International Journal of Innovative Computing, Information and Control
Volume **12**, Number **3**, June **2016**

ICIC International ©2016 ISSN 1349-4198
pp. **859–874**

A MULTI-START LOCAL SEARCH ALGORITHM FOR THE SINGLE ROW FACILITY LAYOUT PROBLEM

Jian Guan[1] and Geng Lin[2,*]

[1]Modern Educational Technology Center
Minjiang University
No. 200, Xiyuangong Road, Shangjie County, Minhou, Fuzhou 350108, P. R. China
gjian_mail@163.com

[2]Department of Mathematics
Minjiang University
No. 1, Wenxian Road, University Town, Fuzhou 350108, P. R. China
*Corresponding author: lingeng413@163.com

Received October 2015; revised February 2016

Abstract. *The single row facility layout problem (SRFLP) is an NP-hard problem of arranging facilities with given lengths on a line, while minimizing the total cost associated with the flows between all pairs of facilities. In this paper, we present a multi-start local search algorithm to solve the SRFLP. A diversification generator based on a probability function is proposed to construct good quality and diverse multiple starting solutions. A fast local search including first-improvement procedure and fast evaluation technique is adopted to enhance the exploitation ability. An efficient restarting mechanism is applied to exploiting the potential search space. Computational experiments show that the proposed algorithm is competitive with other heuristics for solving the SRFLP.*
Keywords: Single row facility layout, Multi-start, Local search, First-improvement

1. Introduction.

The single row facility layout problem (SRFLP) seeks to arrange some facilities in a line with the purpose of minimizing the weighted sum of the distances between all pairs of facilities. It has numerous practical applications including the arrangement of rooms in hospitals, the arrangement of departments in office buildings [1], the assignment of disk cylinders to files in computer storage [2], and the arrangement of machines in flexible manufacturing systems [3, 4]. So the SRFLP has attracted significant attention in recent years, since the problem was first proposed by Simmons in 1969 [1]. Much research has been done in obtaining an exact solution to the problem [5]. A branch and bound algorithm was reported to produce exact solutions for instances of size up to 11 [1]. Integer programming approaches were applied to finding a better model for the SRFLP [6, 7, 8, 9]. A dynamic programming was able to solve SRFLP instances with 20 facilities [10]. Amaral and Letchford presented a polyhedral approach with branch and cut to yield excellent lower and upper bounds very quickly for instances with 30 facilities, but computing times are quite long for larger instances [11]. Even though semidefinite programming approaches were reported to produce exact solutions for large instances, they require much computational time [12, 13, 14, 15]. The SRFLP is known to be NP-hard [16], so exact methods are computationally prohibitive to solve large instances. To solve large size SRFLP instances in acceptable time, researchers have focused on heuristic solution methods [5].

Heragu and Alfa reported an experimental analysis of simulated annealing based algorithms. A modified penalty method was applied to generating the initial solution [17].

De Alvarenga et al. presented simulated annealing and tabu search procedures to solve a class of the facility layout problems in the context of manufactures environment. They observed that the processing time of the tabu search was smaller than the simulated annealing [18]. Solimanpur et al. proposed an ant colony optimization algorithm to solve the SRFLP in a non-linear 0-1 mathematical programming model [19]. Kumar et al. applied a scatter search algorithm to searching the optimum solution by a reference set containing best solutions and diverse solutions [20]. Samarghandi et al. developed a particle swarm optimization algorithm based on factoradic coding technique. The factoradic coding technique was employed to map the discrete feasible space of the problem into a continuous space [21]. Samarghandi and Eshghi presented and proved a theorem to find the optimal solution of a special case of SRFLP. The theorem was utilized to generate a number of better initial solutions for a tabu algorithm [22]. Kothari and Ghosh introduced 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 [23]. Datta et al. developed a permutation-based genetic algorithm. Four techniques were used to initialize good and diversified individuals [24]. Ou-Yang and Utamima proposed a hybrid estimation of distribution algorithm based on particle swarm optimization and tabu search [25]. Palubeckis proposed a fast local search algorithm based on the developed neighborhood exploration procedures. It is faster than the best existing local search techniques [26].

From the literature, it can be seen that many algorithms incorporate a local search procedure. Most of local search procedures probe the best solution in all neighbors of the current solution. Due to the exhaustive neighborhood search, these best-improvement local searches are very time consuming. To the flow shop problem, the first-improvement local search gives significantly better results than the best-improvement, in the same amount of allowed computation time. [27] is the only study which has applied first-improvement to the SRFLP, but it deals with SRFLP instances of size up to 30 facilities only. Two main kinds of neighborhood structures have been used in the local searches for this problem. The first of them is based on pairwise interchanges of facilities whereas the second one is based on insertion movement. From [23], it made an observation that the insertion neighborhood is better than the interchange neighborhood. Clearly, for an SRFLP instance of size n, $O(n^2)$ time will be required to compute the cost of a neighbor, according to the conventional formula. [23] developed a book-keeping technique whose time complexity is $O(n)$. However, the technique is used for the best-improvement local search in fixed order. It is not suitable for the first-improvement local search in random order. Multi-start strategies are designed to provide diversity of the search and overcome local optimality [28]. Multi-start strategies hybridized with other heuristics have been applied to solving some combinatorial optimization problems effectively [29, 30, 31]. Therefore, the primary motivation of this paper is to develop a multi-start algorithm with local search to solve the SRFLP of large size in competitive time. The local search procedure probes a first-improvement solution in the insertion neighborhood by random order. We propose a gain technique to speed up the objective function value calculation for our local search procedure.

The proposed algorithm combines the advantages of the diversification of multi-start strategy and the rapidity of first-improvement local search. The multi-start strategy is responsible for escaping from the local optimum and moving towards new unexplored areas of the search space. In first-improvement local search, the first profitable solution found is the one that is accepted, which can often do less computational effort. The computational results indicate that the proposed MSLSA algorithm is effective and efficient in solving SRFLP.

The major contributions of the proposed algorithm can be summarized as follows.

• A diversification generator is proposed to construct good quality and diverse multiple starting solutions. A probability function utilized in diversification generator is devised to enhance the diversification.

• We incorporate a fast local search procedure into multi-start framework. The local search procedure probes a first-improvement neighbor in random order. Its randomness can explore broader solution areas and prevent search stagnation to some extent. Meanwhile, a gain technique is applied to evaluating the objective function value shortening the computational time.

• To exploit the potential search space more efficiently, a novel and powerful restarting mechanism is employed.

The remainder of the paper is organized as follows. The SRFLP is analyzed and formulated as an unconstrained optimization problem in Section 2. Section 3 describes the details of the multi-start local search algorithm for the SRFLP. In Section 4, we present a computational evaluation of the proposed algorithm through benchmark instances, and compare the result with other heuristics from the literature. Finally, some concluding remarks of the work are offered in Section 5.

2. **The Formulation of the SRFLP Model.**

Given a set $F = \{1, 2, \cdots, n\}$ of $n > 2$ facilities, where facility x has length l_x, and there is a flow c_{xy} between each pair (x, y) of facilities, $x, y \in F$, $x \neq y$, the single row facility layout problem (SRFLP) is to arrange the facilities in a line with the purpose of minimizing the weighted sum of the distances between all pairs of facilities. In other words, it is to find a permutation $\Pi = \{\pi_1, \pi_2, \cdots, \pi_n\}$ of facilities in F that minimizes the total cost given by the expression

$$z(\Pi) = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} c_{\pi_i \pi_j} d_{\pi_i \pi_j}, \tag{1}$$

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 centroid of facilities π_i and π_j in the arranged permutation Π. π_i and π_j denote the facilities at the ith and jth positions in Π respectively. For notational convenience we use the notations i and j to represent π_i and π_j respectively.

3. **Multi-Start Local Search Algorithm.**

Multi-start method is the one that executes sequential independent searches from multiple initial points in the solution space, and identifies the best solution from all searches at the end. It is designed to provide diversity of the search and overcome local optimality. In this section, we will propose a multi-start algorithm with first-improvement local search for solving the SRFLP. Our proposed algorithm consists of three steps: diversification generator, local search and speeding up cost calculation. First, we will introduce a diversification generator to create the initial population with good and diversified solutions. Then we use a first-improvement local search to obtain better solutions. To speed up the first-improvement local search, we propose a different fast cost calculation to reduce the complexity. Finally, the architecture of our multi-start local search algorithm is described.

3.1. **Diversification generator.**

Diversification significantly contributes to the quality of solutions in the algorithm. Our algorithm designs L initial permutations for start exploration to ensure the diversification. They contain L_1 good solutions and L_2 diverse solutions and are stored in a memory called Adaptive Memory (AM). L_1 and L_2 are determined by the dimension of the instance. The method to generate diversification initial solutions, called DIS, is described as Algorithm 1.

Algorithm 1 *DIS*

```
Require: An initial seed permutation $\Pi$, $L_1$, $L_2$.
Ensure: The initial population $IP\_List$ with $L$ permutations.
 1: set $IP\_List \leftarrow \emptyset$.
 2: for $i = 1$ to $L_1$ do
 3:     set $Pi^* \leftarrow \Pi$.
 4:     for $j = 1$ to $\lfloor n/2 \rfloor$ do
 5:         $r \leftarrow$ random number chosen from $[0, 1]$;
 6:         if $(r \le \frac{i}{L_1})$ then
 7:             interchange $Pi^*[j]$ and $Pi^*[n-j+1]$ ;
 8:         end if
 9:     end for
10:     set $IP\_List \leftarrow IP\_List \bigcup \{Pi^*\}$;
11: end for
12: for $i = 1$ to $L_2$ do
13:     generate a completely random permutation $Pi^*$;
14:     set $IP\_List \leftarrow IP\_List \bigcup \{Pi^*\}$;
15: end for
16: output $IP\_List$
```

First, L_1 good permutations are constructed from a seed permutation. In Theorem 1 [22], suppose that flow c_{xy} between each pair of facilities is a constant. When the facilities are sorted in non-decreasing order of their lengths, as $l_1 \le l_2 \le \cdots \le l_n$, the optimum permutation can be given as follows:

$(n)(n-2)\cdots(3)(1)(2)\cdots(n-3)(n-1)$, if n is odd or
$(n)(n-2)\cdots(2)(1)(3)\cdots(n-3)(n-1)$, if n is even.

It is noted that the permutations are still optimum by exchanging the locations of facilities i and j, provided that the same number of facilities is at the left and right sides of i and j, respectively. Therefore, we choose the optimum permutation as an initial seed permutation. The seed permutation is copied as a template for new good permutations. In the ith template, for each j between 1 and $n/2$, the facility located at the jth position is interchanged with the facility located at the $(n-j+1)$th position by a probability function $\frac{i}{L_1}$, where $i = 1, 2, \cdots, L_1$. The probability function is different from those of [23, 32]. It helps to construct good solutions while maintaining the diversification.

Then, L_2 permutations are completely randomly generated. It is considered for obtaining a diversified population.

3.2. **Local search.**

The following two kinds of neighborhood structures are often used in local search procedures for the SRFLP: 2-opt neighborhood and insertion neighborhood. In the 2-opt neighborhood, a neighbor is formed by interchanging the positions of two facilities in a permutation. In the insertion neighborhood, a neighbor is formed by removing a facility from a position and re-inserting it at another position in the permutation. Prior literature has shown that insertion neighborhood search performs better than 2-opt neighborhood search [33].

Most of published literature exploited the best-improvement local search to obtain high quality solutions. Due to the exhaustive neighborhood search, the best improvement is very time consuming. To the flow shop problem, the first-improvement local search gives significantly better results than the best-improvement one, in the same amount of allowed computation time [34]. Therefore, our local search procedure looks for solutions by probing the first-improvement solution in the insertion neighborhood.

To increase the diversification, our local search procedure probes neighbors in random order. Its randomness can explore broader solution areas and prevent search stagnation to some extent. Let (r, s) be an inserting operation, removing a facility at the rth position and inserting it to the sth position in the permutation. A random order is generated based on a pool of inserting operations in the natural order:

$$\{(1,2), \cdots, (1,n), (2,1), (2,3), \cdots, (2,n), \cdots, (r,s) \cdots, (n-1,n)\},$$

where $r \neq s$, with $n \times (n-1)$ numbers of inserting operations.

We use the first-improvement local search algorithm ($FILS$) as the improvement method, and the pseudocode is shown in Algorithm 2. A pool of inserting operations in the natural order is initialized at line 4. It is used to search the neighbors of the current solution in random order (lines 6-8). Once the cost of a neighbor is smaller than those of the current solution, the neighbor is accepted as the current solution and restart the local search procedure (lines 10-14). When no better neighbor is found among all neighbors of the current solution, the local search procedure stops. The algorithm is very simple and effective.

Algorithm 2 $FILS$

```
Require: A current solution $S_0$.
Ensure: The improved solution.
 1: set $flag \leftarrow TRUE$.
 2: while ($flag = TRUE$) do
 3:     $flag \leftarrow FALSE$.
 4:     initialize a pool of inserting operations in the natural order.
 5:     for $i = 1$ to $n \times (n-1)$ do
 6:         $S_0^* \leftarrow S_0$.
 7:         select an operation $(r, s)$ from the pool randomly.
 8:         remove a facility at the $r$th position and insert it to the $s$th position in $S_0^*$.
 9:         remove the operation from the pool.
10:         if ($Z(S_0^*) < Z(S_0)$) then
11:             $S_0 \leftarrow S_0^*$.
12:             $flag \leftarrow TRUE$.
13:             break.
14:         end if
15:     end for
16: end while
17: output the improved solution
```

3.3. **Speeding up cost calculation.** Clearly, for an SRFLP instance of size n, there are $O(n^2)$ neighbors of a permutation in the insertion neighborhood, and $O(n^2)$ time will be required to compute the cost of a neighbor, according to Equation (1). A single implementation of the insertion neighborhood requires $O(n^4)$ time to search the neighborhood exhaustively for the best solution. This makes exhaustive neighborhood search for large SRFLP instances very time consuming. A book-keeping technique is applied to reducing the complexity to $O(n^3)$ time in [23, 33]. However, the technique is used for the best-improvement local search in fixed order. It is not suitable for our first-improvement local search in random order. We propose a different fast evaluation technique to reduce the complexity for our local search.

Suppose that the neighbor Π^* is obtained by moving the facility p from the pth position and inserting it at the qth position in the permutation Π, where $p < q$. In fact, some distances between two facilities in Π^* and Π are the same, which avoids having to repeat all of the computing effort in the implementation. This observation inspires us to propose a gain technique to get the cost of the neighbor, saving the computation time. The cost of Π^* can be calculated by the expression $z(\Pi^*) = z(\Pi) + gain$.

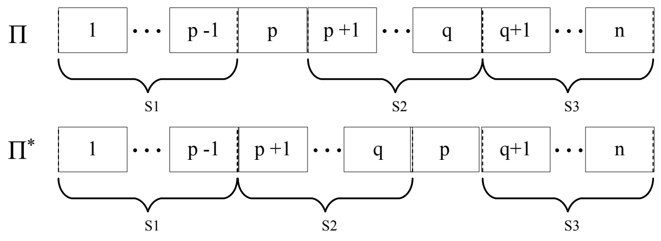

FIGURE 1. Sets partition for the facilities

To explain the gain technique, we partition all facilities in the layout to three distinct sets, as Figure 1 shown:
1. Set $S1$ consists of facilities from position 1 to $(p - 1)$;
2. Set $S2$ consists of facilities from position $(p + 1)$ to q. The sum length of facilities in $S2$ is denoted by $D2$, i.e., $D2 = \sum_{i \in S2} l_i$;
3. Set $S3$ consists of facilities from position $(q + 1)$ to n.

So, the objective function of the SRFLP can be rewritten as:

$$\begin{aligned} z(\Pi) = & \sum_{i \in S1} \sum_{j \in S1, i<j} c_{ij} d_{ij} + \sum_{i \in S1} c_{ip} d_{ip} + \sum_{i \in S1} \sum_{j \in S2} c_{ij} d_{ij} + \sum_{i \in S1} \sum_{j \in S3} c_{ij} d_{ij} + \sum_{i \in S2} c_{pi} d_{pi} \\ & + \sum_{i \in S3} c_{pi} d_{pi} + \sum_{i \in S2} \sum_{j \in S2, i<j} c_{ij} d_{ij} + \sum_{i \in S2} \sum_{j \in S3} c_{ij} d_{ij} + \sum_{i \in S3} \sum_{j \in S3, i<j} c_{ij} d_{ij}. \end{aligned} \tag{2}$$

Observing Figure 1, we can find that the distances $\{d_{ij} | \{i \in S1, j \in S1, i < j\}\}$, $\{d_{ij} | \{i \in S1, j \in S3\}\}$, $\{d_{ij} | \{i \in S2, j \in S2, i < j\}\}$, $\{d_{ij} | \{i \in S3, j \in S3, i < j\}\}$ in Π^* are the same as those in Π. Those components of the cost function do not need to be recalculated, which can save time in the implementation. Therefore, we can formulate the gain as:

$$\begin{aligned} gain = & \; z(\Pi^*) - z(\Pi) \\ = & \sum_{i \in S1} c_{ip} d^*_{ip} + \sum_{i \in S1} \sum_{j \in S2} c_{ij} d^*_{ij} + \sum_{i \in S2} c_{pi} d^*_{pi} + \sum_{i \in S3} c_{pi} d^*_{pi} + \sum_{i \in S2} \sum_{j \in S3} c_{ij} d^*_{ij} \\ & - \left(\sum_{i \in S1} c_{ip} d_{ip} + \sum_{i \in S1} \sum_{j \in S2} c_{ij} d_{ij} + \sum_{i \in S2} c_{pi} d_{pi} + \sum_{i \in S3} c_{pi} d_{pi} + \sum_{i \in S2} \sum_{j \in S3} c_{ij} d_{ij} \right) \\ = & \left(\sum_{i \in S1} c_{ip} d^*_{ip} - \sum_{i \in S1} c_{ip} d_{ip} \right) + \left(\sum_{i \in S1} \sum_{j \in S2} c_{ij} d^*_{ij} - \sum_{i \in S1} \sum_{j \in S2} c_{ij} d_{ij} \right) \\ & + \left(\sum_{i \in S2} c_{pi} d^*_{pi} - \sum_{i \in S2} c_{pi} d_{pi} \right) + \left(\sum_{i \in S3} c_{pi} d^*_{pi} - \sum_{i \in S3} c_{pi} d_{pi} \right) \\ & + \left(\sum_{i \in S2} \sum_{j \in S3} c_{ij} d^*_{ij} - \sum_{i \in S2} \sum_{j \in S3} c_{ij} d_{ij} \right). \end{aligned} \tag{3}$$

Compared to Π, the distance from each facility $i \in S1$ of Π^* to facility p is increased by $D2$, so the following relation is obtained:

$$gain1 = \sum_{i \in S1} c_{ip} d^*_{ip} - \sum_{i \in S1} c_{ip} d_{ip} = \sum_{i \in S1} c_{ip} D2. \tag{4}$$

Compared to Π, the distance from each facility $i \in S1$ of Π^* to each facility $j \in S2$ of Π^* is reduced by lp, so we know that:

$$gain2 = \sum_{i \in S1} \sum_{j \in S2} c_{ij} d^*_{ij} - \sum_{i \in S1} \sum_{j \in S2} c_{ij} d_{ij} = - \sum_{i \in S1} \sum_{j \in S2} c_{ij} lp. \tag{5}$$

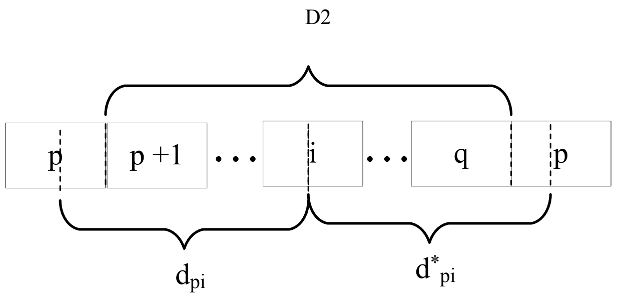

Figure 2. Relationship between d_{pi} and d^*_{pi} in $S2$

According to Figure 2, $d_{pi} + d^*_{pi} = D2 + lp$ can be got, and we have the following result:

$$\begin{aligned} gain3 &= \sum_{i \in S2} c_{pi} d^*_{pi} - \sum_{i \in S2} c_{pi} d_{pi} = \sum_{i \in S2} c_{pi}(D2 + lp - d_{pi}) - \sum_{i \in S2} c_{pi} d_{pi} \\ &= \sum_{i \in S2} c_{pi}(D2 + lp - 2d_{pi}). \end{aligned} \tag{6}$$

Compared to Π, the distance from each facility $i \in S3$ of Π^* to facility p is reduced by $D2$, so we can get:

$$gain4 = \sum_{i \in S3} c_{pi} d^*_{pi} - \sum_{i \in S3} c_{pi} d_{pi} = - \sum_{i \in S3} c_{pi} D2. \tag{7}$$

Compared to Π, the distance from each facility $i \in S2$ of Π^* to each facility $j \in S3$ of Π^* is increased by lp, so we have:

$$gain5 = \sum_{i \in S2} \sum_{j \in S3} c_{ij} d^*_{ij} - \sum_{i \in S2} \sum_{j \in S3} c_{ij} d_{ij} = \sum_{i \in S2} \sum_{j \in S3} c_{ij} lp. \tag{8}$$

If $p > q$, set $S1$ consists of facilities from position 1 to $(q-1)$, set $S2$ consists of facilities from position q to $(p-1)$, and set $S3$ consists of facilities from position $(p+1)$ to n. Using the same method presented above, we can deduce the following result:

$$gain1 = - \sum_{i \in S1} c_{ip} D2, \quad gain2 = \sum_{i \in S1} \sum_{j \in S2} c_{ij} lp,$$

$$gain3 = - \sum_{i \in S2} c_{pi}(D2 + lp - 2d_{pi}),$$

$$gain4 = \sum_{i \in S3} c_{pi} D2, \quad gain5 = - \sum_{i \in S2} \sum_{j \in S3} c_{ij} lp.$$

Finally, we can calculate the gain by the expression

$$gain = gain1 + gain2 + gain3 + gain4 + gain5.$$

It is obvious that the gain technique spends less time than the conventional formula calculation using Equation (1). It is important to emphasize that our technique is suitable for the random insertion operation in the first-improvement search.

3.4. **The architecture of the proposed algorithm.** In this section, we present the architecture of the multi-start local search algorithm called $MSLSA$. The pseudo-code is shown as Algorithm 3.

Algorithm 3 $MSLSA$

```
Require: An SRFLP instance with $n$ facilities.
Ensure: The best solution $S_{gbest}$.
 1: generate $S_{gbest}$ using Theorem 1 proposed in [22].
 2: set $flag \leftarrow TRUE.$
 3: while ($flag = TRUE$) do
 4:   set $flag \leftarrow FALSE.$
 5:   the initial seed permutation $\Pi \leftarrow S_{gbest}$
 6:   fill AM with solutions generated by $DIS$ algorithm.
 7:   for $i = 1$ to $L$ do
 8:     select the $i$th solution $S_i$ in AM
 9:     apply local search $FILS$ to improving $S_i$
10:     obtain an improved solution $S_i^*$
11:     if $(Z(S_i^*) < Z(S_i))$ then
12:       $S_i \leftarrow S_i^*$
13:       if $(Z(S_i^*) < Z(S_{gbest}))$ then
14:         $S_{gbest} \leftarrow S_i^*$
15:         set $flag \leftarrow TRUE.$
16:       end if
17:     end if
18:   end for
19: end while
20: output $S_{gbest}$
```

First an initial seed permutation is generated by Theorem 1 in [22], and the incumbent best solution S_{gbest} is set to the initial seed permutation. Then L elite and diverse solutions are generated from the seed by diversification generator *DIS*. They are stored in a memory (AM) and are used as multi-start points. At each iteration, the first-improvement local search *FILS* is applied to improving each solution S_i in AM. The improved solution of S_i is S_i^*. If the objective function value of S_i^* is lower than that of S_i, S_i is updated by S_i^*. When L solutions in AM are all explored by *FILS*, an iteration of *MSLSA* is finished. In this iteration, the incumbent best solution S_{gbest} is found and recorded. The algorithm is terminated when the incumbent best solution S_{gbest} has not been improved in an iteration. Otherwise, the restarting mechanism works: the improved solution S_{gbest} is accepted as a new seed permutation; the solutions of AM are regenerated from the new seed by *DIS* and next iteration restarts. After the *MSLSA* algorithm is terminated, a (near) global optimal solution can be derived from S_{gbest}.

4. Computational Experiments.

To evaluate the performance of the proposed algorithm, we present the results of computational experiments on benchmark instances that vary in size from 60 to 110 facilities. The instances include three series as follows:

(1) Anjos instances, first reported by Anjos et al. (2005), divided into 4 sets, each containing 5 instances, of sizes 60, 70, 75 and 80, respectively;

(2) sko instances, first reported by Anjos and Yen (2009), including four sets of five instances each of size 64, 72, 81, and 100;

(3) Amaral instances, proposed by Letchford and Amaral (2013), including a set of three instances of size 110.

These benchmark instances have been widely used to test the algorithms for SRFLP in the literature. Our algorithm was coded in Microsoft Visual C++ and run on a notebook PC equipped with Intel Core i3-2350M 2.30GHz CPU and 4 GB RAM. Since heuristics are probabilistic, it is usual to run implementations multiple times for each instance and report the best results from these runs, like 100 times in [23] and 200 times in [35]. We run implementations 50 times per instance. Based on initial experiments, the parameters of the *MSLSA* for the problem are set as follows: $L_1 = \lfloor 0.6n \rfloor$, $L_2 = \lfloor 0.4n \rfloor$.

4.1. Testing of gain technique.

The first experiment is aimed at quantifying the performance of gain technique for saving time. To evaluate the performance of the technique, we tested it on nine benchmark instances of sizes 60, 64, 70, 72, 75, 80, 81, 100 and 110. The *MSLSA* algorithm was run 50 times with cost calculations by Equation (1) and Equation (3) respectively. The average computation time is shown in Table 1.

Table 1. Comparison of execution time (second) for two cost calculations

Instance	Size	Conventional formula	Gain technique	Reduce
Anjos-60-05	60	34.80	11.39	67.27%
Anjos-70-05	70	87.66	26.07	70.26%
Anjos-75-05	75	137.58	46.40	66.27%
Anjos-80-05	80	186.50	59.94	67.86%
sko-64-05	64	76.42	24.02	68.56%
sko-72-05	72	179.63	60.21	66.48%
sko-81-05	81	338.21	110.10	67.44%
sko-100-05	100	1184.02	355.56	69.97%
Amaral-110-03	110	1747.43	529.71	69.68%

Table 1 presents a comparison of the computation time with two kinds of cost calculations: conventional formula calculation by Equation (1) and gain technique calculation by Equation (3). The first column records the name of the instance, with its size in the second column. Columns 3 and 4 refer to the computation time of the conventional formula and of the gain technique, respectively. The fifth column exhibits the percentage of the gain technique in reducing time when compared to the conventional formula calculation. It shows that the gain technique requires about 68% less time than the conventional formula calculation. These results clearly demonstrate the effectiveness of the speed up technique presented in Section 2.3.

4.2. Comparison on Anjos instances.

We conducted the second experiment to compare the *MSLSA* with three good and recently published algorithms on Anjos instances. They are the permutation-based genetic algorithm (PGA) by Datta et al. [24], the hybrid genetic algorithm (HGA) by Ozcelik [36], and the scatter search algorithm (SSA) by Kothari and Ghosh [32].

Table 2 presents the comparison of solution costs with *MSLSA* and the other three algorithms for the SRFLP on the 20 Anjos instances. The table identifies the instance by its name and size n. Then we provide the best layout costs of *MSLSA* and the other three approaches. The results in Table 2 show that the output costs of *MSLSA* are equal to the best costs in the literature for all 20 instances. Table 3 reveals the comparison of the average computational time with *MSLSA* and the other three algorithms. The results show our algorithm is obviously faster than other algorithms for these instances.

TABLE 2. Comparison of solution costs for Anjos instances

Instance	Size	PGA	HGA	SSA	*MSLSA*	Times
Anjos-60-01	60	1,477,834.0	1,477,834.0	1,477,834.0	1,477,834.0	21
Anjos-60-02	60	841,792.0	841,776.0	841,776.0	841,776.0	30
Anjos-60-03	60	648,337.5	648,337.5	648,337.5	648,337.5	15
Anjos-60-04	60	398,468.0	398,406.0	398,406.0	398,406.0	20
Anjos-60-05	60	318,805.0	318,805.0	318,805.0	318,805.0	26
Anjos-70-01	70	1,528,621.0	1,528,537.0	1,528,537.0	1,528,537.0	11
Anjos-70-02	70	1,441,028.0	1,441,028.0	1,441,028.0	1,441,028.0	16
Anjos-70-03	70	1,518,993.5	1,518,993.5	1,518,993.5	1,518,993.5	17
Anjos-70-04	70	968,796.0	968,796.0	968,796.0	968,796.0	17
Anjos-70-05	70	4,218,017.5	4,218,002.5	4,218,002.5	4,218,002.5	22
Anjos-75-01	75	2,393,456.5	2,393,456.5	2,393,456.5	2,393,456.5	12
Anjos-75-02	75	4,321,190.0	4,321,190.0	4,321,190.0	4,321,190.0	7
Anjos-75-03	75	1,248,537.0	1,248,423.0	1,248,423.0	1,248,423.0	6
Anjos-75-04	75	3,941,891.5	3,941,816.5	3,941,816.5	3,941,816.5	15
Anjos-75-05	75	1,791,408.0	1,791,408.0	1,791,408.0	1,791,408.0	16
Anjos-80-01	80	2,069,097.5	2,069,097.5	2,069,097.5	2,069,097.5	9
Anjos-80-02	80	1,921,177.0	1,921,136.0	1,921,136.0	1,921,136.0	29
Anjos-80-03	80	3,251,368.0	3,251,368.0	3,251,368.0	3,251,368.0	11
Anjos-80-04	80	3,746,515.0	3,746,515.0	3,746,515.0	3,746,515.0	19
Anjos-80-05	80	1,588,901.0	1,588,885.0	1,588,885.0	1,588,885.0	5

4.3. **Comparison on sko instances.** The third computational study is carried out to make the comparison on sko instances. The sko instances are larger and more complicated than Anjos instances. Ozcelik applied a hybrid genetic algorithm (HGA) to tackling these instances. However, it gave the experiment results only with the instances of size 100 [36]. Kothari and Ghosh had carried out several computational studies with good experiment results on these instances by different approaches, such as the scatter search algorithm (SSA) [32], the Lin-Kernighan heuristic (LKH) [33] and the efficient genetic algorithm (GENALGO) [35]. So we carry out a computational study to compare the *MSLSA* with these three approaches of Kothari and Ghosh.

The structure of Table 4 is similar to Table 2. The value marked in boldface indicates the best known solution. From Table 4, the results show that for 19 of the 20 instances, the costs of the solutions output by *MSLSA* match the best costs for these instances in the literature. For the sko-100-03 instance, the cost output by *MSLSA* is marginally worse than the best cost in GENALGO.

We next compare the average computational time required by *MSLSA* and the algorithms of Kothari and Ghosh on the sko instances. To make the different CPU comparable, we evaluate the performance of CPU by mark from http://www.cpubenchmark.net, according to the CPU specifications of the related literature.

TABLE 3. Comparison of execution time (second) for Anjos instances

Instance	Size	PGA	HGA	SSA	*MSLSA*
Anjos-60-01	60	19.540	11.642	46.73	11.22
Anjos-60-02	60	22.340	15.265	41.54	11.68
Anjos-60-03	60	68.810	45.125	50.4	17.24
Anjos-60-04	60	20.710	39.285	53.52	12.17
Anjos-60-05	60	26.410	12.803	52.35	11.39
Anjos-70-01	70	64.830	66.439	82.61	27.80
Anjos-70-02	70	77.490	40.708	88.16	24.67
Anjos-70-03	70	68.260	33.820	83.17	25.22
Anjos-70-04	70	100.590	64.399	102.26	35.14
Anjos-70-05	70	60.480	26.148	81.59	26.07
Anjos-75-01	75	125.260	106.495	127.66	41.83
Anjos-75-02	75	128.950	135.831	118.01	41.95
Anjos-75-03	75	157.950	72.470	143.69	45.44
Anjos-75-04	75	119.920	76.394	125.37	35.62
Anjos-75-05	75	101.670	105.650	138.71	46.40
Anjos-80-01	80	75.410	156.687	162.59	139.47
Anjos-80-02	80	68.750	66.286	174.8	113.81
Anjos-80-03	80	85.900	124.992	181.84	71.62
Anjos-80-04	80	77.810	153.003	148.41	68.90
Anjos-80-05	80	196.510	153.003	199.79	59.94

TABLE 4. Comparison of solution costs for sko instances

Instance	Size	LKH	GENALGO	SSA	*MSLSA*	Times
sko-64-01	64	96,933.0	**96,881.0**	96,883.0	**96,881.0**	4
sko-64-02	64	634,338.5	**634,332.5**	**634,332.5**	**634,332.5**	10
sko-64-03	64	**414,323.5**	**414,323.5**	**414,323.5**	**414,323.5**	11
sko-64-04	64	297,205.0	**297,129.0**	**297,129.0**	**297,129.0**	10
sko-64-05	64	**501,922.5**	**501,922.5**	**501,922.5**	**501,922.5**	9
sko-72-01	72	**139,150.0**	**139,150.0**	**139,150.0**	**139,150.0**	6
sko-72-02	72	712,005.0	**711,998.0**	**711,998.0**	**711,998.0**	18
sko-72-03	72	**1,054,110.5**	**1,054,110.5**	**1,054,110.5**	**1,054,110.5**	8
sko-72-04	72	919,635.5	**919,586.5**	**919,586.5**	**919,586.5**	12
sko-72-05	72	428,879.5	**428,226.5**	**428,226.5**	**428,226.5**	6
sko-81-01	81	205,166.0	205,108.5	**205,106.0**	**205,106.0**	2
sko-81-02	81	**521,391.5**	**521,391.5**	**521,391.5**	**521,391.5**	17
sko-81-03	81	970,862.0	**970,796.0**	**970,796.0**	**970,796.0**	9
sko-81-04	81	2,031,979.0	**2,031,803.0**	**2,031,803.0**	**2,031,803.0**	5
sko-81-05	81	1,303,805.0	**1,302,711.0**	**1,302,711.0**	**1,302,711.0**	16
sko-100-01	100	378,614.0	**378,234.0**	**378,234.0**	**378,234.0**	6
sko-100-02	100	2,076,048.5	**2,076,008.5**	**2,076,008.5**	**2,076,008.5**	5
sko-100-03	100	16,148,818.0	**16,145,598.0**	16,145,614.0	16,145,614.5	11
sko-100-04	100	3,232,740.0	**3,232,522.0**	3,232,531.0	**3,232,522.0**	12
sko-100-05	100	1,033,345.5	**1033,080.5**	**1033,080.5**	**1033,080.5**	4

Table 5. Comparison of execution time (second) for sko instances

Instance	Size	LKH	GENALGO	SSA	*MSLSA*
sko-64-01	64	196.98	15.42	82.80	37.55
sko-64-02	64	189.84	14.05	68.31	31.64
sko-64-03	64	185.22	14.16	80.33	32.50
sko-64-04	64	190.68	14.35	91.05	32.69
sko-64-05	64	182.70	14.00	75.86	24.02
sko-72-01	72	465.12	25.45	115.21	66.12
sko-72-02	72	419.04	22.91	192.12	52.54
sko-72-03	72	412.80	22.40	114.16	48.17
sko-72-04	72	418.56	22.23	135.46	50.68
sko-72-05	72	433.44	23.19	116.50	60.21
sko-81-01	81	1,298.16	40.77	311.63	160.93
sko-81-02	81	1,402.38	36.63	226.91	78.88
sko-81-03	81	1,276.56	35.93	241.50	137.22
sko-81-04	81	1,425.06	36.85	193.98	109.79
sko-81-05	81	1,301.40	35.58	287.22	110.10
sko-100-01	100	7,279.14	99.75	877.12	518.03
sko-100-02	100	7,632.90	86.32	526.09	346.39
sko-100-03	100	7,510.80	82.94	599.92	317.32
sko-100-04	100	7,395.96	81.51	592.00	285.90
sko-100-05	100	7,087.08	85.37	591.75	355.60
Average execution time		2,335.19	40.49	275.99	142.81
CPU Mark		1,261	6,494	6,494	2,655

In Table 5, our computer is approximately 2.1 times faster than the computer used in LKH, whose average execution time is 16.3 times more than our algorithm. The computer used in SSA is approximately 2.4 times faster than ours, and the average execution time is 1.9 times more than *MSLSA*. So the execution speed of *MSLSA* is faster than LKH and SSA. The average execution time that we use is approximately 3.5 times more than the one used in GENALGO, whose CPU speed is 2.4 times faster than ours. The execution speed of *MSLSA* is marginally slower than GENALGO.

4.4. **Comparison on Amaral instances.** The final comparison with other algorithms was carried out on Amaral instances. In the published literature, a hybrid genetic algorithm (HGA) proposed by Ozcelik had the lowest cost. The efficient genetic algorithm (GENALGO) proposed by Kothari and Ghosh had the least computation time [35]. Therefore, HGA, SSA and GENALGO are compared with *MSLSA* in this computational experiment. In Table 6, we present the costs of the best permutations obtained by *MSLSA* and compare them with the results of the three algorithms. In Table 7, we make the comparison between *MSLSA* and other algorithms regarding average computational time.

Table 6. Comparison of solution costs for Amaral instances

Instance	Size	HGA	GENALGO	SSA	*MSLSA*	Times
Amaral-110-01	110	**144,296,664.5**	144,296,768.0	144,297,440.0	**144,296,664.5**	14
Amaral-110-02	110	**86,050,037.0**	86,050,112.0	86,050,208.0	**86,050,037.0**	7
Amaral-110-03	110	**2,234,743.5**	**2,234,743.5**	2,234,798.5	**2,234,743.5**	11

TABLE 7. Comparison of execution time (second) for Amaral instances

Instance	Size	HGA	GENALGO	SSA	*MSLSA*
Amaral-110-01	110	2,584.62	191.99	975.11	519.81
Amaral-110-02	110	2,396.77	190.67	680.10	581.09
Amaral-110-03	110	2,332.70	119.57	811.08	529.71
Average execution time		2,438.03	167.41	822.10	543.53
CPU Mark		1,747	6,494	6,494	2,655

We can make the following observations about the results in Tables 6 and 7:

(1) In terms of solution quality, the costs of the best permutations obtained by *MSLSA* are the same as those of the best costs known in the literature for all instances in this set. Our proposed algorithm found better solutions than GENALGO and SSA.

(2) Our computer is approximately 1.5 times faster than the computer used in HGA, whose average execution time is 4.4 times more than our algorithm. The computer used in SSA is approximately 2.4 times faster than ours, and the average execution time is 1.5 times more than *MSLSA*. So the execution speed of *MSLSA* is faster than HGA and SSA. The average execution time that we use is approximately 3.2 times more than the one used in GENALGO, whose CPU speed is 2.4 times faster than what we use. The execution speed of *MSLSA* is marginally slower than GENALGO.

Through a series of experiments, we know that Anjos instances are easier than sko instances and Amaral instances. In order to show the advantage of our proposed algorithm over extant ones visually, we plot a figure based on the data of 23 hard instances from sko-64-01 to Amaral-110-03. Figure 3 compares the percentage deviation of the best objective value (BOV) to the current best known result (BKR) for GENALGO, SSA and *MSLSA*. The percentage deviation is shown in Equation (9). From Figure 3, it can be seen that the

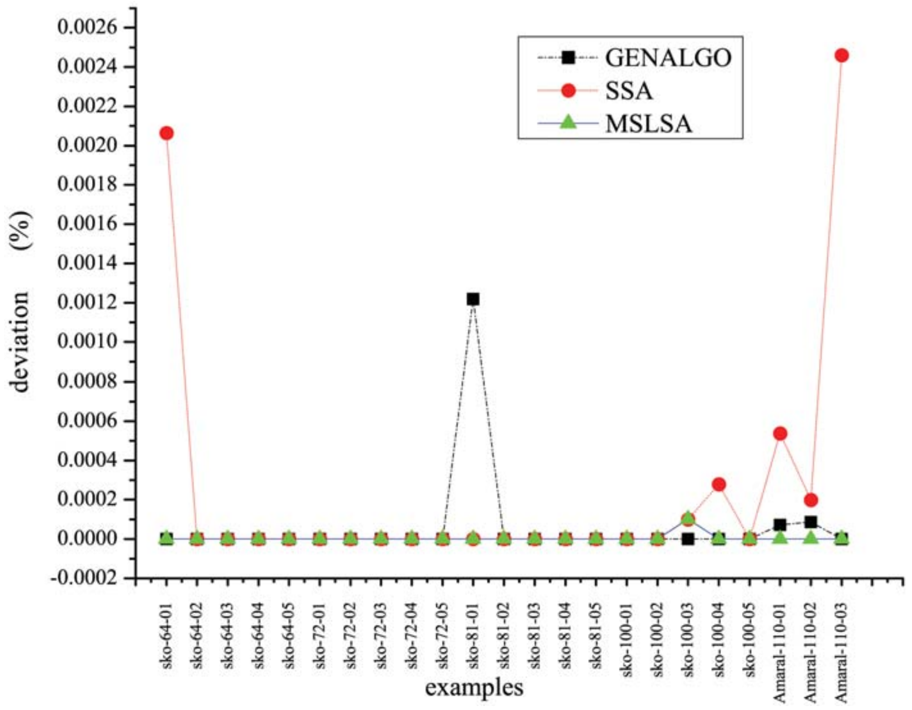

FIGURE 3. The percentage deviation of the best objective value to the current best known result

best objective values of *MSLSA* are closer than those of GENALGO and SSA to the best known values. Based on the results in this figure, the conclusion is that *MSLSA* exhibits a better performance than both GENALGO and SSA in terms of solution quality.

$$deviation = (BOV - BKR)/BKR * 100\%. \tag{9}$$

The above observations show that *MSLSA* has a good performance in terms of solution quality and time restriction for solving large size SRFLP instances.

5. Conclusions.

In this paper, we have presented a multi-start local search algorithm *MSLSA* with diversification and intensification for solving the SRFLP. A diversification generator is designed to diversify the initial elite solutions. Afterward, the proposed algorithm employs a first-improvement local search procedure *FILS* to intensify the quality of solutions. *FILS* probes the neighbors in random order and its randomness can increase the diversification, thus exploring broader solution areas. Moreover, the most important feature of *MSLSA* is that a gain technique is applied to speeding up the cost calculations of neighbors. The restarting mechanism repeatedly calls the diversification generator to drive the search into a new round of local search phase. Additionally, the algorithm *MSLSA* is very simple and easy to be implemented. Computational experiments showed that *MSLSA* quickly attained the optimal solution for 42 of 43 classical instances in the literature. Compared with the conventional formula calculation, the gain technique can save more than 66% of the computational time. Compared with other algorithms, the proposed algorithm is very competitive in solution quality and computation time.

The current study is limited in that it uses one type of neighborhood structure. This has the advantage of making it easy to implement, but more neighborhood structures may yield better results. So, as further research, we intend to test the performance of heuristic *MSLSA* with different neighborhood structures. We also try to solve other layout problems with heuristic *MSLSA*.

Acknowledgment. The authors are grateful to the anonymous referees for their constructive comments that have greatly improved the presentation of this paper. This research was supported partially by the National Natural Science Foundation of China under Grant 11301255, Natural Science Foundation of Fujian Province of China under Grant 2016J01025, and the Science and Technology Project of Minjiang University, China, under Grant MYK15005.

REFERENCES

[1] D. M. Simmons, One-dimensional space allocation: An ordering algorithm, *Operations Research*, vol.17, no.5, pp.812-826, 1969.

[2] J. C. Picard and M. Queyranne, On the one-dimensional space allocation problem, *Operations Research*, vol.29, no.2, pp.371-391, 1981.

[3] S. S. Heragu and A. Kusiak, Machine layout problem in flexible manufacturing systems, *Operations Research*, vol.36, no.2, pp.258-268, 1988.

[4] S. G. Ponnambalam and V. Ramkumar, A genetic algorithm for the design of a single-row layout in automated manufacturing systems, *The International Journal of Advanced Manufacturing Technology*, vol.18, no.7, pp.512-519, 2001.

[5] R. Kothari and D. Ghosh, The single row facility layout problem: State of the art, *Opsearch*, vol.49, no.4, pp.442-462, 2012.

[6] A. R. S. Amaral, On the exact solution of a facility layout problem, *European Journal of Operational Research*, vol.173, no.2, pp.508-518, 2006.

[7] A. R. S. Amaral, An exact approach to the one-dimensional facility layout problem, *Operations Research*, vol.56, no.4, pp.1026-1033, 2008.

[8] S. S. Heragu and A. Kusiak, Efficient models for the facility layout problem, *European Journal of Operational Research*, vol.53, no.1, pp.1-13, 1991.

[9] R. Love and J. Wong, On solving a one-dimensional space allocation problem with integer programming, *INFOR*, vol.14, no.2, pp.139-144, 1976.

[10] P. Kouvelis and W. C. Chiang, Optimal and heuristic procedures for row layout problems in automated manufacturing systems, *Journal of the Operational Research Society*, vol.47, no.6, pp.803-816, 1996.

[11] A. R. S. Amaral and A. N. Letchford, A polyhedral approach to the single row facility layout problem, *Mathematical Programming*, vol.141, nos.1-2, pp.453-477, 2013.

[12] M. F. Anjos, A. Kennings and A. Vannelli, A semidefinite optimization approach for the single-row layout problem with unequal dimensions, *Discrete Optimization*, vol.2, no.2, pp.113-122, 2005.

[13] M. F. Anjos and A. Vannelli, Computing globally optimal solutions for single-row layout problems using semidefinite programming and cutting planes, *Informs Journal on Computing*, vol.20, no.4, pp.611-617, 2008.

[14] M. F. Anjos and G. Yen, Provably near-optimal solutions for very large single-row facility layout problems, *Optimization Methods and Software*, vol.24, nos.4-5, pp.805-817, 2009.

[15] P. Hungerländer and F. Rendl, A computational study and survey of methods for the single-row facility layout problem, *Computational Optimization and Applications*, vol.55, no.1, pp.1-20, 2013.

[16] M. Beghin-Picavet and P. Hansen, Deux problèmes daffectation non linéaires, *RAIRO – Operations Research Opérationnelle*, vol.16, no.3, pp.263-276, 1982.

[17] S. S. Heragu and A. S. Alfa, Experimental analysis of simulated annealing based algorithms for the layout problem, *European Journal of Operational Research*, vol.57, no.2, pp.190-202, 1992.

[18] A. G. de Alvarenga, F. J. Negreiros-Gomes and M. Mestria, Metaheuristic methods for a class of the facility layout problem, *Journal of Intelligent Manufacturing*, vol.11, no.4, pp.421-430, 2000.

[19] M. Solimanpur, P. Vrat and R. Shankar, An ant algorithm for the single row layout problem in flexible manufacturing systems, *Computers & Operations Research*, vol.32, no.3, pp.583-598, 2005.

[20] R. M. S. Kumar, P. Asokan, S. Kumanan and B. Varma, Scatter search algorithm for single row layout problem in fms, *Advances in Production Engineering & Management*, vol.3, no.4, pp.193-204, 2008.

[21] H. Samarghandi, P. Taabayan and F. F. Jahantigh, A particle swarm optimization for the single row facility layout problem, *Computers & Industrial Engineering*, vol.58, no.4, pp.529-534, 2010.

[22] H. Samarghandi and K. Eshghi, An efficient tabu algorithm for the single row facility layout problem, *European Journal of Operational Research*, vol.205, no.1, pp.98-105, 2010.

[23] R. Kothari and D. Ghosh, Tabu search for the single row facility layout problem using exhaustive 2-opt and insertion neighborhoods, *European Journal of Operational Research*, vol.224, no.1, pp.93-100, 2013.

[24] D. Datta, A. R. S. Amaral and J. R. Figueira, Single row facility layout problem using a permutation-based genetic algorithm, *European Journal of Operational Research*, vol.213, no.2, pp.388-394, 2011.

[25] C. Ou-Yang and A. Utamima, Hybrid estimation of distribution algorithm for solving single row facility layout problem, *Computers & Industrial Engineering*, vol.66, no.1, pp.95-103, 2013.

[26] G. Palubeckis, Fast local search for single row facility layout, *European Journal of Operational Research*, vol.246, no.3, pp.800-814, 2015.

[27] A. R. S. Amaral, *Enhanced Local Search Applied to the Single-Row Facility Layout Problem*, http://www.din.uem.br/sbpo/sbpo2008/pdf/arq0026.pdf, 2008.

[28] J. Pacheco, F. Ángel-Bello and A. Álvarez, A multi-start tabu search method for a single-machine scheduling problem with periodic maintenance and sequence-dependent set-up times, *Journal of Scheduling*, vol.16, no.6, pp.661-673, 2013.

[29] M. Alfaki and D. Haugland, A cost minimization heuristic for the pooling problem, *Annals of Operations Research*, vol.222, no.1, pp.73-87, 2014.

[30] W. Li, Seeking global edges for traveling salesman problem in multi-start search, *Journal of Global Optimization*, vol.51, no.3, pp.515-540, 2011.

[31] M. Toril, V. Wille, I. Molina-Fernández and C. Walshaw, An adaptive multi-start graph partitioning algorithm for structuring cellular networks, *Journal of Heuristics*, vol.17, no.5, pp.615-635, 2011.

[32] R. Kothari and D. Ghosh, A scatter search algorithm for the single row facility layout problem, *Journal of Heuristics*, vol.20, no.2, pp.125-142, 2014.

[33] R. Kothari and D. Ghosh, Insertion based Lin-Kernighan heuristic for single row facility layout, *Computers & Operations Research*, vol.40, no.1, pp.129-136, 2013.

[34] T. Stüzle, An ant approach to the flow shop problem, *Proc. of the 6th European Congress on Intelligent Techniques & Soft Computing*, vol.3, pp.1560-1564, 1998.

[35] R. Kothari and D. Ghosh, An efficient genetic algorithm for single row facility layout, *Optimization Letters*, vol.8, no.2, pp.679-690, 2014.

[36] F. Ozcelik, A hybrid genetic algorithm for the single row layout problem, *International Journal of Production Research*, vol.50, no.20, pp.5872-5886, 2012.