Engineering Optimization
Publication details, including instructions for authors and subscription information:
http://www.tandfonline.com/loi/geno20

Robotic U-shaped assembly line balancing using particle swarm optimization

J. Mukund Nilakantan[a] & S.G. Ponnambalam[a]

[a] Advanced Engineering Platform and School of Engineering, Monash University Malaysia, Bandar Sunway, Malaysia

Published online: 12 Jan 2015.

Engineering Optimization, 2015
http://dx.doi.org/10.1080/0305215X.2014.998664

Robotic U-shaped assembly line balancing using particle swarm optimization

J. Mukund Nilakantan and S.G. Ponnambalam*

Advanced Engineering Platform and School of Engineering, Monash University Malaysia, Bandar Sunway, Malaysia

(*Received 14 April 2014; accepted 20 November 2014*)

Automation in an assembly line can be achieved using robots. In robotic U-shaped assembly line balancing (RUALB), robots are assigned to workstations to perform the assembly tasks on a U-shaped assembly line. The robots are expected to perform multiple tasks, because of their capabilities. U-shaped assembly line problems are derived from traditional assembly line problems and are relatively new. Tasks are assigned to the workstations when either all of their predecessors or all of their successors have already been assigned to workstations. The objective function considered in this article is to maximize the cycle time of the assembly line, which in turn helps to maximize the production rate of the assembly line. RUALB aims at the optimal assignment of tasks to the workstations and selection of the best fit robot to the workstations in a manner such that the cycle time is minimized. To solve this problem, a particle swarm optimization algorithm embedded with a heuristic allocation (consecutive) procedure is proposed. The consecutive heuristic is used to allocate the tasks to the workstation and to assign a best fit robot to that workstation. The proposed algorithm is evaluated using a wide variety of data sets. The results indicate that robotic U-shaped assembly lines perform better than robotic straight assembly lines in terms of cycle time.

Keywords: straight robotic assembly line balancing; U-shaped robotic assembly line; particle swarm optimization; cycle time

1. Introduction

Balancing is the process of allocating work (tasks) to workstations in such a manner that all workstations have equal amounts of work assigned to them. Assignment of tasks to the workstation should be on the basis of a precedence relationship. The assembly line balancing (ALB) problem has been studied extensively since 1950 and the first mathematical model was formulated by Salveson (1955). The traditional ALB problem leads to two types of optimization problem: type I and type II (Scholl and Becker 2006). In type I problems, when the cycle time is fixed, the objective is to minimize the required number of workstations; and in type II problems, the aim is to minimize the cycle time for given a fixed number of workstations (Akpinar and Bayhan 2014).

Based on the nature of flow, assembly lines can be classified into two types: straight (traditional) assembly lines, with single and multiple/mixed products, and U-shaped assembly lines (also called U-lines), with single and multiple/mixed products. The main difference between the two types is the design of the assembly line. In the case of U-shaped assembly lines, the entrance

*Corresponding author. Email: sgponnambalam@monash.edu and exit are in the same position (Toklu and özcan 2008). Using a U-shaped layout the number of operators can be increased or decreased to suit changes in demand (Aigbedo and Monden 1997). In U-shaped assembly lines, a task can be assigned to a workstation after all of its predecessors or all successors have been assigned to an earlier or the same workstation. This is the distinguishing feature of U-shaped assembly line balancing (UALB) problems, which must allow for the forward and backward assignment of tasks to workstations (Kara 2008).

Advantages such as reduction in the cycle time and cost of assembly have led to the implementation of U-shaped layouts in different industries (Kubota 2011; Yalaoui *et al.* 2013). Toyota Motors introduced a new assembly line structure (U-shaped layout) at a subsidiary factory which helps in saving capital expenditure and in reducing the production time. A U-shaped layout facilitates the performance of more than one task at a time on a vehicle, such as installing the engine in the front while adding underbody parts in the back. In the new layout, vehicles are carried on a conveyor belt and it may be observed that the assembly line could be cut to one-third of the length when compared to traditional assembly lines. The implementation of a U-shaped layout helped Toyota to save up to 40% on their capital expenditure (Kubota 2011). Boeing implemented U-shaped assembly lines in 2010, and installed the largest integrated moving assembly line in the world (Fetters-Walp 2010). Boeing could achieve a new record rate of assembling 8.3 aircraft per month or 100 per year. It helped the company to reduce flow time and production costs, and the moving U-shaped assembly lines created an easier working environment for the operators (Boeing 2014).

Owing to fast changes in technology and increased demand for greater product variety, robots play an important role in assembly line systems. An assembly robot can be used to work for 24 hours a day without worries about fatigue and failure. The use of robots improves productivity and quality of the product, and allows for flexibility in manufacturing (Gao *et al.* 2009).

An important variant of the ALB problem is robotic assembly line balancing (RALB). The RALB problem is a generalized form of the traditional ALB problem. Although researchers have proposed algorithms to solve problems of this nature, no research has been undertaken in the field of robotic assembly line balancing problem with a U-shaped configuration (RUALB). The main aim of this work was to propose a population-based search algorithm to solve a robotic U-shaped assembly line balancing (RUALB) problem, with the objective of minimizing the cycle time for a fixed number of workstations.

The remainder of the article is presented as follows. The need to balance an assembly line and the possible losses incurred in balanced and unbalanced assembly lines are discussed with an example in Section 2. The relevant literature on line balancing problems and different solution procedures are presented in Section 3. Section 4 provides the assumptions and the mathematical model of the RUALB problem. Section 5 explains the implementation of particle swarm optimization (PSO) in detail. The experimental and computational results are presented in Section 6. Finally, Section 7 presents conclusions on the findings of this research.

2. Need for balancing an assembly line

Balancing of an assembly line implies the equal distribution of the total workload of the line among workstations so that idle times are as low as possible. On robotic/worker-based assembly lines, the key objective is to balance the workload of robots or workers at every workstation, which helps to minimize losses and costs. A production system is a dynamic process with different components such as products, operations, material handling and assembly line characteristics, and the system needs to adapt to any changes that occur to these components without causing much loss. Taking this into consideration, fixing anything in the production system is a source of

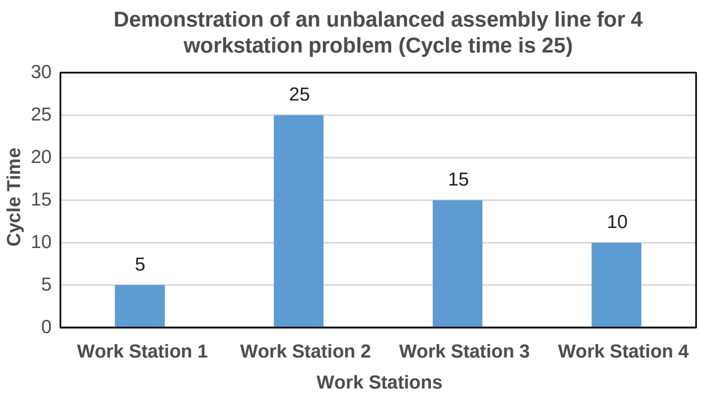

Figure 1. Unbalanced assembly line.

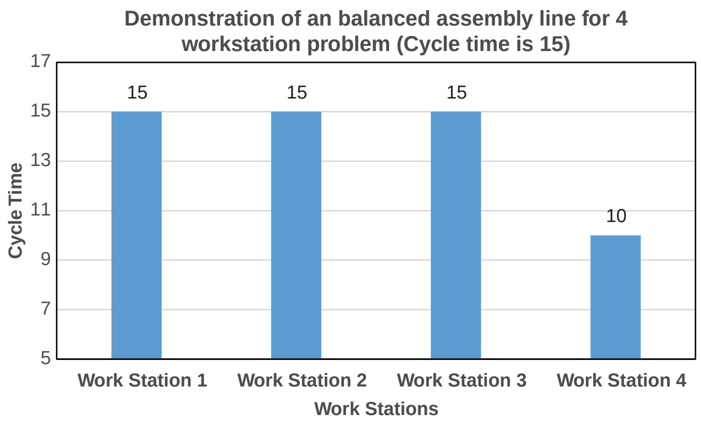

Figure 2. Balanced assembly line.

loss for the whole system, which can be completely removed periodically during the balancing process. Balance losses of an assembly line are bound to happen and obtaining a perfect balance of workload is very unlikely owing to the dynamic features of the system. It is not practically possible to allocate work among the workstations and hence there are always balance losses. However, by understanding these losses, balance losses can be minimized for the assembly line (Törenli 2009). An example problem is used to illustrate the losses of balanced and unbalanced assembly lines. Figure 1 shows two different balancing conditions on the same line for a problem with four workstations. The numbers on top of the bars in the graph shows the workstation time. Figure 1 shows that workstation 2 is overloaded and workstations 3 and 4 need to wait for workstation 2 to complete the job on the line. Figure 2 shows the balanced allocation of workload among the workstations. Idle time at stations is the primary indicator of the balance losses of an assembly line. Idle time at a workstation shows that there is an excess capacity at that station, which is undesired. The idle time of the line is calculated using Equation (1):

$$\text{Idle time of workstations} = \sum_{i=1}^{N_w} (\text{Cycle time of assembly line} - \text{Workstation time}) \tag{1}$$

In the given example, for an unbalanced assembly line the cycle time is 25 and the idle time is found to be 45, whereas for a balanced assembly line the cycle time is 15 and the idle time is 5. It can be concluded from this example that the firm will have to balance the assembly line to avoid the loss due to idle time. Balancing the assembly line helps to promote one piece at a time, avoid overburden at the workstations, minimize wastage and reduce variation.

3. Literature review

In recent decades, manufacturers have started adopting a just-in-time (JIT) approach to manufacturing, which improves the productivity, profits and product quality. Companies that employ JIT techniques find it beneficial for repetitive jobs and for job shop and process manufacturing (Toksarı *et al.* 2008). Owing to the implementation of JIT techniques there is a requirement to replace the traditional straight assembly line with a U-shaped assembly line (Chiang and Urban 2006). The main advantages of the U-shaped assembly line compared to a straight line are the reduction in the repetitive movement of operators and improvements in productivity. U-shaped assembly lines are highly flexible and changes can be made depending on demand. U-shaped lines also improve material handling (Toksarı *et al.* 2008; Hiroyuki and Black 1988; Monden 1995).

There is growing interest in the literature in reorganizing traditional assembly lines as U-lines to improve performance. Although there have been many studies on traditional straight assembly lines, work on U-shaped assembly lines is limited. Monden (1995) brought U-type assembly lines to the attention of researchers through his observations in Japanese companies, such as the Toyota production system. One of the important characteristics of U-shaped assembly line systems is that workers become multi-skilled as they perform various tasks on different stations along the assembly line. This distinct feature allows the assignment of tasks from both the beginning and the end of the precedence list. Owing to this possibility of assignment, U-shaped assembly lines have an advantage over straight assembly lines, with improved results from the reduced precedence relation limitations. However, assignment for the U-type configuration is more complex than for the straight assembly line because of the large search space.

Miltenburg and Wijngaard (1994) introduced and developed a dynamic programming formulation to evaluate a single-model U-type assembly line with the objective of minimizing the number of stations. They proposed an algorithm to solve small-sized problems with 11 tasks, and the ranked positional weight technique to solve large-size problems. Urban (1998) formulated an integer programing problem to solve a U-shaped assembly line problem with 45 tasks.

Different versions of the U-shaped assembly line problem were solved by Scholl and Klein (1999) using the branch-and-bound method. They proposed a technique called the U-line optimizer (ULINO), which uses a depth-first branch-and-bound algorithm that minimizes the number of stations, the cycle time, or both. The proposed algorithm is used to obtain optimal results for problems up to 297 tasks. Similarly, more studies on the use of exact methods have been presented and are available in the literature (Miltenburg 1998; Gökçen *et al.* 2005; Nakade and Ohno 1999; Zhang and Cheng 2010). UALB problems fall under the category of NP hard. Different heuristics and metaheuristics have been used to solve UALB problems. Ajenblit and Wainwright (1998) presented a genetic algorithm (GA) for UALB with the objective of balancing the workload and minimizing total idle time. Erel, Sabuncuoglu, and Aksu (2001) proposed a simulated annealing method for solving a UALB problem. Martinez and Duff (2004) proposed heuristic approaches to solve the UALB augmented by GAs. They used 10 task assignment rules. Sirovetnukul and Chutima (2010) developed a novel algorithm, named particle swarm optimization with negative knowledge (PSONK), to solve a single U-shaped assembly line problem with aim of minimizing the number of workers, and achieving equity of workload and the shortest walking time. The performance of PSONK has been compared with the non-dominated sorting genetic algorithm-II (NSGA-II) and experimental results show that PSONK outperforms NSGA-II. Avikal *et al.* (2013) proposed a heuristic based on the critical path method for the assignment of tasks to the workstations in a U-shaped assembly line layout. The objective of the work was to improve the labour productivity and minimize the number of workstations. The proposed method obtained better results for the U-shaped assembly line compared with a traditional straight assembly line.

Many researchers have used different metaheuristic algorithms, such as GAs, ant colony optimization, tabu search, PSO and simulated annealing, to solve a different variety of UALB problems with different objectives (Kim, Kim, and Kim 2000, 2006; Khaw and Ponnambalam 2009; Baykasoglu 2006; Hwang, Katayama, and Gen 2008; Zha and Yu 2014).

The studies mentioned above are related to exact, heuristic and metaheuristic methods proposed for UALB problems. RALB problems are divided into two types: type I (RALB-I) and type II (RALB-II). Type I RALB deals in minimizing the number of workstations on an assembly line for a fixed cycle time, whereas type II RALB deals in minimizing the cycle time for a fixed number of workstations. The literature on type II straight RALB problems is discussed below. Levitin, Rubinovitz, and Shnits (2006) first developed an algorithm for solving RALB-II problems. Their method aims to distribute tasks to the workstation in a balanced manner and to allocate the best robot to these workstations to perform the tasks. They proposed a GA to solve the problem. They also proposed two heuristics for task assignment and cycle time calculation: a recursive and a consecutive procedure. Gao *et al.* (2009) present a 0–1 integer programming formulation for the RALB-II problem. They proposed a hybrid GA, in which they implemented a partial representation technique. They also proposed new crossover and mutation operators to adapt to the chromosome structure and nature of the problem. To increase the search ability of the algorithm, local search procedures are included. Mukund Nilakantan and Ponnambalam (2012) applied a standard PSO to solve RALB-II problems on a straight assembly line. Yoosefelahi *et al.* (2012) proposed a multi-objective model for RALB-II. The algorithm aimed at minimizing the cycle time, robot set-up costs and robot costs. They also developed a new mixed-integer linear programming model and presented the performance evaluation of the proposed algorithm. Daoud *et al.* (2014) proposed several metaheuristic algorithms and a discrete event simulation model to solve the RALB problem with the objective of maximizing line efficiency, and considered an automated packaging line dedicated to dairy food products as a case study for evaluating the proposed model.

Detailed surveys on different types of metaheuristic algorithm used to solve ALB problems can be found in the literature (Rashid, Hutabarat, and Tiwari 2012; Sivasankaran and Shahabudeen 2014; Battaïa and Dolgui 2013). A literature survey revealed that to date no research has been done on RUALB problems. In this research, a type II RUALB problem with an objective of minimizing cycle time in a U-shaped robotic assembly line is presented.

4. Mathematical model for robotic U-shaped assembly line balancing

Assembly tasks of different types are performed at each workstation. These tasks are to be completed to produce a final product. Precedence constraints are specified and determine the order in which tasks should be executed. Tasks are to be assigned to the workstation and the best robot needs to be allotted to the workstation to perform the tasks. In this article, a U-shaped assembly line is taken into consideration and the main objective is to minimize the cycle time of the assembly line.

The assumptions used in this article, based on the work by Levitin, Rubinovitz, and Shnits (2006) and Gao *et al.* (2009), are listed below.

(1) A task (operation) cannot be split among two or more workstations.
(2) Time taken to perform a task depends on the robot assigned.

(3) Only one robot type at a time can be assigned to a workstation.
(4) The number of workstations is equal to the number of robots.
(5) Any task can be processed at any station by any robot.
(6) All types of robot are available without limitations.
(7) A robot type that can perform the tasks assigned to a workstation in the shortest time among other type of robots is considered to be assigned to a workstation.
(8) Material handling, loading and unloading time, and set-up and tool changing time are negligible, or are included in the task time. This assumption is realistic on a single-model assembly line. Tooling on such robotic lines is usually designed such that tool changes are minimized within a station. If a tool change or another type of set-up task is necessary, it can be included in the task time.
(9) The assembly line is designed for a unique model of a single product.

Based on the definition proposed by Gutjahr and Nemhauser (1964) for a straight line assembly line problem, Miltenburg and Wijngaard (1994) presented a definition for the simple U-shaped assembly line problem. Gao *et al.* (2009) presented a formulation for type II RALB. The formulation presented in this article is based on these definitions.

The problem is: Given a set of tasks $F = \{g \mid g = 1, 2, \ldots ,n\}$ (Salveson 1955), a set of precedence constraints $P = \{ (i, j) \mid$ task i must be completed before task j $\}$, a set of task times $T = \{ t(g) \mid g = 1, 2, \ldots , n \}$, and a cycle time C, find a collection of subsets of F, $(L_1, L_2, \ldots , L_N)$, where $L_a = \{g \mid$ task g is done at workstation a $\}$ and the workstations and tasks are arranged in a U shape.

Indices:

i, j:	Index of assembly tasks
h:	Robot
s:	Workstation
F:	Set of tasks
H:	Set of available robots
S:	Set of workstations
N_w:	Number of workstations
N_a:	Number of tasks
N_r:	Number of robots
C:	Cycle time
sq:	Sequence of tasks representing a feasible solution
t_{ih}:	Processing time of task i by robot h
P:	Set of precedence constraints

Decision variables:

$$x_{is} = \begin{cases} 1 \text{ if task } i \text{ is assigned to workstation } s \\ 0,\text{otherwise} \end{cases}$$

$$y_{sh} = \begin{cases} 1 \text{ if robot } h \text{ is allocated to workstation } s \\ 0,\text{otherwise} \end{cases}$$

According to the notations and the assumptions considered, a 0–1 integer programming model for this problem is formulated as follows:

$$\min c = \max_{1\le s\le N_w} \left\{ \sum_{i=1}^{N_a} \sum_{i=1}^{N_w} t_{ih}.x_{is}.y_{sh} \right. \tag{2}$$

For each task *j*:

$$\text{if}(i,j) \in P, i \in L_a, j \in L_b, \text{then } a \leq b, \quad \text{for all } i; \textit{or}$$
$$\text{if}(j,k) \in P, y \in L_b, k \in L_c, \text{then } c \leq b, \quad \text{for all } k; \tag{3}$$

$$\sum_{s=1}^{N_w} x_{is} = 1 \quad \forall i \tag{4}$$

$$\sum_{s=1}^{N_w} y_{sh} = 1 \quad \forall s \tag{5}$$

$$x_{is} \in \{0,1\} \quad \forall s, i \tag{6}$$

$$y_{sh} \in \{0,1\} \quad \forall h, s \tag{7}$$

The second objective is to minimize the cycle time of the robotic assembly line. Equation (3) ensures that the precedence constraints are not violated on the U-shaped assembly line. Equation (4) ensures that each task has to be assigned to one workstation and Equation (5) ensures that each workstation is equipped with one robot. It is notable that the first objective is nonlinear, hence it is hard for traditional exact optimization techniques to solve the problem.

5. Particle swarm optimization for robotic U-shaped assembly line balancing

Owing to the NP-hard nature of ALB problems, many metaheuristics have been developed to solve these problems since the 1950s (Scholl and Becker 2006). Among the different metaheuristic algorithms, PSO has achieved great success in optimizing engineering problems (Guo *et al.* 2013). PSO is a population-based stochastic optimization technique developed by Kennedy and Eberhart (1995). PSO was inspired by the social behaviour of organisms, such as birds flocking and fish schooling when they search for food. The basic advantages of using the PSO algorithm to solve this problem (Lee and Park 2006; Hu *et al.* 2014) are:

(1) PSO does not have any overlapping and mutation calculation.
(2) The implementation of PSO is very simple compared with other heuristic algorithms.
(3) PSO requires fewer calculations and there are few parameters to tweak.

The pragmatic issue in using this or any other metaheuristics algorithm is to find optimal control parameters. The PSO parameters optimized in this research are initial velocity, initial population, acceleration coefficients and stopping condition. The methods followed to obtain the optimal parameters are explained in Section 6.2.

PSO starts with a population of randomly generated initial solutions and the optimal solution is searched iteratively in the problem space. Solutions are called particles (swarm), which move in the search space, and each particle represents a possible solution (fitness). All particles move around with certain velocity. Each particle remembers the best result achieved so far (personal best) and exchanges information with other particles to determine the best particle (global best) among the swarm. In each generation, the velocity of each particle is updated to its best encountered position and the best position encountered by any particle (Shyh Chyan and Ponnambalam 2012). Pseudo-code of the PSO algorithm is presented in Figure 3. The implementation details of PSO are discussed in this section.

```
$t \rightarrow 0;$

$for(i = 1, N)$
    $Generate\ \ P_i^t;$
    $Evaluate\ Z\ (P_i^t);$
    ${}^eP_i^t \rightarrow P_i^t;$
$end\ i$
$G \rightarrow P_i^t\ \ having\ min\ \{Z({}^eP_i^t);\ \ \ i = 1, N\ \}$

$for(i = 1, N)$
    $Initialize\ \ v_i^t;$
$end\ i$
$do\ \{$
        $for(i = 1, N)$
                $Update\ Velocity\ \ v_i^{t+1}\ \ (Eq.(8))$
                $Update\ Position\ \ P_i^{t+1}\ (Eq.(9)\ )$
        $end\ i$
        $Evaluate\ all\ particles$
        $Update\ {}^eP_i^t\ \ and\ G, (i = 1, N)$
        $t\ \ \rightarrow t + 1$
$\}\ (\ while\ (t < t_{max}))$
$Output\ G$
```

Figure 3. Pseudo-code of particle swarm optimization.

5.1. *Initial swarm generation*

Metaheuristics algorithms generally start with a randomly generated search space which evolves iteratively to find nearer to optimal solutions (Rahimi-Vahed, Mirghorbani, and Rabbani 2007). Instead of starting the search process from a set of random solutions, a set of priority rules reported in the literature is used to reach better solution at a faster rate. Ponnambalam, Aravindan, and Naidu (2000) used 14 priority rules to generate the initial population for the multi-objective GA proposed to solve a simple assembly line balancing (sALB) problem. Out of these 14 rules, six rules are chosen in this research to generate initial particles for the proposed PSO. The remaining eight rules are not selected because they are not suitable for the problem considered in this research. The quality of the solution mainly relies on the PSO search process. Remaining swarm particles are randomly generated. There are 25 particles in the initial swam. The particle represents a sequence of numbers (tasks) arranged in such a way that it meets the precedence relationship. Table 1 shows the six methods and the particles generated, for illustration purposes. Robot task performance times are presented in Table 2 and the precedence graph in Figure 4 is used as the input for creating the particles. In Figure 4, the numbers inside the circles represent task numbers. Each particle presents the task in an order that represents a feasible solution. These

Table 1. Initial population generated using the priority rules.

Rule no.	Rule	Task sequence										
1	Maximum ranked positional weight	1	2	6	3	4	5	7	8	10	9	11
2	Minimum reverse positional weight	1	5	4	3	2	7	9	6	8	10	11
3	Minimum total number of predecessor tasks	1	2	3	4	5	6	8	10	7	9	11
4	Maximum total number of follower tasks	1	2	3	4	5	6	7	8	9	10	11
5	Maximum task time	1	5	2	6	3	4	7	8	10	9	11
6	Minimum task time	1	4	3	2	5	7	9	6	8	10	11

Table 2. Performance time for 11 tasks by four robots.

Tasks	Robot 1	Robot 2	Robot 3	Robot 4	Average time (τ)
1	81	37	51	49	54.5
2	109	101	90	42	85.5
3	65	80	38	52	58.75
4	51	41	91	40	55.75
5	92	36	33	25	46.5
6	77	65	83	71	74
7	51	51	40	49	47.75
8	50	42	34	44	42.5
9	43	76	41	33	48.25
10	45	46	41	77	52.25
11	76	38	83	87	71

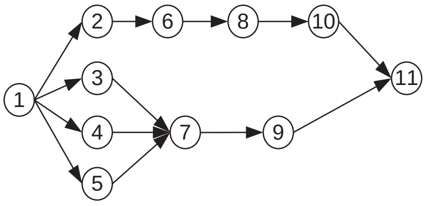

Figure 4. Precedence graph of 11-task problem (numbers in circles represent the task numbers).

particles will be modified iteratively based on collective experiences to improve their solution quality.

5.2. *Position and velocity update*

Using the best position encountered by the particle and the best position encountered among the swarm, the velocity of the particle is updated iteratively, using Equation (8).

$$vi^{t+1} = c_1 v_i{}^t + c_2 U_1({}^eP_i{}^t - P_i{}^t) + c_3 U_2(G - P_i{}^t) \tag{8}$$

where U_1 and U_2 are velocity coefficients (random numbers between 0 and 1), v_i^t is the velocity, ${}^eP_i^t$ is the local best, G is the global best, P_i^t is the current particle position at generation t, and c_1, c_2 and c$_3$ are the learning coefficients. An example is shown to explain how the velocity and position are updated.

Particles move from their current position to the new position using Equation (9). Each particle's position is updated in each generation by adding the velocity vector to the position vector:

$$P_i{}^{t+1} = P_i{}^t + v_i{}^{t+1} \tag{9}$$

The following may be assumed.

Local best, ${}^eP_i^t$: (1,2,6,3,4,5,7,8,10,9,11), global best, G: (1,2,3,4,5,6,7,8,9,10,11).

Particle, P_i^t: (1,2,3,6,5,4,7,8,10,9,11), initial velocity, v_i^t: (2, 3) (4, 5).

These sequences represent the assembly tasks arranged. Parameters chosen for the example are: $c_1 = 1$, $c_2 = 1$ and $c_3 = 2$ are the acceleration coefficients and U_1, U_2 are 0.8 and 0.3.

The velocity of the particle is calculated using Equation (8):

$$\begin{aligned} cv_i^{t+1} &= (2,3)(4,5) + 0.8 * [(1,2,6,3,4,5,7,8,10,9,11) - (1,2,3,6,5,4,7,8,10,9,11)] \\ &\quad + 0.6 * [(1,2,3,4,5,6,7,8,9,10,11) - (1,2,3,6,5,4,7,8,10,9,11)] \end{aligned}$$

$$= (2,3)(4,5) + 0.8 * (2,3)(4,5) + 0.6 * (3,5)(8,9)$$
$$= (2,3)(4,5)(8,9)$$

The position of the particle is updated using Equation (9):

$$P_i^{t+1} = (1,2,3,6,5,4,7,8,10,9,11) + (2,3)(4,5)(8,9) = (1,3,2,5,6,4,7,10,8,9,11)$$

5.3. *Velocity generation*

Initial velocities for the particles are randomly generated and represent the number of pairs of transpositions. Table 3 shows the maximum number of velocity pairs used in this article. The velocity update equation (Equation 8) is used from the second iteration onwards.

5.4. *Fitness evaluation (task and robot allocation)*

An efficient method is used to allocate tasks and robots to workstations. A consecutive heuristic procedure proposed by Levitin, Rubinovitz, and Shnits (2006) for straight line assembly line is adopted in this article. This heuristic allocates tasks and robots to workstations with an objective of minimizing the cycle time of the assembly line. The initial cycle time of the assembly line, C_0, is to be considered to start the procedure. The procedure tries to allocate the maximum number of tasks to be performed at each workstation. For a U-shaped assembly line, tasks are to be allocated to the workstations by moving forwards and backwards through the precedence diagram, in contrast to a typical forward move in the traditional assembly systems. Robots are checked for allotment to the workstation to perform the allotted tasks. If certain tasks cannot to be allocated to workstations for the given initial C_0 value, C_0 is incremented by 1 and this procedure continues until all the tasks have been assigned to all the workstations. The stepwise procedure of the consecutive heuristic is explained in this section.

Step 1. The initial value of C_0 is the mean of the minimum performance time of robots for the tasks. The initial assembly line time is:

$$C_0 = \left\lceil \frac{\sum_{j=1}^{N_a} \min\limits_{1 \le i \le N_r} t_{ih}}{N_w} \right\rceil \tag{10}$$

The following feasible sequence of tasks (Figure 5) is considered for illustration. Initial C_0 is calculated using the robot performance times as shown in Table 2.

Table 3. Maximum number of velocity pairs.

Task range	Maximum velocity pairs
0–20	4
20–40	8
40–60	10
60–80	25
80–100	30
100–120	40
120–140	50
140–200	65
200–300	75

1	3	2	4	5	6	7	9	8	10	11

Figure 5. Example sequence considered for illustration.

The sequence meets the precedence constraints. Initial C_0 for the example is found to be 109, where 37,42,38,40,25,65,50,34,33,41,38 are the minimum robot task times (refer to Table 2):

$$C_0 = \frac{37 + 42 + 38 + 40 + 25 + 65 + 40 + 34 + 33 + 41 + 38}{4} = 108.25.$$

Step 2. Open the first station and allocate tasks in such a way that tasks are chosen from either side of the sequence, if one or more robot can perform the allocated tasks within C_0. The set of assignable tasks is determined by all those tasks whose predecessors or successors have already been assigned. Tasks are allocated to workstations by moving forwards and backwards through the precedence diagram. Each workstation s has a set of preferred/allotted robots H, which is defined as follows:

$$r \in H, \text{if} m(r) \geq m(h), \quad for 1 \leq h \leq N_r \tag{11}$$

where m (h) is the maximal number of tasks a robot h can perform in the given sequence sq during a time less than C_0.

$$T_s(h) = \sum_{r=p1_s}^{p1_s+m(h)} t_{h,sq(r)} < C_0 \leq \sum_{r=p1_s}^{p1_s+m(h)+1} t_{h,sq}(r) \tag{12}$$

Next, the robot to be assigned to the workstation s is defined as:

$$h(s) = r, \text{if} T_s(r) < T_s(h) \quad \forall h \in H \tag{13}$$

Step 3. The start position of the next station ($p1_{s+1}$) is calculated:

$$p1_{s+1} = pr_s + 1 = p1_s + m(h(s)) + 1 \tag{14}$$

Repeat Steps 2 and 3 until all tasks have been assigned to the given number of workstations**.**

Step 4. If it is not possible to assign all tasks to the given number of workstations, C_0 is incremented by 1 and Steps 2 and 3 are repeated until all tasks have been allotted to the given number of workstations.

Step 5. The best fit robot is assigned to each workstation. Table 4 shows how the best fit robot is selected for the example shown in Figure 7. Three tasks (1, 11 and 10) are assigned to workstation 1 and robot 2 is assigned to this workstation, as explained in Table 4. Robot 2 performs the assigned tasks in a shorter time than the other robots. Hence, robot 2 is the best fit robot for workstation 1. The total robot task time for each robot is calculated using Table 2.

Step 6. The maximum workstation time (sum of the minimum robot performance time for the tasks assigned to the station) is the cycle time for the sequence.

Table 4. Illustration of best fit robot selection.

Workstation	Tasks assigned	Total robot task time
1	1, 10 and 11	Robot 1: $81 + 45 + 76 = 202$
		Robot 2: $37 + 46 + 38 = 121$
		Robot 3: $51 + 41 + 83 = 175$
		Robot 4: $49 + 77 + 87 = 213$

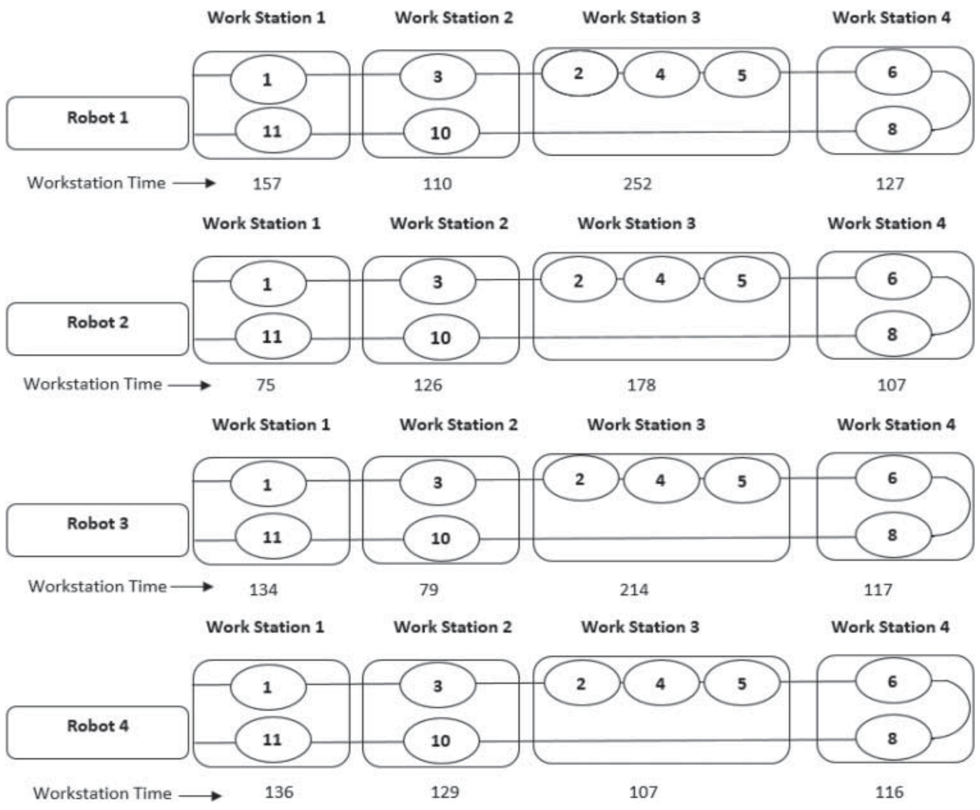

Figure 6. Example of assignment procedure for initial cycle time (numbers in ovals represent the task numbers).

For the sequence shown in Figure 5, C_0 is calculated and is found to be 109. Initially, the algorithm tries to allocate the tasks to the workstations and assign the robots within 109, and it is found that tasks 7 and 9 are left unassigned, as shown in Figure 6. C_0 is incremented by 1 to accommodate all the tasks and the procedure finds a solution for all four workstations after C_0 reaches 121, as shown in Figure 7. The shaded portion in Figure 7 shows the tasks and robot allocation details.

5.5. *Differences between straight and U-shaped robotic assembly lines*

In a straight line configuration, workstations are located along a straight line. In the case of a U-shaped assembly line, the line is configured into a U shape. Figure 8 shows a sample solution where tasks are assigned in a straight line for the sequence mentioned in Section 5.4. Figure 9 shows a sample solution for tasks assigned in a U-shaped configuration.

From Figure 8 it can be understood that tasks are assigned in the order of the sequence generated without violating the precedence constraints for the straight robotic assembly line. In the case of a U-shaped assembly line, each task and any of its successors and/or predecessors can share the same station. Tasks are allocated by searching forwards and backwards through the precedence diagram. Hence, tasks from the beginning and end of the precedence diagram can be assigned to the same station. In U-shaped allocations, it is easier to relocate the robot to balance the workload depending on the demand. This flexibility and adaptability in U-shaped lines make them an attractive approach compared to straight lines. The cycle time for the straight robotic assembly line is 143 and for the U-shaped robotic assembly line it is 121.

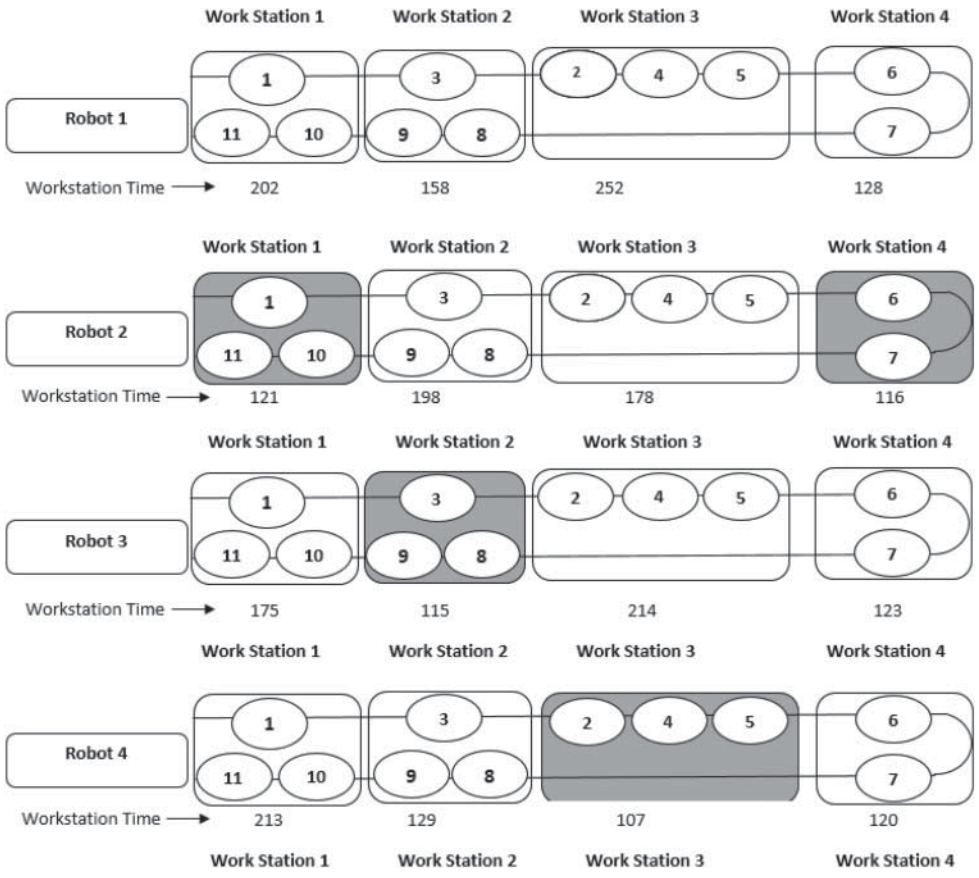
Work Station 1
Work Station 2
Work Station 3
Work Station 4
Robot 1
Workstation Time 202 158 252 128
Robot 2
Workstation Time 121 198 178 116
Robot 3
Workstation Time 175 115 214 123
Robot 4
Workstation Time 213 129 107 120

Figure 7. Final assignment solution. (numbers in ovals represent the task numbers).

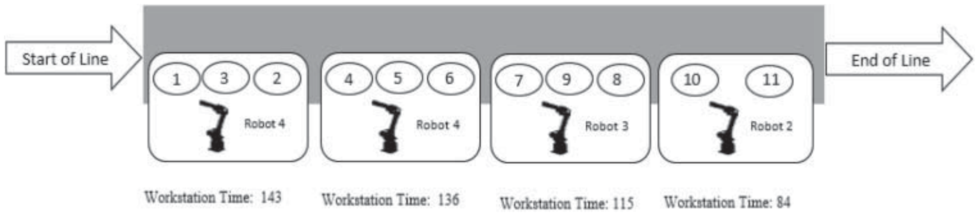

Figure 8. Straight robotic assembly line (numbers in ovals represent the task numbers).

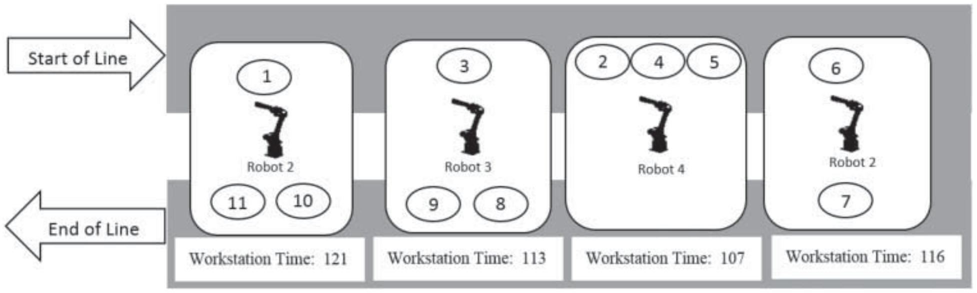

Figure 9. U-shaped robotic assembly line (numbers in ovals represent the task numbers).

6. Computational results

Computational experiments are conducted to test the performance of the proposed PSO on RUALB problems. The algorithm is coded in C + + and tested on an Intel core i5 processor (2.3 GHz). Details of the experiments conducted are presented in this section.

6.1. *Data set used for performance evaluation*

Scholl (1995) generated problems with tasks ranging from 25 to 297 for SALB-I, using eight representative precedence graphs, which are available at http://www.assembly-line-balancing.de/. Gao *et al.* (2009) generated 32 test problems for a robotic assembly line by adding the robot task times in the original data sets prepared by Scholl (1995). These data sets are used in the present research.

6.2. *Parameters used for particle swarm optimization*

The performance of PSO relies mainly on the parameters selected. The parameters used in this article are chosen based on the experiments conducted to obtain a satisfactory solution quality in an acceptable computational time. Experiments are performed to test the influence of each parameter on the solution quality. Three data sets of different sizes are chosen to find the best parameters. Different combinations of the parameters are tested until the best combination is achieved. Solution quality is given more importance than computational time in selecting the parameters. The following parameters are tested and used in the proposed PSO.

- Stopping condition: The proposed method is terminated if the iteration approaches a predefined criterion, usually a sufficiently good fitness; in this case, a predefined maximum number of iterations (generations) is used. Different stopping conditions are tested, such as 5, 10, 15, 25 and 30, and the best solution is obtained when the number of generations is 25. Figure 10 illustrates the performance of the algorithm based on the stopping condition for three data sets (small-, medium- and large-sized data sets). Three problems are taken into consideration for generating the graph. Problems are 35–12, 70–19 and 148–29, where 35, 70 and 148 are the number of tasks and 12, 19 and 29 indicate the number of robots and workstations available to perform the tasks.
- Acceleration coefficients: Different combinations of acceleration coefficients are tested. Table 5 shows the different combinations of c_1, c_2 and c_3 tested. Three problems are taken into consideration for the testing. Problems selected are 53–5, 297–19 and 148–14, where 53, 148 and 297 refer to the number of tasks and 5, 19 and 14 indicate the number of robots and workstations available to perform the tasks. Figure 11 shows that group D has the best combination of acceleration coefficients.

6.3. *Performance of U-shaped versus straight robotic assembly line*

All 32 test instances are evaluated using the proposed PSO algorithm. The non-deterministic nature of the algorithm and problem makes it necessary to run the same problem multiple times. Each problem is run 10 times and most of the runs converge to the same solution for each of the problems. The results obtained by evaluating 32 test problems are presented in Table 6. Column 1 lists the type of problem. The problems are classified into three categories: small (up to 35 tasks), medium (up to 89 tasks) and large (above 100 tasks). Column 2 in Table 6 lists the problem names. Columns 3 and 4 show the number of tasks and number of workstations

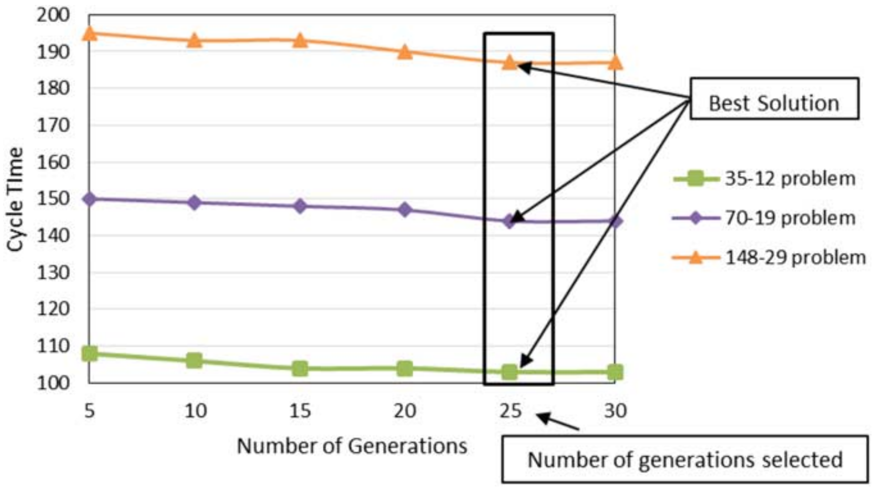

Figure 10. Comparison of the performance of particle swarm optimization in terms of stopping condition.

Table 5. Selection of c_1, c_2 and c_3.

Group	c_1	c_2	c_3
A	1	1	1
B	1	1	2
C	1	2	1
D	1	2	2
E	1	3	3
F	1	4	4
G	1	3	4
H	1	2	4

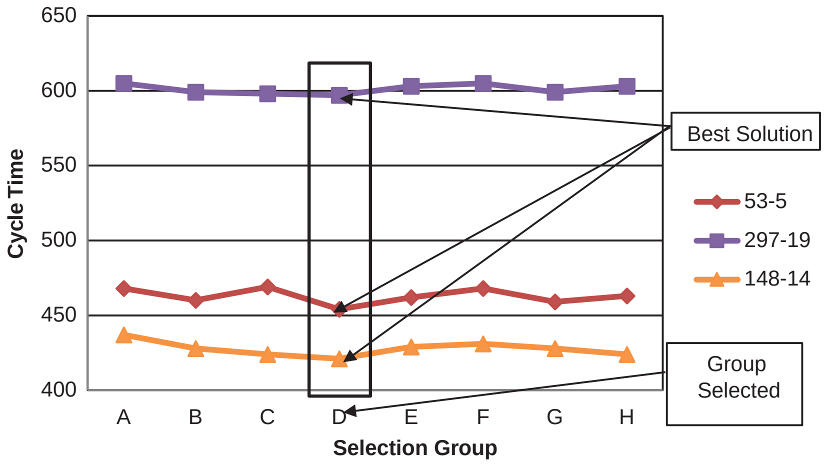

Figure 11. Selection of acceleration coefficients based on the performance of the algorithm.

considered for the evaluation. Column 5 shows the WEST ratios of the problems. Section 6.4 explains the calculation of the WEST ratio for the problems considered in this article. Column 6 reports the cycle time for a straight robotic assembly line, as reported by Gao *et al.* (2009). Column 7 shows the results obtained from LINGO (optimization software). LINGO can only find

Table 6. Results of the 32 benchmark problems.

Problem type	Problem name	Tasks	Workstations/ robots	WEST ratio	Cycle time: Straight RALB (Gao *et al.* 2009)	Cycle time: LINGO	Cycle time: PSO RUALB	Percentage deviation (%)	Average percentage deviation (%)	Computational time (s)	Relative complexity
Small-sized problem	Rosenberg	25	3	8.33	503	500*	500	0.5	6.99	8.0	0
			4	6.25	327	318*	318	2.75		9.2	0.15
			6	4.17	213	188*	188	11.73		10.5	0.31
			9	2.78	123	114*	114	7.31		13.5	0.68
	Gunther	35	4	8.75	449	–	355	20.93		16.8	1.10
			5	7.00	344	–	332	3.48		19.5	1.43
			7	5.00	222	–	221	0.45		27.5	2.43
			12	2.92	113	–	103	8.84		31.5	2.93
Medium-sized problem	Hahn	53	5	10.60	554	–	459	17.14	6.32	32.5	3.06
			7	7.57	320	–	286	10.62		34.8	3.35
			10	5.30	230	–	220	4.34		35.6	3.45
			14	3.39	162	–	148	8.64		41.2	4.15
	Tonge	70	7	10.00	449	–	447	0.44		60.1	6.51
			10	7.00	272	–	272	0		66.8	7.35
			14	5.00	204	–	211	–3.43		72.4	8.05
			19	3.68	154	–	144	6.49		82.2	9.27
	Lutz	89	8	11.13	494	–	496	–0.49		84.5	9.56
			12	7.42	370	–	326	11.89		87.1	9.88
			16	5.56	236	–	224	5.08		91.2	10.40
			21	4.24	205	–	174	15.12		95.3	10.91
Large-sized problem	Arcus	111	9	12.33	557	–	545	2.15	4.85	234.2	28.27
			13	8.54	319	–	320	–0.313		253.7	30.71
			17	6.53	257	–	256	0.38		298.5	36.31
			22	5.05	192	–	186	3.12		348.9	42.61
	Bartholdi	148	10	14.80	600	–	629	–4.83		445.8	54.72
			14	10.57	427	–	421	1.40		519.2	63.9
			21	7.05	300	–	283	5.66		595.1	73.38
			29	5.10	202	–	187	7.42		655.3	80.91
	Scholl	297	19	15.63	646	–	597	7.58		1573.2	195.65
			29	10.24	430	–	394	8.37		1693.8	210.72
			38	7.82	344	–	293	14.8		1752.9	218.11
			50	5.94	256	–	224	12.5		1802.3	224.28

Note: Optimal solutions found with LINGO.

optimal solutions for four small-sized problems (25–3, 25–4,2 5–6 and 25–9) in an acceptable computational time, and cannot obtain solutions for the rest of the problems. Column 8 reports the cycle time obtained for the same problems for the U-shaped robotic assembly line using the proposed PSO. Column 9 shows the percentage deviation of the cycle time and column 10 reports the average percentage deviation for the three problem categories. Column 11 shows the computational time for the problems. Column 12 shows the relative complexity. The calculation of relative complexity is explained in Section 6.4.

The results presented in Table 6 indicate that the proposed PSO algorithm finds a better solution for 28 out of the 32 instances. The proposed PSO generates better results for all small-sized problems compared to the straight line allocation. The average percentage improvement for the small size category is found to be 6.99%. The average percentage improvement in cycle time for medium-sized problems is 6.32%. It is observed that only two out of 12 problems in this category could not yield a better solution. The average percentage improvement for the large size category is 4.85%. It can be seen that 10 out of 12 problems could yield a better solution compared with the straight robotic assembly line. The results presented in Table 6 reveal that the cycle time of a U-shaped robotic assembly line is better than that of a straight robotic assembly line. For the problem addressed here, selection of the best available robots helps to reduce the cycle time and, in turn, increases the productivity of the assembly line.

An actual assembly line balancing problem is used to test the proposed algorithm. The final assembly operation of a major automobile manufacturer, comprising the assembly of the engine cradle, is considered (Gunther, Johnson, and Peterson 1983). There are 35 tasks to be performed and five robots are chosen to perform these 35 tasks on the assembly line. The problem consists of 45 direct precedence relations among the 35 tasks, and the processing times of the 35 tasks by the five robots are shown in Table 7. The solutions obtained for the problem using PSO for a U-shaped assembly line, and using a hybrid GA for a straight robotic assembly line are reported in Table 8. The table shows details of the tasks and robots to be allocated at each workstation. It can be clearly seen that the cycle time for the U-shaped robotic assembly line is lower than that for the straight robotic assembly line.

6.4. *Complexity of the problem*

The computational complexity of the sALB is known to be a non-deterministic polynomial NP-hard problem (Karp 1972). Faaland *et al.* (1992) stated that procedures attempting to find the optimal solution have a complexity of at least 2^N. Hence, developing metaheuristic procedures to solve problems of a practical size remains the only option for researchers. Measures reported in the literature could be used to show the complexity of RUALB. The F-ratio, proposed by Mansoor and Yadin (1971), the WEST ratio, proposed by Dar-El (1973), and relative complexity, proposed by Bhattacharjee and Sahu (1990), are used in this article to show the complexity of RUALB problems. It has already been proven by many researches that the sALB problem is NP hard (Gutjahr and Nemhauser 1964). The RUALB problem is further complicated by the addition of robots and the U-shaped configuration. Therefore, RUALB is also NP hard.

To measure the complexity of the given RUALB problem, the total nodes need to be calculated. The total nodes measures the time required to reach a solution, by counting the total number of nodes generated in the search process. The total number of nodes of the problem is directly proportional to the number of iterations in the algorithm, and hence, the computation time (Rubinovitz, Bukchin, and Lenz 1993).

The following parameters are used to characterize the RALB problem complexity:

1. Assembly flexibility: The F-ratio measures the flexibility in creating assembly sequences. It was developed by Mansoor and Yadin (1971) and is defined as follows. Let P_{ij} be an element

Table 7. Performance times of 35 task problem with five robots.

Task	Predecessor tasks	Robot 1	Robot 2	Robot 3	Robot 4	Robot 5
1	–	142	67	88	56	84
2	1	92	45	183	56	69
3	2	56	25	36	37	37
4	3	64	61	53	45	47
5	1	62	29	92	35	95
6	5	68	51	132	83	177
7	1,6	93	90	137	71	158
8	6	59	73	90	51	116
9	8	29	36	36	51	50
10	1	53	55	37	36	43
11	4	63	40	85	59	51
12	1	42	73	49	109	49
13	9	42	36	91	64	47
14	7,10	77	46	93	68	66
15	14	37	45	50	37	83
16	15	28	28	73	47	37
17	–	65	49	41	53	51
18	7,12	93	49	63	151	101
19	18	103	37	62	54	89
20	17,19	38	55	38	29	34
21	16,20	51	83	122	67	117
22	21	36	43	57	47	60
23	22	70	87	74	63	146
24	23	42	83	108	49	49
25	21	103	55	66	54	83
26	25	36	78	64	34	48
27	24,26	44	82	49	68	46
28	11,13	105	36	69	119	94
29	28	58	31	59	37	60
30	21	43	37	59	53	64
31	30	42	32	51	82	113
32	21,31	40	39	89	45	91
33	11,13,27,32	32	31	99	57	46
34	27	93	37	50	53	91
35	33	37	50	72	84	53

of a precedence matrix P, such that:

$$P_{ij} = \begin{cases} 1 & \text{if task } i \text{ precedes task } j \text{ is assigned to robot } h \text{ in stations} \\ 0, & \text{otherwise} \end{cases}$$

Then, the F-ratio is calculated as: $\mathrm{F-Ratio} = 2\mathrm{x}Z/(N_a\mathrm{x}(N_a - 1))$, where Z is the number of zeroes in P, and N_a is the number of assembly tasks. The F-ratio value is therefore between 0 and 1. When there are no precedence constraints between tasks (any sequence is feasible) the F-ratio is 0, and when only a single assembly sequence is feasible the F-ratio is 1. Assembly tasks are often characterized by relatively low F-ratios. Problems with eight levels of F-ratio are generated and evaluated. Figure 12 shows F-ratio versus computational time for eight problems sets (see column 2 in Table 6). It is noted that computational time is an increasing function of F-ratio. A high F-ratio indicates that there are fewer alternatives for assigning tasks to workstations, whereas a low F-ratio indicates different ways of assignment. The complexity of the line balancing problem depends on the F-ratio (Bhattacharjee and Sahu 1990).

2. WEST ratio: This ratio, defined by Dar-El (1973), measures the average number of tasks per workstation. This measure shows the expected quality of achievable solutions and the complexity of the problem. Gao *et al.* (2009) generated WEST ratios ranging from 2 to 15

Table 8. Solutions obtained for 35 task problem with five robots.

Workstation no.	Tasks allocated	Robot	Workstation time
Hybrid GA task sequence for straight RALB: 1 5 10 6 7 8 14 12 17 18 19 9 20 15 16 21 22 23 24 13 25 26 27 2 3 4 30 34 31 32 11 28 33 35 29			
Workstation 1	1 5 10 6 7 8	Robot 4	332
Workstation 2	14 12 17 18 19 9	Robot 3	344
Workstation 3	20 15 16 21 22 23 24 13	Robot 1	344
Workstation 4	25 26 27 2 3 4	Robot 5	330
Workstation 5	30 34 31 32 11 28 33 35 29	Robot 2	333
PSO task sequence for RUALB: 1 12 10 5 6 7 18 14 19 15 2 17 16 20 21 22 30 8 23 3 25 31 4 9 11 13 24 26 27 32 28 33 34 35 29			
Workstation 1	1 29 34 2 3 35 28 33	Robot 2	322
Workstation 2	4 27 26 32 31 30 24	Robot 1	311
Workstation 3	11 17 10 5 6 8 9	Robot 2	333
Workstation 4	7 23 25 22 21 20	Robot 4	331
Workstation 5	14 19 15 16 13 18 12	Robot 2	314

Note: GA = genetic algorithm; RALB = robotic assembly line balancing; PSO = particle swarm optimization; RUALB = robotic U-shaped assembly line balancing.

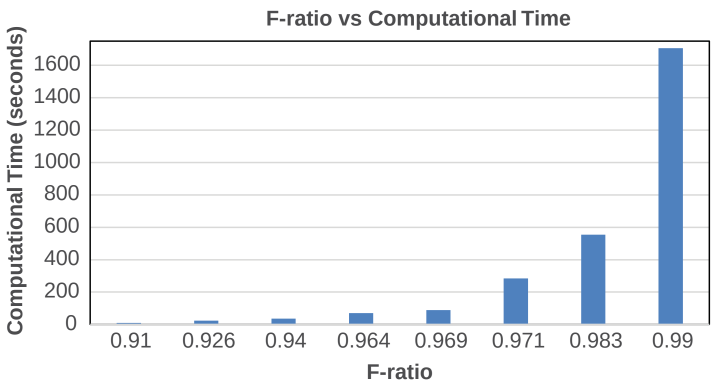

Figure 12. F-ratio vs computational time.

for 32 RALB problems. For each problem, the number of workstations is equal to the number of robots, and every task can be processed on any robot. The WEST ratios considered in this research are shown in Table 6.

3. Relative complexity: $R = (V - Q)/Q$, where V is the computational time for the solution of the problem whose complexity is to be measured, and Q is the minimum computational time for the set of the problems under study (Bhattacharjee and Sahu 1990). Table 6 shows the relative complexity of the allocation for a U-shaped robotic assembly line. The relative complexity shown in Table 6 significantly increases when the problem size increases.

7. Conclusion

The robotic U-shaped assembly line balancing (RUALB) problem is considered in this article. The literature on this problem is very limited. The PSO algorithm is proposed to solve this problem to minimize the cycle time of the assembly line. The tasks and robot assignment in the U-shaped configuration are highly complex compared to a straight assembly line. Thirty-two benchmark problems originally proposed by earlier researchers to solve RALB were adopted to the test the performance of RUALB. Extensive computational experiments were conducted and the results are reported. The results of the experiments show that the proposed PSO algorithm for a robotic U-shaped assembly line reports minimal cycle time compared to that of a straight robotic assembly line for 28 out of 32 problems. The complexity of RUALB is also presented with various complexity measures. The computational time for the developed algorithm is high for large data sets, which could be due to the large search space. The quality of the solution is given more importance than the computational time in this study. The performance of the proposed PSO on the RUALB reported in this article is for the benchmark problems with eight precedence graphs only.

The limitations of this work are that only one robot can be assigned to only one workstation. It cannot handle multiple workstations, the workstations in the assembly line cannot split the tasks and the computational time increases significantly when the problem size increases. Some areas for future research could include RALB problems for balancing and model sequencing of two-sided assembly lines, mixed-model (straight or U-shaped robotic) assembly lines, balancing of parallel robotic assembly lines and multi-objective models for U-shaped robotic assembly lines.

References

Aigbedo, H., and Y. Monden. 1997. “A Parametric Procedure for Multicriterion Sequence Scheduling for Just-in-Time Mixed-Model Assembly Lines.” *International Journal of Production Research* 35 (9): 2543–2564.

Ajenblit, Debora A., and Roger L. Wainwright. 1998. “Applying Genetic Algorithms to the U-shaped Assembly Line Balancing Problem.” Proceedings of IEEE World Congress on Computational Intelligence. pp. 96–101.

Akpinar, Sener, and G. Mirac Bayhan. 2014. “Performance Evaluation of Ant Colony Optimization-Based Solution Strategies on the Mixed-Model Assembly Line Balancing Problem.” *Engineering Optimization* 46 (6): 842–862.

Avikal, Shwetank, Rajeev Jain, P. K. Mishra, and H. C. Yadav. 2013. “A Heuristic Approach for U-shaped Assembly Line Balancing to Improve Labor Productivity.” *Computers & Industrial Engineering* 64 (4): 895–901.

Battaïa, Olga, and Alexandre Dolgui. 2013. “A Taxonomy of Line Balancing Problems and their Solution Approaches.” *International Journal of Production Economics* 142 (2): 259–277.

Baykasoglu, Adil. 2006. “Multi-rule Multi-objective Simulated Annealing Algorithm for Straight and U Type Assembly Line Balancing Problems.” *Journal of Intelligent Manufacturing* 17 (2): 217–232.

Bhattacharjee, T. K., and S. Sahu. 1990. “Complexity of Single Model Assembly Line Balancing Problems.” *Engineering Costs and Production Economics* 18 (3): 203–214.

Boeing. 2014. “Boeing 777 Moving Production Line.” Boeing. Accessed October 29. http://www.boeing.com/boeing/commercial/777family/777movingline.page

Chiang, Wen-Chyuan, and Timothy L. Urban. 2006. “The Stochastic U-line Balancing Problem: A Heuristic Procedure.” *European Journal of Operational Research* 175 (3): 1767–1781.

Daoud, Slim, Hicham Chehade, Farouk Yalaoui, and Lionel Amodeo. 2014. “Solving a Robotic Assembly Line Balancing Problem Using Efficient Hybrid Methods.” *Journal of Heuristics* 20 (3): 235–259.

Dar-El, E. M. 1973. “MALB—A Heuristic Technique for Balancing Large Single-Model Assembly Lines.” *AIIE Transactions* 5 (4): 343–356.

Erel, E., I. Sabuncuoglu, and B. A. Aksu. 2001. “Balancing of U-type Assembly Systems Using Simulated Annealing.” *International Journal of Production Research* 39 (13): 3003–3015.

Faaland, Bruce H., Theodore D. Klastorin, Thomas G. Schmitt, and Avraham Shtub. 1992. “Assembly Line Balancing with Resource Dependent Task Times.” *Decision Sciences* 23 (2): 343–364.

Fetters-Walp, Eric. 2010. “Moving to a Quicker Pace.” http://www.boeing.com/Features/2010/05/bca_moving_line_05_24_10.html.

Gao, Jie, Linyan Sun, Lihua Wang, and Mitsuo Gen. 2009. “An Efficient Approach for Type-II Robotic Assembly Line Balancing Problems.” *Computers & Industrial Engineering* 56 (3): 1065–1080.

Gökçen, Hadi, Kürşat Ağpak, Cevriye Gencer, and Emel Kizilkaya. 2005. “A Shortest Route Formulation of Simple U-type Assembly Line Balancing Problem.” *Applied Mathematical Modelling* 29 (4): 373–380.

Gunther, Richard E., Gordon D. Johnson, and Roger S. Peterson. 1983. “Currently Practiced Formulations for the Assembly Line Balance Problem.” *Journal of Operations Management* 3 (4): 209–221.

Guo, Weian, Wuzhao Li, Qun Zhang, Lei Wang, Qidi Wu, and Hongliang Ren. 2013. “Biogeography-Based Particle Swarm Optimization with Fuzzy Elitism and its Applications to Constrained Engineering Problems.” *Engineering Optimization* 46 (11): 1465–1484.

Gutjahr, Allan L., and George L. Nemhauser. 1964. “An Algorithm for the Line Balancing Problem.” *Management Science* 11 (2): 308–315.

Hiroyuki, HIRANO, and J. T. Black. 1988. *JIT Factory Revolution*. Cambridge, MA: Productivity Press.

Hu, Xiaomei, Yangyang Zhang, Ning Zeng, and Dong Wang. 2014. “A Novel Assembly Line Balancing Method Based on PSO Algorithm.” *Mathematical Problems in Engineering* Volume(2014), Article ID 743695: Pages 10.

Hwang, Rea Kook, Hiroshi Katayama, and Mitsuo Gen. 2008. “U-shaped Assembly Line Balancing Problem with Genetic Algorithm.” *International Journal of Production Research* 46 (16): 4637–4649.

Kara, Yakup. 2008. “Line Balancing and Model Sequencing to Reduce Work Overload in Mixed-Model U-line Production Environments.” *Engineering Optimization* 40 (7): 669–684.

Karp, Richard M. 1972. “Reducibility among Combinatorial Problems.” In *Complexity of Computer Computations*, edited by Raymond E. Miller, James W. Thatcher, and Jean D. Bohlinger, 85–103. New York: Springer US.

Kennedy, James, and Russell Eberhart. 1995. “Particle Swarm Optimization”. Proceedings of IEEE International Conference on Neural Networks. pp. 1942–1948.

Khaw, Christopher LE, and Sivalinga Govindarajan Ponnambalam. 2009. “Multi-rule Multi-objective Ant Colony Optimization for Straight and U-type Assembly Line Balancing Problem.” Proceedings of IEEE International Conference on Automation Science and Engineering (CASE). pp. 177–182.

Kim, Yeo Keun, Sun Jin Kim, and Jae Yun Kim. 2000. “Balancing and Sequencing Mixed-Model U-lines with a Co-evolutionary Algorithm.” *Production Planning & Control* 11 (8): 754–764.

Kim, Yeo Keun, Jae Yun Kim, and Yeongho Kim. 2006. “An Endosymbiotic Evolutionary Algorithm for the Integration of Balancing and Sequencing in Mixed-model U-lines.” *European Journal of Operational Research* 168 (3): 838–852.

Kubota, Yoko. 2011. “Toyota’s New Assembly Line Saves Time, Costs: Report.” http://www.reuters.com/assets/print?aid=USTRE70M07H20110123.

Lee, Kwang Y., and Jong-Bae Park. 2006. “Application of Particle Swarm Optimization to Economic Dispatch Problem: Advantages and Disadvantages”. Proceedings of IEEE Power Systems Conference and Exposition, PSCE’06 pp.188–192.

Levitin, Gregory, Jacob Rubinovitz, and Boris Shnits. 2006. “A Genetic Algorithm for Robotic Assembly Line Balancing.” *European Journal of Operational Research* 168 (3): 811–825.

Mansoor, Ezekiel Meir, and Micha Yadin. 1971. “On the Problem of Assembly Line Balancing.” In *Developments in Operations Research*, edited by B. Avi-Itzhak, 361–375. New York: Gordon and Breach.

Martinez, Ulises, and William S. Duff. 2004. “Heuristic Approaches to Solve the U-shaped Line Balancing Problem Augmented by Genetic Algorithms.” Proceedings of IEEE Systems and Information Engineering Design Symposium. pp. 287–293.

Miltenburg, John. 1998. “Balancing U-lines in a Multiple U-line Facility.” *European Journal of Operational Research* 109 (1): 1–23.

Miltenburg, G. J., and Jacob Wijngaard. 1994. “The U-line Line Balancing Problem.” *Management Science* 40 (10): 1378–1388.

Monden, Yasuhiro. 1995. “Toyota Production System.” *Journal of the Operational Research Society* 46 (5): 669–670.

Mukund Nilakantan, J., and S. G. Ponnambalam. 2012. “An Efficient PSO for type-II Robotic Assembly Line Balancing Problem.” Proceedings of IEEE International Conference on Automation Science and Engineering (CASE), pp. 600–605.

Nakade, Koichi, and Katsuhisa Ohno. 1999. “An Optimal Worker Allocation Problem for a U-shaped Production Line.” *International Journal of Production Economics* 60-61: 353–358.

Ponnambalam, S. G., P. Aravindan, and G. Mogileeswar Naidu. 2000. “A Multi-objective Genetic Algorithm for Solving Assembly Line Balancing Problem.” *International Journal of Advanced Manufacturing Technology* 16 (5): 341–352.

Rahimi-Vahed, A. R., S. M. Mirghorbani, and M. Rabbani. 2007. “A Hybrid Multi-objective Particle Swarm Algorithm for a Mixed-Model Assembly Line Sequencing Problem.” *Engineering Optimization* 39 (8): 877–898.

Rashid, Mohd Fadzil Faisae, Windo Hutabarat, and Ashutosh Tiwari. 2012. “A Review on Assembly Sequence Planning and Assembly Line Balancing Optimisation Using Soft Computing Approaches.” *International Journal of Advanced Manufacturing Technology* 59 (1–4): 335–349.

Rubinovitz, Jacob, Joseph Bukchin, and Ehud Lenz. 1993. “RALB—A Heuristic Algorithm for Design and Balancing of Robotic Assembly Lines.” *CIRP Annals-Manufacturing Technology* 42 (1): 497–500.

Salveson, Melvin E. 1955. “The Assembly Line Balancing Problem.” *Journal of Industrial Engineering* 6 (3): 18–25.

Scholl, Armin. 1995. “Data of Assembly Line Balancing Problems.” In Darmstadt Technical University, Department of Business Administration, Economics and Law, Institute for Business Studies (BWL).

Scholl, Armin, and Christian Becker. 2006. “State-of-the-Art Exact and Heuristic Solution Procedures for Simple Assembly Line Balancing.” *European Journal of Operational Research* 168 (3): 666–693.

Scholl, Armin, and Robert Klein. 1999. “ULINO: Optimally Balancing U-shaped JIT Assembly Lines.” *International Journal of Production Research* 37 (4): 721–736.

Shyh Chyan, Goh, and S. G. Ponnambalam. 2012. “Obstacle Avoidance Control of Redundant Robots Using Variants of Particle Swarm Optimization.” *Robotics and Computer-Integrated Manufacturing* 28 (2): 147–153.

Sirovetnukul, R., and P. Chutima. 2010. “Multi-objective Particle Swarm Optimization with Negative Knowledge for U-shaped Assembly Line Worker Allocation Problems.” Proceedings of IEEE International Conference on Industrial Engineering and Engineering Management (IEEM), pp. 2033–2038.

Sivasankaran, P., and P. Shahabudeen. 2014. “Literature Review of Assembly Line Balancing Problems.” *The International Journal of Advanced Manufacturing Technology* 73 (9–12): 1665–1694. doi:10.1007/s00170-014-5944-y.

Toklu, Bilal, and Uğur özcan. 2008. “A Fuzzy Goal Programming Model for the Simple U-line Balancing Problem with Multiple Objectives.” *Engineering Optimization* 40 (3): 191–204.

Toksarı, M. Duran, Selçuk K. İşleyen, Ertan Güner, and Ömer Faruk Baykoç. 2008. “Simple and U-type Assembly Line Balancing Problems with a Learning Effect.” *Applied Mathematical Modelling* 32 (12): 2954–2961.

Törenli, Artun. 2009. “Assembly Line Design and Optimization.” Ph.D. Dissertation, Chalmers University of Technology, Sweden.

Urban, Timothy L. 1998. “Note. Optimal Balancing of U-shaped Assembly Lines.” *Management Science* 44 (5): 738–741.

Yalaoui, Alice, Hicham Chehade, Farouk Yalaoui, and Lionel Amodeo. 2013. *Optimization of Logistics*. USA: John Wiley & Sons.

Yoosefelahi, A., M. Aminnayeri, H. Mosadegh, and H. Davari Ardakani. 2012. “Type-II Robotic Assembly Line Balancing Problem: An Evolution Strategies Algorithm for a Multi-objective Model.” *Journal of Manufacturing Systems* 31 (2): 139–151.

Zha, Jing, and Jian-jun Yu. 2014. “A Hybrid Ant Colony Algorithm for U-line Balancing and Rebalancing in Just-in-Time Production Environment.” *Journal of Manufacturing Systems* 33 (1): 93–102.

Zhang, Zeqiang, and Wenming Cheng. 2010. “An Exact Method for U-shaped Assembly Line Balancing Problem.” Proceedings of 2nd International Workshop on Intelligent Systems and Applications (ISA) pp.1–4.