Insertion based Lin-Kernighan heuristic for single row facility layout

Ravi Kothari*, Diptesh Ghosh

P&QM Area, IIM Ahmedabad, Vastrapur, Ahmedabad 380015, Gujarat, INDIA

Abstract

The single row facility layout problem (SRFLP) is the problem of arranging facilities with given lengths on a line, while minimizing the weighted sum of the distances between all pairs of facilities. The problem is known to be NP-hard. In this paper, we present a neighborhood search heuristic called LK-INSERT which uses a Lin-Kernighan neighborhood structure built on insertion neighborhoods. To the best of our knowledge this is the first such heuristic for the SRFLP. Our computational experiments show that LK-INSERT is competitive for all instances, and it improves the best known solutions for several large sized benchmark SRFLP instances.

Keywords: Facilities planning and design; Single Row Facility Layout, Lin-Kernighan Neighborhood, Insertion Neighborhood, Local search

1. Introduction

Given a set F of facilities, where each facility has a particular length, and there is a weight corresponding to each pair of facilities, the single row facility layout problem (SRFLP) is the NP-hard problem of arranging the facilities in a line so as to minimize the weighted sum of the distances between facility pairs, where the distance between a pair of facilities is the distance between their centroids. The size of a SRFLP instance is the number of facilities in the instance. This problem was first proposed in [29] and was shown to be NP-Hard in [8]. Formally stated the SRFLP is defined as follows:

Given: A set $F = \{1, 2, \ldots, n\}$ of $n > 2$ facilities, where facility j has length l_j, and weights c_{ij} for each pair (i, j) of facilities, $i, j \in F$, $i \neq j$.

Objective: To find a permutation $\Pi = (\pi_1, \pi_2, \ldots, \pi_n)$ of facilities in F that minimizes the

*Corresponding author. Tel.: +91 7878088002

Email addresses: ravikothari@iimahd.ernet.in (Ravi Kothari), diptesh@iimahd.ernet.in (Diptesh Ghosh)

cost of the permutation

$$z(\Pi) = \sum_{1 \le i < j \le n} c_{\pi_i \pi_j} d_{\pi_i \pi_j}$$

where $d_{\pi_i \pi_j} = l_{\pi_i}/2 + \sum_{i<k<j} l_{\pi_k} + l_{\pi_j}/2$ is the distance between the centroids of facilities π_i and π_j when the facilities in F are ordered as per the permutation Π.

In this paper we present a competitive heuristic for the SRFLP. Our paper is organized as follows. In Section 2 we review the literature on the SRFLP, concentrating on the solution methodologies for the problem. We present the details of Lin-Kernighan neighborhood search for the SRFLP in Section 3 and describe our LK-INSERT heuristic using this neighborhood in Section 4. We present results from our computational experiments with LK-INSERT in Section 5. Finally we conclude the paper in Section 6 with a summary of the work and future research directions.

2. A review of the literature on the SRFLP

The literature on the SRFLP (see [15] for a comprehensive review) is rich in numerous practical applications. The SRFLP has been used to design arrangements of rooms in hospitals, departments in office buildings or in supermarkets [29], to arrange machines in flexible manufacturing systems [12], to assign files to disk cylinders in computer storage, and to design warehouse layouts [24]. Apart from these direct applications, there are a large number of applications of the special case of the SRFLP in which the facilities have equal lengths. These include the triangulation problem of input output tables in economics [20], and ranking of teams in sports [23].

Various exact and approximate solution approaches to the problem have been proposed since 1969. Exact methods are used to solve the SRFLP to optimality, and include branch and bound [29], linear mixed integer programming [2, 3, 13, 22], cutting planes [4], dynamic programming [18, 24], branch and cut [1], and semidefinite programming [5–7, 14]. Since the SRFLP is NP-hard, exact methods are computationally prohibitive to solve large instances. The exact methods mentioned above have been able to obtain optimal solutions to SRFLP instances with up to 42 facilities.

For large sized instances, the literature on the SRFLP suggests the use of heuristics, non-exact methods that obtain near-optimal solutions within reasonable time. Construction heuristics for the SRFLP have been presented in [9, 12, 25]. However these have been superseded by improvement heuristics. Most improvement heuristics for the SRFLP are meta-heuristics, e.g. simulated annealing [11, 17, 26], ant colony optimization [30], scatter search [19], tabu search [27], particle swarm optimization [28], and genetic algorithms [10].

Among these, the tabu search implementation in [27] and the genetic algorithm implementation in [10] yield best results for benchmark SRFLP instances of large sizes.

Neighborhood search heuristics such as tabu search and simulated annealing use small sized neighborhoods such as 2-opt and insertion neighborhoods. The sizes of both these neighborhoods are $O(n^2)$, and since the calculation of the cost of a SRFLP solution requires quadratic time, the time to search such neighborhoods is $O(n^4)$. Most neighborhood search implementations for the SRFLP thus do not search neighborhoods exhaustively. However [16] presents techniques which allow exhaustive search of both 2-opt and insertion neighborhoods in $O(n^3)$ time, thus resulting in the TS-2OPT and TS-INSERT heuristics which perform better than other neighborhood search heuristics.

In this paper, we consider a more elaborate neighborhood for the SRFLP. This neighborhood can be described in terms of the moves that define it. Consider a basic move in conventional neighborhood search algorithms for the SRFLP like a 2-opt move or an insertion move. Now consider a composite move which consist of several successive basic moves starting from a given permutation. The set of permutations that can be obtained by such composite moves make up the neighborhood of the permutation. This type of moves have been known to be very attractive in traveling salesman problems and is commonly known as Lin-Kernighan neighborhoods (see the Lin-Kernighan heuristic described in [21] for more details). Till date there has been no study using such Lin-Kernighan neighborhood search for the SRFLP. In this paper, we use the insertion move as the basic move, since both [26] and [16] recommend the use of the insertion neighborhood over the 2-opt neighborhood for SRFLP. Moves in the Lin-Kernighan neighborhood are expensive to compute and we make use of the speed-up techniques in [16] in our implementation. We present details of our search over the Lin-Kernighan neighborhood in the next section.

3. Lin-Kernighan neighborhood search for the SRFLP

The Lin-Kernighan neighborhood structure was first used in the context of graph partitioning and then was most widely used to solve the traveling salesman problem. The motivation behind this neighborhood is the fact that using conventional neighborhoods (such as the 2-opt or insertion neighborhood) local search algorithms quickly get trapped in potentially bad quality local optima. In the Lin-Kernighan neighborhood, a move is essentially a composite move constructed as a sequence of basic moves in a conventional neighborhood, even though such a sequence of moves take the search through local optimal in the conventional neighborhood structure. The number of basic moves allowed in the sequence depends on the quality of solutions that are obtained at various stages in the sequence. Hence a Lin-Kernighan neighborhood is a variable depth neighborhood structure. This neighborhood

structure was found to be very effective for the traveling salesman problem, and we use it in this paper in the context of the SRFLP.

Lin and Kernighan [21] control their neighborhood search through a cumulative gain function $G(.)$ defined on the number of successive basic moves in the conventional neighborhood. Each basic move results in choosing the best neighbor of the current solution in the conventional neighborhood. Consider the k-th successive basic move in a composite move. Suppose we have a minimization problem and that the solution at the beginning of the basic move was Π_1 and the move chose the best solution Π_2 in the conventional neighborhood of Π_1. The gain $g(k)$ through such a move is defined as $g(k) = z(\Pi_1) - z(\Pi_2)$ where $z(.)$ is the objective function. The function $G(.)$ is defined recursively as $G(k) = G(k-1) + g(k)$ with $G(0) = 0$. Successive moves are considered until for a pre-specified value k_{max} of k. Note that $G(.)$ is not monotonic in k. Let d be the value of k ($1 \leq k \leq k_{max}$) for which $G(k)$ is the maximum. The composite move is then the sequence of the first d consecutive basic moves. If the value of $G(k)$ is uniformly less than or equal to 0 for any solution, then that solution is said to be locally optimal for the Lin-Kernighan neighborhood structure.

The neighborhood structure described above can be easily adapted for the SRFLP. ([23] suggests two such neighborhoods for a special case of the SRFLP called the linear ordering problem.) There are two conventional neighborhood structures for the SRFLP, the 2-opt neighborhood in which a neighbor is formed by interchanging the positions of two facilities in a permutation, and the insertion neighborhood in which a neighbor is formed by removing a facility from the permutation and re-inserting it at another position in the permutation. [26] and [16] both suggest the use of the insertion neighborhood and we use this neighborhood in our heuristic. However, once a facility has been re-positioned after a basic move during the Lin-Kernighan neighborhood search, we do not consider it for repositioning during later basic moves while searching the same Lin-Kernighan neighborhood. Sources [see, e.g. 27] report that an exhaustive search over a conventional neighborhood is prohibitively expensive, however we use the technique suggested in [16] to search the insertion neighborhood exhaustively in $O(n^3)$ time. Although the technique to speed up the insertion neighborhood search is described in [16], we add the description here for the sake of completeness. For notational convenience, in the remainder of this section we write $c_{\pi_i\pi_j}$ as $c_{i,j}$ and $d_{\pi_i\pi_j}$ as $d_{i,j}$.

The idea behind speeding up the search over the insertion neighborhood is to avoid computing the costs of neighbors directly, but to obtain them indirectly by computing the difference between the cost of the initial solution and the neighbor and subtracting this difference from the cost of the initial solution to obtain the cost of the neighbor. Consider the following pseudocode of an algorithm to output the best insertion neighbor of a given permutation Π for a SRFLP.

ALGORITHM INSERT-NBD-SEARCH

Input: A SRFLP instance of size n, a permutation Π.

Output: An insertion neighbor of Π which has the minimum cost among all of Π's insertion neighbors.

Code

```
1.  begin
2.     set nbr ← UNDEFINED and nbrcost ← ∞;
3.     for p from 1 to n do begin                          (* p-loop *)
4.        for q from 1 to n but not p do begin             (* q-loop *)
5.           generate an insertion neighbor Π_q^p of Π by removing π_p from the p-th
             position in Π and inserting it at the q-th position in Π;
6.           set cost ← cost of Π_q^p;
7.           if (cost < nbrcost) then
8.              set nbr ← Π_q^p and nbrcost ← cost;
9.        end;                                             (* end q-loop *)
10.    end;                                                (* end p-loop *)
11.    output nbr and nbrcost;
12. end.
```

Let $\Pi = (\pi_1, \pi_2, \ldots, \pi_n)$ and let the cost $z(\Pi)$ of the permutation Π be known. This cost needs to be computed only once during the search and can be obtained in $O(n^2)$ time. Let S_L be the permutation of facilities to the left of π_k in Π and S_R be the permutation of facilities to the right of π_k in Π. Both S_L and S_R exclude π_k. A k-contraction of Π is the permutation $\Pi^k = (S_L, S_R)$, i.e., the permutation obtained by removing facility π_k from Π.

The difference $\psi_k = z(\Pi) - z(\Pi^k)$ is given by

$$\psi_k = \sum_{\pi_j \in S_L} c_{kj} d_{kj} + \sum_{\pi_j \in S_R} c_{kj} d_{kj} + l_k \sum_{\pi_i \in S_L} \sum_{\pi_j \in S_R} c_{ij}.$$

If $k = 1$, $S_L = \emptyset$ and

$$\psi_1 = \sum_{\pi_j \in S_R} c_{kj} d_{kj} \tag{1}$$

which can be computed in $O(n)$ time.

Now assume that we know the value of the $\sum_{\pi_i \in S_L} \sum_{\pi_j \in S_R} c_{ij}$ in the expression for ψ_k. This sum is 0 when $k = 1$. Consider the expression for ψ_{k+1}. For notational convenience we

denote $S_L \cup \{\pi_k\}$ as S'_L and $S_R \setminus \pi_{(k+1)}$ as S'_R.

$$\begin{aligned}\psi_{k+1} &= \sum_{\pi_j \in S'_L} c_{(k+1)j} d_{(k+1)j} + \sum_{\pi_j \in S'_R} c_{kj} d_{(k+1)j} + l_{(k+1)} \sum_{\pi_i \in S'_L} \sum_{\pi_j \in S'_R} c_{ij} \\ &= \sum_{\pi_j \in S'_L} c_{(k+1)j} d_{(k+1)j} + \sum_{\pi_j \in S'_R} c_{kj} d_{(k+1)j} \\ &\quad + l_{k+1} \left(\sum_{\pi_i \in S_L} \sum_{\pi_j \in S_R} c_{ij} + \sum_{\pi_j \in S_R} c_{kj} - \sum_{\pi_i \in S_L} c_{i(k+1)} - c_{k(k+1)} \right). \end{aligned} \tag{2}$$

This can be computed in $O(n)$ time since the value of the only term under double summation is known from the computation of ψ_k. This shows that starting from ψ_1, the value of ψ_k can be computed in $O(n)$ time for any value of k.

Next, we consider inserting facilities in k-contractions. Suppose we want to insert facility π_k at the q-th position in a k-contraction Π^k. Let T_L be the permutation of facilities in the first $q-1$ positions of Π^k and T_R be the permutation of facilities from the q-th position in Π^k to the end, so that $\Pi^k = (T_L, T_R)$. Then the increase ϱ_{kq} in the cost of Π^k when facility π_k is inserted at the q-th position in Π^k is

$$\varrho_{kq} = \sum_{\pi_i \in T_L} c_{ik} d_{ik} + \sum_{\pi_j \in T_R} c_{kj} d_{kj} + l_k \sum_{\pi_i \in T_L} \sum_{\pi_j \in T_R} c_{ij}.$$

In particular if $k > 1$ and $q = 1$, implying $T_L = \emptyset$,

$$\varrho_{k1} = \sum_{\pi_j \in T_R} c_{kj} d_{kj}. \tag{3}$$

Also, if $k = 1$ and $q = 2$,

$$\varrho_{12} = c_{1k} d_{1k} + \sum_{\pi_j \in \Pi^k} c_{kj} d_{kj} + l_k \sum_{\pi_j \in T_R} c_{1j}. \tag{4}$$

Note that the ϱ values in both these special cases can be computed in $O(n)$ time.

Assume that we know the value of the $\sum_{\pi_i \in T_L} \sum_{\pi_j \in T_R} c_{ij}$ in the expression for ϱ_{kq}. If $q = 1$, this sum is 0. Consider the expression for $\varrho_{k(q+1)}$. Again for notational convenience

we denote $T_L \cup \{\pi_q\}$ as T'_L and $T_R \setminus \pi_q$ as T'_R.

$$\begin{aligned}\varrho_{k(q+1)} &= \sum_{\pi_i \in T'_L} c_{ik}d_{ik} + \sum_{\pi_j \in T'_R} c_{kj}d_{kj} + l_k \sum_{\pi_i \in T'_L} \sum_{\pi_j \in T'_R} c_{ij} \\ &= \sum_{\pi_i \in T'_L} c_{ik}d_{ik} + \sum_{\pi_j \in T'_R} c_{kj}d_{kj} + l_k \left(\sum_{\pi_i \in T_L} \sum_{\pi_j \in T_R} c_{ij} + \sum_{\pi_j \in T_R} c_{qj} - \sum_{\pi_i \in T_L} c_{iq} \right). \qquad (5)\end{aligned}$$

This can be computed in $O(n)$ time since the only term under double summation is known from the computation of ϱ_{kq}.

Equations (1) through (5) can be used to implement the INSERT-NBD-SEARCH algorithm in $O(n^3)$ time. In the algorithm, we initially compute $z(\Pi)$ in $O(n^2)$ time and store its value. In the first iteration of the p-loop, $p = 1$. In the first iteration of the q-loop corresponding to this p-loop, $q = 2$. The cost of the neighbor Π_2^1 is $z(\Pi_2^1) = z(\Pi) - \psi_1 + \varrho_{12}$ which can be computed in $O(n)$ time using equations (1) and (4). Next consider the $(m+1)$ iteration of the q-loop during this iteration of the p-loop $(1 < m < n)$. The cost of the neighbor $\Pi^1_{(m+1)}$ is $z(\Pi^1_{(m+1)}) = z(\Pi) - \psi^1 + \varrho_{1(m+1)}$ which can be calculated in $O(n)$ time using equations (1) and (5) since the values of the terms that constitute ϱ_{1m} are known. So the first iteration of the p-loop requires $O(n^2)$ time. Now consider the $(r+1)$-th iteration of the p-loop $(1 \leq r < n)$. In the first iteration of the q-loop corresponding to this p-loop, $q = 1$. The cost of the neighbor $\Pi_1^{(r+1)}$ is $z(\Pi_1^{(r+1)}) = z(\Pi) - \psi^{(r+1)} + \varrho_{(r+1)1}$. Since the terms involved in the calculation of ψ^r are known, the value of $\psi^{(r+1)}$ can be calculated in $O(n)$ time using equation (2). Also the value of $\varrho_{(r+1)1}$ can be calculated in $O(n)$ time using equation (3). Hence the cost of Π_1^{r+1} can be computed in $O(n)$ time. Next consider the $(m+1)$ iteration of the q-loop during this iteration of the p-loop $(1 < m < n)$. The cost of the neighbor $\Pi^{(r+1)}_{(m+1)}$ is $z(\Pi^{(r+1)}_{(m+1)}) = z(\Pi) - \psi^{(r+1)} + \varrho_{(r+1)(m+1)}$ which can be calculated in $O(n)$ time using equations (2) and (5) since the values of the terms that constitute $\varrho_{(r+1)m}$ and ψ_r are known. So the q-loop corresponding to the $(r+1)$-th iteration of the p-loop also requires $O(n^2)$ time. Since there are n iterations of the p-loop in the INSERT-NBD-SEARCH algorithm, it requires $O(n^3)$ time to search for the best insertion neighbor.

Now that we have a description of the Lin-Kernighan neighborhood for the SRFLP and the speed up in searching insertion neighborhoods, we are in a position to describe our LK-INSERT heuristic in the next section.

4. The LK-INSERT heuristic for the SRFLP

LK-INSERT is a multi-start heuristic. We generate a population of a user specified number L of starting permutations for our heuristic. The number L is chosen to be large

enough to take the search to sufficiently large number of areas in the space of all permutations. We also prescribe that the initial permutations be of sufficiently low cost. This is because Lin-Kernighan neighborhood search iterations are computationally expensive, and low cost solutions are expected to reduce the number of Lin-Kernighan moves required to a small value.

We generate the starting permutations for LK-INSERT as follows. For the first permutation we use Theorem 1 in [28]. In this theorem the authors characterize optimal solutions for a special case of the SRFLP when all weights are identical. In such cases, the optimal solution is obtained by sorting the facilities in non-decreasing order of their lengths, putting the first facility in the center, and then alternating the placement of the other facilities in sorted order to the left and to the right of the first facility. So if there are n facilities $\pi_1, \pi_2, \ldots, \pi_n$ sorted in non-decreasing order of lengths, the optimal permutation is

$$\begin{cases} (\pi_n, \pi_{(n-2)}, \ldots, \pi_3, \pi_1, \pi_2, \pi_4, \ldots, \pi_{(n-3)}, \pi_{(n-1)}) & \text{when } n \text{ is odd; and} \\ (\pi_{(n-1)}, \pi_{(n-3)}, \ldots, \pi_3, \pi_1, \pi_2, \pi_4, \ldots, \pi_{(n-2)}, \pi_n) & \text{when } n \text{ is even.} \end{cases}$$

We use this permutation of facilities as the first starting permutation for LK-INSERT. Each of the other $L-1$ starting permutations in the population are obtained by switching the locations of facilities at the i-th position and $(n-i)$-th positions in the first starting permutation with probability 0.5 for i values varying between 1 and $n/2$.

The choice of the value of k_{max} is also crucial for LK-INSERT. If it is too large, each Lin-Kernighan iteration takes an unacceptably long time and is not practical. Also, note that a Lin-Kernighan iteration essentially creates a large neighborhood of the initial permutation by combining several conventional neighborhoods, and searches for good neighboring solution along one path in this neighborhood starting from the initial permutation, ignoring good quality permutations that lie outside the path. If the value of k_{max} is too large, then the chance that this search process misses out permutations of good quality becomes high. If the value of k_{max} is too small, then the search process runs the risk of getting trapped in local optima of the conventional neighborhood. So the value of k_{max} for LK-INSERT must be a small number but not too small.

LK-INSERT starts by generating the population of starting permutations, and then for each permutation in the population it performs a local search using the Lin-Kernighan neighborhood to obtain locally optimal permutations. It finally outputs the best locally optimal permutation that it encountered during the search. The pseudocode for this heuristic is given below.

ALGORITHM LK-INSERT

Input: A SRFLP instance of size n, parameters L and k_{max}.

Output: The best neighbor of Π which has the minimum cost among all of Π's neighbors encountered by the algorithm.

Code

1. begin
2. set `nbr` $\leftarrow$ UNDEFINED, `bestnbr` $\leftarrow$ UNDEFINED, and `nbrcost` $\leftarrow \infty$;
3. obtain a permutation Π using Theorem 1 in [27] and store it in the population;
4. use Π to generate the remaining $L-1$ permutations in the population;
5. for i from 1 to L do begin
6. select the i-th permutation Π_i from the population;
7. $G(0) \leftarrow 0$, `best`$G \leftarrow 0$, $d \leftarrow 0$, and `exclude` $\leftarrow \{\}$;
8. for j from 1 to k_{max} do begin
9. find the best insertion neighbor Π_{nbr} of Π_i which does not involve repositioning of facilities in `exclude`;
10. add the facility whose position has changed to yield the best insertion neighbor to `exclude`;
11. $G(j) \leftarrow G(j-1) +$ `cost`$(\Pi) -$ `cost`(Π_{nbr})
12. if $(G(j) >$ `best`$G)$ then
13. set `best`$G \leftarrow G(j)$, $d \leftarrow j$, and `bestnbr` $\leftarrow \Pi_{nbr}$;
14. end;
15. if (`best`$G > 0$) then
16. set $\Pi_i \leftarrow$ `bestnbr` and go to Step 7;
17. if (`nbrcost` $>$ `cost`(Π_i)) then
18. set `nbr` $\leftarrow \Pi_i$ and `nbrcost` $=$ `cost`(Π_i);
19. end;
20. output `nbr` and `nbrcost`;
21. end;

We implemented this heuristic and performed computational experiments with our implementation. We provide the details of our experience in the next section.

5. Computational experience

We coded the LK-INSERT heuristic in C, compiled it with a `gcc`-4.6 compiler, and experimented with it on a machine with Intel Core 2 Duo processor (E4500 2.2 GHz $\times$ 2) running Ubuntu 11.10. In our implementation we restrict the depth of search for the maximum value of the $G(.)$ function to $\lfloor n/15 \rfloor$ and use $\lfloor 2n/3 \rfloor$ starts for a problem of size n, since from our preliminary experiments we found that these choices yield best results.

We ran our heuristic on 40 benchmark instances. The first 20 of these instances are due to Anjos and have been frequently experimented with in the literature, e.g. in [7, 10, 14, 27]. The other 20 are `sko` instances based on benchmark instances for the quadratic assignment problem. In the published literature, these instances have been experimented with in [7] and [14]. Apart from the published literature, [16] present two tabu search implementations TS-2OPT and TS-INSERT which perform well on these problems. We benchmark the performance of our LK-INSERT heuristic against these algorithms. Since the generation of initial solutions is randomized in LK-INSERT, we run the heuristic five times for each instance, and report the cost of the best permutation we obtain.

Table 1 presents the comparison of LK-INSERT with other algorithms for the SRFLP on the 20 Anjos instances. The first column records the name of the instance, and the second column its size. The third column reports the cost of the best solution to the problem instance found in the literature. Each entry in the column has a superscript which is a combination of the letters 's', 'd' and 'h'. The presence of the letter 's' indicates that a permutation with the cost reported has been obtained in [28], while the letters 'd' and 'h' indicate that a permutation with that cost has been obtained in [10] and [14] respectively. Results in [7] are not presented in the table since they have all been superceded in the literature. The fourth column reports the cost of the best solution obtained in [16] for the instance. Here there is a superscript which is a combination of '2' and 'i'. The presence of '2' in the superscript indicates that a permutation with that cost has been obtained by the TS-2OPT algorithm in [16], and the presence of an 'i' indicates that a permutation with that cost has been obtained by the TS-INSERT algorithm. The last column reports the cost of the best permutation obtained by the LK-INSERT heuristic described in this paper. The results show that output from the LK-INSERT heuristic matches the best known permutations for each of the 20 Anjos instances. This includes the five instances in which the results in [16] are better than the best permutations known in the published literature.

Table 2 presents the comparison of the quality of solutions output by the LK-INSERT heuristic for the `sko` instances with those available in the literature. In the published literature results are available in [7] and [14]. However, [16] also reports the costs of the best permutations obtained by the TS-2OPT and TS-INSERT algorithms on these instances. In

Table 1: Costs of the best permutations obtained for the Anjos instances

Instance	Size	Published	K&G(2012)	LK-INSERT
Anjos-60-01	60	1477834.0 ^{sdh}	1477834.0 ^{2i}	1477834.0
Anjos-60-02	60	841776.0 ^{h}	841776.0 ^{i}	841776.0
Anjos-60-03	60	648337.5 ^{sdh}	648337.5 ^{2i}	648337.5
Anjos-60-04	60	398406.0 ^{h}	398406.0 ^{2i}	398406.0
Anjos-60-05	60	318805.0 ^{sdh}	318805.0 ^{2i}	318805.0
Anjos-70-01	70	1528560.0 ^{sdh}	1528537.0 ^{i}	1528537.0
Anjos-70-02	70	1441028.0 ^{sdh}	1441028.0 ^{2i}	1441028.0
Anjos-70-03	70	1518993.5 ^{sdh}	1518993.5 ^{2i}	1518993.5
Anjos-70-04	70	968796.0 ^{d}	968796.0 ^{2i}	968796.0
Anjos-70-05	70	4218002.5 ^{h}	4218002.5 ^{2i}	4218002.5
Anjos-75-01	75	2393456.5 ^{d}	2393456.5 ^{i}	2393456.5
Anjos-75-02	75	4321190.0 ^{sd}	4321190.0 ^{2i}	4321190.0
Anjos-75-03	75	1248537.0 ^{d}	1248423.0 ^{2i}	1248423.0
Anjos-75-04	75	3941845.5 ^{h}	3941816.5 ^{2i}	3941816.5
Anjos-75-05	75	1791408.0 ^{sd}	1791408.0 ^{2i}	1791408.0
Anjos-80-01	80	2069097.5 ^{sd}	2069097.5 ^{2i}	2069097.5
Anjos-80-02	80	1921177.0 ^{sd}	1921136.0 ^{2i}	1921136.0
Anjos-80-03	80	3251368.0 ^{d}	3251368.0 ^{i}	3251368.0
Anjos-80-04	80	3746515.0 ^{sd}	3746515.0 ^{2i}	3746515.0
Anjos-80-05	80	1588901.0 ^{d}	1588885.0 ^{i}	1588885.0

the table, the first two columns report the names of the instances and their sizes. The third column reports the costs of the best layouts obtained in [14]. Here too, the results in [7] do not appear in the table since they have been superseded in [14]. The fourth column reports the costs of the best permutations obtained by the TS-2OPT and TS-INSERT algorithms in [16]. As in Table 1, each entry in this column has a superscript which is a combination of '2' and 'i' indicating which of TS-2OPT and TS-INSERT obtained the permutation reported in this column. The last column reports the cost of the permutation obtained by the LK-INSERT heuristic. We see here that the LK-INSERT heuristic was able to improve on all the layouts obtained for all the `sko` instances except `sko`-64-02, in which the layout reported in [14] is still the best one obtained. In 11 of the 20 instances (i.e., 55% of the `sko` instances), the LS-INSERT heuristic generated better permutations than those available in the literature. The costs of these permutations are shown in boldface in Table 2 and the corresponding permutations are reported in the Appendix to this paper. In two other instances it matched the better of the permutations output by TS-2OPT and TS-INSERT. In the remaining 6 instances, LK-INSERT output permutations whose costs were higher than the better of the permutations output by TS-2OPT and TS-INSERT.

Table 2: Costs of the best permutations obtained for the sko instances

Instance	Size	H&L(2011)	K&G(2012)	LK-INSERT
sko-64-01	64	97194.0	96915.0 [i]	96933.0
sko-64-02	64	634332.5	634563.5 [i]	634338.5
sko-64-03	64	414384.5	414327.5 [i]	**414323.5**
sko-64-04	64	298155.0	297332.0 [i]	**297205.0**
sko-64-05	64	502063.5	501922.5 [2i]	501922.5
sko-72-01	72	139231.0	139179.0 [i]	**139150.0**
sko-72-02	72	715611.0	712011.0 [i]	**712005.0**
sko-72-03	72	1061762.5	1054110.5 [2i]	1054110.5
sko-72-04	72	924019.5	920086.5 [i]	**919635.5**
sko-72-05	72	430288.5	428248.5 [2]	428879.5
sko-81-01	81	207063.0	205145.0 [2]	205166.0
sko-81-02	81	526157.5	521399.5 [i]	**521391.5**
sko-81-03	81	979281.0	970912.0 [i]	**970862.0**
sko-81-04	81	2035569.0	2032143.0 [i]	**2031979.0**
sko-81-05	81	1311166.0	1302833.0 [2]	1303805.0
sko-100-01	100	380562.0	378626.0 [i]	**378614.0**
sko-100-02	100	2084924.5	2076023.5 [2]	2076048.5
sko-100-03	100	16216076.5	16149000.0 [i]	**16148818.0**
sko-100-04	100	3263493.0	3233362.0 [2]	**3232740.0**
sko-100-05	100	1040929.5	1033338.5 [2]	1033345.5

From these results, we conclude that the LK-INSERT heuristic is competitive for solving large sized SRFLP instances.

6. Summary and future research directions

In this paper we deal with a NP-Hard problem called the single row facility layout problem (SRFLP). This is a combinatorial optimization problem in which the objective function is of a quadratic form. Exact algorithms to solve this problem have only been able to report optimal solutions for relatively small sized instances with up to 42 facilities. For larger sized problems, the focus of research is on meta-heuristics.

In this paper, we present a variable depth neighborhood search based heuristic called LK-INSERT to solve this problem. The neighborhood searched bears resemblance to the Lin-Kernighan neighborhood for the traveling salesman problem. While the Lin-Kernighan neighborhood uses the k-opt neighborhood as a basic neighborhood in the case of the traveling salesman problem, in this paper, we use the insertion neighborhood as the basic neighborhood. We prefer the insertion neighborhood, because prior experience [see 16, 26] has shown that it is a better neighborhood to search than the 2-opt neighborhood for the SR-

FLP. We are not aware of any previous meta-heuristics for the SRFLP using Lin-Kernighan neighborhoods.

We compare the performance of LK-INSERT with that of other algorithms known in the literature on 40 benchmark instances, and see that LK-INSERT improves on the best known solutions to 11 of the 40 instances. Thus we conclude that Lin-Kernighan based heuristics like the LK-INSERT heuristic presented in this paper are competitive for large sized SRFLP instances.

The work presented in this paper can be extended in several ways. We suggest two such directions in this section. We have implemented our heuristic using the insertion neighborhood. It is possible that with carefully chosen parameters, a 2-opt neighborhood based Lin-Kernighan heuristic can generate good quality solutions. Further, the heuristic presented in this paper is a local search heuristic which terminates on reaching a local optimum. The heuristic can be extended to create tabu search implementations which can search the solution space more effectively.

Acknowledgements

The authors thank A.M.S. Amaral and P. Hungerländer for sharing the benchmark instances used in this paper.

References

[1] A. Amaral, A.N. Letchford, A polyhedral approach to the single row facility layout problem (2011). Available at http://eprints.lancs.ac.uk/id/eprint/49043.

[2] A.R.S. Amaral, On the exact solution of a facility layout problem, European Journal of Operational Research 173 (2006) 508–518.

[3] A.R.S. Amaral, An Exact Approach to the One-Dimensional Facility Layout Problem, Operations Research 56 (2008) 1026–1033.

[4] A.R.S. Amaral, A new lower bound for the single row facility layout problem, Discrete Applied Mathematics 157 (2009) 183–190.

[5] M. Anjos, a. Kennings, a. Vannelli, A semidefinite optimization approach for the single-row layout problem with unequal dimensions, Discrete Optimization 2 (2005) 113–122.

[6] M.F. Anjos, A. Vannelli, Computing Globally Optimal Solutions for Single-Row Layout Problems Using Semidefinite Programming and Cutting Planes, INFORMS Journal on Computing 20 (2008) 611–617.

[7] M.F. Anjos, G. Yen, Provably near-optimal solutions for very large single-row facility layout problems, Optimization Methods and Software 24 (2009) 805–817.

[8] M. Beghin-Picavet, P. Hansen, Deux problèmes daffectation non linéaires, RAIRO, Recherche Opérationnelle 16 (1982) 263–276.

[9] M. Braglia, Heuristics for single-row layout problems in flexible manufacturing systems, Production Planning & Control 8 (1997) 558–567.

[10] D. Datta, A.R. Amaral, J.R. Figueira, Single row facility layout problem using a permutation-based genetic algorithm, European Journal of Operational Research 213 (2011) 388–394.

[11] S.S. Heragu, A.S. Alfa, Experimental analysis of simulated annealing based algorithms for the layout problem, European Journal of Operational Research 57 (1992) 190–202.

[12] S.S. Heragu, A. Kusiak, Machine Layout Problem in Flexible Manufacturing Systems, Operations Research 36 (1988) 258–268.

[13] S.S. Heragu, A. Kusiak, Efficient models for the facility layout problem, European Journal Of Operational Research 53 (1991) 1–13.

[14] P. Hungerländer, F. Rendl, A computational study for the single-row facility layout problem (Unpublished results, 2011). Available at www.optimization-online.org/DB_FILE/2011/05/3029.pdf.

[15] R. Kothari, D. Ghosh, The single row facility layout problem: State of the art (w.p. no. 2011-12-02) (2011). Ahmedabad, India: IIM Ahmedabad, Production & Quantitative Methods. Available at http://www.iimahd.ernet.in/assets/snippets/workingpaperpdf/7736113342011-12-02.pdf.

[16] R. Kothari, D. Ghosh, Tabu search for the single row facility layout problem using exhaustive 2-opt and insertion neighborhoods (w.p. no. 2012-01-03) (2012). Ahmedabad, India: IIM Ahmedabad, Production & Quantitative Methods. Available at http://www.iimahd.ernet.in/assets/snippets/workingpaperpdf/7520037992012-01-03.pdf.

[17] P. Kouvelis, W.C. Chiang, A simulated annealing procedure for single row layout problems in flexible manufacturing systems, International Journal of Production Research 30 (1992) 717–732.

[18] P. Kouvelis, W.C. Chiang, Optimal and Heuristic Procedures for Row Layout Problems in Automated Manufacturing Systems, Journal of the Operational Research Society 47 (1996) 803–816.

[19] S. Kumar, P. Asokan, S. Kumanan, B. Varma, Scatter search algorithm for single row layout problem in fms, Advances in Production Engineering & Management 3 (2008) 193–204.

[20] M. Laguna, R. Martí, V. Campos, Intensification and diversification with elite tabu search solutions for the linear ordering problem, Computers & OR 26 (1999) 1217–1230.

[21] S. Lin, B.W. Kernighan, An Effective Heuristic Algorithm for the Traveling-Salesman Problem, Operations Research 21 (1973) 498–516.

[22] R.F. Love, J.Y. Wong, On solving a one-dimensional space allocation problem with integer programming, INFOR 14 (1976) 139–144.

[23] R. Martí, G. Reinelt, The Linear Ordering Problem, Springer-Verlag Berlin Heidelberg, 2011.

[24] J.C. Picard, M. Queyranne, On the one-dimensional space allocation problem, Operations Research 29 (1981) 371–391.

[25] K. Ravi Kumar, G.C. Hadejinicola, T.L. Lin, A heuristic procedure for the single-row facility layout problem, European Journal of Operational Research 87 (1995) 65–73.

[26] D. Romero, A. Sánchez-Flores, Methods for the one-dimensional space allocation problem, Computers & Operations Research 17 (1990) 465–473.

[27] H. Samarghandi, K. Eshghi, An efficient tabu algorithm for the single row facility layout problem, European Journal of Operational Research 205 (2010) 98–105.

[28] H. Samarghandi, P. Taabayan, F.F. Jahantigh, A particle swarm optimization for the single row facility layout problem, Computers & Industrial Engineering 58 (2010) 529–534.

[29] D.M. Simmons, One-Dimensional Space Allocation: An Ordering Algorithm, Operations Research 17 (1969) 812–826.

[30] M. Solimanpur, P. Vrat, R. Shanker, An ant algorithm for the single row layout problem in flexible manufacturing systems, Computers & Operations Research 32 (2005) 583–598.

Appendix

We provide details of the permutations for the `sko` instances in which we have improved the best permutation known in the literature. Note that the facilities are numbered from 0 through $n-1$ where n is the problem size.

Instance	Size	Cost	Permutation
sko-64-03	64	414323.5	14 11 8 60 55 40 41 48 12 28 3 51 21 22 15 45 35 50 63 54 20 26 30 2 43 13 57 56 23 52 9 24 62 42 17 46 29 34 16 37 33 44 0 38 4 59 25 27 39 10 53 1 7 32 36 18 31 47 19 6 49 58 61 5
sko-64-04	64	297205.0	14 58 56 52 10 53 31 40 29 7 18 9 32 55 24 16 1 37 49 25 33 36 0 6 44 38 46 43 57 23 34 20 48 3 63 61 4 15 28 54 22 11 12 21 8 27 47 13 39 60 51 17 5 19 45 50 2 62 26 42 35 41 59 30
sko-72-01	72	139150.0	33 2 23 53 44 10 4 24 68 71 61 3 42 65 19 57 18 14 40 47 39 41 67 50 35 56 54 43 70 5 8 15 38 31 62 36 12 49 32 48 45 58 37 6 0 28 60 22 51 64 46 25 27 21 13 1 20 66 16 29 26 69 9 55 7 34 17 59 63 30 52 11
sko-72-02	72	712005.0	11 17 55 13 22 58 12 31 36 62 42 2 52 64 46 51 69 9 27 34 21 59 45 37 1 26 29 6 48 20 30 49 7 50 41 47 28 24 32 38 66 23 54 43 25 56 67 35 60 14 68 71 15 5 53 65 3 18 0 16 70 39 8 19 63 61 10 44 40 57 4 33
sko-72-04	72	919635.5	7 16 58 31 1 20 25 51 46 0 27 26 6 29 9 12 69 70 32 34 13 21 48 22 45 59 37 28 30 54 56 43 39 41 50 47 53 15 66 67 8 38 18 4 17 52 5 24 68 33 57 36 62 61 42 65 3 14 60 64 35 40 19 10 49 44 71 23 63 55 2 11
sko-81-02	81	521391.5	21 54 37 48 24 34 80 13 51 47 55 5 14 60 20 2 68 16 32 65 73 69 40 49 22 66 35 29 64 41 3 10 8 33 25 30 18 43 1 44 11 31 79 28 78 4 57 0 77 50 45 12 67 36 72 58 39 70 74 52 42 23 17 19 38 9 26 62 6 61 53 71 46 15 75 56 63 59 76 27 7
sko-81-03	81	970862.0	29 34 38 73 78 28 54 10 27 71 47 5 4 8 30 14 13 1 60 24 2 69 37 32 11 18 79 80 25 35 65 40 64 49 16 22 41 66 3 55 45 31 70 7 0 72 74 21 44 68 57 51 48 6 42 77 23 61 26 20 56 67 63 17 12 50 36 19 62 58 59 75 39 33 9 53 43 15 52 76 46

Instance	Size	Cost	Permutation
sko-81-04	81	2031979.0	65 73 7 24 25 64 68 22 50 2 13 19 1 15 49 67 44 26 8 18 43 11 69 56 10 16 47 45 59 63 75 33 32 14 71 36 55 17 77 38 41 31 0 3 48 34 35 66 54 53 61 51 39 9 74 46 62 57 29 20 60 12 72 23 76 30 52 5 27 80 37 42 58 6 40 4 28 21 78 70 79
sko-100-01	100	378614.0	2 34 43 28 16 20 6 75 52 44 11 35 49 40 47 60 22 48 1 69 81 91 37 71 80 63 89 65 46 50 45 25 99 68 19 39 42 29 84 72 66 98 93 95 3 7 23 92 67 97 55 88 41 58 18 13 8 83 62 30 76 27 56 31 61 85 51 74 33 53 5 14 21 15 79 9 26 54 12 59 24 10 94 4 87 57 73 86 0 38 78 82 32 70 36 90 96 17 77 64
sko-100-03	100	16148818.0	2 57 27 7 32 59 75 44 48 52 91 39 60 99 46 40 69 68 42 89 65 67 86 85 71 18 37 28 47 93 66 51 9 50 14 55 97 58 41 49 21 25 6 45 22 63 16 31 87 80 33 19 56 76 5 83 78 24 17 0 73 54 92 94 3 35 11 95 98 29 84 72 1 64 61 8 13 15 10 74 82 12 26 62 30 90 81 70 36 4 20 53 79 23 88 38 34 96 77 43
sko-100-04	100	3232740.0	48 41 38 78 70 24 31 4 96 17 92 93 51 67 7 97 82 8 15 87 21 32 42 20 26 74 79 23 59 66 85 27 30 73 18 88 53 14 0 55 95 64 3 90 84 54 12 10 77 62 56 61 36 76 58 80 60 49 91 47 89 99 37 45 25 81 68 52 34 71 65 69 35 50 9 39 19 29 46 72 13 40 86 28 11 16 43 22 63 33 94 1 98 75 57 44 83 5 2 6