Co-evolutionary particle swarm optimization algorithm for two-sided robotic assembly line balancing problem

Li, Zixiang; Janardhanan, Mukund Nilakantan; Tang, Qiuhua; Nielsen, Peter

Published in:
Advances in Mechanical Engineering

DOI (link to publication from Publisher):
10.1177/1687814016667907

Publication date:
2016

Document Version
Publisher's PDF, also known as Version of record

Special Issue Article

Advances in Mechanical Engineering

Advances in Mechanical Engineering
2016, Vol. 8(9) 1–14

DOI: 10.1177/1687814016667907
aime.sagepub.com
SAGE

Co-evolutionary particle swarm optimization algorithm for two-sided robotic assembly line balancing problem

Zixiang Li[1], Mukund Nilakantan Janardhanan[2], Qiuhua Tang[1] and Peter Nielsen[2]

Abstract

Industries utilize two-sided assembly lines for producing large-sized volume products such as cars and trucks. By employing robots, industries achieve a high level of automation in the assembly process. Robots help to replace human labor and execute tasks efficiently at each workstation in the assembly line. From the literature, it is concluded that not much work has been conducted on two two-sided robotic assembly line balancing problems. This article addresses the two-sided robotic assembly line balancing problem with the objective of minimizing the cycle time. A mixed-integer programming model of the proposed problem is developed which is solved by the CPLEX solver for small-sized problems. Due to the problems in non-polynomial- -hard nature, a co-evolutionary particle swarm optimization algorithm is developed to solve it. The co-evolutionary particle swarm optimization utilizes local search on the global best individual to enhance intensification, modification of global best to emphasize exploration, and restart mechanism to escape from local optima. The performances of the proposed co-evolutionary particle swarm optimization are evaluated on the modified seven well-known two-sided assembly line balancing problems available in the literature. The proposed algorithm is compared with five other well-known metaheuristics, and computational and statistical results demonstrate that the proposed co-evolutionary particle swarm optimization outperforms most of the other metaheuristics for majority of the problems considered in the study.

Keywords

Two-sided robotic assembly line, particle swarm optimization, assembly line balancing problems, cycle time, local search

Date received: 14 April 2016; accepted: 13 August 2016

Academic Editor: Bin Yu

Introduction

Manufacturing companies extensively use assembly lines and the assembly process is considered to be one of the critical processes in manufacturing systems.[1,2] Different layout types (traditional straight line, U-shaped, two-sided, and parallel) have been widely utilized in industries based on the size and type of products.[3] For assembly of large-sized volume products, for example, cars, trucks, and buses utilize two-sided assembly lines. When compared with traditional

[1]Department of Industrial Engineering, Wuhan University of Science and Technology, Wuhan, China
[2]Department of Mechanical and Manufacturing Engineering, Aalborg University, Aalborg, Denmark

Corresponding author:
Mukund Nilakantan Janardhanan, Department of Mechanical and Manufacturing Engineering, Aalborg University, Aalborg 9220, Denmark.
Email: mnj@m-tech.aau.dk

straight assembly lines, two-sided assembly line provides several advantages such as shorter assembly line length, reduced throughput time, less material handling, and lower cost of tools and fixtures.[4] In a two-sided assembly line, different assembly tasks are performed on the same product item in parallel at both (left and right) sides of the line.[5]

Assembly lines can be of manually operated, automated, or of mixed design. Due to the tedious and repetitive nature of tasks performed in assembly lines, robots have replaced human labor and this helps in improving both the speed of assembly and quality of the products assembled.[6] Robots can be programmed to employ them for performing different types of tasks in assembly systems and these are called robotic assembly lines. Balancing assembly lines have, due to their prevalence in industry, become a critical process that aims at allocating work (task) to the workstations in such a manner that all workstations have an equal amount of task assigned to them. Most existing research is devoted to solving simple assembly line balancing (SALB) problems with different objective functions.[7,8] In the case of robotic assembly lines, an efficient balanced assembly line is very necessary since the investments in such assembly line are high and are typically based on long-term strategic decision.[9] Robotic assembly line balancing (RALB) problems are an extension of SALB problems.[6] RALB aims mainly at assigning tasks to the workstations and allocating the best fit robot to each workstation in such a way the productivity is improved. The two main types of RALB problems addressed by researchers are as follows: type-I RALB and type-II RALB.[6] Type-I RALB mainly aims at minimizing the number of workstations in an assembly line when cycle time is fixed and type-II RALB mainly aims at minimizing the cycle time when the number of workstations is fixed.[2]

Researchers have classified assembly line balancing (ALB) problem in the category of non-polynomial (NP) hard.[10] Due to this nature of the problem, researchers have over the years proposed different techniques to solve ALB problems. A detailed literature overview of different techniques used to solve two-sided ALB and RALB is given in the following. The two-sided ALB problem is initially proposed by Bartholdi.[4] Since then different techniques such as exact methods, heuristic, and metaheuristics methods have been proposed to solve this type of problem. Wu et al.[11] develop a branch-and-bound algorithm to optimize the two-sided ALB and test the proposed model on well-known problems from the literature. Xiaofeng et al.[12] develop a branch-and-bound algorithm to solve the two-sided ALB problem with an objective of minimizing the length of the assembly line. A heuristic procedure named group assignment is proposed by Lee et al.[13] to maximize the work relatedness and the work slackness with little or any loss in cycle time and the number of workstations. Different metaheuristics have been proposed to solve the balancing problem in two-sided assembly lines. Kim et al.[5] propose a genetic algorithm (GA) to solve this type of problem. Tabu search algorithm, simulated annealing (SA) algorithm, particle swarm optimization (PSO), and bee algorithm have been proposed to solve different objectives in two-sided ALB problems.[14–17] The literature mentioned so far mainly deals with single model problems. A few researchers have focused on mixed and multi-model two-sided ALB problems and used metaheuristics to solve these problems.[18,19]

The RALB problem is first addressed by Rubinovitz and Bukchin[20] who later proposed a branch-and-bound algorithm[21] to balance the robotic assembly line. Levitin et al.[6] propose to use GAs to solve RALB problem with the objective of minimizing the cycle time (type-II RALB). Gao et al.[9] develop a 0-1 integer programming problem for solving type-II RALB problem and also propose a hybrid genetic algorithm (hGA) to solve the proposed problem. Yoosefelahi et al.[22] propose a multi-objective model for RALB to minimize the cycle time, robot setup costs, and robot costs. They propose a new mixed-integer linear programming model to solve the problem and also propose three versions of multi-objective evolution strategies (MOES). Recently, Nilakantan et al.[2] and Nilakantan and Ponnambalam[23] use PSO algorithm to solve two types of layout of RALB problems and test the models on benchmark problems. In the case of two-sided assembly line with robotic systems with a mixed model, Aghajani et al.[24] propose a SA-based approach with an objective of minimizing the cycle time.

Although researchers have focused on two-sided ALB problems and RALB problems, the literature review suggests that very limited number of researchers focus on the two-sided robotic assembly line balancing problem (TRALB). Note that TRALB problems with the objective of minimizing cycle time as these types of assembly lines are widely used in a number of industries and optimizing this objective is a very critical process. Hence, the main focus of this article is to optimize cycle time of a TRALB problem. This article mainly presents four contributions to this research field: (1) A new TRALB (type-II TRALB) problem is proposed and a set of benchmark problems are generated. (2) A mathematical model for the proposed problem is presented and CPLEX solver is applied to obtain the optimal solutions for small-sized problems. (3) Co-evolutionary particle swarm optimization (C-PSO) as a metaheuristic method is developed to solve the proposed problem due to its NP-hard nature. The proposed C-PSO utilizes local search on the global best individual, modification of global best, and restart mechanism to emphasize intensification and exploration. (4) Solutions obtained

using the proposed metaheuristic are compared with other well-known metaheuristic algorithms. From this study, it could be seen that the proposed metaheuristic algorithm performs better than other metaheuristic algorithms for most of the problems.

The remainder of this article is organized as follows. Section "Problem description and mathematical model for TRALB" presents the mathematical model and assumptions considered. In section "C-PSO algorithm," a detailed implementation of the proposed metaheuristic algorithm is presented. Section "Computational results" presents the comparative study of the computational results and in section "Managerial implications and conclusion," the findings of this research are concluded.

Problem description and mathematical model for TRALB

In this section, problem is described along with the assumptions considered in this study. Mathematical model of the addressed TRALB problem is discussed in detail.

Problem description and assumptions

Two-sided robotic assembly lines are usually utilized to produce high volume products, in which robots rather than human beings perform the tasks. This line has great ramifications in industry, such as car assembly, since it has larger flexibility in the face of diverse customers' demands by pre-programming the robots. This line can also preserve the quality of the assembled products since the robots can perform the tasks continually without the worries of fatigue.

In this line, the mated-stations are connected together with a material handling system and each mated-station comprises two facing workstations. A two-sided robotic assembly line is illustrated in Figure 1. In the given figure, there are four robots and eight tasks assigned to two stations in the two-sided assembly line. There are three types of assembly tasks which are to be performed at each workstation (L-type tasks, R-type tasks, and E-type tasks). L-type (R-type) tasks must be allocated to the left (right) side of a mated-station, whereas E-type tasks can be allocated to either side. Each workstation is allocated with a robot to perform the assigned tasks.

TRALB consists of two sub-problems: ALB and robot allocation. In the ALB, all the tasks must be assigned to workstations while satisfying the precedence relationship constraint, direction constraint, and cycle time constraint. In robot allocation, the best fit robot is allocated to each workstation to perform the assigned tasks optimizing one criterion. The assumptions considered for the proposed problem are listed as follows:

1. The assembly line is designed for a unique model of a single product.
2. The operation time of a task depends on the type of the assigned robot and it is deterministic.
3. Each workstation is allocated with one robot and the number of workstations is equal to the number of available robots.
4. Each task can be performed by any robot and each robot can be allocated to any workstation.
5. The setup times between tasks are ignored.
6. No work-in-process inventory is considered.

Notations

The following notations are used for the proposed problem.

Indices

i, h, p:	Tasks
j, g:	Mated-stations
k, l:	A side of the line; $k = \begin{cases} 1, & \text{left side} \\ 2, & \text{right side} \end{cases}$
(j, k):	Station of mated-station j at side k
R:	Robot type

Parameters

nt:	Number of tasks
nm:	Number of mated-stations
nr:	Number of robots
I:	Set of tasks, $I = \{1, 2, \ldots, i, \ldots, nt\}$
J:	Set of mated-stations, $J = \{1, 2, \ldots, j, \ldots, nm\}$
R:	Set of robot types, $r = \{1, 2, \ldots, j, \ldots, nm\}$
t_{ir}:	Processing time of task i by robot r.
AL:	Set of tasks with left direction, $AL \subseteq I$
AR:	Set of tasks with right direction, $AR \subseteq I$
AE:	Set of tasks either direction, $AE \subseteq I$
P_0:	Set of tasks that have no immediate predecessors
$P(i)$:	Set of immediate predecessors of the task i
$Pa(i)$:	Set of all predecessors of the task i
$Sa(i)$:	Set of successors of tasks i
$S(i)$:	Set of immediate successors of the task i
ψ:	Large positive number
$C(i)$:	Set of tasks whose operation directions are opposite to that of task i $C(i) = \begin{cases} AL & \text{if } i \in AR \\ AR & \text{if } i \in AL \\ \varphi & \text{if } i \in AE \end{cases}$
$K(i)$:	Set of integers that indicate the preferred directions of the task i

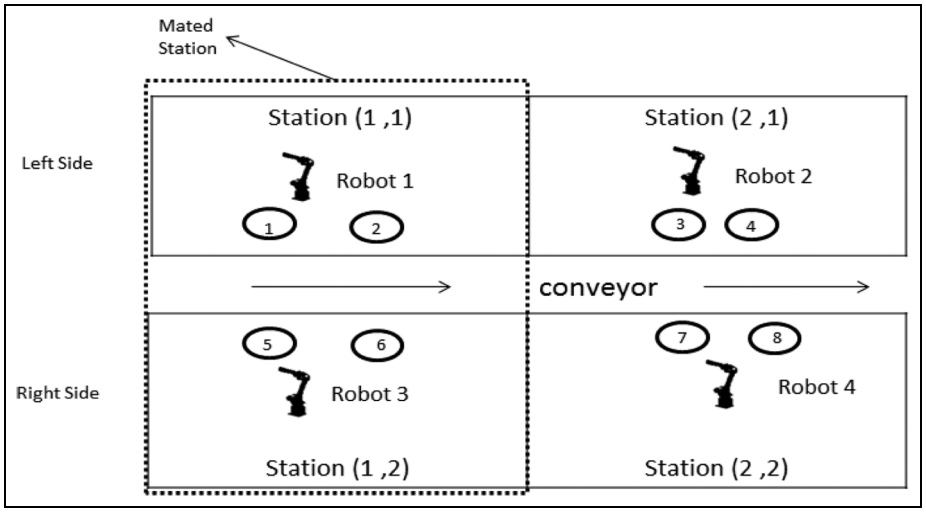

Figure 1. Two-sided robotic assembly line.

$$K(i) = \begin{cases} \{1\} & \text{if } i \in AL \\ \{2\} & \text{if } i \in AR \\ \{1,2\} & \text{if } i \in AE \end{cases}$$

PC: Set of pairs of tasks and predetermined stations for positional constraint

PZ: Set of pairs of tasks for positive zoning constraint

NZ: Set of pairs of tasks for negative zoning constraint

SC: Set of pair of tasks for synchronism constraint

Decision variables

CT: Cycle time

x_{ijk}: 1, if task i is assigned to mated-station j at side k; 0, otherwise

y_{rjk}: 1, if robot r is assigned to mated-station j at side k; 0, otherwise

t_i^f: Finishing time of task i

Indicator variables

z_{ip}: 1, if task i is assigned earlier than task p in the same station; 0, otherwise

Mathematical formulation of TRALB

A mathematical model for TRALB problem with an objective of minimizing cycle time is developed below with the objective of minimizing the cycle time based on the notations and the considered assumptions

$$\text{Min } CT \tag{1}$$

$$\sum_{j\in J}\sum_{k\in K(i)} x_{ijk} = 1 \ \ \forall i \in I \tag{2}$$

$$\sum_{g\in J}\sum_{k\in K(h)} g \cdot x_{hgk} \le \sum_{j\in J}\sum_{k\in K(i)} j \cdot x_{ijk} \ \ \forall i \in I - P_0, \ h \in P(i) \tag{3}$$

$$t_i^f \le CT \ \ \forall i \in I \tag{4}$$

$$t_i^f - t_h^f + \psi(1 - x_{ijk}) + \psi\left(1 - \sum_{l\in K(h)} x_{hjl}\right) \ge \sum_{r=1}^{nr} t_{ir} y_{rjk}$$
$$\forall i \in I - P_0, \ h \in P(i), \ j \in J, \ k \in K(i) \tag{5}$$

$$t_p^f - t_i^f + \psi(1 - x_{ijk}) + \psi(1 - x_{pjk}) + \psi(1 - z_{ip}) \ge \sum_{r=1}^{nr} t_{pr} \cdot y_{rjk} \ \ \forall i \in I$$
$$p \in \{r | r \in I - (P_a(i) \cup S_a(i) \cup C(i)) \text{ and } i < r\}, \ j \in J, \ k \in K(i) \cap K(p) \tag{6}$$

$$t_i^f - t_p^f + \psi(1 - x_{ijk}) + \psi(1 - x_{pjk}) + \psi \cdot z_{ip} \ge \sum_{r=1}^{nr} t_{ir} \cdot y_{rjk} \ \ \forall i \in I$$
$$p \in \{r | r \in I - (P_a(i) \cup S_a(i) \cup C(i)) \text{ and } i < r\}, \ j \in J, \ k \in K(i) \cap K(p) \tag{7}$$

$$t_i^f + \psi(1 - x_{ijk}) \ge \sum_{r=1}^{nr} t_{ir} \cdot y_{rjk} \ \ \forall \text{i} \in I, \ j \in J, \ k \in K(i) \tag{8}$$

$$\sum_{r=1}^{nr} y_{rjk} \ \ \forall j \in J, \ k = 1,2 \tag{9}$$

$$\sum_{j=1}^{nm}\sum_{k=1}^{2} y_{rjk} = 1 \ \ \forall r \in R \tag{10}$$

$$x_{ijk} = 1 \ \ \forall (i,(j,k)) \in PC), \ k \in K(i) \tag{11}$$

$$x_{ijk} - x_{hjk} = 0 \ \ \forall (i,h) \in PZ, \ k \in K(i) \cap K(h) \tag{12}$$

$$\sum_{k\in K(i)} x_{ijk} + \sum_{k\in K(h)} x_{hjk} \le 1 \ \ \forall (i,h) \in NZ \tag{13}$$

$$x_{ijf} - x_{hjk} = 0 \ \ \forall (i,h) \in SC, \ k \in K(h), \ f \in K(i), \ k \ne f \tag{14}$$

$$t_i^f - t_i = t_h^f - t^h \ \ \forall (i,h) \in SC \tag{15}$$

The objective function (1) minimizes the cycle time. Constraint (2) ensures that each task is allocated exactly a side of one mated-station. Constraint (3) is the precedence constraint and constraint (4) is the cycle time constraint. Constraints (5)–(7) control the sequence-dependent finishing time. For a pair of task i and h, if h is an immediate predecessor of i on a same mated-station, then constraint (5) becomes active and it is reduced to $t_i^f - t_h^f \ge \sum_{r=1}^{nr} t_{ir} \cdot y_{rjk}$. If a pair of task i and task h has no precedence relations and they are allocated to one same station, then constraints (6) and (7) become active. If task i is assigned earlier than task p, then constraint (6) becomes active and it is reduced to $t_p^f - t_i^f \ge \sum_{r=1}^{nr} t_{pr} \cdot y_{rjk}$. Otherwise, constraint (7) becomes active and it is reduced to $t_i^f - t_p^f \ge \sum_{r=1}^{nr} t_{ir} \cdot y_{rjk}$. Constraint (8) ensures that the finishing time of task i is larger than or equal to the operation time of task i. Constraint (9) guarantees that only one robot is allocated to each station and constraint (10) enforces that each robot can be assigned to only one station. Constraint (11) handles the positional constraint. Constraints (12) and (13) deal with the positive zoning constraint and the negative zoning constraint, respectively. Constraints (14) and (15) enforce that each pair of tasks in the synchronism constraint is allocated to the opposite sides of a same mated-station with the same starting time.

C-PSO algorithm

The ALB problem is a well-known NP-hard problem. The proposed problem also falls under this category due to additional constraints. Hence, a metaheuristic based on PSO is proposed to solve them. Researchers have extensively implemented PSO algorithm (a population-based stochastic optimization technique developed by Kennedy and Eberhart[25] to solve ALB problems). The social behavior of bird flocking and fish schooling when they are in search of food serves as inspiration for the development of PSO. The major advantages of using PSO[26] compared to other metaheuristics are as follows: ease of implementation, robustness, and very few parameters to be fine-tuned for achieving the results. However, PSO algorithm may easily get trapped in a local optimum when tackling complex problems.[27] In this article, concept of co-evolution is incorporated to the standard PSO to suit the problem and is termed as C-PSO algorithm that is used to evaluate the problem addressed in this article.

The proposed C-PSO consists of two sub-swarms and each sub-swarm tackles one sub-problem. The two sub-swarms evolve alternately and only one sub-swarm evolves each time, while the other is kept fixed. To evaluate the individuals of a sub-swarm, the best individual of the other sub-swarm is taken as the context vector and then a solution can be constructed. The fitness of the individual and the context vector is regarded with the fitness of these individuals. Once each objective is calculated, the population can proceed with the evolution mechanism of PSO. Cooperative mechanisms of co-evolutionary algorithms are used and three improvement strategies are proposed: local search for the global best solution, modification of global best, and restart mechanism. Detailed implementations of the strategies are presented in the following section.

Solution representation

Nilakantan et al.[2] develop a task permutation-oriented encoding and decoding procedure for the robotic assembly line, and the robot which can finish the tasks with smallest time is selected. This method is practicable only for one-sided assembly line problems and this method will be very complex to be used in solving two-sided ALB problems. This is due to the fact that in two-sided assembly line, there is an interference of the tasks on two sides of mated-station which cannot be ignored and the robots that can finish the tasks with smallest time need to be decided as a pair of two robots. Therefore, it is complex to obtain the best combinations for robots on each mated-station.

Hence, in this article, a new way of representing the solution is presented and it is described as follows: taking the first mated-station as example, $2nm \times (2nm-1)$ combinations should be checked before selecting the best combination with smallest operation time. Therefore, two vectors μ_p and λ_p are utilized for task permutation and robot assignment, respectively, where $p(p \in \{1, 2, \ldots, PS\})$ is a solution in the population. An example for a 12-task problem with initial cycle time of 8 time units is depicted in Figure 2.

The vector λ_p determines the assignment of robots and each element represents a station. For instance, the number in the first position is 3 and thus, robot 3 is allocated to the first station (the left side of the first mated-station). The vector μ_p determines the task permutation and each element indicates a task. Task 3 is in the first position, and therefore, task 3 has highest priority and is allocated first. Once the two vectors are determined, the detail assignment of all the tasks is done using a decoding procedure which is explained as follows. Note that the cycle time constraint in Step 1 can be violated for the last mated-station so as to

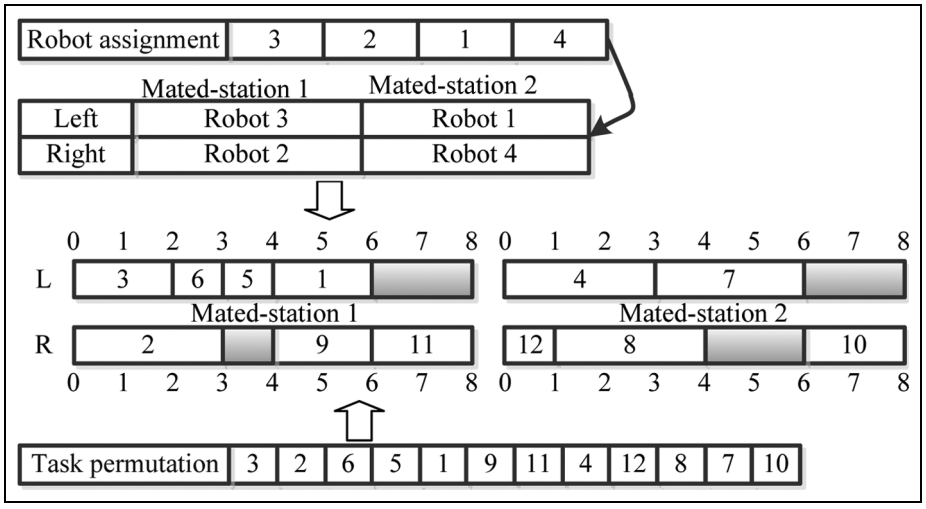

Figure 2. Solution representation for 12-task problem.

obtain a feasible solution and the largest finishing time of mated-stations is taken as current cycle time:

Step 1: Decide whether assignable task exists and a task is assignable while satisfying constraints (2–8).
Step 2: If no assignable task exists, this procedure ends. Otherwise, execute Step 3.
//Station selection mechanism
Step 3: Obtain assignable task set for both sides and then execute Step 4.
Step 4: If only one side has an assignable task, this side is selected and go to Step 6. Otherwise, execute Step 5.
Step 5: Select the side with smaller ending time or select the left side by default when the capacities of both sides are equal. Then, execute Step 6.
//Task selection mechanism
Step 6: Delete tasks which result in idle times when there are tasks which can be operated at the earliest start time of the current station.
Step 7: Select the assignable task which is the former position of task permutation and then execute Step 1.

The improved decoding has mainly two features: station selection mechanism and task selection mechanism. The station selection mechanism selects the side with large capacity, which can lead to better balance of the two sides. The selection of the left side by default can reduce the search space to a large extent. The task selection mechanism deals with sequence-dependent idle times and guarantees that the tasks are assigned without generating idle times being selected at first.

Population evolution

Due to the discrete attribute of the ALB problem, the original PSO which is designed for continuous optimization problem cannot be applied directly. A transformation is necessary, and there are many methods published, including the random key method in Bean[28] and the method used by Nilakantan et al.[2] This article develops a simple population updating mechanism, which is described with the following two expressions for task permutation and robot assignment, respectively

$$\mu_p = \begin{cases} g(h(\mu_p), \mu_{GBest}), & \text{if } Rand() \le c \\ g(h(\mu_p), \mu_{LBest}) \end{cases} \tag{16}$$

$$\lambda_p = \begin{cases} g(h(\lambda_p), \lambda_{GBest}), & \text{if } Rand() \le c \\ g(h(\lambda_p), \lambda_{LBest}) \end{cases} \tag{17}$$

where $h(\mu_p)$ or $h(\lambda_p)$ means the self-modification, $g(h(\mu_p), \mu_{GBest})$ or $g(h(\lambda_p), \lambda_{GBest})$ means updating positions to global best solution, $g(h(\mu_p), \mu_{LBest})$ or $g(h(\lambda_p), \lambda_{LBest})$ indicates updating positions to local best solution. These three operations respond to the initial velocity, moving to global best and moving to local best, respectively. *Rand*() is a random number between 0 and 1, and c is an acceleration coefficient within (0,1). h (*) is achieved with swap operator or insert operator and g (*, *) can be realized with crossover operator.

In this article, the insert operator, swap operator, and two-point crossover operator are applied to $h(\mu_p)$, $h(\lambda_p)$, and g (*,*), respectively, after testing insert operator, swap operator, one-point crossover operator, and two-point crossover operator. An example of two-point crossover operator for task permutation is depicted in Figure 3. After executing positions update, the greedy acceptance is applied to preserve the better one between new solution and incumbent one. This greedy acceptance is helpful to overcome prematurity and preserve the diversity of the population.

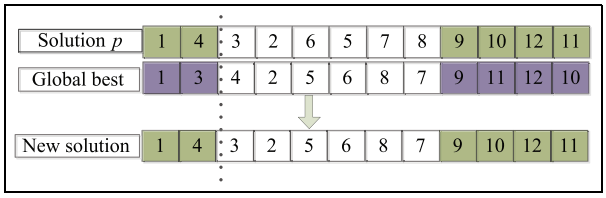

Figure 3. Two-point crossover operator for task permutation.

```
Procedure Local search for global best
Iter=0;
While (Iter<nt*nt) do
  Iter: =Iter+1;
  If (Rand () <0.5)
   Apply insert operator to task permutation once;
   If (Rand () <0.5)
    Apply swap operator to robot assignment once;
   Endif
  else
   Apply swap operator to robot assignment once;
   If (Rand () <0.5)
    Apply insert operator to task permutation once;
   Endif
  Apply greedy acceptance
```

Figure 4. Local search for global best.

Local search for the global best

It is clear that the two sub-swarms interact through the global best, and the quality of global best affects the quality of the final solution to a large extent. Thus, a strong local search method is developed and it is executed only when a new global best is obtained. This local search permits the variation of only task permutation or robot assignment and also allows the variation of both task permutation and robot assignment. This allowance of modification of two vectors simultaneously can increase the search space to a large extent. The termination criterion is set as an iteration time fixed to the square of the number of tasks so that there is a large possibility for each task being allocated to all the possible positions. The procedure for the local search is presented in Figure 4.

Modification of global best and restart mechanism

The C-PSO algorithm is easily trapped into local optima if the current global best remains unchanged. To obtain a new global best, other particle in the population or a restart mechanism must be employed. This article develops two methods, called modification of global best and restart mechanism.

In the modification of global best, an individual in one sub-swarm cooperates with a neighborhood of the best individual of the other sub-swarm. For vector μ_p, a new solution is obtained by cooperating with μ'_{GBest} applied with a swap operator. After all the individuals in this sub-swarm for task permutation are tested, an individual in the other sub-swarm cooperates with μ'_{GBest} applied with the insert operator. This procedure can further explore the neighborhood of global best solution and the search space of other solutions. In fact, the similarity of global best and other individuals grows with each iteration and the above procedure can expand the search space. The modification of global best is applied when the global best has not been improved in "θ" iterations and it continues only when a new global best solution is obtained.

If the modification of global best cannot further improve the global best, restart mechanism is utilized. It consists of two steps: replacing the repeated individual with a random generated vector and selecting the best combination (μ_p, λ_p) $\forall p \in \{1, 2, \ldots, PS\}$ as the current global best. After involution, some repeated individuals may arise and thus, these individuals are replaced with a random generated vector. For the new global best, all combinations (μ_p, λ_p) are tested and the best one is selected, which guarantees the quality of new global best. After generating a new global best, the local search in section "Local search for the global best" can be applied to further improve the quality of the global best. The restart mechanism is utilized only when global best has not been improved in φ' iterations.

Overall procedure of the C-PSO

The proposed C-PSO is improved to solve the investigated problem with the incorporation of a restart mechanism and the modification of the best solution. The procedure of the C-PSO is depicted in Appendix 1. Here, PS is the size of each sub-swarm, φ is the number of iterations before carrying out restart mechanism procedure, and θ is the number of iterations before executing modification on the best solution.

Computational results

This section aims at testing the performance of the proposed co-evolutionary algorithm and carries out a comprehensive comparison with different methods. First, the details of the benchmark problems and the metaheuristic algorithms are explained. In the later part of this section, details of the parameter calibration are presented. Then, the experimental results are described in detail and the performance of the proposed algorithm is compared with other algorithms using statistical techniques.

Table 1. Details of datasets.

Problem	Category	Number of tasks	Source
P9	Small-sized	9	Kim et al.[5]
P12	Small-sized	12	Kim et al.[5]
P16	Small-sized	16	Lee et al.[13]
P24	Small-sized	24	Kim et al.,[5]
P65	Large-sized	65	Lee et al.[13]
P148	Large-sized	148	Bartholdi[4]
P205	Large-sized	205	Lee et al.[13]

Experimental design

Since there is limited literature dealing with two-sided robotic assembly lines, a set of benchmark cases are generated based on the well-known benchmark problems of two-sided ALB problems. Seven problems are modified from the original TRALB problems to suit the proposed problem. The details of the seven problems used are presented in Table 1.

P9, P12, and P16 are classified as small-sized problems and P24, P65, P148, and P205 are categorized as large-sized problems. The seven problems cover all the benchmark problems on two-sided ALB problem and there are several cases with different numbers of mated-station for each problem. The precedence diagrams and the preferred directions of tasks are taken from the literature directly. To fit to the proposed problem in the article, the operation time of task i by robot r is randomly generated between $[t_i \times 0.8, t_i \times 1.2]$, where t_i is the original published operation time in the above-mentioned literature. Due to space constraints, the operation times of tasks by robots are not presented and they are available upon request.

To test the performance of the proposed C-PSO, five other competitive algorithms are selected, including PSO algorithm, a GA,[29] artificial bee colony (ABC) algorithm,[30] SA algorithm,[31] and a co-evolutionary genetic algorithm (C-GA).[29] PSO algorithm shares the same procedure as the C-PSO except for utilizing best individual of the other sub-swarm, and PSO has only one swarm. GA and ABC also have only one swarm, while C-GA has two sub-swarms and utilizes the best individual of the other sub-swarm for population evolution. GA, ABC, SA, and C-GA are selected since they have been applied to solve ALB problems, and they have been utilized to solve the problem with two sub-problems, namely ALB and sequencing problems. All the algorithms are carefully re-implemented and a test campaign among them is carried out with three different stopping times ($t = nt \times nt \times \rho$ ms, $\rho = 10, 20, 30$) on the seven problems. The termination criteria are set based on the computational time for decoding once which can be with multiple $nt \times nt$ approximately. This expression guarantees that the times of decoding are similar for different cases and three different termination criteria can make the result comparison more reasonable. All the algorithms are coded in C++ programming language on the platform of Microsoft Visual Studio 2012. The proposed algorithms are tested on computers with Intel(R) Core2(TM) CPU 2.33 GHZ and 3.036 GB RAM. All the programming is available upon request from readers.

Calibration of the algorithms

This section determines the best combination of parameters and shows the effectiveness of the improvement strategies in section "C-PSO algorithm," including the local search for the global best solution, modification of global best, and restart mechanism. The C-PSO has five more parameters: the particle number in each swarm, the number of swarms, the acceleration coefficient (c), the consecutive iterations for modification of global best (θ), and the consecutive iterations for restart mechanism (φ). Based on preliminary experiments, the possible values of parameters are set as follows: particle number in each swarm with three levels (20, 40, and 60); number of swarms at three levels (4, 6, and 8); c at four levels (0.4, 0.5, 0.6, and 0.7); θ at four levels (20, 50, 100, and 200); and φ at four levels (20, 40, 60, and 80). All these levels result in a total of $3 \times 3 \times 4 \times 4 \times 4 = 576$ different configurations and more configurations arise if three improvement strategies are considered.

To avoid tremendous computational time, other factors are fixed first and three improvements are tested: with or without utilization of the local search, with or without utilization of modification of global best, and with or without application of restart mechanism. To avoid over-calibration, one case of the largest problem (P205 with six mated-stations) is selected and each combination tests this case five times. Note that the combination of parameters for the largest case can be applied to the small-sized problem with the cost of large computational times. In the experiments, the algorithms with different parameters can obtain similar results for small-sized problems with the computational time increasing. Once these experiments are finished, the relative percentage increase (RPI) is applied to transfer the experimental data. The expression $\text{RPI} = (Sol_{some} - Sol_{best})/Sol_{best} \times 100$ is utilized to calculate the RPI, where Sol_{some} is the solution by an algorithm for one case and Sol_{best} is the best solution among all the algorithms for the same tested case. The multifactor analysis of variance (ANOVA) technique[32] is proposed to analyze these RPI values and select the best parameter combination after checking three hypotheses (independence of the residuals, homogeneity of variance, and normality of the residuals). As for the ANOVA results, the p value is an important factor

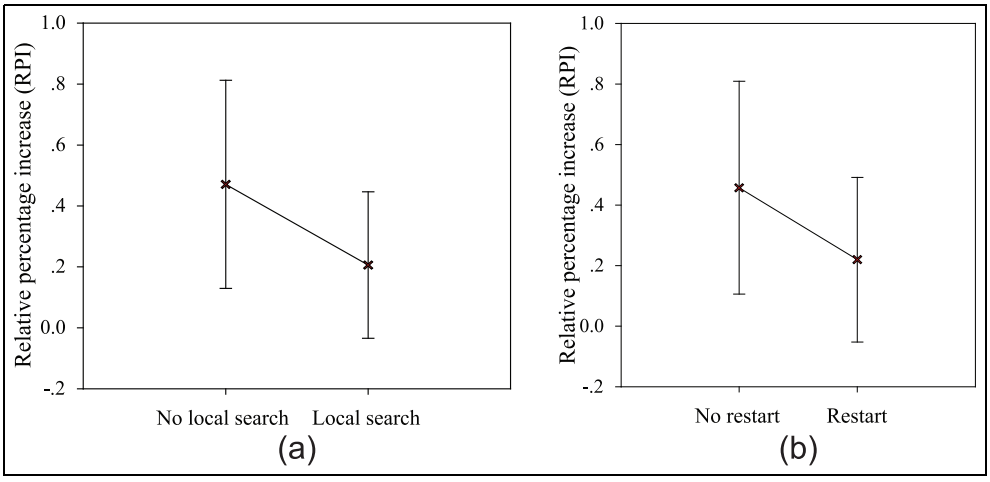

Figure 5. Means plots and 95% Tukey's HSD confidence intervals of two improvements of C-PSO: (a) mean plot for the local search and (b) mean plot for the restart mechanism.

and the effect of different levels is significantly different statistically if the p value is less than a predetermined number (0.05 is set in this article).

All the p values for three improvements are less than 0.05, and thus, they have important effect on the improved C-PSO. Based on the ANOVA results, the local search has the smallest p value, which suggests the local search has the most important effect on the final fitness. The mean plot of local search is depicted in Figure 5(a) with 95% Tukey's honest significant difference (HSD) confidence intervals. The mean plot suggests that the C-PSO with local search performs better than the one without local search. The mean plot of restart mechanism which achieves the second smallest p values is also plotted in Figure 5(b).

It is observed that the restart mechanism can improve the performance of C-PSO statistically. Consecutively, the modification of the global best can also improve the performance of C-PSO. All the experimental results demonstrate the efficiency of the three improvements for C-PSO.

More experiments are carried out to select the best combination for five more parameters, and ANOVA technique is also applied to analyze the final results. The final combination used in this article is as follows: particle number in each swarm is 40, swarm number is 6, $c = 0.5$, $\theta = 40$, and $\varphi = 50$.

Comparison of computational results among algorithms

This section details the campaign of experiments and analyzes the computational results. A total of 39 cases of the 7 problems are solved by each algorithm, and a total of 20 independent runs are performed for each case. The proposed problem is tested on the well-known metaheuristics such as GA, ABC, SA, and PSO. The proposed model is also tested on CPLEX optimization solver and it could be seen that optimal solution could be achieved only for the small-sized problems (P9, P12, and P16) and in the case of large-sized problems, CPLEX could not solve them due to longer computational time. The RPI ($\text{RPI} = (Sol_{some} - Sol_{best})/\ Sol_{best} \times 100$) is applied to transfer the computational results obtained through these metaheuristics and CPLEX. For small-sized problems, results obtained using the CPLEX solver are taken as Sol_{best}. For large-sized problems, Sol_{best} is the best solution among all algorithms within 20 times' run when $\rho = 30$ since there is no optimal solution available using CPLEX solver. The average RPI values of each problem for 20 runs are reported in Table 2, and the results for three termination criteria are all reported.

From Table 2, it is observed that the C-PSO performs well for all the cases regarding to the overall RPI values and it outperforms the others under all three termination criteria. If one focuses on the computational results with at $\rho = 10$, it can be seen that the first smallest overall RPI value belongs to C-PSO which is only 1.47% at $\rho = 10$, the second smallest RPI value belongs to PSO which is 1.52%, and the third smallest one corresponds to C-GA which is 2.52%. C-PSO obtains the smallest RPI values for the three small cases and largest-size case. GA, ABC, and SA show much worse overall RPI values, and the cooperative method C-GA obtains better results, which demonstrate the effectiveness of the cooperation for TRALB problems.

For other two termination criteria, $\rho = 20,\ 30$, it can be seen that all the algorithms can improve their results with more computational time, but SA as a local search method can improve only a little. The proposed C-PSO

Table 2. Average RPI values for algorithms.

Problem	*nm*	Average relative percentage increase						CPU time (s)
		PSO	GA	ABC	SA	C-GA	C-PSO	
$\rho = 10.00$								
P9	2, 3	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	0.81
P12	2, 3, 4, 5	**0.00**	**0.00**	**0.00**	**0.00**	2.19	**0.00**	1.44
P16	2, 3, 4, 5	**0.00**	**0.00**	**0.00**	3.23	**0.00**	**0.00**	2.56
P24	2, 3, 4, 5	**1.06**	2.18	1.08	3.49	2.24	1.19	5.76
P65	4, 5, 6, 7, 8	2.49	4.33	3.33	5.03	3.37	2.14	42.25
P148	4, 5, 6, 7, 8, 9, 10, 11, 12	**1.93**	3.87	3.11	4.34	2.68	2.04	219.04
P205	4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14	2.29	4.60	4.65	4.27	3.58	**2.15**	420.25
Average RPI for all cases		1.52	2.97	2.57	3.54	2.52	**1.47**	–
$\rho = 20.00$								
P9	2, 3	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	1.62
P12	2, 3, 4, 5	**0.00**	**0.00**	**0.00**	**0.00**	0.62	**0.00**	2.88
P16	2, 3, 4, 5	**0.00**	**0.00**	**0.00**	3.23	**0.00**	**0.00**	5.12
P24	2, 3, 4, 5	0.91	2.18	0.91	3.44	1.62	**0.83**	11.52
P65	4, 5, 6, 7, 8	2.28	3.86	2.77	4.98	2.84	**1.53**	84.50
P148	4, 5, 6, 7, 8, 9, 10, 11, 12	1.45	3.57	2.70	4.24	2.27	**1.44**	438.08
P205	4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14	1.56	4.13	4.13	4.15	3.10	**1.23**	840.50
Average RPI for all cases		1.16	2.71	2.23	3.47	1.99	**0.96**	–
$\rho = 30.00$								
P9	2, 3	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	2.43
P12	2, 3, 4, 5	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	4.32
P16	2, 3, 4, 5	**0.00**	**0.00**	**0.00**	3.23	**0.00**	**0.00**	7.68
P24	2, 3, 4, 5	0.85	1.80	0.80	3.44	1.48	**0.74**	17.28
P65	4, 5, 6, 7, 8	2.06	3.52	2.47	4.95	2.62	**1.24**	126.75
P148	4, 5, 6, 7, 8, 9, 10, 11, 12	1.25	3.40	2.47	4.13	2.05	**1.13**	657.12
P205	4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14	1.31	3.84	3.90	4.11	2.85	**0.84**	1260.75
Average RPI for all cases		1.01	2.50	2.07	3.43	1.76	**0.73**	–

RPI: relative percentage increase; CPU: central processing unit; PSO: particle swarm optimization; GA: genetic algorithm; ABC: artificial bee colony; SA: simulated annealing; C-GA: co-evolutionary genetic algorithm; C-PSO: co-evolutionary particle swarm optimization.
Best solution is presented in bold.

again achieves better overall RPI value than the other four methods. Based on the results in Table 2, it is reasonable to state that the proposed C-PSO method is a highly effective method for solving TRALB problems.

In order to show the interaction between the computational time and algorithms, a multifactor ANOVA test is carried out. The CPU times and the algorithms are set as two factors. To have a better picture, important four algorithms are compared; PSO, GA, C-GA, and C-PSO, and ANOVA results show that there is a significant difference among the two factors. The mean plots of the interaction of the two factors are plotted in Figure 6. It can be observed that the C-PSO outperforms the other three methods clearly for all three termination criteria.

The best results obtained by all metaheuristics for all the problem cases are reported and compared. Detailed results obtained for different cases for each problem are presented in Table 3. Table 3 also reports the optimal solutions (OPT) by CPLEX and the best results by algorithms at $nt \times nt \times 30$ ms. In the case of the small-sized problems, all metaheuristic algorithms can obtain optimal solutions and they utilize much less computational time. However, CPLEX is not able to solve the large-sized problems while the near-optimal results obtained using the metaheuristic within acceptable computational time are presented.

The proposed C-PSO obtains the best results for almost all problems. In addition, the C-PSO obtains best results for almost all large-sized cases. Specifically, the C-PSO obtains best results for 22 cases of P65, P148, and P205, and C-GA obtains best results for one case of P65, P148, and P205. If the C-PSO is removed, PSO ranks second in the number of cases obtaining best result and the C-GA ranks third. The above comparison further demonstrates the exploration capacity of the C-PSO, PSO, and C-GA and also proves the superiority of the C-PSO over other methods regarding to the best cycle time.

To explain the superiority of the C-PSO, a detailed comparison is carried out on the evolution process on P205 with six mated-stations. Note that the restart mechanism for PSO and C-PSO is not employed in this case.

To obtain a better picture, parts of the algorithms are reported in Figure 7(a) and (b). Figure 7(a) depicts

Table 3. Result comparison among algorithms.

Problem	*nm*	CPLEX		PSO	GA	ABC	SA	C-GA	C-PSO
		CT	CPU time (s)						
P9	2	4	0.21	**4**	**4**	**4**	**4**	**4**	**4**
	3	3	0.18	**3**	**3**	**3**	**3**	**3**	**3**
P12	2	6	0.38	**6**	**6**	**6**	**6**	**6**	**6**
	3	4	0.36	**4**	**4**	**4**	**4**	**4**	**4**
	4	4	1.45	**4**	**4**	**4**	**4**	**4**	**4**
	5	3	1.33	**3**	**3**	**3**	**3**	**3**	**3**
P16	2	21	1.57	**21**	**21**	**21**	**21**	**21**	**21**
	3	14	0.92	**14**	**14**	**14**	**14**	**14**	**14**
	4	13	412.15	**13**	**13**	**13**	**13**	**13**	**13**
	5	9	2.38	**9**	**9**	**9**	**9**	**9**	**9**
P24	2	34	>3600	**34**	**34**	**34**	**34**	**34**	**34**
	3	N/A	N/A	**22**	**22**	**22**	**22**	**22**	**22**
	4	N/A	N/A	**17**	**17**	**17**	**17**	**17**	**17**
	5	N/A	N/A	**14**	**14**	**14**	**14**	**14**	**14**
P65	4	N/A	N/A	583	583	587	592	585	**577**
	5	N/A	N/A	459	458	459	465	457	**455**
	6	N/A	N/A	383	383	386	388	382	**379**
	7	N/A	N/A	330	336	335	338	334	**330**
	8	N/A	N/A	**290**	298	293	294	291	291
P148	4	N/A	N/A	581	584	585	588	582	**579**
	5	N/A	N/A	469	471	472	472	469	**464**
	6	N/A	N/A	388	398	394	395	389	**386**
	7	N/A	N/A	334	338	337	338	336	**333**
	8	N/A	N/A	292	294	294	296	293	**291**
	9	N/A	N/A	**259**	262	264	261	260	260
	10	N/A	N/A	236	238	237	238	**233**	235
	11	N/A	N/A	215	219	218	218	215	**215**
	12	N/A	N/A	197	203	199	202	196	**196**
P205	4	N/A	N/A	2747	2763	2752	2750	2748	**2726**
	5	N/A	N/A	2222	2233	2221	2242	2232	**2197**
	6	N/A	N/A	1826	1839	1841	1852	1838	**1806**
	7	N/A	N/A	1560	1573	1584	1575	1556	**1549**
	8	N/A	N/A	1367	1372	1390	1384	1369	**1357**
	9	N/A	N/A	1214	1223	1241	1229	1211	**1201**
	10	N/A	N/A	1088	1106	1115	1097	1095	**1085**
	11	N/A	N/A	992	1019	1027	996	1013	**984**
	12	N/A	N/A	920	956	939	933	938	**917**
	13	N/A	N/A	843	868	865	857	861	**832**
	14	N/A	N/A	789	823	818	811	802	**786**

CPU: central processing unit; CT: cycle time; N/A: not available; PSO: particle swarm optimization; GA: genetic algorithm; ABC: artificial bee colony; SA: simulated annealing; C-GA: co-evolutionary genetic algorithm; C-PSO: co-evolutionary particle swarm optimization.
Best solution is presented in bold.

the best cycle time, and Figure 7(b) depicts the average results of the population. In Figure 7(a), it is observed that C-PSO outperforms the others from the beginning to the end of 1400 s and it can obtain better results with increasing computational time. The C-GA can also find better best solution, whereas the GA and ABC can improve a little or not improve after reaching 600 s. These results prove that the C-PSO has stronger capacity of finding a new best solution. In Figure 7(b), it seems that the PSO and ABC can converge fast, whereas there are also large deviations of the average results for GA, C-GA, and C-PSO. Nevertheless, this conclusion can be criticized due to the co-evolutionary mechanism for the C-GA and C-PSO. As mentioned in section "C-PSO algorithm," the best individual of one sub-swarm is taken as the context vector to evaluate the individuals of the other sub-swarm. Once a new best individual is obtained, the best individual for one sub-swarm is updated and the combination of this new best individual of one sub-swarm and most of the individuals from the other sub-swarm may have poor performance resulting in a large average cycle time. However, if the best individual is fixed, the C-PSO can convergence within 300 s (0–300 s) for the first time, within 100 s (300 s–400 s) for the second time, etc. The PSO, on the contrary, takes more than 400 s (0–400 s)

for convergence. Based on the results, it can be concluded that the C-PSO can converge fast once the new global best solution is found.

As a matter of fact, the high efficiency of C-PSO results from two aspects: co-evolutionary mechanism and three improvements especially designed for TRALB on the C-PSO. This article utilizes two vectors, task permutation vector and robot assignment vector, to obtain a feasible solution. The co-evolutionary algorithm is more practical for optimizing several vectors simultaneously, and thus, the C-PSO outperforms the PSO. This conclusion is further proved by the advantage of C-GA over original GA. The C-PSO outperforms the C-GA in three aspects, namely the local search for the global best solution, modification of global best, and restart mechanism. The performance of C-PSO mainly depends on the quality of global best individual, and thus, the local search is developed to emphasize the intensification and obtain a high-quality global best solution. The modification of global best further explores the neighborhoods of the global best solution and the main feature is that an individual in one sub-swarm cooperates with a neighborhood of the best individual resulting in a much larger search space. The restart mechanism is utilized to avoid this algorithm trapped into local optima. These three improvements lead to the higher performance of the C-PSO over the C-GA.

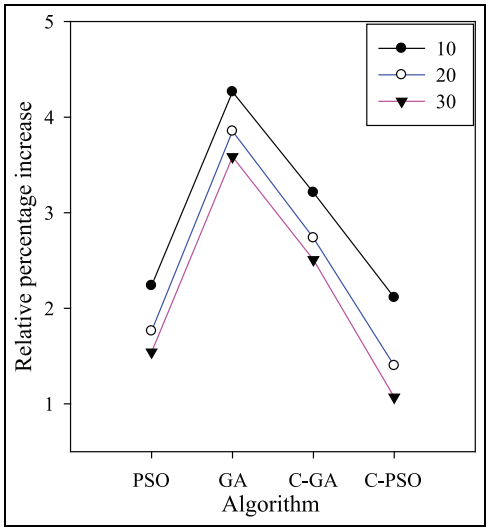

Figure 6. Mean plots for interactions between CPU time and algorithms (p = 10, 20, 30).

Managerial implications and conclusion

Robotic assembly lines are being extensively used by industries due to the benefits such as flexibility and quality of the product. Two-sided assembly lines are being used mainly in automobile industries where large quantities of the same products are assembled. Optimizing cycle time is an important task in assembly lines and to the author's knowledge, there has been no work reported on optimizing cycle time for two-sided robotic assembly lines. This article presents a new type of TRALB problem with an objective of minimizing the cycle time. Optimizing cycle time is an important problem in manufacturing and industry would like to minimize the cycle time to improve productivity and reduce the production cost and throughput time. The proposed model in this article has significant managerial implications. Since the implementation of two-

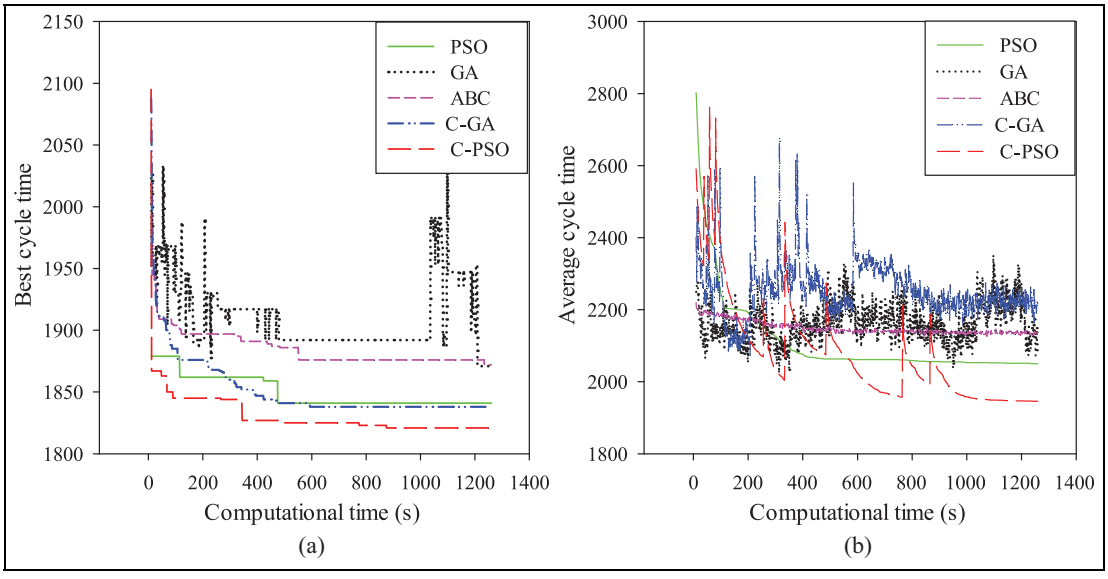

Figure 7. Best cycle time and average cycle time during evolution process: (a) best cycle time with computational times and (b) average cycle time with computational times.

sided robotic assembly line is a cost-intensive process, usage of this proposed model can help the managers or decision-makers at the industries to estimate the resources required for this type of assembly line configuration and their corresponding performance. This study will also help in balancing the resources required and performance of the TRALB. This proposed model can also help for better planning and control of activities in different scenarios.

Since this proposed problem falls in the category of NP-hard problem, a metaheuristic algorithm based on PSO approach is proposed. Co-evolution concept is incorporated to the standard PSO and new strategies to suit the problem are presented. Simulation experiments are conducted on seven modified benchmark data and the obtained results are compared with other well-known metaheuristic algorithms such as GA, PSO, and ant bee colony optimization. From the experimental study, it can be concluded that the proposed C-PSO algorithm obtains better solutions in terms of cycle time and computation time for most of the cases. Results obtained by CPLEX solver are also presented and it could be seen that the CPLEX could find optimal solutions for small-sized problems and the results obtained using C-PSO approach achieve optimal or near-optimal solutions for most of the problems.

In future work, realistic problems can be addressed to eliminate the gap between scientific research and practical implementations by considering more constraints in real application. New methods are also interesting to solve the multi-objective two-sided robotic assembly balancing problem, and the multi-objective cooperative co-evolutionary algorithm may be a good choice. The two-sided robotic assembly line with mixed and multi-models can also be proposed. Another research avenue is the ALB with both robotics and human beings which have great ramifications in industry.

Declaration of conflicting interests

The author(s) declared no potential conflicts of interest with respect to the research, authorship, and/or publication of this article.

Funding

The author(s) disclosed receipt of the following financial support for the research, authorship, and/or publication of this article: This research work is partially funded by the National Natural Science Foundation of China (Grant No. 51275366).

References

1. Rashid MFF, Hutabarat W and Tiwari A. A review on assembly sequence planning and assembly line balancing optimisation using soft computing approaches. *Int J Adv Manuf Tech* 2012; 59: 335–349.
2. Nilakantan JM, Ponnambalam S, Jawahar N, et al. Bio-inspired search algorithms to solve robotic assembly line balancing problems. *Neural Comput Appl* 2015; 26: 1379–1393.
3. Ağpak K, Yegül MF and Gökçen H. Two-sided U-type assembly line balancing problem. *Int J Prod Res* 2012; 50: 5035–5047.
4. Bartholdi J. Balancing two-sided assembly lines: a case study. *Int J Prod Res* 1993; 31: 2447–2461.
5. Kim YK, Kim Y and Kim YJ. Two-sided assembly line balancing: a genetic algorithm approach. *Prod Plan Control* 2000; 11: 44–53.
6. Levitin G, Rubinovitz J and Shnits B. A genetic algorithm for robotic assembly line balancing. *Eur J Oper Res* 2006; 168: 811–825.
7. Kilincci O and Bayhan GM. A Petri net approach for simple assembly line balancing problems. *Int J Adv Manuf Tech* 2006; 30: 1165–1173.
8. Baybars I. A survey of exact algorithms for the simple assembly line balancing problem. *Manage Sci* 1986; 32: 909–932.
9. Gao J, Sun L, Wang L, et al. An efficient approach for type II robotic assembly line balancing problems. *Comput Ind Eng* 2009; 56: 1065–1080.
10. Gutjahr AL and Nemhauser GL. An algorithm for the line balancing problem. *Manage Sci* 1964; 11: 308–315.
11. Wu E-F, Jin Y, Bao J-S, et al. A branch-and-bound algorithm for two-sided assembly line balancing. *Int J Adv Manuf Tech* 2008; 39: 1009–1015.
12. Xiaofeng H, Erfei W, Jinsong B, et al. A branch-and-bound algorithm to minimize the line length of a two-sided assembly line. *Eur J Oper Res* 2010; 206: 703–707.
13. Lee TO, Kim Y and Kim YK. Two-sided assembly line balancing to maximize work relatedness and slackness. *Comput Ind Eng* 2001; 40: 273–292.
14. Özcan U and Toklu B. A tabu search algorithm for two-sided assembly line balancing. *Int J Adv Manuf Tech* 2009; 43: 822–829.
15. Özcan U. Balancing stochastic two-sided assembly lines: a chance-constrained, piecewise-linear, mixed integer program and a simulated annealing algorithm. *Eur J Oper Res* 2010; 205: 81–97.
16. Chiang W-C, Urban TL and Luo C. Balancing stochastic two-sided assembly lines. *Int J Prod Res* 2015; 54: 6232–6250.
17. Özbakır L and Tapkan P. Bee colony intelligence in zone constrained two-sided assembly line balancing problem. *Expert Syst Appl* 2011; 38: 11947–11957.
18. Simaria AS and Vilarinho PM. 2-ANTBAL: an ant colony optimisation algorithm for balancing two-sided assembly lines. *Comput Ind Eng* 2009; 56: 489–506.
19. Chutima P and Chimklai P. Multi-objective two-sided mixed-model assembly line balancing using particle swarm optimisation with negative knowledge. *Comput Ind Eng* 2012; 62: 39–55.
20. Rubinovitz J and Bukchin J. Design and balancing of robotic assembly lines. In: *Proceedings of the fourth world conference on robotics research*, Pittsburgh, PA, 17–19

September 1991. Dearborn, MI: Society of Manufacturing Engineers (SME).
21. Rubinovitz J, Bukchin J and Lenz E. RALB—a heuristic algorithm for design and balancing of robotic assembly lines. *CIRP Ann: Manuf Techn* 1993; 42: 497–500.
22. Yoosefelahi A, Aminnayeri M, Mosadegh H, et al. Type II robotic assembly line balancing problem: an evolution strategies algorithm for a multi-objective model. *J Manuf Syst* 2012; 31: 139–151.
23. Nilakantan JM and Ponnambalam S. Robotic U-shaped assembly line balancing using particle swarm optimization. *Eng Optimiz* 2016; 48: 231–252.
24. Aghajani M, Ghodsi R and Javadi B. Balancing of robotic mixed-model two-sided assembly line with robot setup times. *Int J Adv Manuf Tech* 2014; 74: 1005–1016.
25. Kennedy J and Eberhart R (eds). Particle swarm optimization. In: *Proceedings of the IEEE international conference on neural networks*, Perth, WA, Australia, 27 November–1 December 1995. New York: IEEE.
26. Lee KY and Park J-B (eds). Application of particle swarm optimization to economic dispatch problem: advantages and disadvantages. In: *Proceedings of the power systems conference and exposition (PSCE'06)*, Atlanta, GA, 29 October–1 November 2006. New York: IEEE.
27. He Q and Wang L. An effective co-evolutionary particle swarm optimization for constrained engineering design problems. *Eng Appl Artif Intel* 2007; 20: 89–99.
28. Bean JC. Genetic algorithms and random keys for sequencing and optimization. *ORSA J Comput* 1994; 6: 154–160.
29. Kim YK, Kim JY and Kim Y. A coevolutionary algorithm for balancing and sequencing in mixed model assembly lines. *Appl Intell* 2000; 13: 247–258.
30. Saif U, Guan Z, Liu W, et al. Multi-objective artificial bee colony algorithm for simultaneous sequencing and balancing of mixed model assembly line. *Int J Adv Manuf Tech* 2014; 75: 1809–1827.
31. Mosadegh H, Zandieh M and Ghomi SF. Simultaneous solving of balancing and sequencing problems with station-dependent assembly times for mixed-model assembly lines. *Appl Soft Comput* 2012; 12: 1359–1370.
32. Montgomery DC. *Design and analysis of experiments*. Hoboken, NJ: John Wiley & Sons, 2008.

Appendix 1

The procedure of improved C-PSO.

Algorithm The C-PSO algorithm
Input: RTALB data, C-PSO parameters,
Output: Best solution so far
Begin:
Initial the two sub-swarms for task permutation,$\{\mu_1, \mu_2, \ldots, \mu_{PS}\}$, and robot assignment $\{\lambda_1, \lambda_2, \ldots, \lambda_{PS}\}$
Select the best individual as the global best solution $(\mu_{GBest}, \lambda_{GBest})$ by testing (μ_p, λ_p) $\forall p \in \{1,2,\ldots,PS\}$
Execute local search on the global best solution
While (Termination criterion is not met) **do**
//Update task permutation
Obtain population with global best robot assignment λ_{Best} and their own task permutation μ_p
Calculate the fitness and determine the local best solutions
For $i = 1,2,\ldots,PS$ **do**
Obtain new task permutation μ_p'
Decode with μ_p' and λ_{GBest}
Apply the greedy acceptance
//Update robot assignment
Obtain population with global best task permutation μ_{GBest} and their own robot assignment λ_p
Calculate the fitness and determine the local best solutions
For $i = 1,2,\ldots,PS$ **do**
Obtain new robot assignment λ_p'
Decode with μ_{GBest} and λ_p'
Apply the greedy acceptance
Update global best solution if necessary and execute local search on the new global best solution
//Modification of global best
Execute modification of global best if the global best has not been improved in θ' iterations
//Restart mechanism
Execute restart mechanism and local search if the global best has not been improved in ϕ' iterations