**Model and metaheuristics for robotic two-sided assembly line balancing problems with setup times**

Zixiang Li[1,2], Mukund Nilakantan Janardhanan[3*], Qiuhua Tang[1,2], S. G. Ponnambalam[4]

[1]Key Laboratory of Metallurgical Equipment and Control Technology, Wuhan University of Science and Technology, Wuhan, Hubei, China

[2] Hubei Key Laboratory of Mechanical Transmission and Manufacturing Engineering, Wuhan University of Science and Technology, Wuhan, Hubei, China

E-mail: zixiangliwust@gmail.com, tangqiuhua@wust.edu.cn

[3]Mechanics of Materials Group, Department of Engineering, University of Leicester, Leicester, United Kingdom

Email: mukund.janardhanan@leicester.ac.uk

[4]Faculty of Manufacturing Engineering, University Malaysia Pahang, Malaysia

Email: sgponnambalam@ump.edu.my

*Corresponding author

**Abstract:** Two-sided robotic assembly lines are employed to assemble large-sized high-volume products, where robots are allocated to the workstations to perform the tasks and human workers are replaced for achieving lower cost and greater flexibility in production. In the two-sided robotic assembly lines, setup times are unavoidable and it has been ignored in most of the reported works. There has been limited attention on this till date. This paper focusses on the robotic two-sided assembly line with consideration of sequence-dependent setup times and robot setup times. A new mixed integer linear programming model is developed with the objective of optimizing the cycle time. Due to the NP-hard nature of the considered problem, this paper proposes a set of metaheuristics to solve this considered problem, where two main scenarios with low and high setup time's variability are considered. Computational results verify that this new model is capable to achieve the optimal solutions for small-size instances whereas the simple adoption of the published mathematical model might produce wrong solutions for the considered problem. A comprehensive study with 13 algorithms demonstrates that the two variants of artificial bee colony algorithm and migrating bird optimization algorithm are capable to achieve the optimality for small-size instances and to obtain promising results for large-size instances.

**Keywords:** Assembly line balancing; Robotic two-sided assembly line; Setup times; Integer programming; Metaheuristic

## 1. Introduction

Assembly line balancing problem has been classified as a well-known combinatorial optimization problem [1, 2], which has great applications in modern day industries for assembling different kinds of products. The basic model of the assembly line balancing problem is the simple assembly line balancing problem, where a set of tasks are allocated to a minimum number of workstations with the fulfillment of cycle time constraint and precedence constraint. Cycle time constraint ensures the total operation time in one workstation is less than a give cycle time; the precedence constraint ensures the predecessors of one task are allocated to the former workstation or the former position in the same workstation. The simple assembly line balancing problem might be criticized to be too theoretical, and hence the variants of the assembly line balancing

problem are studied extensively due to the diverse application in real world.

Among the variants in the layouts of assembly lines, two-sided assembly lines are widely utilized to assemble large-size large-volume products [3-5], such as cars, trucks and motorcycles and etc. In this type of assembly line, two operators operate the tasks in parallel on the two facing workstations, referred as one mated-station. Modern assembly lines are fitted with robots by replacing human workers to operate the assembly tasks due to their higher flexibility and reduced cost [6]. These type of assembly lines are referred as robotic assembly line and one of the popular problems researched is robotic assembly line balancing problem (RALBP). RALBP mainly deals with both the general task assignment, the robot selection and allocation. Two-sided robotic assembly line are also extensively used by industries and robotic two-sided assembly line balancing problem (RTALBP) inherits the main features of two-sided assembly line balancing and RALBP, and it involves two sub-problems: task assignment and robot allocation.

In the literature, research classify RALBP into two main types: RALBP-I to minimize the workstation number and RALBP-II to minimize the cycle time. The following paragraph presents the most relevant literature reported on the two types of RALB problems.

Rubinovitz and Bukchin [7] presented the first study on RALBP-I, and later Rubinovitz et al. [8] employed a branch and bound method. More heuristics, exact methods and metaheuristic methods are being utilized to tackle RALBPs. Levitin et al. [9] utilized the genetic algorithm and Gao et al. [6] presented an improved genetic algorithm with local search. Yoosefelahi et al. [10] studied the multi-objective RALBP-II using three multi-objective evolution strategies. Daoud et al. [11] employed several hybrid methods to maximize the line efficiency and to balance the different tasks. Çil et al. [12] presented a goal programming technique to solve multi-objective RALBP-II using a case study. Borba et al. [13] presented an iterative beam search heuristic and a branch, bound and remember algorithm to solve RALBP-II. There are more contributions reported where different objectives are studied. Mukund Nilakantan et al. [14] presented the first study to minimize energy consumption using a particle swarm optimization. Nilakantan et al. [15] studied the carbon footprint and presented a multi-objective co-operative co-evolutionary algorithm to minimize carbon footprint and maximum line efficiency. Nilakantan et al. [16] studied the cost-based RALBP-II with the objective of minimizing assembly line cost and cycle time utilizing differential evolution algorithm. Pereira et al. [17] investigated the cost-oriented RALBP using a memetic algorithm. Due to the diverse industrial environments, many researches are presented to study the RALBP with different assembly line layouts or multiple products. Çil et al. [18] studied the parallel RALBP-II with a beam search approach. Regarding the literature on mixed-model RALBP, Çil et al. [19] tackled the mixed-model using a beam search method to optimize the cycle time. Rabbani et al. [20] presented two multi-objective algorithms to solve multi-objective mixed-model RALB-II problem. Li et al. [21] formulated the simultaneous balancing and sequencing robotic mixed-model assembly lines, and they also presented two algorithms to tackle large-size instances. Following paragraphs will present the studies reported on two-sided assembly

balancing problem (TALBP) and related problems. The applied methods include heuristic methods [3, 22], exact methods [23-25] and metaheuristic methods [26-32], to cite just a few. Detailed review of these problems and methods are presented in Zhang et al. [33], Li et al. [4] and Abdullah Make et al. [5].

However, the studies on RTALBP are limited. Li et al. [34] formulated the RTALBP-II and employed a co-evolutionary particle swarm optimization algorithm. Li et al. [35] later developed two discrete cuckoo search algorithms, which outperform the co-evolutionary particle swarm optimization algorithm. Li et al. [36] studied the energy consumption in two-sided RALBP-II, and employ a multi-objective simulated annealing algorithm to minimize energy consumption and cycle time simultaneously. Aghajani et al. [37] tackle the two-sided mixed-model RALBP with setup time with a mathematical formulation and a simulated annealing algorithm.

In most reported studies, the setup times are generally ignored or assumed that it is included into the task operation time. Nevertheless, in some occasions the setup times cannot be ignored and need to be studied separately [38] especially in today's flexible manufacturing systems. In general, the setup times are divided into two types: sequence-independent setup time and sequence-dependent setup time [39]. Sequence-independent setup time depends on the current task, whereas sequence-dependent setup time depends on the preceding task and the current task. In real world applications, it is very necessary to consider sequence-dependent setup times between tasks in a two-sided robotic assembly line. To perform a task directly before another task may influence the latter task inside the station because some setup may be required for performing the latter task. If a task is assigned next to another task in a same station, it may require a setup time. If such a setup is required, it must be calculated to find the finishing time of that task. Additionally, if a task is assigned as the last one to a station, then it may require a setup to complete the first task assigned to that station since the tasks are performed cyclically [40]. Following paragraph presents the most relevant literature reported on setup times with regards to assembly lines.

Andrés et al. [38] presented the first study on sequence-dependent setup times. Scholl et al. [41] study the different situation of setup times, where sequence-dependent task time increments are investigated. Afterwards, Yolmeh and Kianfar [42] studied the assembly line balancing and scheduling problems with setup times utilizing a hybrid genetic algorithm. Scholl et al. [43] extended the model in Andrés et al. [38] with a new model formulation and present a set of heuristic methods to tackle this problem. Özcan and Toklu [40] studied the sequence-dependent setup times in two-sided assembly line. Seyed-Alagheband et al. [44] presented a simulated annealing algorithm to solve the assembly line balancing and scheduling problems with setup times to minimize the cycle time. Hamta et al. [45] studied the multi-objective assembly line balancing problem with flexible operation times, sequence-dependent setup times and learning effect, and they implemented hybrid particle swarm optimization to tackle this problem. Akpinar and Baykasoğlu [46] investigated the mixed model assembly line balancing problems with setup times utilizing multiple colony bee algorithms. Another recent study is carried out by Janardhanan et al. [47], where RALBP-II with setup times are investigated. They formulated the problem as a mixed integer programming model and

presented a set of metaheuristics to tackle the large-size instances for a straight robotic assembly line.

From the aforementioned literature review, it can be concluded that there is only one related paper considering the setup times in RTALBP by Aghajani et al. [37] and there is only one metaheuristic, simulated annealing algorithm in Aghajani et al. [37], is developed to solve the RTALBP with setup times. However, the model presented by Aghajani et al. [37] has several drawbacks in dealing with the setup times and might achieve possibly wrong solutions (see Section 5.2). Hence, it is necessary to develop new and accurate mathematical model to deal with the setup times in two-sided robotic assembly line. It is important to utilize new metaheuristic algorithms to solve the considered problem and identify the most effective ones as industries require efficient schedules in a reasonable computation time Hence, this study presents two major contributions to the literature as follows. 1) Firstly, a new corrected mathematical model is developed to formulate the RTALBP with setup times, and an illustrated example is used to demonstrate that this model could obtain the correct solution. This model might be regarded as the first correct model to for RTALBP with setup times. 2) This type problem has a high-combinatorial structure that makes it difficult to obtain an optimal solution when the problem size increases. A set of 13 high-performing algorithms are extended to solve the considered problem. Due to the consideration of the setup times and the optimization of two sub-problems, all the algorithms utilize new encoding scheme and decoding procedure. This is for the first time 12 of these 13 algorithms are applied to solve the RTALBP with setup times. A comprehensive experimental study on two sets of newly generated instances is carried out to test the performance of these implemented algorithms and identify the best-performing ones. The comparative study demonstrates that migrating bird optimization algorithm and two variants of artificial bee colony algorithm are the most effective methods.

The remainder of this paper is organized as follows. Section 2 introduces the problem description and mathematical formulation in detail. Section 3 presents the proposed metaheuristic methodologies, along with detailed encoding and coding. An illustrated example is provided in Section 4 to highlight the main features of the RTALBP with setup times. Later, Section 5 presents the computational study to test the developed model and the implemented algorithms. Finally, Section 6 concludes this research and gives several future research directions.

## 2. Mathematical model

Among the reported works as presented in the previous section, there is only one paper presenting the mathematical model for mixed-model RTALBP with setup times [37]. However, this model has several drawbacks in dealing with the setup times as presented in Section 5.2, this model might lead to achieving wrong solutions. Hence, this paper formulates a new mathematical model based on Li et al. [34] and Janardhanan et al. [47]. The problem assumptions are presented in section 2.1 and the detailed mathematical model proposed is presented in section 2.2.

## 2.1 Problem description

This research tackles the RTALBP with setup times with the objective of minimizing the cycle time on the basis of Andrés et al. [38], Özcan and Toklu [40], Janardhanan et al. [47] and Aghajani et al. [37]. Study by Scholl et al. [43] divides the setup times into two categories: forward setup times and backward setup times. Specifically, forward setup occurs when task $j$ is operated immediately after task $i$ in the same cycle; backward setup occurs when task $\underline{i}$ is the last one operated at the work piece of a cycle $p$ and the worker has to move to the next work piece which is to be assembled in cycle $p$+1. This situation does not apply to robotic assembly line where positions of robots and products are usually fixed when executing the tasks observed in industry[47]. The assumptions of the considered RTALBP with setup times are presented as follows.

1) Only a single type of product is assembled in the robotic two-sided assembly line.
2) The operation times of tasks and robot setup times of tasks depend on the type of the robot processing them, and they are known and deterministic.
3) The sequence-dependent setup times are considered following Janardhanan et al. [47] and Aghajani et al. [37], where only forward setup times in Scholl et al. [43] are considered. The sequence-dependent setup time between two tasks depends on the type of the robot processing them, and it is known and deterministic.
4) Each robot is allocated to one workstation and the number of the available robots is equal to the number of workstations.
5) The times for material handling, loading and unloading are negligible or included into the operation times of tasks.
6) Work-in-process inventory and parallel workstation are not considered.

The considered RTALBP with setup times can be described as allocating a set of $Nt$ tasks to a set of $Nj$ mated-stations, each equipped with one robot, with the objective of minimizing the cycle time while satisfying cycle time constraint, direction constraint and precedence constraint. Cycle time demands that the all the completion times of tasks are less than or equal to the cycle time, where the robot setup times of tasks and sequence-dependent setup times are considered. In direction constraint, the tasks are portioned into three types: L-type tasks that must be allocated to the left side, R-type tasks that must be allocated to the right side, and E-type tasks that can be allocated to both side. Precedence constraint demands the predecessors of one task are allocated to the former mated-station (workstation) or the former position in the same mated-station. Figure 1 illustrates on layout of RTALBP with 9 tasks and 4 workstations. One mated-station comprises of two facing workstations, and mated-stations are connected by a transportation system, such as belt conveyor. Each workstation is equipped with one robot to operate the assigned tasks, and the tasks on latter mated-station are operated only when the tasks on the former mated-stations are completed.

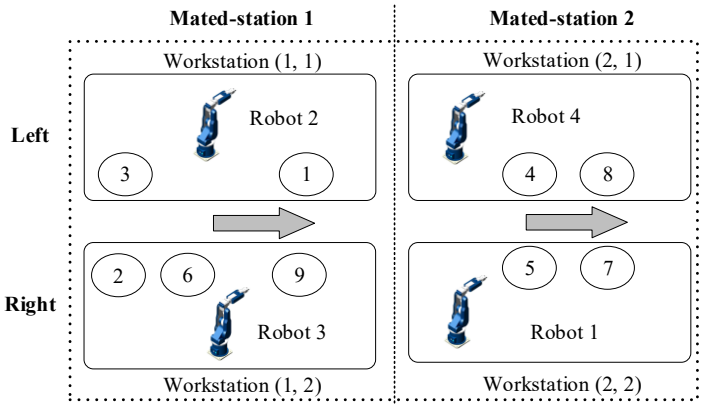


**Figure 1** Layout of robotic two-sided assembly line

Figure 2 depicts the detailed task assignment with setup times on the mated-station 2 in Fig. 1, where the $s$ refers to the robot setup time and $se$ refers to the sequence-dependent setup time. Regarding task 8, there are two setup times before the operation of task 8: sequence-dependent setup time between task 4 and task 8 by robot 4 and robot setup time of task 8 by robot 4. For the task 4 in the first position, the robot setup time of task lies before the operation of task 4, and the sequence-dependent setup time between task 8 and task 4 lies at the rear of this workstation. Due to the utilization of two sides and precedence relations, task 7 cannot be operated before task 4, resulting in sequence depended idle time. Sequence depended idle time is the special idle time in RTALBP, and it might be reduced by optimizing the task sequence on each workstation. In short, the RTALBP with setup times needs to deal with the robot allocation, task assignment to workstations and the scheduling of task sequence on each workstation.

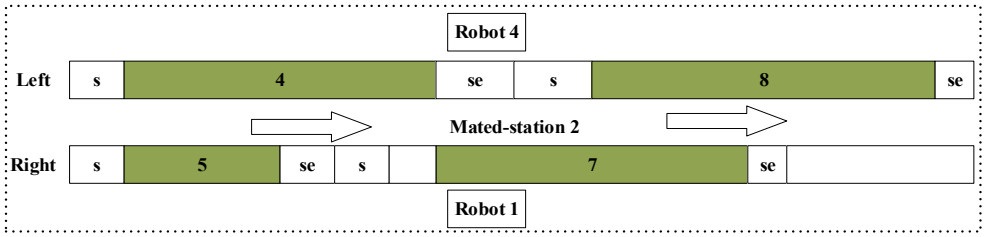


**Figure 2** The detailed assignment of tasks with setup times on one mated-station

## 2.2 Newly developed model

The utilized notations in the mathematical model, decision variables and indicator variable are presented below.

**Indices:**

| | |
|---|---|
| $i, h, p$ | Task index. |
| $j, g$ | Mated-station index |
| $k, f$ | Side of the line; $k, f = \begin{cases} 1 & \text{if the side is left} \\ 2 & \text{if the side is right} \end{cases}$. |
| $(j, k)$ | The $k$ side workstation of the mated-station $j$. |
| $s$ | A position inside the task operation sequence of a workstation $(j, k)$; |
| $r$ | Robot index. |

**Parameters:**

| | |
|---|---|
| $I$ | Set of tasks in the precedence diagram, $I = \{1,2,\dots,i,\dots,Nt\}$. |
| $J$ | Set of mated-stations, $J = \{1,2,\dots,j,\dots,Nj\}$. |
| $W(j,k)$ | Subset of all tasks that can be assigned to workstation $(j,k)$. |
| $\|\|W(j,k)\|\|$ | Number of tasks in subset $W(j,k)$. |
| $W_{max}$ | Maximum number of tasks that can be assigned to a workstation, $W_{max} = \max_{(j,k)}\{\|\|W(j,k)\|\|\}$. |
| $NT(j,k)$ | Set of operation sequence of tasks in a workstation $(j,k)$ ; $NT(j,k) = \{1,2,\cdots,s\cdots,\|\|W(j,k)\|\|\}\ \forall j \in J, k = 1,2$. |
| $R$ | Set of robot types, $R = \{1,2,\dots,r,\dots,Nr\}$. |
| $AL$ | Set of tasks which should be performed on the left-side workstation, $AL \subseteq I$. |
| $AR$ | Set of tasks which should be performed on the right-side workstation, $AR \subseteq I$. |
| $AE$ | Set of tasks which can be performed on left or right side of a mated-station, $AE \subseteq I$. |
| $P_0$ | Set of tasks that have no immediate predecessors. |
| $P(i)$ | Set of immediate predecessors of task $i$. |
| $P_a(i)$ | Set of all predecessors of task $i$. |
| $S(i)$ | Set of immediate successors of task $i$. |
| $S_a(i)$ | Set of all successors of task $i$. |
| $C(i)$ | Set of tasks whose operation directions are opposite to that of task $i$, $C(i) = \begin{cases} AL & \text{if } i \in AR \\ AR & \text{if } i \in AL \\ \emptyset & \text{if } i \in AE \end{cases}$. |
| $K(i)$ | The set of integers which indicate the preferred operation direction of task $i$, $K(i) = \begin{cases} \{1\} & \text{if } i \in AR \\ \{2\} & \text{if } i \in AL \\ \{1,2\} & \text{if } i \in AE \end{cases}$. |
| $\psi$ | A large positive number. |
| $t_{ir}$ | Processing time of task $i$ by robot $r$. |
| $tsu_{ipr}$ | Setup time between two successive task $i$ and $p$ with robot $r$ if $i$ and $p$ are assigned to the same workstation. |
| $tsr_{ir}$ | Setup time of a robot $r$ for processing task $i$. |

**Decision variables:**

| | |
|---|---|
| $CT$ | Cycle time. |
| $x_{ijks}$ | 1, if task $i$ is assigned to workstation $(j,k)$ in position $s$ of its operation sequence; 0, otherwise. |
| $w_{ijk}$ | 1, if task $i$ is the last task in the operation sequence of tasks assigned to workstation $(j,k)$; 0, otherwise. |
| $y_{rjk}$ | 1, if robot $r$ is allocated to workstation $(j,k)$; 0, otherwise. |
| $t_i^f$ | Completion time of task $i$. |

**Indicator variables:**

| | |
|---|---|
| $z_{ipjk}$ | 1, if task $i$ is performed immediately before task $p$ in the same workstation $(j,k)$ in the same cycle or in the next cycle; 0, otherwise. |

The detailed mathematical formulations are developed on the basis of the mathematical model reported in Li et al. [34] and Janardhanan et al. [47] and are presented below.

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

$$\sum_{j\in J}\sum_{k\in K(i)}\sum_{s\in NT(j,k)} x_{ijks} = 1 \quad \forall i \in I \tag{2}$$

$$\sum_{i\in W(j,k)} x_{ijks} \le 1 \quad \forall j \in J, k \in K(i), s \in NT(j,k) \tag{3}$$

$$\sum_{i\in W(j,k)} x_{ijk,s+1} - \sum_{i\in W(j,k)} x_{ijks} \le 0 \quad \forall j \in J, k \in K(i), s \in NT(j,k) \wedge s < \|W(j,k)\| \tag{4}$$

$$\sum_{g\in J}\sum_{k\in K(h)}\sum_{s\in NT(g,k)} (W_{max}\cdot(g-1)+s)\cdot x_{hgks} \le \sum_{j\in J}\sum_{k\in K(i)}\sum_{s\in NT(j,k)} (W_{max}\cdot(j-1)+s)\cdot x_{ijks} \tag{5}$$

$$\forall i \in I - P_0, h \in P(i)$$

$$t_i^f \le CT \quad \forall i \in I \tag{6}$$

$$t_i^f - t_h^f + \psi\left(1 - \sum_{s\in NT(j,k)} x_{ijks}\right) + \psi\left(1 - \sum_{f\in K(h)}\sum_{s\in NT(j,k)} x_{hjfs}\right) \ge \sum_{r=1}^{nr} t_{ir}\cdot y_{rjk} \tag{7}$$

$$, \ \forall i \in I - P_0, h \in P(i), k \in K(i), j \in J$$

$$t_p^f - t_i^f + \psi\left(1 - \sum_{s\in NT(j,k)\wedge s>1} x_{pjks}\right) + \psi\left(1 - \sum_{s\in NT(j,k)\wedge s<\|W(j,k)\|} x_{ijks}\right) + \psi\left(1 - z_{ipjk}\right) \ge \sum_{r=1}^{nr}\left(t_{pr} + tsr_{pr} + tsu_{ipr}\right)\cdot y_{rjk} \tag{8}$$

$$\forall (i,p) | i \ne p \wedge i,p \in W(j,k), j \in J, k \in K(i) \cap K(p)$$

$$t_i^f + \sum_{r=1}^{nr} tsu_{ipr}\cdot y_{rjk} \le CT + \psi\left(1 - x_{pjk1}\right) + \psi\left(1 - w_{ijk}\right) + \psi\left(1 - z_{ipjk}\right) \tag{9}$$

$$\forall (i,p) | i \ne p \wedge i,p \in W(j,k), j \in J, k \in K(i) \cap K(p)$$

$$x_{ijks} + x_{hjk,s+1} \le 1 + z_{ihjk}$$

$$\forall j \in J, k \in K(i) \cap K(h), s \in NT(j,k) \wedge s < \|W(j,k)\|, (i,h) | i \ne h \wedge i,h \in W(j,k) \wedge h \notin P(i) \tag{10}$$

$$x_{ijks} - \sum_{\forall h\in W(j,k) | i\ne h \wedge h\notin P(i)} x_{hjk,s+1} \le w_{ijk} \tag{11}$$

$$\forall j \in J, k \in K(i), i \in W(j,k), s \in NT(j,k) \wedge s < \|W(j,k)\|$$

$$w_{ijk} + x_{hjk1} \leq 1 + z_{ihjk}$$
$$\forall j \in J, k \in K(i) \cap K(h), (i,h) | i \neq h \wedge i,h \in W(j,k) \wedge i \notin P(h) \tag{12}$$

$$t_i^f + \psi(1 - x_{ijks}) \geq \sum_{r=1}^{nr} (t_{ir} + tsr_{ir}) \cdot y_{rjk} \tag{13}$$
$$\forall i \in I, s \leq \|W(j,k)\|, j \in J, k \in K(i)$$

$$\sum_{r=1}^{nr} y_{rjk} = 1 \quad \forall j \in J, k = 1,2 \tag{14}$$

$$\sum_{j=1}^{nj} \sum_{k=1}^{2} y_{rjk} = 1 \quad \forall r \in R \tag{15}$$

The objective in expression (1) minimizes the cycle time. Constraint (2) deals with the precedence constraint and the task assignment, indicating that a task must be assigned to only one position inside one workstation. Constraint (3) means that there should be no more than one task in each position inside one workstation, and constraint (4) denotes that the tasks should be assigned in increasing positions inside one workstation. Constraint (5) handles precedence constraint, ensuring that the predecessor $h$ of task $i$ should be allocated to the former mated-station or the former position inside the same workstation. Constraint (6) is the cycle time constraint, ensuring that all the tasks are finished within the given cycle time. Constraint (7) ensures that the successor $i$ of task $h$ cannot be operated until task $h$ is completed. Constraint (8) deals with the task assignment inside one workstation, and it is reduced to $t_p^f - t_i^f \geq \sum_{r=1}^{nr}(t_{pr} + tsr_{pr} + tsu_{ipr}) \cdot y_{rjk}$ when task $i$ is performed immediately before task $p$ in the same workstation. Constraint (9) deals with the setup time between the last task and the first task inside one workstation, indicating that the cycle time is larger than or equal to the completion time of the last task and the setup time between the last task and the first task inside one workstation. Constraints (10-12) calculates the values of $z_{ipjk}$ and $w_{ijk}$. Specifically, constraint (10) ensures that $z_{ipjk}$ is equal to 1 when task $i$ is performed immediately before task $p$ in the same workstation. Constraint (11) indicates the $w_{ijk}$ is equal to 1 when task $i$ is the last task in the operation sequence of tasks assigned to one workstation. And constraint (12) ensures that $z_{ihjk}$ is equal to 1 when task $i$ is the last task and task $h$ is the first task inside one workstation. Constraint (13) indicates that the completion time of task $i$ is larger than its operation time and setup time, and it is reduced to $t_i^f \geq \sum_{r=1}^{nr}(t_{ir} + tsr_{ir}) \cdot y_{rjk}$ when $x_{ijks} = 0$. Constraint (14) and constraint (15) deal with the robot allocation, ensuring a workstation is allocated with one robot and a robot must be allocated to only one workstation.

The optimization of CT in expression (1) is one of the most important and practical

objectives in real applications and researches, and it helps the smooth the assembly line and increase the line efficiency. In real applications, there are multiple conflicting objectives, including balancing and trading-off the resources, and machine/robot loads. As this study aims at testing the model utilizing the standard general solver and there is no general solver that can solve the RTALBP with multiple conflicting objective effectively, the multiple conflicting objectives are not considered in this study. However, these realistic objectives can be achieved by extending the developed model. It is to be noted that this developed model is not the simple adaption of the model in mixed-model RALBP in Aghajani et al. [37]. The published model has several drawbacks in dealing with the setup times, and the proposed model on the contrary has remedied these possible drawbacks. Specifically, the new model proposed new indicator variables $z_{ipjk}$ and takes the sequence-dependent setup time between the last task and the first task inside one workstation into account. This new model and the model by Aghajani et al. [37], which is presented in Appendix A are compared in Section 5.2 in detail.

## 3. Proposed metaheuristic methodologies

There are many algorithms in solving kinds of combinatorial optimization problems, whereas most of these algorithms cannot be applied to solve the considered problem due to the special characteristics of the considered problem. There are three main characteristics for the considered problem: A) The considered problem is discrete optimization problem; B) there are two sub-problems needed to be optimized simultaneously; C) two vectors as presented in Section 3.1 are necessary for decoding to determine the task permutation vector and robot allocation respectively. And the published algorithms in solving other optimization problems might not produce promising results due to the discrete attribute and other special features of the RTALBP observed in the preliminary experiments. For instance, most of the reported algorithms applied to solve the general two-sided assembly line balancing problems [4] cannot be applied to solve RTALBP directly and need major modifications. Hence, this study mainly selects the algorithms applied to solve RALBP or RTALBP. A set of 13 algorithms are implemented and modified to solve this new RTALBP with setup times. The proposed encoding and decoding are presented at first and all the algorithms utilize the same encoding and decoding. Later, this section provides the detailed descriptions of the tested algorithms. They are categorized into three types: local search algorithm, swarm intelligence algorithm (original edition) and co-evolutionary swarm intelligence algorithm (co-evolutionary edition). For the algorithms in each category, only one or two high-performing algorithms are described in detail.

### 3.1 Solution representation

This research utilizes task permutation vector and robot allocation vector for encoding based on the ones reported in Li et al. [34] and Janardhanan et al. [47]. Task permutation vector determines the task assignment; robot allocation decides the robot allocation. An example with 9 tasks and 4 robots is illustrated in Figure 3. In task permutation vector, the tasks in former positions have higher priority and are assigned at first, e.g. task 2 is

assigned at first. In robot allocation vector, the robots are allocated to the workstations in sequence, e.g. the first robot 2 is allocated to the first workstation (1, 1).

| Task permutation | 3 | 2 | 1 | 6 | 9 | 4 | 5 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|---|
| Robot allocation | 2 | 3 | 4 | 1 | | | | | |

**Figure 3** Example of encoding scheme

To transfer the encoding to a feasible solution, a decoding procedure and an initial cycle time are needed. The decoding procedure is described as follows. In Step 3, one task is assignable when direction constraint, precedence constraint and cycle time constraint are satisfied when the current mated-station is not the last mated-station. The cycle time constraint, on the contrary, is allowed to be violated when the current mated-station is the last mated-station in order to obtain the complete infeasible solution. And the maximum value of the completions times of tasks on all the mated-station is regarded as the achieved cycle time. The detailed decoding procedure is presented in Appendix B and a detailed procedure to transfer the encoding vectors into a feasible solution when solving the small-size instance in Section 4 is also presented in Appendix C.

**Algorithm 1:** Decoding procedure for RTALBP with setup times

**Input:** Task permutation, robot allocation and initial cycle time

**Step 1:** Open a new mated-station.

**Step 2:** Decide the allocated robots to two sides of the current mated-station.

**Step 3:** Select the assignable tasks for both sides fulfilling direction constraint, precedence constraint and cycle time constraint (except for the tasks on the last mated-station).

% Regarding the cycle time constraint, the constraint (9) in section 2.2 must be satisfied: For any task $i$, the sum of the completion time of task $i$ and the setup time between task $i$ and the first task on the same workstation is less than the given cycle time ($t_i^f + \sum_{r=1}^{nr} tsu_{ipr} \cdot y_{rjk} \leq CT$).

**Step 4:** **If** no assignable task exists **and** there remains some unassigned tasks

Execute Step 1;

**Else**

Execute Step 5;

**Step 5:** Determines the side to allocate the tasks.

//The side with larger capacity is selected or the left side is selected by default when both sides have the same capacities.

**Step 6:** Select one task from assignable task set and assign it to the selected side.

//The task with the minimum sequence depended idle time and in the former position is selected.

**Step 7:** Update the remaining capacities of the two sides.

**Step 8:** **If** all tasks have been assigned

Terminate the decoding procedure.

**Else**

Go to Step 3.

**Output:** The achieved cycle time and detailed task assignment and robot allocation

Determining the proper initial cycle time is another important problem and an iterative mechanism in Li et al. [4] and Janardhanan et al. [47] is utilized to update the cycle time. The iterative mechanism is embedded into the evolution process of algorithms and it is explained in Algorithm 2. This iterative mechanism has two main advantages. Firstly, the iterative mechanism is fast, as each individual only need to execute one decoding procedure (or several times if new best cycle time is found). It avoids the problem of executing the decoding procedure for many times to find the proper cycle time. Secondly, all the individuals are evaluated using the same initial cycle time and thus minor improvements are preserved.

---

**Algorithm 2:** Iterative mechanism for cycle time update

**Step 1:** Set the initial cycle time *CT* and best cycle time $CT_{Best}$ as a very large positive number.
% The initial cycle time *CT* is set as $CT = 2 \cdot \sum_{i \in I} \sum_{r \in R} t_{ir} / (Nr \cdot 2 \cdot Nj)$ where the utilized indices and parameters are defined in Section 2.1, $Nr$ is the number of utilized robots and $Nj$ is the number of mated-stations.

**Step 2:** This individual is decoded with *CT* as the initial cycle time using Algorithm 1.
Update $CT_{Best}$ if the smaller cycle time is obtained and update $CT$ using $CT = CT_{Best} - 1$.
All individuals are decoded with *CT* as the initial cycle time using Algorithm 1.

**Step 3:** **If** smaller best cycle time is found
Update $CT_{Best}$ and update $CT$ using $CT = CT_{Best} - 1$;
All the individuals are re-decoded using CT as the initial cycle time and update $CT_{Best}$ when necessary.
**Endif**

**Step 4:** Go to Step 3 if the termination criterion is not met; or terminate and output the best cycle time $CT_{Best}$, otherwise.

---

### 3.2 Local search algorithms

This paper tests three local search algorithms such as simulated annealing (SA) algorithm, restarted simulated annealing (RSA) algorithm [21] and late acceptance hill-climbing (LAHC) algorithm [30]. The main procedure of SA is presented in Algorithm 3. The main property of SA is accepting a worse solution with a certain probability or SA is accepting a worse solution when $\text{Rand} \leq exp^{-\triangle/(T \times Fit(S))}$ where Rand is a random number within [0, 1], $T$ is the temperature, $Fit(S)$ is the objective value for solution $S$ and $\triangle = \text{Fit}(S') - \text{Fit}(S)$ ( $S'$ is the neighbor solution of the current solution $S$). Clearly, the probability decreases with the algorithm iterating. RSA firstly utilized by Nilakantan et al. [48] is an improved edition of SA algorithm to enhance exploration and avoid being trapped into local optima. RSA replaces the current temperature *T* with a restart temperature $T_R$ when no improvement on the best cycle is achieved for a number of consecutive times.

**Algorithm 3:** Procedure of SA algorithm for RTALBP

Set $T$:=$T_0$ and obtain an initial solution $S$;

**While (**Termination criterion is not satisfied**)**

  **For** $n$:=0 to $N$ **do**

    Obtain a neighbor solution $S'$;

    Calculate $\triangle = \mathrm{Fit}(S') - \mathrm{Fit}(S)$;

    % $\mathrm{Fit}(S)$ is the fitness of solution $S$.

    **If** $(\triangle \leq 0)$ $S \leftarrow S'$;

    **Else If** (Rand $\leq exp^{-\triangle/(T \times Fit(S))}$) $S \leftarrow S'$;

    % Rand is a randomly generated number within [0,1]

  **Endfor**

  $T = T \times \alpha$

**Endwhile**

%Parameter $T_0$ is the initial temperature; parameter $\alpha$ is the cooling rate; parameter $N$ is the iteration time before the temperature updates.

### 3.3 Swarm intelligence algorithms

This paper implements six swarm intelligence algorithms: particle swarm optimization (PSO) algorithm [34], genetic algorithm (GA) [47], discrete cuckoo search (DCS) algorithm[35], bees algorithm (BA) [46], two artificial bee colony (ABC) algorithms[49], referred to as ABC1 and ABC2, and migrating bird optimization algorithm (MBO) [47]. This section mainly provides the detailed description of artificial bee colony algorithms and MBO algorithm for their superiority and the detailed applications of other methods (GA, PSO and DCS) is available in Janardhanan et al. [47].

The main procedure of ABC algorithm (ABC1) is presented in Algorithm 4. ABC1 starts with initializing the swarm, and later the main loop consisting of employed bee phase, onlooker phase and scout phase is iteratively repeated until a termination criterion is met. In employed bee phase, a neighbor solution is generated for each individual and greedy acceptance is applied. In onlooker phase, an individual is selected using roulette wheel selection and subsequently neighbor solution is generated, and later greedy acceptance is applied again. Scout phase is utilized to emphasize exploration and it replaces a duplicated solution or the solution with the worst fitness with a neighbor solution of a randomly selected solution from the remaining individuals. The applied scout phase has a large probability in achieving high-quality solution, and it outperforms the method of replacing the selected solution with a randomly generated solution by a significant margin in preliminary experiments. As the onlooker phase is quite complex in selecting one individual using roulette wheel selection, ABC2 utilizes a binary tournament selection method to select a solution. Specifically, two solutions are randomly selected and the better one is selected. It can be noted that ABC2 is much easier in implementation.

**Algorithm 4:** Procedure of ABC algorithm for RTALBP

Initialize the *PS* food sources;

**While (**Termination criterion is not satisfied**)**

  **For** *p*:=1 to *pop* **do** **% Employed bee phase**

    Generate a neighbor solution $p'$ of the individual *p*;

    Replace *p* with $p'$when $\mathrm{Fit}(p') \leq \mathrm{Fit}(p)$;

  **Endfor**

  **For** *p*:=1 to *pop* **do** **% Onlooker phase**

    Obtain the probability value for each individual;

    Select one individual *S* using roulette wheel selection scheme;

    Generate a neighbor solution $S'$ of the individual *S*;

    Replace *S* with $S'$when $\mathrm{Fit}(S') \leq \mathrm{Fit}(S)$

  **Endfor**

  **% Scout phase**

  Select one duplicated solution or the solution with the worst fitness;

  Replace selected individual with the neighbor solution of a randomly selected solution from the remained individuals;

**Endwhile**

**%**Parameter *PS* is population size; $\mathrm{Fit}(p)$ is the fitness of solution *p*.

The main procedure of MBO algorithm is illustrated in Algorithm 5. MBO starts with population initialization, and later the main cycle comprising leader improvement, block improvement and leader replacement is iteratively repeated until a termination criterion is met [50]. Within the cycle, leader improvement attempts to improve the leader by generating $k$ neighbor individuals. Afterward, block improvement tries to improve the remaining individuals by evaluating $x$ unused best neighbor solutions of that in the front and its ($k$-$x$) neighbor individuals. After executing leader improvement and block improvement for consecutively $m$ times, the leader replacement is employed to move the leader individual to the end and forward the immediate following individuals. The main feature of the MBO is the benefit mechanism which shares the neighbor solutions with the individual in the back. The benefit mechanism promotes evolution of the whole population and replaces the poor individuals with the high-quality neighbor solutions of other individuals very fast. To avoid being premature or trapped into local optima, MBO utilizes a simulated annealing-like acceptance criterion when the current best cycle time remained unchanged for a number of iterations. The incumbent individual *S* is replaced with the new neighbor solution $S'$ when $\mathrm{Fit}(S') \leq \mathrm{Fit}(S)$ or with a probability of $exp^{-(\mathrm{Fit}(S\prime)-\mathrm{Fit}(S))/(T\times Fit(S))}$, where $\mathrm{Fit}(S)$ is the fitness of solution *S* and *T* is the temperature. The value of *T* is updated using $T = T \times \alpha$ after each iteration, where $\alpha$ is the cooling rate. If the improvement on the current best cycle time is obtained, the original greedy acceptance is again applied. In summary, this new acceptance criterion enhances the exploration capacity of the MBO algorithm by accepting poor individual, and hence helps the algorithm to escape from being trapped into local optima.

**Algorithm 5:** Procedure of MBO algorithm for RTALBP

Initialize the individuals;

**While (**Termination criterion is not satisfied**)**

**For** *i*:=1 to *m* **do**

**For** *j*:=1 to *k* **do** **%Leader improvement**

Generate a neighbor solution $S'$ of the leader *S*;

Replace *S* with $S'$when $\mathrm{Fit}(S') \leq \mathrm{Fit}(S)$;

**Endfor**

Replace *S* with the best individual $S''$ from *k* neighbor solutions $S'$ when $\mathrm{Fit}(S'') \leq \mathrm{Fit}(S)$;

Set a large value as the $\mathrm{Fit}(S')$ when $\mathrm{Fit}(S')$ is equal to the $\mathrm{Fit}(S)$ for each neighbor solution $S'$;

**For** each individual on left and right sides **%Block improvement**

**For** *j*:=1 to (*k*-*x*) **do**

Generate a neighbor solution $p'$ of this individual *p*;

Replace *p* with $p'$when $\mathrm{Fit}(p') \leq \mathrm{Fit}(p)$;

**Endfor**

Replace *p* with the best individual $p''$from the (*k*-*x*) neighbor solutions $p'$ and *x* unused best neighbor solution of the individual in the front when $\mathrm{Fit}(p'') \leq \mathrm{Fit}(p)$;

Set a large value as the $\mathrm{Fit}(p')$ when $\mathrm{Fit}(p')$ is equal to the $\mathrm{Fit}(p)$ for each neighbor solution $p'$;

**Endfor**

**Endfor**

Move the leader individual to the end; **%Leader replacement**

Forward the immediately following individuals;

**Endwhile**

%Parameter *m* denotes that leader improvement and block improvement are executed consecutively *m* times before conducting the leader replacement procedure; parameter *k* is the number of neighbor solutions for the leader individual; parameter *x* is the number of shared neighbors of the individual in the front.

## 3.4 Co-evolutionary swarm intelligence algorithms

This research implements three co-evolutionary swarm intelligence algorithm: co-evolutionary particle swarm optimization (CoPSO) [34], co-evolutionary cuckoo search (CoCS) [35] and co-evolutionary genetic algorithm (CoGA) [21]. This section mainly presents the procedure of the CoGA algorithm due to its superiority, and the other two methods have the similar procedure in co-evolution and detailed description are available in the cited articles. The main procedure of CoGA is illustrated in Algorithm 6. CoGA consists of two sub-swarms: task assignment sub-swarm and robot allocation sub-swarm. CoGA starts with the initialization of two sub-swarms and obtaining the best solution by testing the *i*th individuals from two sub-swarms, where the best solution is utilized to connect the two sub-problemsTask assignment sub-swarm evolution and robot allocation sub-swarm evolution are iteratively repeated until a termination criterion is met. In task assignment sub-swarm, the individuals are

decoded utilizing the task assignment vector in task assignment sub-swarm and the robot allocation vector in the best solution. Similarly, in robot allocation sub-swarm, the individuals are decoded utilizing the robot allocation vector in robot allocation sub-swarm and the task assignment vector in the best solution. To avoid being trapped into local optima, a restart mechanism is employed when the best solution remains unchanged for consecutive *RT* iterations. The restart mechanism performs neighbor operations for two times on the two vectors of the best solutions randomly to obtain a number of solutions (set to 200). The best one among them is selected and later, the incumbent best solution is replaced with the selected new individual. Afterwards, two sub-swarms are reinitialized in a way the incumbent individuals are replaced with the new ones by conducting two times' neighbor operations. This restart mechanism aims at increasing the exploration capacity and help the CoGA method to escape from getting trapped into local optima.

---

**Algorithm 6:** Procedure of CoGA algorithm for RTALBP

Set $i$=0 and initialize two sub-swarms for two sub-problems;

Obtain and select the best solution by testing the $i$th inviduals from two sub-swarms;

**While (**Termination criterion is not satisfied**)**

**%Task assignment sub-swarm evolution**

Decode using the task assignment vectors in task assignment sub-swarm and the other vector from the best solution;

Obtain new sub-swarm utilizing tournament selection, crossover operator and mutation operator;

Update the best solution when better fitness is achieved;

Replace the last individual in the sub-swarm with the corresponding vector in the best solution;

**%End of task assignment sub-swarm evolution**

**%Robot allocation sub-swarm evolution**

Decode using the robot allocation vectors in robot allocation sub-swarm and the other vector from the best solution;

Obtain new sub-swarm utilizing tournament selection, crossover operator and mutation operator;

Update the best solution when better fitness is achieved;

Replace the last individual in the sub-swarm with the corresponding vector in the best solution;

**%End of robot allocation sub-swarm evolution**

**If** (New best solution is obtained) $i$:=0; **Else** $i$:=$i$+1;

**If** ($i \geq$ RT) **% Restart mechanism**

Conduct restart mechanism to update the best solution and the two sub-swarms;

**Endif**

**Endwhile**

% RT denotes that the restart mechanism is conducted when the best solution remained unchanged in $RT$ iterations.

---

## 3.5 Neighbor structures

For all the algorithms, it is necessary to achieve new generation or new solution utilizing neighbor structures. In this study, the insert operator and swap operator are utilized to obtain the new neighbor solution of one incumbent one. And two-point

crossover operator are utilized to combine two incumbent individuals to obtain two new individuals. Figure 4 depicts an example of insert operator and swap operator in task permutation vector and robot allocation vector. For one algorithm, the task permutation vector or robot allocation vector is randomly selected at first, and later the insert operator or swap operator is randomly applied to obtain a new vector. Insert operator and swap operator have been widely applied in literature for different kinds of assembly line balancing problems, and the utilization of two neighbor operators helps in increasing search space and enhances exploration capacity. Figure 5 also depicts an example of utilizing two-point crossover operator for task assignment vectors.

For instance, in SA algorithm presented in Algorithm 3, the neighbor solution $S'$ achieved by executing the insert operator or swap operator on the incumbent solution *S*. For CoGA algorithm in Algorithm 6, the two-point crossover operator is utilized for crossover operator, and insert operator or swap operator is randomly selected as mutation operator. In this paper, all the implemented algorithms share the same neighbor structures for a fair comparison.

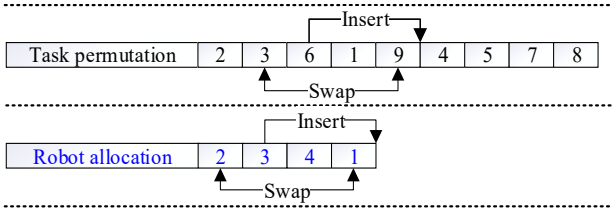


**Figure 4** Insert operator and swap operator for two vectors

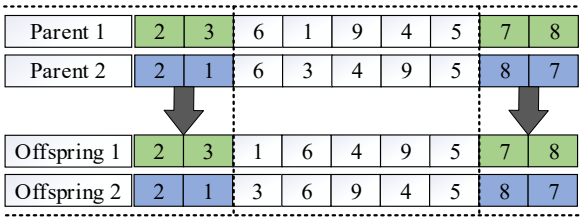


**Figure 5** Two two-point crossover operator for task assignment vectors

## 4. An illustrated example

This section illustrates an example problem with 9 tasks and 2 mated-stations to highlight the main features of the considered problem. Table 1 presents the precedence relations, the preferred directions of tasks and the operation times of tasks by robots. For instance, the preferred direction of task 2 is 'R', denoting that task 2 must be allocated to the right side and the operation times of task 2 by robots are 4, 3, 2 and 4; respectively. Table 2 and Table 3 illustrate the sequence-dependent setup times between tasks and robot setup times before operating a task. In Table 2, the sequence-dependent setup times between tasks depend on the robots, and they are different when tasks are operated by different robots. Namely, the sequence-dependent setup times are determined by the sequence of executing two tasks and the robots to operate them. In Table 3, it is observed that the robot setup times depend on the robots, and different

robots might need different setup times before operating a task. Figure 6 illustrates the optimal robot allocation and task assignment by the model in Section 2, where $s$ refers to the robot setup time and $se$ refers to the sequence-dependent setup time. It is clear that the optimal cycle time is 4.9, and the robot 2, robot 3, robot 4 and robot 1 are allocated to workstation (1,1), workstation (1,2), workstation (2,1) and workstation (2,2) and there are sequence-dependent setup time and robot setup time before operating one task. For instance, there are sequence-dependent setup time and robot setup time before task 1. For the task 3 in the first position of workstation (1,1), the sequence-dependent setup time lies at the rear of this workstation. From the figure, it is observed that all the constraints are satisfied and the model in Section 2 produces the correct solution for this tested instance.

**Table 1** Precedence relations and operation times of tasks

| **Tasks** | **Successors** | **Operation times** | | | | |
|---|---|---|---|---|---|---|
| | | **Preferred direction** | **Robot 1** | **Robot 2** | **Robot 3** | **Robot 4** |
| 1 | 4 | L | 2 | 2 | 2 | 2 |
| 2 | 5, 6 | R | 4 | 3 | 2 | 4 |
| 3 | 6 | E | 2 | 2 | 2 | 2 |
| 4 | 7 | L | 3 | 3 | 4 | 2 |
| 5 | 7, 8 | R | 1 | 1 | 1 | 1 |
| 6 | 9 | E | 1 | 1 | 1 | 1 |
| 7 | | E | 2 | 2 | 2 | 2 |
| 8 | | L | 2 | 2 | 2 | 2 |
| 9 | | E | 1 | 1 | 1 | 1 |

**Table 2** Sequence-dependent setup times

| | **Robot 1** | | | | | | | | | **Robot 2** | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| 1 | 0 | 0.1 | 0.3 | 0.3 | 0.1 | 0.2 | 0.3 | 0.3 | 0.2 | 0 | 0.3 | 0.2 | 0.3 | 0.2 | 0.1 | 0.2 | 0.3 | 0.1 |
| 2 | 0.2 | 0 | 0.2 | 0.2 | 0.2 | 0.3 | 0.3 | 0.1 | 0.2 | 0.2 | 0 | 0.2 | 0.1 | 0.1 | 0.1 | 0.2 | 0.3 | 0.3 |
| 3 | 0.3 | 0.1 | 0 | 0.3 | 0.2 | 0.2 | 0.2 | 0.3 | 0.2 | 0.2 | 0.2 | 0 | 0.3 | 0.3 | 0.3 | 0.3 | 0.3 | 0.3 |
| 4 | 0.2 | 0.3 | 0.3 | 0 | 0.3 | 0.3 | 0.1 | 0.1 | 0.3 | 0.2 | 0.1 | 0.3 | 0 | 0.1 | 0.3 | 0.2 | 0.2 | 0.1 |
| 5 | 0.2 | 0.1 | 0.1 | 0.2 | 0 | 0.3 | 0.3 | 0.2 | 0.2 | 0.3 | 0.3 | 0.3 | 0.3 | 0 | 0.3 | 0.1 | 0.3 | 0.3 |
| 6 | 0.1 | 0.2 | 0.3 | 0.3 | 0.3 | 0 | 0.2 | 0.1 | 0.2 | 0.3 | 0.1 | 0.3 | 0.1 | 0.2 | 0 | 0.2 | 0.2 | 0.3 |
| 7 | 0.2 | 0.1 | 0.2 | 0.1 | 0.1 | 0.3 | 0 | 0.1 | 0.2 | 0.2 | 0.3 | 0.3 | 0.2 | 0.2 | 0.3 | 0 | 0.2 | 0.1 |
| 8 | 0.3 | 0.2 | 0.2 | 0.3 | 0.1 | 0.3 | 0.2 | 0 | 0.2 | 0.1 | 0.2 | 0.2 | 0.2 | 0.1 | 0.3 | 0.3 | 0 | 0.3 |
| 9 | 0.2 | 0.1 | 0.3 | 0.1 | 0.3 | 0.2 | 0.3 | 0.2 | 0 | 0.2 | 0.2 | 0.3 | 0.2 | 0.1 | 0.2 | 0.3 | 0.3 | 0 |
| | **Robot 3** | | | | | | | | | **Robot 4** | | | | | | | | |
| 1 | 0 | 0.1 | 0.3 | 0.2 | 0.2 | 0.3 | 0.1 | 0.2 | 0.2 | 0 | 0.3 | 0.3 | 0.2 | 0.3 | 0.2 | 0.2 | 0.3 | 0.3 |
| 2 | 0.3 | 0 | 0.3 | 0.2 | 0.3 | 0.1 | 0.1 | 0.3 | 0.1 | 0.3 | 0 | 0.3 | 0.3 | 0.1 | 0.2 | 0.2 | 0.2 | 0.2 |
| 3 | 0.1 | 0.3 | 0 | 0.1 | 0.1 | 0.3 | 0.2 | 0.2 | 0.3 | 0.1 | 0.2 | 0 | 0.2 | 0.3 | 0.2 | 0.2 | 0.2 | 0.3 |
| 4 | 0.1 | 0.1 | 0.1 | 0 | 0.3 | 0.2 | 0.2 | 0.3 | 0.2 | 0.2 | 0.1 | 0.2 | 0 | 0.1 | 0.1 | 0.3 | 0.3 | 0.1 |
| 5 | 0.3 | 0.2 | 0.1 | 0.1 | 0 | 0.3 | 0.2 | 0.3 | 0.2 | 0.3 | 0.1 | 0.3 | 0.2 | 0 | 0.3 | 0.2 | 0.3 | 0.3 |
| 6 | 0.3 | 0.3 | 0.3 | 0.3 | 0.3 | 0 | 0.3 | 0.3 | 0.1 | 0.1 | 0.3 | 0.2 | 0.1 | 0.2 | 0 | 0.2 | 0.3 | 0.3 |
| 7 | 0.2 | 0.3 | 0.1 | 0.3 | 0.1 | 0.2 | 0 | 0.2 | 0.3 | 0.3 | 0.3 | 0.2 | 0.3 | 0.3 | 0.3 | 0 | 0.1 | 0.2 |
| 8 | 0.1 | 0.2 | 0.2 | 0.3 | 0.3 | 0.1 | 0.2 | 0 | 0.3 | 0.2 | 0.2 | 0.3 | 0.1 | 0.2 | 0.2 | 0.2 | 0 | 0.3 |
| 9 | 0.3 | 0.1 | 0.1 | 0.1 | 0.2 | 0.3 | 0.2 | 0.3 | 0 | 0.2 | 0.2 | 0.1 | 0.1 | 0.2 | 0.3 | 0.3 | 0.2 | 0 |

**Table 3** Setup times of one robot before operating a task

| Tasks | Setup times of robots | | | |
|---|---|---|---|---|
| | Robot 1 | Robot 2 | Robot 3 | Robot 4 |
| 1 | 0.2 | 0.3 | 0.3 | 0.3 |
| 2 | 0.3 | 0.3 | 0.2 | 0.2 |
| 3 | 0.1 | 0.1 | 0.2 | 0.3 |
| 4 | 0.2 | 0.2 | 0.2 | 0.2 |
| 5 | 0.2 | 0.3 | 0.2 | 0.2 |
| 6 | 0.3 | 0.1 | 0.2 | 0.2 |
| 7 | 0.3 | 0.2 | 0.3 | 0.3 |
| 8 | 0.2 | 0.1 | 0.3 | 0.3 |

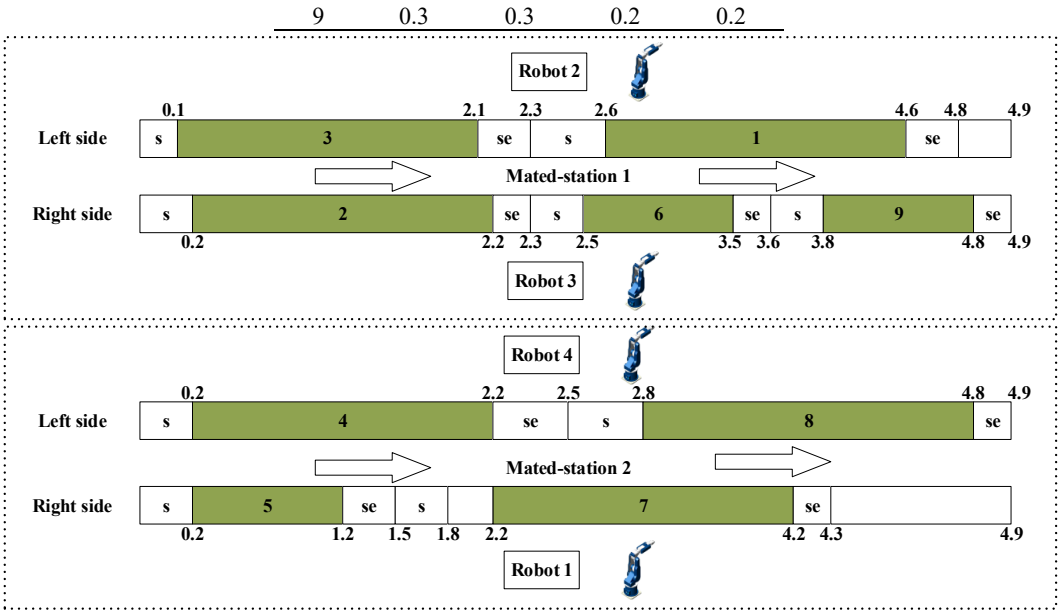


**Figure 6** Detailed task assignment and robot allocation by the proposed model

## 5. Computational study

This section presents the computational study, where the experimental design is first introduced. Later, the new proposed model is compared with the simple model from the reported work in Aghajani et al. [37]. Finally, a comprehensive comparative study on the results obtained using the 13 implemented metaheuristic algorithms.

### 5.1 Experimental design

As there are no available benchmark instances for RTALBP with setup times, this paper generates two sets of datasets on the basis of the utilized dataset in Li et al. [34]: low variability and high variability. The dataset in Li et al. [34] contains 39 problems with different task number or mated-station numbers: P9 with 2 and 3 mated-stations, P12 with 2, 3, 4 and 5 mated-stations, P16 with 2, 3, 4 and 5 mated-stations, P24 with 2, 3, 4 and 5 mated-stations, P65 with 4, 5, 6, 7 and 8 mated-stations, P148 with 4, 5, 6, 7, 8, 9, 10, 11 and 12 mated-stations, P205 with 4, 5, 6, 7, 8, 9, 10, 11,12, 13 and 14 mated-stations. The precedence relations of this dataset are taken from that in the traditional two-sided assembly line, and the operation time of task $i$ by robot $r$ ($t_{ir}$) are randomly generated between $[t_i \times 0.8, t_i \times 1.2\ ]$ where $t_i$ is the original operation time in TALBP. In this study, for each set of dataset, the precedence relations, the operation times of tasks by robots and the direction constraints are taken from Li et al. [34] directly. The setup times are generated following Andrés et al. [38] and Janardhanan et al. [47] as follows.

**Low variability:** The sequence-dependent setup time $tsu_{ipr}$ and the robot setup time $tsr_{ir}$ are randomly generated based on the uniform discrete distribution $U[0, 0.25 * min_{\forall h \in I} t_{hr}]$.

**High variability:** The sequence-dependent setup time $tsu_{ipr}$ and the robot setup time $tsr_{ir}$ are randomly generated based on the uniform discrete distribution $U[0, 0.75 * min_{\forall h \in I} t_{hr}]$.

For each dataset, there are 39 problems with different task number or robot number. With two levels of variability in setup times there are 78 instances in total. The detailed datasets are available upon request. All the implemented methodologies are tested utilizing the same termination criterion utilized in Janardhanan et al. [47]; the termination criterion is the maximum elapsed CPU time equal to $Nt \times Nt \times \tau$ milliseconds, where $Nt$ is the number of tasks and $\tau$ is set to 10, 20, 30, 40, 50 and 60; respectively. The utilization of six termination criteria allows the observation of the algorithms' performance from short running time to large running time.

To calibrate the parameters of the tested algorithms, each parameter of one tested algorithm is set to 2 or 3 high-performing levels on the basis of the reported literature to keep the calibration at manageable level. This study employs a Design of Experiments approach where each parameter is regarded as a controlled factor. All the combinations of the parameters are tested in full factorial experimental design and each combination of the parameters solve ten large-size instances for 10 times. As different instances are utilized, the relative percentage deviation (RPD) is utilized to transfer the achieved the cycle times using $RPD = 100 \cdot (CT_{some} - CT_{Best})/CT_{Best}$, where $CT_{some}$ is the cycle time achieved by one combination and $CT_{Best}$ is the best cycle time obtained by all the combinations of the parameters. Finally, the ANOVA test is conducted to select the most effective level for each parameter, where the average $RPD$ of 10 instances in one run is regarded as the response variable following Li et al. [34] and Janardhanan et al. [47]. Namely, there are 10 average RPD values for each combination that are utilized in the ANOVA test. The selected parameters of the tested and re-implemented algorithms are presented in Table 4.

All the experiment results will be available in public domain like Research gate and also available on request. All the tested algorithms are coded in C++ programming language and executed on a set of virtual computers in one tower type server. This server is equipped with two Intel Xeon E5-2680 v2 processors at 2.8 GHz with 20 processor cores in each processor, and a total of 64 GB of RAM memory. Each virtual computer is equipped with 1 processor core and 2 GB of RAM memory.

**Table 4** Selected for the tested and re-implemented algorithms

| *Algorithm* | *Parameters* | *Selected value* |
|---|---|---|
| SA | Initial temperature | 1.0 |
| | Cooling rate | 0.95 |
| | Iteration time before the temperature updates | 500 |
| RSA | Initial temperature | 0.5 |
| | Cooling rate | 0.9 |
| | Iteration time before the temperature updates | 500 |
| | Restart temperature | 0.0001 |
| | The number of consecutive times before replacing the current temperature with the restart temperature | 100 |
| LAHC | List length | 10 |
| PSO | Population size | 160 |
| | Number of swarms | 8 |
| | Parameter c | 0.7 |
| GA | Population size | 160 |
| | Selection type | Binary tournament selection |
| | Crossover probability | 0.6 |
| | Mutation probability | 0.4 |
| DCS | Population size | 20 |

| | | |
|---|---|---|
| | Abandonment rate | 0.1 |
| BA | Population size | 40 |
| | Number of employed bees | 20 |
| | Number of best employed bees | 2 |
| | Number of onlookers for each best employed bee | 10 |
| | Number of onlookers for each employed bee except for the best employed bee | 1 |
| ABC1 | Population size | 20 |
| ABC2 | Population size | 20 |
| MBO | Population size | 5 |
| | Number of neighbor solutions for the leader individual | 11 |
| | Number of shared neighbors of the individual in the front | 5 |
| | The number of executed consecutively times before conducting the leader replacement procedure | 20 |
| | The number of iterations where the current best cycle time remained unchanged before utilizing the simulated annealing-like acceptance criterion | 500 |
| | Restart temperature in the simulated annealing-like acceptance criterion | 0.2 |
| | Cooling rate in the simulated annealing-like acceptance criterion | 0.95 |
| CoPSO | Population size | 80 |
| | Number of swarms | 8 |
| | Parameter c | 0.7 |
| | The number of iterations when the best solution remained unchanged before conducting the restart mechanism | 50 |
| CoGA | Population size | 160 |
| | Selection type | Binary tournament selection |
| | Crossover probability | 0.4 |
| | Mutation probability | 0.6 |
| | The number of iterations when the best solution remained unchanged before conducting the restart mechanism | 50 |
| CoCS | Population size | 20 |
| | Abandonment rate | 0.1 |
| | The number of iterations when the best solution remained unchanged before conducting the restart mechanism | 200 |

## 5.2 Model evaluation

This section evaluated the newly developed model in Section 2 and compared with the published model which is shown in Appendix A. Figure 7 illustrates the detailed task assignment and robot allocation by the model in Appendix A. From the figure, it is observed that the model in Appendix A has three drawbacks. Firstly, the robot setup time for the first task on one workstation is ignored. Secondly, the robot setup times and sequence-dependent setup times among the tasks, which have precedence relations and are allocated to the same workstation, are ignored. Thirdly, the setup time between the last task and the first task in one workstation is ignored.

The proposed model, on the contrary, is capable to rectify these drawbacks and the task assignment and robot allocation shown in Figure 6 using the new proposed model satisfies all the constraints.

Table 5 presents the results obtained by the newly developed model using CPLEX solver in the General Algebraic Modeling System 23.0 with running time limited to 3600 seconds (s). The best cycle times by all implemented algorithms are also presented here. From this table, it is observed that the proposed model achieves optimal solutions for three former instances with lower or high setup times. The implemented metaheuristic algorithms obtain the same values, and clearly the results obtained by the model are in accordance with that of the algorithms. However, the model in Appendix

A achieves smaller and wrong cycle times (see Figure 7). It is observed that the RTALBP with setup times are much more complex than RTALBP. CPLEX solver could obtain 10 optimal solutions for RTALBP and only three optimal solutions for RTALBP with low or high setup times. Hence, metaheuristic methodologies are proposed and are necessary to achieve optimal or near optimal solutions in acceptable computation time.

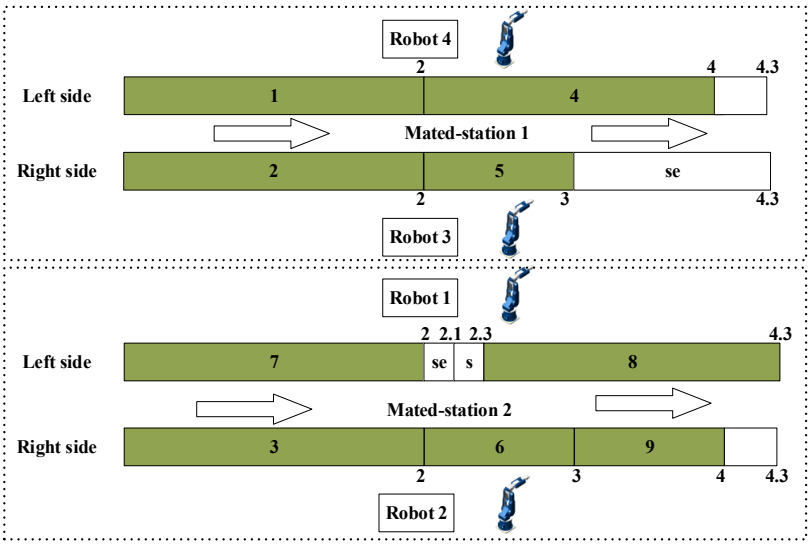


**Figure 7** Detailed task assignment and robot allocation by the model in Appendix A

**Table 5** Results by the newly developed model

| Instances | *Nj* | CPLEX solver | | | | | | Algorithms | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | No setup | | Low setup | | High setup | | No setup | Low | High |
| | | CT | CPU(s) | CT | CPU(s) | CT | CPU(s) | CT | CT | CT |
| P9 | 2 | 4 | 0.21 | 4.9 | 16.76 | 5.7 | 52.56 | 4 | 4.9 | 5.7 |
| P9 | 3 | 3 | 0.18 | 3.3 | 69.89 | 3.9 | 67.74 | 3 | 3.3 | 3.9 |
| P12 | 2 | 6 | 0.38 | 6.5 | 947.38 | 7.5 | 891.04 | 6 | 6.5 | 7.5 |
| P12 | 3 | 4 | 0.36 | 5.4 | 3600 | 5.5 | 3600 | 4 | 4.4 | 5.5 |
| P12 | 4 | 4 | 1.45 | - | - | - | - | 4 | 4.1 | 4.6 |
| P12 | 5 | 3 | 1.33 | - | - | - | - | 3 | 3.2 | 3.4 |
| P16 | 2 | 21 | 1.57 | - | - | - | - | 21 | 22.1 | 25.5 |
| P16 | 3 | 14 | 0.92 | - | - | - | - | 14 | 15.2 | 17 |
| P16 | 4 | 13 | 412.15 | - | - | - | - | 13 | 13.3 | 14 |
| P16 | 5 | 9 | 2.38 | - | - | - | - | 9 | 9.7 | 11.1 |

## 5.3 Comparative study

This section presents a comprehensive and statistical evaluation of the implemented algorithms. All the instances are solved for 10 independent times under six termination criteria, and the achieved cycle times are recorded. As different instances are tackled, the cycle times are transferred using relative percentage deviation or RPD. Namely, there are $78 \times 10 \times 6$ data to evaluate the algorithms' performance. This rich dataset provides a better observation of the algorithms' performance under different termination criteria and helps obtain sound findings. For space reasons, this section mainly presents the average RPD results to have a better evaluation of the algorithms' performance. It is to be noted that, this study has also been calculated best cycle time, the average cycle time and standard deviation for one instance by each algorithm. As it is difficult to evaluate different instances using these evaluation metrics, the average

RPD is selected as the evaluation metric and the detailed results regarding other evaluation metrics are available upon request.

Table 6 illustrates the average RPD values of all the instances in 10 repetitions by all the tested algorithms. These algorithms might be divided into four categories, in which the algorithms show similar performances. The best performing category includes MBO, ABC2 and ABC1, the second-best performing category includes SA and RSA, the third-best performing category includes RCoGA, CoCS and DCS, and the worst performing category includes PSO, CoPSO, GA, LAHC and BA. For the three best performing algorithms in the first category, they all employ efficient techniques to enhance exploration and escape from being trapped into local optima. Specifically, MBO utilizes new acceptation criterion to accept poor solutions with a certain probability; ABC1 and ABC2 employ the scout phase to replace the duplicated or the worst performing individual with neighbor solution. Local search methods, SA and RSA, belong to the second-best performing category. SA and RSA outperform eight algorithms (PSO, CoPSO, GA, LAHC, BA, RCoGA, CoCS and DCS), whereas they are outperformed by three swam intelligence based algorithms (MBO, ABC2 and ABC1). PSO and CoPSO are the two worst performers.

It is to be noted that the performance of the algorithms in Table 6 are in accordance with those reported in Janardhanan et al. [47], however they seem to be in conflict with that reported in Li et al. [35]. The reason behind this is that the proposed two ABC methods replace the selected solution with a neighbor solution of a randomly selected solution from the remaining individuals in scout phase to have a new high-quality solution. Another main reason is that all the tested algorithms utilize the iterative mechanism presented in Section 3.1, which ensures that the individuals are evaluated using the same initial cycle time, to preserve the minor improvements. As the iterative mechanism improves the algorithm's performance by a significant margin observed in preliminary experiments, it is applied in this computational test.

**Table 6** The average RPD values by tested algorithms

| Low setup times | | | | | | |
|---|---|---|---|---|---|---|
| Algorithm | Nt×Nt×10 | Nt×Nt×20 | Nt×Nt×30 | Nt×Nt×40 | Nt×Nt×50 | Nt×Nt×60 |
| PSO | 7.97 | 7.51 | 7.26 | 7.13 | 7.02 | 6.92 |
| CoPSO | 4.41 | 3.73 | 3.46 | 3.31 | 3.18 | 3.10 |
| GA | 3.67 | 3.28 | 3.05 | 2.89 | 2.73 | 2.62 |
| LAHC | 4.02 | 3.43 | 3.11 | 2.86 | 2.70 | 2.58 |
| BA | 4.04 | 3.26 | 2.89 | 2.62 | 2.43 | 2.27 |
| RCoGA | 2.61 | 2.33 | 2.19 | 2.10 | 2.06 | 2.03 |
| CoCS | 3.19 | 2.67 | 2.33 | 2.22 | 2.10 | 2.02 |
| DCS | 3.02 | 2.54 | 2.26 | 2.07 | 1.95 | 1.88 |
| SA | 2.97 | 2.37 | 2.05 | 1.85 | 1.67 | 1.55 |
| RSA | 2.92 | 2.35 | 2.02 | 1.82 | 1.65 | 1.53 |
| MBO | 2.31 | 1.77 | 1.52 | 1.35 | 1.24 | 1.15 |
| ABC2 | 2.54 | 1.80 | 1.48 | 1.32 | 1.16 | 1.05 |
| ABC1 | 2.49 | 1.79 | 1.50 | 1.26 | 1.13 | 1.02 |
| High setup times | | | | | | |
| Algorithm | Nt×Nt×10 | Nt×Nt×20 | Nt×Nt×30 | Nt×Nt×40 | Nt×Nt×50 | Nt×Nt×60 |
| PSO | 8.45 | 8.12 | 7.92 | 7.76 | 7.66 | 7.57 |
| CoPSO | 4.75 | 4.12 | 3.86 | 3.69 | 3.56 | 3.45 |
| GA | 4.46 | 3.99 | 3.74 | 3.53 | 3.35 | 3.25 |
| LAHC | 4.41 | 3.77 | 3.43 | 3.19 | 3.02 | 2.89 |
| BA | 4.47 | 3.70 | 3.31 | 3.06 | 2.87 | 2.70 |
| RCoGA | 2.87 | 2.69 | 2.57 | 2.50 | 2.44 | 2.41 |

| | | | | | | |
|---|---|---|---|---|---|---|
| CoCS | 3.63 | 3.08 | 2.82 | 2.64 | 2.55 | 2.48 |
| DCS | 3.37 | 2.82 | 2.56 | 2.42 | 2.28 | 2.15 |
| SA | 3.48 | 2.82 | 2.51 | 2.29 | 2.13 | 2.01 |
| RSA | 3.57 | 2.91 | 2.59 | 2.37 | 2.20 | 2.07 |
| MBO | 2.58 | 2.07 | 1.84 | 1.66 | 1.54 | 1.44 |
| ABC2 | 2.78 | 2.14 | 1.84 | 1.64 | 1.50 | 1.37 |
| ABC1 | 2.79 | 2.11 | 1.79 | 1.58 | 1.43 | 1.32 |

To have a better observation of the performance of algorithms on different instances, Table 7 shows the average RPD of one dataset for ten repetitions of RTALBP with low setup times and a termination criterion of running time limited to $Nt \times Nt \times 60$ milliseconds. It can be observed that ABC1 is the best performer with the average RPD of 1.02, and the remaining algorithms are listed in the increasing order of the average RPD such as ABC2, MBO, RSA, SA, DCS, CoCS, RCOGA, BA, LAHC, GA, CoPSO and PSO. ABC1 outperforms ABC2, MBO, RSA and SA for 18, 20, 30 and 27 cases; respectively. Moreover, the algorithms show slightly different performance under different termination criteria. For example, MBO is the best performer for RTALBP with low setup times with the running time limited to $Nt \times Nt \times 10$ milliseconds based on the average RPD value. It can be concluded that MBO, ABC2 and ABC1 in the best performing category are the best three algorithms due to much smaller average RPD.

In order to have an observation of the algorithms under different termination criteria and ascertain that the observed difference is statistically significant, this section also conducts statistical analysis, using the ANOVA test[51] after conducting the normality test and equal variance test. In the ANOVA test, algorithm types and termination criteria are regarded as two controllable factors and the average RPD of all instances in one run is utilized as the response variable based on the procedure in Janardhanan et al. [47]. The ANOVA test shows that P-values of algorithm types or termination criteria are less than 0.001 in solving RTALBP with low setup times and high setup times, indicating that there are statistically significant differences in algorithm types or termination criteria. Table 8 illustrates the detailed ANOVA results in solving RTALBP with high setup times, and it is clear that there is statistically significant difference in the algorithms' performance.

To have a better observation of the algorithms' performance, Fig.8 depicts the means plots of the average RPD value by six best performing algorithms in solving RTALBP with low setup times when $\tau$=20, 40 and 60 for a better clarity. Figure 9 depicts the means plots of the average RPD value by these algorithms in solving RTALBP with high setup times. In Figure 8, it is clear the ABC1, ABC2 and MBO are the three best performers, and they outperform the remaining methods under all the three termination criteria. Recall that the overlapped interval in this figure denotes statistically insignificant difference at 95% confidence level. Hence, it is observed that there is no statistically significant difference between ABC1, ABC2 and MBO whereas ABC1, ABC2 and MBO outperform RSA, SA and DCS statistically. Similarly, ABC1, ABC2 an MBO in Figure 9 outperform RSA, SA and DCS statistically, and there is no statistically significant difference among ABC1, ABC2 and MBO. From the above statistical analysis, it can be concluded that ABC1, ABC2 and MBO are three best

performers for RTALBP with setup times and they outperform the remaining implemented methodologies statistically with a 95% confidence level.

**Table 7** Average RPD by tested algorithms of RTALBP with low setup times

| Problem | Nj | PSO | CoPSO | GA | LAHC | BA | RCOGA | CoCS | DCS | SA | RSA | MBO | ABC2 | ABC1 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| P9 | 2 | 0.00 | 0.00 | 0.00 | 2.45 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| P9 | 3 | 0.00 | 0.00 | 0.00 | 2.12 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.30 | 0.91 | 0.00 | 0.00 |
| P12 | 2 | 0.00 | 0.00 | 0.00 | 3.23 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| P12 | 3 | 0.00 | 1.36 | 0.45 | 14.09 | 0.00 | 0.00 | 0.00 | 6.36 | 0.00 | 1.36 | 0.00 | 0.00 | 0.00 |
| P12 | 4 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| P12 | 5 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| P16 | 2 | 0.00 | 0.00 | 0.00 | 4.80 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| P16 | 3 | 0.46 | 0.26 | 0.13 | 3.29 | 0.00 | 0.07 | 0.00 | 0.00 | 0.20 | 0.20 | 0.00 | 0.13 | 0.00 |
| P16 | 4 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| P16 | 5 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| P24 | 2 | 1.32 | 0.44 | 1.10 | 2.38 | 0.90 | 1.51 | 0.58 | 0.58 | 1.01 | 0.44 | 0.22 | 0.44 | 0.47 |
| P24 | 3 | 2.42 | 0.87 | 1.38 | 1.54 | 0.83 | 0.88 | 0.75 | 1.29 | 0.96 | 1.04 | 0.58 | 1.04 | 0.83 |
| P24 | 4 | 1.23 | 0.89 | 1.23 | 1.51 | 0.61 | 1.12 | 0.39 | 0.84 | 0.95 | 0.78 | 0.11 | 0.39 | 0.61 |
| P24 | 5 | 5.44 | 1.48 | 1.68 | 0.13 | 0.74 | 0.54 | 1.28 | 0.34 | 0.13 | 0.13 | 0.00 | 0.40 | 0.27 |
| P65 | 4 | 8.27 | 4.04 | 3.50 | 2.47 | 3.12 | 2.31 | 2.19 | 2.52 | 2.08 | 2.42 | 1.71 | 2.24 | 2.18 |
| P65 | 5 | 9.88 | 4.11 | 4.04 | 2.09 | 2.96 | 2.41 | 2.85 | 2.55 | 2.43 | 2.18 | 1.56 | 2.09 | 1.97 |
| P65 | 6 | 10.26 | 3.72 | 3.30 | 2.20 | 2.83 | 2.28 | 2.32 | 1.75 | 1.76 | 1.73 | 1.28 | 1.41 | 1.52 |
| P65 | 7 | 10.35 | 4.12 | 4.33 | 1.93 | 2.96 | 2.42 | 2.48 | 2.43 | 1.69 | 1.88 | 1.35 | 1.85 | 2.14 |
| P65 | 8 | 11.22 | 5.60 | 4.83 | 1.90 | 4.02 | 3.25 | 3.41 | 2.99 | 1.63 | 1.86 | 1.51 | 1.91 | 1.81 |
| P148 | 4 | 10.15 | 4.34 | 4.73 | 3.19 | 3.55 | 3.78 | 3.55 | 3.17 | 2.65 | 2.78 | 3.48 | 2.43 | 2.51 |
| P148 | 5 | 8.49 | 3.13 | 2.48 | 2.17 | 2.59 | 1.89 | 1.71 | 1.56 | 2.30 | 1.82 | 1.43 | 0.96 | 1.07 |
| P148 | 6 | 10.66 | 5.13 | 4.44 | 4.10 | 4.24 | 3.28 | 3.59 | 2.48 | 3.58 | 2.88 | 2.73 | 1.49 | 2.29 |
| P148 | 7 | 9.41 | 4.58 | 3.74 | 2.83 | 3.78 | 3.17 | 2.97 | 2.58 | 2.45 | 2.49 | 1.82 | 1.11 | 1.48 |
| P148 | 8 | 9.85 | 4.57 | 3.61 | 2.87 | 3.40 | 3.74 | 3.02 | 2.53 | 2.71 | 2.63 | 1.41 | 1.76 | 1.12 |
| P148 | 9 | 9.70 | 4.69 | 3.08 | 2.45 | 3.33 | 2.92 | 2.68 | 2.40 | 2.21 | 1.93 | 1.27 | 1.31 | 1.63 |
| P148 | 10 | 10.38 | 4.86 | 3.63 | 2.92 | 3.50 | 3.58 | 3.17 | 2.56 | 2.51 | 2.65 | 1.67 | 1.31 | 1.66 |
| P148 | 11 | 10.44 | 4.57 | 3.62 | 2.45 | 3.05 | 2.89 | 3.04 | 2.07 | 2.32 | 2.39 | 1.84 | 1.05 | 1.12 |
| P148 | 12 | 11.09 | 5.64 | 4.45 | 2.97 | 4.02 | 3.70 | 4.01 | 3.57 | 2.86 | 2.59 | 2.33 | 1.30 | 1.87 |
| P205 | 4 | 7.75 | 2.52 | 2.29 | 1.03 | 1.76 | 2.03 | 1.57 | 1.78 | 0.80 | 0.70 | 1.54 | 0.85 | 0.68 |
| P205 | 5 | 8.89 | 3.25 | 3.19 | 2.31 | 2.90 | 1.55 | 2.30 | 2.10 | 2.12 | 1.85 | 2.20 | 1.62 | 1.60 |
| P205 | 6 | 9.09 | 4.07 | 3.18 | 2.13 | 3.10 | 3.08 | 2.62 | 2.08 | 1.85 | 1.76 | 2.06 | 1.24 | 1.23 |
| P205 | 7 | 9.37 | 4.24 | 3.27 | 1.47 | 2.56 | 2.36 | 2.54 | 1.63 | 1.05 | 1.25 | 1.09 | 1.09 | 0.88 |
| P205 | 8 | 10.61 | 5.13 | 3.83 | 2.55 | 4.17 | 3.20 | 3.67 | 2.99 | 2.65 | 2.39 | 1.84 | 1.97 | 1.68 |
| P205 | 9 | 11.68 | 5.54 | 4.37 | 2.94 | 3.92 | 3.35 | 3.34 | 2.46 | 2.32 | 2.06 | 1.54 | 1.62 | 1.71 |
| P205 | 10 | 11.68 | 5.21 | 4.25 | 2.16 | 3.54 | 2.97 | 3.28 | 2.52 | 1.99 | 1.54 | 1.35 | 1.94 | 0.78 |
| P205 | 11 | 10.95 | 4.56 | 3.44 | 4.17 | 3.83 | 2.97 | 3.03 | 3.59 | 3.57 | 4.17 | 1.88 | 1.63 | 1.18 |
| P205 | 12 | 12.09 | 5.49 | 3.79 | 3.29 | 3.95 | 3.71 | 3.85 | 3.02 | 2.32 | 2.41 | 1.19 | 0.88 | 0.82 |
| P205 | 13 | 12.45 | 5.94 | 4.45 | 2.66 | 4.03 | 4.41 | 3.78 | 3.02 | 2.16 | 2.06 | 0.96 | 1.01 | 0.93 |
| P205 | 14 | 14.38 | 6.57 | 6.39 | 3.83 | 4.43 | 3.62 | 4.78 | 3.54 | 3.14 | 2.96 | 1.85 | 2.60 | 1.71 |
| AVG | | 6.92 | 3.10 | 2.62 | 2.58 | 2.27 | 2.03 | 2.02 | 1.88 | 1.55 | 1.53 | 1.15 | 1.05 | 1.02 |

**Table 8** Detailed ANOVA results

| Source of Variation | DF | SS | MS | F | P |
|---|---|---|---|---|---|
| Algorithm types | 5 | 23.858 | 4.772 | 221.922 | **<0.001** |
| Termination criteria | 2 | 17.576 | 8.788 | 408.722 | **<0.001** |
| Algorithm types×Termination criteria | 10 | 0.194 | 0.0194 | 0.901 | 0.534 |
| Residual | 162 | 3.483 | 0.0215 | | |
| Total | 179 | 45.112 | 0.252 | | |

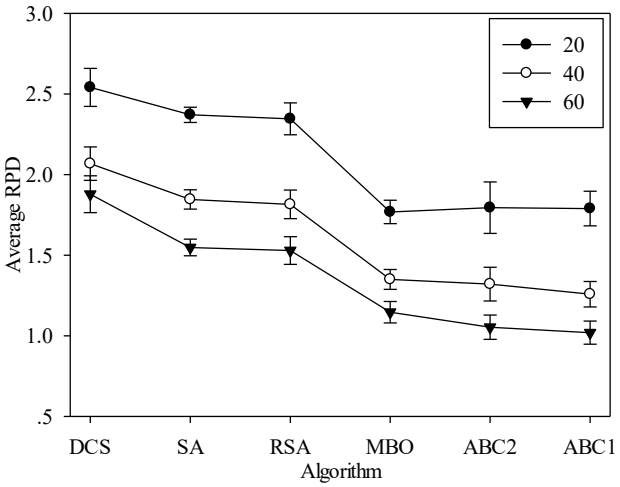


**Figure 8** Means plot and 95% Tukey HSD confidence intervals for the interactions between algorithms and termination criteria in solving RTALBP with low setup times

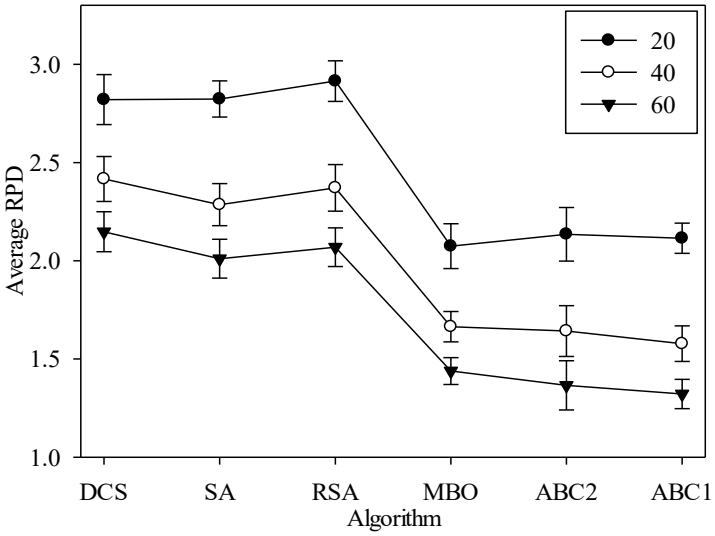


**Figure 9** Means plot and 95% Tukey HSD confidence intervals for the interactions between algorithms and termination criteria in solving RTALBP with high setup times

## 6. Conclusion and future research

Robotic two-sided assembly line balancing problem (RTALBP) with robot setup times and sequence-dependent setup times to minimize the cycle time is studied in this paper. A new mixed-integer program mathematical model is formulated, which is capable to solve the small-size instances optimally using CPLEX solver. Thirteen metaheuristic methodologies are implemented to tackle the large-size instances in acceptable computation time utilizing the iterative mechanism. All the algorithms solve two sets of instances: RTALBP with low setup times and RTALBP with high setup times, and the relative percentage deviation (RPD) is utilized to transfer the obtained cycle times to generalize the results for different instances.

Computational results show that the simple adaption of the model in mixed-model RTALBP produces wrong solution whereas the proposed model has rectified all the possible drawbacks. A comprehensive study is carried out to evaluate the performances

of these tested algorithms, and the statistical analysis demonstrates that two artificial bee colony algorithm and migrating bird optimization algorithms are the three best performers and outperform the remaining algorithms statistically with a 95% confidence level. The work done in this paper is very relevant to modern smart factories, where decisions are to be taken in short computation time that helps the production floor manager for real time decision support.

Regarding the managerial implications, the developed model and algorithms can be employed by the line managers to improve the line efficiency, reduce the time waste and thus reduce the assembly cost to makes the factories competitive. The developed algorithms help the managers obtain satisfying schedule within short computation time. Future studies might extend the presented algorithms to other scheduling problems, like parallel robotic assembly line balancing problems. Another research venue is to implement exact methods for this RTALBP as there is no exact method published up to date. And it is also suggested to develop some tight lower bounds to evaluate the final solution. There are multiple constraints that should also be studied such as including positional constraint, zoning constraint and so on. There are also multiple conflicting objectives, including balancing and trading-off the resources, and machine/robot loads, that needed to be optimized simultaneously in real applications. It is also observed that in industry there are workers and robots simultaneously operating the tasks, and hence it will be interesting to study more realistic cases and the relationship between workers and robots.

## Acknowledgment

This project is supported by National Natural Science Foundation of China under grants 61803287 and 51875421 and China Postdoctoral Science Foundation under grant 2018M642928. The authors are grateful for the anonymous referees for their insightful comments to improve this paper.

### Appendix A The adaption of published model

This model is the simple adaption of the model reported in Aghajani et al. [37] by removing the mixed-model constraints. This model utilizes two different variables as follows.

$x_{ijk}$ : 1, if task $i$ is assigned to workstation $(j,k)$; 0, otherwise.

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

This adapted model is presented using expressions (A1-A10), and for the detailed description please refer to Aghajani et al. [37].

$$\text{Min} \quad CT \tag{A1}$$

$$\sum_{j\in J}\sum_{k\in K(i)} x_{ijk} = 1 \quad \forall i \in I \tag{A2}$$

$$\sum_{g\in J}\sum_{k\in K(h)} g\cdot x_{hgk} \le \sum_{j\in J}\sum_{k\in K(i)} j\cdot x_{ijk} \quad \forall i \in I - P_0, h \in P(i) \tag{A3}$$

$$t_i^f \le CT \quad \forall i \in I \tag{A4}$$

$$t_i^f - t_h^f + \psi(1 - x_{ijk}) + \psi(1 - \sum_{f \in K(h)} x_{hjf}) \geq \sum_{r=1}^{nr} t_{ir} \cdot y_{rjk} \ , \quad \forall i \in I - P_0, h \in P(i), k \in K(i), j \in J \tag{A5}$$

$$t_p^f - t_i^f + \psi(1 - x_{ijk}) + \psi(1 - x_{pjk}) + \psi(1 - z_{ip}) \geq \sum_{r=1}^{nr} (t_{pr} + tsr_{pr} + tsu_{ipr}) \cdot y_{rjk} \tag{A6}$$

$$\forall i \in I, p \in \{c | c \in I - (P_a(i) \cup S_a(i) \cup C(i)) \text{ and } i < c\}, j \in J, k \in K(i) \cap K(p)$$

$$t_i^f - t_p^f + \psi(1 - x_{ijk}) + \psi(1 - x_{pjk}) + \psi \cdot z_{ip} \geq \sum_{r=1}^{nr} (t_{ir} + tsr_{ir} + tsu_{pir}) \cdot y_{rjk} \tag{A7}$$

$$\forall i \in I, p \in \{c | c \in I - (P_a(i) \cup S_a(i) \cup C(i)) \text{ and } i < c\}, j \in J, k \in K(i) \cap K(p)$$

$$t_i^f + \psi(1 - x_{ijk}) \geq \sum_{r=1}^{nr} t_{ir} \cdot y_{rjk} \quad \forall i \in I, j \in J, k \in K(i) \tag{A8}$$

$$\sum_{r=1}^{nr} y_{rjk} = 1 \quad \forall j \in J, k = 1,2 \tag{A9}$$

$$\sum_{j=1}^{nj} \sum_{k=1}^{2} y_{rjk} = 1 \quad \forall r \in R \tag{A10}$$

**Appendix B Detailed decoding procedure**

Part of the utilized notations in decoding procedure are presented in Section 2.1 and the remains are presented as follows.

| | |
|---|---|
| $Nt$ | Number of tasks. |
| $Nj$ | Number of mated-stations. |
| $t_i^f$ | Completion time of task $h$. |
| $wl_j$ | The completion time of the left-side workstation of the mated-station $j$ (including the idle time). |
| $wr_j$ | The completion time of the right-side workstation of the mated-station $j$ (including the idle time). |
| $SL$ | Number of allocated tasks that have been allocated to the left side of mated-station $j$. |
| $SL_j$ | Set of allocated tasks that have been allocated to the left side of mated-station $j$. |
| $SR$ | Number of allocated tasks that have been allocated to the right side of mated-station $j$. |
| $SR_j$ | Set of allocated tasks that have been allocated to the right side of mated-station $j$. |
| $ATL_j$ | Set of assignable tasks that can be allocated to the left side of mated-station $j$. |
| $ATR_j$ | Set of assignable tasks that can be allocated to the right side of mated-station $j$. |
| $sdit_{ijk}$ | Sequence depended idle time of task $i$ when task is allocated to $k$ side workstation of the mated-station $j$. |
| $SOL_j$ | The operation sequence of tasks that have been allocated to the left side of mated- |

| | |
|---|---|
| | station $j$. |
| $SOL_j(s)$ | The tasks in the *s*th position of the operation sequence $SOL_j$. |
| $SOR_j$ | The operation sequence of tasks that have been allocated to the right side of mated-station $j$. |
| $SOR_j(s)$ | The tasks in the *s*th position of the operation sequence $SOR_j(s)$. |
| $CT$ | The initial cycle time as an input parameter. |
| $NCT$ | The new cycle time achieved by the decoding procedure. |

The detailed decoding procedure is presented as follows.

**Input:** Task permutation vector and robot allocation vector and initial cycle time CT

**Step 1** Set $h = 1,\ j = 0,\ NCT = 0$;

**Step 2** Open a new mated-station and set $j = j + 1,\ wl_j = 0,\ wr_j = 0,\ SL = 0,\ SR = 0,\ SL_j = \emptyset,\ SR_j = \emptyset,\ SOL_j = \emptyset$ and $SOR_j = \emptyset$.

**Step 3** Allocate the robots in the $j \cdot 2$ position and $j \cdot 2 + 1$ position of the robot allocation vector to the workstation$(j, 1)$ and workstation$(j, 2)$; respectively.

**Step 4** Determine the assignable task sets for both sides: $ATL_j = \{i | i \in AL \cup AE, i$ is not assigned and any $p \in P(i)$ has been allocted $\}$ ; $ATR_j = \{i | i \in AR \cup AE, i$ is not assigned and any $p \in P(i)$ has been allocted $\}$.

**For** task $i \in ATL_j$

**If** $j < Nj$

**5.1** If $t_{ir} + tsr_{ir} + wl_j + tsu_{SOL_j(SL-1),i,r} + tsu_{i,SOL_j(1),r} > CT$, remove task $i$ from $ATL_j$ and continue;

**5.2** If $t_{ir} + tsr_{ir} + tf_{Right} + tsu_{i,SOL_j(1),r} > CT$ $(tf_{Right} = max\{tf_p | p \in P(i) \cap SR_j\})$, remove task $i$ from $ATL_j$ and continue;

**Endif**

**5.3** Calculate $sdit_{ij1} = max\{tf_{Right} - wl_j, 0\}$;

**Endfor**

**Step 5**

**For** task $i \in ATR_j$

**If** $j < Nj$

**5.4** If $t_{ir} + tsr_{ir} + wr_j + tsu_{SOR_j(SR-1),i,r} + tsu_{i,SOR_j(1),r} > CT$, remove task $i$ from $ATR_j$ and continue;

**5.5** If $t_{ir} + tsr_{ir} + tf_{Left} + tsu_{i,SOR_j(1),r} > CT$ $(tf_{Left} = max\{tf_p | p \in P(i) \cap SL_j\})$, remove task $i$ from $ATR_j$ and continue;

**Endif**

**5.6** Calculate $sdit_{ij2} = max\{tf_{Left} - wr_j, 0\}$;

**Endfor**

**Step 6**

**If** $ATL_j = \emptyset,\ ATR_j = \emptyset$ and $h \le Nt$

Execute Step 2;

**Else**

Execute Step 7;

**Step 7**

**If** $ATR_j = \emptyset$ or $(ATL_j \neq \emptyset$ and $ATR_j \neq \emptyset$ and $wl_j \le wr_j)$

**7.1** Select the task $i$ with minimum value of the $sdit_{ij1}$ and in the former position in the task permutation;

**7.2** Allocate task $i$ to the left side; $tf_i = max\left\{t_{ir} + tsr_{ir} + wl_j + tsu_{SOL_j(SL-1),i,r}, t_{ir} + tsr_{ir} + tf_{Right}\right\}$, $wl_j = tf_i$, $SL = SL + 1$, $SL_j = SL_j + \{i\}$, $SOL_j = SOL_j + \{i\}$, $SOL_j(SL) = i$ and $h = h + 1$.

**7.3** Set $NCT = max\left\{tf_i + tsu_{i,SOL_j(1),r}, NCT\right\}$ and execute Step 8;

**Else**

**7.4** Select the task $i$ with minimum value of the $sdit_{ij2}$ and in the former position in the task permutation;

**7.5** Allocate task $i$ to the right side; $tf_i = max\left\{t_{ir} + tsr_{ir} + wr_j + tsu_{SOR_j(SR-1),i,r}, t_{ir} + tsr_{ir} + tf_{Left}\right\}$, $wr_j = tf_i$, $SR = SR + 1$, $SR_j = SR_j + \{i\}$, $SOR_j = SOR_j + \{i\}$, $SOR_j(SR) = i$ and $h = h + 1$.

**7.6** Set $NCT = max\left\{tf_i + tsu_{i,SOR_j(1),r}, NCT\right\}$ and execute Step 8;

**Endif**

**If** all tasks have been allocated $(h > Nt)$

Terminate the decoding procedure.

**Step 8**

**Else**

Go to Step 4.

**Output:** CT

**Appendix C Illustrated example to transfer the encoding vectors into a feasible solution**

The procedure to transfer the encoding vectors into a feasible solution is presented utilizing the example in Section 4. Figurer A1 illustrate the detailed decoding procedures with a cycle time of 4.9, where $s$ refers to the robot setup time and $se$ refers to the sequence-dependent setup time. Table A1 presents the example procedure to obtain the feasible line balance following the decoding procedure in Appendix B.

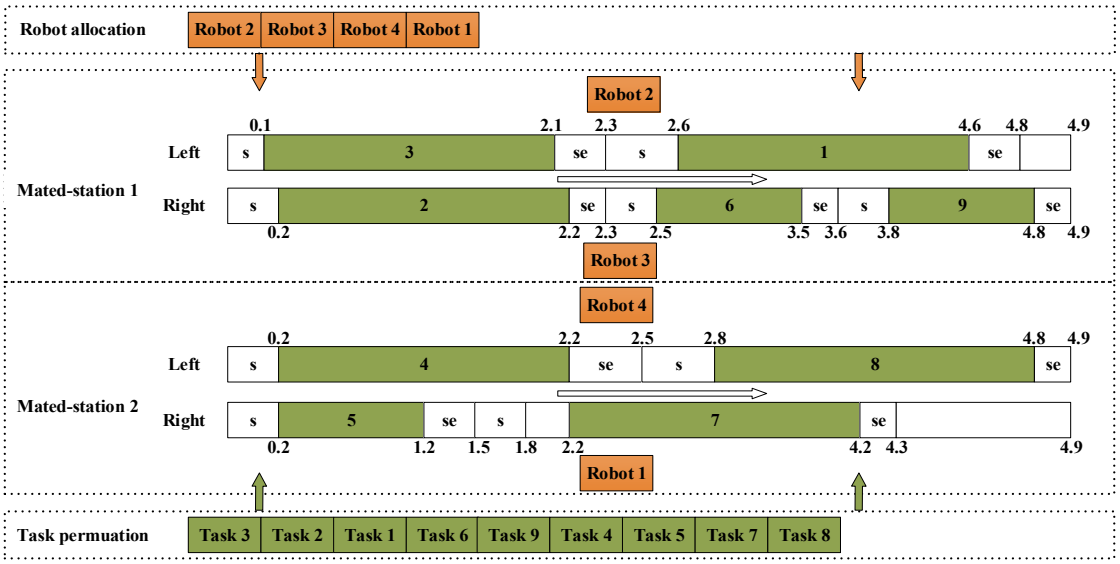

**Figure A1** Detailed encoding and decoding for the illustrated example

**Table A1** Example procedure to obtain the feasible line balance

| Assignable task | Select side and task | Assigning tasks to workstations |
|---|---|---|
| | Open a new mated-station; $j = 1$;<br>Allocate robot 2 to workstation (1, 1);<br>Allocate robot 3 to workstation (1, 2); | |
| $SL_j = \{1, 3\}$;<br>$SR_j = \{1, 2\}$; | Select left side;<br>Select task 3; | $tf_3 = 2.1;\ wl_1 = tf_3$;<br>$SL_1 = SL_1 + \{3\}$;<br>$SOL_1 = SOL_1 + \{3\}$; |
| $SL_j = \{1\}$;<br>$SR_j = \{2\}$; | Select right side;<br>Select task 2; | $tf_2 = 2.2;\ wr_1 = tf_2$;<br>$SR_1 = SR_1 + \{2\}$;<br>$SOR_1 = SOR_1 + \{2\}$; |
| $SL_j = \{1,6\}$;<br>$SR_j = \{5,6\}$; | Select left side;<br>Select task 1; | $tf_1 = 4.6;\ wl_1 = tf_1$;<br>$SL_1 = SL_1 + \{1\}$;<br>$SOL_1 = SOL_1 + \{1\}$; |
| $SL_j = \{\ \}$;<br>$SR_j = \{5,6\}$; | Select right side;<br>Select task 6; | $tf_6 = 3.5;\ wr_1 = tf_6$;<br>$SR_1 = SR_1 + \{6\}$;<br>$SOR_1 = SOR_1 + \{6\}$; |
| $SL_j = \{\ \}$;<br>$SR_j = \{9\}$; | Select right side;<br>Select task 9; | $tf_6 = 4.8;\ wr_1 = tf_9$;<br>$SR_1 = SR_1 + \{9\}$;<br>$SOR_1 = SOR_1 + \{9\}$; |
| $SL_j = \{\ \}$;<br>$SR_j = \{\ \}$; | Open a new mated-station; $j = 2$;<br>Allocate robot 4 to workstation (2, 1);<br>Allocate robot 1 to workstation (2, 2); | |
| $SL_j = \{4\}$;<br>$SR_j = \{5\}$; | Select left side;<br>Select task 4; | $tf_4 = 2.2;\ wl_2 = tf_4$;<br>$SL_2 = SL_2 + \{4\}$;<br>$SOL_2 = SOL_2 + \{4\}$; |
| $SL_j = \{\ \}$;<br>$SR_j = \{5,7\}$; | Select right side;<br>Select task 5; | $tf_5 = 1.2;\ wr_2 = tf_5$;<br>$SR_2 = SR_2 + \{5\}$;<br>$SOR_2 = SOR_2 + \{5\}$; |
| $SL_j = \{8\ \}$;<br>$SR_j = \{7\}$; | Select right side;<br>Select task 7; | $tf_7 = 4.2;\ wr_2 = tf_7$;<br>$SR_2 = SR_2 + \{7\}$;<br>$SOR_2 = SOR_2 + \{7\}$; |
| $SL_j = \{8\ \}$;<br>$SR_j = \{\ \}$; | Select left side;<br>Select task 8; | $tf_8 = 4.8;\ wl_2 = tf_8$;<br>$SL_2 = SL_2 + \{8\}$;<br>$SOL_2 = SOL_2 + \{8\}$; |
| $SL_j = \{\ \}$;<br>$SR_j = \{\ \}$; | Complete | |

**References**

[1] Scholl A, Becker C. State-of-the-art exact and heuristic solution procedures for simple assembly line balancing. European Journal of Operational Research. 2006;168(3):666-93.

[2] Battaïa O, Dolgui A. A taxonomy of line balancing problems and their solution approaches.

International Journal of Production Economics. 2013;142(2):259-77.
[3] Bartholdi JJ. Balancing two-sided assembly lines: a case study. International Journal of Production Research. 1993;31(10):2447-61.
[4] Li Z, Kucukkoc I, Nilakantan JM. Comprehensive review and evaluation of heuristics and meta-heuristics for two-sided assembly line balancing problem. Computers & Operations Research. 2017;84:146-61.
[5] Abdullah Make MR, Ab. Rashid MFF, Razali MM. A review of two-sided assembly line balancing problem. The International Journal of Advanced Manufacturing Technology. 2017;89(5-8):1743-63.
[6] Gao J, Sun L, Wang L, Gen M. An efficient approach for type II robotic assembly line balancing problems. Computers & Industrial Engineering. 2009;56(3):1065-80.
[7] Rubinovitz J, Bukchin J. Design and balancing of robotic assembly lines: Society of Manufacturing Engineers; 1991.
[8] Rubinovitz J, Bukchin J, Lenz E. RALB–A heuristic algorithm for design and balancing of robotic assembly lines. CIRP Annals-Manufacturing Technology. 1993;42(1):497-500.
[9] Levitin G, Rubinovitz J, Shnits B. A genetic algorithm for robotic assembly line balancing. European Journal of Operational Research. 2006;168(3):811-25.
[10] Yoosefelahi A, Aminnayeri M, Mosadegh H, Ardakani HD. Type II robotic assembly line balancing problem: An evolution strategies algorithm for a multi-objective model. Journal of Manufacturing Systems. 2012;31(2):139-51.
[11] Daoud S, Chehade H, Yalaoui F, Amodeo L. Solving a robotic assembly line balancing problem using efficient hybrid methods. Journal of Heuristics. 2014;20(3):235-59.
[12] Çil ZA, Mete S, Ağpak K. A Goal Programming Approach for Robotic Assembly Line Balancing Problem. IFAC-PapersOnLine. 2016;49(12):938-42.
[13] Borba L, Ritt M, Miralles C. Exact and heuristic methods for solving the Robotic Assembly Line Balancing Problem. European Journal of Operational Research. 2018;270(1):146-56.
[14] Mukund Nilakantan J, Huang GQ, Ponnambalam SG. An investigation on minimizing cycle time and total energy consumption in robotic assembly line systems. Journal of Cleaner Production. 2015;90:311-25.
[15] Nilakantan JM, Li Z, Tang Q, Nielsen P. Multi-objective co-operative co-evolutionary algorithm for minimizing carbon footprint and maximizing line efficiency in robotic assembly line systems. Journal of Cleaner Production. 2017;156:124-36.
[16] Nilakantan JM, Nielsen I, Ponnambalam SG, Venkataramanaiah S. Differential evolution algorithm for solving RALB problem using cost- and time-based models. The International Journal of Advanced Manufacturing Technology. 2017;89(1):311-32.
[17] Pereira J, Ritt M, Vásquez ÓC. A memetic algorithm for the cost-oriented robotic assembly line balancing problem. Computers & Operations Research. 2018;99:249-61.
[18] Çil ZA, Mete S, Özceylan E, Ağpak K. A beam search approach for solving type II robotic parallel assembly line balancing problem. Applied Soft Computing. 2017;61:129-38.
[19] Çil ZA, Mete S, Ağpak K. Analysis of the type II robotic mixed-model assembly line balancing problem. Engineering Optimization. 2017;49(6):990-1009.
[20] Rabbani M, Mousavi Z, Farrokhi-Asl H. Multi-objective metaheuristics for solving a type II robotic mixed-model assembly line balancing problem. Journal of Industrial and Production Engineering. 2016;33(7):472-84.
[21] Li Z, Janardhanan MN, Tang Q, Nielsen P. Mathematical model and metaheuristics for simultaneous

balancing and sequencing of a robotic mixed-model assembly line. Engineering Optimization. 2018;50(5):877-93.
[22] Lee TO, Kim Y, Kim YK. Two-sided assembly line balancing to maximize work relatedness and slackness. Computers & Industrial Engineering. 2001;40(3):273-92.
[23] Wu E-F, Jin Y, Bao J-S, Hu X-F. A branch-and-bound algorithm for two-sided assembly line balancing. The International Journal of Advanced Manufacturing Technology. 2008;39(9):1009-15.
[24] Hu X, Wu E, Jin Y. A station-oriented enumerative algorithm for two-sided assembly line balancing. European Journal of Operational Research. 2008;186(1):435-40.
[25] Xiaofeng H, Erfei W, Jinsong B, Ye J. A branch-and-bound algorithm to minimize the line length of a two-sided assembly line. European Journal of Operational Research. 2010;206(3):703-7.
[26] Kim YK, Kim Y, Kim YJ. Two-sided assembly line balancing: A genetic algorithm approach. Production Planning & Control. 2000;11(1):44-53.
[27] Özcan U, Toklu B. A tabu search algorithm for two-sided assembly line balancing. The International Journal of Advanced Manufacturing Technology. 2008;43(7):822-9.
[28] Baykasoglu A, Dereli T. Two-sided assembly line balancing using an ant-colony-based heuristic. The International Journal of Advanced Manufacturing Technology. 2008;36(5):582-8.
[29] Khorasanian D, Hejazi SR, Moslehi G. Two-sided assembly line balancing considering the relationships between tasks. Computers & Industrial Engineering. 2013;66(4):1096-105.
[30] Yuan B, Zhang C, Shao X. A late acceptance hill-climbing algorithm for balancing two-sided assembly lines with multiple constraints. Journal of Intelligent Manufacturing. 2015;26(1):159-68.
[31] Li D, Zhang C, Shao X, Lin W. A multi-objective TLBO algorithm for balancing two-sided assembly line with multiple constraints. Journal of Intelligent Manufacturing. 2016;27(4):725-39.
[32] Li Z, Tang Q, Zhang L. Two-sided assembly line balancing problem of type I: Improvements, a simple algorithm and a comprehensive study. Computers & Operations Research. 2017;79:78-93.
[33] Zhang Y, Hu X, Wu C. A modified multi-objective genetic algorithm for two-sided assembly line re-balancing problem of a shovel loader. International Journal of Production Research. 2018;56(9):3043-63.
[34] Li Z, Nilakantan JM, Tang Q, Nielsen P. Co-evolutionary particle swarm optimization algorithm for two-sided robotic assembly line balancing problem. Advances in Mechanical Engineering. 2016;8(9):1-14.
[35] Li Z, Dey N, Ashour AS, Tang Q. Discrete cuckoo search algorithms for two-sided robotic assembly line balancing problem. Neural Comput Appl. 2018;30(9):2685-96.
[36] Li Z, Tang Q, Zhang L. Minimizing energy consumption and cycle time in two-sided robotic assembly line systems using restarted simulated annealing algorithm. Journal of Cleaner Production. 2016;135:508-22.
[37] Aghajani M, Ghodsi R, Javadi B. Balancing of robotic mixed-model two-sided assembly line with robot setup times. The International Journal of Advanced Manufacturing Technology. 2014;74(5-8):1005-16.
[38] Andrés C, Miralles C, Pastor R. Balancing and scheduling tasks in assembly lines with sequence-dependent setup times. European Journal of Operational Research. 2008;187(3):1212-23.
[39] Allahverdi A, Soroush H. The significance of reducing setup times/setup costs. European Journal of Operational Research. 2008;187(3):978-84.
[40] Özcan U, Toklu B. Balancing two-sided assembly lines with sequence-dependent setup times. International Journal of Production Research. 2010;48(18):5363-83.

[41] Scholl A, Boysen N, Fliedner M. The sequence-dependent assembly line balancing problem. OR Spectrum. 2008;30(3):579-609.
[42] Yolmeh A, Kianfar F. An efficient hybrid genetic algorithm to solve assembly line balancing problem with sequence-dependent setup times. Computers & Industrial Engineering. 2012;62(4):936-45.
[43] Scholl A, Boysen N, Fliedner M. The assembly line balancing and scheduling problem with sequence-dependent setup times: problem extension, model formulation and efficient heuristics. OR Spectrum. 2013;35(1):291-320.
[44] Seyed-Alagheband S, Ghomi SF, Zandieh M. A simulated annealing algorithm for balancing the assembly line type II problem with sequence-dependent setup times between tasks. International Journal of Production Research. 2011;49(3):805-25.
[45] Hamta N, Ghomi SF, Jolai F, Shirazi MA. A hybrid PSO algorithm for a multi-objective assembly line balancing problem with flexible operation times, sequence-dependent setup times and learning effect. International Journal of Production Economics. 2013;141(1):99-111.
[46] Akpinar Ş, Baykasoğlu A. Modeling and solving mixed-model assembly line balancing problem with setups. Part II: A multiple colony hybrid bees algorithm. Journal of Manufacturing Systems. 2014;33(4):445-61.
[47] Janardhanan MN, Li Z, Bocewicz G, Banaszak Z, Nielsen P. Metaheuristic algorithms for balancing robotic assembly lines with sequence-dependent robot setup times. Applied Mathematical Modelling. 2019;65:256-70.
[48] Nilakantan JM, Li Z, Tang Q, Nielsen P. MILP models and metaheuristic for balancing and sequencing of mixed-model two-sided assembly lines European J Industrial Engineering. 2017;11(3):353-79.
[49] Tang Q, Li Z, Zhang L. An effective discrete artificial bee colony algorithm with idle time reduction techniques for two-sided assembly line balancing problem of type-II. Computers & Industrial Engineering. 2016;97:146-56.
[50] Meng T, Pan Q-K, Li J-Q, Sang H-Y. An improved migrating birds optimization for an integrated lot-streaming flow shop scheduling problem. Swarm and Evolutionary Computation. 2018;38:64-78.
[51] Wu C-C, Chen J-Y, Lin W-C, Lai K, Liu S-C, Yu P-W. A two-stage three-machine assembly flow shop scheduling with learning consideration to minimize the flowtime by six hybrids of particle swarm optimization. Swarm and Evolutionary Computation. 2018;41:97-110.