Tabu search for the single row facility layout problem in FMS using a 3-opt neighborhood

Ravi Kothari
Diptesh Ghosh

W.P. No. 2012-02-01
February 2012

INDIAN INSTITUTE OF MANAGEMENT
AHMEDABAD – 380015
INDIA

Tabu search for the single row facility layout problem in FMS using a 3-opt neighborhood

Ravi Kothari
Diptesh Ghosh

Abstract

Since material handling agents in a FMS are most efficient when moving in straight lines, a common layout of machines in a FMS is a single row layout. This allows a floor designer to model the problem of generating an optimal machine layout in a FMS as a single row facility layout problem (SRFLP). Due to the computational complexity involved in solving the SRFLP, researchers have developed several heuristics to solve large instances of the problem. In this paper, we present a tabu search implementation based on a 3-opt neighborhood search scheme. We also present a technique to speed up the exhaustive 3-opt neighborhood search process significantly. Our computational experiments show that speed up of the 3-opt search is effective, and our tabu search implementation is competitive. The results we present here are better than the currently known best layouts for several large sized benchmark SRFLP instances, and competitive for other benchmark instances.
Keywords: Flexible manufacturing system; Single Row Facility Layout; Facility layout; Tabu Search

1 Introduction

A flexible manufacturing system (FMS) is a production system composed of three major components; a set of machines, a flexible material handling system like a robot or an automated guided vehicle (AGV) and a computer system that controls the overall functioning of the system. FMS is different from a classical systems because of its higher degree of automation, comparably fewer number of machines, frequent set-ups and higher volume and flow of information (Solimanpur et al. 2005). Machines in any FMS are an important resource and it is critical to ensure that they do not unnecessarily remain idle because of a badly designed material handling system. Therefore, an important problem in a FMS is to obtain a layout of machines such that material handling is the most efficient. The literature (see, e.g. Heragu and Kusiak 1988, Solimanpur et al. 2005) points out that the most efficient material handling occurs when robots or AGVs move in a straight line, and hence the problem of laying out machines is one of laying them out in a single row. This constraint motivates the designer in a FMS to model the machine layout problem using the well-known single row facility layout problem (SRFLP). This constraint forces the designer of a FMS to model it using a single row facility layout problem (SRFLP). The objective in this problem is to minimize the total material handling expense by using an optimal layout of machines.

The SRFLP is the NP-hard problem of arranging n facilities of given lengths on a line so as to minimize the weighted sum of the distances between pairs of facilities. The size of a SRFLP instance is the number of facilities in the problem. Formally stated the problem is defined as follows:

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

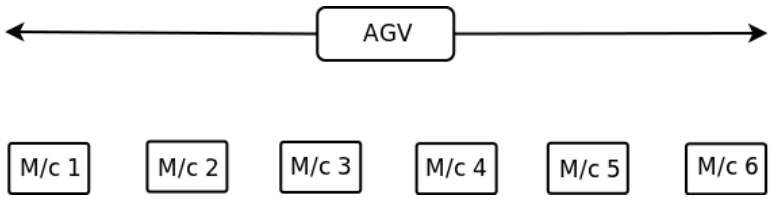

Figure 1: Single row machine layout with AGV

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

$$\sum_{1 \le i < j \le n} c_{\pi_i \pi_j} d_{\pi_i \pi_j}$$

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.

This problem was first proposed in Simmons (1969) and was shown to be NP-Hard in Beghin-Picavet and Hansen (1982).

In the context of a FMS, the facilities are the machines and the weights c_{ij} between a machine pair (i, j) is the material handling expense between the machines. The distance between the machines is usually the distance between their centroids. The designer of a FMS needs to solve the SRFLP to come up with a optimal layout of the machines.

In this paper our objective is to present a technique to obtain good quality layouts of machines for large sized FMSs. The next section provides an overview of the literature and points out an important aspect of neighborhood search which this study addresses in order to come up with good quality layouts for large sized FMSs.

2 Review of the literature

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, Braglia 1997, Solimanpur et al. 2005), assignment of files to disk cylinders in computer storage, and design of warehouse layouts (Picard and Queyranne 1981).

Various exact and approximate solution approaches to the problem have been proposed since 1969. Kothari and Ghosh (2011) provides a current and comprehensive review of the literature on the SRFLP. 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). These methods have been able to obtain optimal solutions to SRFLP instances with up to 42 facilities. This means that single row machine layout problems in FMSs with up to 42 machines can be solved optimally using these techniques.

For solving larger sized SRFLP instances, researchers have focused on heuristics. These heuristics are of two types; construction and improvement. Heragu and Kusiak (1988) presented two construction heuristics to solve the machine layout problem in FMS. In both heuristics, in each iteration, an adjusted flow matrix was computed based on the weight matrix and the solution built before the

iteration. Then a pair of facilities was selected and connected to form a part of the solution. Another construction heuristic was presented in Kumar et al. (1995). This heuristic ignored the lengths of the facilities, and tried to assign facilities with the largest inter-facility weight to adjacent locations in the solution. It differed from other heuristics in the literature since it allowed the assignment of more than one facilities to an existing sequence of facilities at any iteration in the heuristic. A third greedy heuristic was presented in Braglia (1997). It derived ideas from another heuristic for a scheduling problem. Other than these, an insertion based two step heuristic was proposed in Djellab and Gourgand (2001) to solve the SRFLP.

However construction heuristics have since been superseded by improvement heuristics in the literature. Improvement heuristics based on simulated annealing (Romero and Sánchez-Flores 1990, Kouvelis and Chiang 1992, Heragu and Alfa 1992) were used to obtain single row machine layouts for FMSs. Later, ant colony optimization (Solimanpur et al. 2005), and scatter search (Kumar et al. 2008) have been used to solve the machine layout problems in FMSs. Tabu search (Samarghandi and Eshghi 2010), particle swarm optimization (Samarghandi et al. 2010), and genetic algorithms (Datta et al. 2011) have also been used to solve the SRFLP. Among improvement heuristics, 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. The recency of these studies clearly demonstrate continuing interest of researchers to solve the single row machine layout problem in FMS.

In this paper, we present a tabu search algorithm to obtain the layout of machines in large sized FMSs. The 3-opt neighborhood search was presented by Heragu and Alfa (1992) and since then it has never been used in the literature. So we use it with our tabu search algorithm for the SRFLP. The only tabu search implementation for the SRFLP was presented in Samarghandi and Eshghi (2010). The implementation embeds a 2-opt neighborhood search in the tabu search implementation. Apart from using a different neighborhood search technique, our tabu search implementation differs from the one in Samarghandi and Eshghi (2010) in several places. It is interesting to note that Samarghandi and Eshghi (2010) 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. Drawing from the above statement, examining the 3-opt neighborhood would be highly time consuming or may be impossible as the complexity of a 3-opt neighborhood search is $O(n^5)$ while the same for a 2-opt neighborhood search is $O(n^4)$. In this paper we present techniques to speed up the search significantly. We reduce the complexity of search from $O(n^5)$ to $O(n^4)$ which also helps in a time reduction of more than 90% in the neighborhood search process.

The remainder of the paper is organized as follows. In Section 3 we present techniques to speed up searching the 3-opt neighborhood of a solution to a SRFLP instance. We then describe our tabu search implementations using these speed up techniques in Section 4 and present results of our computational experiments in Section 5. We conclude the paper in Section 6 with a summary of the work and directions for future research.

3 Speeding up 3-opt neighborhood search

The 3-opt neighborhood search has been used only in an experimental analysis in Heragu and Alfa (1992). It has never been used in local search based approaches to the solve the SRFLP. In the 3-opt neighborhood, a neighbor of a permutation is obtained by interchanging the locations of at most three of the facilities in the permutation. There are five possibilities of such an interchange, three of which corresponds to permutations in which the locations of exactly two of the facilities have been interchanged and another two in which the locations of exactly three of the facilities have been interchanged. Clearly, for a SRFLP instance of size n, there are $O(n^3)$ neighbors of each permutation in the 3-opt neighborhood, and since computing the cost of a permutation requires

$O(n^2)$ time, a naïve implementation of the 3-opt neighborhood requires $O(n^5)$ time to search the neighborhood exhaustively for the best 3-opt neighbor. This makes exhaustive neighborhood search for large SRFLP instances a very slow process. In this section, we reduce the search time for an exhaustive search of the 3-opt neighborhood of a permutation to $O(n^4)$.

The pseudocode for a program to search the 3-opt neighborhood of a given permutation is given below.

ALGORITHM 3-OPT-NBD-SEARCH

Input: A SRFLP instance of size n, a permutation Π.
Output: A 3-opt neighbor of Π which has the minimum cost among all of Π's 3-opt neighbors.
Code

```
1.  begin
2.      set nbr ← UNDEFINED; nbrcost ← ∞;
3.      for p from 1 to n − 2 do begin                          (* p-loop *)
4.          for q from p + 1 to n − 1 do begin                  (* q-loop *)
5.              for r from q + 1 to n do begin                  (* r-loop *)
6.                  generate all five 3-opt neighbors of Π by interchaning the facilities
                    in positions p, q and, r in Π and set the neighbor with the minimum
                    cost among the five neighbors as 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;                                            (* end of r-loop *)
13.         end;                                                (* end of q-loop *)
14.     end;                                                    (* end of p-loop *)
15.     output nbr and nbrcost;
16. end.
```

Note that in the first iteration of the above algorithm the facilities that need to be interchanged are adjacent to each other i.e., they are the first three facilities in the permutation Π. Also note that in successive iterations of the r-loop inside any p-loop and q loop, the position p and q remains fixed, and the position r shifts one place to the right at a time. Also when p remains fixed, the successive iterations of the q-loop inside any p-loop observes a shift in the position of q one place at a time to the right. We will use these observations to present book-keeping techniques that reduce the complexity of searching for the best 3-opt neighbor of a permutation Π in $O(n^4)$ time.

Consider a permutation $\Pi = (\pi_1, \pi_2, \ldots, \pi_n)$ of facilities, in which π_i denotes the facility at the i-th position in Π. The generation of the 3-opt neighborhood of a solution can be considered to be an outcome of a two-stage process. In the first stage of the process, we generate the set of all triads, i.e., sets of three facilities from the set of facilities defining the instance. In the second stage, for each of the triads generated, we generate the five 3-opt neighbors of the permutation Π by interchanging the facilities whose locations are given by the triad generated. The set of all permutations generated in the second stage forms the required 3-opt neighborhood of Π.

Consider three positions p, q and r between 1 and n and without the loss of generality let $p < q < r$, i.e., generate a triad (p, q, r). Let S_1 be the permutation of facilities to the left of π_p in Π, S_2 be the permutation of facilities between π_p and π_q (both excluded) in Π, S_3 be the permutation of facilities between π_q and π_r (both excluded) and, S_4 be the permutation of facilities to the right of π_r in Π. Let L_i denote the sum of lengths of all the facilities in the permutation S_i $\forall i = 1, 2, 3$ and, 4. The permutation Π is thus of the form $S_1\, \pi_p\, S_2\, \pi_q\, S_3\, \pi_r\, S_4$. For notational convenience we will use

c_{pq} and d_{pq} to represent $c_{\pi_p\pi_q}$ and $d_{\pi_p\pi_q}$ throughout this paper. The cost $z(\Pi)$ of the permutation Π can thus be written as

$$\begin{aligned} z(\Pi) = & \sum_{i\in S_1}\sum_{j\in S_1} c_{ij}d_{ij} + \sum_{i\in S_1} c_{ip}d_{ip} + \sum_{i\in S_1}\sum_{j\in S_2} c_{ij}d_{ij} + \sum_{i\in S_1} c_{iq}d_{iq} + \sum_{i\in S_1}\sum_{j\in S_3} c_{ij}d_{ij} + \sum_{i\in S_1} c_{ir}d_{ir} \\ & + \sum_{i\in S_1}\sum_{j\in S_4} c_{ij}d_{ij} + \sum_{j\in S_2} c_{pj}d_{pj} + c_{pq}d_{pq} + \sum_{j\in S_3} c_{pj}d_{pj} + c_{pr}d_{pr} + \sum_{j\in S_4} c_{pj}d_{pj} + \sum_{i\in S_2}\sum_{j\in S_2} c_{ij}d_{ij} \\ & + \sum_{i\in S_2} c_{iq}d_{iq} + \sum_{i\in S_2}\sum_{j\in S_3} c_{ij}d_{ij} + \sum_{i\in S_2} c_{ir}d_{ir} + \sum_{i\in S_2}\sum_{j\in S_4} c_{ij}d_{ij} + \sum_{j\in S_3} c_{qj}d_{qj} + c_{qr}d_{qr} \\ & + \sum_{j\in S_4} c_{qj}d_{qj} + \sum_{i\in S_3}\sum_{j\in S_3} c_{ij}d_{ij} + \sum_{i\in S_3} c_{ir}d_{ir} + \sum_{i\in S_3}\sum_{j\in S_4} c_{ij}d_{ij} + \sum_{j\in S_4} c_{rj}d_{rj} + \sum_{i\in S_4}\sum_{j\in S_4} c_{ij}d_{ij} \end{aligned} \tag{1}$$

Five 3-opt neighbors are generated when the locations of the facilities at positions p, q and, r, given by the triad (p, q, r), are interchanged. We denote the five 3-opt neighbors of Π as neighbors of type 1, 2, 3, 4 and 5 respectively. All these neighbors of Π are shown in Figure 2. In the type 1 neighbor of Π the locations of all facilities except at positions p, q, and r are identical in the neighbor and the original permutation Π. The facilities at positions p, q, and r in the neighbor of type 1 correspond respectively to the facilities at positions q, r, and p in the original permutation. In type 2 neighbor of Π, the locations of all facilities except at positions p, q, and r are identical in the neighbor and the original permutation Π. The facilities at positions p, q, and r in the neighbor of type 2 correspond respectively to the facilities at positions r, p, and q in the original permutation. The neighbors of type 3, type 4 and type 5 are same as Π with the facilities at positions p & q, q & r and, p & r interchanged in the original permutation Π. So, the neighbors of type 3, 4 and, 5 are essentially 2-opt neighbors of Π. Thus the 2-opt neighborhood of a permutation Π is a proper subset of the 3-opt neighborhood of Π. Hence if $\Pi = S_1\,\pi_p\,S_2\,\pi_q\,S_3\,\pi_r\,S_4$ then its 3-opt neighbors generated using the triad (p, q, r) are type 1: $S_1\,\pi_q\,S_2\,\pi_r\,S_3\,\pi_p\,S_4$, type 2: $S_1\,\pi_r\,S_2\,\pi_p\,S_3\,\pi_q\,S_4$, type 3: $S_1\,\pi_q\,S_2\,\pi_p\,S_3\,\pi_r\,S_4$, type 4: $S_1\,\pi_p\,S_2\,\pi_r\,S_3\,\pi_q\,S_4$, and type 5: $S_1\,\pi_r\,S_2\,\pi_q\,S_3\,\pi_p\,S_4$.

Given the triad (p, q, r), let us consider a 3-opt neighbor Π_1 (of type 1) of Π, then the permutation Π_1 is $\Pi_1 = (\pi_1, \ldots, \pi_{p-1}, \pi_q, \pi_{p+1}, \ldots, \pi_{q-1}, \pi_r, \pi_{q+1}, \ldots, \pi_{r-1}, \pi_p, \pi_{r+1}, \ldots \pi_n)$. Comparing with Π the positions of all facilities in S_1 and S_4 remain unchanged, while the positions of all facilities in S_2 and S_3 undergo a change. Denoting the sum of the lengths of all facilities between π_i and π_j in Π as b_{ij} and using the fact that $c_{ij} = c_{ji}$ and $b_{ij} = b_{ji}$ for every pair π_i and π_j of facilities, the difference Δ_{qrp} in costs of Π and Π_1 i.e., $z(\Pi) - z(\Pi_1)$ can be written as

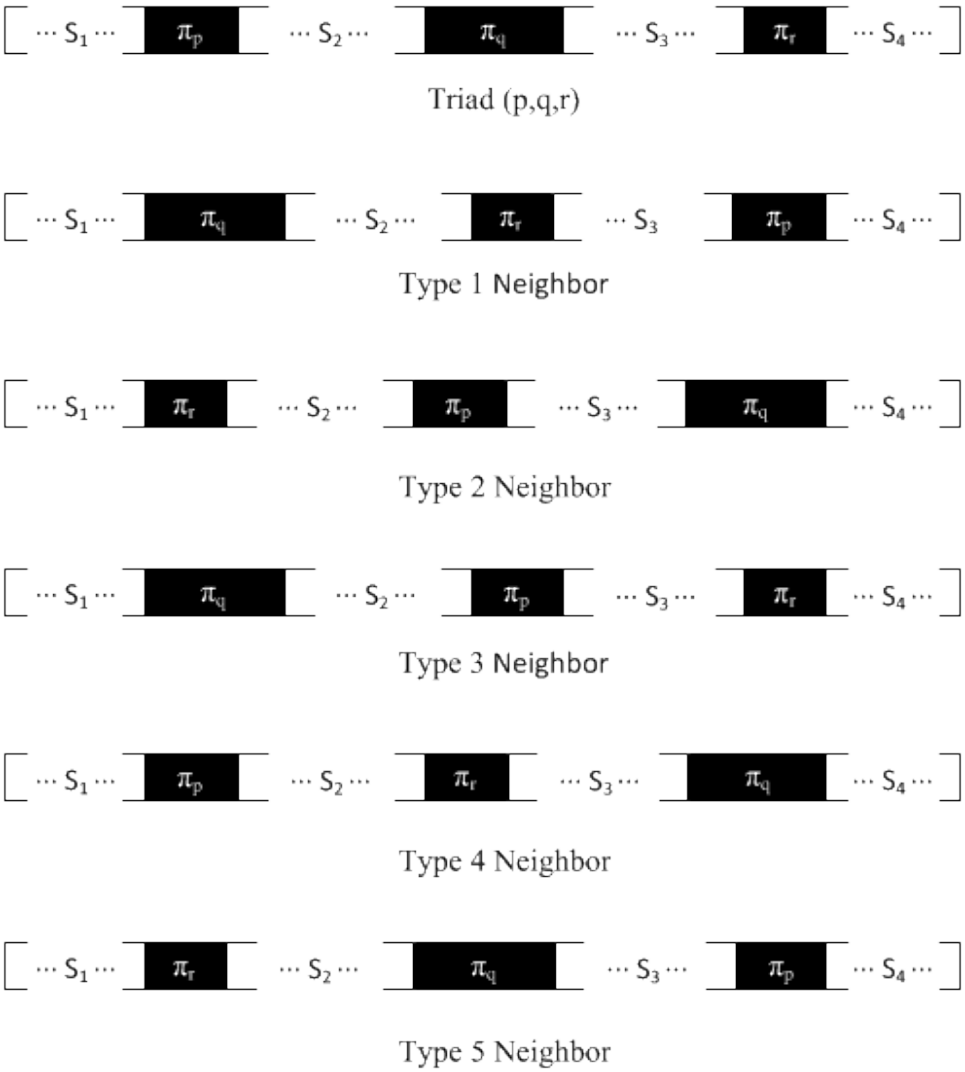

Figure 2: 3-opt neighbors of a triad (p, q, r) in Π

$$
\begin{aligned}
\Delta_{qrp} = {} & L_2\left\{\sum_{i\in S_1}(c_{iq}-c_{ip})+\sum_{i\in S_3\cup S_4}(c_{ip}-c_{iq})+(c_{pr}-c_{qr})\right\} \\
& + L_3\left\{\sum_{i\in S_1\cup S_2}(c_{ir}-c_{ip})+\sum_{i\in S_4}(c_{ip}-c_{ir})+(c_{qr}-c_{pq})\right\} \\
& + l_q\left\{\sum_{i\in S_2}\left(\sum_{j\in S_3}c_{ij}+\sum_{j\in S_4}c_{ij}-\sum_{j\in S_1}c_{ij}+c_{ir}\right)+\sum_{j\in S_3\cup S_4}c_{pj}-\sum_{i\in S_1}c_{ip}+c_{pr}\right\} \\
& + l_r\left\{\sum_{i\in S_3}\left(\sum_{j\in S_4}c_{ij}-\sum_{j\in S_2}c_{ij}-\sum_{j\in S_1}c_{ij}-c_{iq}\right)-\sum_{j\in S_1\cup S_2}c_{pj}+\sum_{j\in S_4}c_{pj}-c_{pq}\right\} \\
& + l_p\left\{\sum_{i\in S_1}\left(\sum_{j\in S_2}c_{ij}+\sum_{j\in S_3}c_{ij}\right)-\sum_{j\in S_4}\left(\sum_{i\in S_2}c_{ij}+\sum_{i\in S_3}c_{ij}\right)+\sum_{i\in S_1}(c_{iq}+c_{ir})\right. \\
& \left.-\sum_{j\in S_4}(c_{qj}+c_{rj})\right\}+\sum_{i\in S_2}(b_{ip}-b_{iq})(c_{ip}-c_{iq})+\sum_{i\in S_3}(b_{ir}-b_{iq})(c_{ir}-c_{ip}) \qquad (2)
\end{aligned}
$$

Note that if π_p, π_q, and π_r are the first three facilities in that order in Π_1, i.e., the triad is $(1, 2, 3)$, then $S_1 = S_2 = S_3 = \emptyset$. Then using equation (2)

$$\Delta_{qrp} = l_q(\sum_{j \in S_4} c_{pj} + c_{pr}) + l_r(\sum_{j \in S_4} c_{pj} - c_{pq}) - l_p \sum_{j \in S_4} (c_{qj} + c_{rj}). \tag{3}$$

Which can be computed in $O(n)$ time, i.e., Δ_{231} can be computed in $O(n)$ time. A similar exercise can be done to obtain the expressions for the difference Δ_{rpq} in costs of Π and a type 2 neighbor Π_2 i.e., $z(\Pi) - z(\Pi_2)$, the difference Δ_{pq} in costs of Π and a type 3 neighbor Π_3 i.e., $z(\Pi) - z(\Pi_3)$, the difference Δ_{qr} in costs of Π and a type 4 neighbor Π_4 i.e., $z(\Pi) - z(\Pi_4)$ and, the difference Δ_{pr} in costs of Π and a type 5 neighbor Π_5 i.e., $z(\Pi) - z(\Pi_5)$. For a type 2 neighbor generated using the triad $(1, 2, 3)$, $S_1 = S_2 = S_3 = \emptyset$ and the value of Δ_{rpq} is

$$\Delta_{rpq} = -l_p(\sum_{j \in S_4} c_{rj} + c_{rq}) - l_q(\sum_{j \in S_4} c_{rj} - c_{pr}) + l_r\left\{\sum_{j \in S_4} (c_{pj} + c_{qj})\right\} \tag{4}$$

In this special case Δ_{rpq} can be computed in $O(n)$ time, i.e., Δ_{312} can be computed in $O(n)$ time. Again for the neighbors of type 3, 4 and, 5 only two of the facilities are swapped and the third facility is in the same location as in Π and a special case for such a scenario is when the facilities being swapped are adjacent to each other i.e., $S_2 = \emptyset$ when π_p and π_q are adjacent. In that case the expression for Δ_{pq} is given by

$$\Delta_{pq} = l_p\left\{\sum_{i \in S_1} c_{iq} - \sum_{\substack{i \in S_3 \cup \\ \pi_r \cup S_4}} c_{iq}\right\} + l_q\left\{\sum_{\substack{i \in S_3 \cup \\ \pi_r \cup S_4}} c_{pj} - \sum_{j \in S_1} c_{pj}\right\}, \tag{5}$$

which can be computed in $O(n)$ time. Similar expressions can be obtained for Δ_{pr} and Δ_{qr} which can be computed in $O(n)$ time. Next we define the concept of a 1-step neighbor of a triad (p, q, r).

Definition 1 (1-step neighbor of a triad (p, q, r))
Let v, u, and t respectively be the locations of facilities immediately to the right of facilities located at positions p, q, and r in a permutation Π. Then the triads (v, q, r), (p, u, r), and (p, q, t) are called 1-step neighbors of triad (p, q, r).

Now we show that the costs of the 3-opt neighbors (of types 1, 2, 3, 4 and, 5) generated using the 1-step neighbors of the triad (p, q, r), can be obtained in $O(n)$ time when the components of the expression of Δ values corresponding to the 3-opt neighbors (of types 1, 2, 3, 4 and, 5) generated using the triad (p, q, r) are known.

Cost of a type 1 neighbor Π_6 generated from the triad (v, q, r)

Let π_v be the facility immediately to the right of π_p in Π and $\pi_v \neq \pi_q$. If the facilities at the positions given by the triad (v, q, r) in Π are interchanged to obtain a 3-opt neighbor Π_6 of type 1 (Figure 5), then the permutation of facilities S_1' to the left of π_v is S_1 with the facility π_p appended, the permutation S_2' of facilities between π_v and π_q is S_2 with facility π_v removed from the extreme left, the permutation of facilities S_3' between π_q and π_r is identical to S_3 and, the permutation of facilities S_4' to the right of facility π_r is identical to S_4. The sum L_2' of lengths of facilities in S_2' is given by $L_2 - l_v$. Hence $\Pi_6 = S_1' \, \pi_q \, S_2' \, \pi_r \, S_3 \, \pi_v \, S_4$ as shown in Figure 5.

We now show that if the component expressions in Δ_{qrp} (equation (2)) are known then Δ_{qrv} can be computed in $O(n)$ time, The expression for $\Delta_{qrv} = z(\Pi) - z(\Pi_6)$ (with a form similar to

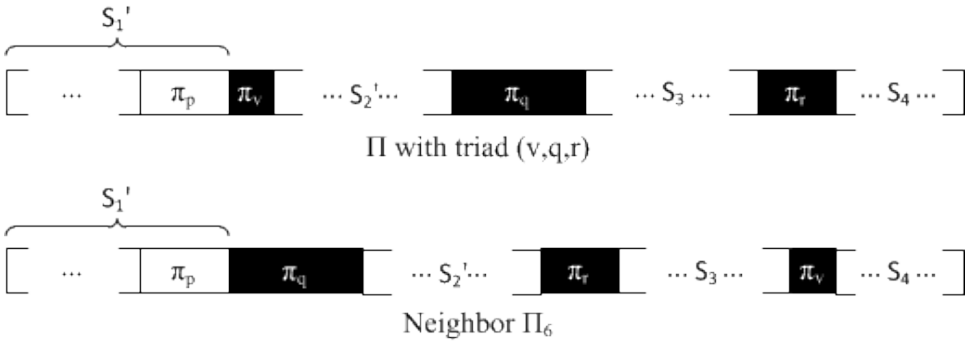

Figure 3: 3-opt neighbor of type 1 of the triad (v, q, r)

equation (2)) can be rearranged to yield

$$\begin{aligned}
\Delta_{qrv} = {} & L_2^{'}\left\{\sum_{i\in S_1^{'}}(c_{iq}-c_{iv})+\sum_{i\in S_3\cup S_4}(c_{iv}-c_{iq})+(c_{vr}-c_{qr})\right\} \\
& + L_3\left\{\sum_{i\in S_1^{'}\cup S_2^{'}}(c_{ir}-c_{iv})+\sum_{i\in S_4}(c_{iv}-c_{ir})+(c_{qr}-c_{vq})\right\} \\
& + l_q\left\{\sum_{i\in S_2^{'}}\left(\sum_{j\in S_3}c_{ij}+\sum_{j\in S_4}c_{ij}-\sum_{j\in S_1^{'}}c_{ij}+c_{ir}\right)+\sum_{j\in S_3\cup S_4}c_{vj}-\sum_{i\in S_1^{'}}c_{iv}+c_{vr}\right\} \\
& + l_r\left\{\sum_{i\in S_3}\left(\sum_{j\in S_4}c_{ij}-\sum_{j\in S_2^{'}}c_{ij}-\sum_{j\in S_1^{'}}c_{ij}-c_{iq}\right)-\sum_{j\in S_1^{'}\cup S_2^{'}}c_{vj}+\sum_{j\in S_4}c_{vj}-c_{vq}\right\} \\
& + l_v\left\{\sum_{i\in S_1^{'}}\left(\sum_{j\in S_2^{'}}c_{ij}+\sum_{j\in S_3}c_{ij}\right)-\sum_{j\in S_4}\left(\sum_{i\in S_2^{'}}c_{ij}+\sum_{i\in S_3}c_{ij}\right)+\sum_{i\in S_1^{'}}(c_{iq}+c_{ir})\right. \\
& \left.-\sum_{j\in S_4}(c_{qj}+c_{rj})\right\}+\sum_{i\in S_2^{'}}(b_{iv}-b_{iq})(c_{iv}-c_{iq})+\sum_{i\in S_3}(b_{ir}-b_{iq})(c_{ir}-c_{iv}). \qquad (6)
\end{aligned}$$

The expression in equation (6) can be computed in $O(n)$ time since

$$\sum_{i\in S_1^{'}}\sum_{j\in S_2^{'}}c_{ij}=\sum_{i\in S_1}(\sum_{j\in S_2}c_{ij}-c_{iv})+(\sum_{j\in S_2}c_{pj}-c_{pv}), \qquad (7)$$

$$\sum_{i\in S_1^{'}}\sum_{j\in S_3}c_{ij}=\sum_{j\in S_3}(\sum_{i\in S_1}c_{ij}+c_{pj}), \qquad (8)$$

$$\sum_{i\in S_2^{'}}\sum_{j\in S_3}c_{ij}=\sum_{j\in S_3}(\sum_{i\in S_2}c_{ij}-c_{vj}) \text{ and,} \qquad (9)$$

$$\sum_{i\in S_2^{'}}\sum_{j\in S_4}c_{ij}=\sum_{j\in S_4}(\sum_{i\in S_2}c_{ij}-c_{vj}); \qquad (10)$$

and since the terms under double summation signs in equations (7) through (10) are already known from equation (2) and the remaining terms on the right hand side of equation (2) can be computed in $O(n)$ time. Hence the value of Δ_{qrv} can be computed on $O(n)$ time. So the objective function value of the neighbor Π_6 which is $z(\Pi)-\Delta_{qrv}$ can be computed in $O(n)$ time.

Cost of a type 1 **neighbor** Π_7 **generated from the triad** (p, u, r)

Let π_u be the facility immediately to the right of π_q in Π and $\pi_u \neq \pi_r$. If the facilities at the positions (p, u, r) in Π are interchanged to obtain a 3-opt neighbor Π_7 of type 1, then the permutation of facilities S_1' to the left of π_p is identical to S_1, the permutation S_2' of facilities between π_p and π_u is S_2 with the facility π_u appended, the permutation of facilities S_3' between π_u and π_r is S_3 with the facility π_u removed from the extreme left and, the permutation of facilities S_4' to the right of facility π_r is identical to S_4. The sum L_2' and L_3' of lengths of facilities in S_2' and S_3' are given by $L_2 + l_q$ and $L_3 - l_u$ respectively. Hence $\Pi_7 = S_1\, \pi_u\, S_2'\, \pi_r\, S_3'\, \pi_p\, S_4$ as shown in Figure **??**.

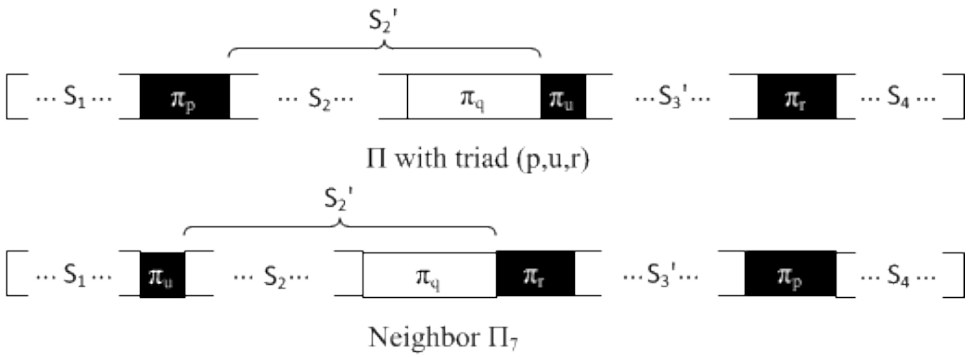

Figure 4: 3-opt neighbor of type 1 of the triad (p, u, r)

We now show that if the component expressions in Δ_{qrp} (equation (2)) are known then Δ_{urp} can be computed in $O(n)$ time, The expression for $\Delta_{urp} = z(\Pi) - z(\Pi_7)$ (with a form similar to equation (2)) can be rearranged to yield

$$
\begin{aligned}
\Delta_{urp} = {} & L_2^{'}\left\{\sum_{i\in S_1}(c_{iu}-c_{ip}) + \sum_{i\in S_3'\cup S_4}(c_{ip}-c_{iu}) + (c_{pr}-c_{ur})\right\} \\
& + L_3^{'}\left\{\sum_{i\in S_1\cup S_2'}(c_{ir}-c_{ip}) + \sum_{i\in S_4}(c_{ip}-c_{ir}) + (c_{ur}-c_{pu})\right\} \\
& + l_u\left\{\sum_{i\in S_2'}\left(\sum_{j\in S_3'}c_{ij} + \sum_{j\in S_4}c_{ij} - \sum_{j\in S_1}c_{ij} + c_{ir}\right) + \sum_{j\in S_3'\cup S_4}c_{pj} - \sum_{i\in S_1}c_{ip} + c_{pr}\right\} \\
& + l_r\left\{\sum_{i\in S_3'}\left(\sum_{j\in S_4}c_{ij} - \sum_{j\in S_2'}c_{ij} - \sum_{j\in S_1}c_{ij} - c_{iu}\right) - \sum_{j\in S_1\cup S_2'}c_{pj} + \sum_{j\in S_4}c_{pj} - c_{pu}\right\} \\
& + l_p\left\{\sum_{i\in S_1}\left(\sum_{j\in S_2'}c_{ij} + \sum_{j\in S_3'}c_{ij}\right) - \sum_{j\in S_4}\left(\sum_{i\in S_2'}c_{ij} + \sum_{i\in S_3'}c_{ij}\right) + \sum_{i\in S_1}(c_{iu}+c_{ir})\right. \\
& \left. - \sum_{j\in S_4}(c_{uj}+c_{rj})\right\} + \sum_{i\in S_2'}(b_{ip}-b_{iu})(c_{ip}-c_{iu}) + \sum_{i\in S_3'}(b_{ir}-b_{iu})(c_{ir}-c_{ip}) \qquad (11)
\end{aligned}
$$

The expression in equation (11) can be computed in $O(n)$ time since

$$\sum_{i\in S_1}\sum_{j\in S_2'} c_{ij} = \sum_{i\in S_1}(\sum_{j\in S_2} c_{ij} + c_{iq}), \tag{12}$$

$$\sum_{i\in S_1}\sum_{j\in S_3'} c_{ij} = \sum_{i\in S_1}(\sum_{j\in S_3} c_{ij} - c_{iu}), \tag{13}$$

$$\sum_{i\in S_2'}\sum_{j\in S_3'} c_{ij} = \sum_{i\in S_2}(\sum_{j\in S_3} c_{ij} - c_{iu}) + (\sum_{j\in S_3} c_{qj} - c_{qu}), \tag{14}$$

$$\sum_{i\in S_2'}\sum_{j\in S_4} c_{ij} = \sum_{j\in S_4}(\sum_{i\in S_2} c_{ij} + c_{qj}) \text{ and,} \tag{15}$$

$$\sum_{i\in S_3'}\sum_{j\in S_4} c_{ij} = \sum_{j\in S_4}(\sum_{i\in S_3} c_{ij} - c_{uj}); \tag{16}$$

and since the terms under double summation signs in equations (12) through (16) are already known from equation (2) and the remaining terms on the right hand side of equation (2) can be computed in $O(n)$ time. Hence the value of Δ_{urp} can be computed on $O(n)$ time. So the objective function value of the neighbor Π_7 which is $z(\Pi) - \Delta_{urp}$ can be computed in $O(n)$ time.

Cost of a type 1 **neighbor** Π_8 **generated from the triad** (p, q, t)

Let π_t be the facility immediately to the right of π_r in Π and $\pi_r \neq \pi_n$. If the facilities at the positions (p, q, t) in Π are interchanged to obtain a 3-opt neighbor Π_8 of type1 (Figure **??**), then the permutation of facilities S_1' to the left of π_p is identical to S_1, the permutation S_2' of facilities between π_p and π_q is identical to S_2 , the permutation of facilities S_3' between π_q and π_t is S_3 with the facility π_r appended and, the permutation of facilities S_4' to the right of facility π_t is S_4 with the facility π_t removed from the extreme left. The sum L_3' of lengths of facilities in S_3' is given by $L_3 + l_r$ respectively. Hence $\Pi_8 = S_1\, \pi_q\, S_2\, \pi_t\, S_3'\, \pi_p\, S_4'$.

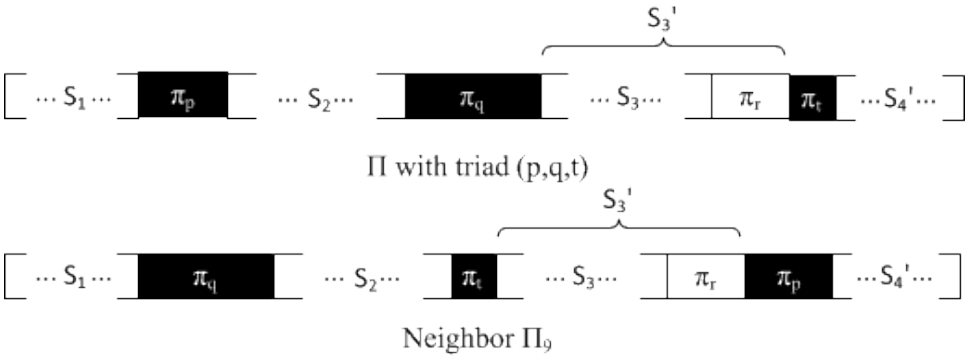

Figure 5: 3-opt neighbor of type 1 of the triad (p, q, t)

We now show that if the component expressions in Δ_{qrp} (equation (2)) are known then Δ_{qtp} can be computed in $O(n)$ time, The expression for $\Delta_{qtp} = z(\Pi) - z(\Pi_8)$ (with a form similar to

equation (2)) can be rearranged to yield

$$\begin{aligned}\Delta_{qtp} = {} & L_2\left\{\sum_{i\in S_1}(c_{iq}-c_{ip}) + \sum_{i\in S_3'\cup S_4'}(c_{ip}-c_{iq}) + (c_{pt}-c_{qt})\right\} \\ & + L_3'\left\{\sum_{i\in S_1\cup S_2}(c_{it}-c_{ip}) + \sum_{i\in S_4'}(c_{ip}-c_{it}) + (c_{qt}-c_{pq})\right\} \\ & + l_q\left\{\sum_{i\in S_2}\left(\sum_{j\in S_3'}c_{ij} + \sum_{j\in S_4'}c_{ij} - \sum_{j\in S_1}c_{ij} + c_{it}\right) + \sum_{j\in S_3'\cup S_4'}c_{pj} - \sum_{i\in S_1}c_{ip} + c_{pt}\right\} \\ & + l_t\left\{\sum_{i\in S_3'}\left(\sum_{j\in S_4'}c_{ij} - \sum_{j\in S_2}c_{ij} - \sum_{j\in S_1}c_{ij} - c_{iq}\right) - \sum_{j\in S_1\cup S_2}c_{pj} + \sum_{j\in S_4'}c_{pj} - c_{pq}\right\} \\ & + l_p\left\{\sum_{i\in S_1}\left(\sum_{j\in S_2}c_{ij} + \sum_{j\in S_3'}c_{ij}\right) - \sum_{j\in S_4'}\left(\sum_{i\in S_2}c_{ij} + \sum_{i\in S_3'}c_{ij}\right) + \sum_{i\in S_1}(c_{iq}+c_{it})\right. \\ & \left. - \sum_{j\in S_4'}(c_{qj}+c_{tj})\right\} + \sum_{i\in S_2}(b_{ip}-b_{iq})(c_{ip}-c_{iq}) + \sum_{i\in S_3'}(b_{it}-b_{iq})(c_{it}-c_{ip}) \end{aligned} \tag{17}$$

The expression in equation (17) can be computed in $O(n)$ time since

$$\sum_{i\in S_1}\sum_{j\in S_3'}c_{ij} = \sum_{i\in S_1}(\sum_{j\in S_3}c_{ij} + c_{ir}), \tag{18}$$

$$\sum_{i\in S_2}\sum_{j\in S_3'}c_{ij} = \sum_{i\in S_2}(\sum_{j\in S_3}c_{ij} + c_{ir}), \tag{19}$$

$$\sum_{i\in S_2}\sum_{j\in S_4'}c_{ij} = \sum_{i\in S_2}(\sum_{j\in S_4}c_{ij} - c_{it}) \text{ and,} \tag{20}$$

$$\sum_{i\in S_3'}\sum_{j\in S_4'}c_{ij} = \sum_{i\in S_3}(\sum_{j\in S_4}c_{ij} - c_{it}) + \sum_{j\in S_4}c_{rj} - c_{rt}; \tag{21}$$

and since the terms under double summation signs in equations (18) through (21) are already known from equation (2) and the remaining terms on the right hand side of equation (2) can be computed in $O(n)$ time. Hence the value of Δ_{qtp} can be computed on $O(n)$ time. So the objective function value of the neighbor Π_8 which is $z(\Pi) - \Delta_{qtp}$ can be computed in $O(n)$ time.

Hence we have shown that if the component expressions for the difference in cost of the original permutation Π and the type 1 neighbor Π_1 generated using the triad of positions (p, q, r) is known, i.e., Δ_{qrp} is known (equation (2)), then all the costs of 3-opt neighbors(of type 1) of Π generated using the 1-step neighbors of the triad (p, q, r) can be computed in $O(n)$ time. Similarly, if the component expressions for the difference in cost of the original permutation Π and the type i neighbor Π_i are known, i.e., $\Delta_{rpq}, \Delta_{pq}, \Delta_{qr}$ and, Δ_{pr} are known, then all the costs of 3-opt neighbors(of type i) of Π generated using the 1-step neighbors of the triad (p, q, r) can be computed in $O(n)$ time.

At the start of the program 3-OPT-NBD-SEARCH, $p = 1$, $q = 2$ and, $r = 3$ i.e., the triad generated is $(1, 2, 3)$ and the Δ values corresponding to the neighbors of type 1, 2, 3, 4 and, 5 (equations 3, 4 and 5) can be used to compute the cost of all the five 3-opt neighbors of Π in $O(n)$ time. Once these Δ values are known, then repetitively using the equations (6),(11) and, (17) and similar other equations for the remaining type i neighbors, $i = 2, 3, 4$ and 5, and repetitively generating the 1-step neighbors of the previously generated triads, the cost of all the 3-opt neighbors generated using the triads, which are 1-step neighbors of earlier generated triads starting from the triad $(1, 2, 3)$, can be computed in $O(n)$ time. Thus the r-loop in 3-OPT-NBD-SEARCH (steps 5

through 11) requires $O(n^2)$ time, and so 3-OPT-NBD-SEARCH runs in $O(n^4)$ time when the above presented techniques are used. Hence searching the 3-opt neighborhood of a permutation to obtain the best 3-opt neighbor requires $O(n^4)$ time.

In order to test the performance of the speeded up neighborhood search processes, we implemented the 3-OPT-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 specifies the sizes of the problems considered. The second and third columns report the times required by a naïve implementation and our implementation using the speed up techniques to search the 3-opt 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.

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

Neighborhood	Size	Naïve	Enhanced	Speed up
3-Opt	60	109.1	11.7	89.3%
	110	2040.8	131.7	93.5%
	160	13683.2	608.3	95.5%

The table clearly demonstrates the effectiveness of the speed-up techniques presented 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.

4 3-opt neighborhood search based tabu search for the SR-FLP

Our implementation, called TS-3OPT, uses the speeded up 3-opt neighborhood search procedure described in the previous section in a tabu search framework. It is a parallel multi-start tabu search approach which has a structure similar to the one suggested in Samarghandi and Eshghi (2010). Individual runs starting from different initial solutions are called starts, and a start is allowed to execute a number of tabu search iterations that depends on the costs of the permutations that the start encounters in its search.

TS-3OPT starts by generating a user-specified number L of starting permutations. The first permutation is generated using Theorem 1 in Samarghandi and Eshghi (2010) as follows. The facilities are sorted in non-decreasing order of their lengths. The first two facilities in this ordering are placed next to each other. The j-th facility, $j = 3, \ldots, n$, in the ordering is placed on the unoccupied side of the $(j-2)$-th facility. The other $L-1$ starting permutations are generated as follows. For each of the other permutations, the first permutation is copied to the permutation. Then the facility in position i, $1 \leq i \leq n/2$, is interchanged with the facility in position $(n-i)$ with a probability 0.5.

Each start is associated with a tabu list that stores the last few interchanges made by TS-3OPT iterations using that start. In any iteration, if the interchange made by TS-3OPT involves the reversal of any interchange in the list, the corresponding neighbor generated is marked tabu. The number θ of interchanges stored in the tabu list is specified by the user.

In each iteration, TS-3OPT chooses one of the starts and performs one tabu search iteration. The start is chosen in a probabilistic manner. To do this, the costs of the latest permutation obtained for each of the starts are considered. These costs are arranged in non-increasing order. The probability that start i in this order is picked up for the iteration is given by $2i/(L \times (L+1))$. Thus a start which has yielded a higher cost permutation until this iteration has a lower rank in the list and has a lower chance of being chosen in the next iteration. Once a start has been chosen, TS-3OPT performs a 3-opt tabu search iteration with the latest permutation in that start as the incumbent solution. The neighborhood chosen is a restricted 3-opt neighborhood. This neighborhood is a 3-opt neighborhood with the restriction that in the incumbent permutation, the positions of the three facilities being repositioned are not too far away. We use a user-specified parameter called SPAN, and consider switching positions of facilities which are not separated by more than SPAN positions in the incumbent solution. This restriction is implemented so that the execution time for a 3-opt neighborhood search is further reduced. TS-3OPT searches the neighborhood of the incumbent permutation and accepts its best non-tabu neighbor, unless a tabu neighbor satisfies an aspiration criterion, i.e., is the best permutation encountered by the algorithm up to that stage. In that case TS-3OPT accepts the tabu neighbor as the best neighboring permutation. The tabu list for the chosen start is then updated and the iteration is complete.

Once tabu search completes a user specified number k of iterations, the best among the latest permutations for all starts is chosen for a final intensification step. In this step TS-3OPT performs a local search starting from the chosen permutation using a 3-opt neighborhood. It then terminates after it outputs the better among the best permutation that it has encountered during the k tabu search iterations and the locally optimal solution obtained at the end of the final intensification step.

We end this section with a pseudocode for our tabu search implementation.

ALGORITHM TS-3OPT (need to go through)

Input: A SRFLP instance of size n, the number of starts L, the Tabu tenure θ, the SPAN and total number of tabu search iterations k.

Output: The best neighbor of Π which has the minimum cost among all of Π's 3-opt neighbors encountered by the algorithm.

Code

1. begin
2. set `nbr` $\leftarrow$ UNDEFINED; `nbrcost` $\leftarrow \infty$;
3. sort the facilities in non-decreasing order of their lengths to obtain a permutation Π using Theorem 1 in Samarghandi and Eshghi (2010);
4. use Π to obtain L initial permutations for the L starts;
5. sort the permutations in a non-decreasing order based on the costs of the permutations generated in Step 4;
6. for *iter* from 1 to k do begin
7. select a start based on the costs of the permutations obtained from that start and choose the latest permutation Π_1 for that start;
8. obtain the best tabu and non-tabu neighbor of Π_1 by searching the 3-opt neighborhood such that the positions of the facilities are separated at most by SPAN
9. select the best neighbor Π_1^{nbr} keeping a check on the aspiration criterion and update the Tabu list using Tabu Tenure θ;
10. replace Π_1 by Π_1^{nbr} as the latest permutation for the selected start;
11. end;
12. perform a neighborhood search on the best permutation among the latest permutations; (* Final intensification *)
13. obtain the best permutation Π^* encountered during the search;
14. set `nbr` $\leftarrow \Pi^*$; set `nbrcost` $\leftarrow$ cost of Π^*;
15. output `nbr` and `nbrcost`;

16. end.

In the next section we describe our computational experience with the tabu search implementation.

5 Computational experience

We implemented the TS-3OPT algorithm in C and compiled it using the `gcc4` compiler. We ran our experiments on a personal computer with Intel i-5 2500 3.30 GHz processor with 4GB RAM running Ubuntu Linux version 11.10.

We ran preliminary experiments to fix the values of the four parameters k, L, θ and SPAN required to be specified by the user for the TS-3OPT algorithm. We saw that the cost of the best permutation obtained by TS-3OPT improved as the maximum number k of tabu search iterations increased. The rate of improvement however tapered off when k increased to very high values. The time required by TS-3OPT was approximately linear in the values of k. For a fixed value of k the number of starts L had a more complicated effect on the costs of permutations output. As L increased from a low value the costs of permutations output improved. This was probably because with increasing values of L, the initial solutions were more widely dispersed in the solution space, and hence better solutions were being obtained. However the costs of the best permutations obtained by TS-3OPT worsened when the value of L was set too high. This is possibly because when the value of L was too high keeping the value of k fixed, individual starts were not allowed sufficient number of iterations to obtain low cost permutations. The lengths of tabu tenure had no consistent effect on the cost of permutations output, unless they were set to values that were too low to make them effective or too high to make them exceedingly restrictive. The effect of the value of SPAN on the costs of permutations was also interesting. At very low values of SPAN, TS-3OPT was quite fast but generated high cost permutations since only a very small part of the 3-opt neighborhood was being explored by tabu search. As the value of SPAN increased the time required by TS-3OPT increased exponentially, but surprisingly, at high values of SPAN, the cost of solutions being output also increased. We believe that this is because a large value of SPAN causes the algorithm to generate better permutations at each iteration, but forces it in a path that does not lead to low cost permutations in the long run. Hence the balance these two effects the value of SPAN needs to be chosen at an intermediate value. After our preliminary experiments, we chose the maximum number of tabu search iterations $k = 50n$, the number of starts L as $\lfloor 2n/3 \rfloor$, the length of tabu tenure θ as $\lfloor 2n/3 \rfloor$, and the value of SPAN as $\lfloor n/3 \rfloor$. We observed that these parameters yielded low cost permutations in reasonable time.

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 instances that we use are randomly generated instances due to Anjos. The sizes of these SRFLP instances vary from 60 to 80. The Anjos instances consist of five instances each of sizes 60, 70, 75, and 80. Since TS-3OPT has some randomness built into it, for each instance considered we ran the implementation 50 times and report the best results from these runs.

Table 2 compares the performance of our implementations on these problem instances against the results published in the literature. Competitive results on these instances have been reported in the literature in Samarghandi and Eshghi (2010) and Datta et al. (2011). In Table 2 the first and second columns show the name of the instance and its size. The costs of the best permutations reported in the published literature are presented in the third and fourth columns respectively. The last column reports the costs of the best permutations for these problem instances obtained by our TS-3OPT algorithm.

Table 2: Costs of best permutations for large sized Anjos instances

Instance	Size	S&E[a]	DA&F[b]	TS-3OPT
Anjos-60-01	60	1477834.0	1477834.0	1477834.0
Anjos-60-02	60	841792.0	841792.0	**841776.0**
Anjos-60-03	60	648337.5	648337.5	648337.5
Anjos-60-04	60	398511.0	398468.0	398468.0
Anjos-60-05	60	318805.0	318805.0	318805.0
Anjos-70-01	70	1529197.0	1528621.0	**1528537.0**
Anjos-70-02	70	1441028.0	1441028.0	1441028.0
Anjos-70-03	70	1518993.5	1518993.5	1518993.5
Anjos-70-04	70	969130.0	968796.0	968796.0
Anjos-70-05	70	4218230.0	4218017.5	**4218002.5**
Anjos-75-01	75	2393483.5	2393456.5	2393456.5
Anjos-75-02	75	4321190.0	4321190.0	4321190.0
Anjos-75-03	75	1248551.0	1248537.0	1248607.0
Anjos-75-04	75	3942013.0	3941981.5	**3941816.5**
Anjos-75-05	75	1791408.0	1791408.0	1791408.0
Anjos-80-01	80	2069097.5	2069097.5	2069097.5
Anjos-80-02	80	1921177.0	1921177.0	1921177.0
Anjos-80-03	80	3251413.0	3251368.0	3253168.0
Anjos-80-04	80	3746515.0	3746515.0	3746675.0
Anjos-80-05	80	1589061.0	1588901.0	**1588885.0**

a: results reported in Samarghandi and Eshghi (2010)
b: results reported in Datta et al. (2011)

From the table we see that the TS-3OPT algorithm is competitive with the other implementations in almost all the instances. In 5 of the 20 instances, it outputs permutations that are better than the ones reported in the literature. The costs of these permutations are depicted in boldface in Table 2.

6 Summary and discussion

The single row facility layout problem has been widely used to model the machine layout problem in a flexible manufacturing system. The problem is computationally difficult and researchers have focused on improvement heuristics to obtain good quality layouts in reasonable time. The machine layout problem becomes more challenging in large sized FMSs.

In this paper we present a multi-start tabu search algorithm TS-3OPT which uses a 3-opt neighborhood structure to obtain machine layouts for large sized FMSs. Searching the 3-opt neighborhood for a SRFLP is very expensive and we present techniques in Section 3 to speed up the search process. Our technique reduces the complexity of searching a 3-opt neighborhood from $O(n^5)$ to $O(n^4)$ thus allowing us to search the 3-opt neighborhood exhaustively, as opposed to existing algorithms which sample permutations from neighborhoods, even when such neighborhoods are of smaller size like the 2-opt neighborhood. In our algorithm we use a parameter called SPAN to reduce the search time further while still generating good layouts. The techniques in Section 3 help us implement the TS-3OPT tabu search algorithm more effectively for large sized instances of the problem.

After setting the parameters of our algorithm through preliminary experiments, we test the performance of TS-3OPT on several large sized SRFLP instances selected from the literature and compare its performance with the best algorithms available for SRFLP. Our tabu algorithm TS-3OPT improves on 5 out of 20 large sized instances and is competitive for the remaining 15. instances.

The experiments conclude that the proposed tabu algorithm is effective and efficient for the single row machine layout problems.

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 daffectation 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.

Djellab, H. and Gourgand, M. (2001). A new heuristic procedure for the single -row facility layout problem. *International Journal of Computer Integrated Manufacturing*, 14(3):270–280.

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.

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

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

Love, R. F. and Wong, J. Y. (1976). On solving a one-dimensional space allocation problem with integer programming. *INFOR*, 14(2):139–144.

Picard, J.-C. and Queyranne, M. (1981). On the one-dimensional space allocation problem. *Operations Research*, 29(2):371–391.

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.