature. Afterwards, we evaluate the overall performance of the algorithm by comparing the obtained results with those reported in the literature (the state-of-the-art). This is done using two different stopping criterion: (1) maximum number of evaluations performed and (2) maximum execution time limit. Conclusions are drawn on the basis of the Bayesian statistical analysis introduced by Benavoli et al. [7][4].

As stated in Section 2, although a number of benchmarks have been proposed for evaluating the performance of the algorithms, nowadays, only some of them are useful for that purpose. In this sense, we have used the benchmarks xLOLIB and xLOLIB2 (proposed in [38] and [12], respectively) for analyzing the different design options of the CD-RVNS, and extended them with the *RandA1*, *RandA2* and *RandB* benchmarks [28] when evaluating the performance of CD-RVNS with respect to the state-of-the-art algorithm, ILS_r. Finally, though less considered in the literature, we also run CD-RVNS on the set of very large instances used in [34, 35].

5.1. Construction heuristic

One of the key points in this proposal is to evaluate whether the construction strategy C-LOP based on the precedences' set representation, is competitive when compared to other existing constructive algorithms (note that this algorithm is used to provide an initial good solution to CD-RVNS). To that end, we will compare C-LOP with the two best constructive algorithms reported by Marti [27]: Becker's algorithm [6] and Best Insertion (BI) algorithm [27].

With this in mind, we executed the three algorithms, i.e., C-LOP, Becker's

[4]Source codes, instances and additional experimental results are available at: `https://github.com/sgpceurj/Precedences_LOP`.

algorithm, and BI on the xLOLIB and xLOLIB2 instances. C-LOP was run starting from an empty solution, moreover, it is worth pointing out that Becker's algorithm is deterministic, while the other two proposals are stochastic. In this sense, these algorithms were run 20 times for each instance in the benchmarks. Results are summarized in Fig. 1 as average relative percentage deviation (ARPD) with respect to the best known solutions (the new best solutions obtained in this work), and grouped as box-plots according to the size of the instance.

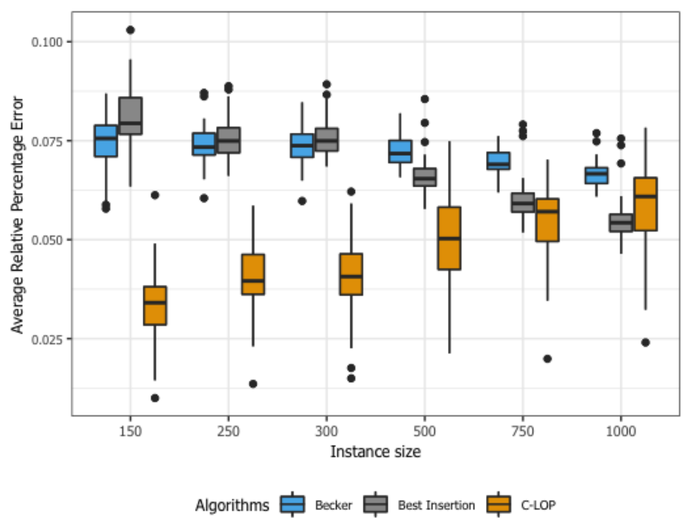


Figure 1: Performance comparison of Becker's algorithm, Best Insertion algorithm and C-LOP. Results are shown as ARPD w.r.t. best known solutions reported.

Conducted experiments shown in Fig. 1 reveal that the proposed strategy obtains, in general, better results than Becker's algorithm and BI. Only for instances of size 1000, BI shows similar or better behavior. Anyway, it is worth noting that the variance of our strategy is higher, i.e., the obtained set of results is more heterogeneous. A characteristic that is interesting when the constructive algorithm is used inside a shaking method, as in our case.

5.2. Shaking method

Another critical part of the proposed algorithm is to evaluate the suitability of the different shaking methods. In this case, we aim to evaluate the performance of the Constructive-Destructive (CD) procedure proposed for shaking the solutions, and compare it to the most used method in the literature: performing swaps of $n/10$ random pairs of items [38, 11].

To that end, we run the CD-RVNS, and a similar algorithm in which we have replaced the CD shake with the swap shake. Both algorithms were run with a limit of $1000n^2$ objective function evaluations. As both algorithms are stochastic, we performed 20 executions per instance. Results are summarized as box-plots in Fig. 2. Conducted experiments point out that, definitively,

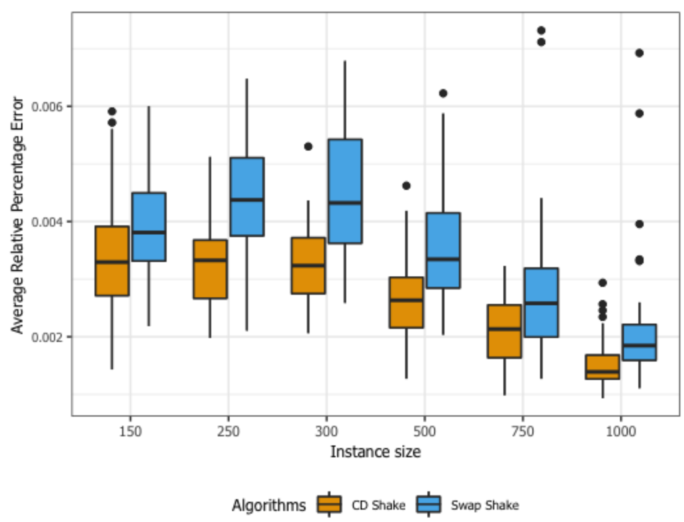


Figure 2: Performance comparison pairwise precedence and swap shake algorithms. Results are shown as relative error w.r.t. best known solutions reported.

the proposal that implements the CD shake outperforms the shake procedure proposed in [38, 11]. Moreover, as the size of the instance increases, results show that the variance of the algorithm with CD shake is lower than its counterpart.

5.3. Comparing to the state-of-the-art

In order to analyze the overall performance of the designed algorithm, we carried out a broad experimental analysis whose results are summarized in this section.

As stated in Section 2, according to latest works in the literature of LOP [38, 28, 12], ILS, and especially, ILS_r (restricted version of ILS introduced in [12]) can be seen as the algorithm with the highest performance rates on the most challenging benchmarks. Thus, in this experimental study, we will compare the results obtained by the CD-RVNS with those of ILS_r.

Finally, in order to carry out a comparison of the algorithms as fairly as possible, both algorithms have been set with the same stopping criterion: a maximum of $1000n^2$ objective function evaluations[5]. Both algorithms are executed 20 times on each instance of the xLOLIB and xLOLIB2 benchmarks, and the *RandA1, RandA2* and *RandB* benchmarks (from now on *Rand* benchmarks).

Results are summarized in Table 1 grouped in equal size instances (for xLOLIB and xLOLIB2), and grouped in benchmarks for *Rand* type instances. Numerical values describe the number of instances for which CD-RVNS obtained better mean objective value (over 20 repetitions) than ILS_r.

	xLOLIB benchmarks						*Rand*			
	150	250	300	500	750	1000	*A1*	*A2*	*B*	
CD-RVNS vs. ILS_r	25	30	33	43	50	50	99	33	41	404
Total instances	39	39	50	50	50	50	100	75	90	543

Table 1: Summary of the number of instances for which CD-RVNS shows a better average performance than ILS_r.

The summary reveals that CD-RVNS is highly competitive and obtains better results than ILS_r in 231 instances out of 278 (83%) in xLOLIB benchmarks, and in 173 instances out of 265 (65.2%) in *Rand* benchmarks. Not limited to

[5]Calculating the objective value of a neighboring solution, despite being efficient and having a lower time complexity than in Eq. 1, counts as one, since both algorithms implement equally efficient insert neighborhood revisions.

that, the proposed algorithm is able to obtain new best known solutions in 234 out of 278 instances (all of them for xLOLIB and xLOLIB2). In general, CD-RVNS obtained on average the best fitness value for 404 instances out of 543 (74%).

With the aim of gaining some intuition with regard to the performance difference between both algorithms, ARPD results have been illustrated as box-plots in Figs. 3 and 4.

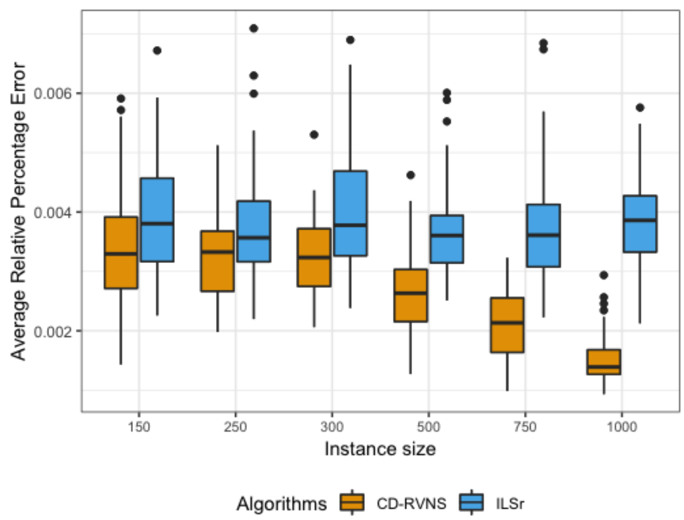


Figure 3: Performance comparison of CD-RVNS and ILS_r on the xLOLIB and xLOLIB2 benchmarks of instances. Results are shown as ARPD w.r.t. the best known solutions found, and have been grouped according to the size of the instance.

The figures confirm the results in Table 1. As regards to xLOLIB benchmarks (see Fig. 3), the larger the size of the instance, the higher the difference between the algorithms. Moreover, the box-plots show that for the smallest instances ($n = 150$ and $n = 250$), although the two algorithms obtained similar results, CD-RVNS has better median values. In relation to the *Rand* benchmarks (see Fig. 4), we observe that *RandA1* is the most challenging benchmark as differences are larger, however, CD-RVNS beats ILS_r. As regards *RandA2* and *RandB*, ILS_r is slightly more competitive, however, the distances with re-

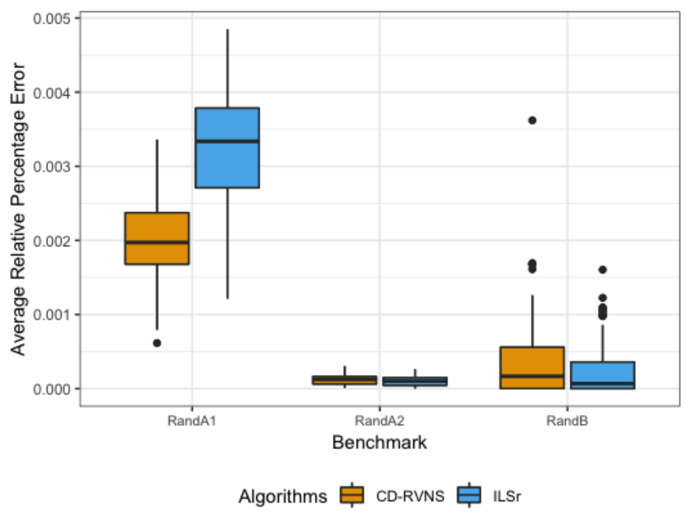
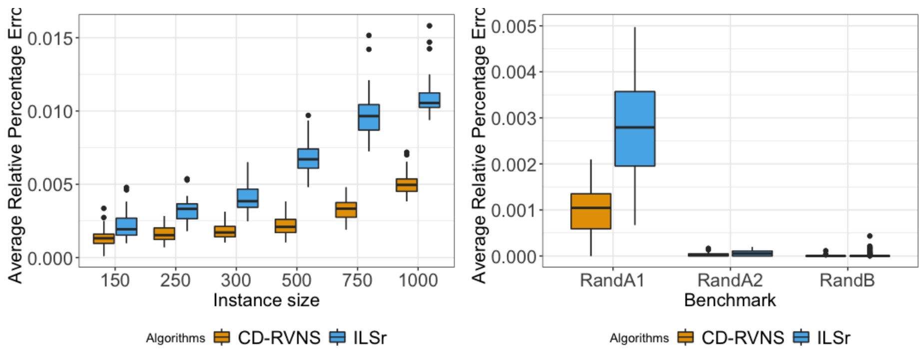


Figure 4: Performance comparison of CD-RVNS and ILS_r on the *Rand* benchmarks of instances. Results are shown as ARPD w.r.t. the best known solutions found.

spect to the best known values in either cases are very small.

Not limited to the comparison above, we have also compared the performance of CD-RVNS and ILS_r using an execution time limit as stopping criterion. In particular, we repeated the experiment above for four different limits: 50s, 100s, 200s and 500s. The executions were carried out on a cluster of 20 nodes, each of them equipped with two Intel Xeon X5650 CPUs and 48GB of memory. Both algorithms were implemented in C++ and the same compiler was used for building the binaries. Results obtained for 50s time limit are depicted in Fig. 5. As can be observed, when considering the results on xLOLIB benchmarks, CD-RVNS outperforms ILS_r for all the instance sizes, and the performance difference becomes larger as the size of the instances increases. In relation to the results on *Rand* benchmarks, values observed in Fig. 4 are almost repeated. In fact, CD-RVNS is more (or equally) competitive with respect to ILS_r in the three benchmarks. Similar results were observed for the rest of the time limits considered. The interested reader is referred to the supplementary material in the Github repository (see footnote 4) for the figures with $100s, 200s$

and 500s.



(a) xLOLIB benchmarks

(b) *Rand* benchmarks

Figure 5: Performance comparison of CD-RVNS and ILS_r with 50 seconds of execution limit as stopping criterion.

Additionally, in Appendix B the best values obtained by CD-RVNS in the presented experimentation have been introduced. Note that in 202 instances out of 543 it was able to reach to the best known values reported in the literature, and in other the 278 cases new best known solutions were obtained.

5.4. Bayesian statistical analysis

In order to statistically assess the results obtained, we have followed the Bayesian approach presented in [7], as it provides a deeper insight into the results than the classical null hypothesis significance tests. In particular, as we cannot assume that the experimental results come from a Gaussian distribution, we have used the Bayesian equivalent of the Wilcoxon's test[6].

The Bayesian analysis was conducted on the ARPDs obtained by each algorithm in the 20 repetitions. The procedure used requires the definition of what is understood as 'practical equivalence' or 'rope' in [7]. In our case we have

[6]We have used the implementation available in the development version of the **scmamp** R package [9] available at `https://github.com/b0rxa/scmamp`.

considered that both approaches are equivalent when the difference in ARPD is smaller than 5×10^{-4}[7]. Fig. 6 shows a summary of the results in a Simplex plot.

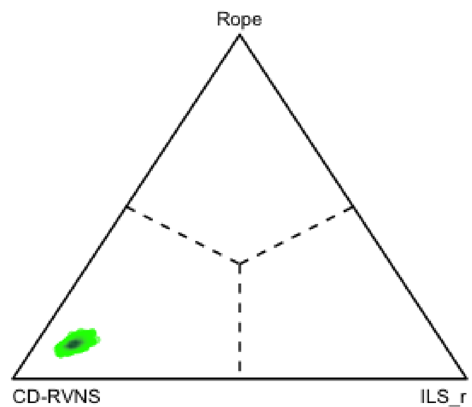


Figure 6: Simplex Plot. Average Posterior probabilities of being the winners: CD-RVNS: 0.8121, ILS_r: 0.0859 and Rope: 0.1019. This is done by assuming that a difference between the algorithms lower than 0.0001 points out practical equivalence.

Briefly, the points in the plot represent a sampling of the posterior distribution of the probability of win-lose-tie. In other words, the closer a point is to the CD-RVNS vertex of the triangle (or, equivalently, to the ILS_r or the Rope vertices), the more probable it is for CD-RVNS to produce better results (or equivalently, ILS_r or both algorithms being equal). Therefore, the three areas delimited by the dashed lines show the dominance regions, i.e., the area where the highest probability corresponds to its vertex. For more details, see [7].

The plot shows that there is almost no uncertainty about the results. The probability mass of the posterior is on the side of the CD-RVNS. Hence, we can summarize the distribution shown in the plot, estimating the average posterior

[7]We can set this rope at any reasonable value. In our case, we have fixed its value according to the magnitude of the ARPD produced by both algorithms, and mainly with visualization purposes.

probability for each situation (CD-RVNS being better, equal or worse than ILS_r). In that case, the expected probabilities are 0.8121 that CD-RVNS obtains better results, 0.1019 that the results of both algorithms are equivalent and 0.0859 that ILS_r obtains better results. In other words, it is almost 8 times more probable that CD-RVNS outperforms ILS_r than the opposite.

5.5. Additional experiments on very large instances

For the sake of completeness, CD-RVNS has been run on the very large instances used in [34] and [35]. These are 150 instances with sizes that range in $[500, 8000]$, and are specifically designed to highlight the characteristics of the TREE technique introduced in [35]. This technique was proposed as an algorithm to speed up the computation of a local search iteration in the LOP. Particularly, authors introduced it in the framework of ILS, proposing the ILS-TREE [34].

It is worth noting that ILS-TREE does not use the restricted neighborhood used in ILS_r and in our work, it can thus be considered a clever algorithmic speed-up technique that, in principle, can also be applied in the restricted neighborhood framework.

In order to carry out a fair comparison of ILS-TREE and CD-RVNS, it is necessary to know the number of evaluations performed in the experimentation in [34]. Unfortunately, the authors of [34] do not specify the number of evaluations performed in their experimentation. Furthermore, they only provide relative deviations (with respect to the objective results reported in [35]) averaged on every one of the 30 (n, p) instance configurations[8]. Therefore, in our experimentation on this benchmark suite, we executed CD-RVNS, 20 times per instance, for $\min\{1000n^2, 10^{10}\}$ evaluations. We computed the relative deviations as done in [35] and we compared our results with those of ILS-TREE when executed with its best parameters setting. CD-RVNS obtained better results on 21 out of 30 (n, p) instance configurations. The detailed results are provided in

[8]In [34] and [35], n is the instance size, while p is the instance density.

the supplementary material available in the online repository (see footnote 4).

6. Conclusions & Future Work

In this paper, we have put into practice an old claim about designing optimization problems, that is, gain as much information and intuition as possible about the problem, and then design an approach that is able to efficiently and effectively use all that information.

To that end, we took the Linear Ordering Problem (LOP), we studied two structural properties of LOP solutions, and we exploited them in order to design an effective meta-heuristic for the LOP.

The proposed algorithm, CD-RVNS, mainly employs three algorithmic components that iteratively interact with each other: a variable neighborhood search procedure working with restricted neighborhood definitions, and two iterative greedy heuristic procedures for constructing and destructing LOP solutions by means of a novel precedences set representation of permutations.

Remarkably, the interaction among the different components of CD-RVNS has been designed in such a way that no parameters need to be set by the practitioner.

For the sake of validating the presented approach, we conducted a thorough experimental study in three blocks: (1) evaluate the construction and destruction procedures, (2) compare the overall performance of CD-RVNS with respect to the state-of-the-art algorithm on two different stopping criterion, and (3) statistically assess the obtained results. From the observed results, we clearly concluded that the proposed idea takes a step forward in LOP optimization by outperforming the current state-of-the-art algorithms. The outstanding results might be motivated mainly by two reasons: a very efficient VNS scheme devised by means of the restricted neighborhood technique, and the efficient and effective iterated greedy destructive-constructive procedures based on the new precedence-based representation for the LOP solutions.

The research work carried out in this paper demonstrated that considering the relevant information about the structure of the problem when designing algorithms may make headway in terms of quality of solutions obtained. In this sense, we think there are still possibilities for future research by developing the properties of the LOP. One of these is an extension of the property of the LOP presented in the introduction:

"Given a solution σ of the LOP, and the item $\sigma(i)$ that is located at position i, then the ordering of the items in positions $1 \ldots i-1$ does not affect the contribution $\sigma(i)$ to the objective function (the same for the ordering of the items at positions $i+1 \ldots n$."

Nonetheless, this property can be extended to any subset of consecutive positions in the solution, i.e.,

"Given a solution σ of the LOP, and $m+1$ items at positions $k \ldots k+m$, $\{\sigma(k), \sigma(k+1), \ldots, \sigma(k+m)\}$, then the ordering of the items at positions $i < k$ (or those in $i > k+m$) does not affect the contribution of the items from k to $k+m$."

Taking into account the property above, then, given a solution for the problem, it is possible to develop algorithms that fix some positions of the solution, and focus exclusively optimizing the subset of items that has not been fixed. The property guarantees that this type of strategy cannot obtain, in any case, worse quality solutions than the initial one.

Acknowledgements

This work has been partially supported by: the Research Groups 2019-2020 (IT1244-19) and Elkartek Program (PROMISE project) from the Basque Government, the TIN2016-78365-R research project from the Spanish Ministry of Economy, Industry and Competitiveness, and EMPHATIC H2020 project from the European Research Council, and the research grant "Fondi per i progetti di ricerca scientifica di Ateneo 2019" of the University for Foreigners of Perugia under the project "Algoritmi evolutivi per problemi di ottimizzazione e modelli di apprendimento automatico con applicazioni al Natural Language Processing".

References

[1] Aledo JA, Gámez JA, Molina D. Tackling the rank aggregation problem with evolutionary algorithms. Applied Mathematics and Computation 2013;222:632 –44.

[2] Ali A, Meilă M. Experiments with kemeny ranking: What works when? Mathematical Social Sciences 2012;64(1):28 – 40. Computational Foundations of Social Choice.

[3] Aujac H. La hiérarchie des industries dans un tableau des échanges interindustriels. Revue économique 1960;11(2):169–238.

[4] Baioletti M, Milani A, Santucci V. Linear ordering optimization with a combinatorial differential evolution. In: 2015 IEEE International Conference on Systems, Man, and Cybernetics. 2015. p. 2135–40.

[5] Baioletti M, Milani A, Santucci V. A new precedence-based ant colony optimization for permutation problems. In: Proc. of Asia-Pacific Conference on Simulated Evolution and Learning (SEAL 2017). Cham: Springer International Publishing; 2017. p. 960–71.

[6] Becker O. Das helmstädtersche reihenfolgeproblem – die effizienz verschiedener näherungsverfahren -. In: Computers Uses in the Social Sciences, Berichteiner Working Conference. Vienna; 1967. .

[7] Benavoli A, Corani G, Demšar J, Zaffalon M. Time for a change: a tutorial for comparing multiple classifiers through bayesian analysis. The Journal of Machine Learning Research 2017;18(1):2653–88.

[8] Blum C, Sampels M. Ant colony optimization for fop shop scheduling: a case study on different pheromone representations. In: Proc. of the 2002 Congress on Evolutionary Computation. 2002. p. 1558–1563 vol.2.

[9] Calvo B, Santafe G. scmamp: Statistical comparison of multiple algorithms in multiple problems. The R Journal 2015;8(1):248–56.

[10] Campos V, Glover F, Laguna M, Martí R. An experimental evaluation of a scatter search for the linear ordering problem. Journal of Global Optimization 2001;21(4):397–414.

[11] Ceberio J, Irurozki E, Mendiburu A, Lozano JA. A distance-based ranking model estimation of distribution algorithm for the flowshop scheduling problem. IEEE Transactions on Evolutionary Computation 2014;18(2):286–300.

[12] Ceberio J, Mendiburu A, Lozano JA. The Linear Ordering Problem Revisited. European Journal of Operational Research 2014;241(3):686–96.

[13] Charon I, Hudry O. A survey on the linear ordering problem for weighted or unweighted tournaments. 4OR 2007;5(1):5–60.

[14] Chenery HB, Watanabe T. International comparisons of the structure of production. Econometrica 1958;26(4):487–521.

[15] Chira C, Pintea CM, Crisan GC, Dumitrescu D. Solving the linear ordering problem using ant models. In: Proceedings of the 11th Annual conference on Genetic and evolutionary computation. New York, NY, USA: ACM; GECCO '09; 2009. p. 1803–4.

[16] Cormen TH, Leiserson CE, Rivest RL, Stein C. Introduction to algorithms. MIT press, 2009.

[17] Garcia CG, Pérez-Brito D, Campos V, Martí R. Variable neighborhood search for the linear ordering problem. Computers & Operations Research 2006;33(12):3549 –65.

[18] Garey MR, Johnson DS. " Strong " NP-completeness results: Motivation, examples, and implications. J ACM 1978;25(3):499–508.

[19] Garey MR, Johnson DS. Computers and Intractability: A Guide to the Theory of NP-Completeness. New York, USA: W. H. Freeman & Co., 1979.

[20] Glover F, Klastorin T, Klingman D. Optimal weighted ancestry relationships. Management science report series. Business Research Division, Graduate School of Business Administration, University of Colorado, 1972.

[21] Grötschel M, Jünger M, Reinelt G. A cutting plane algorithm for the linear ordering problem. Operations research 1984;32(6):1195–220.

[22] Hoos H, Sttzle T. Stochastic Local Search: Foundations & Applications. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2004.

[23] Kemeny JG. Mathematics without numbers. Daedalus 1959;88:577–91.

[24] Knuth D. The Stanford GraphBase: A Platform for Combinatorial Computing. ACM Press, 1993.

[25] Leontief W. Input-Output Economics. Oxford University Press, 1966.

[26] Leontief WW. Quantitative input and output relations in the economic systems of the united states. The Review of Economics and Statistics 1936;18(3):105–25.

[27] Martí R, Reinelt G. The linear ordering problem: exact and heuristic methods in combinatorial optimization. volume 175. Springer, 2011.

[28] Martí R, Reinelt G, Duarte A. A benchmark library and a comparison of heuristic methods for the linear ordering problem. Comput Optim Appl 2012;51(3):1297–317.

[29] Mishra S, Sikdar K. On approximability of linear ordering and related NP-optimization problems on graphs. Discrete Applied Mathematics 2004;136(2):249 –69. The 1st Cologne-Twente Workshop on Graphs and Combinatorial Optimization.

[30] Mitchell J, Borchers B. Solving real-world linear ordering problems using a primal-dual interior point cutting plane method. Annals of Operations Research 1996;62(1):253–76.

[31] Mitchell J, Borchers B. Solving linear ordering problems with a combined interior point/simplex cutting plane algorithm. In: Frenk H, Roos K, Terlaky T, Zhang S, editors. High Performance Optimization. Springer US; volume 33 of *Applied Optimization*; 2000. p. 349–66.

[32] Mladenovic N, Hansen P. Variable neighborhood search. Computers & Operations Research 1997;24(11):1097 –100.

[33] Newman A, Vempala S. Fences are futile: On relaxations for the linear ordering problem. In: Aardal K, Gerards B, editors. Integer Programming and Combinatorial Optimization. Berlin, Heidelberg: Springer Berlin Heidelberg; 2001. p. 333–47.

[34] Sakuraba CS, Ronconi DP, Birgin EG, Yagiura M. Metaheuristics for large-scale instances of the linear ordering problem. Expert Systems with Applications 2015;42(9):4432 –42.

[35] Sakuraba CS, Yagiura M. Efficient local search algorithms for the linear ordering problem. International Transactions in Operational Research 2010;17(6):711–37.

[36] Santucci V, Baioletti M, Milani A. An algebraic differential evolution for the linear ordering problem. In: Proceedings of the Companion Publication of the 2015 Conference on Genetic and Evolutionary Computation. New York, USA: ACM; 2015. p. 1479–80.

[37] Schiavinotto T, Stützle T. Search Space Analysis of the Linear Ordering Problem. In: Applications of Evolutionary Computing. Springer Berlin Heidelberg; volume 2611 of *LNCS*; 2003. p. 322–33.

[38] Schiavinotto T, Stützle T. The linear ordering problem: instances, search space analysis and algorithms. Journal of Mathematical Modelling and Algorithms 2004;.

[39] Simon HA, Newell A. Heuristic problem solving: The next advance in operations research. Operations Research 1958;6(1):1–10.

[40] Slater P. Inconsistencies in a schedule of paired comparisons. Biometrika 1961;48(3/4):303–12.

[41] Song J, Zhao H, Zhou T, Tao Y, Lü Z. Solving the linear ordering problem via a memetic algorithm. In: Proceedings of the Future Technologies Conference (FTC) 2018. Cham: Springer International Publishing; 2019. p. 421–30.

[42] Sörensen K. Metaheuristicsthe metaphor exposed. International Transactions in Operational Research 2015;22(1):3–18.

Appendix A. Complexity of C-LOP

Here, we analyze the time complexity of C-LOP when invoked starting from an empty set of precedences $P = \emptyset$, i.e., the worst possible input for P, while we do not make any assumption on the parameter α and the problem instance H.

We start by describing the data structure adopted. A partial solution P is maintained by using three lists for each item $i \in [n]$: the predecessors of i in P, the successors of i in P, and the items without any relation with i in P. For each list, we also maintain their inverted maps. This structure allows us to efficiently add a new precedence in P and to quickly compute its transitive closure by efficiently scanning the assigned and unassigned precedences. Moreover, in order to perform the roulette wheel selection (line 8, algorithm 2) in $O(n)$ time, we maintain the row sums of H by taking into account only the entries corresponding to the precedences in C. Lastly, we initially sort the H entries in descending order and we set a pointer to the largest (first) entry. This allows us to efficiently select the largest heuristic value required in line 6 of algorithm 2.

Since C-LOP is a randomized heuristic, we analyze both its worst case and average case execution scenarios. Obviously, the latter is much more important for the context where we use C-LOP.

Note that, at any iteration of the main loop, at least one precedence is introduced in P. Therefore, the maximum number of $\binom{n}{2} = \Theta(n^2)$ iterations is obtained when exactly one precedence per iteration is introduced in P. In such a case, the transitive closure does not add any extra precedence, so the complexity of a single iteration is only given by the precedence selection procedures. Since the H entries have been ordered, the selection in line 6 simply requires to move the pointer to the next available precedence. Hence, the whole complexity of an iteration is given by the roulette wheel procedure that costs $O(n)$. Therefore, in the worst case scenario, C-LOP requires $O(n^3)$ operations.

Nevertheless, in the average case, C-LOP requires a more affordable computational time. Given a consistent set P of precedences, we say that a permutation π agrees with P if and only if P is a subset of the precedences set representation of π. Initially, $P = \emptyset$ and all the $n!$ permutations of $\mathcal{S}_n$ agree with P. Since at every iteration a randomly selected precedence $i \prec j$ is added to P (together with its induced precedences), we have that the set of permutations which agree with P is roughly halved with respect to the previous iteration (in average, half of the permutations agreeing with P have $i \prec j$) . Therefore, after about $\log_2 n!$ iterations, only one permutation agrees with P, thus $|P| = \binom{n}{2}$. Since, by Stirling approximation [16], $\log_2 n! = \Theta(n \log n)$, we have that C-LOP requires $\Theta(n \log n)$ iterations in the average case. Now, by amortized analysis, we derive the average complexity of a single iteration. Since a complete set of precedences has cardinality $\binom{n}{2} = \Theta(n^2)$, the average number of precedences added to P in a single iteration is $\Theta(n^2/(n \log n)) = \Theta(n/\log n) = O(n)$. Hence, also the pointer to the largest entry of H is moved by no more than $O(n)$ steps. Furthermore, as before, the roulette wheel selection requires $O(n)$ time. Summing up, we have $\Theta(n \log n)$ iterations, each one with an average cost of $O(n)$ steps. As a consequence, the average case complexity of C-LOP is $O(n^2 \log n)$.

Appendix B. Best Known results

Instance	Fitness	Instance	Fitness	Instance	Fitness
t1d100.01	**106852**	t1d150.10	**234821**	t1d200.18	**407728 ***
t1d100.02	**105947**	t1d150.11	**234157**	t1d200.19	412825
t1d100.03	**109819**	t1d150.12	**236318**	t1d200.20	406418
t1d100.04	**109252**	t1d150.13	**237116**	t1d200.21	408037
t1d100.05	108847	t1d150.14	234453	t1d200.22	**407339 ***
t1d100.06	**108201**	t1d150.15	232065	t1d200.23	408552
t1d100.07	**108803**	t1d150.16	**232948**	t1d200.24	410583
t1d100.08	**107480**	t1d150.17	**236656**	t1d200.25	**406476 ***
t1d100.09	**108549**	t1d150.18	234348	t1d500.01	**2402576 ***
t1d100.10	108755	t1d150.19	**234994**	t1d500.02	2411570
t1d100.11	107920	t1d150.20	**235411**	t1d500.03	2404784
t1d100.12	**108389**	t1d150.21	233956	t1d500.04	**2414133 ***
t1d100.13	**108702**	t1d150.22	**235415**	t1d500.05	2391486
t1d100.14	**105583**	t1d150.23	**233492**	t1d500.06	2399394
t1d100.15	**108667**	t1d150.24	**236016**	t1d500.07	2400739
t1d100.16	**107426**	t1d150.25	**236428**	t1d500.08	**2413914 ***
t1d100.17	**105612**	t1d200.01	**410992 ***	t1d500.09	2406223
t1d100.18	**107861**	t1d200.02	407729	t1d500.10	**2404744 ***
t1d100.19	**108026**	t1d200.03	407223	t1d500.11	2416286
t1d100.20	**109968**	t1d200.04	410101	t1d500.12	2402581
t1d100.21	**107255**	t1d200.05	411522	t1d500.13	2405118
t1d100.22	**108250**	t1d200.06	**406451**	t1d500.14	2410693
t1d100.23	**106146**	t1d200.07	412482	t1d500.15	**2411961 ***
t1d100.24	**108782**	t1d200.08	408850	t1d500.16	2416067
t1d100.25	**106933**	t1d200.09	**409308**	t1d500.17	2401800
t1d150.01	234928	t1d200.10	406453	t1d500.18	2421159
t1d150.02	**234421**	t1d200.11	410159	t1d500.19	2404029
t1d150.03	**236319**	t1d200.12	412831	t1d500.20	2414713
t1d150.04	**234510 ***	t1d200.13	**409270 ***	t1d500.21	2405615
t1d150.05	234601	t1d200.14	408879	t1d500.22	2408164
t1d150.06	**234465**	t1d200.15	**409061**	t1d500.23	2408689
t1d150.07	235283	t1d200.16	408059	t1d500.24	**2402740 ***
t1d150.08	237230	t1d200.17	**410280**	t1d500.25	2405718
t1d150.09	237253				

Table B.2: Best results obtained by CD-RVNS for the *RandA1* benchmark. Boldfaced results denote best known values, and those marked with (*) identify **new** best known results.

Instance	Fitness	Instance	Fitness	Instance	Fitness
t2d100.01	**25362**	t2d150.01	**76041**	t2d200.01	147740
t2d100.02	**28326**	t2d150.02	**73624**	t2d200.02	**144218**
t2d100.03	**25886**	t2d150.03	**69705**	t2d200.03	**141378**
t2d100.04	**26076**	t2d150.04	**73963**	t2d200.04	150870
t2d100.05	**25118**	t2d150.05	**79723**	t2d200.05	**150236**
t2d100.06	**25380**	t2d150.06	**75440**	t2d200.06	**141254**
t2d100.07	**27144**	t2d150.07	**73858**	t2d200.07	**149752**
t2d100.08	**23784**	t2d150.08	**67463**	t2d200.08	**149910**
t2d100.09	**27752**	t2d150.09	**70739**	t2d200.09	**141958**
t2d100.10	**26690**	t2d150.10	**69029**	t2d200.10	149628
t2d100.11	**25106**	t2d150.11	**72800**	t2d200.11	147540
t2d100.12	**26782**	t2d150.12	**72181**	t2d200.12	**152470**
t2d100.13	**27878**	t2d150.13	**74580**	t2d200.13	**137618**
t2d100.14	**25878**	t2d150.14	**68132**	t2d200.14	**144384**
t2d100.15	**24232**	t2d150.15	**76831**	t2d200.15	**140442**
t2d100.16	**28206**	t2d150.16	**72018**	t2d200.16	**147448**
t2d100.17	**26704**	t2d150.17	**70185**	t2d200.17	**131874**
t2d100.18	**26928**	t2d150.18	**73191**	t2d200.18	**151196**
t2d100.19	**28760**	t2d150.19	**75958**	t2d200.19	**137314**
t2d100.20	**25220**	t2d150.20	**67370**	t2d200.20	**146508**
t2d100.21	**24452**	t2d150.21	**70297**	t2d200.21	**143568**
t2d100.22	**27230**	t2d150.22	**69287**	t2d200.22	**146920**
t2d100.23	**25588**	t2d150.23	**74799**	t2d200.23	**145034**
t2d100.24	**24800**	t2d150.24	**70063**	t2d200.24	**151260**
t2d100.25	**23742**	t2d150.25	**73853**	t2d200.25	**149128**

Table B.3: Best results obtained by CD-RVNS for the *RandA2* benchmark. Results in bold are the best values reported in the literature.

Instance	Fitness	Instance	Fitness	Instance	Fitness
p40-01	**29457**	p44-11	**34016**	p44-41	**48137**
p40-02	**27482**	p44-12	**33850**	p44-42	**49511**
p40-03	**28061**	p44-13	**35385**	p44-43	**51014**
p40-04	**28740**	p44-14	**35801**	p44-44	**51949**
p40-05	**27450**	p44-15	**33827**	p44-45	**52857**
p40-06	**29164**	p44-16	**36188**	p44-46	**52776**
p40-07	**28379**	p44-17	**35454**	p44-47	**54122**
p40-08	**28267**	p44-18	**36669**	p44-48	**54355**
p40-09	**30578**	p44-19	**36436**	p44-49	**57279**
p40-10	**31737**	p44-20	**37438**	p44-50	**56444**
p40-11	**30658**	p44-21	**37786**	p50-01	**44667**
p40-12	**30986**	p44-22	**36722**	p50-02	**43835**
p40-13	**33903**	p44-23	**36605**	p50-03	**44256**
p40-14	**34078**	p44-24	**38286**	p50-04	**43928**
p40-15	**34659**	p44-25	**38129**	p50-05	**42907**
p40-16	**36044**	p44-26	**39107**	p50-06	**42325**
p40-17	**38201**	p44-27	**39170**	p50-07	**42640**
p40-18	**37562**	p44-28	**40264**	p50-08	**42666**
p40-19	**38956**	p44-29	**41819**	p50-09	**43711**
p40-20	**39658**	p44-30	**40387**	p50-10	**43575**
p44-01	**35948**	p44-31	**43817**	p50-11	**43527**
p44-02	**35314**	p44-32	**42545**	p50-12	**42809**
p44-03	**34335**	p44-33	**42355**	p50-13	**43169**
p44-04	**33551**	p44-34	**44988**	p50-14	**44519**
p44-05	**34827**	p44-35	**44114**	p50-15	**44866**
p44-06	**33962**	p44-36	**45575**	p50-16	**45310**
p44-07	**33171**	p44-37	**45297**	p50-17	**46011**
p44-08	**34127**	p44-38	**47414**	p50-18	**46897**
p44-09	**33403**	p44-39	**48979**	p50-19	**47212**
p44-10	**33778**	p44-40	**47774**	p50-20	**46779**

Table B.4: Best results obtained by CD-RVNS for the *RandAB* benchmark. Results in bold are the best values reported in the literature.

Instance	$n = 150$	$n = 250$
N-be75eec	**3482740***	**8900531***
N-be75np	**7182409***	**17794274***
N-be75oi	**2246816***	**5911016***
N-be75tot	**12288727***	**30967397***
N-stabu1	**2873330***	7728901
N-stabu2	**4327870***	**11501824***
N-stabu3	**4510445***	**11901168***
N-t59b11xx	**3239480***	**8398070***
N-t59d11xx	**1462418***	**3839167***
N-t59f11xx	**1543733***	**3986281***
N-t59n11xx	**318934***	**824554***
N-t65b11xx	**6455861***	**17256477***
N-t65d11xx	**3558388***	**9346304***
N-t65f11xx	**3156457***	**8409733***
N-t65l11xx	**253344***	**666851***
N-t65n11xx	**550569***	**1429072***
N-t69r11xx	**11858249***	**31784901***
N-t70b11xx	**9642436***	**25386762***
N-t70d11xn	**5820630***	**15192999***
N-t70d11xx	**6174178***	**16027224***
N-t70f11xx	**5145527***	**13565303***
N-t70l11xx	**436882***	**1111703***
N-t70n11xx	**948896***	**2443448***
N-t74d11xx	**9391042***	**24426402***
N-t75d11xx	**9639371***	**25044635***
N-t75e11xx	**41571407***	**106699067***
N-t75k11xx	**1541594***	**4093582***
N-t75n11xx	**1740685***	**4518265***
N-tiw56n54	**837155***	**2096274***
N-tiw56n58	**1155333***	**2901784***
N-tiw56n62	**1626254***	**4141193***
N-tiw56n66	**2107619***	**5366303***
N-tiw56n67	**2372805***	**6318149***
N-tiw56n72	**4135952***	**11148631***
N-tiw56r54	**957718***	**2385963***
N-tiw56r58	**1219043***	**3060680***
N-tiw56r66	**1940755***	**4944620***
N-tiw56r67	**2056123***	**5284335***
N-tiw56r72	**2823771***	**7454042***

Table B.5: Best results obtained by CD-RVNS for the *xLOLIB* benchmark. Boldfaced results denote best known values, and those marked with (*) identify **new** best known results.

Instance	$n = 300$	$n = 500$	$n = 750$	$n = 1000$
N-be75eec	**12401915***	**33335021***	**71335951***	**122183020***
N-be75np	**26058695***	**66706038***	142235433	**245965411***
N-be75oi	**9389582***	**25344958***	**57446206***	95107894
N-be75tot	**43728689***	**113769897***	**246647833***	**420848949***
N-stabu70	**9980631***	**27206945***	**58260994***	**100678451***
N-stabu74	**15007346***	**41098727***	**87467424***	**151099343***
N-stabu75	**15561023***	**42671550***	**90599301***	**156432889***
N-t59b11xx	**10400625***	**27630465***	**59760540***	101988220
N-t59d11xx	**5025078***	**13246941***	**29824157***	**50832719***
N-t59f11xx	**5070127***	**13437621***	**29041799***	**49213897***
N-t59i11xx	**360306443***	**936090712***	**2062282459***	**3471997519***
N-t59n11xx	**1004921***	**2610904***	**5667969***	**9569747***
N-t65b11xx	**22149491***	**59610554***	**129443488***	**222403731***
N-t65d11xx	**11864597***	**31716674***	**67649620***	**116725645***
N-t65f11xx	**11166902***	**29285209***	**63208271***	**108197222***
N-t65i11xx	**862881768***	**2244382489***	**4929127538***	8390948152
N-t65l11xx	**827462***	**2328136***	**4917873***	8668534
N-t65n11xx	**1788165***	**4651136***	**10042759***	**17044635***
N-t65w11xx	**7339707255***	**19371907957***	**41737976722***	**71696369800***
N-t69r11xx	**41051301***	**108552960***	235618981	**397093542***
N-t70b11xx	**31628508***	**84271904***	**181536910***	**312618951***
N-t70d11xx	**20804250***	**55349813***	**118269267***	**204397975***
N-t70d11xxb	**19620114***	**52016871***	**112175736***	**193829282***
N-t70f11xx	**17936682***	**47857287***	**103368757***	**177714930***
N-t70i11xx	**1347488506***	**3486936667***	**7611464838***	**13019571398***
N-t70k11xx	**2906605200***	**7688243300***	**16654285500***	**28351747000***
N-t70l11xx	**1429357***	**3800456***	**7870995***	**14118116***
N-t70n11xx	**3053791***	**7949514***	**16959052***	**28863606***
N-t70u11xx	**1054204000***	**2753258600***	**5936431800***	**10228859000***
N-t70w11xx	**11924272532***	**31482656885***	**68013529104***	**117160728531***
N-t70x11xx	**14699161134***	**38846619405***	**83877382884***	**144521167089***
N-t74d11xx	**31692802***	**84276952***	**180000000***	**310738504***
N-t75d11xx	**32626939***	**86590015***	**184396164***	319087189
N-t75e11xx	**145044286***	**374756997***	**818104894***	**1388110876***
N-t75i11xx	**3771954709***	9733195049	**21233222778***	**36319330625***
N-t75k11xx	**5329666***	**14159503***	**30431677***	**52008325***
N-t75n11xx	**5767516***	**14882254***	**31793642***	**53916364***
N-t75u11xx	**3081481709***	**8062758937***	**17292801492***	**29557647434***
N-tiw56n54	**2654641***	**6973683***	**15123184***	**25823018***
N-tiw56n58	**3603597***	**9513598***	**20633749***	**35281101***
N-tiw56n62	**5148045***	**13560366***	**29427585***	**50477512***
N-tiw56n66	**6673467***	**17629994***	**38223009***	**65543988***
N-tiw56n67	**7684860***	**20583929***	**44623467***	**75645448***
N-tiw56n72	**13180215***	**35188140***	**76855797***	130000246
N-tiw56r54	**2989753***	**7880480***	**17095151***	**29147432***
N-tiw56r58	**3757785***	**9956130***	**21591768***	**36886320***
N-tiw56r66	**6179233***	**16283320***	**35321403***	**60577648***
N-tiw56r67	**6852120***	**17971377***	**39570834***	66272465
N-tiw56r72	**8974248***	**23635996***	**51520889***	**87558976***
N-usa79	**28509942***	**75962230***	**157251449***	272318160

Table B.6: Best results obtained by CD-RVNS for the *xLOLIB2* benchmark. Boldfaced results denote best known values, and those marked with (*) identify **new** best known results.