



**Tabu search for the single row facility layout problem  
using exhaustive 2-opt and insertion neighborhoods**

Ravi Kothari  
Diptesh Ghosh

**W.P. No. 2012-01-03**  
January 2012

The main objective of the Working Paper series of IIMA is to help faculty members, research staff, and doctoral students to speedily share their research findings with professional colleagues and to test out their research findings at the pre-publication stage.

INDIAN INSTITUTE OF MANAGEMENT  
AHMEDABAD – 380015  
INDIA

# TABU SEARCH FOR THE SINGLE ROW FACILITY LAYOUT PROBLEM USING EXHAUSTIVE 2-OPT AND INSERTION NEIGHBORHOODS

Ravi Kothari  
Diptesh Ghosh

## 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 NP-hard. In this paper, we present two tabu search implementations, one involving an exhaustive search of the 2-opt neighborhood and the other involving an exhaustive search of the insertion neighborhood. We also present techniques to significantly speed up the search of the two neighborhoods. Our computational experiments show that the speed up techniques are effective, and our tabu search implementations are competitive. Our tabu search implementations improved several previously known best solutions for large sized benchmark SRFLP instances.

Keywords: Facilities planning and design; Single Row Facility Layout, Tabu Search

## 1 Introduction

The single row facility layout problem (SRFLP) is the NP-hard problem of arranging a given set of facilities on a line so as to minimize the weighted sum of the distances between all pairs of facilities. The weights for each of the pairs of facilities as well as the lengths of each of the facilities are given, and the distance between a pair of facilities is defined as the distance between the centroids of the facilities. The number of facilities in a particular instance is called the size of the instance. This problem was first proposed in [Simmons \(1969\)](#) and was shown to be NP-Hard in [Beghin-Picavet and Hansen \(1982\)](#). Formally stated the SRFLP is defined as follows:

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

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

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

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  $\pi_i$  and  $\pi_j$  when the facilities in  $F$  are ordered as per the permutation  $\Pi$ .

Note that in contrast with several common combinatorial optimization problems, the complexity of determining the cost of a solution is  $O(n^2)$  in a SRFLP instance of size  $n$ .

The SRFLP has been used to model numerous practical situations. It has been a model for arrangement of rooms in hospitals, departments in office buildings or in supermarkets ([Simmons 1969](#)), arrangement of machines in flexible manufacturing systems ([Heragu and Kusiak 1988](#)), assignment of files to disk cylinders in computer storage, and design of warehouse layouts ([Picard and Queyranne 1981](#)). 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 (Laguna et al. 1999), and ranking of teams in sports (Martí and Reinelt 2011).

Several exact methods have been applied to solve the SRFLP to optimality in the literature. These methods include branch and bound (Simmons 1969), mathematical programming (Love and Wong 1976, Heragu and Kusiak 1991, Amaral 2006; 2008), cutting planes (Amaral 2009), dynamic programming (Picard and Queyranne 1981, Kouvelis and Chiang 1996), branch and cut (Amaral and Letchford 2011), and semidefinite programming (Anjos et al. 2005, Anjos and Vannelli 2008, Anjos and Yen 2009, Hungerländer and Rendl 2011). These methods have been able to obtain optimal solutions to SRFLP instances with up to 42 facilities.

Researchers have focused on heuristics for solving larger sized SRFLP instances. These heuristics are of two types; construction and improvement. Construction heuristics for the SRFLP have been presented in Heragu and Kusiak (1988), Ravi Kumar et al. (1995), and Braglia (1997). However these have later been superceded by improvement heuristics. Most improvement heuristics for the SRFLP are metaheuristics, e.g. simulated annealing (Romero and Sánchez-Flores 1990, Kouvelis and Chiang 1992, Heragu and Alfa 1992), ant colony optimization (Solimanpur et al. 2005), scatter search (Kumar et al. 2008), tabu search (Samarghandi and Eshghi 2010), particle swarm optimization (Samarghandi et al. 2010), and genetic algorithms (Datta et al. 2011). Among these, the tabu search implementation in Samarghandi and Eshghi (2010) and the genetic algorithm implementation in Datta et al. (2011) yield best results for benchmark SRFLP instances of large sizes.

In this paper, we present two tabu search implementations for the SRFLP. Like Samarghandi and Eshghi, our implementations are parallel multi-start tabu search implementations, which are different from the way tabu search is generally implemented to solve combinatorial optimization problems. Our implementations differ significantly from the one presented in Samarghandi and Eshghi (2010), both in implementation details of tabu search and the methods of searching neighborhoods. Samarghandi and Eshghi remark in their paper that examining the complete 2-opt neighborhood of a permutation “can be very time consuming or even impossible” (see p.101 in Samarghandi and Eshghi 2010). They therefore sample the 2-opt neighborhood of permutations to obtain the 2-opt neighbors. In our tabu search implementations we use techniques to speed up the neighborhood search significantly which allows us to search the neighborhoods exhaustively. Also since Samarghandi and Eshghi use the 2-opt neighborhood in their implementation and Romero and Sánchez-Flores recommend the use of insertion neighborhoods in their experiments with simulated annealing, in this paper, one of our tabu search implementations searches the 2-opt neighborhood, and the other searches the insertion neighborhood.

Our paper is organized as follows. In Section 2 we present techniques to speed up searching the 2-opt and insertion neighborhoods of a solution to a SRFLP instance. We then describe our tabu search implementations using these speed up techniques in Section 3 and present results of our computational experiments in Section 4. We conclude the paper in Section 5 with a summary of the work.

## 2 Speeding up neighborhood search

The 2-opt and insertion neighborhood structures have been used in local search based approaches to solve the SRFLP. In the 2-opt neighborhood, a neighbor of a permutation is obtained by swapping the positions of exactly two of the facilities. In an insertion neighborhood, a neighbor of a permutation is obtained by removing a facility from its position in the permutation and re-inserting it at another position in the permutation. Clearly, for a SRFLP instance of size  $n$ , there are  $O(n^2)$  neighbors of each permutation in both neighborhoods, and since computing the cost of a permutation requires  $O(n^2)$  time, a naïve implementation of either of the neighborhoods requires  $O(n^4)$  time to search

the neighborhood exhaustively for the best neighbor. This makes exhaustive neighborhood search for large SRFLP instances a very slow process. In this section, we reduce this search time to  $O(n^3)$  for both the neighborhood structures.

## 2.1 2-opt neighborhoods

The pseudocode for a program to search the 2-opt neighborhood of a given permutation is given below.

### ALGORITHM 2-OPT-NBD-SEARCH

**Input:** A SRFLP instance of size  $n$ , a permutation  $\Pi$ .

**Output:** A 2-opt neighbor of  $\Pi$  which has the minimum cost among all of  $\Pi$ 's 2-opt neighbors.

#### Code

```

1. begin
2.   set nbr ← UNDEFINED; nbrcost ← ∞;
3.   for p from 1 to n − 1 do begin
4.     for q from p + 1 to n do begin
5.       generate 2-opt neighbor Π' of Π by interchanging the facilities in
6.         positions p and q in Π;
7.         set cost ← cost of Π';
8.         if (cost < nbrcost) then begin
9.           set nbr ← Π';
10.          set nbrcost ← cost;
11.        end;
12.      end;
13.    output nbr and nbrcost;
14.  end.
```

Note that in the first iteration of the  $q$ -loop for any iteration of the  $p$ -loop, the facilities that need to be interchanged are adjacent. Also note that in successive iterations of the  $q$ -loop inside any  $p$ -loop, the position  $p$  remains fixed, and the position  $q$  shifts one place to the right at a time. We will use these observations to present book-keeping techniques that reduce the complexity of searching for the best 2-opt neighbor of a permutation  $\Pi$  to  $O(n^3)$  time.

Consider a permutation  $\Pi = (\pi_1, \pi_2, \dots, \pi_n)$  of facilities, in which  $\pi_i$  denotes the facility at the  $i$ -th position in  $\Pi$ . Consider two positions  $p$  and  $q$  between 1 and  $n$  with  $p < q$ . Let  $S_1$  be the permutation of facilities to the left of  $\pi_p$  in  $\Pi$ ,  $S_2$  be the permutation of facilities between  $\pi_p$  and  $\pi_q$  (both excluded) in  $\Pi$ , and  $S_3$  be the permutation of facilities to the right of  $\pi_q$  in  $\Pi$ . For notational convenience we use  $c_{pq}$  and  $d_{pq}$  to represent  $c_{\pi_p \pi_q}$  and  $d_{\pi_p \pi_q}$  throughout the paper. We also use the notation  $i \in S_j$  to mean  $\pi_i \in S_j$ . Then the cost  $z(\Pi)$  of the permutation  $\Pi$  can be written as

$$\begin{aligned}
z(\Pi) = & \sum_{i \in S_1} \sum_{j \in S_1} c_{ij} d_{ij} + \sum_{i \in S_1} c_{ip} d_{ip} + \sum_{i \in S_1} \sum_{j \in S_2} c_{ij} d_{ij} + \sum_{i \in S_1} c_{iq} d_{iq} + \sum_{i \in S_1} \sum_{j \in S_3} c_{ij} d_{ij} \\
& + \sum_{j \in S_2} c_{pj} d_{pj} + c_{pq} d_{pq} + \sum_{j \in S_3} c_{pj} d_{pj} + \sum_{i \in S_2} \sum_{j \in S_2} c_{ij} d_{ij} \sum_{i \in S_2} c_{iq} d_{iq} \\
& + \sum_{i \in S_2} \sum_{j \in S_3} c_{ij} d_{ij} + \sum_{j \in S_3} c_{qj} d_{qj} + \sum_{i \in S_3} \sum_{j \in S_3} c_{ij} d_{ij}, \tag{1}
\end{aligned}$$

where  $d_{ij}$  denotes the distance between the centroids of facilities  $\pi_i$  and  $\pi_j$  in  $\Pi$ .

If facilities at locations  $p$  and  $q$  are interchanged, leading to a 2-opt neighbor  $\Pi'$  of  $\Pi$ , then the positions of all facilities in  $S_1$  and  $S_3$  remain unchanged, while the positions of all facilities in  $S_2$

uniformly shift by a distance  $|l_p - l_q|$ . Then denoting the sum of the lengths of all facilities between  $\pi_i$  and  $\pi_j$  in  $\Pi$  as  $b_{ij}$  the difference  $\Delta_{pq}$  in costs of  $\Pi$  and  $\Pi'$  can be written as

$$\begin{aligned}\Delta_{pq} &= z(\Pi) - z(\Pi') \\ &= \sum_{i \in S_1} \sum_{j \in S_2} c_{ij}(l_p - l_q) + \sum_{i \in S_1} c_{iq}(L_2 + l_p) + \sum_{j \in S_2} c_{pj}(b_{pj} - b_{qj}) + \sum_{j \in S_3} c_{pj}(L_2 + l_q) \\ &\quad + \sum_{j \in S_2} c_{iq}(b_{iq} - b_{ip}) + \sum_{i \in S_2} \sum_{j \in S_3} c_{ij}(l_q - l_p) - \sum_{i \in S_1} c_{ip}(L_2 + l_q) - \sum_{j \in S_3} c_{qj}(L_2 + l_p).\end{aligned}\quad (2)$$

Since  $c_{ij} = c_{ji}$  and  $b_{ij} = b_{ji}$  for every pair  $i$  and  $j$  of facilities, the expression above can be rewritten as

$$\begin{aligned}\Delta_{pq} &= L_2 \left\{ \sum_{i \in S_1} (c_{iq} - c_{ip}) + \sum_{i \in S_3} (c_{ip} - c_{iq}) \right\} + l_p \left\{ \sum_{i \in S_1} c_{iq} - \sum_{i \in S_3} c_{iq} \right\} \\ &\quad + l_q \left\{ \sum_{j \in S_3} c_{pj} - \sum_{j \in S_1} c_{pj} \right\} + (l_q - l_p) \left\{ \sum_{i \in S_2} \left( \sum_{j \in S_3} c_{ij} - \sum_{j \in S_1} c_{ij} \right) \right\} \\ &\quad + \sum_{j \in S_2} (b_{pj} - b_{qj})(c_{pj} - c_{qj}),\end{aligned}\quad (3)$$

where  $L_2 = \sum_{\pi_j \in S_2} l_j$ .

Note that if  $p$  and  $q$  are adjacent,  $S_2 = \emptyset$  and the difference  $\Delta_{pq}$  is given by

$$\Delta_{pq} = l_p \left\{ \sum_{i \in S_1} c_{iq} - \sum_{i \in S_3} c_{iq} \right\} + l_q \left\{ \sum_{j \in S_3} c_{pj} - \sum_{j \in S_1} c_{pj} \right\},\quad (4)$$

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

Next, let  $\pi_r$  be the facility immediately to the right of  $\pi_q$  in  $\Pi$ . We now show that  $\Delta_{pr}$  can be computed in  $O(n)$  time. If facilities  $\pi_p$  and  $\pi_r$  are to be interchanged to yield a 2-opt neighbor  $\Pi''$  of  $\Pi$ , then the permutation of facilities  $S_1''$  to the left of  $\pi_p$  is identical to  $S_1$ , the permutation  $S_2''$  of facilities between  $\pi_p$  and  $\pi_r$  is  $S_2$  with facility  $\pi_q$  appended, and the permutation of facilities  $S_3''$  to the right of  $\pi_r$  is the permutation  $S_3$  with facility  $\pi_r$  removed from the extreme left. The sum  $L_2''$  of lengths of facilities in  $S_2''$  is given by  $L_2 + l_q$ . The expression for  $\Delta_{pr} = z(\Pi) - z(\Pi'')$  (with a form similar to equation (2)) can be rearranged to yield

$$\begin{aligned}\Delta_{pr} &= l_r \left\{ \sum_{j \in S_3} c_{pj} - c_{pr} - \sum_{j \in S_1} c_{pj} \right\} + (l_r - l_p) \sum_{i \in S_2} \left( \sum_{j \in S_3} c_{ij} - \sum_{j \in S_1} c_{ij} \right) \\ &\quad + L_2'' \left\{ \sum_{i \in S_1} c_{ir} - \sum_{i \in S_3''} c_{ir} + \sum_{i \in S_3} c_{ip} - c_{rp} - \sum_{i \in S_1} c_{ip} \right\} \\ &\quad + l_p \left\{ \sum_{i \in S_1} c_{ir} - \sum_{i \in S_3''} c_{ir} \right\} + (l_r - l_p) \left\{ \sum_{j \in S_3} c_{qj} - \sum_{j \in S_1} c_{qj} - \sum_{i \in S_2''} c_{ir} \right\} \\ &\quad + \sum_{j \in S_2''} (b_{pj} - b_{rj})(c_{pj} - c_{rj}).\end{aligned}\quad (5)$$

Note that since the values of  $\sum_{j \in S_1} c_{pj}$ ,  $\sum_{j \in S_3} c_{pj}$ , and  $\sum_{i \in S_2} (\sum_{j \in S_3} c_{ij} - \sum_{j \in S_1} c_{ij})$  are known from the computation of the value of  $\Delta_{pq}$ , the values of  $l_r \left\{ \sum_{j \in S_3} c_{pj} - c_{pr} - \sum_{j \in S_1} c_{pj} \right\} + (l_r - l_p) \sum_{i \in S_2} (\sum_{j \in S_3} c_{ij} - \sum_{j \in S_1} c_{ij})$  can be computed in constant time and the remaining part of the right hand side of equation (5) can be computed in  $O(n)$  time. Hence we can compute the value of  $\Delta_{pr}$  in  $O(n)$  time.

Let  $\Pi'$  be a 2-opt neighbor of  $\Pi$  obtained by interchanging the positions of facilities  $\pi_p$  and  $\pi_q$  in  $\Pi$ . Since  $z(\Pi') = z(\Pi) - \Delta_{pq}$ , we use the expressions for  $\Delta_{pq}$  to compute the cost of the 2-opt neighbor  $\Pi'$ . Note that since right hand side of equation (4) can be computed in  $O(n)$  time, the cost of the first 2-opt neighbor generated in any  $q$ -loop in 2-OPT-NBD-SEARCH can be computed in linear time. Again since the right hand side of equation (5) can be computed in  $O(n)$  time, the costs of each of the neighbors, after the first one, generated within a  $q$ -loop can be computed in linear time. Hence the  $q$ -loop in 2-OPT-NBD-SEARCH (steps 4 through 11) requires  $O(n^2)$  time, and so 2-OPT-NBD-SEARCH runs in  $O(n^3)$  time when the above presented techniques are used. Hence searching the 2-opt neighborhood of a permutation to obtain the best 2-opt neighbor requires  $O(n^3)$  time.

## 2.2 Insertion neighborhoods

The pseudocode for a program to search the insertion neighborhood of a given permutation is given below.

### ALGORITHM INSERT-NBD-SEARCH

**Input:** A SRFLP instance of size  $n$ , a permutation  $\Pi$ .

**Output:** An insertion neighbor of  $\Pi$  which has the minimum cost among all of  $\Pi$ 's insertion neighbors.

#### Code

```

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

As in the case of 2-opt neighborhood search we will use book-keeping techniques to reduce the complexity of searching for the best insertion neighbor of a permutation to  $O(n^3)$  time.

Consider a permutation  $\Pi = (\pi_1, \pi_2, \dots, \pi_n)$ . We define a  $p$ -contraction of  $\Pi$  as a permutation  $\Pi^p = (\pi_1, \pi_2, \dots, \pi_{p-1}, \pi_{p+1}, \dots, \pi_n)$  of facilities in  $F \setminus \{\pi_p\}$ . If  $\pi_p$  is inserted at the  $q$ -th position in  $\Pi$ , we have an insertion neighbor  $\Pi_q^p$  of  $\Pi$ .

Let  $S_L$  be the permutation of facilities to the left of  $\pi_p$  in  $\Pi$  and  $S_R$  be the permutation of facilities to the right of  $\pi_p$  in  $\Pi$ . Both  $S_L$  and  $S_R$  exclude  $\pi_p$ . If  $z(\Pi)$  and  $z(\Pi^p)$  are the costs of permutations  $\Pi$  and  $\Pi^p$  respectively then the value  $\psi_p = z(\Pi) - z(\Pi^p)$  is

$$\psi_p = \sum_{j \in S_L} c_{pj} d_{pj} + \sum_{j \in S_R} c_{pj} d_{pj} + l_p \sum_{i \in S_L} \sum_{j \in S_R} c_{ij}.$$

In particular if  $p = 1$ ,  $\psi_1 = \sum_{j \in S_R} c_{1j} d_{1j}$ .

Let  $p < q$ , thus  $\Pi^p = (\pi_1, \pi_2, \dots, \pi_{p-1}, \pi_{p+1}, \dots, \pi_q, \pi_{q+1}, \dots, \pi_n)$ , and suppose that we insert the facility  $\pi_p$  at the  $q$ -th position of  $\Pi^p$  to obtain a permutation  $\Pi_q^p$ . The new permutation  $\Pi_q^p$  is

$(\pi_1, \pi_2, \dots, \pi_{p-1}, \pi_{p+1}, \dots, \pi_q, \pi_p, \pi_{q+1}, \dots, \pi_n)$ . Let  $T_L$  be the permutation of facilities to the left of  $\pi_p$  in  $\Pi_q^p$ , and  $T_R$  be the permutation of facilities to the right of  $\pi_p$  in the permutation  $\Pi_q^p$ .

The increase  $\varrho_q^p$  in cost when  $\pi_p$  is inserted in the  $q$ -th position of  $\Pi^p$  is given by

$$\varrho_q^p = \sum_{i \in T_L} c_{ip} d_{ip} + \sum_{j \in T_R} c_{pj} d_{pj} + l_p \sum_{i \in T_L} \sum_{j \in T_R} c_{ij}.$$

If  $q = 1$ , then  $T_L = \emptyset$ , and  $\varrho_1^p = \sum_{j \in T_R} c_{pj} d_{pj}$ .

So if an insertion neighbor  $\Pi_q^p$  of a permutation  $\Pi$  is obtained by removing the facility  $\pi_p$  from  $\Pi$  and inserting it at the  $q$ -th position in the contracted permutation formed by the removal of the facility  $\pi_p$ , the reduction in cost  $\Delta_{pq} = z(\Pi) - z(\Pi_q^p)$  is

$$\begin{aligned} \Delta_{pq} &= \psi_p - \varrho_q^p \\ &= \sum_{j \in S_L} c_{pj} d_{pj} + \sum_{j \in S_R} c_{pj} d_{pj} + l_p \sum_{i \in S_L} \sum_{j \in S_R} c_{ij} - \\ &\quad \sum_{i \in T_L} c_{ip} d_{ip} - \sum_{j \in T_R} c_{pj} d_{pj} - l_p \sum_{i \in T_L} \sum_{j \in T_R} c_{ij}. \end{aligned} \quad (6)$$

Of special interest are the cases when  $p = 1, q = 2$  and when  $p = 2, q = 1$ . If  $p = 1$  and  $q = 2$ , the equation (6) reduces to

$$\Delta_{12} = \sum_{j \in S_R} c_{1j} d_{1j} - \sum_{i \in T_L} c_{i1} d_{i1} - \sum_{j \in T_R} c_{1j} d_{1j} - l_1 \sum_{i \in T_L} \sum_{j \in T_R} c_{ij},$$

Since  $T_L$  is a singleton in this case, the above expression can be simplified to

$$\Delta_{12} = \sum_{j \in S_R} c_{1j} d_{1j} - c_{21} d_{21} - \sum_{j \in T_R} c_{1j} d_{1j} - l_1 \sum_{j \in T_R} c_{1j}, \quad (7)$$

which can be computed in  $O(n)$  time. Now, if  $p = 2$  and  $q = 1$  equation (6) reduces to

$$\Delta_{21} = \sum_{j \in S_L} c_{pj} d_{pj} + \sum_{j \in S_R} c_{pj} d_{pj} + l_2 \sum_{i \in S_L} \sum_{j \in S_R} c_{ij} - \sum_{j \in T_R} c_{pj} d_{pj}.$$

Since  $S_L$  is a singleton in this case, the above expression can be simplified to

$$\Delta_{21} = c_{21} d_{21} + \sum_{j \in S_R} c_{pj} d_{pj} + l_2 \sum_{j \in S_R} c_{1j} - \sum_{j \in T_R} c_{pj} d_{pj}. \quad (8)$$

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

Next, let  $\pi_{q+1}$  be the facility immediately to the right of facility  $\pi_q$  in the permutation  $\Pi$ . We now show that if the components of  $\Delta_{pq}$  are known then  $\Delta_{p(q+1)}$  can be computed in  $O(n)$  time.

If the facility  $\pi_p$  is removed from the  $p$ -th position in the permutation  $\Pi$  and inserted at the  $(q+1)$ -th position in  $\Pi$  to obtain an insertion neighbor  $\Pi_{q+1}^p$  of  $\pi$ , then the permutation of facilities  $S'_L$  to the left of  $\pi_p$  in  $\Pi$  is identical to  $S_L$ , the permutation of facilities  $S'_R$  to the right of  $\pi_p$  in  $\Pi$  is identical to  $S_R$ , the permutation  $T'_L$  of facilities to the left of  $\pi_p$  in  $\Pi_{q+1}^p$  is  $T_L$  appended with the facility  $\pi_{(q+1)}$  and, the permutation  $T'_R$  of facilities to the right of  $\pi_p$  in  $\Pi_{q+1}^p$  is  $T_R$  with facility  $\pi_{(q+1)}$  removed from the extreme left. The expression for  $\Delta_{p(q+1)} = z(\Pi) - z(\Pi_{q+1}^p)$  (with a form similar to equation(6)) can be written as

$$\begin{aligned} \Delta_{p(q+1)} &= \psi_p - \varrho_{q+1}^p \\ &= \sum_{j \in S'_L} c_{pj} d_{pj} + \sum_{j \in S'_R} c_{pj} d_{pj} + l_p \sum_{i \in S'_L} \sum_{j \in S'_R} c_{ij} - \sum_{i \in T'_L} c_{ip} d_{ip} - \sum_{j \in T'_R} c_{pj} d_{pj} - l_p \sum_{i \in T'_L} \sum_{j \in T'_R} c_{ij} \end{aligned}$$

which can be further simplified to

$$\begin{aligned} \Delta_{p(q+1)} = & \sum_{j \in S_L} c_{pj} d_{pj} + \sum_{j \in S_R} c_{pj} d_{pj} + l_p \sum_{i \in S_L} \sum_{j \in S_R} c_{ij} - \sum_{i \in T_L} c_{ip} d_{ip} - \sum_{j \in T_R} c_{pj} d_{pj} - l_p \sum_{i \in T_L} \sum_{j \in T_R} c_{ij} \\ & - l_p \sum_{j \in T_R} c_{(q+1)j} + l_p \sum_{i \in T_L} c_{i(q+1)} - c_{(q+1)p} d_{(q+1)p} + c_{p(q+1)} d_{p(q+1)} \end{aligned} \quad (9)$$

Now, given the values of various components that yield the value of  $\Delta_{pq}$ , the value of  $\sum_{j \in S_L} c_{pj} d_{pj} + \sum_{j \in S_R} c_{pj} d_{pj} + l_p \sum_{i \in S_L} \sum_{j \in S_R} c_{ij} - \sum_{i \in T_L} c_{ip} d_{ip} - \sum_{j \in T_R} c_{pj} d_{pj} - l_p \sum_{i \in T_L} \sum_{j \in T_R} c_{ij}$  can be computed in constant time and the remaining expressions in right hand side of equation (9) can be computed in  $O(n)$  time. Thus we can compute the value of  $\Delta_{p(q+1)}$  in  $O(n)$  time.

If  $p > q$ , the expression for  $\Delta_{pq}$  remains the same as in equation (6). While calculating the value of  $\Delta_{p(q+1)}$  we observe that the permutation of facilities  $S'_L$  to the left of  $\pi_p$  in  $\Pi$  is identical to  $S_L$ , the permutation of facilities  $S'_R$  to the right of  $\pi_p$  in  $\Pi$  is identical to  $S_R$ , the permutation  $T'_L$  of facilities to the left of  $\pi_p$  in  $\Pi_{q+1}^p$  is  $T_L$  appended with the facility  $\pi_q$  and, the permutation  $T'_R$  of facilities to the right of  $\pi_p$  in  $\Pi_{q+1}^p$  is  $T_R$  with facility  $\pi_q$  removed from the extreme left. Thus the component expressions of  $\Delta_{pq}$  can be used in a similar way as explained earlier to compute  $\Delta_{p(q+1)}$  in  $O(n)$  time.

Now, let  $\pi_{p+1}$  be the facility immediately to the right of facility  $\pi_p$  in the permutation  $\Pi$  and we want to compute the value of  $\Delta_{(p+1)q}$ . Let there be at least one facility between the facilities  $\pi_p$  and  $\pi_q$  in  $\Pi$ . (If there is no such facility, then the expression of  $\Delta_{(p+1)q}$  is meaningless.) We show that if the components of  $\Delta_{pq}$  are known then  $\Delta_{(p+1)q}$  can be computed in  $O(n)$  time.

If the facility  $\pi_{(p+1)}$  is removed from the  $(p+1)$ -th position in the permutation  $\Pi$  and inserted at the  $q$ -th position in  $\Pi^{(p+1)}$  to obtain an insertion neighbor  $\Pi_q^{(p+1)}$  of  $\pi$ , then the permutation of facilities  $S'_L$  to the left of  $\pi_{(p+1)}$  in  $\Pi$  is  $S_L$  appended with the facility  $\pi_p$ , the permutation of facilities  $S'_R$  to the right of  $\pi_{(p+1)}$  in  $\Pi$  is  $S_R$  with facility  $\pi_{(p+1)}$  removed from the extreme left, the permutation  $T'_L$  of facilities to the left of  $\pi_{(p+1)}$  in  $\Pi_q^{(p+1)}$  is the permutation  $T_L$  with the facility  $\pi_{(p+1)}$  replaced by the facility  $\pi_p$  at the same position in the permutation and, the permutation  $T'_R$  of facilities to the right of  $\pi_{(p+1)}$  in  $\Pi_q^{(p+1)}$  is identical to  $T_R$ . The expression for  $\Delta_{(p+1)q} = z(\Pi) - z(\Pi_q^{(p+1)})$  (with a form similar to equation(6)) can be written as

$$\begin{aligned} \Delta_{(p+1)q} = & \psi_{p+1} - \varrho_q^{p+1} \\ = & \sum_{j \in S'_L} c_{(p+1)j} d_{(p+1)j} + \sum_{j \in S'_R} c_{(p+1)j} d_{(p+1)j} + l_{(p+1)} \sum_{i \in S'_L} \sum_{j \in S'_R} c_{ij} \\ & - \sum_{i \in T'_L} c_{i(p+1)} d_{i(p+1)} - \sum_{j \in T'_R} c_{(p+1)j} d_{(p+1)j} - l_{(p+1)} \sum_{i \in T'_L} \sum_{j \in T'_R} c_{ij} \end{aligned}$$

which can be further simplified to

$$\begin{aligned} \Delta_{(p+1)q} = & \sum_{j \in S'_L} c_{(p+1)j} d_{(p+1)j} + \sum_{j \in S'_R} c_{(p+1)j} d_{(p+1)j} - \sum_{i \in T'_L} c_{i(p+1)} d_{i(p+1)} - \sum_{j \in T'_R} c_{(p+1)j} d_{(p+1)j} \\ & + l_{(p+1)} \left\{ \sum_{j \in S_R} c_{pj} - \sum_{i \in S_L} c_{i(p+1)} - c_{p(p+1)} \right\} - l_{(p+1)} \left\{ \sum_{j \in T_R} c_{pj} - \sum_{j \in T_R} c_{(p+1)j} \right\} \\ & + l_{(p+1)} \sum_{i \in S_L} \sum_{j \in S_R} c_{ij} - l_{(p+1)} \sum_{i \in T_L} \sum_{j \in T_R} c_{ij} \end{aligned} \quad (10)$$

Since the components of  $\Delta_{pq}$  are known, the value of  $\sum_{i \in S_L} \sum_{j \in S_R} c_{ij}$  and  $\sum_{i \in T_L} \sum_{j \in T_R} c_{ij}$  can be computed in constant time and the remaining expressions on the right hand side of equation (10) can be computed in  $O(n)$  time. Hence we can compute the value of  $\Delta_{(p+1)q}$  in  $O(n)$  time.



If  $p > q$  and the component expressions of  $\Delta_{pq}$  are known then in calculating the value of  $\Delta_{(p+1)q}$  it should be noted that the permutation of facilities  $S'_L$  to the left of  $\pi_{(p+1)}$  in  $\Pi$  is  $S_L$  appended with the facility  $\pi_p$ , the permutation of facilities  $S'_R$  to the right of  $\pi_{(p+1)}$  in  $\Pi$  is  $S_R$  with facility  $\pi_{(p+1)}$  removed from the extreme left, the permutation  $T'_R$  of facilities to the right of  $\pi_{(p+1)}$  in  $\Pi_q^{(p+1)}$  is the permutation  $T_R$  with the facility  $\pi_{(p+1)}$  replaced by the facility  $\pi_p$  at the same position in the permutation and, the permutation  $T'_L$  of facilities to the left of  $\pi_{(p+1)}$  in  $\Pi_q^{(p+1)}$  is identical to  $T_L$ . Using these observations and the techniques presented earlier the value of  $\Delta_{(p+1)q}$  can be computed in  $O(n)$  time.

Let  $\Pi_q^p$  be an insertion neighbor of  $\Pi$  obtained by removing  $\pi_p$  from the  $p$ -th position in  $\Pi$  and inserting at the  $q$ -th position in  $\Pi^p$ . Since  $z(\Pi_q^p) = z(\Pi) - \Delta_{pq}$ , we use the expressions for  $\Delta_{pq}$  to compute the cost of the insertion neighbor  $\Pi_q^p$ . Note that the values of  $\Delta_{12}$  and  $\Delta_{21}$  can be computed in  $O(n)$  time using equations (7) and (8), and then by repeated applications of equations (9) and (10), all the values of  $\Delta_{pq}$  can be computed in  $O(n)$  time for any value of  $p$  and  $q$ . Hence computing the cost of any insertion neighbor of  $\Pi$  requires  $O(n)$  time, so that the search of the insertion neighborhood of  $\Pi$  for the best insertion neighbor requires  $O(n^3)$  time using our techniques presented above.

In order to test the performance of the neighborhood search processes described above, we implemented each of 2-OPT-NBD-SEARCH and INSERT-NBD-SEARCH once using the naïve approach and once using the techniques presented in this section. Table 1 shows the time required by the implementations to perform neighborhood searches on problem instances with sizes varying from 60 to 160. The first column in the table describes the neighborhood structures that were used in these experiments. The second column specifies the sizes of the problems considered. The third and fourth columns report the times required by naïve implementations and implementations using our speed up techniques to search the neighborhoods of 100 permutations of different problem sizes. The last column reports the speed ups achieved by using our techniques. The speed up is calculated as the ratio of the difference in time required by the naïve implementation and the implementation using our speed up techniques to the time required by the naïve implementation and is expressed as a percentage. The table clearly demonstrates the effectiveness of the speed-up techniques presented

Table 1: CPU times (in seconds) required to perform 100 neighborhood searches

Neighborhood	Size	Naïve	Enhanced	Speed up
2-Opt	60	2.8	0.2	92.9%
	110	28.2	1.1	95.9%
	160	129.6	3.5	97.3%
Insertion	60	5.7	0.6	89.5%
	110	56.6	3.6	93.6%
	160	259.7	11.2	95.7%

in this section. It also shows that the speed ups become more effective as problem sizes increase. In the next section, we embed the neighborhood search techniques developed in this section with tabu search algorithm.

### 3 Exhaustive neighborhood search based tabu search

A tabu search implementation to solve SRFLP instances has been described in [Samarghandi and Eshghi \(2010\)](#). The implementation is impressive in that it achieves low cost permutations within very short execution times. The broad structure of our tabu search implementations is similar to

the implementation in [Samarghandi and Eshghi \(2010\)](#). Our implementations however differ from [Samarghandi and Eshghi](#)'s implementation in the details.

[Samarghandi and Eshghi](#)'s tabu search implementation (called S&E from now onwards) is itself different from conventional tabu search implementations for solving combinatorial problems and adopts a population approach. It requires four parameters, the cardinality  $L$  of an adaptive memory list, the number  $\alpha$  which limits the number of futile attempts in the neighborhood search of a particular permutation in a tabu search iteration, the tabu tenure  $\theta$ , and the maximum number of iterations  $k$  that the algorithm executes.

S&E starts by generating  $L$  permutations and storing them in an adaptive memory (AM) list sorted in non-decreasing order of their costs, and creating an empty tabu list. It then performs  $k$  tabu search iterations. Each S&E iteration starts by picking a solution at random from the AM list. The probability of choosing a particular solution in the list depends on its position in the list, with a solution higher in the list having a larger probability of it being chosen. The algorithm then examines randomly generated 2-opt neighbors of the permutation till there are  $\alpha$  futile attempts in probing the neighborhood. A neighbor is marked tabu if the pair of facilities exchanged to create that neighbor are present in the tabu list, and is marked non-tabu otherwise. S&E then chooses the best non-tabu neighbor of the permutation, except when the best tabu neighbor is better than any solution that the algorithm has encountered thus far, or is the only solution that has a cost lower than that of the permutation chosen, in which case the best tabu neighbor is chosen, and added to the AM list. The pair of neighbors whose positions were interchanged to obtain the neighbor chosen by the algorithm are added to the tabu list, to remain there for the next  $\theta$  iterations of the algorithm. The AM list is then updated by removing the worst solution from the list and re-sorting the AM list. This completes one iteration of the tabu search procedure.

Once S&E performs  $k$  iterations, it chooses the best solution in the AM list and subjects it to a final intensification process. This intensification process is a restricted 2-opt local search on the chosen permutation in which the facilities to be interchanged must be adjacent. The best permutation obtained at the end of this intensification process is output by the algorithm. S&E thus has four components; adaptive memory (AM) list creation, tabu search iteration, AM list and tabu list update, and final intensification. We next describe the details of the differences between S&E and our implementations in each of these four components.

**AM list creation:** S&E generates the AM list as follows. It uses Theorem 1 in [Samarghandi and Eshghi \(2010\)](#) to generate the first permutation in the list. The  $(j + 1)$ -th permutation in the list is created using the first permutation by interchanging the positions of the two facilities located  $j$  positions to the left and the right of the middle-most facility in the first permutation.

For our implementations we experimented with two additional methods, METHOD I and METHOD II, of generating the original permutations in the AM list. In METHOD I, we generate the first permutation in the list using Theorem 1 in [Samarghandi and Eshghi \(2010\)](#). The other permutations are generated as follows. We copy the first permutation as a template for the permutation. Then for each  $i$  between 1 and  $n/2$ , with probability of 0.5 we interchange the facility located at the  $i$ -th position with the facility located in the  $(n - i)$ -th position. In METHOD II, we again generate the first permutation in the list using Theorem 1 in [Samarghandi and Eshghi \(2010\)](#). The other permutations are generated as follows. We copy the first permutation as a template for the permutation. We then generate two random integers  $r_1$  between 1 and  $n/2$  and  $r_2$  between 1 and  $n/2 - r_1$ . Then for each  $i$  between  $r_1$  and  $r_1 + r_2$ , we interchange the facility located at the  $i$ -th position with the facility located in the  $(n - i)$ -th position.

Our initial experiments showed that the best solutions were obtained more often if we used METHOD II. So we use METHOD II in our implementations to create the AM list.

**Tabu search iteration:** Samarghandi and Eshghi point out in their paper that searching the entire neighborhood of a permutation is time consuming and may not even be feasible, and hence S&E restricts itself to searching a partial neighborhood of a permutation to obtain a good neighbor.

In our implementations, in every iteration of tabu search, we choose one of the permutations in the AM list and search its neighborhood. The choice of permutation from the AM list is done in exactly the same way as in S&E. Consider a permutation in the AM list, which is the  $i$ -th worst solution in the list. The probability of selecting this permutation for neighborhood search is  $2i/(|L| \times |L + 1|)$ . We search the entire neighborhood of the chosen permutation in our implementations, since the speed up techniques described in Section 2 allows us to do so within reasonable time. We accept the best non-tabu neighbor of the permutation in its entire neighborhood, unless we encounter a tabu neighbor which is the best permutation encountered by the algorithm up to that stage. In that case we accept the tabu neighbor as the best neighboring permutation. This process of accepting tabu neighbors is called aspiration. Additionally, while S&E uses the 2-opt neighborhood, we create two implementations, one using the 2-opt neighborhood and the other using the insertion neighborhood in our tabu search algorithm.

**AM list and tabu list update:** At the end of a neighborhood search in S&E, the permutation obtained at the end of the search is added to the AM list and the highest cost permutation is removed from it. The pair of facilities which were interchanged to create the permutation are added to the tabu list.

In S&E the choice of neighbors to examine is random. Hence, starting from the same permutation  $\Pi$ , two neighborhood searches can yield two different permutations  $\Pi_1$  and  $\Pi_2$ . Both  $\Pi_1$  and  $\Pi_2$  may be of high quality, and hence it makes sense to retain  $\Pi$  in the AM list after generating one of these neighbors. Our implementations handle this process differently. We search neighborhoods exhaustively, and hence if  $\Pi$  is used for neighborhood search, it will always return the same neighbor (say  $\Pi_{bn}$ ), barring tabu restrictions. So it does not make sense to store both  $\Pi$  and  $\Pi_{bn}$  in the AM list. Hence in our implementation, if we choose  $\Pi$  for neighborhood search we replace it with its best neighbor found by the exhaustive neighborhood search process.

As a result of our AM list update strategy, our implementations differ fundamentally from S&E. They essentially become multi-start tabu search algorithms, where the number of iterations a particular start is allowed depends on the costs of the solutions it generates in the history of the algorithm. In our implementations, the tabu searches starting from the different initial solutions are independent, and so it does not make sense to maintain a single tabu list for all permutations in the AM list. We therefore maintain separate tabu lists and iteration counters for each of the members of the AM list. If a particular permutation is chosen from the AM list for neighborhood search in our implementations, the tabu list for that member of the AM list is updated. The tabu list then restricts the pair of facilities interchanged for the next  $\theta$  iterations which start from the same permutation or its descendants in the list.

**Final intensification:** Samarghandi and Eshghi use a restricted 2-opt neighborhood for their algorithm. In our implementations we use a complete neighborhood search for the intensification step.

We end this section with a template for our tabu search implementations.

## ALGORITHM TABUSEARCH-TEMPLATE

**Input:** A SRFLP instance of size  $n$ , the cardinality of the adaptive memory (AM) list i.e.,  $L$ , the Tabu tenure  $\theta$  and total number of tabu search iterations i.e., MAXITER.

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

**Code**

```

1. begin
2.   set  $\mathbf{nbr} \leftarrow \text{UNDEFINED}$ ;  $\mathbf{nbrcost} \leftarrow \infty$ ;
3.   sort the facilities in non-decreasing order of their lengths to obtain
   a permutation  $\Pi$  using Theorem 1 in Samarghandi and Eshghi \(2010\);
4.   use  $\Pi$  and our solution generation method METHOD  $\Pi$  to obtain  $L$ 
   permutations and save them in the AM list;
5.   sort the AM list in a non-decreasing order based on the costs of
   the permutations generated in Step 4;
6.   for  $iter$  from 1 to MAXITER do begin
7.     probabilistically select a permutation  $\Pi_1$  from the AM list;
8.     obtain the best tabu and non-tabu neighbor of  $\Pi_1$  by using an
     exhaustive neighborhood search procedure;
9.     select the best neighbor  $\Pi_1^{nbr}$  keeping a check on the aspiration
     criterion and update the Tabu list using Tabu Tenure  $\theta$ ;
10.    replace  $\Pi_1$  in the AM list by  $\Pi_1^{nbr}$ ;
11.    sort the AM list in a non-decreasing order based on the costs;
12.  end;
13.  perform a neighborhood search on the best permutation in the AM list
   to obtain the best neighbor  $\Pi^*$ ; (* Final intensification *)
14.  set  $\mathbf{nbrcost} \leftarrow \text{cost of } \Pi^*$ ;
15.  set  $\mathbf{nbr} \leftarrow \Pi^*$ ;
16.  output  $\mathbf{nbr}$  and  $\mathbf{nbrcost}$ ;
17. end.

```

We create two implementations from TABUSEARCH-TEMPLATE. We call the first implementation TS-2OPT. In this implementation, the neighborhood searched used in Step 8 is the 2-opt neighborhood. The tabu list for this implementation stores the pair of facilities that were interchanged to generate the neighbor returned by the neighborhood search process. We call the second implementation TS-INSERT. In this implementation, the neighborhood searched in Step 8 of TABUSEARCH-TEMPLATE is the insertion neighborhood. The tabu list for this implementation stores the facility that was re-positioned to generate the neighbor returned by the neighborhood search process.

In the next section we describe our computational experience with these implementations.

## 4 Computational experience

We implemented the TS-2OPT and TS-INSERT algorithms in C and compiled them using the gcc4 compiler. We run our experiments on a personal computer with Intel i-5 2500 3.30 GHz processor with 4GB RAM running Ubuntu Linux version 11.10. Following the implementation of [Samarghandi and Eshghi \(2010\)](#) for problems of size  $n$ , we set the size of the AM list to  $L = \lfloor 2n/3 \rfloor$ , the length of tabu tenure  $\theta = \lfloor n/3 \rfloor$ , and the maximum number of tabu search iterations  $k = 50n$ .

We use large sized SRFLP instances available in the literature to benchmark the performance of our implementations against other implementations available in the literature. The sizes of these SRFLP instances vary from 60 to 100. The instances on which we run our implementation include

(a) Anjos instances of sizes 60, 70, 75, and 80; and (b) QAP based `sko` instances of sizes 64, 72, 81, and 100. There are five problem instances for each problem size. Since both TS-2OPT and TS-INSERT rely on random numbers to pick permutations from the AM list, for each instance considered we run the implementations 100 times and report the best results from these runs.

Table 2 compares the performance of our implementations on these problem instances against the results reported in the literature. The first and second columns in the table show the name of the instance and its size. The third, fourth and fifth columns report the costs of the best permutations for these instances obtained by Samarghandi and Eshghi (2010), Datta et al. (2011) and Hungerländer and Rendl (2011) respectively. The last two columns report the costs of the best permutations for these problems obtained by our two implementations. Anjos and Yen (2009) also reported results from experiments with these instances; however we do not report their results in the table since these have been subsequently superceded in the other studies. Notice that among the implementations reported in the literature, the genetic algorithm in Datta et al. (2011) consistently either matches or betters the best permutations obtained by tabu search implementation in Samarghandi and Eshghi (2010). The SDP relaxation technique used in Hungerländer and Rendl (2011) occasionally betters the best permutations in Datta et al. (2011), but sometimes outputs permutations that are worse than both Samarghandi and Eshghi (2010) and Datta et al. (2011).

Table 2: Costs of best permutations for Anjos instances of sizes 60–80

Instance	Size	S&E	DA&F	H&R	TS-2OPT	TS-INSERT
Anjos-60-01	60	1477834.0	1477834.0	1477834.0	1477834.0	1477834.0
Anjos-60-02	60	841792.0	841792.0	841776.0	841790.0	841776.0
Anjos-60-03	60	648337.5	648337.5	648337.5	648337.5	648337.5
Anjos-60-04	60	398511.0	398468.0	398406.0	398406.0	398406.0
Anjos-60-05	60	318805.0	318805.0	318805.0	318805.0	318805.0
Anjos-70-01	70	1529197.0	1528621.0	1528560.0	1528604.0	<b>1528537.0</b>
Anjos-70-02	70	1441028.0	1441028.0	1441028.0	1441028.0	1441028.0
Anjos-70-03	70	1518993.5	1518993.5	1518993.5	1518993.5	1518993.5
Anjos-70-04	70	969130.0	968796.0	969150.0	968796.0	968796.0
Anjos-70-05	70	4218230.0	4218017.5	4218002.5	4218002.5	4218002.5
Anjos-75-01	75	2393483.5	2393456.5	2393600.5	2393483.5	2393456.5
Anjos-75-02	75	4321190.0	4321190.0	4322492.0	4321190.0	4321190.0
Anjos-75-03	75	1248551.0	1248537.0	1249251.0	<b>1248423.0</b>	<b>1248423.0</b>
Anjos-75-04	75	3942013.0	3941981.5	3941845.5	<b>3941816.5</b>	<b>3941816.5</b>
Anjos-75-05	75	1791408.0	1791408.0	1791469.0	1791408.0	1791408.0
Anjos-80-01	80	2069097.5	2069097.5	2070391.5	2069097.5	2069097.5
Anjos-80-02	80	1921177.0	1921177.0	1921202.0	<b>1921136.0</b>	<b>1921136.0</b>
Anjos-80-03	80	3251413.0	3251368.0	3251413.0	3251413.0	3251368.0
Anjos-80-04	80	3746515.0	3746515.0	3747829.0	3746515.0	3746515.0
Anjos-80-05	80	1589061.0	1588901.0	1590847.0	1588901.0	<b>1588885.0</b>

The TS-2OPT tabu search implementation matches the best permutations generated by the tabu search implementation in Samarghandi and Eshghi (2010) in 11 of the 20 instances, and generates better, i.e., lower cost permutations in the other 9 instances. Compared to the best known permutations in the literature, it generates better permutations for 3 of the 20 instances (marked in boldface in column 6 of Table 2), reproduces the best permutation in 13, and produces worse permutations in 4 of the instances. On average, this implementation required 8.6 CPU seconds per run.

The TS-INSERT tabu search implementation matches the best permutations generated by the implementation of Samarghandi and Eshghi (2010) in 8 of the 20 instances and in the other 12 instances provides better permutations. It betters the best permutations known in the literature in 5 of the 20 instances (marked in boldface in column 7 in Table 2). Of these five, it outputs

permutations better than those output by TS-2OPT in two of the instances. In the remaining 15 instances, its output matches the best known permutations in the literature. On average, this implementation required 18.8 CPU seconds per run.

The results presented in Table 2 indicate that TS-2OPT and TS-INSERT implementations output better permutations than the implementation in Samarghandi and Eshghi (2010) at least for these problems. Since both of them better the best permutations known in the literature for several of the benchmark instances, they are competitive algorithms. Further, since TS-INSERT generates at least as good a permutation as any other algorithm known in the literature for all the instances, and generates better permutations than previously known for 5 out of the 20 instances, it is to be preferred to TS-2OPT for generating good permutations for SRFLP instances.

We next check the consistency of the results that TS-2OPT and TS-INSERT output during multiple runs on the same Anjos instance. For each implementation and for each instance, we keep track of the costs of the best, i.e., lowest cost and the worst, i.e., highest cost permutations that it outputs at the end of a run over the 100 runs, the first run in which the implementation outputs the best permutation over the 100 runs, and the number of runs in which the implementation outputs this permutation. These four values are reported in Table 3 under the labels “Best”, “Worst”, “First” and “Times” respectively. The results in the table allow us to compare the TS-2OPT and

Table 3: Consistency of TS-2OPT and TS-INSERT on Anjos instances of sizes 60–80

Instance	TS-2OPT				TS-INSERT			
	Best	First	Times	Worst	Best	First	Times	Worst
Anjos-60-01	1477834.0	2	68	1479234.0	1477834.0	1	99	1477840.0
Anjos-60-02	841790.0	13	4	841872.0	841776.0	1	48	841918.0
Anjos-60-03	648337.5	15	3	650699.5	648337.5	3	41	650484.5
Anjos-60-04	398406.0	16	2	399630.0	398406.0	1	19	401769.0
Anjos-60-05	318805.0	11	15	321891.0	318805.0	5	24	321214.0
Anjos-70-01	1528604.0	5	7	1531777.0	1528537.0	3	15	1528618.0
Anjos-70-02	1440128.0	2	31	1444804.0	1441028.0	1	65	1441515.0
Anjos-70-03	1518993.0	1	11	1525273.5	1518993.5	1	85	1519583.5
Anjos-70-04	968796.0	26	7	971409.0	968796.0	3	28	970062.0
Anjos-70-05	4218002.0	3	4	4220374.0	4218002.5	1	83	4218451.5
Anjos-75-01	2393456.5	43	1	2398581.5	2393456.5	6	34	2395644.5
Anjos-75-02	4321190.0	3	8	4323463.0	4321190.0	1	60	4322615.0
Anjos-75-03	1248607.0	57	1	1251596.0	1248423.0	1	30	1251900.0
Anjos-75-04	3941816.5	19	11	3947472.5	3941816.5	5	39	3950037.5
Anjos-75-05	1791408.0	55	1	1797945.0	1791408.0	1	55	1798053.0
Anjos-80-01	2069145.5	63	2	2072983.5	2069097.5	15	7	2071131.5
Anjos-80-02	1921136.0	70	2	1926642.0	1921136.0	1	99	1921196.0
Anjos-80-03	3251368.0	42	2	3264547.0	3251368.0	11	7	3269153.0
Anjos-80-04	3746515.0	25	4	3750294.0	3746515.0	1	79	3747798.0
Anjos-80-05	1588885.0	11	3	1589215.0	1588885.0	2	76	1589026.0

TS-INSERT implementations in more detail.

In 14 of the 20 instances, the difference between the costs of the best and worst permutations is higher for the TS-2OPT implementation than the TS-INSERT implementation. In the other 6 instances the differences are identical. This shows that there is a higher chance of obtaining a higher cost permutation in a particular run of TS-2OPT than in TS-INSERT. Also, TS-INSERT always encounters the best permutation over all 100 runs much earlier (except in Anjos-70-03) and more frequently than TS-2OPT. These observations, along with the observations on the costs of permutations that TS-2OPT and TS-INSERT output at the end of 100 runs makes TS-INSERT the favored implementation for solving large sized SRFLP instances. This is interesting, since the

2-opt neighborhood has consistently been preferred over the insertion neighborhood in recent local search based implementations to solve the SRFLP. Our experience from computational experiments is that the insertion neighborhood is a better neighborhood to explore for the SRFLP, at least for the Anjos instances.

Finally, we present the results from TS-2OPT and TS-INSERT on the *sko* instances. These instances have not been used in the literature on solving SRFLP using local search. Computational experiments on these instances using SDP based relaxations have been reported in Anjos and Yen (2009) Hungerländer and Rendl (2011). Although the primary aim of SDP based relaxations is to obtain good lower bounds, and not good feasible solutions, the results of Hungerländer and Rendl (2011) presented in Table 2 show that the layouts they obtain are quite competitive for medium sized SRFLP instances. Here too, we do not report results from Anjos and Yen (2009) since they have been superseded in Hungerländer and Rendl (2011).

Table 4: Computational results for QAP based *sko* instances of sizes 64–100

Problem	Size	H&R	TS-2OPT	TS-INSERT
<i>sko</i> -64-01	64	97194.0	<b>96915.0</b>	96969.0
<i>sko</i> -64-02	64	634332.5	634563.5	634595.5
<i>sko</i> -64-03	64	414384.5	<b>414327.5</b>	414338.5
<i>sko</i> -64-04	64	298155.0	<b>297332.0</b>	297399.0
<i>sko</i> -64-05	64	502063.5	<b>501922.5</b>	<b>501922.5</b>
<i>sko</i> -72-01	72	139231.0	139195.0	<b>139179.0</b>
<i>sko</i> -72-02	72	715611.0	<b>712011.0</b>	712217.0
<i>sko</i> -72-03	72	1061762.5	<b>1054110.5</b>	<b>1054110.5</b>
<i>sko</i> -72-04	72	924019.5	<b>920086.5</b>	921268.5
<i>sko</i> -72-05	72	430288.5	428617.5	<b>428248.5</b>
<i>sko</i> -81-01	81	207063.0	205161.0	<b>205145.0</b>
<i>sko</i> -81-02	81	526157.5	<b>521399.5</b>	521402.5
<i>sko</i> -81-03	81	979281.0	971169.0	<b>970912.0</b>
<i>sko</i> -81-04	81	2035569.0	2032361.0	<b>2032143.0</b>
<i>sko</i> -81-05	81	1311166.0	1304266.0	<b>1302833.0</b>
<i>sko</i> -100-01	100	380562.0	<b>378626.0</b>	378634.0
<i>sko</i> -100-02	100	2084924.5	2076231.5	<b>2076023.5</b>
<i>sko</i> -100-03	100	16216076.5	16159760.0	<b>16149000.0</b>
<i>sko</i> -100-04	100	3263493.0	3238812.0	<b>3233362.0</b>
<i>sko</i> -100-05	100	1040929.5	<b>1033338.5</b>	1033421.5

The results in Table 4 show that TS-2OPT and TS-INSERT generate better results than the method used in Hungerländer and Rendl (2011). The costs of permutations output by both the implementations are lower than the costs of permutations reported in Hungerländer and Rendl (2011) in 19 of the 20 instances. The best solutions obtained for these 19 instances are marked in boldface in Table 4. However these results are interesting in that they do not demonstrate the clear superiority of the TS-INSERT implementation over the TS-2OPT implementation. For the smaller instances of sizes 64 and 72, the TS-2OPT generally outputs better permutations than TS-INSERT. However, as problem sizes increase, TS-INSERT starts to output better permutations in more instances than TS-2OPT.

The tests for consistency of the output of TS-2OPT and TS-INSERT for *sko* instances is identical to that for the Anjos instances. The results of these tests are shown in Table 5. From this table we see that in contrast to the Anjos instances, TS-2OPT encounters the permutation that it outputs faster than TS-INSERT for 8 of the 20 instances. In 4 of these instances, it also outputs a better permutation. However, 3 of these 4 instances are of relatively smaller sizes. Another interesting point to note is that the frequencies with which the two implementations encounter the permutations they output at the end of 100 runs are much smaller for these instances than for the Anjos instances,

Table 5: Consistency of TS-2OPT and TS-INSERT on *sko* instances of sizes 64–100

Instance	TS-2OPT				TS-INSERT			
	Best	First	Times	Worst	Best	First	Times	Worst
<i>sko</i> -64-01	96915.0	2	4	98243.0	96969.0	13	4	97833.0
<i>sko</i> -64-02	634563.5	11	1	641434.5	634595.5	42	1	635739.5
<i>sko</i> -64-03	414327.5	29	2	417157.0	414338.5	5	8	416555.5
<i>sko</i> -64-04	297332.0	74	1	299164.0	297399.0	29	1	299921.0
<i>sko</i> -64-05	501922.5	28	2	507073.5	501922.5	10	6	503509.5
<i>sko</i> -72-01	139195.0	29	1	139916.0	139179.0	45	1	139740.0
<i>sko</i> -72-02	712011.0	28	1	715911.0	712217.0	4	25	714987.0
<i>sko</i> -72-03	1054110.5	15	1	1057721.5	1054110.5	36	4	1057234.5
<i>sko</i> -72-04	920086.5	53	2	923771.5	921268.5	22	3	927308.5
<i>sko</i> -72-05	428617.5	35	1	431201.5	428248.5	46	1	430103.5
<i>sko</i> -81-01	205161.0	97	1	206974.0	205145.0	35	1	206486.0
<i>sko</i> -81-02	521399.5	44	2	525414.5	521402.5	52	4	524994.5
<i>sko</i> -81-03	971169.0	37	1	974278.0	970912.0	28	2	973443.0
<i>sko</i> -81-04	2032361.0	12	1	2037886.0	2032143.0	9	3	2034954.0
<i>sko</i> -81-05	1304266.0	39	1	1310419.0	1302833.0	17	2	1311433.0
<i>sko</i> -100-01	378626.0	62	1	380456.0	378634.0	32	1	380179.0
<i>sko</i> -100-02	2076231.5	57	1	2086077.5	2076023.5	7	1	2080712.5
<i>sko</i> -100-03	16159760.0	74	1	16253166.0	16149000.0	15	3	16251524.0
<i>sko</i> -100-04	3238812.0	20	1	3249066.0	3233362.0	84	1	3251315.0
<i>sko</i> -100-05	1033338.5	26	1	1036083.5	1033421.5	27	3	1035178.5

which leads us to believe that these instances are harder for TS-2OPT and TS-INSERT than the Anjos instances. For *sko* instances therefore, we conclude that TS-2OPT is a better implementation than TS-INSERT for smaller sized instances, while TS-INSERT is superior for larger sized instances.

## 5 Summary and discussion

The single row facility layout problem (SRFLP) is a NP-hard problem and current exact algorithms have not been able to solve instances of this problem with more than 42 facilities. Hence for large sized SRFLP instances, research has focused on metaheuristics which can produce near optimal solutions in reasonable time. In this paper we develop two tabu search implementations for the SRFLP which are loosely based on the implementation in [Samarghandi and Eshghi \(2010\)](#). Our implementations search the 2-opt and insertion neighborhoods of permutations depicting solutions to a SRFLP instance. They search the neighborhoods exhaustively in contrast to existing algorithms which only sample permutations from the neighborhood in large instances.

In order to keep computational times for our implementations within acceptable limits, in Section 2 we present techniques to speed up the search over the 2-opt and insertion neighborhoods by between 10 and 20 times. We then use these techniques in Section 3 to create tabu search implementations that are superior to existing tabu search implementations.

In Section 4 we present computational results of our implementations for 40 benchmark SRFLP instances studied in the literature. Our implementations generate permutations that are better than the best known in the literature for 24 of the 40 benchmark instances for this problem. In this section we also make the surprising observation that the insertion neighborhood which has been overlooked in recent tabu search implementations in favor of the 2-opt neighborhood is a better neighborhood to search for large SRFLP instances.



## Acknowledgements

The authors thank A.M.S. Amaral and P. Hungerländer for sharing the benchmark instances used in this paper, and H. Samarghandi for explaining several issues relating to the tabu search implementation reported in [Samarghandi and Eshghi \(2010\)](#).

## References

- Amaral, A. and Letchford, A. N. (2011). A polyhedral approach to the single row facility layout problem. Available at <http://eprints.lancs.ac.uk/id/eprint/49043>.
- Amaral, A. R. S. (2006). On the exact solution of a facility layout problem. *European Journal of Operational Research*, 173(2):508–518.
- Amaral, A. R. S. (2008). An Exact Approach to the One-Dimensional Facility Layout Problem. *Operations Research*, 56(4):1026–1033.
- Amaral, A. R. S. (2009). A new lower bound for the single row facility layout problem. *Discrete Applied Mathematics*, 157(1):183–190.
- Anjos, M., Kennings, a., and Vannelli, a. (2005). A semidefinite optimization approach for the single-row layout problem with unequal dimensions. *Discrete Optimization*, 2(2):113–122.
- Anjos, M. F. and Vannelli, A. (2008). Computing Globally Optimal Solutions for Single-Row Layout Problems Using Semidefinite Programming and Cutting Planes. *INFORMS Journal on Computing*, 20(4):611–617.
- Anjos, M. F. and Yen, G. (2009). Provably near-optimal solutions for very large single-row facility layout problems. *Optimization Methods and Software*, 24(4-5):805–817.
- Beghin-Picavet, M. and Hansen, P. (1982). Deux problèmes d’affectation non linéaires. *RAIRO, Recherche Opérationnelle*, 16(3):263–276.
- Braglia, M. (1997). Heuristics for single-row layout problems in flexible manufacturing systems. *Production Planning & Control*, 8(6):558–567.
- Datta, D., Amaral, A. R., and Figueira, J. R. (2011). Single row facility layout problem using a permutation-based genetic algorithm. *European Journal of Operational Research*, 213(2):388–394.
- Heragu, S. S. and Alfa, A. S. (1992). Experimental analysis of simulated annealing based algorithms for the layout problem. *European Journal of Operational Research*, 57(2):190–202.
- Heragu, S. S. and Kusiak, A. (1988). Machine Layout Problem in Flexible Manufacturing Systems. *Operations Research*, 36(2):258–268.
- Heragu, S. S. and Kusiak, A. (1991). Efficient models for the facility layout problem. *European Journal Of Operational Research*, 53:1–13.
- Hungerländer, P. and Rendl, F. (2011). A computational study for the single-row facility layout problem. Available at [www.optimization-online.org/DB\\_FILE/2011/05/3029.pdf](http://www.optimization-online.org/DB_FILE/2011/05/3029.pdf).
- Kouvelis, P. and Chiang, W.-C. (1992). A simulated annealing procedure for single row layout problems in flexible manufacturing systems. *International Journal of Production Research*, 30(4):717–732.
- Kouvelis, P. and Chiang, W.-C. (1996). Optimal and Heuristic Procedures for Row Layout Problems in Automated Manufacturing Systems. *Journal of the Operational Research Society*, 47(6):803–816.

- Kumar, S., Asokan, P., Kumanan, S., and Varma, B. (2008). Scatter search algorithm for single row layout problem in fins. *Advances in Production Engineering & Management*, 3(4):193–204.
- Laguna, M., Martí, R., and Campos, V. (1999). Intensification and diversification with elite tabu search solutions for the linear ordering problem. *Computers & OR*, 26(12):1217–1230.
- Love, R. F. and Wong, J. Y. (1976). On solving a one-dimensional space allocation problem with integer programming. *INFOR*, 14(2):139–144.
- Martí, R. and Reinelt, G. (2011). *The Linear Ordering Problem*. Springer-Verlag Berlin Heidelberg.
- Picard, J.-C. and Queyranne, M. (1981). On the one-dimensional space allocation problem. *Operations Research*, 29(2):371–391.
- Ravi Kumar, K., Hadejinicola, G. C., and Lin, T.-L. (1995). A heuristic procedure for the single-row facility layout problem. *European Journal of Operational Research*, 87(1):65–73.
- Romero, D. and Sánchez-Flores, A. (1990). Methods for the one-dimensional space allocation problem. *Computers & Operations Research*, 17(5):465–473.
- Samarghandi, H. and Eshghi, K. (2010). An efficient tabu algorithm for the single row facility layout problem. *European Journal of Operational Research*, 205(1):98–105.
- Samarghandi, H., Taabayan, P., and Jahantigh, F. F. (2010). A particle swarm optimization for the single row facility layout problem. *Computers & Industrial Engineering*, 58(4):529–534.
- Simmons, D. M. (1969). One-Dimensional Space Allocation: An Ordering Algorithm. *Operations Research*, 17(5):812–826.
- Solimanpur, M., Vrat, P., and Shanker, R. (2005). An ant algorithm for the single row layout problem in flexible manufacturing systems. *Computers & Operations Research*, 32(3):583–598.

## Appendix

We provide details of the permutations for the 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
Anjos-70-01	70	1528537.0	52 46 64 1 39 27 61 21 14 7 31 8 62 30 68 50 67 0 3 15 63 60 40 37 55 66 69 43 9 25 13 18 32 41 48 4 29 35 22 54 59 12 17 20 23 26 53 10 11 57 5 58 51 6 19 65 2 33 44 45 24 42 47 16 28 56 38 34 36 49
Anjos-75-03	75	1248423.0	46 68 41 9 18 32 14 16 42 50 40 45 28 22 67 25 59 3 38 73 63 60 55 19 35 11 26 12 47 70 10 64 56 4 66 44 20 27 34 23 8 74 57 72 39 6 31 5 48 51 58 33 2 15 61 30 29 43 36 1 37 65 69 17 71 7 24 54 52 62 13 0 53 49 21
Anjos-75-04	75	3941816.0	35 59 4 13 14 49 6 74 9 41 61 36 7 69 29 46 21 56 19 40 28 39 32 38 45 11 2 63 34 64 15 51 27 52 43 72 33 17 23 44 12 31 0 66 1 18 54 47 55 62 65 25 22 57 58 53 42 70 3 30 10 73 60 50 5 24 26 67 68 37 71 48 8 16 20
Anjos-80-02	80	1921136.0	21 56 17 23 9 16 54 4 57 13 45 14 55 31 73 46 11 53 40 28 15 32 76 0 71 26 27 22 24 34 20 5 66 2 52 19 3 37 38 35 8 39 33 7 64 1 49 43 61 41 74 12 25 67 18 58 75 72 29 69 6 78 36 70 30 51 63 77 59 79 50 68 62 48 42 47 44 60 65 10

Instance	Size	Cost	Permutation
Anjos-80-05	80	1588885.0	24 16 47 40 57 59 68 77 12 48 41 71 11 28 15 50 13 63 25 27 23 14 55 4 56 49 45 22 62 39 44 19 79 3 17 76 38 31 2 58 64 54 65 70 34 52 42 5 43 53 74 78 33 29 61 35 32 75 8 18 30 7 69 0 21 10 73 36 9 67 66 20 72 60 26 37 51 46 6 1
sko-64-01	64	96915.0	14 35 11 26 60 40 33 2 45 31 53 10 27 42 29 59 39 47 4 38 17 5 46 50 37 54 19 61 28 13 51 22 23 21 15 6 58 20 3 48 12 63 44 32 55 0 43 16 34 36 49 57 25 7 18 1 62 24 41 52 9 8 56 30
sko-64-03	64	414327.0	14 11 60 8 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 34 16 37 33 29 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	297332.0	30 59 35 41 42 26 62 19 45 50 2 5 51 60 17 39 13 47 27 8 21 12 3 11 23 61 22 15 48 20 28 54 4 34 57 63 43 46 38 44 6 0 36 33 25 49 37 1 16 24 55 32 9 18 7 29 40 31 53 10 52 56 58 14
sko-64-05	64	501922.5	61 58 14 24 52 49 25 16 18 1 43 0 36 56 6 7 31 37 33 32 38 10 53 40 44 34 20 55 29 23 47 59 27 39 22 2 26 3 30 46 50 42 4 5 15 41 54 57 12 63 9 13 51 11 60 45 19 48 62 28 21 17 8 35
sko-72-01	72	139179.0	11 52 30 63 59 17 34 55 26 69 9 7 29 16 66 20 13 21 64 46 25 27 1 51 22 28 0 31 6 60 15 38 58 48 37 45 12 32 49 62 36 8 5 70 43 54 56 35 50 67 41 39 47 40 14 18 57 19 65 42 3 61 71 68 24 4 10 44 53 23 33 2
sko-72-02	72	712011.0	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 28 50 41 47 24 32 38 66 23 54 43 25 56 67 68 35 14 60 71 15 53 5 65 3 18 0 16 70 39 8 19 63 61 10 44 40 57 4 33
sko-72-03	72	1054110.5	30 13 55 59 17 38 16 66 8 27 25 29 9 69 58 60 61 20 7 49 45 12 62 64 21 63 22 47 32 53 26 15 34 46 52 5 42 36 11 24 68 51 41 23 67 18 65 31 10 44 39 56 43 28 35 3 0 33 70 19 54 40 50 37 48 1 2 4 14 71 6 57
sko-72-04	72	920086.5	11 2 55 63 23 71 44 49 35 40 19 10 3 42 70 33 62 14 60 61 36 65 57 68 24 5 18 8 50 67 4 41 39 47 53 56 43 54 15 66 38 28 52 17 30 37 59 45 22 48 34 21 32 13 64 69 12 9 29 6 26 27 20 1 51 46 25 0 31 58 16 7
sko-72-05	72	428248.5	50 28 33 39 70 8 43 22 0 6 32 37 54 15 40 57 58 45 64 5 18 14 19 13 48 65 67 60 35 3 56 4 44 68 24 12 23 10 42 25 41 51 31 46 61 62 21 49 36 20 11 1 34 71 16 30 66 63 59 38 47 29 27 9 69 53 26 17 55 2 52 7
sko-81-01	81	205145.0	7 46 59 75 15 52 53 33 39 63 45 27 9 28 71 70 76 79 11 78 74 62 36 50 23 58 42 72 17 19 57 26 4 0 61 12 67 31 56 6 51 44 77 43 1 10 41 38 69 34 25 21 54 35 30 48 37 18 14 13 80 20 40 5 3 16 22 68 32 64 66 55 47 8 49 24 29 60 2 73 65
sko-81-02	81	521399.5	7 27 76 59 63 56 75 15 46 71 53 61 6 62 26 9 38 19 17 23 42 52 74 70 39 58 72 36 67 12 45 50 77 0 57 4 78 28 79 31 11 44 1 43 18 30 25 33 8 10 3 41 64 29 35 66 22 49 40 69 73 65 16 68 32 2 20 60 14 5 55 47 51 13 80 34 24 48 37 54 21
sko-81-03	81	970912.5	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 68 51 44 0 7 74 57 72 21 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	2032143.0	79 70 78 21 28 4 40 6 58 42 15 37 67 56 75 76 23 52 12 62 46 30 72 63 74 57 61 26 51 44 0 54 35 34 17 9 53 41 77 55 36 66 3 59 48 31 33 39 71 38 45 32 47 69 14 16 18 10 8 43 11 27 5 20 60 29 1 49 19 80 13 2 50 22 68 64 25 24 7 73 65
sko-81-05	81	1302833.0	39 33 63 21 3 78 53 28 37 46 74 64 29 79 34 25 41 59 42 18 17 11 62 36 75 22 32 72 68 76 58 50 23 66 57 19 24 9 70 48 52 13 40 27 80 31 10 45 0 49 16 20 43 15 12 30 56 69 77 38 61 6 44 1 26 67 60 71 54 2 4 8 7 14 5 35 51 47 55 73 65
sko-100-01	100	378626.0	2 43 34 20 16 28 6 75 52 44 35 11 49 40 47 60 22 48 1 69 81 91 37 98 19 13 45 80 63 71 50 89 25 46 99 68 65 39 42 93 84 66 29 95 3 7 67 97 51 85 18 55 88 72 58 41 86 61 21 8 83 15 92 30 87 31 76 33 74 27 14 5 62 56 53 73 54 23 79 26 4 12 10 59 24 94 9 57 0 96 90 78 38 32 82 70 36 17 77 64
sko-100-02	100	2076023.5	32 28 16 98 2 52 11 60 76 40 35 75 47 49 93 37 71 6 25 39 67 7 80 42 13 58 99 69 68 19 45 34 66 44 91 48 50 1 20 51 97 46 65 89 43 85 18 0 81 22 3 55 54 30 27 14 88 15 79 5 56 36 73 86 33 21 78 83 74 12 82 70 62 53 61 95 29 84 8 63 59 94 10 31 24 87 72 17 92 9 64 41 23 38 4 26 57 96 90 77
sko-100-03	100	161490000.0	2 57 27 7 32 59 75 44 48 52 91 39 60 99 46 40 69 68 42 1 89 65 67 86 85 18 71 37 35 47 28 50 14 49 58 63 6 22 45 16 51 66 97 25 93 41 21 80 33 76 31 87 19 5 56 78 83 17 0 24 73 54 3 9 94 55 92 11 95 98 84 29 72 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	3233362.0	6 2 5 83 44 57 75 98 1 16 11 22 28 43 94 63 33 86 40 13 72 46 29 19 39 9 50 35 69 27 65 71 52 34 68 81 25 45 89 99 37 47 91 60 49 80 76 58 90 56 36 61 84 62 77 54 12 10 3 64 95 14 55 88 0 53 30 73 18 85 66 59 23 79 74 26 20 42 32 21 87 15 8 82 97 7 67 51 93 92 17 96 4 31 24 70 78 38 41 48
sko-100-05	100	1033338.5	2 43 34 20 16 28 6 75 52 44 35 11 49 40 47 60 22 48 1 69 81 91 37 98 19 13 45 80 63 71 50 89 25 46 99 68 65 39 42 93 84 66 29 95 3 7 67 97 51 85 18 55 88 72 58 41 86 61 21 8 83 15 92 30 87 31 76 33 74 27 14 5 62 56 53 73 54 23 79 26 4 12 10 59 24 94 9 57 0 96 90 78 38 32 82 70 36 17 77 64