GRASP with path relinking for the single row facility layout problem

Manuel Rubio-Sánchez, Micael Gallego, Francisco Gortázar, Abraham Duarte*

Department of Computer Science and Statistics, Universidad Rey Juan Carlos, Spain

**Abstract**

The single row facility layout problem (SRFLP) is an NP-hard problem that consists of finding an optimal arrangement of a set of rectangular facilities (with equal height and different lengths), placing them next to each other along a line. The SRFLP has practical applications in contexts such as arranging rooms along corridors, setting books on shelves, allocating information on magnetic disks, storing items in warehouses, or designing layouts for machines in manufacturing systems. This paper combines the greedy randomized adaptive search procedure (GRASP) methodology, and path relinking (PR) in order to efficiently search for high-quality solutions for the SRFLP. In particular, we introduce: (i) several construction procedures, (ii) a new fast local search strategy, and (iii) an approach related to the Ulam distance in order to construct short path relinking trajectories. We also present a new set of large challenging instances, since previous sets do not allow to determine significant differences among advanced metaheuristics. Experiments show that our procedure outperforms state-of-the-art methods in all of the scenarios we considered. Firstly, the GRASP with PR finds the best known solutions for previous instances used in the literature, but employing considerably less computing time than its competitors. Secondly, our method outperforms the current state-of-the-art methods in 38 out of 40 new instances when running for the same amount of computing time. Finally, nonparametric tests for detecting differences between algorithms report p-values below $10^{-11}$, which supports the superiority of our approach.

© 2016 Elsevier B.V. All rights reserved.

1. Introduction

Facility location problems are concerned with finding optimal locations of facilities (machine tools, work centers, manufacturing cells, machine shops, etc.) in a given area. Their objective function may reflect several types of costs (e.g., transportation, transmission, or communication), or simply adjacency preferences among machines. In this paper, we focus on the single row facility layout problem (SRFLP), also known as the one-dimensional space allocation problem [51]. It has been applied in several domains in order to solve problems related to the arrangement of rooms along corridors (e.g., hospitals or office buildings), setting books on shelves, allocating information on magnetic disks, storing items in warehouses, or designing layouts for machines in manufacturing systems [20,39,51].

The SRFLP is an NP-hard problem that consists of finding an optimal arrangement of a set of rectangular facilities, placing them next to each other along a line. In particular, the goal is to obtain an ordering of the facilities that minimizes a weighted sum of the distances between the centers of all pairs of facilities. Formally, the SRFLP is defined as follows: let $F = \{1, 2, \ldots, n\}$ be a set of $n > 2$ rectangular facilities with fixed height but different lengths $l_i > 0$, for $i \in F$. Additionally, let $c_{ij} = c_{ji} \geq 0$, for $i, j \in F$, be the weight between facilities $i$ and $j$, which usually models some transmission cost between them. A particular solution to this problem is an ordering $\pi = (\pi(1), \pi(2), \ldots, \pi(n))$ of the facilities in $F$, whose cost $C(\pi)$ is defined according to the following objective function:

$$C(\pi) = \sum_{1 \leq q < r \leq n} c_{\pi(q)\pi(r)} \cdot d_{\pi(q)\pi(r)},$$

where $d_{\pi(q)\pi(r)}$ represents the distance between the centers of facilities $\pi(q)$ and $\pi(r)$ (i.e., located in the ordering $\pi$ at positions $q$ and $r$, respectively), and is computed as:

$$d_{\pi(q)\pi(r)} = l_{\pi(q)}/2 + \sum_{q < s < r} l_{\pi(s)} + l_{\pi(r)}/2.$$

The optimization problem therefore consists of finding an ordering $\pi^*$ that minimizes (1). Formally:

$$\pi^* = \arg\min_{\pi \in \Pi_n} C(\pi),$$

where $\Pi_n$ is the set of permutations of the first $n$ positive integers.

An instance of the SRFLP of size $n = |F|$ is therefore defined by specifying the list $l$ of facility lengths (of size $n$), and a symmetric
Fig. 1. Instance of size \( n = 5 \) of the SRLP, defined through a list \( l \) of facility lengths, and a symmetric square weight matrix \( c \).

\( n \times n \) cost matrix \( c \) that contains the weights between the facilities. Fig. 1 shows an instance of size \( n = 5 \). Note that the facility lengths \( \{l_i\} \) range from 2 (facility 5) to 6 (facility 3). Additionally, the diagonal elements of \( c \) are typically set to 0 since the objective function does not consider them, but other off-diagonal weights may also be equal to 0, indicating that a pair does not contribute to the cost function, regardless of the placement among an ordering of the particular facilities.

Fig. 2 illustrates a solution for the instance described in Fig. 1. It is represented as the ordering \( \pi = (4, 5, 1, 2, 3) \), which means that facility 4 is placed in the first position of the ordering (i.e., \( \pi(1) = 4 \)), followed by facility 5 (i.e., \( \pi(2) = 5 \)), and so on. The distance between a pair of facilities in a given ordering \( \pi \) involves computing the distance between their centers. In the example the distance between facilities 4 and 5 is \( d_{45} = (1/2) \times l_4 + (1/2) \times l_5 = 3 \). Similarly, the distance between facility 5 and 3 is \( d_{53} = (1/2) \times l_5 + l_1 + l_2 + (1/2) \times l_3 = 12 \). The contribution of a pair of facilities to the objective function is computed as the product of the distance and weight between the facilities. For example, the contribution of the pair \( (4, 5) \) is \( d_{45} \times c_{45} = 3.0 \times 5 = 15 \). Similarly, for facilities 5 and 3, the contribution is \( d_{53} \times c_{53} = 12 \times 0 = 0 \). In the figure we have illustrated these numerical contributions next to bent lines connecting the associated pairs of facilities. In particular, the shade of gray of a line depends on the weight between the facilities (larger weights are represented by darker shades of gray). Lastly, for the sake of clarity, we have only included nonzero contributions to the cost.

In this paper we propose an adaptation of the greedy randomized adaptive search procedure (GRASP) methodology [14] in order to efficiently search for high-quality solutions for the SRLP. In particular, we introduce several constructive procedures with a different trade-off between intensification and diversification. Furthermore, we present a new fast local search strategy that is based on a hybrid approach between the classical first and best improvement search methods. We additionally present a post-processing strategy based on path relinking [18], where we construct short path trajectories through an approach related to the ULam distance between permutations [53]. Lastly, this work introduces a new set of more challenging instances that contain a larger number of facilities than those previously used throughout the literature.

The rest of this paper is organized as follows. Section 2 describes the most relevant approaches presented in the related literature. The GRASP approach is introduced in Section 3, where we focus on the description of the proposed constructive and local search algorithms. Section 4 describes several strategies for generating trajectories in the context of path relinking, while Section 5 introduces the proposed combination of GRASP and path relinking. Finally, Section 6 reports the results of computational experiments, and Section 7 summarizes the most relevant conclusions.

2. Literature review

There exists an extensive literature related to the family of facility location problems. Among them, the SRLP currently emerges as one of the most active problems. Surveys regarding the state of the art on the SRLP are available in Keller and Buscher [25], Kothari and Ghosh [26]. Researchers have developed exact algorithms for solving the problem that include branch-and-bound strategies [51], dynamic programming [39], linear mixed integer programming [23,21,37], branch and cut [5], cutting planes [44,49], semidefinite programming [8,9,23], or a combination of the last two approaches [7]. Currently, optimal solutions are known for instances of at most 42 facilities. For larger sized instances recent research has focused on employing efficient metaheuristics in order to search for high-quality approximate solutions, since exact methods are currently computationally prohibitive. Finally, exact methods have also been proposed for a variant in which all of the facilities have the same length [22].

From a heuristic perspective, construction procedures are the simplest ones. These algorithms consider either the weights between the facilities [20,33], or the length of the facilities [47], and obtain solutions very efficiently. However, their quality is not acceptable for real applications. Therefore, these approaches are generally used as part of more sophisticated metaheuristics. For instance, the construction approach proposed in Samarghandi and Eshghi [47] has been used within a Tabu Search framework, for a multi-start strategy that improves each constructed solution with a Lin–Kernighan insertion neighborhood [29], or as an initial seed to the diversification stage of the scatter search approach in Kothari and Ghosh [32].

A variety of other metaheuristics have also been used to tackle the SRLP. These include genetic algorithms [11,31], particle swarm [48] and ant colony optimization [52], simulated annealing [19,45], or scatter search [32,34]. In addition, some approaches combine different metaheuristic strategies. For instance, simulated annealing is coupled with genetic algorithms in Ramkumar and Ponnambalam [40], while in Kothari and Ghosh [27] path relinking is applied to solutions generated by the approaches in Kothari and Ghosh [29,30,32].

To the best of our knowledge, the best approaches for finding approximate solutions for the SRLP through metaheuristics are the ones introduced in Kothari and Ghosh [31] and Kothari and Ghosh [32]. Specifically, the first work presents a straightforward genetic algorithm that uses the partially matched crossover (PMX) operator (see Larrañaga et al. [36] for a detailed description of this operator), and mutations based on insert moves. Additionally, it improves solutions by running a local search algorithm based on insert moves and by considering the best improvement strategy. The second paper proposes a scatter search method that borrows components from the first. For instance, it also uses the PMX operator in order to combine solutions, and uses the same exhaustive local search strategy. The paper uses an additional alternating combination approach, and proposes two diversification strategies in order to generate an initial reference set of solutions. The genetic algorithm and the scatter search approach were able to obtain 42 and 41 of the best solutions found so far, respectively, on the 43 benchmark instances used in the recent literature. This indicates a need to analyze the algorithms on larger and more challenging instances, since the algorithms often find the same solutions, which are likely the optimal ones. Lastly, the slightly better performance of the genetic algorithm can be due to the fact that it employs considerably more computing time.

3. GRASP

The GRASP methodology is a metaheuristic developed in the late 1980s [14] and formally introduced in Feo et al. [13]. Recent and thorough surveys of the method are presented in Resende and Ribeiro [42,43]. GRASP is a multi-start methodology where each iteration consists of two stages. The first performs a greedy,
randomized, and adaptive construction of a solution. The second stage improves the constructed solution (according to the objective function of the problem) by applying a search procedure until reaching a local optimum. These two steps are repeated until a termination criterion is met. The rest of this section is organized as follows. Section 3.1 introduces constructive procedures for the SRFLP, while Section 3.2 describes local search algorithms.

3.1. Constructive methods

A GRASP constructive procedure builds a solution \( \pi \) of size \( n \) for the SRFLP iteratively by inserting facilities one at a time (through random and greedy strategies) in a partial solution \( \pi_p \) containing less than \( n \) facilities. For this purpose we propose the three greedy functions illustrated in Fig. 3. The first one, denoted as \( g_1 \), appends new facilities unidirectionally (e.g., from left to right) to only one extreme of the partial sequence. A second approach, called \( g_2 \), considers appending a facility to either extreme. Finally, a third strategy, denoted as \( g_3 \), contemplates the possibility of inserting a facility at any location along the partial solution.

In order to select which facility to incorporate, as well as its location along \( \pi_p \) when using \( g_2 \) or \( g_3 \), we rely on the fact that the cost of a partial solution with \( m \leq n \) facilities can be evaluated. In particular, we measure the cost of a partial solution according to the objective function of the problem, but by only considering the facilities that belong to the partial sequence. Formally:

\[
C(\pi_p) = \sum_{1 \leq q < r \leq m} c_{\pi_p(q)\pi_p(r)} \cdot d_{\pi_p(q)\pi_p(r)}.
\]

On the one hand, this partial cost allows to determine the best location along \( \pi_p \) for a new facility when using \( g_2 \) or \( g_3 \), i.e., the one that minimizes \( C(\pi_p) \). On the other hand, we rank the facilities not yet included in \( \pi_p \) according to these optimal costs, which allows to apply greedy and randomized strategies for selecting the particular facility that will be incorporated into \( \pi_p \) at each iteration. Lastly, note that, in the context of metaheuristics, \( g_1 \) favors diversification (i.e., randomness and broad explorations of the search space), while \( g_3 \) favors intensification (i.e., greediness and exploitation of regions in the search space). Thus, \( g_2 \) can be considered as a compromise between both approaches.

In this paper we use two families of constructive algorithms. The first one follows the standard GRASP template. Algorithm 1, denoted as Greedy-Random (GR), shows the corresponding pseudocode. It starts by creating a list of candidates (CL), which contains the facilities that have not yet been included in the partial solution under construction. Initially, CL contains every facility (step 1). Then, the method randomly selects a first facility \( i \) from CL (step 2), includes it in the partial solution \( \pi_p \) (step 3), and updates the candidate list by removing the chosen facility (step 4). The method then iterates until it obtains a solution with \( n = |F| \) facilities (steps 5 to 11). At each iteration this family of algorithms calculates a restricted candidate list (RCL) by selecting the best size facilities from CL (step 7). This is carried out by previously sorting the facilities according to the rankings associated with a particular greedy function. The value of size depends on a parameter \( \alpha \in [0, 1] \) (step 6).

![Fig. 2. Example of solution \( \pi = \{4, 5, 1, 2, 3\} \) for the instance defined in Fig. 1, whose cost is 110. The (positive) contributions of each pair of facilities to the objective function of the problem are illustrated next to bent lines whose color is associated with the value of the weight between the facilities (in particular, larger weights are represented by darker shades of gray).](image-url)

![Fig. 3. Greedy functions. A new facility can be incorporated into a partial solution by appending it to one extreme of the sequence \( \{g_1\} \), to the best of both extremes \( \{g_2\} \), or by inserting it at the best possible location \( \{g_3\} \).](image-url)

Algorithm 1: Greedy-Random (GR).

1: CL ← F
2: \( i \leftarrow \text{SelectRandom}(\text{CL}) \)
3: \( \pi_p \leftarrow \langle i \rangle \)
4: CL ← CL \( \setminus \{i\} \)
5: while \( CL \neq \emptyset \) do
6: \( \text{size} \leftarrow \max\{ \lfloor \alpha \cdot |CL| \rfloor, 1 \} \)
7: RCL ← SelectBest(CL, \( \pi_p \), \( \text{size} \))
8: \( i' \leftarrow \text{SelectRandom}(\text{RCL}) \)
9: \( \pi_p \leftarrow \text{IncludeFacility}(\pi_p, i') \)
10: CL ← CL \( \setminus \{i'\} \)
11: end while
12: return \( \pi_p \)
Algorithm 2: Random-Greedy (RG).

1: $CL \leftarrow F$
2: $i \leftarrow \text{SelectRandom}(CL)$
3: $\pi_p \leftarrow \{i\}$
4: $CL \leftarrow CL \setminus \{i\}$
5: while $CL \neq \emptyset$ do
6:   $\text{size} \leftarrow \max\{\lfloor \alpha \cdot |CL| \rfloor, 1\}$
7:   $RCL \leftarrow \text{SelectRandom}(CL, \text{size})$
8:   $r^* \leftarrow \text{SelectBest}(RCL, \pi_p)$
9:   $\pi_p \leftarrow \text{IncludeFacility}(\pi_p, r^*)$
10: $CL \leftarrow CL \setminus \{r^*\}$
11: end while
12: return $\pi_p$

The second family of constructive procedures, denoted as Random-Greedy (RG), is based on a different strategy introduced in Resende and Werneck [44], which has been applied with success recently in Campos et al. [10], Duarte et al. [12], Pantrigo et al. [38], Resende et al. [41]. Specifically, this alternative construction swaps the greedy and random stages of a standard GRASP construction, as illustrated in Algorithm 2. In this case, $RCL$ contains size elements that are selected at random from $CL$ (step 7), while in step 8 the best facility is greedily chosen from $RCL$.

Lastly, note that $\alpha$ controls the greediness/randomness of the GRASP constructive procedures. Specifically, if $\alpha = 0$ the corresponding methods are purely greedy algorithms, while if $\alpha = 1$ they are totally random procedures.

3.2. Local search

The second stage of a GRASP algorithm consists in improving the constructed solutions by using a local search method. The idea consists of applying some strategy progressively that modifies an ordering of facilities in order to decrease its cost according to (1). This search process, denoted as move operator, is repeated until no further improvements are possible (i.e., it guides the search towards a local optimum).

In particular, for the SRFLP we define two different move operators. The first one is referred to as swap. Given a solution $\pi = (\pi(1), \ldots, \pi(q), \ldots, \pi(r), \ldots, \pi(n))$, we define $\text{swap}(\pi, q, r)$ as the move that exchanges in $\pi$ the facility $\pi(q)$ in position $q$ with the facility $\pi(r)$ in position $r$, producing a new solution $\pi' = (\pi(1), \ldots, \pi(q-1), \pi(r), \pi(q+1), \ldots, \pi(r-1), \pi(q), \pi(r+1), \ldots, \pi(n))$. In this scenario, the set of orderings that can be obtained by performing a swap move, typically denoted as neighborhood, has size $n(n-1)/2$.

The second move operator is known as insert. Specifically, given a solution $\pi$, the move $\text{insert}(\pi, q, r)$ consists of removing facility $\pi(q)$ from its current position $q$ and inserting it at position $r$. The new solution $\pi'$ depends on the relative values of $q$ and $r$. If $q > r$, then $\pi' = (\pi(1), \ldots, \pi(q-1), \pi(q), \pi(r), \pi(q+1), \ldots, \pi(r-1), \pi(q), \pi(r+1), \ldots, \pi(n))$. In contrast, if $q < r$ then $\pi' = (\pi(1), \ldots, \pi(q-1), \pi(q+1), \ldots, \pi(r-1), \pi(q), \pi(r), \pi(q+1), \ldots, \pi(n))$. In this case, the associated neighborhood has size $n(n-1)/2$.

There exist two typical strategies to explore a neighborhood: best improvement and first improvement. The former explores all of the solutions in the neighborhood by a fully deterministic procedure, and the best move (i.e., the one that leads to a solution $\pi'$ with minimum associated cost) is applied at each iteration. Previous approaches for tackling the SRFLP use the best improving strategy to either identify the best swap or insert move. A naive implementation of these approaches would result in an algorithm of computational complexity $\Theta(n^3)$, since there is a quadratic number of neighbors, and the cost of evaluating each one is computed in $\Theta(n^2)$. Nevertheless, through appropriate bookkeeping, it is possible to obtain the neighbors and their costs in $\mathcal{O}(n^2)$ time, as described in Kothari and Ghosh [28]. When using insert moves we denote this type of local search as LS-BEST.

The first improvement strategy explores the neighborhood of an initial solution and performs the first move that enhances the resulting cost. The procedure usually chooses different moves at random in order to obtain diverse solutions, and halts when the entire neighborhood has been explored and no moves are able to improve the cost of the initial solution. In the context of the SRFLP, this approach is inefficient since a purely randomized search cannot use bookkeeping. In particular, note that the algorithm runs in $\mathcal{O}(n^4)$ time in the worst case, which occurs, for example, when it halts arriving at a local minimum.

We now introduce a new hybrid local search method, denoted as LS-HYBRID, which uses insert moves and combines the first and best strategies. Given a particular solution, the idea consists of selecting a facility at random in each step (similarly to the first improvement strategy), and then finding the best insert move only for such facility (as carried out by the best improvement strategy). If the value of the objective function does not decrease the algorithm randomly chooses a different facility and looks for the new best insert move for that facility. This process is initialized and repeated as it finds better solutions. Finally, the algorithm reaches a local minimum and halts when no insertion operations can improve the cost of a previous solution.

Our local search method is based on insert moves since they usually exhibit a better performance in practice than swap moves [31]. In order to speed up the search through bookkeeping, the method analyzes intermediate swap moves of consecutive positions in the search for the best improving move, instead of inserting a facility directly in a target position. Specifically, given a facility $\pi(q)$, at position $q$, we exchange it with the facility at position $q - 1$, recording the associated change in the objective function (denoted as move value). Then, we exchange the facilities at positions $q - 1$ and $q - 2$, storing again the associated move value. The method proceeds in a similar way, until it reaches position 1. Symmetrically, the local search performs swaps between consecutive positions from $q + 1$ to $n$. Fig. 4 illustrates how the hybrid local search proceeds when performing an insert move of facility $\pi(q)$.

This proposed hybrid local search computes the move values in an incremental way since the evaluation of a swap move between consecutive positions can be calculated in linear time. In particular, given a solution $\pi$ and a facility $\pi(q)$, the cost of a solution $\pi'$ after performing the move $\text{swap}(\pi, q, q + 1)$ can be computed as:

$$C(\pi') = C(\pi) + I_{\pi(q+1)} \left( \sum_{s=1}^{q-1} C_{\pi(s)\pi(q)} - \sum_{s=q+2}^{n} C_{\pi(q)\pi(s)} \right) + I_{\pi(q)} \left( \sum_{s=q+2}^{n} C_{\pi(q)\pi(s)} - \sum_{s=1}^{q-1} C_{\pi(s)\pi(q+1)} \right).$$

which runs in $\Theta(n)$ time. The first term of the right hand side of the equation identifies the cost of solution $\pi$. The second one adjusts the cost associated with facility $\pi(q + 1)$, while the last term corrects the cost for $\pi(q)$. The combination of the composition of swap moves to obtain a general insert move, together with the incremental computation of the cost value defined above, makes the local search very efficient, as we will show in our computational results (see Section 6). Note that an insert move requires $\Theta(n^4)$ time, since each of the $n-1$ swap operations to be carried out runs in linear time. Lastly, the algorithm requires $\mathcal{O}(n^3)$ time when reaching a local minimum (i.e., in the worst case).
4. Path relinking

Path relinking (PR) is a metaheuristic originally proposed as a methodology to integrate intensification and diversification strategies in the context of tabu search [15,16]. Its current algorithm template is defined in Glover et al. [17] as a general strategy to combine solutions. The method generates new solutions by iteratively modifying an initial one, transforming it into another in its neighborhood, until reaching a final guiding solution. Therefore, the process builds a trajectory of new intermediate solutions with the hope that these can eventually be better than the high-quality solutions being connected.

The choice of the move operator determines how the path is constructed. In this paper we first describe the approach of Kothari and Ghosh [27], which we denote as PR1. In addition, we introduce two alternative strategies to construct paths between high-quality solutions, denoted as PR2 and PR3. In order to implement these strategies efficiently we use an alternative representation of a solution. Specifically, if $\pi$ is a solution for the SRFLP (where facility $\pi(q)$ is located at position $q$), then it can also be specified through its inverse representation $\pi^{-1}$, which indicates the positions where the facilities are located in $\pi$ (i.e., facility $i$ is located at position $\pi^{-1}(i)$ in $\pi$). For instance, let $\pi_x = (2, 3, 1, 4, 5)$ be some initial ordering and $\pi_y = (5, 2, 1, 4, 3)$ a guiding solution. The inverse representation of $\pi_x$ is $\pi_x^{-1} = (3, 2, 5, 4, 1)$. This alternative representation allows the PR algorithms to efficiently locate a facility and its corresponding position.

The first algorithm based on the path relinking methodology (PR1) is based on swap moves, which are carried out in a deterministic order. PR1 analyzes each element in $\pi_x$ and $\pi_y$ in order, say, from left to right. If at some iteration $q$ (with $1 \leq q \leq n$), the facility $\pi_x(q)$ is different than $\pi_y(q)$, then a swap move is performed on $\pi_x$ so that the $q$-th facility in $\pi_x$ will be the same as that in $\pi_y$. Formally, the move is defined through $\text{swap}((\pi_x, q, \pi_x^{-1}(\pi_y(q))))$. Considering the previous solutions $\pi_x$ and $\pi_y$, the first move would swap facilities 2 and 5 in $\pi_x$, generating the intermediate solution $\pi_1 = (5, 3, 1, 4, 2)$, where $\pi_1(1) = \pi_y(1) = 5$. The second one swaps facilities 2 and 3, which produces a new solution $\pi_2 = (5, 2, 1, 4, 3)$, whose two first facilities coincide with those in the guiding solution $\pi_y$. The subsequent steps do not generate moves since $\pi_2 = \pi_y$.

The second path relinking approach, denoted as PR2, favors the diversification of the constructed paths, and can be considered to be a randomized variant of PR1. This new approach starts by identifying the set $D$ of facilities in both solutions that are located at different positions. For instance, considering the previous solutions $\pi_x$ and $\pi_y$, this set is $D = (5, 2, 3)$. Since facilities 1 and 4 are located in both permutations at positions 3 and 4, respectively, PR2 selects a facility $i \in D$ at random, whose positions in $\pi_x$ and $\pi_y$ are $\pi_x^{-1}(i)$ and $\pi_y^{-1}(i)$, respectively. Subsequently, it performs the move $\text{swap}((\pi_x, \pi_x^{-1}(i), \pi_y^{-1}(i)))$, which

\[
\begin{align*}
\pi(1) & \ldots \pi(q-2) \pi(q-1) \pi(q) \pi(q+1) \ldots \pi(n) \\
\pi(1) & \ldots \pi(q-2) \pi(q) \pi(q+1) \pi(q+2) \ldots \pi(n) \\
\vdots & \\
\pi(q) & \pi(1) \ldots \pi(q-2) \pi(q-1) \pi(q+1) \ldots \pi(n) \\
\end{align*}
\]

\[
\begin{align*}
\pi(1) & \ldots \pi(q-2) \pi(q) \pi(q+1) \pi(q+2) \ldots \pi(n) \\
\pi(1) & \ldots \pi(q-2) \pi(q+1) \pi(q+2) \ldots \pi(n) \\
\vdots & \\
\pi(q) & \pi(1) \ldots \pi(q-2) \pi(q+1) \pi(q+2) \ldots \pi(n) \\
\end{align*}
\]

Fig. 4. Insert moves of a facility (at position $q$) through contiguous swap operations. The swaps are performed towards the left (a) and right (b). The entire process requires $\Theta(n^2)$ time.
guarantees that facility $i$ will be located at the same position in the intermediate and guiding solutions. For instance, suppose that PR2 randomly selects facility $i=3$, whose positions in $\pi_x$ and $\pi_y$ can be efficiently determined by using the inverse representation $(\pi^{-1}_x(3) = 2$ and $\pi^{-1}_y(3) = 5$). In order to situate facility 3 at the same position in both solutions, PR2 performs the move $\text{swap}(\pi_x, \pi_x^{-1}(3), \pi_y^{-1}(3)) = \text{swap}(\pi_x, 2, 5)$, producing the intermediate solution $\pi_1 = (2, 5, 1, 4, 3)$. The next step would select either facility 2 or 5, generating a swap move between the first two facilities in $\pi_1$ that would produce the guiding solution.

We propose a third, more sophisticated, path relinking strategy, denoted as PR3. It is related to the calculation of the Ulam distance [53], which measures the minimum number of insert moves necessary to transform one ordering into another. It can also be understood as $n$ minus the length of the longest common subsequence between two orderings, which can be calculated in $O(n^2)$ time by a straightforward dynamic programming algorithm [50]. However, since the solutions for the SRFLP are orderings of the first $n$ integers, it is possible to calculate the Ulam distance by solving the longest increasing subsequence problem instead, which can be computed in $O(n \log n)$, as we describe below [24].

The first step of PR3 consists of identifying the largest set of elements in both permutations that preserve the same relative order between them. For example, let $\pi_x = (3, 7, 8, 4, 1, 2, 5, 6)$ and $\pi_y = (3, 2, 1, 8, 4, 5, 6, 7)$ be two solutions for some instance of the SRFLP. Only facility 3 is located at the same (first) position in both orderings. However, we can observe that facilities 3, 8, 4, 5, and 6 share the same relative ordering in both solutions. In other words, $\pi_x^{-1}(3) < \pi_x^{-1}(8) < \pi_x^{-1}(4) < \pi_x^{-1}(5) < \pi_x^{-1}(6)$ and $\pi_y^{-1}(3) < \pi_y^{-1}(8) < \pi_y^{-1}(4) < \pi_y^{-1}(5) < \pi_y^{-1}(6)$ (see Fig. 5). Let $\lambda$ represent this largest set of facilities, which is not unique in general. For notational convenience, we will represent it as a list $\lambda = (3, 8, 4, 5, 6)$ in the example.

Note that if $\pi_x$ were the ordering $(1, 2, \ldots, n)$ then $\lambda$ would correspond to the longest increasing subsequence in $\pi_x$. However, since $\pi_x$ can be any ordering, it is necessary to modify the labels of the facilities in order to obtain $\lambda$ by computing a longest increasing subsequence. Specifically, facility $\pi_x(1)$ would have to be renamed as 1, $\pi_x(2)$ as 2, and so on, which is achieved by applying $\pi_x^{-1}$ to the initial orderings, as shown in Fig. 5. Therefore, $\lambda$ can be found by first computing the longest increasing subsequence in following ordering:

$$\tau = \pi_x^{-1}(\pi_y) = (\pi_x^{-1}(\pi_y(1)), \pi_x^{-1}(\pi_y(2)), \ldots, \pi_x^{-1}(\pi_y(n))).$$

Note that $\tau$ encodes the location in $\pi_x$ of the facilities in $\pi_y$ (i.e., $\tau(q) = \pi_x^{-1}(\pi_y(q))$ is the position in $\pi_x$ of the $q$-th facility in $\pi_y$). In other words, it indicates the relative ordering in $\pi_x$ of the facilities in $\pi_y$. In the example $\tau = (1, 6, 5, 3, 4, 7, 8, 2)$, since the first facility in $\pi_y$ is located at the first position in $\pi_x$, the second in $\pi_y$ is located at the sixth position in $\pi_x$, and so on.

Subsequently, let $\sigma$ represent the longest increasing subsequence in $\tau$, which is $(1, 3, 4, 7, 8)$ in the example (note that the elements in $\sigma$ are not necessarily contiguous). Finally, $\lambda$ can be recovered through $\sigma$ in two different ways, as illustrated in Fig. 6. On the one hand, $\sigma$ provides the locations of the elements of $\lambda$ in $\pi_x$. On the other hand, the elements of $\lambda$ are located in $\pi_y$ at the same positions as the elements of $\sigma$ are located in $\tau$.

Finally, the algorithm selects the insert moves to be carried out at random. In particular, let $D$ be the set of facilities that are not in $\lambda$ (in the example, $D = [1, 2, 7]$). Firstly, PR3 selects a facility from $D$ at random and inserts it in an adequate position in $\pi_x$ in order to increase the number of facilities that share the same relative ordering in both permutations. For example, suppose that PR3 chooses facility 1. Since it must be inserted between facilities 3 and 8, it can be inserted at either position 2 or 3. Our proposed algorithm selects one of these possible moves at random. Lastly, this process is repeated until all of the elements in $D$ have been properly inserted.

5. GRASP with path relinking

Path relinking can be adapted in the context of GRASP as a form of post-optimization strategy [35]. The combination of both approaches operates on a set of solutions of size $b$ that is usually called the elite set (ES = $E_1, E_2, \ldots, E_b$). It is typically sorted according to the quality of the solutions (from best, $E_1$, to worst, $E_b$). The approach first initializes ES with solutions constructed by a GRASP procedure. Afterwards, the algorithm continues generating more GRASP solutions that are combined with those in the elite set through a path relinking approach. The resulting solutions replace existing ones in the elite set, improving its overall quality.

Algorithm 3 shows the pseudocode of our proposed GRASP with path relinking procedure. The elite set is initially populated
Algorithm 3: GRASP with path relinking algorithm.

1: $I \leftarrow \emptyset$
2: for $i = 1$ to $m$ do
3: $\pi_i \leftarrow \text{GRASP}()$
4: $I \leftarrow I \cup \{\pi_i\}$
5: end for
6: $ES \leftarrow \text{SelectBest}(I, b)$
7: while $\text{execution \_time} < T_{\text{max}}$ do
8: $\pi_{\text{GRASP}} \leftarrow \text{GRASP}()$
9: for $i = 1$ to $b$ do
10: $\pi_{\text{PR}} \leftarrow \text{PathRelinking}(\pi_{\text{GRASP}}, ES_i)$
11: if $(\pi_{\text{PR}} \notin ES)$ and $(C(\pi_{\text{PR}}) < C(ES_i))$ then
12: $j \leftarrow \text{arg min}[d(\pi_{\text{PR}}, ES_i)]$
13: if $(j \neq 1)$ or $(C(\pi_{\text{PR}}) < C(ES_j))$ then
14: $ES_j \leftarrow \pi_{\text{PR}}$
15: $ES \leftarrow \text{sort}(ES)$
16: end if
17: end if
18: end for
19: end while
20: return $ES_1$

by retaining the best $b$ solutions generated by a GRASP procedure (see Section 3) that is executed $m \geq b$ times (steps 2 to 6). Since we focus mainly on the quality of the solutions stored in $ES$ we choose a large value for $m$ that depends on the size of the problem.

Afterwards the algorithm iterates generating a new GRASP solution $\pi_{\text{GRASP}}$ (step 8), which will be combined with solutions in the elite set through a path relinking procedure, until reaching a fixed amount of execution time $T_{\text{max}}$ (steps 7 to 19). In particular, instead of generating a path between $\pi_{\text{GRASP}}$ and a solution probabilistically selected from $ES$ (as is customary), we compute all path trajectories between $\pi_{\text{GRASP}}$ and each of the solutions in $ES$ (steps 9 to 18). Let $\pi_{\text{PR}}$ represent the best solution found in each trajectory (step 10), which has been additionally optimized through one of the local search procedures described in Section 3.2. The algorithm proceeds by evaluating $\pi_{\text{PR}}$ in order to determine whether it should be included in $ES$. Firstly, it cannot be already present in $ES$, and its associated cost must be at least as large as the cost of the worst solution in $ES$ (step 11). If these conditions are met the algorithm proceeds by calculating the nearest solution in $ES$ (denoted as $ES_j$) to $\pi_{\text{PR}}$ (step 12). In particular, we compute the dissimilarity between solutions through the distance between permutations previously used in Kothari and Ghosh [32]. Formally, this distance is defined as:

$$d(\pi_x, \pi_y) = \min\{\delta(\pi_x, \pi_y), \delta(\bar{\pi}_x, \bar{\pi}_y)\},$$

where $\bar{\pi}_y$ is the reverse ordering of $\pi_y$ (i.e., $\bar{\pi}_y = \langle \pi_y(n), \pi_y(n-1)\ldots, \pi_y(1) \rangle$); and

$$\delta(\pi_x, \pi_y) = \sum_{i=1}^{n} |\pi_x^{-1}(i) - \pi_y^{-1}(i)|$$

is the “deviation distance” [50]. Finally, $\pi_{\text{PR}}$ is introduced in the elite set, while $ES_j$ is removed (step 14), and $ES$ is resorted (notice that this operation can be performed in linear time (step 15)). However, this occurs only if $\pi_{\text{PR}}$ is the best solution found so far, or as long as the solution to be deleted is not the best one in the elite set (step 13). Thus, if $\pi_{\text{PR}}$ is better than $ES_1$ it is always admitted into $ES$. In addition, if it is worse than $ES_1$, then $ES_1$ will not be removed from the elite set.

Finally, Fig. 7 contains a summarized flow diagram of the approach. The shaded circles represent solutions to the SRFLP, where their associated costs are represented by the size of the circles (smaller circles depict better solutions). Observe that path relinking is applied between $\pi_{\text{GRASP}}$ and solutions in $ES$. The best solution along the trajectory $(\pi_{\text{PR}})$ is chosen, improved, and replaces a solution in $ES$ depending on the conditions in steps 11 and 13 of Algorithm 3.

6. Computational results

In this section we report on the computational experiments performed in order to test the efficiency and effectiveness of the proposed strategies. We performed all of the experiments on a personal computer with a fourth generation Intel® Core™ i7-4712HQ 3.3 GHz processor and 16 GB of RAM. All of the code was written in C and compiled with the gcc compiler. Lastly, we built our own implementations of the methods in Kothari and Ghosh [31] and Kothari and Ghosh [32] since the code/executable was not available.

We ran experiments on the three sets of benchmark instances used in the recent literature on the SRFLP, which are publicly available at [6]. The first one was introduced in Anjos et al. [9] and consists of four groups (for $n = 60, 70, 75$, and 80) of five instances. These groups are usually known as Anjos instances. Another popular set, originally introduced in Anjos and Yen [8], is based on the Quadratic Assignment problem and named as sko. In this paper we use the four groups (for $n = 64, 72, 81$, and 100) of five instances. Finally, a third set, referred to as Amaran1, is due to Amaran and Letchford [5] (see the associated working paper) and consists of three instances, each of size $n = 110$.

Since the latest studies essentially report the same results on these instances, we have generated new larger and more challenging instances in order to compare algorithms. In particular, we have created two new sets, Anjos-large (40 instances) and sko-large (40 instances), where lengths and costs were drawn from distributions that resembled as closely as possible those used in the Anjos and sko sets of instances. Each set contains 40 instances with considerably larger size. Specifically, we have considered values five instances for values of $n$ in $\{150, 200, 250, 300, 350, 400, 450, 500\}$. These sets are available in [46].

In this section we first describe a preliminary experimentation with a subset of training instances that allows to compare the different alternatives introduced in the previous sections, and to identify a particular one as our final algorithm. This training set consists of the 40 instances of sizes 150, 250, 350, and 450, included in the Anjos-large and sko-large sets. Lastly, we report further computational experiments carried out with a different test set of instances in order to compare our final algorithm with state-of-the-art methods. In particular, we compare results with respect to: (i) the genetic algorithm GENALGO described in Kothari and Ghosh [31], and (ii) the scatter search variant SS-2P reported in Kothari and Ghosh [32], which we found to perform slightly better than the other three variants described in such paper. Finally, the test set contains the remaining 40 instances, of sizes 200, 300, 400, and 500, included in the Anjos-large and sko-large sets.

6.1. Preliminary experiments

We carried out an exhaustive combination of the algorithmic elements described in previous sections in order to examine their performance on the training instances. Particularly, we executed a single full-factorial experiment where we considered the greedy functions $g_1$, $g_2$ and $g_3$, the GR and RG algorithms, the two local search methods LS-BEST and LS-HYBRID, and the path relinking strategies PR1, PR2 and PR3. We do not include variations of the $\alpha$ parameter since we did not observe relevant differences. In
particular, we tested $\alpha = [0.25, 0.5, 0.75]$, but none of them showed a significant superiority over the rest. Therefore, the reported results only consider a standard value of $\alpha = 0.5$ as a trade-off between intensification and diversification. Additionally, the main parameters of the GRASP with PR method depend on the size of the problem ($n$). Specifically, the number of initial GRASP iterations, $m$, is set to $m = \lceil n/2 \rceil$, while the size of the elite set, $b$, is fixed to $b = \lceil n/20 \rceil$. In addition, since our final experiments were planned considering time limits of $n$ and 3600 s, we set the maximum execution time to $2n$ s as a compromise, and to avoid any possible overfitting of the parameters. Lastly, we have experimentally tested that small variations of these parameters barely affect the performance of the compared algorithms.

Fig. 8 shows a comparison of 36 algorithmic variants. The X-axis indicates a particular greedy function and construction algorithm, while the Y-axis measures the average ranking of each variant. The dotted line (in the middle of the graph) separates the algorithms that use LS-BEST (left side) from those that use LS-HYBRID (right side). Finally, for each algorithm indicated in the X-axis we consider three different columns that correspond to one of the three path relinking strategies.

The average ranking is computed by considering the results on the 40 instances of the training set. Specifically, if an algorithm is the best (i.e., provides the lowest cost) on some instance, its ranking is 1 for that particular instance. Instead, if it were the worst, its ranking would be 36 (since that is the number of algorithms being compared). In addition, if two algorithms provide the same cost on an instance they are assigned the same ranking. Finally, we executed each variant once (for $2n$ s) on each instance.

The results clearly show that the proposed hybrid local search strategy outperforms the best insertion approach used in the state-of-the-art algorithms. In particular, all variants based on LS-HYBRID clearly outperform those based on LS-BEST. Focusing only
on variants that use LS-HYBRID, we can conclude that the path relinking strategy based on the Ulam distance (PR3) is superior to PR1 (which is described in the state of the art) and PR2 (typically used in path relinking). Notice that this does not occur for the variants based on LS-BEST. This can be partially explained by the longer computing times required by LS-BEST, which can cause less path relinking iterations. Finally, among the greedy functions and construction algorithms, the combination of $g_3$ and RG provides the best results. Therefore, we define our final algorithm for solving the SRLF, denoted as GRASP-PR, as the combination of LS-HYBRID, PR3, $g_3$, and RG (with $\alpha = 0.5$).
6.2. Comparison with state-of-the-art algorithms

The final comparison with current state-of-the-art methods is divided into three different experiments. In the first one we compare our best variant, GRASP-PR, with the GENALGO and SS-2P algorithms over the sets Anjos, sko, and Amaral. We show in Table 1 the associated results, where the first column indicates the name of each instance (the first number indicates its size) and the remaining columns report, for each algorithm, the cost of the best solution found and the associated computing time in seconds. It is worth mentioning our GRASP-PR algorithm, when running for \( n/2 \) s, found a better solution on one of the instances (sko_81_1), while it was able to match the best results published in the literature on the remaining 42 instances (we obtained the same results after running GRASP-PR for 1 h). In contrast, GENALGO and SS-2P did not achieve the best cost in one and four instances (marked in square brackets and with an asterisk), respectively. Finally, the running times of GENALGO and SS-2P were directly obtained from the original papers.

It is difficult to make a fair comparison when considering CPU time, since GENALGO and SS-2P were executed on a different computer (Intel i-5 2500 3.30 GHz processor with 4 GB of RAM). Nevertheless, to our best estimation, it appears that GRASP-PR is faster than SS-2P on large instances, and much faster than GENALGO when it is run 200 times, as it is done in Kothari and Ghosh [31]. In order to support this claim, we have implemented these two methods, denoted as GENALGO_R and SS-2P_R, by following the indications described in the corresponding original papers, and in the same programming language (C). Table 2 shows the cost and computing time for both methods after executing them on the Anjos, sko and Amaral sets. Our implementations reach a similar performance in considerably smaller computing times than those reported in the literature. On the one hand, GENALGO_R finds the best known results in all instances (note that the original version failed in one of them), while the computing time is reduced in about 70% with respect to the original reported results. On the other hand, SS-2P_R found the best solutions in all but three instances (i.e., one more than SS-2P), while reducing the computing time in almost 80%. Although the increase in speed can be due in part to the use of a faster computer, it is sufficiently large to claim that our implementations are valid.

We now compare GRASP-PR with GENALGO_R and SS-2P_R when using the (40) test instances of sizes 200, 300, 400, and 500, in the Anjos-large and sko-large sets. In order to evaluate the performance of these three algorithms over short and long time horizons, we executed them for \( n \) and 3600 s. Tables 3 and 4 show the associated costs, where the best found values for each of the instances are highlighted in boldface.

Our proposed algorithm was able to find the best solutions on all of the sko-large instances in both time scenarios. Additionally, it also achieved the best results in 19 (out of 20) instances in the set Anjos-large when considering short and long time horizons. On the other hand, GENALGO_R achieved the best result in only one instance (in both time scenarios), while SS-2R-R found the best value (tying with GRASP-PR) in one instance when considering 3600 s. Thus, it is apparent that GRASP-PR exhibits a better performance.

In order to support the previous claim, we applied a Friedman test to the data in Tables 3 and 4. This nonparametric statistical test is similar to the repeated measures ANOVA test, and is usually used to detect differences in algorithm performance across the same set of instances. The test ranks each method on each instance (row), and finally considers the values of the ranks by algorithms (columns). The results are shown in Table 5, where the middle columns report the average ranking across all instances of each approach. The average ranking for our GRASP-PR method is very close to 1, which reflects that it generally obtains higher-quality solutions. The low \( p \)-values clearly indicate that there is enough statistical evidence to confirm that there are differences between the three algorithms.

Having confirmed the existence of differences between the methods, we conducted nonparametric statistical sign tests in order to detect consistent differences between GRASP-PR and the two previous methods. Tables 6 and 7 summarize the results of the tests, where sign is the value of the test sign statistic, which indicates the number of times GRASP-PR outperforms a competitor algorithm. The final column reports the \( p \)-values (for one-sided tests) associated with each experiment. Given the low \( p \)-values, the statistical test clearly supports the superiority of GRASP-PR over the genetic algorithm described in Kothari and Ghosh [31], and the scatter search procedure proposed in Kothari and Ghosh [32].

In the previous experiments the algorithms were run for a fixed time limit. In order to compare convergence times we executed SS-2R_R and GENALGO_R until they reached the value attained by our GRASP-PR after running it for 1 h on the new instances (Table 4 indicates the associated costs), or for a maximum of 10 h. Table 8 shows the halting times of the state-of-the-art algorithms, where
cells containing " > 36000" indicate that the methods could not match or enhance the solutions of GRASP-PR after running for 10 h. Note that this situation occurred in 59 out of the 80 cases.

Lastly, Fig. 9 shows a TTT (time-to-target) plot [1] resulting from 40 executions with different random seeds of SS-2R_R, GENALGO_R, and GRASP-PR over instance Anjos_200_1. This graphic shows the probability of reaching a fixed cost value (in particular, 305788558 for the instance) in a certain time, and illustrates the general behavior of the three algorithms. It is apparent that our approach converges to higher quality solutions faster than the state-of-the-art methods.

### Table 3
Comparison of SRFLP algorithms on Anjos-large and sko-large instances, when executing them for n seconds.

<table>
<thead>
<tr>
<th>Instance</th>
<th>GENALGO_R</th>
<th>SS-2P_R</th>
<th>GRASP-PR</th>
</tr>
</thead>
<tbody>
<tr>
<td>Anjos_200_1</td>
<td>305,497,735</td>
<td>305,461,895</td>
<td>305,461,862</td>
</tr>
<tr>
<td>Anjos_200_2</td>
<td>178,807,686.5</td>
<td>178,852,729.5</td>
<td>178,816,295.5</td>
</tr>
<tr>
<td>Anjos_200_3</td>
<td>61,893,221</td>
<td>61,891,688</td>
<td>61,891,275</td>
</tr>
<tr>
<td>Anjos_200_4</td>
<td>127,745,269</td>
<td>127,736,617</td>
<td>127,736,350</td>
</tr>
<tr>
<td>Anjos_300_1</td>
<td>89,059,441.5</td>
<td>89,141,170.5</td>
<td>89,057,182.5</td>
</tr>
<tr>
<td>Anjos_300_2</td>
<td>1550,458,661</td>
<td>1552,008,518</td>
<td>1549,663,689</td>
</tr>
<tr>
<td>Anjos_300_3</td>
<td>956,868,890.5</td>
<td>957,367,664.5</td>
<td>955,572,080.5</td>
</tr>
<tr>
<td>Anjos_300_4</td>
<td>308,456,080.5</td>
<td>308,866,436.5</td>
<td>308,257,766.5</td>
</tr>
<tr>
<td>Anjos_400_1</td>
<td>603,280,369.5</td>
<td>603,679,994.5</td>
<td>602,873,363.5</td>
</tr>
<tr>
<td>Anjos_400_2</td>
<td>467,360,444</td>
<td>468,172,304</td>
<td>466,160,315</td>
</tr>
<tr>
<td>Anjos_400_3</td>
<td>500070,139,5</td>
<td>500032,341.5</td>
<td>500057,526,665</td>
</tr>
<tr>
<td>Anjos_400_4</td>
<td>2916,949,529</td>
<td>2917,158,558</td>
<td>2910,279,558</td>
</tr>
<tr>
<td>Anjos_500_1</td>
<td>922,299,658</td>
<td>922,534,333</td>
<td>921,216,592</td>
</tr>
<tr>
<td>Anjos_500_2</td>
<td>1809,796,198</td>
<td>1809,847,357</td>
<td>1806,067,255</td>
</tr>
<tr>
<td>Anjos_500_3</td>
<td>1404,633,352.5</td>
<td>1405,722,365.5</td>
<td>1402,779,504.5</td>
</tr>
<tr>
<td>Anjos_500_4</td>
<td>1238,818,681</td>
<td>12326,499,609</td>
<td>12300,427,281</td>
</tr>
<tr>
<td>Anjos_500_5</td>
<td>7594,421,872.5</td>
<td>7505,998,255.5</td>
<td>7493,143,632.5</td>
</tr>
<tr>
<td>Anjos_600_1</td>
<td>2485,064,290</td>
<td>2483,586,441</td>
<td>2479,334,789</td>
</tr>
<tr>
<td>Anjos_600_2</td>
<td>4294,960,811.5</td>
<td>4294,352,718.5</td>
<td>4285,972,468.5</td>
</tr>
<tr>
<td>Anjos_600_3</td>
<td>3689,059,932.5</td>
<td>3680,497,633.5</td>
<td>3678,038,149.5</td>
</tr>
</tbody>
</table>

### Table 4
Comparison of SRFLP algorithms on Anjos-large and sko-large instances, when executing them for 1 h.

<table>
<thead>
<tr>
<th>Instance</th>
<th>GENALGO_R</th>
<th>SS-2P_R</th>
<th>GRASP-PR</th>
</tr>
</thead>
<tbody>
<tr>
<td>Anjos_200_1</td>
<td>305,497,727</td>
<td>305,461,818</td>
<td>305,461,818</td>
</tr>
<tr>
<td>Anjos_200_2</td>
<td>178,807,197.5</td>
<td>178,852,677.5</td>
<td>178,816,261.5</td>
</tr>
<tr>
<td>Anjos_200_3</td>
<td>61,892,040</td>
<td>61,891,652</td>
<td>61,891,275</td>
</tr>
<tr>
<td>Anjos_200_4</td>
<td>127,745,257</td>
<td>127,736,464</td>
<td>127,735,691</td>
</tr>
<tr>
<td>Anjos_300_1</td>
<td>89,059,083.5</td>
<td>89,141,158.5</td>
<td>89,057,121.5</td>
</tr>
<tr>
<td>Anjos_300_2</td>
<td>1550,443,050</td>
<td>1550,822,258</td>
<td>1549,663,657</td>
</tr>
<tr>
<td>Anjos_300_3</td>
<td>955,749,527.5</td>
<td>957,249,994.5</td>
<td>955,572,066.5</td>
</tr>
<tr>
<td>Anjos_300_4</td>
<td>308,372,773.5</td>
<td>308,701,657.5</td>
<td>308,257,360.5</td>
</tr>
<tr>
<td>Anjos_400_1</td>
<td>603,268,519.5</td>
<td>603,078,289.5</td>
<td>602,873,168.5</td>
</tr>
<tr>
<td>Anjos_400_2</td>
<td>467,717,899</td>
<td>466,162,354</td>
<td>466,160,264</td>
</tr>
<tr>
<td>Anjos_400_3</td>
<td>5004,786,474.5</td>
<td>5005,963,135.5</td>
<td>5000,752,142.5</td>
</tr>
<tr>
<td>Anjos_400_4</td>
<td>2916,949,529</td>
<td>2916,726,055</td>
<td>2910,276,759</td>
</tr>
<tr>
<td>Anjos_500_1</td>
<td>9213,146,203</td>
<td>922,110,067</td>
<td>921,216,455</td>
</tr>
<tr>
<td>Anjos_500_2</td>
<td>1809,520,966</td>
<td>1809,564,367</td>
<td>1806,061,379</td>
</tr>
<tr>
<td>Anjos_500_3</td>
<td>1404,580,939.5</td>
<td>1405,152,456.5</td>
<td>1402,779,472.5</td>
</tr>
<tr>
<td>Anjos_500_4</td>
<td>12318,818,681</td>
<td>12300,427,281</td>
<td>12290,409,839</td>
</tr>
<tr>
<td>Anjos_500_5</td>
<td>7505,406,907.5</td>
<td>7501,448,013.5</td>
<td>7493,120,635.5</td>
</tr>
<tr>
<td>Anjos_600_1</td>
<td>2483,990,108</td>
<td>2483,197,897</td>
<td>2479,333,773</td>
</tr>
<tr>
<td>Anjos_600_2</td>
<td>4288,172,936.5</td>
<td>4293,459,100.5</td>
<td>4285,937,468.5</td>
</tr>
<tr>
<td>Anjos_600_3</td>
<td>3680,445,362.5</td>
<td>3680,144,207.5</td>
<td>3678,038,066.5</td>
</tr>
</tbody>
</table>

### Table 5
Results of Friedman's test applied to the data in Table 3 and Table 4.

<table>
<thead>
<tr>
<th>Execution time</th>
<th>Average ranking</th>
<th>p-value</th>
</tr>
</thead>
<tbody>
<tr>
<td>n seconds</td>
<td>GENALGO_R</td>
<td>SS-2P_R</td>
</tr>
<tr>
<td>1 h</td>
<td>2.45</td>
<td>2.53</td>
</tr>
<tr>
<td>2 h</td>
<td>2.50</td>
<td>2.46</td>
</tr>
</tbody>
</table>

### Table 6
Results of the statistical sign test applied to the data in Table 3 regarding experiments over n seconds.

<table>
<thead>
<tr>
<th>Compared algorithms</th>
<th>Sign</th>
<th>p-value</th>
</tr>
</thead>
<tbody>
<tr>
<td>GRASP-PR vs. GENALGO_R</td>
<td>39</td>
<td>3.7 × 10^{-11}</td>
</tr>
<tr>
<td>GRASP-PR vs. SS-2P_R</td>
<td>40</td>
<td>9.1 × 10^{-11}</td>
</tr>
</tbody>
</table>

### Table 7
Results of the statistical sign test applied to the data in Table 4 regarding experiments over 1 h.

<table>
<thead>
<tr>
<th>Compared algorithms</th>
<th>Sign</th>
<th>p-value</th>
</tr>
</thead>
<tbody>
<tr>
<td>GRASP-PR vs. GENALGO_R</td>
<td>39</td>
<td>3.7 × 10^{-11}</td>
</tr>
<tr>
<td>GRASP-PR vs. SS-2P_R</td>
<td>40</td>
<td>1.8 × 10^{-12}</td>
</tr>
</tbody>
</table>
7. Conclusions

This paper has described an algorithm for finding high-quality solutions to the SRLP based on coupling a GRASP procedure with a path relinking approach. Since state-of-the-art algorithms essentially achieve the same results on the instances used throughout the literature, we generated new instances of larger size in order to compare algorithms. In general, our proposed method was able to find better solutions on these large-sized instances, in comparison with previous methods. The key features of the algorithm are several GRASP construction procedures, a new fast local search strategy, and an approach related to the Ulam distance in order to construct path relinking trajectories. In addition, our algorithm not only obtained the best solutions reported previously in the literature efficiently, but it was also able to find a better solution for one instance.

Before engaging in competitive testing, we performed a single full-factorial experimentation to determine the contribution of the various elements that we have designed. We believe that the reader can find them very useful since valuable lessons can be learned from them, and applied to other problems. The extensive final comparison between the proposed GRASP with path relinking approach and the two best identified methods in the literature (a genetic algorithm [27], and a scatter search method [26]), reveals that our algorithm is able to outperform the current state-of-the-art methods in both short and long time horizons. In particular, our approach finds the best known results in previously used instances in considerably shorter computing time. Additionally, it produces higher-quality solutions in 39 out of 40 instances when running the algorithms for n seconds, and 38 out of 40 instances when executing them for 1 h. The superiority of our method is further supported by the low p-values (below 10−11) associated with non-parametric tests for detecting statistical significant differences between the algorithms.

Future research on the problem can examine other complex metaheuristics (e.g., bioinspired approaches such as Ant Colony Optimization or Artificial Bee Colony), variants of the proposed approach, or test algorithms on different instances. For example, the weight matrices of the instances of this problem are rather sparse (i.e., they contain a large number of zeros). Therefore, the performance of these algorithms on instances with dense weight matrices has not been studied yet.

Finally, the best orderings and costs found for the instances in the Anjos, sko and Amaral sets, as well as in the Anjos-large and sko-large test sets for (n = 200, 300, 400, and 500) are available in [46].

Acknowledgments

This research has been partially supported by the Spanish Ministry of “Economía y Competitividad” with grants ref. TIN2015-65460-C2-2-P, TIN2015-66731-C2-1-R and “Comunidad de Madrid” with grants ref. S2013/ICE-2894.

References
