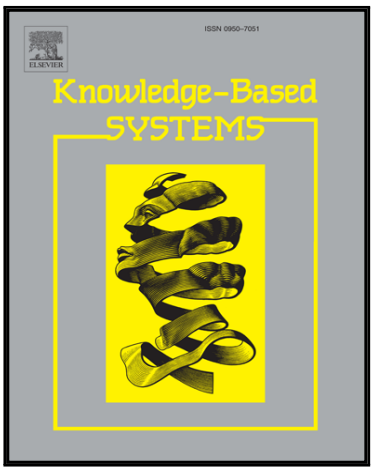


GRASP with path relinking for the single row facility layout problem

Manuel Rubio-Sánchez, Micael Gallego, Francisco Gortazar, Abraham Duarte

PII: S0950-7051(16)30120-4
DOI: 10.1016/j.knosys.2016.05.030
Reference: KNOSYS 3527

To appear in: *Knowledge-Based Systems*

Received date: 9 October 2015
Revised date: 12 May 2016
Accepted date: 13 May 2016

Please cite this article as: Manuel Rubio-Sánchez, Micael Gallego, Francisco Gortazar, Abraham Duarte, GRASP with path relinking for the single row facility layout problem, *Knowledge-Based Systems* (2016), doi: 10.1016/j.knosys.2016.05.030

GRASP WITH PATH RELINKING FOR THE SINGLE ROW FACILITY LAYOUT PROBLEM

MANUEL RUBIO-SÁNCHEZ, MICAEL GALLEGO, FRANCISCO GORTAZAR, AND ABRAHAM DUARTE*

Abstract. The single row facility layout problem (SRFLP) is an $\mathcal{NP}$-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.

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 (Simmons, 1969). It has been 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 (Simmons, 1969; Picard and Queyranne, 1981; Heragu and Kusiak, 1988).

The SRFLP is an $\mathcal{NP}$-hard problem that consists of finding an optimal arrangement of a set of rectangular facilities, placing them next to each other along a line.

Key words and phrases. Metaheuristics, Facilities planning and design, GRASP, Path relinking.

*Corresponding author: Manuel Rubio-Sánchez (manuel.rubio@urjc.es).

Facility lengths:

i	1	2	3	4	5
l_i	5	3	6	4	2

Weights between facilities:

c_{ij}	1	2	3	4	5
1	0	5	2	4	1
2	5	0	3	0	2
3	2	3	0	0	0
4	4	0	0	0	5
5	1	2	0	5	0

FIGURE 1. Instance of size $n = 5$ of the SRFLP, defined through a list l of facility lengths, and a symmetric square weight matrix c.

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 $\pi = \langle \pi(1), \pi(2), \ldots, \pi(n) \rangle$ of the facilities in F, whose cost $C(\pi)$ is defined according to the following objective function:

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

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 π at positions q and r, respectively), and is computed as:

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

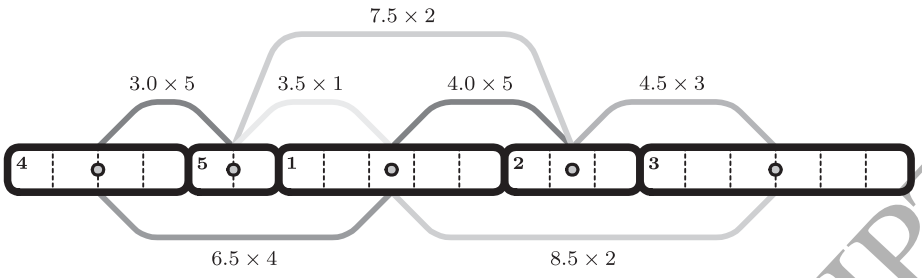
The optimization problem therefore consists of finding an ordering $\pi^\star$ that minimizes (1). Formally:

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

where Π_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 $n \times n$ cost matrix c that contains the weights between the facilities. Figure 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 along an ordering of the particular facilities.

Figure 2 illustrates a solution for the instance described in Figure 1. It is represented as the ordering $\pi = \langle 4, 5, 1, 2, 3 \rangle$, 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 π 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



Solution: $\pi = \langle 4, 5, 1, 2, 3 \rangle$
Cost: $C(\pi) = (3.0 \times 5) + (6.5 \times 4) + (3.5 \times 1) + (7.5 \times 2) + (4.0 \times 5) + (8.5 \times 2) + (4.5 \times 3) = 110$

Solution: $\pi = \langle 4, 5, 1, 2, 3 \rangle$
Cost: $C(\pi) = (3.0 \times 5) + (6.5 \times 4) + (3.5 \times 1) + (7.5 \times 2) + (4.0 \times 5) + (8.5 \times 2) + (4.5 \times 3) = 110$

FIGURE 2. Example of solution $\pi = \langle 4, 5, 1, 2, 3 \rangle$ for the instance defined in Figure 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).

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 (Feo and Resende, 1989) in order to efficiently search for high-quality solutions for the SRFLP. 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 (Glover, 1998), where we construct short path trajectories through an approach related to the Ulam distance between permutations (Ulam, 1972). 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 SRLFP currently emerges as one of the most active problems. Surveys regarding the state of the art on the SRFLP are available in Kothari and Ghosh (2012a); Keller and Buscher (2015). Researchers have developed exact algorithms for solving the problem that include branch-and-bound strategies (Simmons, 1969), dynamic programming (Picard and Queyranne, 1981), linear mixed integer programming (Love and Wong, 1976; Heragu and Kusiak, 1991; Amaral, 2006; 2008), branch and cut (Amaral and Letchford, 2013), cutting planes (Amaral, 2009; Sanjeevi and Kianfar, 2010), semidefinite programming (Anjos et al., 2005; Anjos and Yen, 2009; Hungerländer and Rendl, 2013), or a combination of the last two approaches (Anjos and Vannelli, 2008). 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 (Hungerländer, 2014).

From a heuristic perspective, construction procedures are the simplest ones. These algorithms consider either the weights between the facilities (Heragu and Kusiak, 1988; Kumar et al., 1995), or the length of the facilities (Samarghandi and Eshghi, 2010), 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 (2010) has been used within a Tabu Search framework, for a multi-start strategy that improves each constructed solution with a Lin-Kernigham insertion neighborhood (Kothari and Ghosh, 2013a), or as an initial seed to the diversification stage of the scatter search approach in Kothari and Ghosh (2014b).

A variety of other metaheuristics have also been used to tackle the SRFLP. These include genetic algorithms (Datta et al., 2011; Kothari and Ghosh, 2014a), particle swarm (Samarghandi et al., 2010) and ant colony optimization (Solimanpur et al., 2005), simulated annealing (Romero and Sánchez-Flores, 1990; Heragu and Alfa, 1992), or scatter search (Kumar et al., 2008; Kothari and Ghosh, 2014b). In addition, some approaches combine different metaheuristic strategies. For instance, simulated annealing is coupled with genetic algorithms in Ramkumar and Ponnambalam (2004), while in Kothari and Ghosh (2012b) path relinking is applied to solutions generated by the approaches in Kothari and Ghosh (2013a;b; 2014b).

To the best of our knowledge, the best approaches for finding approximate solutions for the SRFLP through metaheuristics are the ones introduced in Kothari and Ghosh (2014a) and Kothari and Ghosh (2014b). Specifically, the first work presents a straightforward genetic algorithm that uses the partially matched crossover (PMX) operator (see Larrañaga et al. (1999) 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

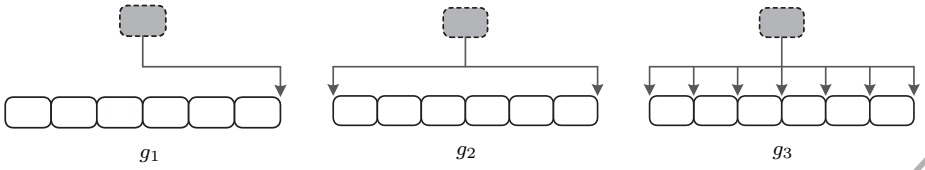


FIGURE 3. Greedy functions. A new facility can be incorporated into a partial solution by appending it to one extreme of the sequence (g_1), to the best of both extremes (g_2), or by inserting it at the best possible location (g_3).

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 (Feo and Resende, 1989) and formally introduced in Feo et al. (1994). Recent and thorough surveys of the method are presented in Resende and Ribeiro (2010; 2014). 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 π of size n for the SRFLP iteratively by inserting facilities one at a time (through random and greedy strategies) in a partial solution π_p containing less than n facilities. For this purpose we propose the three greedy functions illustrated in Figure 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 π_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:

$$C(\pi_p) = \sum_{1 \leq q < r \leq m} c_{\pi_p(q)\pi_p(r)} \cdot d_{\pi_p(q)\pi_p(r)}. \tag{2}$$

Algorithm 1: Greedy-Random (GR)

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

Algorithm 2: Random-Greedy (RG)

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

On the one hand, this partial cost allows to determine the best location along π_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 π_p according to these optimal costs, which allows to apply greedy and randomized strategies for selecting the particular facility that will be incorporated into π_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 π_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). Afterwards, the algorithm selects one facility from *RCL* at random (step 8), includes it in the partial solution (step 9), and updates the candidate list (step 10). The method finally returns the completed solution (note that π_p will contain exactly n facilities).

The second family of constructive procedures, denoted as Random-Greedy (RG), is based on a different strategy introduced in Resende and Werneck (2004), which has been applied with success recently in Campos et al. (2013); Resende et al. (2010); Pantrigo et al. (2012); Duarte et al. (2011). 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 α 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 $\pi = \langle \pi(1), \ldots, \pi(q), \ldots, \pi(r), \ldots, \pi(n) \rangle$, we define $swap(\pi, q, r)$ as the move that exchanges in π the facility $\pi(q)$ in position q with the facility $\pi(r)$ in position r, producing a new solution $\pi' = \langle \pi(1), \ldots, \pi(q-1), \pi(r), \pi(q+1), \ldots, \pi(r-1), \pi(q), \pi(r+1), \ldots, \pi(n) \rangle$. 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 π, the move $insert(\pi, q, r)$ consists of removing facility $\pi(q)$ from its current position q and inserting it at position r. The new solution π' depends on the relative values of q and r. If $q > r$, then $\pi' = \langle \ldots, \pi(r-1), \pi(q), \pi(r), \pi(r+1), \ldots, \pi(q-1), \pi(q+1), \ldots \rangle$. In contrast, if $q < r$ then $\pi' = \langle \ldots, \pi(q-1), \pi(q+1), \ldots, \pi(r-1), \pi(r), \pi(q), \pi(r+1), \ldots \rangle$. In this case, the associated neighborhood has size $n(n-1)$.

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 π' 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 $\Theta(n^4)$, 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 $\mathcal{O}(n^3)$ time, as described in Kothari and Ghosh (2012c). 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 (Kothari and Ghosh, 2014a). 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. Figure 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 π and a facility $\pi(q)$, the cost of a solution π' after performing the move $swap(\pi, q, q+1)$ can be computed as:

$$C(\pi') = C(\pi) + l_{\pi(q+1)} \left(\sum_{s=1}^{q-1} c_{\pi(s)\pi(q)} - \sum_{s=q+2}^{n} c_{\pi(q)\pi(s)} \right) + l_{\pi(q)} \left(\sum_{s=q+2}^{n} c_{\pi(q+1)\pi(s)} - \sum_{s=1}^{q-1} c_{\pi(s)\pi(q+1)} \right). \tag{3}$$

which runs in $\Theta(n)$ time. The first term of the right hand side of the equation identifies the cost of solution π. 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 $\Theta(n^2)$ 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).

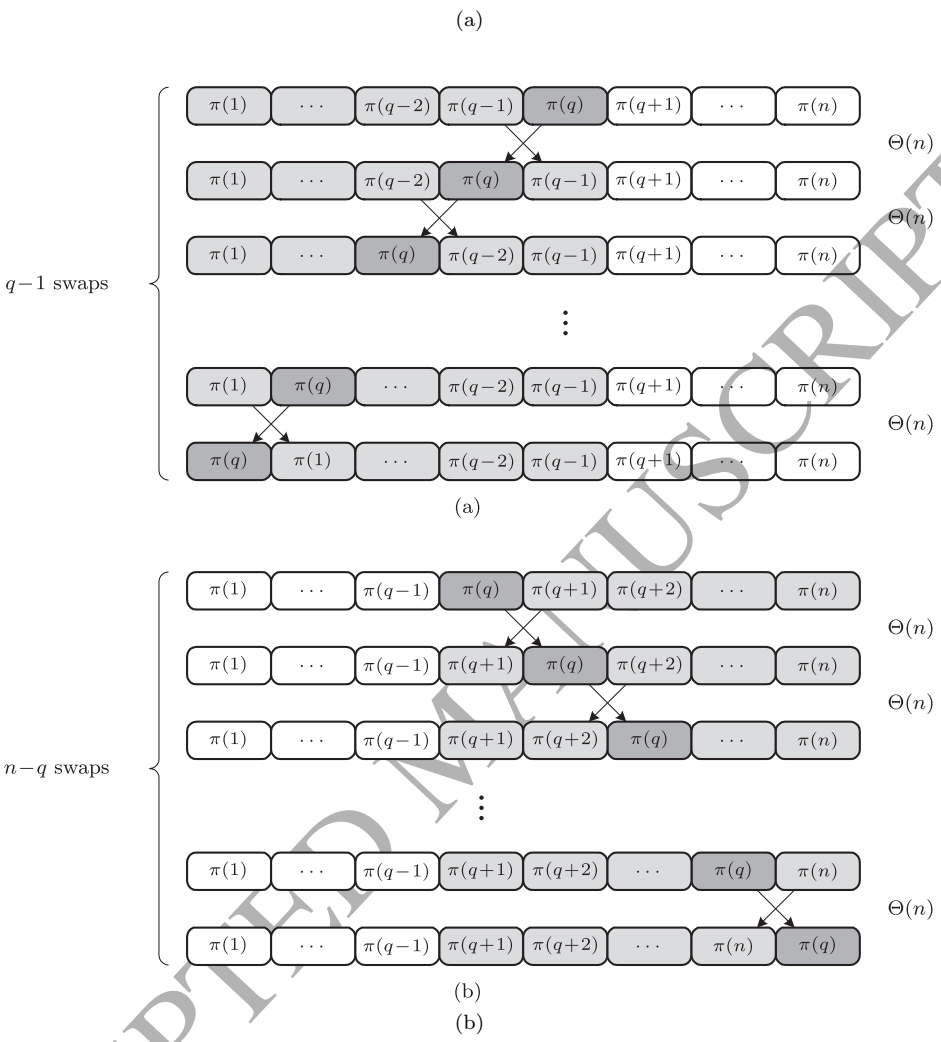


FIGURE 4. Insert moves of a facility (at position q) through contiguous swap operations. The swaps are performed towards the left (a) and right (b). The entire process requires $\Theta(n^2)$ time.

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 (Glover, 1989; 1990). Its current algorithm template is defined in Glover et al. (1998) 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 (2012b), 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 π 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 π^{-1}, which indicates the positions where the facilities are located in π (i.e., facility i is located at position $\pi^{-1}(i)$ in π). For instance, let $\pi_x = \langle 2, 3, 1, 4, 5\rangle$ be some initial ordering and $\pi_y = \langle 5, 2, 1, 4, 3\rangle$ a guiding solution. The inverse representation of π_x is $\pi_x^{-1} = \langle 3, 1, 2, 4, 5\rangle$, since facility 1 is located at the third position in π, facility 2 is located at the first position, and so on. Similarly, the inverse representation of π_y is $\pi_y^{-1} = \langle 3, 2, 5, 4, 1\rangle$. 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 π_x and π_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 π_x so that the q-th facility in π_x will be the same as that in π_y. Formally, the move is defined through $swap(\pi_x, q, \pi_x^{-1}(\pi_y(q)))$. Considering the previous solutions π_x and π_y, the first move would swap facilities 2 and 5 in π_x, generating the intermediate solution $\pi_1 = \langle 5, 3, 1, 4, 2\rangle$, where $\pi_1(1) = \pi_y(1) = 5$. The second one swaps facilities 2 and 3, which produces a new solution $\pi_2 = \langle 5, 2, 1, 4, 3\rangle$, whose two first facilities coincide with those in the guiding solution π_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 facilitates in both solutions that are located at different positions. For instance, considering the previous solutions π_x and π_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 π_x and π_y are $\pi_x^{-1}(i)$ and $\pi_y^{-1}(i)$, respectively. Subsequently, it performs the move $swap(\pi_x, \pi_x^{-1}(i), \pi_y^{-1}(i))$, which 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 π_x and π_y can be efficiently determined by using the inverse representation ($\pi_x^{-1}(3) = 2$ and $\pi_y^{-1}(3) = 5$). In order to situate facility 3 at the same position in both solutions, PR2 performs the move $swap(\pi_x, \pi_x^{-1}(3), \pi_y^{-1}(3)) = swap(\pi_x, 2, 5)$, producing the intermediate solution $\pi_1 = \langle 2, 5, 1, 4, 3\rangle$. The next step would select either facility 2 or 5, generating a swap move between the first two facilities in π_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 (Ulam, 1972), 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 $\mathcal{O}(n^2)$ time by a straightforward dynamic programming algorithm (Sevaux and Sörensen, 2005). However, since the solutions for the SRFLP are orderings of the first n integers,

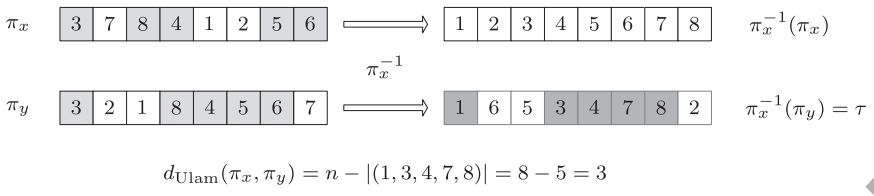


FIGURE 5. The Ulam distance indicates the minimum number of insertions needed to transform some ordering (π_x) into another (π_y). It can be computed by firstly applying π_x^{-1} to both permutations, which simply changes the labeling of the facilities. Subsequently, the longest increasing subsequence $\sigma = (1,3,4,7,8)$ of $\pi_x^{-1}(\pi_y) = \tau$, whose elements appear shaded (in dark gray), indicates the elements $(3, 8, 4, 5, 6)$ that share the same relative ordering in the initial permutations (shown shaded in light gray). The Ulam distance is $n - |\sigma| = 3$, where $|\cdot|$ denotes length, and the path relinking trajectory can be formed by inserting facilities 1, 2, and 7.

it is possible to calculate the Ulam distance by solving the longest increasing subsequence problem instead, which can be computed in $\mathcal{O}(n \log n)$, as we describe below (Hunt and Szymanski, 1977).

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 $\pi_x = \langle 3, 7, 8, 4, 1, 2, 5, 6\rangle$ and $\pi_y = \langle 3, 2, 1, 8, 4, 5, 6, 7\rangle$ 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 Figure 5). Let λ 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 π_x were the ordering $(1, 2, \ldots, n)$ then λ would correspond to the longest increasing subsequence in π_y. However, since π_x can be any ordering, it is necessary to modify the labels of the facilities in order to obtain λ 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 π_x^{-1} to the initial orderings, as shown in Figure 5. Therefore, λ can be found by first computing the longest increasing subsequence in following ordering:

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

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

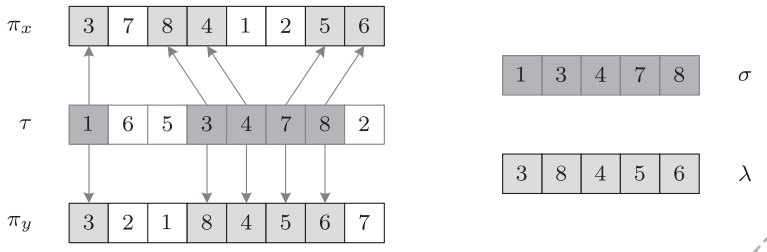


FIGURE 6. Recovering λ in two different ways. On the one hand, σ provides the locations of the elements of λ in π_x. On the other hand, the elements of λ are located in π_y at the same positions as the elements of σ are located in τ.

Subsequently, let σ represent the longest increasing subsequence in τ, which is $(1, 3, 4, 7, 8)$ in the example (note that the elements in σ are not necessarily contiguous). Finally, λ can be recovered through σ in two different ways, as illustrated in Figure 6. On the one hand, σ provides the positions where the elements of λ are located in π_x (upward oblique arrows), i.e., $\lambda = \pi_x(\sigma) = (\pi_x(\sigma(1), \pi_x(\sigma(2)), \ldots)$. Observe in the example that $\lambda = (3, 8, 4, 5, 6) = \pi_x(1, 3, 4, 7, 8)$. On the other hand, the positions in τ of the elements in σ are precisely the positions where the elements of λ are located in π_y (downward arrows). In the example $\lambda = \pi_y(\tau^{-1}(\sigma))$, where $\lambda = (3, 8, 4, 5, 6) = \pi_y(\tau^{-1}((1, 3, 4, 7, 8))) = \pi_y((1, 4, 5, 6, 7))$.

The Ulam distance is the difference between n and the size of σ (which is the same as the size of λ). In the example above it is: $8 - 5 = 3$, which means that we can transform π_x into π_y by performing only three insert moves. Therefore, PR3 would generate a trajectory with only 2 intermediate solutions.

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 λ (in the example, $D = \{1, 2, 7\}$). Firstly, PR3 selects a facility from D at random and inserts it in an adequate position in π_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 (Laguna and Martí, 1999). The combination of both approaches operates on a set of solutions of size b that is usually called the elite set ($ES = \{ES_1, ES_2, \ldots, ES_b\}$). It is typically sorted according to the quality of the solutions (from best, ES_1, to worst, ES_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: GRASP with path relinking algorithm

```
1:  I ← ∅
2:  for i = 1 to m do
3:      π_i ← GRASP()
4:      I ← I ∪ {π_i}
5:  end for
6:  ES ← SelectBest(I, b)
7:  while execution_time < T_max do
8:      π_GRASP ← GRASP()
9:      for i = 1 to b do
10:         π_PR ← PathRelinking(π_GRASP, ES_i)
11:         if (π_PR ∉ ES) and (C(π_PR) ≤ C(ES_b)) then
12:             j ← arg min_{1≤i≤n} {d(π_PR, ES_i)}
13:             if (j ≠ 1) or (C(π_PR) < C(ES_1)) then
14:                 ES_j ← π_PR
15:                 ES ← sort(ES)
16:             end if
17:         end if
18:     end for
19: end while
20: return ES_1
```

Algorithm 3 shows the pseudocode of our proposed GRASP with path relinking procedure. The elite set is initially populated 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 π_{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_{max} (steps 7 to 19). In particular, instead of generating a path between π_{GRASP} and a solution probabilistically selected from ES (as is customary), we compute all path trajectories between π_{GRASP} and each of the solutions in ES (steps 9 to 18). Let π_{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 π_{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 π_{PR} (step 12). In particular, we compute the dissimilarity between solutions through the distance between permutations previously used in Kothari and Ghosh (2014b). Formally, this distance is defined as:

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

where $\bar{\pi}_y$ is the reverse ordering of π_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" (Sevaux and Sörensen, 2005). Finally, π_{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 π_{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 π_{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, Figure 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 π_{GRASP} and solutions in ES. The best solution along the trajectory (π_{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 (2014a) and Kothari and Ghosh (2014b) 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 Anjos (2016). The first one was introduced in Anjos et al. (2005) 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 (2009), 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 `Amaral`, is due to Amaral and Letchford (2013) (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 Rubio-Sánchez et al. (2016).

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 (2014a), and (ii) the scatter search variant SS-2P reported in Kothari

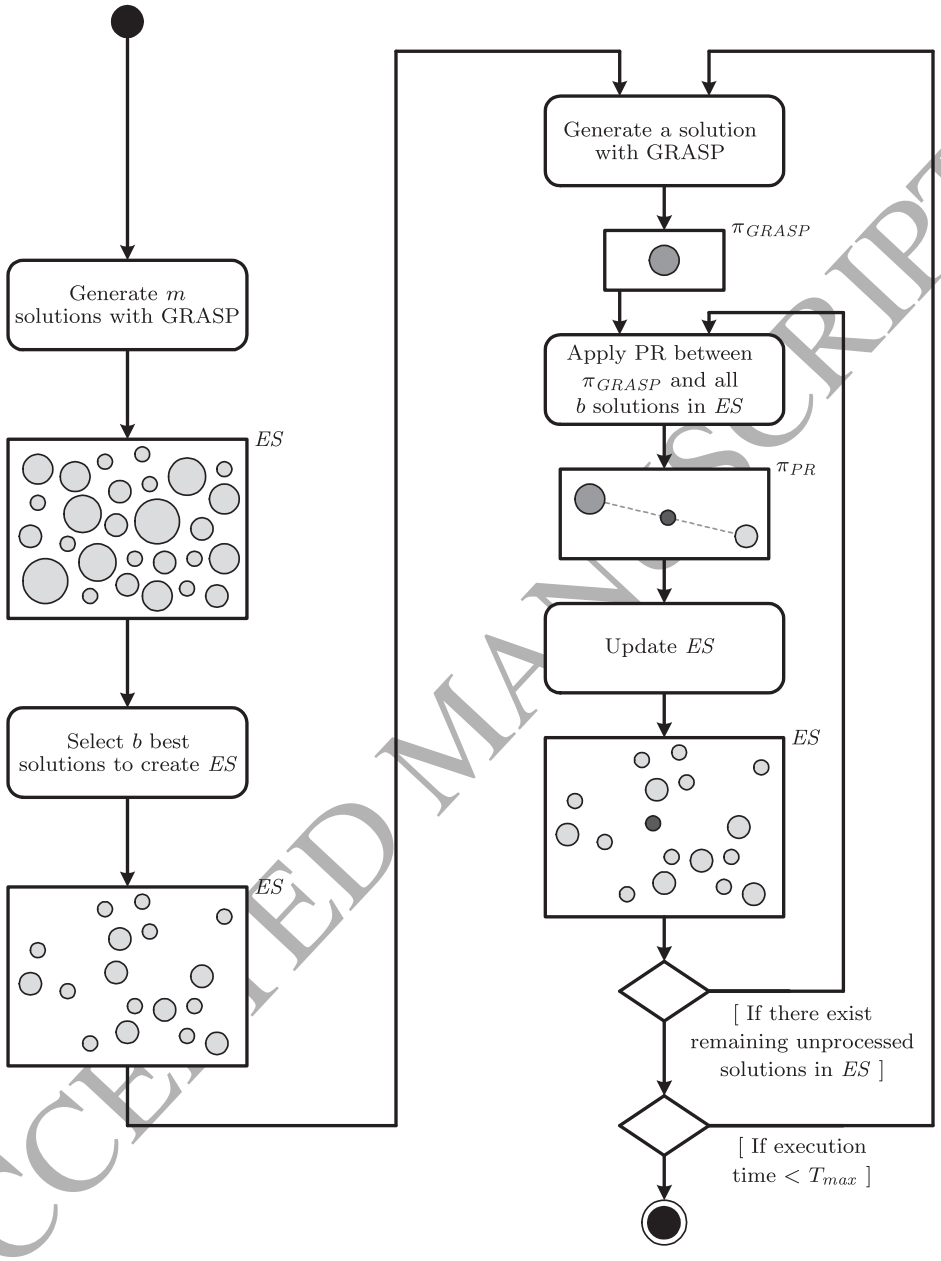


FIGURE 7. Flow diagram of our GRASP with path relinking approach.

and Ghosh (2014b), 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.

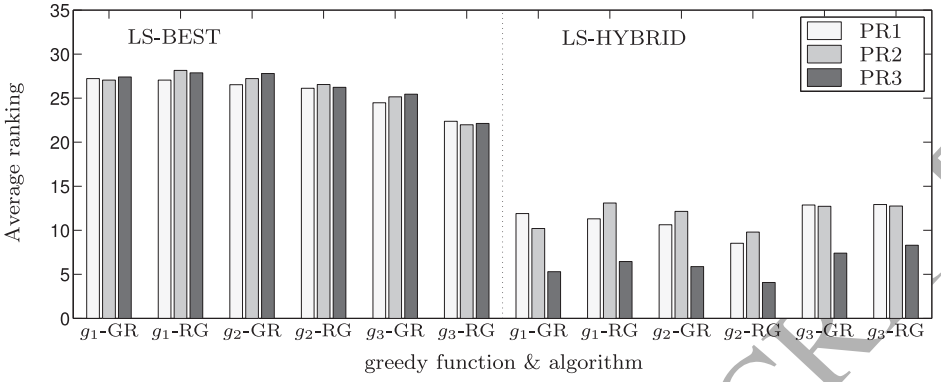


FIGURE 8. Average ranking of 36 GRASP and path relinking algorithms across the 40 training instances. The variants compared use the best or hybrid local searches, the greedy functions g_1, g_2 or g_3, the GR or RG algorithms, and the path relinking strategies PR1, PR2 or PR3.

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 α 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 seconds, we set the maximum execution time to $2n$ seconds 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.

Figure 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 $2n$ seconds) 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_2 and RG provides the best results. Therefore, we define our final algorithm for solving the SRFLP, denoted as GRASP-PR, as the combination of LS-HYBRID, PR3, g_2, 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 $n/2$ seconds, 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 one hour). 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 (2014a). 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.

TABLE 1. Comparison of performance of GENALGO, SS-2P, and GRASP-PR on `Anjos`, `sko`, and `Amaral` instances.

	GENALGO		SS-2P		GRASP-PR	
Instance	Cost	Time	Cost	Time	Cost	Time
Anjos_60_1	1477834	2110	1477834	41	1477834	30
Anjos_60_2	841776	2126	841776	47	841776	30
Anjos_60_3	648337.5	2200	648337.5	65	648337.5	30
Anjos_60_4	398406	2154	398406	53	398406	30
Anjos_60_5	318805	2102	318805	48	318805	30
Anjos_70_1	1528537	3670	1528537	91	1528537	35
Anjos_70_2	1441028	3556	1441028	97	1441028	35
Anjos_70_3	1518993.5	3658	1518993.5	100	1518993.5	35
Anjos_70_4	968796	3786	968796	95	968796	35
Anjos_70_5	4218002.5	3572	4218002.5	80	4218002.5	35
Anjos_75_1	2393456.5	5008	2393456.5	119	2393456.5	37.5
Anjos_75_2	4321190	4996	4321190	137	4321190	37.5
Anjos_75_3	1248423	4994	1248423	137	1248423	37.5
Anjos_75_4	3941816.5	4974	3941816.5	122	3941816.5	37.5
Anjos_75_5	1791408	5108	1791408	121	1791408	37.5
Anjos_80_1	2069097.5	6532	2069097.5	172	2069097.5	40
Anjos_80_2	1921136	6304	1921136	167	1921136	40
Anjos_80_3	3251368	6362	3251368	147	3251368	40
Anjos_80_4	3746515	6336	3746515	182	3746515	40
Anjos_80_5	1588885	6562	1588885	205	1588885	40
sko_64_1	96881	3084	[96890]*	93	96881	32
sko_64_2	634332.5	2810	634332.5	89	634332.5	32
sko_64_3	414323.5	2832	414323.5	68	414323.5	32
sko_64_4	297129	2870	297129	85	297129	32
sko_64_5	501922.5	2800	501922.5	78	501922.5	32
sko_72_1	139150	5090	139150	150	139150	36
sko_72_2	711998	4582	711998	226	711998	36
sko_72_3	1054110.5	4480	1054110.5	112	1054110.5	36
sko_72_4	919586.5	4446	919586.5	132	919586.5	36
sko_72_5	428226.5	4638	[428228.5]*	172	428226.5	36
sko_81_1	[205108.5]*	8154	[205112]*	274	205106	40.5
sko_81_2	521391.5	7326	521391.5	227	521391.5	40.5
sko_81_3	970796	7186	970796	263	970796	40.5
sko_81_4	2031803	7370	2031803	237	2031803	40.5
sko_81_5	1302711	7116	1302711	257	1302711	40.5
sko_100_1	378234	19950	378234	790	378234	50
sko_100_2	2076008.5	17264	2076008.5	468	2076008.5	50
sko_100_3	16145614.5	16588	[16149444]*	530	16145614.5	50
sko_100_4	3232522	16302	3232522	532	3232522	50
sko_100_5	1033080.5	17074	1033080.5	619	1033080.5	50
Amaral_110_1	144296664.5	38398	144296664.5	777	144296664.5	55
Amaral_110_2	86050037	38134	86050037	775	86050037	55
Amaral_110_3	2234743.5	23914	2234743.5	611	2234743.5	55

$[\cdot]^*$ Did not find the best value.

TABLE 2. Performance of our implementations of SS-2P and GENALGO on `Anjos`, `sko`, and `Amaral` instances.

	GENALGO_R		SS-2P_R	
Instance	Cost	Time	Cost	Time
Anjos_60_1	1477834	585	1477834	10.0
Anjos_60_2	841776	617	841776	11.6
Anjos_60_3	648337.5	678	648337.5	8.2
Anjos_60_4	398406	686	398406	12.3
Anjos_60_5	318805	615	318805	7.3
Anjos_70_1	1528537	1150	1528537	13.9
Anjos_70_2	1441028	1149	1441028	23.0
Anjos_70_3	1518993.5	1159	1518993.5	23.7
Anjos_70_4	968796	1294	968796	22.3
Anjos_70_5	4218002.5	1047	4218002.5	19.2
Anjos_75_1	2393456.5	1518	2393456.5	18.1
Anjos_75_2	4321190	1634	4321190	38.1
Anjos_75_3	1248423	1735	1248423	36.8
Anjos_75_4	3941816.5	1503	3941816.5	25.5
Anjos_75_5	1791408	1657	1791408	35.1
Anjos_80_1	2069097.5	2078	2069097.5	33.5
Anjos_80_2	1921136	1917	1921136	33.5
Anjos_80_3	3251368	1918	3251368	37.0
Anjos_80_4	3746515	1993	3746515	22.8
Anjos_80_5	1588885	2200	1588885	46.2
sko_64_1	96881	1158	96881	14.0
sko_64_2	634332.5	964	634332.5	13.1
sko_64_3	414323.5	995	414323.5	15.1
sko_64_4	297129	1081	297129	22.6
sko_64_5	501922.5	965	501922.5	17.4
sko_72_1	139150	1694	139150	42.1
sko_72_2	711998	1706	711998	45.8
sko_72_3	1054110.5	1716	[1054480.5]*	18.5
sko_72_4	919586.5	1564	919586.5	21.6
sko_72_5	428226.5	1683	428226.5	41.4
sko_81_1	205106	3297	205106	59.0
sko_81_2	521391.5	2620	521391.5	33.6
sko_81_3	970796	2881	970796	47.7
sko_81_4	2031803	2632	[2031826]*	38.9
sko_81_5	1302711	2643	1302711	30.5
sko_100_1	378234	7835	378234	101.1
sko_100_2	2076008.5	6607	2076008.5	143.8
sko_100_3	16145614.5	6324	16145614.5	90.0
sko_100_4	3232522	6262	3232522	177.6
sko_100_5	1033080.5	6861	[1033320.5]*	86.6
Amaral_110_1	144296664.5	9899	144296664.5	153.0
Amaral_110_2	86050037	9675	86050037	112.5
Amaral_110_3	2234743.5	8931	2234743.5	225.3

$[\cdot]^*$ Did not find the best value.

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

Instance	GENALGO_R	SS-2P_R	GRASP-PR
Anjos_200_1	305497735	305461895	**305461862**
Anjos_200_2	**178807686.5**	178852729.5	178816295.5
Anjos_200_3	61893221	61891688	**61891275**
Anjos_200_4	127743269	127736617	**127736350**
Anjos_200_5	89059441.5	89141170.5	**89057182.5**
Anjos_300_1	1550458661	1552008518	**1549663680**
Anjos_300_2	956868890.5	957367664.5	**955572080.5**
Anjos_300_3	308456080.5	308868436.5	**308257766.5**
Anjos_300_4	603280369.5	603679094.5	**602873363.5**
Anjos_300_5	467360444	466172304	**466160315**
Anjos_400_1	5007041358.5	5007312343.5	**5000752665.5**
Anjos_400_2	2916949529	2917158558	**2910279558**
Anjos_400_3	922299658	922534333	**921216592**
Anjos_400_4	1809796198	1809847357	**1806067255**
Anjos_400_5	1404633352.5	1405722365.5	**1402779504.5**
Anjos_500_1	12318818681	12326499609	**12300427281**
Anjos_500_2	7514421872.5	7505998255.5	**7493143632.5**
Anjos_500_3	2485064290	2483586441	**2479334789**
Anjos_500_4	4294960811.5	4294352718.5	**4285937468.5**
Anjos_500_5	3689059932.5	3680497633.5	**3678038149.5**
sko_200_1	3233194	3231656	**3231383**
sko_200_2	7758971	7758940	**7758937**
sko_200_3	12741321	12741126	**12739045**
sko_200_4	20263543	20268410	**20260611**
sko_200_5	26873333.5	26873161.5	**26871979.5**
sko_300_1	11279331	11269514	**11251962**
sko_300_2	29008482	29061358	**28993837**
sko_300_3	48875875.5	48899319.5	**48800528.5**
sko_300_4	71313267.5	71342723.5	**71194215.5**
sko_300_5	86834415.5	86820444.5	**86792129.5**
sko_400_1	26793018	26790436	**26718566**
sko_400_2	68058002	68111630	**67996135**
sko_400_3	116179989	116182660	**115942730**
sko_400_4	163542874	163544831	**163099036**
sko_400_5	228268699.5	228153858.5	**227787876.5**
sko_500_1	53118100	53130663	**52952086**
sko_500_2	127971045	127664532	**127428491**
sko_500_3	231573411.5	231487769.5	**231044722.5**
sko_500_4	341660232	340848573	**340390687**
sko_500_5	446537443	446702649	**445750842**

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 seconds. Tables 3 and 4 show the associated costs, where the best found values for each of the instances are highlighted in boldface.

TABLE 4. Comparison of SRFLP algorithms on `Anjos-large` and `sko-large` instances, when executing them for one hour.

Instance	GENALGO_R	SS-2P_R	GRASP-PR
Anjos_200_1	305497727	**305461818**	**305461818**
Anjos_200_2	**178807197.5**	178852677.5	178816261.5
Anjos_200_3	61892040	61891652	**61891275**
Anjos_200_4	127743257	127736464	**127735691**
Anjos_200_5	89059083.5	89141158.5	**89057121.5**
Anjos_300_1	1550443050	1550822258	**1549663657**
Anjos_300_2	955749527.5	957249994.5	**955572066.5**
Anjos_300_3	308372773.5	308701657.5	**308257630.5**
Anjos_300_4	603268519.5	603678289.5	**602873168.5**
Anjos_300_5	466717889	466162354	**466160264**
Anjos_400_1	5004496747.5	5005963158.5	**5000752142.5**
Anjos_400_2	2916949529	2916726055	**2910276759**
Anjos_400_3	921346203	922110067	**921216455**
Anjos_400_4	1809520966	1809564367	**1806061379**
Anjos_400_5	1404580939.5	1405152456.5	**1402779472.5**
Anjos_500_1	12311278683	12304668851	**12300409839**
Anjos_500_2	7505406907.5	7501448013.5	**7493120635.5**
Anjos_500_3	2483990108	2483197897	**2479333773**
Anjos_500_4	4288172936.5	4293459100.5	**4285937468.5**
Anjos_500_5	3685342697.5	3680148707.5	**3678038066.5**
sko_200_1	3233188	3231654	**3231379**
sko_200_2	7758950	7758939	**7758927**
sko_200_3	12740236	12741031	**12739043**
sko_200_4	20263483	20268399	**20260531**
sko_200_5	26873277.5	26873153.5	**26871976.5**
sko_300_1	11278708	11263368	**11251960**
sko_300_2	29008472	29061348	**28993831**
sko_300_3	48828528.5	48894938.5	**48800510.5**
sko_300_4	71303673.5	71259866.5	**71194203.5**
sko_300_5	86834403.5	86806304.5	**86792128.5**
sko_400_1	26780355	26754928	**26718485**
sko_400_2	68058002	68046421	**67996107**
sko_400_3	116150616	116088506	**115942726**
sko_400_4	163359654	163371846	**163099026**
sko_400_5	228119749.5	228115330.5	**227778641.5**
sko_500_1	53030465	53117838	**52951975**
sko_500_2	127631162	127648186	**127428477**
sko_500_3	231465645.5	231408793.5	**231041993.5**
sko_500_4	341175620	340848111	**340390652**
sko_500_5	446498553	446641044	**445738609**

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

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

Execution time	Average ranking			p-value
	GENALGO_R	SS-2P_R	GRASP-PR	
n seconds	2.45	2.53	1.03	3.9×10^{-13}
1 hour	2.50	2.46	1.04	7.0×10^{-13}

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

Compared algorithms	*sign*	p-value
GRASP-PR vs. GENALGO_R	39	3.7×10^{-11}
GRASP-PR vs. SS-2P_R	40	9.1×10^{-13}

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

Compared algorithms	*sign*	p-value
GRASP-PR vs. GENALGO_R	39	3.7×10^{-11}
GRASP-PR vs. SS-2P_R	39	1.8×10^{-12}

GRASP-PR) in one instance when considering 3600 seconds. 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 (2014a), and the scatter search procedure proposed in Kothari and Ghosh (2014b).

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

TABLE 8. Execution time needed by SS-2R_R and GENALGO_R in order to match or enhance the solutions provided by GRASP-PR after running the algorithm for one hour.

	Halting time (in seconds)	
Instance	SS-2R_R	GENALGO_R
Anjos_200_1	2045	16087
Anjos_200_2	> 36000	754
Anjos_200_3	> 36000	> 36000
Anjos_200_4	> 36000	> 36000
Anjos_200_5	> 36000	7441
Anjos_300_1	> 36000	> 36000
Anjos_300_2	> 36000	27662
Anjos_300_3	> 36000	> 36000
Anjos_300_4	> 36000	> 36000
Anjos_300_5	7556	> 36000
Anjos_400_1	> 36000	> 36000
Anjos_400_2	> 36000	> 36000
Anjos_400_3	11454	10978
Anjos_400_4	> 36000	> 36000
Anjos_400_5	> 36000	13070
Anjos_500_1	> 36000	33609
Anjos_500_2	> 36000	32365
Anjos_500_3	> 36000	> 36000
Anjos_500_4	> 36000	24413
Anjos_500_5	18777	> 36000
Sko_200_1	> 36000	> 36000
Sko_200_2	> 36000	> 36000
Sko_200_3	> 36000	> 36000
Sko_200_4	> 36000	17074
Sko_200_5	> 36000	> 36000
Sko_300_1	> 36000	> 36000
Sko_300_2	> 36000	> 36000
Sko_300_3	> 36000	> 36000
Sko_300_4	8246	10584
Sko_300_5	> 36000	> 36000
Sko_400_1	> 36000	> 36000
Sko_400_2	> 36000	27070
Sko_400_3	> 36000	> 36000
Sko_400_4	> 36000	13900
Sko_400_5	13691	29891
Sko_500_1	> 36000	33649
Sko_500_2	> 36000	> 36000
Sko_500_3	> 36000	> 36000
Sko_500_4	> 36000	> 36000
Sko_500_5	> 36000	> 36000

they reached the value attained by our GRASP-PR after running it for one hour on the new instances (Table 4 indicates the associated costs), or for a maximum of 10 hours. 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

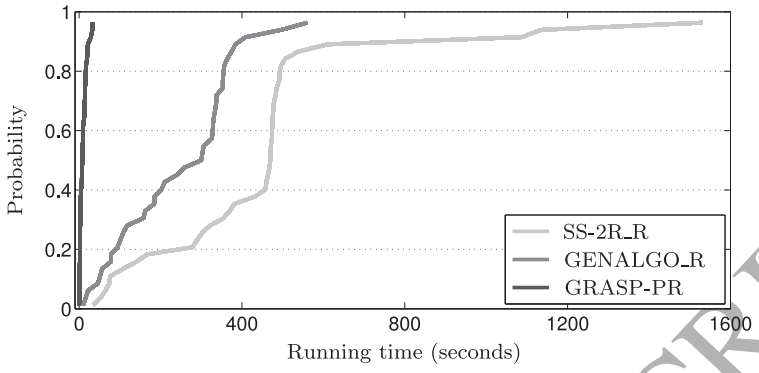


FIGURE 9. Time-to-target plot for SS-2R_R, GENALGO_R, and GRASP-PR after running them 40 times over instance Anjos_200_1.

the solutions of GRASP-PR after running for 10 hours. Note that this situation occurred in 59 out of the 80 cases.

Lastly, Figure 9 shows a TTT (time-to-target) plot Aiex et al. (2006) 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.

7. Conclusions

This paper has described an algorithm for finding high-quality solutions to the SRFLP 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 (Kothari and Ghosh, 2012b), and a scatter search method (Kothari and Ghosh, 2012a)), 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 one hour. 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 Rubio-Sánchez et al. (2016).

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

Renata M. Aiex, Mauricio G. C. Resende, and Celso C. Ribeiro. Ttt plots: a perl program to create time-to-target plots. *Optimization Letters*, 1(4):355–366, 2006. ISSN 1862-4480. doi: 10.1007/s11590-006-0031-4.

André R. S. Amaral. On the exact solution of a facility layout problem. *European Journal of Operational Research*, 173(2):508–518, 2006. ISSN 0377-2217. doi: 10.1016/j.ejor.2004.12.021.

André R. S. Amaral. An exact approach to the one-dimensional facility layout problem. *Operations Research*, 56(4):1026–1033, August 2008. doi: 10.1287/opre.1080.0548.

André R. S. Amaral. A new lower bound for the single row facility layout problem. *Discrete Applied Mathematics*, 157(1):183–190, January 2009. ISSN 0166-218X. doi: 10.1016/j.dam.2008.06.002.

André R. S. Amaral and Adam N. Letchford. A polyhedral approach to the single row facility layout problem. *Mathematical Programming*, 141(1-2):453–477, October 2013. ISSN 0025-5610. doi: 10.1007/s10107-012-0533-z.

Miguel F. Anjos, 2016. URL `www.miguelanjos.com/flplib`.

Miguel F. Anjos and Anthony Vannelli. Computing globally optimal solutions for single-row layout problems using semidefinite programming and cutting planes. *INFORMS Journal on Computing*, 20(4):611–617, May 2008. doi: 10.1287/ijoc.1080.0270.

Miguel F. Anjos and Ginger Yen. Provably near-optimal solutions for very large single-row facility layout problems. *Optimization Methods Software*, 24(4-5):805–817, August 2009. ISSN 1055-6788. doi: 10.1080/10556780902917735.

Miguel F. Anjos, Andrew Kennings, and Anthony Vannelli. A semidefinite optimization approach for the single-row layout problem with unequal dimensions. *Discrete Optimization*, 2(2):113–122, June 2005. ISSN 1572-5286. doi: 10.1016/j.disopt.2005.03.001.

V. Campos, R. Martí, J. Sanchez-Oro, and A. Duarte. GRASP with PR for the orienteering problem. *Journal of the Operational Research Society*, 2013. doi: 10.1057/jors.2013.156.

Dilip Datta, André R.S. Amaral, and José Rui Figueira. Single row facility layout problem using a permutation-based genetic algorithm. *European Journal of Operational Research*, 213(2):388–394, 2011. ISSN 0377-2217. doi: 10.1016/j.ejor.2011.03.034.

A. Duarte, R. Martí, M.G.C. Resende, and R.M.A. Silva. GRASP with path relinking heuristics for the antibandwidth problem. *Networks*, 58(3):171–189, 2011.

T. A. Feo, M. G. C. Resende, and S. H. Smith. A greedy randomized adaptive search procedure for maximum independent set. *Operations Research*, 42:860–878, 1994.

Thomas A. Feo and Mauricio G. C. Resende. A probabilistic heuristic for a computationally difficult set covering problem. *Operations Research Letters*, 8:67–71, 1989. doi: 10.1016/0167-6377(89)90002-3.

F. Glover. Tabu search - part 1. *ORSA Journal on Computing*, 1(2):190–206, 1989.

F. Glover. Tabu search - part 2. *ORSA Journal on Computing*, 2(1):4–32, 1990.

F. Glover, C.C. Kuo, and K.S. Dhir. Heuristic algorithms for the maximum diversity problem. *Journal of Information and Optimization Sciences*, 19(1):109–132, 1998.

Fred Glover. A template for scatter search and path relinking. In *Selected Papers from the Third European Conference on Artificial Evolution*, pages 3–54. Springer-Verlag, 1998. ISBN 3-540-64169-6.

Sunderesh S. Heragu and Attahiru Sule Alfa. Experimental analysis of simulated annealing based algorithms for the layout problem. *European Journal of Operational Research*, 57(2):190–202, 1992. ISSN 0377-2217. doi: 10.1016/0377-2217(92)90042-8.

Sunderesh S. Heragu and Andrew Kusiak. Machine layout problem in flexible manufacturing systems. *Operations Research*, 36(2):258–268, March 1988. doi: 10.1287/opre.36.2.258.

Sunderesh S. Heragu and Andrew Kusiak. Efficient models for the facility layout problem. *European Journal of Operational Research*, 53(1):1–13, 1991. ISSN 0377-2217. doi: 10.1016/0377-2217(91)90088-D.

Philipp Hungerländer. Single-row equidistant facility layout as a special case of single-row facility layout. *International Journal of Production Research*, 52(5):1257–1268, 2014. ISSN 0020-7543. doi: 10.1080/00207543.2013.828163.

Philipp Hungerländer and Franz Rendl. A computational study and survey of methods for the single-row facility layout problem. *Computational Optimization and Applications*, 55(1):1–20, May 2013. ISSN 0926-6003. doi: 10.1007/s10589-012-9505-8.

James W. Hunt and Thomas G. Szymanski. A fast algorithm for computing longest common subsequences. *Communications of the ACM*, 20(5):350–353, May 1977. ISSN 0001-0782. doi: 10.1145/359581.359603.

Birgit Keller and Udo Buscher. Single row layout models. *European Journal of Operational Research*, 245(3):629–644, 2015. ISSN 0377-2217. doi: 10.1016/j.

ejor.2015.03.016. To appear.

Ravi Kothari and Diptesh Ghosh. The single row facility layout problem: state of the art. *OPSEARCH*, 49(4):442–462, December 2012a. ISSN 0030-3887. doi: 10.1007/s12597-012-0091-4.

Ravi Kothari and Diptesh Ghosh. Path relinking for single row facility layout. Technical Report 2012-05-01, Indian Institute of Management, May 2012b.

Ravi Kothari and Diptesh Ghosh. Tabu search for the single row facility layout problem using exhaustive 2-opt and insertion neighborhoods. Technical Report 2012-01-03, Indian Institute of Management, January 2012c.

Ravi Kothari and Diptesh Ghosh. Insertion based lin-kernighan heuristic for single row facility layout. *Computers and Operations Research*, 40(1):129–136, January 2013a. ISSN 0305-0548. doi: 10.1016/j.cor.2012.05.017.

Ravi Kothari and Diptesh Ghosh. Tabu search for the single row facility layout problem using exhaustive 2-opt and insertion neighborhoods. *European Journal of Operational Research*, 224(1):93–100, 2013b. ISSN 0377-2217. doi: 10.1016/j.ejor.2012.07.037.

Ravi Kothari and Diptesh Ghosh. An efficient genetic algorithm for single row facility layout. *Optimization Letters*, 8(2):679–690, February 2014a. ISSN 1862-4472. doi: 10.1007/s11590-012-0605-2.

Ravi Kothari and Diptesh Ghosh. A scatter search algorithm for the single row facility layout problem. *Journal of Heuristics*, 20(2):125–142, April 2014b. ISSN 1381-1231. doi: 10.1007/s10732-013-9234-x.

K. Ravi Kumar, George C. Hadjinicola, and Ting li Lin. A heuristic procedure for the single-row facility layout problem. *European Journal of Operational Research*, 87(1):65–73, 1995. ISSN 0377-2217. doi: 10.1016/0377-2217(94)00062-H.

Satheesh Kumar, Asokan P. Kumanan, and S. Varma. Scatter search algorithm for single row layout problem in FMS. *Advances in Production Engineering & Management*, 3(4):193–204, 2008. ISSN 1854-6250.

M. Laguna and R. Martí. GRASP and path relinking for 2-layer straight line crossing minimization. *INFORMS Journal on Computing*, 11(1):44–52, 1999.

P. Larrañaga, C. M. H. Kuijpers, R. H. Murga, I. Inza, and S. Dizdarevic. Genetic algorithms for the travelling salesman problem: A review of representations and operators. *Artif. Intell. Rev.*, 13(2):129–170, April 1999. ISSN 0269-2821.

Robert F. Love and Jsun Y. Wong. On solving a one-dimensional space allocation problem with integer programming. *INFOR: Information Systems and Operational Research*, 14:139–144, January 1976.

J.J. Pantrigo, R. Martí, A. Duarte, and E.G. Pardo. Scatter search for the cutwidth minimization problem. *Annals of Operations Research*, 199(1):285–304, 2012.

Jean-Claude Picard and Maurice Queyranne. On the one-dimensional space allocation problem. *Operations Research*, 29(2):371–391, April 1981. doi: 10.1287/opre.29.2.371.

A. S. Ramkumar and S. G. Ponnambalam. Design of single-row layouts for flexible manufacturing systems using genetic algorithm and simulated annealing algorithm. In *2004 IEEE Conference on Cybernetics and Intelligent Systems*, volume 2, pages 1143–1147, December 2004. doi: 10.1109/ICCIS.2004.1460751.

M. G. C. Resende, R. Martí, M. Gallego, and A. Duarte. GRASP and path relinking for the max-min diversity problem. *Computers & Operations Research*, 37(3):498–508, 2010.

Mauricio G. C. Resende and Celso C. Ribeiro. Greedy randomized adaptive search procedures: Advances, hybridizations, and applications. In M. Gendreau and J.-Y. Potvin, editors, *Handbook of Metaheuristics*, volume 146 of *International Series in Operations Research & Management Science*, pages 283–319. Springer US, 2nd edition, 2010.

Mauricio G. C. Resende and Celso C. Ribeiro. GRASP: Greedy randomized adaptive search procedures. In E.K. Burke and G. Kendall, editors, *Search Methodologies*, pages 287–312. Springer US, 2014.

Mauricio G. C. Resende and R. F. Werneck. A hybrid heuristic for the p-median problem. *Journal of Heuristics*, 10(1):59–88, 2004. ISSN 1381-1231.

David Romero and Adolfo Sánchez-Flores. Methods for the one-dimensional space allocation problem. *Computers & Operations Research*, 17(5):465–473, June 1990. ISSN 0305-0548. doi: 10.1016/0305-0548(90)90051-8.

Manuel Rubio-Sánchez, Micael Gallego, Francisco Gortázar, and Abraham Duarte, 2016. URL `www.optsicom.es/srflp/`.

Hamed Samarghandi and Kourosh Eshghi. An efficient tabu algorithm for the single row facility layout problem. *European Journal of Operational Research*, 205(1): 98–105, 2010. ISSN 0377-2217. doi: 10.1016/j.ejor.2009.11.034.

Hamed Samarghandi, Pouria Taabayan, and Farzad Firouzi Jahantigh. A particle swarm optimization for the single row facility layout problem. *Computers & Industrial Engineering*, 58(4):529–534, 2010. ISSN 0360-8352. doi: 10.1016/j.cie.2009.11.015.

Sujeevraja Sanjeevi and Kiavash Kianfar. A polyhedral study of triplet formulation for single row facility layout problem. *Discrete Applied Mathematics*, 158(16): 1861–1867, 2010. ISSN 0166-218X. doi: 10.1016/j.dam.2010.07.005.

Marc Sevaux and Kenneth Sörensen. Permutation distance measures for memetic algorithms with population managment. In *MIC2005, Proceedings of 6th Metaheuristics International Conference*, pages 832–838, 2005.

Donald M. Simmons. One-dimensional space allocation: An ordering algorithm. *Operations Research*, 17(5):812–826, October 1969. doi: 10.1287/opre.17.5.812.

M. Solimanpur, Prem Vrat, and Ravi Shankar. An ant algorithm for the single row layout problem in flexible manufacturing systems. *Computers & Operations Research*, 32(3):583–598, 2005. ISSN 0305-0548. doi: 10.1016/j.cor.2003.08.005.

S. M. Ulam. Some ideas and prospects in biomathematics. *Annual review of biophysics and bioengineering*, 1:277–292, 1972. ISSN 0084-6589. doi: 10.1146/annurev.bb.01.060172.001425.

(M. Rubio-Sánchez) Department of Computer Science and Statistics, Universidad Rey Juan Carlos, Spain.
E-mail address: `manuel.rubio@urjc.es`

(M. Gallego) Department of Computer Science and Statistics, Universidad Rey Juan Carlos, Spain.
E-mail address: `micael.gallego@urjc.es`

(F. Gortazar) Department of Computer Science and Statistics, Universidad Rey Juan Carlos, Spain.
E-mail address: `francisco.gortazar@urjc.es`

(A. Duarte) Department of Computer Science and Statistics, Universidad Rey Juan Carlos, Spain.
E-mail address: `abraham.duarte@urjc.es`