Neural Comput & Applic (2015) 26:1379–1393
DOI 10.1007/s00521-014-1811-x

ORIGINAL ARTICLE

Bio-inspired search algorithms to solve robotic assembly line balancing problems

J. Mukund Nilakantan · S. G. Ponnambalam · N. Jawahar · G. Kanagaraj

Received: 30 June 2014 / Accepted: 19 December 2014 / Published online: 4 January 2015
© The Natural Computing Applications Forum 2014

Abstract Robots are employed in assembly lines to increase the productivity. The objective of robotic assembly line balancing (rALB) problem is to balance the assembly line, by allocating equal amount of tasks to the workstations on the line while assigning the most efficient robot to perform the assembly task at the workstation. In this paper, bio-inspired search algorithms, viz. particle swarm optimization (PSO) algorithm and a hybrid cuckoo search and particle swarm optimization (CS-PSO), are proposed to balance the robotic assembly line with the objective of minimizing the cycle time. The performance of the proposed PSO and hybrid CS-PSO is evaluated using the 32 benchmark problems available in the literature. The simulation results show that both PSO and hybrid CS-PSO are capable of providing solutions within the upper bound obtained by hybrid GA, the only metaheuristic reported so far for rALB in the literature and comparable to the solutions obtained by IBM CPLEX Optimization solver. It is also observed that hybrid CS-PSO is performing better than PSO in terms of cycle time.

J. Mukund Nilakantan (✉) · S. G. Ponnambalam
Advanced Engineering Platform and School of Engineering, Monash University Malaysia, 46150 Bandar Sunway, Malaysia
e-mail: mukund.janardhanan@monash.edu

S. G. Ponnambalam
e-mail: sgponnambalam@monash.edu

N. Jawahar · G. Kanagaraj
Department of Mechanical Engineering, Thiagarajar College of Engineering, Madurai, India
e-mail: jawahartce@yahoo.co.uk

G. Kanagaraj
e-mail: gkmech@tce.edu

Keywords Robotic assembly line balancing · Particle swarm optimization · Cuckoo search · Local search

1 Introduction

Assembly operation is of high economic importance in a manufacturing sector. There have been extensive efforts to improve the efficiency and cost effectiveness of assembly operations. Assembly production lines can be manually operated, automated or of mixed design. In an assembly line, the tasks are assigned to the workstations in such a way that the precedence constraints are satisfied. Tedious and repetitive tasks are wearisome and exhaustive for workers. In the recent years, robots have been replacing the human labor in an assembly line and it improves the productivity by increasing the assembly speed and quality of the assembly. Robots have been widely used in assembly systems, and these are called robotic assembly lines. It also helps to produce faster than traditional methods and reduce the cycle time. The use of robots creates a leaner and more efficient manufacturing cycle. Most important advantages of using robots are as follows: increases productivity, quality of the product increases, less need of skilled labors and safety. Most important problem in this context is how assembly lines are managed and how the assembly line is well balanced.

Assembly line balancing (ALB) problem is first mathematically formulated by Salveson [1]. Various optimization models were introduced on ALB problems, and these are widely reported in the literature. Most of the research in ALB has been devoted to modeling and solving the simple assembly line balancing (SALB) problem. SALB problems can be classified by its objective function, and their different problem versions are SALB-1, SALB-2, SALB-F

and SALB-E [2–4]. The objective of the SALB-1 problem is to minimize the number of workstations for a given cycle time, whereas SALB-2 problem minimizes the cycle time given a predetermined number of workstations [5]. Unlike the previous two versions, SALB-F determines whether or not a feasible assembly configuration exists for a given combination of cycle time and number of workstations. SALB-E attempts to maximize the line efficiency by minimizing the number of workstations and cycle time simultaneously.

Robotic assembly line balancing (rALB) problem is an extension of SALB problems by replacing human labor. rALB mainly aims at assigning tasks to workstation and allocates the best available robot for each station to improve the productivity. Different types of robots are available with different capabilities and efficiencies to execute assembly tasks. In case of manual assembly lines, optimal balance is rather of theoretical importance because actual processing times for performing the assembly tasks vary considerably, whereas in case of the performance of robotic assembly lines, it strictly depends on the quality of their balance and robot assignment [6].

Robotic assembly line needs to be designed and balanced very well to function efficiently because investment is very high and it is a long-term decision. Robotic assembly line balancing problems are divided into two types: type-I rALB and type-II rALB. Type-I rALB deals at minimizing the number of workstations of assembly line for a fixed cycle time, whereas type-II rALB deals at minimizing the cycle time for a fixed number of workstations [7].

This paper aims at proposing two bio-inspired algorithms for solving a type-II rALB problem with an objective of minimizing the cycle when the number of workstations is fixed. The remainder of this paper is structured as follows. Section 2 provides a detailed literature survey, and Sect. 3 provides the assumptions and the mathematical model of the problem. Section 4 explains the proposed PSO and hybrid CS-PSO algorithm in detail. Section 5 details the computational results and statistical analysis conducted. Finally, Sect. 6 concludes the findings of this paper.

2 Literature survey

This section gives a detailed summary of the most related literature available for rALB problems and different heuristics, metaheuristics used to solve this problem. The literature related to type-I rALB is discussed below.

Graves and Lamar [8] addressed the problem of choosing the workstations from a set of nonidentical candidates and simultaneously assigning assembly tasks to the chosen workstations. The major goal of their work is to satisfy the maximum workload of the system and to minimize its total cost. Graves and Redfield [9] considered an assembly line where multi-products are assembled in a line. They proposed an algorithm to minimize the total cost. They adopted a method presented in [10]. The algorithm determines all feasible stations and selects the best equipment for each work station. Pinto et al. [11] worked on designing an assembly line with identical and parallel machines. Later, Pinto et al. [12] dealt with a SALB with different processing times and cost associated with it. The method developed includes the option of different manufacturing alternatives and assignment of tasks to stations in such a way that it reduces the total cost in which the cycle time is predetermined. Nicosia et al. [13] considered the problem of allocating tasks in an ordered sequence to nonidentical stations satisfying the precedence relationship for the given cycle time. The main objective of their work is to minimize the cost of the workstations. The formulation of this work is similar to the rALB problem. They developed a dynamic programming algorithm was developed for the problem, where several fathoming rules are used to reduce the number of states.

Rubinovitz et al. [14] formulated the rALB problem by allocating equal amounts of tasks to workstations on the assembly line while assigning the most efficient robot from a given set of available robots for each workstation. Their objective is to minimize the number of workstations for a given cycle time. They presented heuristics to limit and guide a branch-and-bound frontier search to reach solutions for very large and complex problems.

Bukchin and Tzur [15] developed a branch-and-bound algorithm which aimed at minimizing the equipment cost. They solved small- or medium-size problems and also proposed few heuristics to solve large-size problems. Tsai and Yao [16] proposed an integer programming model combined with a simulation adjustment phase. The method developed is used to design a flexible robotic assembly line, for producing a set of family products with data related to work to be done at each workstation, and the demand of each product and budget constraint are provided. The method determines the robot type and number of robots required for each workstation along a serial robotic line. Their objective was to minimize the standard deviation of the output rates of all workstations, which is the measure for a balanced line. Kim and Park [17] developed a mathematical formulation and a cutting plane algorithm for assigning of assembly tasks on a serial robotic assembly line with the aim of minimizing the total number of robotic cells.

The literatures related to type-II rALB are discussed below where the objective is to minimize the number of workstations or the cost of assembly systems given the cycle time.

Levitin et al. [6] developed a method for the rALB problem. Their method mainly aims at achieving a balanced distribution of activities among the workstations and assigning the best robot to the stations to perform the activities. They proposed two heuristic methods to assign tasks and robots called recursive and consecutive procedure. They proposed a genetic algorithm (GA) to solve the problem. A local exchange procedure is also used to further improve the quality of solutions. Best possible combinations of GA parameters are reported by conducting sensitivity analysis on randomly generated datasets.

Gao et al. [7] presented a 0–1 integer programming problem for rALB and proposed a hybrid genetic algorithm (hGA) to find efficient solutions for type-II rALB problem. The proposed GA uses partial representation technique, which expresses only part of the decision information about a candidate solution in the chromosome. The coding space contains only partial candidate solutions including the optimal one. They presented new crossover and mutation operators to adapt to the nature of the problem. In order to strengthen the search ability, local search procedures are also implemented by them.

Yoosefelahi et al. [18] presented a multi-objective model for rALB to minimize the cycle time, robot setup costs and robot costs. They developed a new mixed-integer linear programming model. As the rALB problem is NP-hard, they presented three versions of multi-objective evolution strategies (MOES) are employed. They concluded that the proposed hybrid MOES is efficient based on the simulation results obtained.

Daoud et al. [19] proposed several metaheuristic algorithms and a discrete event simulation model to solve rALB problem with an objective of maximizing line efficiency, and they considered an automated packaging line dedicated for dairy food products as a case study for evaluating the proposed model.

Since ALB problems are classified as NP-hard [20], solving it optimally by total enumeration is not practical with real-world or large-sized problems. Researchers shift their focus toward metaheuristic approaches as a popular way to address hard problems. Metaheuristics are efficient as they are fast and simple to implement. Most metaheuristics are generally problem specific, and their applications are limited. The focus of research thus shifts toward the development of powerful metaheuristic algorithms. Various optimization problems are solved by using metaheuristics, which provide a general algorithmic framework [21]. Metaheuristic algorithms are developed based on the learning from nature system. Metaheuristic algorithms are often bio-inspired, and they are widely used algorithms for optimization problems [22]. Different types of bio-inspired algorithms are GAs, simulated annealing, differential evolution, ant and bee algorithms, bat algorithm, particle swarm optimization (PSO), harmony search, firefly algorithm, cuckoo search and others.

Many bio-inspired algorithms have been proposed to solve different types of ALB problems [23]. Detailed literature survey on different types of metaheuristic algorithms used to solve ALB problems can be found in [4, 24].

From the literature review, it could be postulated that hybrid GA is the only metaheuristic proposed to solve type-II rALB problem. The two bio-inspired algorithms (PSO and hybrid CS-PSO) used in this research are developed based on the social and breeding behavior of birds [25]. Since these two bio-inspired algorithms have not been applied to solve the rALB problem, this work mainly aims at using these two bio-inspired search algorithms to solve rALB problem, with an objective of minimizing the cycle time for a fixed number of workstations.

3 Type-II rALB

Manufacturers started using robotic assembly lines to produce high-volume product and to maintain the quality of the product. In an assembly line, sequence of workstations is connected together with a material handling system. It helps to assemble components into a final product. In an assembly line setup, each work station is required to perform certain amount of tasks to produce certain products. The precedence constraints need to be specified, and it specifies the order in which the tasks are to be executed. The system needs to be configured for production of the product by assigning tasks to a particular station and assigning the best available robot to the work station in order to minimize the cycle time of the assembly line.

3.1 Assumptions and notations

The following assumptions are considered in this paper, which are adopted from [6, 7].

1. Assembly line is designed for a unique model of a single product.
2. Assembly tasks cannot be subdivided, and the tasks cannot be processed in random sequence due to precedence requirements.
3. Tasks cannot be divided among two or more stations.
4. Time taken for performing an activity depends on the type of the robot allocated.
5. A robot type which could perform the tasks assigned to a station in the least time among other type of robots is considered to assign it to a station.
6. Only one robot can be assigned to a station. Any task can be performed by any station and any robot. However, the precedence relation has to be

maintained, and task is to be carried out by a robot at a station where it is assigned.

7. All robots are available without any limitations (i.e., number of robots of same capability is unrestricted). Cost of the robot is not considered.
8. In a single-model assembly line, the material handling, loading and unloading time, as well as setup and tool changing time are negligible, because the tooling changes are minimized. This assumption is realistic on a single-model assembly line. If tool change or other type of setup activity is necessary, it can be included in the task time.

The following notations are used in this paper:

- Indices
  i, j: Index of assembly tasks, $i, j = 1, 2, \ldots, N_a$
  h: Index of robot types, $h = 1, 2, \ldots, N_r$
  s: Index of workstation, $s = 1, 2, \ldots, N_w$
- Parameters
  N_w: total number of workstations (robots)
  N_a: total number of tasks
  C: cycle time
  t_{ih}: processing time of task i by robot type h
  T_s: total execution time for workstation s
  $pre(i)$: set of immediate predecessors of task i
- Decision variables

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

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

3.2 Mathematical formulation of rALB-II

Levitin et al. [6] used the fifth assumption listed in Sect. 3.1 in the illustration presented in their paper. The two main objectives mentioned in their paper are as follows: optimal balance of the assembly line and allocation of the best-fit robot to each workstation. Achieving these two objectives is possible with the fifth assumption. Since this consideration helps to reduce the cycle time of the assembly line and assign a best-fit robot to workstations, it is followed in this paper. The model presented below is the modification of the one presented by Gao et al. [7], considering the fifth assumption. The zero–one integer programming (IP) model for the problem considered in this study is presented below.

3.2.1 Zero–one integer formulation of rALB-II

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

$$s.t. \sum_{s=1}^{N_w} s.x_{is} - \sum_{s=1}^{N_w} s.x_{js} \le , \quad 0 \forall i \in pre(j); j \tag{2}$$

$$\sum_{s=1}^{N_w} x_{is} = 1 \quad \forall i \tag{3}$$

$$\sum_{s=1}^{N_w} y_{sh} = 1 \quad \forall s \tag{4}$$

$$x_{is} \in \{0, 1\} \quad \forall s, i \tag{5}$$

$$y_{sh} \in \{0, 1\} \quad \forall h, s \tag{6}$$

The objective 1 is to minimize the cycle time of the robotic assembly line. Equation 2 defines the precedence relationship between the tasks. It ensures that for a pair of tasks with precedence relation, the precedent cannot be assigned to a workstation after the one to which its successor is assigned. Equation 3 ensures that each task has to be assigned to one workstation, and Eq. 4 ensures that each workstation is equipped with one robot. It is notable that objective 1 is nonlinear; hence, it is hard for traditional exact optimization techniques to solve the problem.

4 Bio-inspired search algorithms

A variety of bio-inspired search algorithms have been proposed till date. Different manufacturing system problems such as ALB problems, flexible manufacturing system and batch scheduling problems which are hard optimization problems [26] are evaluated using bio-inspired algorithms. Two types of bio-inspired search algorithms are proposed in this research: PSO and hybrid CS-PSO. To the best of the authors' knowledge, PSO and the hybrid CS-PSO are not extensively used to solve the rALB problem except for a literature on PSO using a recursive method to allocate tasks and robots [27].

4.1 Particle swarm optimization

Particle swarm optimization is a computational method developed by Kennedy and Eberhart [28]. This method is motivated by simulation of social behavior. This algorithm has been widely used to solve optimization problems. Ease of implementation, robustness and very few parameters to be fine-tuned for achieving the results makes PSO more attractive [29].

PSO algorithm starts with a population of randomly generated initial solutions called particles (swarm). It is to be noted that the particle structure is a string and consists of tasks to be performed in rALB problem satisfying the precedence constraints. After the swarm is initialized, each

Fig. 1 Pseudocode of particle swarm optimization

```
for each particle
   Initialize particle
end
do
   for each particle
      Calculate fitness value
      If the fitness value is better than the best fitness value (Local Best) in history
         set current value as the new Local Best
end
   Choose the particle with the best fitness value of all the particles as the Global Best
   for each particle
      Calculate particle velocity according Equation 7
      Update particle position according Equation 8
   end
while maximum iterations criteria is not attained
```

Table 1 Data and parameters for the example

Local best, ${}^{e}P_i^t$: (1, 2, 6, 3, 4, 5, 7, 8, 10, 9, 11)	*Global best*, G^t: (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).
$c_1 = 1$, $c_2 = 1$, $c_3 = 2$	$U_1 = 0.8$, $U_2 = 0.3$

particle is assigned with random velocity and length of the velocity of each particle is generated randomly. Fitness value (cycle time) of the particles is evaluated. 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 their best encountered position and the best position encountered by any particle. Velocity of the particle is updated iteratively using Eq. 7. Velocity update equation:

$$vi^{t+1} = c_1 v_i^t + c_2 U_1({}^{e}P_i^t - P_i^t) + c_3 U_2(G^t - P_i^t) \tag{7}$$

Particle moves from their current position to the new position using Eq. 8. Each particle's position is updated in each generation by adding the velocity vector to the position vector. Position update equation:

$$P_i^{t+1} = P_i^t + v_i^{t+1} \tag{8}$$

where U_1 and U_2 are known as velocity coefficients (random numbers between 0 and 1), c_1, c_2 and c_3 are known as acceleration coefficients, v_i^t is the initial velocity, ${}^{e}P_i^t$ is the local best, G^t is the global best solution at generation 't' and P_i^t is the current particle position. Pseudocode of the PSO algorithm is presented in Fig. 1 [30]. An example is shown to explain the velocity and position update. Table 1 shows the data used for the explanation.

These particles represents the assembly tasks arranged satisfying the precedence constraints.

The velocity for the particle is updated using Eq. 7.

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

Position of the particle 'i' is updated using Eq. 8.

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

4.1.1 Fitness evaluation

To evaluate the fitness, a consecutive heuristic procedure proposed by Levitin et al. [6] is adopted in this paper. This heuristic allocates tasks and robots to workstations with an objective of minimizing the cycle time of the assembly line. The method uses the following input: tasks sequence, precedence constraints, total number of robots and workstations and processing time of the tasks by the robots. Initial cycle time of the assembly line, C_0 is to be considered to start the procedure. Procedure tries to allocate maximum number of tasks to be performed at each work station. Robots are checked for allotment to the workstations to perform the tasks allotted in. If all the tasks are not possible to be assigned within the assumed C_0, C_0 is increment by '1' and procedure is repeated until all the tasks are assigned to the workstations. Initial value of C_0 is the mean of the minimum performance time of robots for the tasks. Initial assembly line time is calculated as follows.

$$C_0 = \left[\sum_{j=1}^{N_a} \min_{1 \le i \le N_r} t_{ih} / N_w \right] \tag{9}$$

Table 2 Performance time for 11 tasks by 4 robots

Task no	Robot 1	Robot 2	Robot 3	Robot 4	Avg. time
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

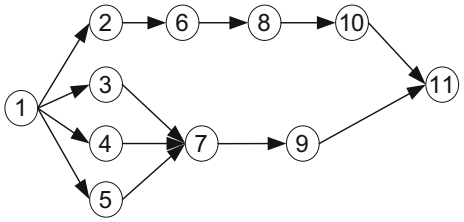

Fig. 2 Precedence graph of 11 task problem

An illustration of the calculation of cycle time for an 11 task, 4 robots problem is presented in this section. The robot performance time details are presented in Table 2. The precedence graph is shown in Fig. 2. Let the particle that meets precedence constraints be *1-4-5-3-7-9-2-6-8-10-11*. Using the robot performance times shown in Table 2, C_0 is calculated using Eq. 9 and it is found to be 109.

$$C_0 = [37 + 42 + 38 + 40 + 25 + 65 + 40 + 34 + 33 + 41 + 38]/4 = 108.25.$$

Here, 37, 42, 38, 40, 25, 65, 50, 34, 33, 41 and 38 are the minimum robot task times (refer Table 2). Procedure tries to find the optimal assignment within this C_0, and it is not possible to find a solution for four stations within this cycle time as shown in Fig. 3. From Fig. 3, it could be understood that tasks 6, 8, 10 and 11 are left to be unassigned within this C_0.

Since the tasks could not be assigned to the four stations within the assumed C_0, C_0 is to be incremented by 1 and the procedure is repeated. As a result, C_0 is incremented by 1 and the procedure could allot all the tasks to the workstations and assign the robots when C_0 value reaches 157. In the solution illustration shown in Fig. 4, the rows Robot 1–Robot 4 correspond to the four available robots. The row highlighted in gray indicates the robot which can perform the maximum tasks at a workstation while minimizing the

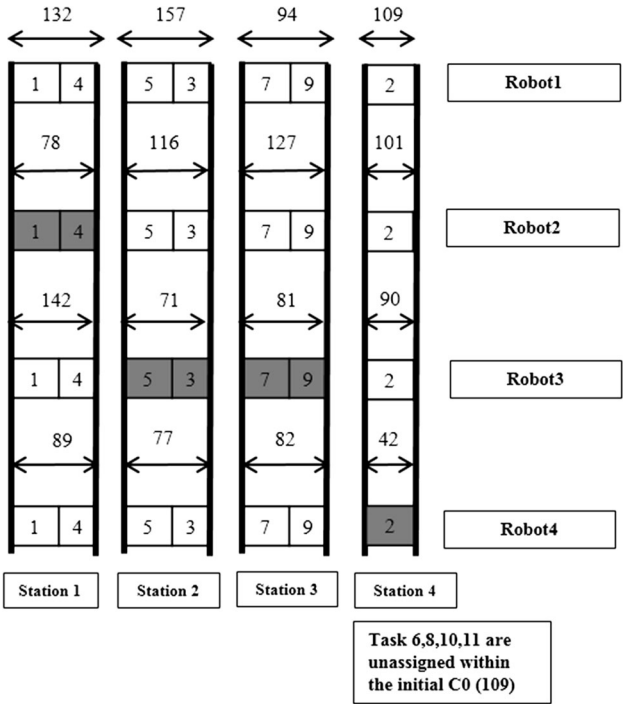

Fig. 3 Example of the consecutive assignment procedure when $C_0 = 109$

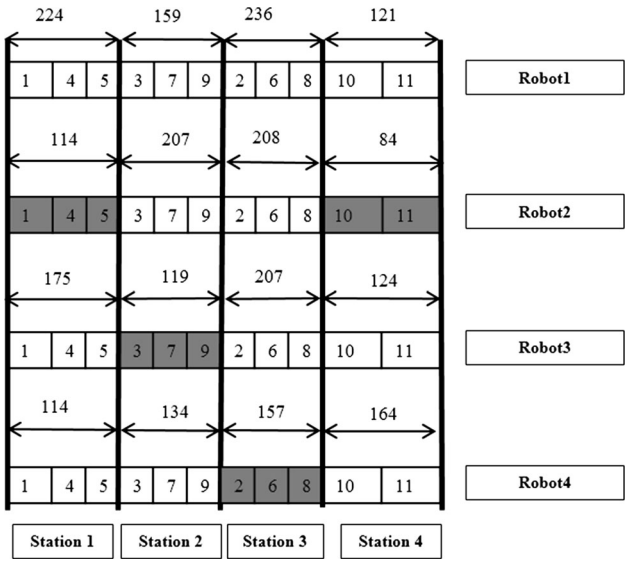

Fig. 4 Final task and robot allocation using consecutive procedure

total execution time of the station. Values on top of each workstation show the time for performing the assigned tasks by corresponding robot.

4.1.2 Initial population generation

Metaheuristic algorithms generally start with a randomly generated search space [31] which evolves iteratively to

find nearer to optimal solutions. Instead of starting the search process from a set of random solutions, a set of priority rules reported in the literature are used to reach better solution at a faster rate. Six particles are generated using the six heuristic rules [32]: maximum rank positional weight, minimum inverse positional weight, minimum total number of predecessors tasks, maximum total number of follower tasks, maximum and minimum task time. Remaining swarm particles are randomly generated. There are 25 particles in the initial swam. Example of the sequences generated using these heuristics is presented in [27].

4.1.3 Initial velocity

Initial velocities for the particles are randomly generated, and they represent the number of pairs of transpositions. Table 3 shows the maximum number of velocity pairs used in this paper. The velocity update Eq. 7 is used from the second iteration onwards. The number of velocity pairs is selected for different problems depending on the tasks size. The same size of velocity pair is used in all generations.

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

For example, if the task size within the range of 0–20, the number of velocity pair to be selected should be 4. If the number of velocity pairs is more than the required number of pairs in subsequent generations, the excess pairs are excluded.

4.1.4 Parameters used for PSO

Performance of PSO mainly depends on the parameters used. Extensive experiments are done to find the best optimal parameters. Initially, three datasets are chosen to find the parameters, which produce good solution. The three problems selected are 35 tasks-12 robots, 70 tasks-19 robots and 148 tasks-29 robots problems. These problems are solved for all combination of the parameters for ten test runs. Quality of solution is given importance compared to the computational time. The details of the analysis conducted and the parameters chosen for the proposed PSO is explained in this section.

Stopping condition: The proposed method is terminated if the iteration approaches a predefined criteria, usually a sufficiently good fitness, or in this case, a predefined maximum number of iterations (generations) are used. Different stopping conditions are tested such as 5, 10, 15, 25 and 30, and best solution is obtained when number of generation is 25, which is shown in Fig. 5. It could be observed that after 25th iteration the procedure produces the same solution for most of the runs.

Acceleration coefficients: Different combination of acceleration coefficients are tested, which are shown in Table 4. Figure 6 shows the performance of the algorithm based on these parameters for the three problems. It is observed that Group D has the best combination of acceleration coefficients to get the minimum cycle time for all the three problems considered.

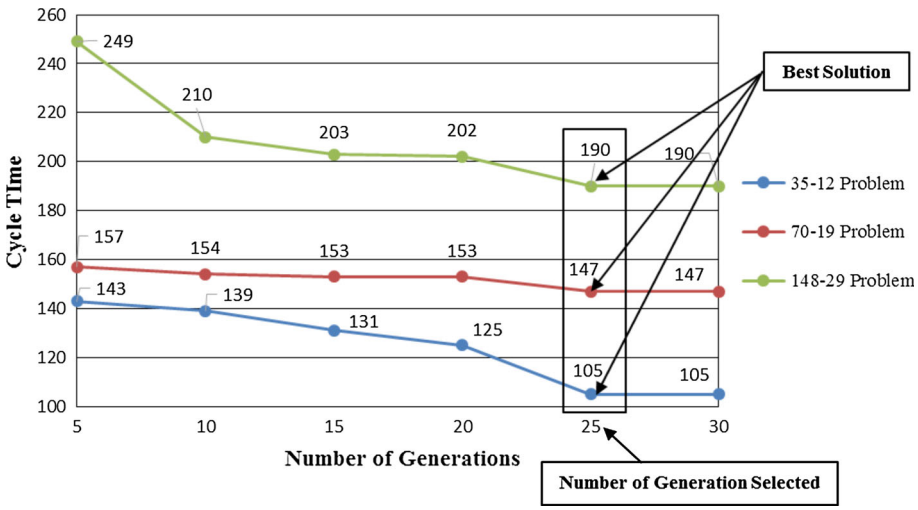

Fig. 5 Comparison of PSOs' performance in terms of stopping condition

Population size: Different population size of the swarm is tested such as 5, 10, 15, 25, and best solution is obtained when the population size is 25. Figure 7 shows the performance of PSO for different population size for three problems.

Table 4 Selection of c_1, c_2 and c_3

Acceleration coefficients			
Group	c_1	c_2	c_3
A	1	1	1
B	1	1	2
C	1	2	1
D	1	2	2
E	2	1	1
F	2	1	2
G	2	2	1
H	2	2	2

4.2 Hybrid CS-PSO algorithm

Yang and Deb [33] proposed a new metaheuristic algorithm which they called cuckoo search (CS) because they tried to emulate breeding behavior of cuckoo. CS algorithm has shown good performance for solving both benchmark unconstrained functions and real-world problems in manufacturing scheduling [34, 35]. The CS is based on three idealized rules [33, 34].

- Each cuckoo lays one egg at a time and dumps it in a randomly chosen host nest;
- The best nests with high quality of eggs (solutions) will carry over to the next generations;
- The number of available host nests is fixed, and a host can discover an alien egg with a probability p_a. In this case, the host bird can either throw the egg away or abandon the nest to build a completely new nest in a new location.

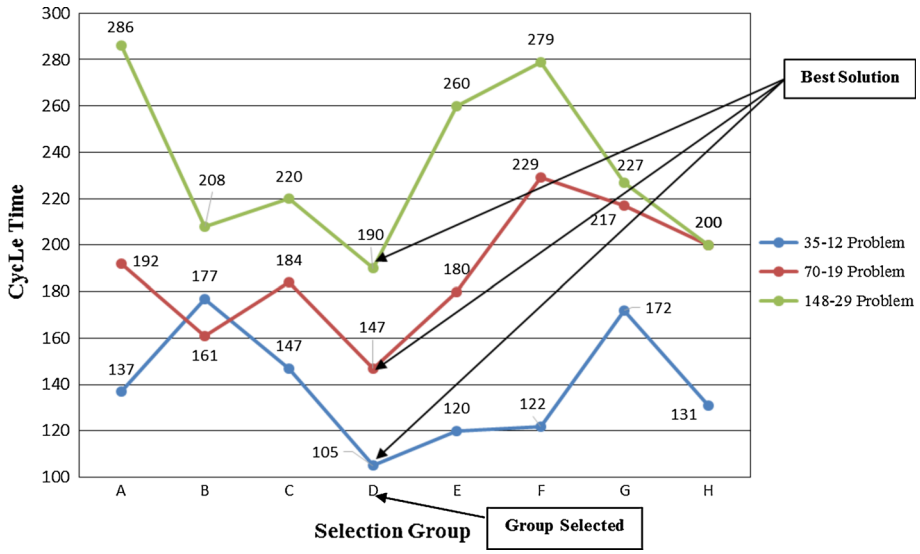

Fig. 6 Selection of acceleration coefficients based on the performance of PSO algorithm

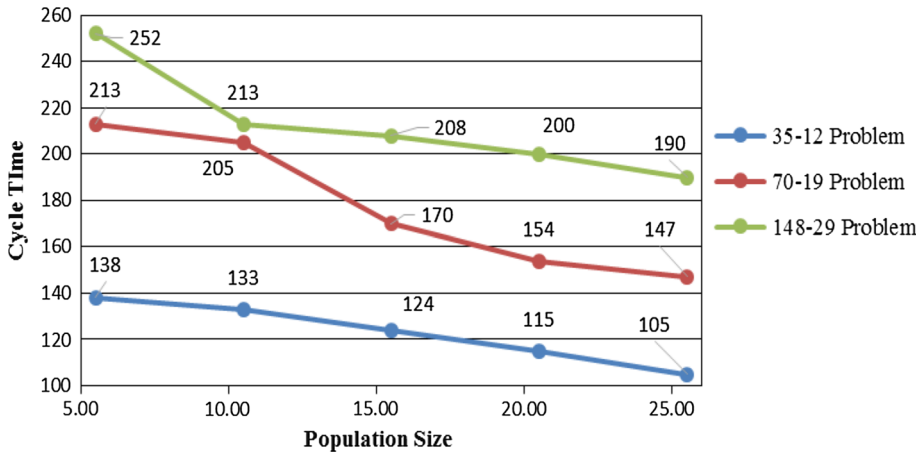

Fig. 7 Performance of PSO algorithm based on population size

In this section, proposed hybrid CS-PSO algorithm is explained. The nature of cuckoo birds is that they do not raise their own eggs and never build their own nests; instead, they lay their eggs in the nest of other host birds. If the alien egg is discovered by the host bird, it will either throw these alien eggs away or simply abandon its nest and build a new nest elsewhere. Thus, cuckoo birds always look for a better place in order to decrease the chance of their eggs to be discovered [34]. In the proposed algorithm, the ability of communication for cuckoo birds has been incorporated by hybridizing with PSO. The goal of this communication is to inform each other about their position and help each other to migrate to a better place. Each cuckoo bird will record the best personal experience as local best ($^{e}p_i^t$) during its own life. In addition to this, the local among all the birds called global best (G^t) is also recorded. The cuckoo birds' communication is established through the local, global best and they update their position and velocity using these parameters. Using Eqs. 10 and 11, the velocity and the position updates are carried out.Velocity update equation:

$$vi^{t+1} = c_1 v_i^t + c_2 U_1 (^{e}P_i^t - P_i^t) + c_3 U_2 (G^t - P_i^t) \quad (10)$$

Position update equation:

$$P_i^{t+1} = P_i^t + v_i^{t+1} \quad (11)$$

where U_1 and U_2 are known as velocity coefficients in the range of 0 and 1, c_1, c_2 and c_3 are known as acceleration coefficients, v_i^t is the initial velocity, $^{e}P_i^t$ is the local best, G is the global best and P_i^t is the current position of the cuckoo. The pseudocode of the hybrid CS-PSO is given in the Fig. 8.

```
begin
    Generate  initial a population of n host nests, x_i(i=1,2,...n);
    Evaluate objective function x=(x_1,...x_d)^T;

    while(t<MaxGeneration) or (stop criterion);
        Randomly select two cuckoos in the population;
        Generate a new cuckoo using OX operator;
        Evaluate the new cuckoo and record the local best (F_i);
    Choose a nest among (say j) randomly;

    if(Fi < F j)
    Replace j by the new solution;
    end

    Move cuckoo birds using Equation 10 and 11;
    Abandon a fraction (pa) of worse nests;
    Build new ones at new locations using singlepoint crossover;
    Rank the solutions and find the current best;
  end while
    Record the global best
end
```

Fig. 8 Pseudocode of hybrid CS-PSO algorithm

4.2.1 New cuckoo generation

New cuckoos are generated by using OX operator, which is proposed by Davis [36]. Sequence (cuckoo) in case of rALB problem is the group of tasks arranged in such a way that it satisfies the precedence conditions.

OX Operator works as follows:

- Select a subsection of task sequence from one parent at random.
- Produce a proto-child by copying the substring of task sequence into the corresponding positions.
- Delete the tasks that are already in the substring from the second parent. The resulted sequence of tasks contains tasks that the proto-child needs.
- Place the tasks into the unfixed positions of the proto-child from left to right according to the order of the sequence in the second parent.

An illustration is presented in Fig. 9. Select two cuckoos from the population and apply OX operator to generate a new cuckoo. The new child cuckoo generated is shown in the figure.

4.2.2 Replacement of abandoned cuckoos

New cuckoos are generated from the abandoned cuckoos using single-point crossover is a crossover method where a single crossover point on both parents' organism strings is selected. All data beyond that point is swapped between the two parent organisms. The resulting sequences are the children. An illustration is presented in Fig. 10.

4.2.3 Repair mechanism

If the child cuckoos created using the crossover methods are not meeting the precedence constraints, the cuckoo is repaired using a reordering procedure, which restores the feasibility of the generated sequence according to the precedence constraints. Detailed description of the reordering procedure can be found in [37].

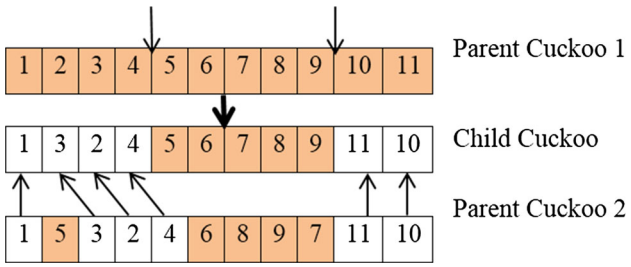

Fig. 9 Illustration of the OX operator

4.2.4 Parameters of hybrid CS-PSO algorithm

The parameter setting of hybrid CS-PSO algorithm is described in this section. Fraction of worst nests chosen (p_a) is fixed as a constant for all generations in the traditional CS algorithm. In this hybrid algorithm, p_a is dynamically updated [38] using Eq. 12

$$p_a(gn) = p_{a\max} - \frac{gn}{N}(p_{a\max} - p_{a\min}) \quad (12)$$

where N and gn are the number of total iterations and the current iteration, respectively. It is found that $p_{a\max} = 0.8$ and $p_{a\min} = 0.1$ generates better results. The population size and acceleration coefficients are the same for PSO and CS-PSO algorithms as explained in Sect. 4.1.4. The proposed method is terminated if the iteration approaches a predefined criteria, usually a sufficiently good fitness, or in this case, a predefined maximum number of iterations (generations) are used. Different stopping conditions are tested such as 5, 10, 15, 25, 30 and 35. Best solution is obtained when number of generation is 30. It could be observed that after 30th iteration, the procedure produces the same solution for most of the runs. Figure 11 shows the performance of algorithm based on the number of generations for three problems (35–12, 70–19, 148–29).

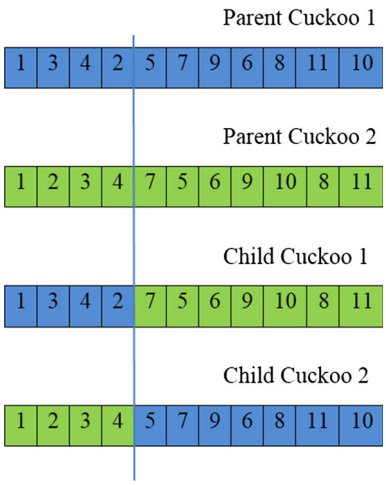

Fig. 10 Illustration of the single-point crossover

4.2.5 Local exchange procedure

To improve the quality and performance of the proposed algorithms, an exchange procedure is applied to the final solution obtained from the two proposed algorithms. Detailed description of this procedure is available in [27].

5 Computational and statistical analysis

The results of computational analysis and statistical analysis are presented in this section.

5.1 Computational results

The computational experiments are conducted in order to test the performance of the proposed PSO and hybrid CS-PSO on rALB problems. The details of the experiments conducted are presented in this section. Large set of benchmark problems are evaluated to check the performance of the proposed algorithms. Eight representative precedence graphs are selected from http://www.assembly-line-balancin.de/, which are widely used in SALB-I literature [39]. Problem size varies between 25 and 297 tasks. Gao et al. [7] generated 32 test problems for rALB problems by adding the robot task times to the original datasets prepared by Scholl [39]. The two proposed algorithms are

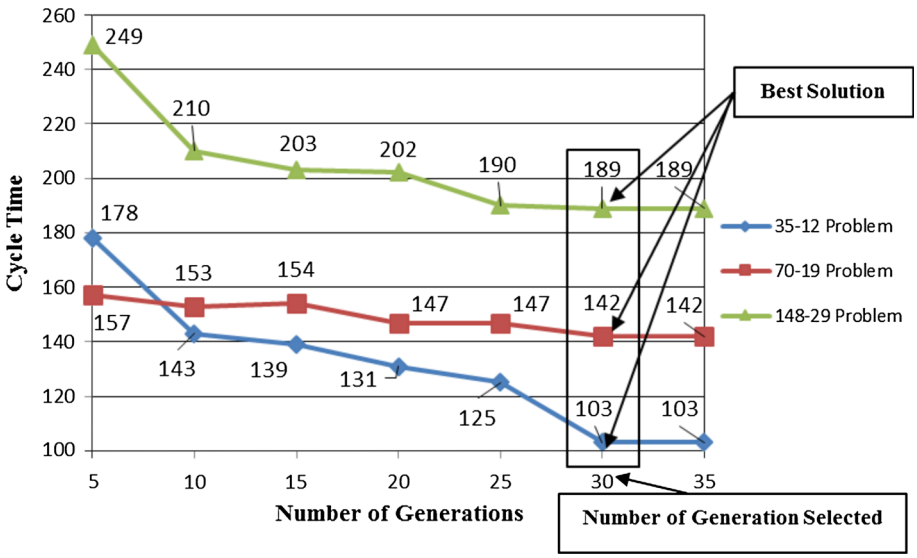

Fig. 11 Comparison of hybrid CS-PSOs' performance in terms of stopping condition

Table 5 Results of the 32 rALB-II problems

Problem number	No. of tasks	No. of work station/robots	Hybrid GA [7]			Cplex		Proposed algorithm (PSO)		Proposed algorithm (hybrid CS-PSO)	
			Cycle time	CPU time (sec)	Modified CPU time *m t (s)* (sec)	Cycle time	CPU time (sec)	Cycle time	CPU time (sec)	Cycle time	CPU time (sec)
1	25	3	503	5	2	503	0.5	503	2.65	503	3.6
2		4	327	5	2	292	8.85	327	2.9	327	3.9
3		6	213	6	2	200	463	208	3.0	200	4.2
4		9	123	7	3	109	4002	114	3.25	110	4.5
5	35	4	449	12	5	341	59	344	4.9	341	5.2
6		5	344	16	7	329	562	336	5.4	332	6.3
7		7	222	22	9	201	3672	214	6.9	211	6.9
8		12	113	30	12	106	3620	105	8.5	103	8.9
9	53	5	554	17	7	449	72	454	13.1	449	13.5
10		7	320	21	9	283	3680	301	14.9	294	16.8
11		10	230	27	11	221	3627	224	16.2	221	17.9
12		14	162	35	14	142	3729	146	19.9	142	20
13	70	7	449	40	16	394	4007	431	29.0	430	32.9
14		10	272	53	22	245	3700	269	32.5	264	35.8
15		14	204	64	26	N/A	N/A	200	39.1	194	43.3
16		19	154	82	33	N/A	N/A	147	43.4	140	47.8
17	89	8	494	46	19	N/A	N/A	463	41.9	460	45.7
18		12	370	58	24	N/A	N/A	355	50.4	320	51.6
19		16	236	71	29	N/A	N/A	234	59.6	219	63.3
20		21	205	89	36	N/A	N/A	176	75.3	170	80.5
21	111	9	557	157	64	N/A	N/A	526	82.3	523	85.5
22		13	319	192	78	N/A	N/A	316	89.5	321	92.5
23		17	257	229	93	N/A	N/A	254	98.5	240	107.4
24		22	192	271	110	N/A	N/A	185	110.8	182	114.5
25	148	10	600	240	98	N/A	N/A	603	179.8	593	183.5
26		14	427	297	121	N/A	N/A	420	205.5	419	207.9
27		21	300	332	135	N/A	N/A	277	215.9	273	219.5
28		29	202	417	169	N/A	N/A	190	230.3	189	242.2
29	297	19	646	824	335	N/A	N/A	608	891.8	594	1118.3
30		29	430	907	369	N/A	N/A	397	997.6	394	1331.3
31		38	344	996	405	N/A	N/A	295	1269.9	305	1593.5
32		50	256	1103	448	N/A	N/A	245	1390.8	221	1664.3

evaluated using the 32 benchmark problems. In each problem, the number of workstations is equal to the number of robots, one robot type to one workstation, each task is assigned to one workstation and the robot type selected for that workstation without violating precedence constraints, and the processing time of a task at the assigned station is dependent on the type of robot selected for that workstation. In this formulation, robot-type assignment differs from Gao et al.'s model. Gao et al. [7] has confined to only one robot is available in each robot type. Though their model differs in one aspect, the solution of hybrid GA [7] can be used to fix the upper bound for the problem under consideration

The nondeterministic nature of the algorithm and problem makes it necessary to run same problem multiple times. Each problem is run ten times, and most of the runs converged to the same solution for each of the problems. The results obtained by evaluating 32 test problems are presented in Table 5. Column I shows the problem number. Column II shows the task size of the problems evaluated, and Column III shows the number of workstations/robots for each problem. Column IV and V shows the cycle time

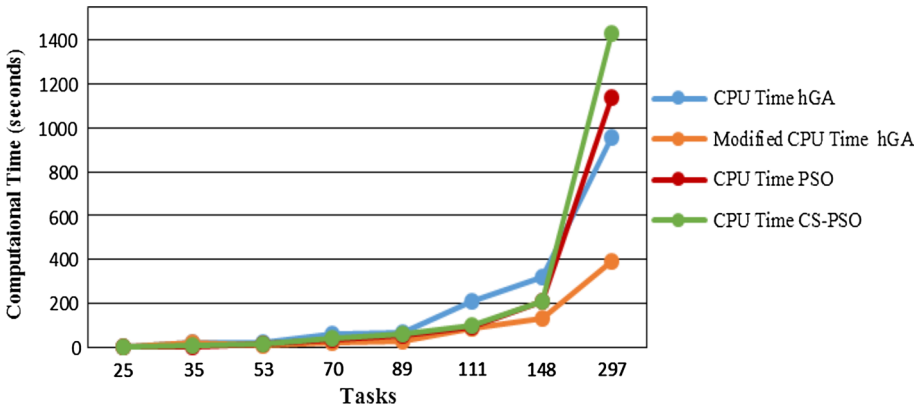

Fig. 12 Comparison of average computational time

and computational time for the 32 problems obtained using the hybrid GA. Column VI shows the modified cycle time for hGA algorithm. Column VII and VIII shows the results obtained using the Cplex solver (optimization solver). Column IX and X shows the cycle time and computational time obtained using the proposed PSO. Column XI and XII shows the cycle time and computational time obtained using the proposed hybrid CS-PSO.

IBM Cplex Optimization studio version 12.6.0.0 is used to solve the problems. It is observed that Cplex solver could not generate solutions for all the problems. Since the objective function of the formulation for this problem is nonlinear, hence, it is hard for traditional exact optimization techniques to solve the problems. Cplex could generate the solutions for the first 14 problems, and for the rest of the problem, Cplex displays an error message as, 'Search Space Exceeded.' Solutions obtained by the proposed algorithms are compared with results obtained using hybrid GA (upper bound for the problem) [7] and Cplex solver solutions. It is found that the results of PSO and CS-PSO are very close to Cplex solutions and it could be clearly seen that hybrid CS-PSO produces better results when compared to PSO and hGA.

The computational time (CPU Time) taken for hybrid CS-PSO algorithm is more compared to PSO for all the 32 datasets but lesser than hGA for 28 out of 32 problems. Since algorithms are tested on different computers, it is difficult to compare the computational times. Passmark Performance Test 8.0 is used for an approximate comparison of CPU execution times. The proposed algorithms are coded in C++ and tested on Intel Core i5 processor (2.3 GHz). The hGA algorithm is executed on a Pentium 4 processor (2.6 GHz). Using the Passmark Performance Test 8.0 software, the factor for the computer used to solve PSO and hybrid CS-PSO is fixed to 1 and for the computer used to solve hGA is found to be 0.406. Since there are too many factors affecting the CPU times, it is difficult to do a fair comparison. In Table 5, column VI with the heading *mt(s)* gives the modified CPU time of the average best solution computation time with the factor belonging to the computer is included. From the table, it could be seen that modified CPU time for hGA is lower than that of the proposed algorithms for most of the datasets. This could be due to large solution space and the local exchange procedure used. However, for the proposed algorithms, quality of the solution is given importance compared to the computational time in this paper. Figure 12 shows that the CPU time the proposed PSO and CS-PSO are comparable to hGA for the problems up to 111 task problems.

In order to demonstrate the performance of proposed methods, a real life ALB problem of a major automobile manufacturer, where the final assembly operation of an engine cradle is considered [40]. Problem with 35 task and 5 robots is chosen. The problem consists of 45 direct precedence relations among 35 tasks as shown in Table 6, and processing times of 35 tasks by 5 robots are also shown in Table 6. The solutions obtained for the problem using PSO and hybrid CS-PSO are given in Table 7 along with upper bound of hybrid GA [7]. It could be analyzed from the Table 7 that hybrid CS-PSO algorithm produces a better result (in terms of cycle time) when compared to other two algorithms. Cycle time obtained using hybrid CS-PSO is 332, whereas cycle time obtained using PSO is 336 and both are under the upper bound obtained using hybrid GA of 344.

5.2 Statistical analysis

The objective of this statistical analysis is to compare the performance of the proposed PSO and hybrid CS-PSO algorithms with hybrid GA (hGA) [7]. Statistical analysis is to be conducted for the joint analysis of the results achieved by various algorithms. A pairwise comparison test should not be used to conduct various comparisons involving a set of algorithms, because the familywise error rate is not controlled. The procedure followed is explained in this section.

Table 6 Performance times of 35 tasks by 5 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

Table 7 Solutions obtained for rALB-II problem for 35 tasks by 5 robots problem

Hybrid GA sequence: 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		
Station start points: 1 14 20 25 30	Robot assignment: 4 3 1 5 2	Cycle time: 344
PSO sequence: 1 10 12 17 2 3 4 5 6 7 18 19 14 15 16 20 21 25 26 22 23 8 9 13 24 27 34 11 30 31 32 33 28 29 35		
Station start points: 1 4 14 22 34	Robot assignment: 2 2 4 1 2	Cycle time: 336
Hybrid CS-PSO sequence: 110 17 5 6 8 2 3 7 12 18 14 15 16 19 20 21 25 26 22 23 4 9 24 27 30 31 32 13 34 11 33 28 29 35		
Station start points: 1 2 15 22 31	Robot assignment: 4 2 4 1 2	Cycle time: 332

Friedman's test is an omnibus test, which is used to carry out multiple comparisons between hGA, PSO and CS-PSO. This test is useful to detect differences between the set of algorithms considered for the analysis. If Friedman's test rejects the null hypothesis, a post hoc test is conducted in order to find the concrete pairwise comparisons between the three algorithms. Three p values for the three pairs of comparisons (hGA-PSO, hGA-CS-PSO

Table 8 Friedman test results

Mean rank for the sample		
hGA	PSO	CS-PSO
2.9	2	1.1

Table 9 Wilcoxon signed-rank test results

	Pairwise comparisons ($\alpha = 0.05$)		
	hGA-PSO	hGA-CSPSO	PSO-CSPSO
p values	0	0	8E − 05
Evaluations	Since $p < \alpha/3$, PSO is better than hGA	Since $p < \alpha/2$, CS-PSO is better than hGA	Since $p < \alpha/1$, CS-PSO is better than PSO

and PSO-CS-PSO) are calculated using Wilcoxon signed-rank test, and they sorted in the ascending order. Let them be p1, p2 and p3. Then, p1, p2 and p3 are compared with corrected significance levels, $\alpha/3$, $\alpha/2$ and $\alpha/1$, respectively, according to the Holm–Bonferroni method [41] of multiple comparisons. If they are smaller than the level of significance (alpha = 0.05), then we can conclude that there is a significant difference with 5 % significance level. The cycle time of the three algorithms are presented in Table 5. Friedman test results are shown in Table 8. Since the cycle time for the first two problems are the same for the three algorithms, Friedman test is conducted for the rest of the 30 results.

The calculated Chi-square value is 50.47 with the degree of freedom of 2 and p value is <0.0001. Since n is 30, which is sufficiently large, the Chi square is a close approximation of the sampling distribution of Chi square with degree of freedom of 2. Since the table value for 0.05 level of significance and the degree of freedom of 2 is 5.99, there is a significant differences between the algorithms considered for analysis. Then, the post hoc test is conducted to do the pairwise comparisons. The Wilcoxon signed-rank test provides the p values for the three pairwise comparisons. The p values obtained and the corrected p values after the sorting are shown in Table 9. Since there are three pairwise comparisons, set $n = 3$ and compare the p values obtained from Wilcoxon signed-rank test.

Based on the statistical analysis conducted, we can conclude that both PSO and CS-PSO performs better than hGA and hybrid CS-PSO is better than PSO.

6 Conclusion

Optimizing cycle time is an important problem in manufacturing systems. In this paper, a study on rALB problem is considered. Since this model addressed here is well known as NP-hard, two bio-inspired algorithms are developed to solve the problem with an objective of minimizing the cycle time. Consecutive method is used to optimally allocate tasks to the workstations and allocation of the best-fit robot for the workstation. This method also evaluates the cycle time of the balanced assembly line. Hybrid CS-PSO algorithm is also developed for solving the same. Local exchange procedure is used to improve the quality of the solution obtained through these methods and to escape from the local optima. Simulation experiments have been carried out over the 32 bench mark datasets available in the literature. IBM Cplex Optimization studio version 12.6.0.0 is used to solve the problems. Cplex could only solve the first 14 problems shown in Table 5. It is found that the results of PSO and CS-PSO are very close to Cplex solutions. Results obtained through the analysis shows that hybrid CS-PSO algorithm outperforms PSO and the other algorithm (hybrid GA) reported in the literature. The solution obtained from the algorithms provides the best robot assignment for the workstations by distributing the tasks among them. Computational time for hybrid CS-PSO algorithm is high for large datasets but quality of the solution is given importance compared to the computational time.

References

1. Salveson ME (1955) The assembly line balancing problem. J Ind Eng 6(3):18–25
2. Kilincci O, Bayhan GM (2006) A Petri net approach for simple assembly line balancing problems. Int J Adv Manuf Technol 30(11–12):1165–1173
3. Baybars I (1986) A survey of exact algorithms for the simple assembly line balancing problem. Manage Sci 32(8):909–932
4. Rashid MFF, Hutabarat W, Tiwari A (2012) A review on assembly sequence planning and assembly line balancing optimisation using soft computing approaches. Int J Adv Manuf Technol 59(1–4):335–349
5. Scholl A, Scholl A (1999) Balancing and sequencing of assembly lines. Physica-Verlag, Heidelberg
6. Levitin G, Rubinovitz J, Shnits B (2006) A genetic algorithm for robotic assembly line balancing. Eur J Oper Res 168(3):811–825
7. Gao J, Sun L, Wang L, Gen M (2009) An efficient approach for type II robotic assembly line balancing problems. Comput Ind Eng 56(3):1065–1080
8. Graves SC, Lamar BW (1983) An integer programming procedure for assembly system design problems. Oper Res 31(3):522–545
9. Graves SC, Redfield CH (1988) Equipment selection and task assignment for multiproduct assembly system design. Int J Flex Manuf Syst 1(1):31–50
10. Gutjahr AL, Nemhauser GL (1964) An algorithm for the line balancing problem. Manage Sci 11(2):308–315
11. Pinto PA, Dannenbring DG, Khumawala BM (1981) Branch and bound and heuristic procedures for assembly line balancing with paralleling of stations. Int J Prod Res 19(5):565–576

12. Pinto PA, Dannenbring DG, Khumawala BM (1983) Assembly line balancing with processing alternatives: an application. Manage Sci 29(7):817–830
13. Nicosia G, Pacciarelli D, Pacifici A (2002) Optimally balancing assembly lines with different workstations. Discrete Appl Math 118(1):99–113
14. Rubinovitz J, Bukchin J, Lenz E (1993) RALB–A heuristic algorithm for design and balancing of robotic assembly lines. CIRP Ann Manuf Technol 42(1):497–500
15. Bukchin J, Tzur M (2000) Design of flexible assembly line to minimize equipment cost. IIE Trans 32(7):585–598
16. Tsai D-M, Yao M-J (1993) A line-balance-based capacity planning procedure for series-type robotic assembly line. Int J Prod Res 31(8):1901–1920
17. Kim H, Park S (1995) A strong cutting plane algorithm for the robotic assembly line balancing problem. Int J Prod Res 33(8):2311–2323
18. Yoosefelahi A, Aminnayeri M, Mosadegh H, Ardakani HD (2012) Type II robotic assembly line balancing problem: an evolution strategies algorithm for a multi-objective model. J Manuf Syst 31(2):139–151
19. Daoud S, Chehade H, Yalaoui F, Amodeo L (2014) Solving a robotic assembly line balancing problem using efficient hybrid methods. J Heuristics 20(3):235–259
20. Scholl A, Becker C (2006) State-of-the-art exact and heuristic solution procedures for simple assembly line balancing. Eur J Oper Res 168(3):666–693
21. Sörensen K, Glover FW (2013) Metaheuristics. In: Gass S, Fu M (eds) Encyclopedia of operations research and management science. Springer, New York, pp 960–970
22. Yang X-S, Deb S (2014) Cuckoo search: recent advances and applications. Neural Comput Appl 24(1):169–174
23. Atasagun Y, Kara Y (2013) Bacterial foraging optimization algorithm for assembly line balancing. Neural Comput Appl 25(1):237–250
24. Sivasankaran P, Shahabudeen P (2014) Literature review of assembly line balancing problems. Int J Adv Manuf Technol 73(9–12):1665–1694
25. Ghodrati A, Lotfi S (2012) A hybrid CS/PSO algorithm for global optimization. In: Pan J-S, Chen S-M, Nguyen N (eds) Intelligent information and database systems. Springer, Heidelberg, pp 89–98
26. Noroozi A, Mokhtari H, Kamal Abadi IN (2013) Research on computational intelligence algorithms with adaptive learning approach for scheduling problems with batch processing machines. Neurocomputing 101:190–203
27. Mukund Nilakantan J, Ponnambalam S (2012) An efficient PSO for type II robotic assembly line balancing problem. In: Proceedings of IEEE international conference on automation science and engineering (CASE), pp 600–605
28. Kennedy J, Eberhart R (1995) Particle swarm optimization. In: Proceedings of IEEE international conference on neural networks, pp 1942–1948
29. Del Valle Y, Venayagamoorthy GK, Mohagheghi S, Hernandez J-C, Harley RG (2008) Particle swarm optimization: basic concepts, variants and applications in power systems. IEEE Trans Evol Comput 12(2):171–195
30. Hu X (2006) PSO tutorial http://www.swarmintelligence.org/tutorials.php
31. Huang K-W, Chen J-L, Yang C-S, Tsai C-W (2014) A memetic particle swarm optimization algorithm for solving the DNA fragment assembly problem. Neural Comput Appl 1–12. doi:10.1007/s00521-014-1659-0
32. Ponnambalam S, Aravindan P, Naidu GM (2000) A multi-objective genetic algorithm for solving assembly line balancing problem. Int J Adv Manuf Technol 16(5):341–352
33. Yang X-S, Deb S (2009) Cuckoo search via Lévy flights. Proc World Congr Nat Biol Inspired Comput NaBIC 2009:210–214
34. Burnwal S, Deb S (2013) Scheduling optimization of flexible manufacturing system using cuckoo search-based approach. Int J Adv Manuf Technol 64(5–8):951–959
35. Long W, Liang X, Huang Y, Chen Y (2014) An effective hybrid cuckoo search algorithm for constrained global optimization. Neural Comput Appl 25(3–4):911–926
36. Davis L (1985) Applying adaptive algorithms to epistatic domains. In: Proceedings of IJCAI, pp 162–164
37. Rubinovitz J, Levitin G (1995) Genetic algorithm for assembly line balancing. Int J Prod Econ 41(1):343–354
38. Valian E, Mohanna S, Tavakoli S (2011) Improved cuckoo search algorithm for global optimization. Int J Commun Inf Technol 1(1):31–44
39. Scholl A (1995) Data of assembly line balancing problems. Darmstadt Technical University, Department of Business Administration, Economics and Law, Institute for Business Studies (BWL)
40. Gunther RE, Johnson GD, Peterson RS (1983) Currently practiced formulations for the assembly line balance problem. J Oper Manag 3(4):209–221
41. Holm S (1979) A simple sequentially rejective multiple test procedure. Scand J Stat 6:65–70