Computers & Industrial Engineering 58 (2010) 529–534

Contents lists available at ScienceDirect

Computers & Industrial Engineering

journal homepage: www.elsevier.com/locate/caie

# A particle swarm optimization for the single row facility layout problem

Hamed Samarghandi [a,*], Pouria Taabayan [b], Farzad Firouzi Jahantigh [c]

[a] Department of Mechanical Engineering, University of Manitoba, Winnipeg, Manitoba, Canada
[b] Department of Management, Iran University of Science and Technology, Tehran, Iran
[c] Department of Industrial Engineering, University of Sistan and Baluchestan, Zahedan, Iran

ARTICLE INFO

*Article history:*
Received 25 July 2009
Received in revised form 25 November 2009
Accepted 26 November 2009
Available online 29 November 2009

*Keywords:*
Facility planning and design
Linear ordering problem
Particle swarm optimization
Factoradics

ABSTRACT

Single Row Facility Layout Problem (SRFLP) consists of arranging a number of rectangular facilities with varying length on one side of a straight line to minimize the weighted sum of the distance between all facility pairs. In this paper we use a Particle Swarm Optimization (PSO) algorithm to solve the SRFLP. We first employ a new coding and decoding technique to efficiently map discrete feasible space of the SRFLP to a continuous space. The proposed PSO will further use this coding technique to explore the continuous solution space. Afterwards, the algorithm decodes the solutions to its respective feasible solution in the discrete feasible space and returns the solutions. Computational results on benchmark problems show the efficiency of the proposed algorithm compared to other heuristics.



## 1. Introduction

Facility Layout Planning (FLP) problems are concerned with the arrangement of a number of facilities in a given space to satisfy an objective function; for example, minimizing total interaction. Single Row Facility Layout Problem (SRFLP) is a special case of the FLP that consists of finding an optimum linear placement of rectangular facilities with varying length on one side of a straight line. Officially, SRFLP can be described as follows: there are $n$ rectangular facilities to be arranged on one side of a straight line in a given direction. Parameters of the problem are the length $l_i$ of each facility $i$, and a $n \times n$ matrix $C = [c_{ij}]$ where $c_{ij}$ is the flow between facilities $i$ and $j$. The distance between each pair of facilities is calculated as the distance between their centers. Eventually, the objective of the problem is to arrange the facilities to minimize the weighted sum of the distance between all facility pairs.

The SRFLP is proven to be a NP-Complete problem (Suresh & Sahu, 1993). Therefore, finding optimum solution of large and practical instances of SRFLP is not possible in reasonable time unless $P = NP$. However, SRFLP has a number of practical applications including the following: arrangement of books on a shelf in a library; arrangement of departments on one side of a corridor in an office building, a hospital or a supermarket (Simmons, 1969); arrangement of machines on one side of a straight line path traveled by an automated guided vehicle (Heragu & Kusiak, 1988; Solimanpur, Vrat, & Shankar, 2005); assignment of disk cylinders to files (Picard & Queyranne, 1981). The above reasons have attracted many researchers to the problem and numerous papers are available in the literature for the SRFLP. For a review on proposed exact methods, reader is referred, for example, to Amaral (2006, 2009), Anjos and Vannelli (2008), and Heragu and Kusiak (1991). In this paper we use a framework for mapping a discrete feasible space to a continuous feasible space. This framework has been previously used in Behroozi (2009). Then we will use a PSO algorithm to search for good-quality solutions of the SRFLP.

The paper structure is summarized as follows. Section 2 consists of a literature review of the related researches in this area. Section 3 describes the problem model and its feasible solutions. Factoradic base, coding, and decoding algorithm are presented in Section 4. Section 5 describes our proposed algorithm and its building blocks. Section 6 includes the stepwise algorithm. Section 7 gives the computational results. Finally, Section 8 contains the conclusions.

## 2. Literature review

Theoretical attractiveness and practical applications have created a rich and growing literature for the SRFLP. In this section we only tend to review heuristic and meta-heuristic approaches suggested to solve the SRFLP.

Neghabat presented a constructive algorithm to obtain a complete solution by adding one machine at a time to the end of the current solution (Neghabat, 1974). Drezner proposed a heuristic method based on the eigenvectors of a transformed flow matrix (Drezner, 1987). Heragu and Kusiak developed a linear mixed-integer formulation of the SRFLP and solved it using a penalty

* Corresponding author. Tel.: +1 2048991362.
*E-mail addresses:* umsamarg@cc.umanitoba.ca (H. Samarghandi), taabayan@ideaonline.ir (P. Taabayan), firouzi@eng.usb.ac.ir (F.F. Jahantigh).

0360-8352/$ - see front matter © 2009 Elsevier Ltd. All rights reserved.
doi:10.1016/j.cie.2009.11.015

technique (Heragu & Kusiak, 1991). In 1992, Heragu and Alfa proposed a simulated annealing algorithm to solve the SRFLP (Heragu & Alfa, 1992).

Braglia developed a hybrid algorithm based on simulated annealing and genetic algorithms to minimize the total backtracking in the linear ordering of facilities (Braglia, 1996). Kumar et al. proposed a constructive greedy heuristic in 1995 (Kumar, Hadjinicola, & Lin, 1995). Fico et al. developed a genetic algorithm able to construct both single and multiple row layouts (Ficko, Brezocnik, & Balic, 2004). Solimanpur et al. presented a new mixed integer model and an ant algorithm for the SRFLP (Solimanpur et al., 2005).

Anjos et al. has constructed a semi-definite programming relaxation providing a lower bound on the optimum value of the SRFLP (Anjos, Kennings, & Vannelli, 2005). Recently proposed meta-heuristic methods include a scatter search algorithm by Kumar and et al. (2008), a hybrid algorithm based on ant colony optimization and PSO by Teo and Ponnambalam (2008), and a genetic algorithm by Lin (2009).

This paper, first, describes feasible solutions of the SRFLP. Afterwards, factoradic coding/decoding method is described to make it possible to present each feasible solution by a number. A PSO algorithm will be developed thereafter. A different version of this algorithm has been introduced in Behroozi (2009). This algorithm employs the factoradic coding/decoding technique to search the feasible space. PSO, a stochastic and population based algorithm, belongs to the class of direct search methods and is used to find an optimal solution to an objective function in a feasible space. PSO is rather a new meta-heuristic approach, first presented in Eberhart and Kennedy (1995), Kennedy and Eberhart (1997). The proposed PSO is very efficient and capable of constructing good-quality solutions in a very short time.

## 3. Problem modeling and description

ABSMODEL, introduced by Heragu and Kusiak (1991), is a simple yet effective modeling of SRFLP. ABSMODEL is as follows:

$$\min z = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} c_{ij} d_{ij} \tag{1}$$

s.t.:

$$d_{ij} \geqslant \frac{1}{2}(l_i + l_j) + s_{ij}; \quad i = 1,2,\ldots,n-1; \quad j = i+1,\ldots,n \tag{2}$$

$$d_{ij} \geqslant 0; \quad i = 1,2,\ldots,n-1; \quad j = i+1,\ldots,n \tag{3}$$

In this model, distance between the centers of the facilities $i$ and $j$ is represented by $d_{ij}$ while $s_{ij}$, another parameter of the model, is the necessary clearance between the two facilities. Heragu and Kusiak defined $d_{ij}$ as follows:

$$d_{ij} = \frac{l_i + l_j}{2} + D_{ij} \tag{4}$$

$D_{ij}$ is the space between facilities $i$ and $j$. $D_{ij}$ is not necessarily equal to $s_{ij}$. If facility $k$ is placed between facilities $i$ and $j$ with $s_{ij} = 0$ then $D_{ij} = l_k$. The proposed algorithm uses (1) to compare different solutions obtained during feasible space exploration.

From a different point of view, if $\Pi_n$ is considered as the set of all permutations $\pi$ of $N = \{1,2,\ldots,n\}$, the SRFLP can be formulated as (Amaral, 2006):

$$\min \left\{ \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} c_{ij} d_{ij}^{\pi} | \pi \in \Pi_n \right\} \tag{5}$$

where $d_{ij}^{\pi}$ is the distance between facilities $i$ and $j$ with respect to the permutation $\pi$. New modeling of the SRFLP in (5) implies that the ABSMODEL searches the set of all permutations of numbers $1, 2, \ldots, n$ to find the permutation which minimizes the objective function. The idea behind the proposed PSO algorithm is to map the set of all permutations ($\Pi_n$) into a set of factoradic base numbers with a one-to-one mapping and efficiently explore this newly generated set to find a good-quality solution for the SRFLP. Then the algorithm uses the inverse mapping to map each factoradic number to a unique member of the set of permutations. Afterwards, this permutation will be returned as the final solution for the SRFLP.

## 4. Factoradic base

Factoradic is a specially constructed number system. Factoradics provide a lexicographical index for permutations (Knuth, 1997). The idea of the factoradic is closely linked to that of the Lehmer code (Knuth, 1997). Factoradic is a factorial-based mixed radix numeral system: the $i$th digit from right side is to be multiplied by $i!$. In this numbering system, the rightmost digit is always 0, the second 0, or 1, the third 0, 1 or 2 and so on (Knuth, 1997). For instance, 38 in decimal base can be shown as $(1_4 2_3 1_2 0_1 0_0)$ in factoradic base.

$$(1_4 2_3 1_2 0_1 0_0) = 1 \times 4! + 2 \times 3! + 1 \times 2! = 38_{10} \tag{6}$$

The factoradic numbering system is unambiguous. No number can be represented in more than one way because the sum of consecutive factorials multiplied by their index is always the next factorial minus one (Knuth, 1997):

$$\sum_{i=0}^{n} i \times i! = (n+1)! - 1 \tag{7}$$

More detailed information about factoradic base and factoradic numbering system can be found in Behroozi (2009), Knuth (1997), Doliskani, Malekian, and Zakerolhosseini (2008), McCaffrey (2003).

### 4.1. Relation between factoradic base and permutations

There is a natural mapping between the integers $0, 1, \ldots, n! - 1$ (or equivalently the factoradic numbers with $n$ digits) and the permutations of $n$ elements in lexicographical order, when the integers are expressed in factoradic form. This mapping has been termed the Lehmer code. For example, with $n = 3$, this mapping is shown in Table 1.

For mapping factoradic numbers into permutations and vice versa, two straightforward algorithms are presented in McCaffrey (2003) and further used in Behroozi (2009). Computational complexity of these algorithms are $O(n)$ which makes the algorithms able to efficiently map the permutations to factoradic numbers, factoradic numbers to decimal numbers and vice versa. These two algorithms are presented as Algorithm 1 and Algorithm 2. Table 2 demonstrates the mapping of $(1_2 1_1 0_0) \rightarrow (2 - 3 - 1)$ based on Algorithm 1.

**Table 1**
Natural mapping between factoradic numbers and permutations when $n = 3$.

| Decimal | Factoradic | Permutation |
|---|---|---|
| $0_{10}$ | $0_2 0_1 0_0$ | 1, 2, 3 |
| $1_{10}$ | $0_2 1_1 0_0$ | 1, 3, 2 |
| $2_{10}$ | $1_2 0_1 0_0$ | 2, 1, 3 |
| $3_{10}$ | $1_2 1_1 0_0$ | 2,3,1 |
| $4_{10}$ | $2_2 0_1 0_0$ | 3,1,2 |
| $5_{10}$ | $2_2 1_1 0_0$ | 3,2,1 |

**Table 2**
Mapping $(1_2 1_1 0_0) \rightarrow (2-3-1)$ based on Algorithm 1.

| Iteration | $k=1$ | $k=2$ | $k=3$ |
|---|---|---|---|
| Factoradic | 1 | 1 | 0 |
| $F$ | {0, 1, 2} | {0, 1, 2} | {0, 1, 2} |
| $P$ | {1, 2, 3} | {1, 3} | {1} |
| Permutation | 2 | 3 | 1 |

Algorithm 1 Mapping factoradic to permutation.
Step 1: Consider a list of possible digits of factoradic base in ascending order $f = \{0, 1, \ldots, n-1\}$, as well as a list of possible numbers in permutation $p = \{1, 2, \ldots, n\}$
Step 2: For $k = 1, 2, \ldots, n$: choose the $k$th digit in factoradic representation, and find this digit in $f$. Suppose that this digit is the $i$th digit in $f$. Choose the $i$th element of $p$, set this element as the $k$th element of permutation, and remove this element from $p$

Note that not only the described procedure is useful in our proposed PSO, but also defines a useful framework for mapping a discrete feasible space to a continuous space by a one-to-one mapping.

Algorithm 2 Mapping permutation to factoradic.
Step 1: Consider a list of possible digits of factoradic base in ascending order $f = \{0, 1, \ldots, n-1\}$, as well as a list of possible numbers in permutation $p = \{1, 2, \ldots, n\}$
Step 2: For $k = 1, 2, \ldots, n$, choose the $k$th digit in permutation, and find this digit in $p$. Suppose that this digit is the $i$th digit in $p$. Choose the $i$th element of $f$, set this element as the $k$th element of factoradic, and remove this element from $p$

Table 3 demonstrates the mapping of $(2-3-1) \rightarrow (1_2 1_1 0_0)$ based on Algorithm 2.

## 5. Proposed PSO algorithm

PSO algorithm has been widely used by researchers to solve combinatorial optimization problems since its introduction in Eberhart and Kennedy (1995), Kennedy and Eberhart (1997). In PSO a number of particles are moved in search space through a systematic approach. In PSO, at time $t$, each particle $i$ has a position, $x_i(t)$, and a velocity, $v_i(t)$. Current position of the particles as well as their best ever position are stored in a memory. Velocity of the particles will be changed based on the historical information stored in the memory and also random information. The new velocities will be used to update the current position of the particles, where their new objective function will be evaluated. Since historical data is used in updating the particle velocity, particles tend to return to their historical best position which results in early convergence. To overcome this unwanted phenomena, different velocity update techniques have been developed. The proposed PSO uses one of the most successful functions available to update the particle velocity. Rest of this section describes the building blocks of the proposed algorithm.

**Table 3**
Mapping $(2-3-1) \rightarrow (1_2 1_1 0_0)$ based on Algorithm 1.

| Iteration | $k=1$ | $k=2$ | $k=3$ |
|---|---|---|---|
| Permutation | 2 | 3 | 1 |
| P | {1, 2, 3} | {1, 3} | {1} |
| F | {0, 1, 2} | {0, 1, 2} | {0, 1, 2} |
| Factoradic | 1 | 1 | 0 |

### 5.1. Initial solutions

The proposed algorithm needs a number of initial solutions to initiate solution space exploration. These initial solutions are basically the particles used during search. Since no particle is born or destroyed during the search, the number of initial solutions is exactly equal to the number of particles of the algorithm during its exploration. This number is a parameter set by the user and is called $l$ in the rest of the paper.

The proposed algorithm generates $l$ random permutations, or equally $l$ random integer numbers between $[0, n!-1]$ as the initial solutions and refers to them as $x_i; \quad i = 1, 2, \ldots, l$ where $n$ is the number of facilities in the problem instance. Using random permutations as initial solutions guarantees a good diversification of the particles. And generating random integer numbers in the mentioned interval assures that the search initiates in feasible space.

Note that based on the one-to-one mapping described in Section 4.1, a random integer number in the mentioned interval is identical to a random permutation of $n$ facilities which is a feasible solution of the SRFLP. Therefore, these two words can be used interchangeably during the rest of the paper.

After generating the initial random integer numbers, the algorithm maps each number to its respective permutation and calculates its objective function by the objective function of the ABSMODEL given in (1).

### 5.2. Particle velocity, neighborhood structure, and stopping criterion

Since the proposed PSO algorithm uses $l$ particles to explore the feasible space, $l$ particle velocities are also needed to update each particle's position during iterations of the algorithm. The proposed algorithm initially generates $l$ random integer numbers as particle velocity in order to update the particle's position. Note that velocities must be in an appropriate interval so that the particles remain in feasible space after being updated. Since the feasible solution interval is $[0, n!-1]$, the appropriate velocity interval which guarantees feasibility of each particle after update in $k$th iteration is $[-x_i^k, (n!-1) - x_i^k]$ for each particle $i = 1, 2, \ldots, l$.

Algorithm must also modify particle velocities during the search to guide the particles through the more desirable areas of the feasible region. Originally, PSO algorithm uses (8) to update velocities (Eberhart & Kennedy, 1995; Kennedy & Eberhart, 1997):

$$\overrightarrow{v_i^{k+1}} \leftarrow \overrightarrow{v_i^{k}} + c_1 r_1 \left( \overrightarrow{p}_i - \overrightarrow{x_i^{k}} \right) + c_2 r_2 \left( \overrightarrow{p}_g - \overrightarrow{x_i^{k}} \right) \tag{8}$$

where $c_1$ and $c_2$ are velocity constants, $r_1$ and $r_2$ are two random numbers in the interval $[0, 1]$, $x_i^k$ is the current position of the $i$th particle in iteration $k$, $p_i$ is the $i$th particle best position so far, and $p_g$ is the global best position in the search so far. However, the proposed PSO employs a development of the original velocity update formula:

$$v_i^{k+1} \leftarrow \chi\left(w v_i^k + c_1 r_1 (p_i - x_i^k) + c_2 r_2 (p_g - x_i^k)\right); \quad i = 1, 2, \ldots, l \tag{9}$$

In which $w$ and $\chi$, inertia weight and constriction coefficient, are calculated as follows:

$$w = w_{\max} - \frac{w_{\max} - w_{\min}}{b} \times k$$
$$\chi = \frac{2}{(c_1 + c_2) - 2 + \sqrt{(c_1 + c_2)^2 - 4(c_1 + c_2)}}; \quad c_1 + c_2 > 4 \tag{10}$$

$w_{max}$ and $w_{min}$ are two parameters set by the user, $b$ is the total number of iterations, and $k$ is the number of current iteration. Unfortunately, (9) does not guarantee that the particles remain in the feasible space after update. Therefore, we consider the following conditions.

$$v_i^{k+1} \leftarrow \begin{cases} r_1' \times (-x_i^k) & \text{if } v_i^k < -x_i^k \\ r_1' \times (n! - 1 - x_i^k) & \text{if } v_i^k > n! - 1 - x_i^k \end{cases} \quad (11)$$

where $r_1'$ is a random number in the interval [0, 1]. Particle position is updated by (12).

$$x_i^{k+1} \leftarrow \lfloor x_i^k + v_i^k \rfloor; \quad i = 1, 2, \ldots, l \quad (12)$$

(12) will transfer each particle to its neighborhood. After updating the particle position, algorithm first maps the number to its unique respective permutation and evaluates the objective function of the particle's new position by calculating the objective function of the ABSMODEL. At this point, the variables $p_i; \quad i = 1, 2, \ldots, l$ and $p_g$ will be updated if necessary. The algorithm will stop and return $p_g$ as the final solution after $b$ iterations. $b$ is a parameter set by the user.

### 5.3. Intensification

Intensification is the search in the neighborhood of the good-quality solutions for constructing better solutions closer to optimum. There are numerous intensification algorithms developed in the literature; namely, pairwise exchange algorithm families. The reader is referred to Francis, McGinnis, and White (1998) and Kusiak and Heragu (1987) for more information in this regard. However, our intensification procedure is more exhaustive and searches the current neighborhood of all of the particles once triggered. Intensification is performed on all particles after $a$ iterations, $2a$ iterations and so on, where $a$ is a parameter set by the user. The proposed algorithm uses a straightforward procedure as its intensification sub-algorithm. First integer numbers – or particles – are mapped into their respective permutation. Then the algorithm exchanges the location of the first two adjacent facilities and evaluates the objective function value of the new permutation. If the objective function of the new permutation has improved by the exchange, algorithm accepts this exchange and restarts the exchange sub-procedure. Nevertheless, if this exchange does not improve the objective function of the solution, the two facilities will move back to their original locations and exchange will be applied to the next two adjacent facilities.

Applying the described procedures to all particles possibly results in permutations with better objective function values. These new permutations will be mapped to their respective integer number using Algorithm 2, and the variables $p_i; \quad i = 1, 2, \ldots, l$ and $p_g$ will be updated if necessary.

## 6. Stepwise algorithm

*Step 1*: Set the values of the control parameters: $l$ (number of particles), $a$ (number of iterations between each two consecutive intensifications), $b$ (total number of iterations), $c_1$ and $c_2$ (velocity constants), and $w_{max}$ and $w_{min}$ (parameters to affect inertia weight). Set $k \leftarrow 0$.
*Step 2*: Generate $l$ random integer numbers $x_i^k; \quad i = 1, 2, \ldots, l$ in the interval $[0, n! - 1]$. Map these numbers to their respective permutation using Algorithm 1. Evaluate the objective function of each permutation. Update variables $p_i; \quad i = 1, 2, \ldots, l$ and $p_g$.
*Step 3*: Generate $l$ random integer numbers $v_i^k; \quad i = 1, 2, \ldots, l$ in the interval $[-x_i^k, (n! - 1) - x_i^k]$.
*Step 4*: Set $x_i^{k+1} \leftarrow \lfloor x_i^k + v_i^k \rfloor; \quad i = 1, 2, \ldots, l$. Use Algorithm 1 to evaluate the objective function of the new permutations. Update variables $p_i; \quad i = 1, 2, \ldots, l$ and $p_g$ if necessary. Set $k \leftarrow k + 1$.
*Step 5*: Run the intensification procedure if $\frac{k}{a} \in N$. Otherwise, proceed to step 6.
*Step 6*: If $k = b$, stop. Return the value of variable $p_g$ and its respective permutation as final solution. Otherwise, proceed to step 7.
*Step 7*: Update particle velocity using Eqs. (9) and (11). Go to step 4.

## 7. Computational results

The proposed algorithm was coded in C++ and run on a PC equipped with a 2 GHz Intel Centrino Duo CPU and 1 GB of RAM. To verify the performance of the algorithm, we have tested it by a set of 32 problems available in the literature. However, for comparative purposes, we have divided this set into three sub-sets. Problem set 1 includes the problems proposed in Nugent, Vollman, and Ruml (1968) and consists of eight problems. Facility size for this set is accessible from Heragu and Kusiak (1988). Set 2 includes eight problems collected from Simmons (1969), Nugent et al. (1968), Love and Wong (1976). Problem data for this set can be found at Anjos and Kong (2007). Problem set 3 includes 16 problems to which a heuristic or meta-heuristic approach is not applied in the literature. However, optimum solution of these problems is known. Problem data for this set can be found at Anjos and Kong (2007). Results for these three sub-sets can be found in Tables 4, 5 and 6 respectively. As mentioned before, the proposed PSO algorithm follows a probabilistic approach to generate the initial solutions and to update the particle position. Consequently, we have conducted five independent runs of the algorithm for each problem; results in Tables 4–6 are the best from the five runs of the algorithm accompanied by their respective CPU time. It should be mentioned that the CPU times in the following tables are not comparable since different computer settings will result in different CPU time.

As seen in step 1 of the stepwise algorithm, seven control parameters should be set for the proposed PSO algorithm to start the search. Values of these parameters severely affect the algorithm's performance. We have performed extensive sensitivity analysis on these parameters to determine the effect of the different values of the parameters on the performance of the algorithm. For the sensitivity analysis, we have employed the approach introduced in Behroozi (2009). Based on this approach, we have chosen 3 different problems from each of the above sets. The chosen problems are amongst the most difficult problems in their respective set. Then, the problems were solved with different combinations of parameter values until the best combination is observed. Based on these observations, the following experimentally derived values are proposed for the parameters:

$$\begin{array}{lll} l = \frac{n}{2} & a = 10 & b = 10n \\ c_1 = 2.1 & c_2 = 2.1 & w_{max} = 1 \\ w_{min} = 0.5 & & \end{array} \quad (13)$$

All the heuristics in Table 4 were obtained on a Sun 3/260 computer except the method proposed in Solimanpur et al. (2005) which was obtained on Pentium III 550 MHz PC. The OFV column in this table shows the Objective Function Value of the method. OFVs in bold show the optimum results for each of the problems. The results for the proposed PSO are shown in the last column. The proposed PSO is able to achieve the best results obtained so far.

Computational results for the second set of problems are shown in Table 5. The proposed PSO is able to achieve the best solutions reported in the literature. We consider that the result reported in Solimanpur et al. (2005) for problem number 8 is a typo since

**Table 4**
Comparative results for problem set 1.

| Prob. no. | No. of Fac. | Hall (1970) | | Neghabat (1974) | | Heragu and Kusiak (1988) | | Kumar et al. (1995) | | Solimanpur et al. (2005) | | Proposed PSO | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | OFV | Time | OFV | Time | OFV | Time | OFV | Time | OFV | Time | OFV | Time |
| 1 | 5 | 1.140 | 0.018 | **1.100** | 0.009 | 1.165 | 0.007 | 1.140 | 0.008 | **1.100** | 0.00 | **1.100** | 0.004 |
| 2 | 6 | **1.990** | 0.034 | 2.030 | 0.015 | 2.085 | 0.008 | **1.990** | 0.010 | **1.990** | 0.00 | **1.990** | 0.005 |
| 3 | 7 | 5.530 | 0.052 | **4.730** | 0.500 | 5.420 | 0.017 | 5.150 | 0.033 | **4.730** | 0.00 | **4.730** | 0.009 |
| 4 | 8 | 7.035 | 0.069 | **6.295** | 5.500 | 7.995 | 0.035 | 6.775 | 0.067 | **6.295** | 0.00 | **6.295** | 0.017 |
| 5 | 12 | 27.805 | 0.120 | 24.675 | 1800 | 31.525 | 0.067 | 24.715 | 0.100 | **23.365** | 0.06 | **23.365** | 0.305 |
| 6 | 15 | 48.375 | 0.153 | 51.500 | 1800 | 62.624 | 0.100 | 54.830 | 0.600 | **44.600** | 0.18 | **44.600** | 0.733 |
| 7 | 20 | 153.260 | 0.255 | 169.920 | 1800 | 178.149 | 0.600 | 141.040 | 4.200 | **119.710** | 1.80 | **119.710** | 3.004 |
| 8 | 30 | 395.770 | 0.624 | 466.800 | 1800 | 414.400 | 4.800 | 405.520 | 14.80 | **334.870** | 37.3 | **334.870** | 18.840 |

**Table 5**
Comparative results for problem set 2.

| Prob. no. | No. of Fac. | Heragu and Kusiak (1991) with infeasible initial solutions | | Heragu and Kusiak (1991) without infeasible initial solutions | | Heragu and Alfa (1992) | | Kumar et al. (1995) | | Solimanpur et al. (2005) | | Proposed PSO | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | OFV | Time | OFV | Time | OFV | Time | OFV | Time | OFV | Time | OFV | Time |
| 1 | 4 | **78.000** | 0.08 | **78.000** | 0.09 | **78.000** | 1.624 | **78.000** | 0.007 | **78.000** | 0.00 | **78.000** | 0.000 |
| 2 | 5 | **151.000** | 0.08 | **151.000** | 0.13 | **151.000** | 10.351 | **151.000** | 0.010 | **151.000** | 0.00 | **151.000** | 0.003 |
| 3 | 8 | 2342.50 | 0.36 | 2341.50 | 0.59 | **2324.50** | 11.803 | **2324.50** | 0.080 | **2324.50** | 0.00 | **2324.50** | 0.013 |
| 4 | 10 | **2781.50** | 1.11 | **2781.50** | 0.84 | **2781.50** | 19.815 | **2781.50** | 0.100 | **2781.50** | 0.01 | **2781.50** | 0.095 |
| 5 | 11 | 7041.50 | 0.96 | 7274.50 | 2.18 | **6933.50** | 23.103 | 6953.50 | 0.120 | **6933.50** | 0.03 | **6933.50** | 0.211 |
| 6 | 11 | **6933.50** | 0.98 | **6933.50** | 0.95 | **6933.50** | 29.176 | 7265.50 | 0.120 | **6933.50** | 0.02 | **6933.50** | 0.172 |
| 7 | 20 | 16265.0 | 10.68 | 16109.0 | 7.82 | 15602.0 | 603.37 | **15549.0** | 8.600 | **15549.0** | 2.30 | **15549.0** | 2.812 |
| 8 | 30 | 46139.0 | 36.43 | 46456.0 | 35.74 | 45111.0 | 585.94 | 44466.5 | 91.80 | **25000.0** | 37.30 | 44965.0 | 12.220 |

**Table 6**
Computational results for problem set 3.

| Problem No. | Reference | Number of facilities | Proposed PSO | | Optimum solution | | Gap (%) |
|---|---|---|---|---|---|---|---|
| | | | OFV | Time | OFV | Reference | |
| 1 | Amaral (2006) | 4 | **638.00** | 0.000 | **638.00** | Amaral (2006) | 0 |
| 2 | Simmons (1969) | 8 | **801.00** | 0.020 | **801.00** | Amaral (2006) | 0 |
| 3 | Simmons (1969) | 9 | **2469.50** | 0.025 | **2469.50** | Amaral (2006) | 0 |
| 4 | Simmons (1969) | 9 | **4695.50** | 0.031 | **4695.50** | Amaral (2006) | 0 |
| 5 | Heragu and Kusiak (1991) | 12 | **18140.23** | 0.517 | **18140.23** | Anjos and Vannelli (2008) | 0 |
| 6 | Amaral (2006) | 15 | **6305.00** | 1.131 | **6305.00** | Amaral (2006) | 0 |
| 7 | Anjos and Vannelli (2008) | 25 | 4619.00 | 8.012 | **4618.00** | Anjos and Vannelli (2008) | 0.02 |
| 8 | Anjos and Vannelli (2008) | 25 | **37116.5** | 6.920 | **37116.50** | Anjos and Vannelli (2008) | 0 |
| 9 | Anjos and Vannelli (2008) | 25 | 24351.0 | 9.560 | **24301.00** | Anjos and Vannelli (2008) | 0.20 |
| 10 | Anjos and Vannelli (2008) | 25 | **48291.5** | 7.592 | **48291.50** | Anjos and Vannelli (2008) | 0 |
| 11 | Anjos and Vannelli (2008) | 25 | **15623.0** | 6.303 | **15623.00** | Anjos and Vannelli (2008) | 0 |
| 12 | Anjos and Vannelli (2008) | 30 | **8247.00** | 14.034 | **8247.00** | Anjos and Vannelli (2008) | 0 |
| 13 | Anjos and Vannelli (2008) | 30 | **21582.5** | 11.146 | **21582.50** | Anjos and Vannelli (2008) | 0 |
| 14 | Anjos and Vannelli (2008) | 30 | 45751.0 | 15.950 | **45449.00** | Anjos and Vannelli (2008) | 0.66 |
| 15 | Anjos and Vannelli (2008) | 30 | 57874.5 | 14.889 | **56873.50** | Anjos and Vannelli (2008) | 1.76 |
| 16 | Anjos and Vannelli (2008) | 30 | **115268.0** | 15.275 | **115268.0** | Anjos and Vannelli (2008) | 0 |

the optimum solution for this problem is proven to be the result obtained by the proposed algorithm (Anjos & Vannelli, 2008). The proposed algorithm of Heragu and Kusiak (1991) was run on an AMDAHL 5870. Heragu and Alfa (1992) obtained their CPU time on an VAX 6420 mainframe computer. Kumar et al. (1995) tested their proposed heuristic on a Sun 3/260 computer while Solimanpur et al. (2005) evaluated their ACO algorithm on a Pentium III 550 MHz PC.

Computational results for the third set of problems are shown in Table 6. First column of Table 6 shows the problem number. Second column of this table presents the reference where the problem has first introduced. Next column is the number of facilities in the problem. Objective function and CPU time of the proposed algorithm are given in columns 4 and 5 while objective function of the optimum solution and the reference where this solution is found are in columns 6 and 7. Last column calculates the gap between the objective function of the optimum solution and the proposed algorithm. Table 6 demonstrates that the proposed algorithm is able to find the optimum solution of 12 problems out of 16 problems (75%).

To demonstrate the important role of intensification sub-algorithm, we have conducted an experiment which graphically depicts the difference between the objective function of a run of the algorithm with and without intensification sub-algorithm. The experiment is run on the problem number 15 from the third set of the problems. All the control parameters are set to their respective values of (13). We have run the algorithm once for this problem and tracked the globally best particle position during all iterations of the algorithm. The result is shown in Fig. 1. In this figure, horizontal axis shows the iteration number and vertical axis depicts the objective function value. In this figure, grey line corresponds to the objective function value of the problem when inten-

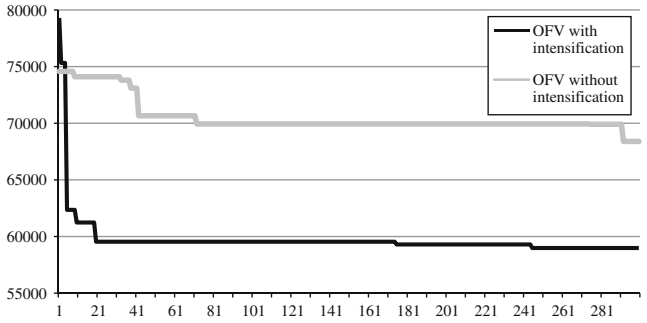


**Fig. 1.** Role of intensification sub-algorithm on the solution of the problem number 15, set 3.

sification sub-algorithm is not applied while black line matches the objective function value of the problem when intensification sub-algorithm is applied. The significant role of intensification sub-algorithm in the quality of the final solution, regardless of the quality of the initial solutions, can be concluded from this figure.

## 8. Conclusion

This paper considers the single row facility layout problem in which the size of facilities are assumed to be different. This problem is NP-Complete and finding optimum solution for the large instances of this problem is not possible in a reasonable time even by modern computers. Although exact and heuristic algorithms have been developed in recent years to find optimal and near-optimal solutions of the SRFLP, we still need to have a fast algorithm that can obtain a near-optimal solution for large instances of the problem.

A particle swarm optimization algorithm was proposed to find near-optimal solutions of this problem. The proposed algorithm employs a coding procedure to map the discrete feasible space of the problem into a continuous space where particle swarm algorithm can efficiently perform exploration. Afterward, the algorithm explores this continuous feasible region for the good-quality solutions of the problem. Moreover, a powerful intensification sub-algorithm was developed to search the promising areas of the feasible space more comprehensively. Experimentally derived rules were used to set the values of the control parameters in different problems.

Performance of the proposed algorithm was tested over a large variety of the problems available from the literature (32 problems) and also compared to many other algorithms existing in the literature. The computational results verify the efficiency of the algorithm in finding good quality, and near-optimum solutions in a similar time compared to the best published heuristics in the literature.

## References


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

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

Anjos, M. F., & Kong, C. (2007). FLP database. <http://flplib.uwaterloo.ca/>.

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

Anjos, M. F., & 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.

Behroozi, M. (2009). A meta-heuristic approach for a special class of job shop scheduling problem. In *Faculty of industrial engineering*. Tehran, Iran: Sharif University of Technology.

Braglia, M. (1996). Optimization of a simulated-annealing-based heuristic for single row machine layout problem by genetic algorithm. *International Transactions in Operational Research, 1*(3), 37–49.

Doliskani, J. N., Malekian, E., & Zakerolhosseini, A. (2008). A cryptosystem based on the symmetric group Sn. *International Journal of Computer Science and Network Security, 8*(2), 226–234.

Drezner, Z. (1987). A heuristic procedure for the layout of a large number of facilities. *Management Science, 7*(33), 907–915.

Eberhart, R. C., & Kennedy, J. (1995). A new optimizer using particle swarm theory. In *Sixth international symposium on micro machine and human science*. Nagoya, Japan.

Ficko, M., Brezocnik, M., & Balic, J. (2004). Designing the layout of single- and multiple-rows flexible manufacturing system by genetic algorithms. *Journal of Material Processing Technology*, 150–158.

Francis, R. L., McGinnis, F., & White, J. A. (1998). *Facility layout and location: An analytical approach*. Prentice Hall Professional Technical Reference.

Hall, K. M. (1970). An r-dimensional quadratic placement algorithm. *Management Science*(17), 219–229.

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

Heragu, S. S., & Kusiak, A. (1988). Machine layout problem in flexible manufacturing systems. *Operations Research*(36), 258–268.

Heragu, S. S., & Kusiak, A. (1991). Efficient models for the facility layout problems. *European Journal of Operational Research*(53), 1–13.

Kennedy, J., & Eberhart, R. C. (1997). A discrete binary version of the particle swarm algorithm. In *IEEE international conference on systems, man, and cybernetics*.

Knuth, D. (1997). *Seminumerical algorithms* (3rd ed.). *The art of computer programming* (Vol. 2). Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc..

Kumar, S. et al. (2008). Scatter search algorithm for single row layout problem in FMS. *Advances in Production Engineering and Management, 3*(4), 193–204.

Kumar, K. R., Hadjinicola, G. C., & Lin, T. L. (1995). A heuristic procedure for the single row facility layout problem. *European Journal of Operational Research*(87), 65–73.

Kusiak, A., & Heragu, S. S. (1987). The facility layout problem. *European Journal of Operational Research, 29*(3), 229–251.

Lin, M. T. (2009). The single-row machine layout problem in apparel manufacturing by hierarchical order-based genetic algorithm. *International Journal of Clothing Science and Technology, 21*(1), 31–43.

Love, R. F., & Wong, J. Y. (1976). On solving a one-dimensional allocation problem with integer programming. *Information Processes and Operation Research (INFOR)*(14), 139–143.

McCaffrey, J. (2003).Using permutations. In *NET for improved systems security*. <http://msdn.microsoft.com/en-us/library/aa302371.aspx>.

Neghabat, F. (1974). An efficient equipment layout algorithm. *Operations Research*(22), 622–628.

Nugent, C. E., Vollman, T. E., & Ruml, J. (1968). An experimental comparison of techniques for the assignment of facilities to locations. *Operations Research, 16*, 150–173.

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

Simmons, D. M. (1969). One dimensional space allocation: An ordering algorithm. *Operations Research*(17), 812–826.

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

Suresh, G., & Sahu, S. (1993). Multiobjective facility layout using simulated annealing. *International Journal of Production Economics, 32*, 239–254.

Teo, Y. T., & Ponnambalam, S. G. (2008). A hybrid ACO/PSO heuristic to solve single row layout problem. In *4th IEEE conference on automation science and engineering*. Washington, DC, USA: Key Bridge Marriott.