Mathematical models and simulated annealing algorithms for the robotic assembly line balancing problem

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

Published in:
Assembly Automation

DOI (link to publication from Publisher):
10.1108/AA-09-2017-115

Publication date:
2018

Document Version
Accepted author manuscript, peer reviewed version

emerald PUBLISHING

Assembly Automation

Mathematical models and simulated annealing algorithms for the robotic assembly line balancing problem

Journal:	*Assembly Automation*
Manuscript ID	AA-09-2017-115.R2
Manuscript Type:	Original Article
Keywords:	Assembly line balancing, Robotic assembly line, Integer programming;, Simulated annealing, Artificial Intelligence

Mathematical models and simulated annealing algorithms for the robotic assembly line balancing problem

Purpose- Robots are utilized in assembly lines due to their higher flexibility and lower costs. The purpose of this paper is to develop mathematical models and simulated annealing algorithms to solve the robotic assembly line balancing to minimize the cycle-time (RALB-II).

Design/methodology/approach - Four mixed-integer linear programming models are developed and encoded in CPLEX solver to find optimal solutions for small-sized problem instances. Two simulated annealing algorithms: original simulated annealing algorithm and restarted simulated annealing algorithm are proposed to tackle large-sized problems. The restart mechanism in the restarted simulated annealing methodology replaces the incumbent temperature with a new temperature. Additionally, the proposed methods employ iterative mechanisms for updating cycle-time and a new objective to select the solution with fewer critical workstations.

Findings- The comparative study among the tested algorithms and other methods adapted verifies the effectiveness of the proposed methods. The results obtained by these algorithm on the benchmark instances shows that 23 new upper bounds out of 32 tested cases are achieved. The restarted simulated annealing algorithm ranks first among the algorithms in the number of updated upper bounds.

Originality/value- Four models are developed for RALBP-II and their performance is evaluated for the first time. A restarted simulated annealing algorithm is developed to solve RALBP-II, where the restart mechanism is developed to replace the incumbent temperature with a new temperature. The proposed methods also employ iterative mechanisms and a new objective to select the solution with fewer critical workstations.

Keywords: Assembly line balancing; Robotic assembly line; Integer programming; Simulated annealing; Artificial intelligence

1. Introduction

Assembly lines have a wide variety of applications in modern automotive and consumer electronics industries to assemble different types of products (Scholl and Becker, 2006, Battaïa and Dolgui, 2013). Manufacturing enterprises face challenges such as increasing cost of labor, customized requests from customers and increasing sizes of product portfolios (Relich and Pawlewski, 2016). To address these challenges, robotic/automated assembly lines have increasingly replaced human-based lines. Robots can operate 24 hours a day without worries of fatigue and with reduced cost and higher flexibility (Gao et al., 2009, Nilakantan et al., 2017, Li et al., 2016b). Assembly line balancing (ALB) problem is a well-known decision problem arising when assembly lines are to be re-configured (Nourmohammadi et al., 2017) and for better utilization of robotic assembly lines, robotic assembly line balancing (RALB) problems are receiving increasing attention from researchers and production line managers. RALB problem without loss of generality, can be described as assigning a

set of tasks to workstations operated by the best-fit robots with one or several optimization criteria.

RALB problems can be divided into two categories: Type I robotic assembly line balancing (RALB-I) problems aim to minimize the number of workstations, and type II robotic assembly line balancing (RALB-II) problems aim to optimize cycle-time. As the simple assembly line balancing is already NP-hard (Scholl and Becker, 2006), the more complex RALB-I and RALB-II problems also belong to the NP-hard category. Since the initial work reported by Rubinovitz and Bukchin (1991), many exact and metaheuristic methods have been applied to solve RALB problems. These contributions can be further categorized into three types based on the assembly line layout, including general RALB problems, robotic U-shaped assembly line balancing (RUALB) problems, and robotic two-sided assembly line balancing (RTALB) problems (Battaïa and Dolgui, 2013).

Regarding general RALB problems where the layout of the assembly is in the form of a straight line, Rubinovitz and Bukchin (1991) present the first attempt to minimize the number of workstations, and, later, Rubinovitz et al. (1993) use a branch-and-bound algorithm for the same RALB-I problem. Levitin et al. (2006) develop a genetic algorithm to tackle RALB-II problems where all types of robots are assumed available without limitations. They develop a recursive assignment procedure and a consecutive assignment procedure for the efficient allocation of best-fit robots to the workstations. Gao et al. (2009) present a mixed-integer nonlinear programming model for a variant of the RALB-II problem in which the available robots are pre-determined. In their work, a type of robot is not available without limitations, and there is only one of each type of robot. They utilize a robot assignment vector to determine robot allocation and propose an improved genetic algorithm along with local search procedures. From their contribution, it is concluded that when all types of robots are available without limitations, the consecutive assignment procedure is a good choice for the selection of the robots. However, when the type of robot is not available without limitations, a robot assignment vector is a good choice for determining the robot allocation. Yoosefelahi et al. (2012) tackle a multi-objective RALB-II problem following the assumptions in Levitin et al. (2006) and present a new mixed-integer linear programming model and three versions of multi-objective evolution strategies. Daoud et al. (2014) propose several hybrid algorithms to maximize line efficiency, among which ant colony optimization with a guided local search achieves the best performance. Hybrid algorithms are well-known to have superior performance for certain problem types (Sitek and Wikarek, 2016, Do et al., 2016, Sitek et al., 2014). Nilakantan et al. (2015b) develop particle swarm optimization and cuckoo search algorithms to tackle the same RALB-II problem reported in Levitin et al. (2006) and present the improved solutions for the benchmark problems. Subsequently, Nilakantan et al. (2015a) present the first paper in the area of minimizing energy consumption in a straight robotic assembly line using particle swarm optimization based on the assumptions in Levitin et al. (2006). Çil et al. (2016) tackle the mixed-model RALB-II problem using beam search to optimize the sum of cycle-times over all models. More recently, Rabbani et al. (2016) solve the

multi-objective mixed-model RALB-II problem using a multi-objective genetic algorithm and particle swarm optimization. Nilakantan et al. (2017) optimize carbon footprint and line efficiency utilizing a multi-objective co-operative co-evolutionary algorithm following the assumptions in Gao et al. (2009).

Regarding a robotic assembly line with U-shaped layout (RUALB) problems, in all reported contributions it is assumed that all types of robots are available without limitations. Specifically, Nilakantan and Ponnambalam (2016) propose a particle swarm optimization algorithm embedded with a consecutive procedure to minimize the cycle-time of robotic assembly lines. In the case of two-sided robotic assembly lines (RTALB problems), all the reported contributions follow the assumption in Gao et al. (2009), where a robot assignment vector is used to determine robot allocation. Li et al. (2016a) optimize cycle-time using a co-evolutionary particle swarm optimization algorithm and they also develop a mixed-integer linear programming model to find optimal solutions for small-size problem instances. The same problem is tackled by Li et al. (2017a) using a discrete cuckoo search algorithm and co-evolutionary cuckoo search algorithm. These algorithms produce better results than those found in Li et al. (2016a). Later, Li et al. (2016b) optimize the energy consumption and cycle-time in RTALB problems using a Pareto simulated annealing algorithm. Aghajani et al. (2014) tackle mixed-model RTALB problems by minimizing the cycle-time. They develop a mixed-integer programming model to achieve the optimal solution for small-size problem instances and propose a simulated annealing algorithm for tackling large-size problem instances.

From the above literature review, two different basic assumptions appear as to whether all types of robots are assumed to be available without limitations. The first assumption in Levitin et al. (2006) is more appropriate for new assembly line design and the first installation of the robots. The consecutive assignment procedure select the robots, and, hence, the general algorithms summarized in Rashid et al. (2012) and Li et al. (2017b) are able to solve this kind of RALB problem directly. In contrast, the second assumption has diverse applications in reconfiguring/redesigning the robotic assembly lines (Gao et al., 2009) where the workstation number and the available robots remained unchanged. For this kind of RALB problem, the robot assignment vector is usually proposed to determine robot allocation. The algorithm for this RALB problem concerns the optimization of two or more vectors, and, hence, general algorithms might not be as effective. It is also observed that there are more contributions on the first type of RALB problem (Levitin et al., 2006, Nilakantan et al., 2015b, Yoosefelahi et al., 2012, Nilakantan et al., 2015a), whereas there is limited research on the second type of RALB problem (Gao et al., 2009, Nilakantan et al., 2017).

For the aforementioned reasons, this research studies the RALB-II problem following Gao et al. (2009) and presents several novel contributions as follows:

1) Four mixed-integer linear programming models are developed to tackle small-size problem instances optimally. In addition, these models are evaluated by solving a set of benchmark problems. It is to be noted that the model presented in Gao et al. (2009) is a non-linear programming model, and only two small-sized cases are solved within

acceptable computational time.

2) Two simulated annealing algorithms are proposed to solve large-sized problem instances in which the first is the original simulated annealing (SA) algorithm and the second is the restarted simulated annealing algorithm (RSA). The proposed RSA employs a restart mechanism to replace the incumbent temperature with a new temperature. In addition, this research proposes two improvements to enhance the performance of the algorithms: an iterative mechanism for cycle-time update and a new objective to select the solution with fewer critical workstations.

3) A comprehensive comparative study is carried out to test the performance of the proposed algorithms. The compared methods include a genetic algorithm, a particle swarm optimization algorithm, a cuckoo search algorithm, and two artificial bee colony algorithms. Statistical analysis compares these algorithms, where RSA achieves the best overall performance. Additionally, these compared algorithms achieve 23 new upper bounds out of 32 tested cases where especially the upper bounds for all large-size cases are updated.

The remainder of this paper is organized as follows. Section 2 provides a detailed description of the proposed four mathematical models. Section 3 illustrates the two proposed simulated annealing algorithms along with a detailed encoding scheme and decoding procedure. Section 4 presents the computational study in which both the models and algorithms are evaluated and compared. Section 5 concludes this paper and gives several suggestions on future research avenues.

2. Mathematical model formulation

This section first describes the problem and the basic problem assumptions and later presents the details of the four proposed integer-programming models for solving the RALB-II problem.

2.1 Problem description

As presented in Section 1, this paper tackles the RALB-II problem based on the work presented in Gao et al. (2009). The assumptions listed here are based on the ones reported in Gao et al. (2009) and Nilakantan et al. (2017):

- A single type of product is assembled in this robotic line.
- The operation times of tasks depend on the assigned robot, and they are deterministic.
- Each robot is allocated to a workstation and each workstation has a robot.
- The number of available robots is equal to the number of workstations.
- A task can be operated by any robot and a robot can be allocated to any workstation.
- Material handling, loading & unloading, setup& tool changing are considered negligible.

In robotic assembly lines, there is a set of workstations allocated with a set of robots. Supposing that there are *Nt* tasks and *Ns* workstations, it is clear that there are also *Nr* robots allocated to *Ns* workstation, where *Nt* is the number of tasks and *Ns* is the number of workstations. RALB-II problem concerns assigning the *Nt* tasks to *Ns*

workstations and allocating *Nr* robots to *Ns* workstations with the objective of minimizing cycle-time. In short, the RALB-II problem consists of two sub-problems that are to be optimized simultaneously: task assignment and robot allocation. Regarding the task assignment, a task can be executed only when all predecessors have been completed and the successors of a task must be assigned to the same workstation or a latter one. Regarding robot allocation, each workstation must be equipped with a robot and a robot must be allocated a workstation. A layout of robotic assembly is depicted in Figure 1 in which there are 25 tasks and six robots. 25 tasks are distributed among six workstations and they are operated in a sequence on the given workstations. Correspondingly, there are six robots allocated to the six workstations. It is to be noted that the largest value of the total operation times of tasks on workstations is regarded as the achieved cycle-time.

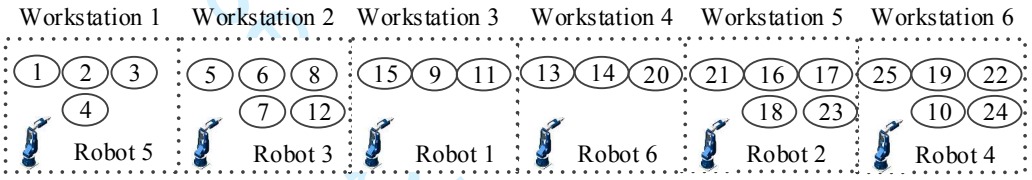

Figure 1. Layout of robotic assembly line

2.2 Integer programming models

The notations to be used by these models are presented as follows.

Notations:

i, *p*, *q*: Task index, $i \in I$
j: Workstation index, $j \in J$
r,s,k: Robot index, $r \in R$
t_{ir} : Operation time of task *i* by robot *r*.
P(*i*): Set of immediate predecessors of the task *i*.
Pa(*i*): Set of all predecessors of the task *i*.
S(*i*): Set of immediate successors of the task *i*.
Sa(*i*): Set of all successors of the task *i*.
CT: Cycle-time.

t_i^f : Operation time of task *i* by robot *r*.

x_{irj}: Binary variable. x_{irj} is equal to 1 when task *i* is operated by robot *r* on station *j*.
y_{ij}: Binary variable. y_{ij} is equal to 1 when task *i* is allocated to station *j*.
w_{rj}: Binary variable. w_{rj} is equal to 1 when robot *r* is allocated to station *j*.
v_{ir}: Binary variable. v_{it} is equal to 1 when task *i* is operated by robot r.
u_i: Binary variable
z_{pq}: Binary variable. z_{pq} is equal to 1 when task *p* is assigned earlier than task *q* on the same workstation.
d_{rs}: Binary variable. d_{rs} is equal to 1 when robot *r* is allocated to the former station than robot *s*.

The first model, referred to as Model 1, is developed by modifying the model reported in Miralles et al. (2008), where the worker assignment problem, which is similar to the robot allocation in this paper, is addressed. This model utilizes a three-index variable and a two-index variable to describe the task assignment and robot allocation as follows.

Minimize CT (1)

$\sum_{r\in R}\sum_{j\in J} x_{irj}=1\ \forall i \in \mathrm{I}$ (2)

$\sum_{r\in R} w_{rj} = 1\ \forall j \in J$ (3)

$\sum_{j\in J} w_{rj} = 1\ \forall r \in R$ (4)

$\sum_{r\in R}\sum_{j\in J} j \cdot x_{prj} - \sum_{r\in R}\sum_{j\in J} j \cdot x_{qrj} \leq 0\ \forall p \in P(q)$ (5)

$\sum_{i\in I}\sum_{r\in R} t_{ir} \cdot x_{irj} \leq CT\ \ \forall j \in J$ (6)

$\sum_{i\in I} x_{irj} \leq \psi \cdot w_{rj}\ \ \forall r \in R, j \in J$ (7)

The objective function in expression (1) minimizes the cycle-time. Equation (2) ensures that each task is assigned to a workstation and operated by a robot. Equation (3) and equation (4) guarantee that each workstation is equipped with a robot and each robot is allocated to a workstation respectively. Equation (5) addresses the precedence relationship ensuring that the successors of a task must be assigned to the same workstation or latter workstation. Equation (6) deals with cycle-time constraint and ensures that the total operation time of tasks on each workstation is less than or equal to the cycle-time. Finally, equation (7) ensures that a task must be operated by the robot allocated to the workstation to which the task is assigned.

The second model, referred to as Model 2, is built based on Li et al. (2016a) and utilizes two two-index variables to describe the task assignment and robot allocation as follows.

Minimize CT (8)

$\sum_{j\in J} y_{ij}=1\ \forall i \in \mathrm{I}$ (9)

$\sum_{r\in R} w_{rj} = 1\ \forall j \in J$ (10)

$\sum_{j\in J} w_{rj} = 1\ \forall r \in R$ (11)

$\sum_{j\in J} j \cdot y_{pj} - \sum_{j\in J} j \cdot y_{qj} \leq 0\ \forall p \in P(q)$ (12)

$t_i^f \leq CT\ \forall i \in \mathrm{I}$ (13)

$t_q^f - t_p^f + \psi(1 - y_{qj}) + \psi(1 - y_{pj}) \geq \sum_{r\in R} t_{qr} \cdot w_{rj}\ \ \forall p, q \in S(p), j \in J$ (14)

$$t_q^f - t_p^f + \psi(1 - y_{qj}) + \psi(1 - y_{pj}) + \psi(1 - z_{pq}) \geq \sum_{r \in R} t_{qr} \cdot w_{rj} \quad (15)$$

$$\forall p, q \in \{r | r \in I - (Pa(P) + Sa(P)) \text{ and } p < r\}, j \in J$$

$$t_p^f - t_q^f + \psi(1 - y_{pj}) + \psi(1 - y_{qj}) + \psi \cdot z_{pq} \geq \sum_{r \in R} t_{pr} \cdot w_{rj} \quad (16)$$

$$\forall p, q \in \{r | r \in I - (Pa(P) + Sa(P)) \text{ and } p < r\}, j \in J$$

$$t_i^f + \psi(1 - y_{ij}) \geq \sum_{r \in R} t_{ir} \cdot w_{rj} \quad \forall i \in \mathrm{I}, j \in J \quad (17)$$

Similarly to the work in Miralles et al. (2008), Equation (8) optimizes the cycle-time. Equation (9) ensures that each task is allocated to a workstation. Equation (10) and Equation (11) deal with the robot allocation. Equation (12) addresses the precedence relationship. Equation (13) addresses the cycle-time constraint by ensuring that all tasks are finished within the cycle-time. Equation (14-16) calculates the completion times of the tasks. Equation (14) ensures that task q can be operated only when its predecessor p has been completed. This equation is reduced to $t_q^f - t_p^f \geq \sum_{r \in R} t_{qr} \cdot w_{rj}$ when task q is the successor of task p and they are allocated to the same workstation. Equations (15-16) handle the situation in which two tasks have no precedence relationship. If task p is assigned before task q on the same workstation, Equation (15) is reduced to $t_q^f - t_p^f \geq \sum_{r \in R} t_{qr} \cdot w_{rj}$; otherwise, Equation (16) is reduced to $t_p^f - t_q^f \geq \sum_{r \in R} t_{pr} \cdot w_{rj}$. Equation (17) guarantees that the completion time of a task is equal to or larger than its operation time.

The third and fourth models (referred to as Model 3 and Model 4) are modified from Borba and Ritt (2014) who solve worker assignments. The main idea behind these models is assigning tasks to robots.

Minimize CT (18)

$\sum_{i \in I} t_{ir} \cdot v_{ir} \leq \mathrm{CT}\ \forall r \in \mathrm{R}$ (19)

$\sum_{r \in R} v_{ir} = 1\ \forall i \in \mathrm{I}$ (20)

$d_{rs} \geq v_{pr} + v_{qs} - 1\ \forall p \in P(q), r, s \in \mathrm{R}$ and $r \neq s$ (21)

$d_{rs} \geq d_{rk} + d_{ks} - 1\ \forall\ r, s, k \in \mathrm{R}, r \neq s, r \neq k$ and $s \neq k$ (22)

$d_{rs} + d_{sr} \leq 1\ \forall\ \ r, s \in \mathrm{R}, r \neq s$ (23)

In Model 3, Equation (18) also minimizes the cycle-time. Inequality (19) addresses the cycle-time constraint ensuring the total operation time of tasks by robot r is less than or equal to the cycle-time. Equation (20) guarantees that each task is executed by exactly one robot. Equation (21) handles precedence constraints and ensures that robot r must precede robot s when task p is assigned to robot r and precedes the task q assigned to robot s. Equation (22) ensures that robot r precedes robot s when robot r

precedes robot k and robot k precedes robot s. Equation (23) ensures the anti-symmetry of the robot dependencies since robot r must be allocated before robot s or after robot s. To increase the search speed, the continuity constraint expressed in Equation (24) is added in Model 4 along with Equations (18-23). Equation (24) ensures that the task p should be assigned to robot r when task p is the successor of task i and the predecessor of task q and task i and task q are assigned to the same robot r.

$$v_{pr} \geq v_{ir} + v_{qr} - 1 \; \forall \; r \in \mathrm{R}, i, p, q \in \mathrm{I}, p \in Sa(i) \text{ and } p \in Pa(q) \qquad (24)$$

The four models for solving the RALB-II problem are presented in the form of mixed-integer linear programming models and encoded in CPLEX solver to achieve optimal or near-optimal solutions. These models are evaluated, and the findings are presented in Section 4.

3. Proposed methodologies

Since the RALB-II problem consists of two interrelated sub-problems, local search methods and co-evolutionary algorithms might be a good choice to produce promising results (Li et al., 2017a). This research utilizes the simulated annealing (SA) algorithm as a local search method to solve the RALB-II problem. SA is selected mainly because SA has no complex operators and is much simpler in implementation when compared to other evolutionary algorithms (Rabbani et al., 2015). SA has shown promising results for solving many optimization problems. For instance, SA has achieved promising results for different types of assembly line balancing problems (Erel et al., 2001, Baykasoglu, 2006, Özcan and Toklu, 2009, Özcan, 2010, Roshani et al., 2012, Fathi et al., 2016, Jayaswal and Agarwal, 2014, Roshani and Nezami, 2017) and mixed-model assembly line balancing and sequencing (Mosadegh et al., 2012, Hamzadayi and Yildiz, 2012, Hamzadayi and Yildiz, 2013). SA especially shows superior performance over the co-evolutionary genetic algorithm in Mosadegh et al. (2012) in optimizing two interrelated sub-problems simultaneously.

This paper adopts two types of SA methodologies: original simulated annealing (SA) and restarted simulated annealing (RSA). In the RSA method, a restart mechanism is developed to replace the incumbent temperature with a new temperature emphasizing exploitation. Two problem-specific improvements are also developed to enhance the SA and RSA: an iterative mechanism for cycle-time update and a new objective to select the solution with less critical workstations detailed in Section 3.1. In the following subsections, the encoding scheme and decoding procedure along with two problem-specific improvements are introduced in Section 3.1, and the two proposed methodologies, SA and RSA, are illustrated in Section 3.2.

3.1 Encoding scheme and decoding procedure

Based on the contributions reported in the following researches (Gao et al., 2009, Li et al., 2016a, Li et al., 2017a), this research proposes two vectors for encoding: task permutation vector and robot allocation vector. Task permutation vector is a $1 \times Nt$

vector denoting the sequence of the tasks being allocated and that the tasks in the former position of the task permutation vector should be assigned first. The robot allocation vector is a $1 \times Ns$ vector, each element denotes the allocation of a robot to a workstation. Suppose that the element in the j^{th} position of this vector is *r,* robot *r* is allocated to workstation *j*. Two examples for the task permutation vector and robot allocation vector are as follows. In the task permutation vector, task 1 has the highest priority and should be assigned first, whereas task 24 should be assigned last. In the robot allocation vector, robot 5, robot 3, robot 1, robot 6, robot 2, and robot 4 should be allocated to workstation 1, workstation 2, workstation 3, workstation 4, workstation 5, and workstation 6 respectively.

Task permutation vector: 1, 2, 3, 4, 5, 6, 8, 7, 12, 15, 9, 11, 13, 14, 20, 21, 16, 17, 18, 23, 25, 19, 22, 10, 24.

Robot allocation vector: 5, 3, 1, 6, 2, 4.

To transfer the two vectors into a feasible solution, a decoding procedure is necessary, where the determination of the initial cycle-time is a non-negligible issue for RALB-II problems. Following Li et al. (2017b), this research proposes an iterative mechanism for cycle-time updating. Figure 2 and Figure 3, respectively, present a detailed iterative mechanism and decoding procedure. In the decoding procedure, each former workstation is assigned as much workload as possible based on the task permutation, and the last workstation endures all the remaining workload. The largest value among the completion times of the workstation is regarded as the achieved cycle-time by an individual. It should be noted that this decoding procedure differs from Gao et al. (2009) as this method allows the allocation of all the remaining workload to the last workstation even when these remaining tasks cannot be finished within the provided initial cycle-time.

Regarding the iterative mechanism, the initial cycle-time is set to a large value at first and this cycle-time is iteratively reduced. In this iterative mechanism, each individual is decoded using *CT-1* as the initial cycle-time at first. If the completion time of the tasks on the last workstation is not bigger than *CT-1*, an individual with a smaller cycle-time is achieved. If no better cycle-time is achieved, this individual is decoded using *CT* as the initial cycle-time. This method guarantees that the *CT* and CT_{Best} gradually decrease, where CT_{Best} is the best cycle-time obtained so far. When the CT_{Best} is reduced, all the individuals are re-decoded using *CT* as the initial cycle-time, and the incumbent fitness values are replaced with this newly achieved ones. This technique ensures that all the individuals are evaluated using the same initial cycle-time. Notice that the proposed iterative mechanism executes decoding procedure only twice to achieve the fitness for one individual.

In Levitin et al. (2006), the reported procedure calculates the lower bound of the cycle-time as the initial cycle-time and increases this initial cycle-time until all tasks can be allocated within the provided initial cycle-time. This method needs to execute decoding procedure many times to obtain the proper initial cycle-time. The procedure reported in Gao et al. (2009) also executes the decoding procedure several times using the bisection method. The method proposed here avoids the possible drawbacks of the

published research in searching for the proper cycle-time and executes the decoding procedure only utilizing the lowest achievable cycle-time.

In the preliminary experiments for solving RALB-II problems, it is observed that many solutions have the same cycle-time and the utilized cycle-time as the optimizing objective is unable to distinguish between these. Hence, on the basis of Gao et al. (2009), this research proposes a new objective expressed in Equation (25), where *Ncs* is the number of critical workstations and a workstation is regarded as a critical workstation when the completion time of this workstation is equal to the initial cycle-time. In our experiments, the second part of the equation 0.1×Ncs is usually less than 1.0, and therefore, the second part takes effect only when the individuals have the same cycle-time. Note that the number 0.1 can be replaced with some other small positive numbers as long as it ensures that second part takes effect only when the individuals have the same cycle-time.

$$\text{Minimize } CT+0.1\times Ncs \tag{25}$$

Iterative mechanism:

% Cycle-time initialization

Step 1: Set the initial cycle-time to a large value as $CT = 2 \cdot \sum_{i\in I}\sum_{r\in R} t_{ir}/(Nt \cdot Ns)$ and CT_{Best} is set as $CT - 1$, where CT_{Best} is the best cycle-time obtained so far.

% Cycle-time iteration during evolution process

Step 2:

For each individual **do**

Step 2.1: Achieve the solutions using CT-1 as the initial cycle-time. If the completion time of the tasks on the last workstation for one individual is not bigger than CT-1, $CT_{Best} = CT - 1$, $CT = CT_{Best}$, the achieved fitness is regarded as the fitness of this individual and continue. Otherwise, go to Step 2.2.

Step 2.2: Achieve one solution using CT as the initial cycle-time and the achieved fitness is regarded as the fitness of this individual.

Endfor

Step 3:

If (CT_{Best} is reduced)

Re-decode all the incumbent individuals using CT as the initial cycle-time and replace the incumbent fitness values with this newly achieved one.

Endif

Step 4: Achieve the new individuals in the algorithm's evolution and execute Step 2 and Step3 until the termination criterion is satisfied.

Figure 2. Proposed iterative mechanism

Decoding procedure:

Determine the initial cycle-time using iterative mechanism.

While (Some tasks are still unallocated)

Open a new workstation;

Do

When the current workstation is not the last workstation, obtain the assignable tasks whose predecessors have been allocated and the completion times are not larger than CT; otherwise, obtain the assignable tasks whose predecessors have been allocated.

Allocate the task on the former position of the task permutation to the current workstation;

Update the remaining capacity of the current workstation;

Until no assignable task exists.

End while

Set the largest value among the completion times of the workstation as the achieved cycle-time.

Figure 3. Proposed decoding procedure

3.2 Proposed simulated annealing algorithms

SA algorithm for RALB-II problem:

Input parameter values: T_0, α and N;

% Algorithm initialization

n:=0, T:=T_0;

Generate an initial solution S;

% Algorithm evolution

Do

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

Achieve a neighbor solution S' using neighbor operator;

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

If $(\Delta \leq 0)$ $S \leftarrow S'$;

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

//Rand is a random number within [0,1]

Endfor

$T = T \times \alpha$

Until (Termination criterion is met)

Figure 4. Procedure of SA algorithm

This section provides the details of the proposed SA and RSA, where the procedure of original SA is first illustrated in Figure 4. This algorithm starts with three input parameters: the initial temperature (T_0), the cooling rate (α), and iteration times before the temperature update (N). Subsequently, an initial individual is generated, and a main loop is repeated until the termination criterion is met. Within the loop, new neighbor solutions are obtained N times and then the current temperature is updated. For each newly generated neighbor solution, it replaces the incumbent one when it achieves a better fitness or with a probability of $exp^{-\Delta/(T\times Fit(S))}$ when it achieves a worse fitness. It is clear that SA to some extent allows for the acceptance of a worse solution to replace the incumbent one, but the probability of accepting the worst solutions decreases during algorithms evolution. Though SA has some ability to escape from local optima, there is still a risk that it might be trapped into local optima. During the preliminary experiment, this issue was observed for especially small-size problem instances. Hence, this research improves the original SA by embedding the restart mechanism, resulting in the RSA method. The general procedure of RSA is presented in Figure 5, and its procedure is similar to that of SA. Apart from the original three parameters in SA, RSA introduces two more parameters: restart temperature (T_R) and restart time (RT) before replacing the current temperature with T_R. The rationality of this restart mechanism is that the current temperature T is replaced with the T_R when no improvement of the best fitness is achieved for consecutive T_R times. This method increases the probability of accepting worse solutions and thus helping the algorithm to escape from local optima. It is to be noted that T_R and RT are critical parameters that must be carefully determined. A large value of T_R might result in reduced intensification whereas RSA with a low value of T_R might achieve the same results as the original SA.

RSA algorithm for RALB-II problem:

Input parameter values: T_0, α, N, T_R, and RT;

% Algorithm initialization

n:=0, T:=T_0, rt :=0;

Generate an initial solution S;

% Algorithm evolution

Do

  NewBest=0; //Check whether new best fitness is achieved

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

    Achieve a neighbor solution S' using neighbor operator;

    Calculate $\Delta = \mathrm{Fit}\left(S'\right) - \mathrm{Fit}(S)$;

    If ($\Delta \leq 0$) $S \leftarrow S'$;

      If (New best cycle-time is achieved) *NewBest*=1;

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

Endfor
If (*NewBest*!=0) *rt*:=0;
Else *rt*:=*rt* +1;
% Restart mechanism
If (($rt \geq RT$) and ($T < T_R$)) $T := T_R$;
Else $T = T \times \alpha$
Until (Termination criterion is met)

Figure 5. Procedure of RSA algorithm

The quality of the initial solution and the neighbor operator have an important effect on the final performance of the SA and RSA. In case of initialization, both algorithms utilize the ranked positional weight heuristic, which has been used by many researchers (Khorasanian et al., 2013) to achieve the initial task permutation. In general, the ranked positional weight of task *i* is the sum of the operation time of task *i* and the operation times of all the successors of task *i*. However, since the operation times of a task by robots are different from each other, this research utilizes the average value of the operation times by robots when utilizing this heuristic. It should be noted that most robots need operation times for the 'difficult' tasks are more and operation times for the 'easy' tasks are short. The average operation time of the 'difficult' tasks are usually larger than the 'easy' tasks, and hence utilizing "average times" make tasks with larger operation times and more successors have higher priorities with a larger probability. The robot allocation vector is randomly generated. With regards to the neighbor operator, this research proposes an insert and a swap operator for both the task permutation vector and robot allocation vector based on Li et al. (2017a). Specifically, a random number between [0, 1] is generated first. If this number is less than 0.5, the task permutation vector is selected, otherwise the robot allocation vector is selected. Once the vector is selected, one of the insert operators or swap operators is selected to modify the selected vector with 50 percentage probability. It should be noted that in the long run both vectors will be selected for almost half of the iteration times, and the insert operator or swap operator will be utilized to modify each vector for almost a quarter of the iteration times.

4 Numerical example

To clarify the proposed methods for solving RALB-II, this section presents a numerical example. The example has 25 tasks and six workstations equipped with six robots, and the precedence relationship and operation times of tasks by robots are presented in Table 1. In the table, the first column presents the task number and column two describes the precedence relationship. The remaining columns provide the details of the operation times by robots, and it is observed that the operation times of a task by robots can differ. For the precedence constraint, one task cannot be operated only all its predecessor have been completed. And the operation times of one task depends on the allocated robot. For instance, if task 1 is operated by robot 1, the corresponding time is 87. Nevertheless, this operation time is reduced to 44 if this task is operated by robot 5.

Table 2 exhibits the detailed task assignment and robot allocation of the achieved

solution using the proposed methodology. The second row shows the task assignment, for example, task 1, task 2, task 3, and task 4 are assigned to workstation 1. The third row presents the robot allocation, for example, robot 5, robot 3, and robot 1 are allocated to workstation 1, workstation 2, and workstation 3 respectively. The fourth row calculates the total operation time of tasks on each workstation. Specifically, for workstation 1, robot 5 is allocated to operate tasks 1, 2, 3, and 4, and the total operation time is calculated as 44+53+61+55=213. For workstation 2, robot 3 is allocated to operate tasks 5, 6, 8, 7, and 12, and the total operation time is 28+51+44+33+50=206. The largest value of the total operation times on a workstation is regarded as the achieved cycle-time presented in the last row. In addition, the line efficiency of this assembly line is nearly 96.24%, and the achieved task assignment and robot allocation are quite effective.

Table 1 Precedence relationship and operation times of tasks by robots

Tasks	Successors	Operation times by robots					
		Robot 1	Robot 2	Robot 3	Robot 4	Robot 5	Robot 6
1	2	87	62	42	60	44	76
2	3	67	47	42	45	53	100
3	4	82	58	54	40	61	60
4	5,8	182	58	62	60	55	100
5	6	71	47	28	57	62	76
6	7,10	139	48	51	73	61	117
7	11,12	98	99	44	49	59	82
8	9,11	70	40	33	29	36	52
9	10,13	60	114	47	72	63	93
10	-	112	67	85	63	49	86
11	13	51	35	41	44	85	69
12	15	79	39	50	80	67	95
13	14	57	47	56	85	41	49
14	16,19,20	139	65	40	38	87	105
15	17,22	95	63	42	65	61	167
16	18	54	48	51	34	71	133
17	18,23	71	28	35	29	32	41
18	25	112	29	49	58	84	69
19	22	109	47	38	37	52	69
20	21,25	63	45	39	43	36	57
21	22,24	75	68	45	79	84	83
22	-	87	36	74	29	82	109
23	25	58	36	55	38	42	107
24	-	44	54	23	21	36	71
25	-	79	64	48	35	48	97

Table 2 Detailed task assignment and robot allocation

	Workstation 1	**Workstation 2**	**Workstation 3**	**Workstation 4**	**Workstation 5**	**Workstation 6**

Task assignment	1, 2, 3, 4	5, 6, 8, 7, 12	15, 9, 11	13, 14, 20	21, 16, 17, 18, 23	25, 19, 22, 10, 24
Robot allocation	5	3	1	6	2	4
Total operation times	213	206	206	211	209	185
Cycle-time	213					

5. Computational study

This section first presents the details of the experimental design, and later presents the findings of the evaluation of the proposed models and finally reports the comparative campaign among implemented algorithms as well as the statistical analysis.

5.1 Design of experiments

To evaluate the proposed models and algorithms, this study conducts two comparative studies on models and algorithms respectively. Both comparative studies use the benchmark problems presented in Gao et al. (2009) for testing. This benchmark set contains eight sets of problems corresponding to eight precedence diagrams: P25, P35, P53, P70, P89, P111, P148, and P297, where the symbol P is the abbreviation of the problem and the numbers denote the task numbers. In addition, each problem contains four cases with different workstations, leading to a total of 32 tested cases. In this research, these tested problems are divided into two categories: small-size problem instances including P25, P35, and P53 and large-size problem instances including P70, P89, P111, P148, and P297.

Regarding the model evaluation, only P25, P35, P53, and P70 or a total of 16 cases are solved by the four models since the CPLEX solver cannot achieve optimality of very large-size problems in acceptable CPU time. The execution terminates when the optimal solution is achieved or elapsed computation time reaches 3600 seconds (s). All the models are solved using CPLEX solver of General Algebraic Modeling System 23.0 and they are tested on a set of personal computers equipped with Intel(R) Core(TM) i7-4790S CPU @ 3.20 GHZ.

With respect to the algorithm evaluation, five other well-known metaheuristic algorithms are modified and re-implemented for the comparative study to tackle all the datasets. These algorithms are taken from literature on algorithms recently used to solve RTALB problems (Li et al., 2016a, Li et al., 2017a) in which both task permutation vector and robot allocation vector are applied. These methods include genetic algorithm (Gao et al., 2009) (GA), particle swarm optimization (Li et al., 2016a) (PSO), discrete cuckoo search (Li et al., 2017a) (DCS), and artificial bee colony (Tang et al., 2016) (ABC1 and ABC2). It is to be noted that there are potentially many variants of an algorithm, which might lead to ambiguous results. To avoid this situation and have a better investigation of the performances of the algorithms, some problem-specific improvements are omitted and the main operators of these re-implemented algorithms are set similar to those presented in Li et al. (2017a). All the tested algorithms share the same neighborhood structures as shown in

Section 3.2. Apart from this, the main operators of the tested algorithms are presented in Table A1 of the appendix.

Before executing the algorithms, there is a need for proper determination of the termination criterion and the parameters of these algorithms. Based on the procedure followed in Li et al. (2016a), Li et al. (2017a), and Nilakantan et al. (2017), this research sets the elapsed CPU time as the termination criterion, which is calculated as $\mathrm{Nt} \times \mathrm{Nt} \times \tau$ milliseconds, where τ is a parameter which is set to 10, 20, and 30 respectively. These termination criteria provide more CPU time to large-size problem instances and make it possible to observe the performances of the algorithm under different elapsed CPU times. For parameter calibration, this research utilizes the full factorial design similar to the ones reported in Li et al. (2016a), Li et al. (2017a), and Li et al. (2017b). The initial levels of the parameters are determined based on the published literature, and they are further reduced by fixing the values of other parameters. Since the best parameter combination on small-sized problem instances might greatly differ from those on large-sized problem instances for some algorithms, this research calibrates the parameters for both sets of problems respectively. Taking the large-sized problem as an example, the largest-sized case with 297 tasks and 29 workstations is solved by each parameter combination 10 times with the termination criterion of $\mathrm{Nt} \times \mathrm{Nt} \times 10$ milliseconds. After completing all the experiments, the relative percentage deviation (RPD) is calculated as the response variable using expression (26). In this expression, CT_{some} is the yield cycle-time by one parameter combination in one time execution, and CT_{Best} is the smallest cycle-time by all parameter combinations in 10 iterations.

$$RPD = 100 \cdot (CT_{Some} - CT_{Best})/CT_{Best} \tag{26}$$

After transferring these cycle-times, the well-known multifactor analysis of variance (ANOVA) technique is applied to select the best parameter values based on the method adopted in Li et al. (2017b). The selected values of parameters are presented in Table A2 of the appendix. Due to space constraint, the detailed ANOVA procedure is not presented in the paper, but this information will be uploaded in Research Gate for readers reference.

5.2 Model evaluation

This section evaluates the four models and the achieved cycle-times (Results), consumed CPU times (Time), and the number of the executed nodes (Nodes) are presented in Table 3 and Table 4. More detailed results are presented in the Table A3-1 and Table A3-2 in the appendix. From Table 3 and Table 4, it can be stated that optimality is achieved for some problem instances when the elapsed CPU time is less than 3600s. From Table 3, it is observed that Model 1, Model 2, Model 3, and Model 4 could achieve optimality for nine cases, zero cases, seven cases, and six cases out of sixteen cases respectively. It is to be noted that the published model in Gao et al. (2009) is able to achieve the optimality only for two smallest cases, P25 with three and four workstations. It is clear that Model 1, Model 3, and Model 4 outperform the

published non-linear model in Gao et al. (2009). It also can be seen that Model 2 achieves no optimality or cannot prove the optimality for the tested problems in the limited CPU time allotted. The reason lies in that Model 2 considers the detailed task sequence on each workstation, resulting in larger CPU times. Regarding the elapsed CPU time to achieve optimality by Model 1, Model 2, and Model 3, Model 1 requires the smallest CPU time, Model 3 consumes the second smallest CPU time, and Model 4 requires the largest CPU time, which is confirmed by a Wilcoxon signed rank test with $p<0.01$ denoting that there is statistical difference. It is evident that Model 2 is the worst performer, but Model 2 reports good performance in solving cases with a larger task number and a larger workstation number. Specifically, Model 2 is the best performer for P53 with 10 and 14 workstations and P70 with 14 and 19 workstations.

Table 3 Achieved results by the proposed models

Problem	Ns	Model 1		Model 2		Model 3		Model 4	
		Results	Time (s)	Results	Time (s)	Results	Time (s)	Results	Time (s)
P25	3	**503**	0.85	503	3600	**503**	0.44	**503**	1.23
	4	**327**	2.28	327	3600	**327**	5.03	**327**	14.13
	6	**213**	55.46	214	3600	**213**	243.59	**213**	83.16
	9	123	3600	125	3600	130	3600	123	3600
P35	4	**449**	6.51	456	3600	**449**	5.04	**449**	70.53
	5	**344**	12.40	348	3600	**344**	21.97	**344**	61.25
	7	**222**	224.69	236	3600	**222**	466.32	**222**	632.80
	12	130	3600	128	3600	125	3600	135	3600
P53	5	**554**	55.27	560	3600	**554**	1453.93	560	3600
	7	**320**	1461.13	329	3600	347	3600	377	3600
	10	345	3600	274	3600	347	3600	407	3600
	14	288	3600	182	3600	529	3600	311	3600
P70	7	**448**	3091.40	507	3600	458	3600	467	3600
	10	363	3600	352	3600	361	3600	308	3600
	14	577	3600	296	3600	1383	3600	888	3600
	19	1023	3600	185	3600	1395	3600	706	3600

In Table 4, it is observed that Model 1 has the smallest values of the nodes among all the models to achieve optimality for P25 with three and four workstations and P35 with four workstations. Model 4 has the smallest values of the nodes to achieve optimality for P25 with six workstations and P35 with five and seven7 workstations. Though the difference of Model 1 and Model 4 on the executed nodes is not clear, it is, however, clear that these two models need many fewer nodes to achieve optimality than Model 2 for P25 with three, four, and six workstations and P35 with four, five, and seven workstations. If one compares the executed nodes per seconds of these models, Model 4 executes the fewest nodes per second and the difference between Model 1, Model 2 and Model 3 is not clear. In summary, Model 1 shows advantages over the others in the number of cases solved to optimality, but no model outperforms the others for all cases.

Table 4 Executed nodes of the proposed models

Problem	Ns	Model 1		Model 2		Model 3		Model 4	
		Nodes	Time (s)	Nodes	Time (s)	Nodes	Time (s)	Nodes	Time (s)
P25	3	65	0.85	16477823	3600	451	0.44	68	1.23
	4	611	2.28	7079287	3600	8198	5.03	978	14.13
	6	15991	55.46	7485976	3600	367636	243.59	3622	83.16
	9	537022	3600	5025133	3600	459797	3600	155859	3600

P35	4	2244	6.51	5147710	3600	13552	5.04	3634	70.53
	5	1847	12.40	4370414	3600	27924	21.97	1327	61.25
	7	36812	224.69	799343	3600	263998	466.32	15559	632.80
	12	54749	3600	1310261	3600	73719	3600	8022	3600
P53	5	7216	55.27	3144291	3600	1182648	1453.93	2037	3600
	7	118910	1461.13	554717	3600	581730	3600	600	3600
	10	70791	3600	366851	3600	46460	3600	144	3600
	14	21798	3600	137086	3600	6654	3600	45	3600
P70	7	250196	3091.40	147177	3600	549926	3600	496	3600
	10	43502	3600	25672	3600	41297	3600	149	3600
	14	13091	3600	7507	3600	7090	3600	46	3600
	19	7156	3600	61061	3600	912	3600	8	3600

5.3 Algorithm evaluation

This section exhibits the comparative study of the algorithms to test the performance of the two proposed methodologies. All the implemented algorithms solve the aforementioned benchmark cases for 20 iterative times under three termination criteria ($\tau = 10, 20, 30$). After completing all the experiments, the relative percentage deviation or RPD is again applied to transfer the achieved cycle-times. Since there are 32 cases solved 20 times under three termination criteria, each algorithm has 1920 RPD values. Table 5 presents the average RPD values for each problem, each average RPD value corresponding to the average value of four cases in 20 time repetitions or 80 RPD values. In this table, the three values for parameter τ denote three termination criteria and the symbol Avg means the overall RPD value of all tested cases, that is, the average value of 640 RPD values.

It can be seen in Table 5 that RSA is the best performer in terms of the overall RPD values under all the three termination criteria. RSA is followed by SA when $\tau = 10, 20$ and by ABC1 when $\tau = 30$. Specifically, RSA yields the best performance for P25, P35, P89, P111, P148, and P297 when $\tau = 10$ and for P25, P35, P53, P111, P148, and P297 when $\tau = 20, 30$. Especially, RSA and SA are the two best performers for the three largest-size problems (P111, P148, and P297) under the three termination criteria. If one sorts the tested algorithms in increasing order of the overall RPD values, the sequence is: RSA, SA, ABC2, ABC1, DCS, GA, and PSO when $\tau = 10$. This sequence is modified to RSA, SA, ABC1, ABC2, DCS, GA and PSO when $\tau = 20$ and RSA, ABC1, SA, ABC2, DCS, GA, and PSO when $\tau = 30$. These results suggest that the proposed RSA performs best among these compared problems and the proposed SA also shows promising results in solving large-size problems.

Table 5 Achieved average RPD values by tested methodologies

Problem	Average relative percentage deviation							CPU time(s)
	GA	PSO	DCS	ABC1	ABC2	SA	RSA	
τ = 10								
P25	0.92	1.37	0.64	0.10	0.35	0.48	**0.10**	6.3
P35	4.22	6.46	3.77	1.22	3.05	3.00	**0.82**	12.3
P53	3.55	5.75	2.76	**2.23**	2.91	2.82	2.36	28.1
P70	5.30	12.37	4.04	**2.80**	3.05	3.62	3.19	49.0
P89	3.71	9.83	2.85	2.28	2.46	2.54	**2.14**	79.2
P111	6.54	17.01	4.93	4.56	3.76	3.11	**2.95**	123.2
P148	8.02	20.72	5.91	6.14	4.45	3.23	**2.91**	219.0
P297	9.43	22.05	6.05	7.75	5.34	3.15	**3.02**	882.1
Avg.	5.21	11.95	3.87	3.38	3.17	2.75	**2.19**	

τ = 20								
P25	0.83	1.27	0.48	0.03	0.28	0.48	**0.00**	12.5
P35	3.92	5.78	3.55	1.00	2.77	3.00	**0.42**	24.5
P53	3.40	5.22	2.71	2.04	2.77	2.77	**1.92**	56.2
P70	4.98	11.39	3.75	2.25	2.87	3.54	2.79	98.0
P89	3.42	9.05	2.61	1.79	2.30	2.43	1.89	158.4
P111	5.86	15.99	4.52	3.40	3.32	2.85	**2.46**	246.4
P148	7.02	19.41	5.27	4.61	3.63	2.67	**2.29**	438.1
P297	8.17	21.32	5.13	6.14	4.00	2.38	**2.19**	1764.2
Avg.	4.70	11.18	3.50	2.66	2.74	2.51	**1.75**	
τ = 30								
P25	0.83	1.27	0.46	0.03	0.20	0.48	**0.00**	18.8
P35	3.91	5.50	3.28	0.82	2.63	3.00	**0.31**	36.8
P53	3.31	5.00	2.68	1.98	2.64	2.71	**1.82**	84.3
P70	4.67	10.95	3.59	1.96	2.80	3.46	2.60	147.0
P89	3.25	8.68	2.49	1.55	2.24	2.39	1.63	237.6
P111	5.42	15.27	4.37	2.95	3.12	2.70	**2.20**	369.6
P148	6.65	18.76	4.83	3.89	3.30	2.41	**2.07**	657.1
P297	7.56	20.93	4.65	5.26	3.57	1.90	**1.84**	2646.3
Avg.	4.45	10.80	3.29	2.31	2.56	2.38	**1.56**	

*Best average RPD values in bold.

To check whether the observed difference is statistically significant, this research also carries out the multifactor ANOVA test with the algorithm type and elapsed CPU time ($\tau = 10, 20, 30$) as two factors. Since there is a big difference in the performance of an algorithm on different problems, the proposed ANOVA test utilizes the average RPD value of 32 cases in a single run as the response variables based on the work done in Li et al. (2017b). There are 20 average RPD values for each algorithm under one termination criterion. After checking the fulfillment of the three main hypotheses required for ANOVA (independence of the residuals, homogeneity of variance, and normality), the ANOVA test is conducted. The results of the analysis suggest a significant statistical difference between algorithms and elapsed CPU time. For brevity's sake, the detailed ANOVA table is not presented, but the mean plots for the interaction between algorithm type and elapsed CPU time are presented in Figure 6. In this figure, the numbers 10, 20, and 30 indicate the three values of parameter τ in the termination criteria.

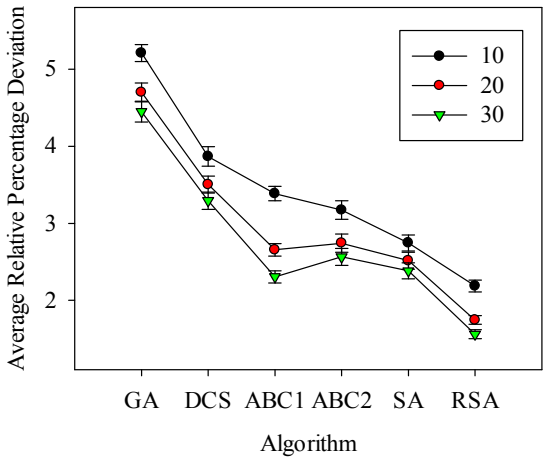

Figure 6. Mean plots with Tukey HSD confidence intervals for the interaction between algorithm type and elapsed CPU time ($\tau = 10, 20, 30$)

It is evident from Figure 6 that RSA is the best performer under all three termination criteria, SA is the second-best performer when $\tau = 10, 20$, and ABC1 is the second-best performer when $\tau = 30$. In short, the analysis results correspond with those presented in Table 5. It also can be seen that there are no overlapping confidence intervals between RSA and other methods. It can be seen that the overlapping confidence intervals denote that the observed difference between the two overlapped means is statistically insignificant. Hence, it is sufficient to say that the proposed RSA is statistically better than benchmark methods.

This paper also presents the best achieved cycle-time in 20 times iterations by the implemented algorithms in Table 6. In this table, OPT indicates the achieved optimal cycle-times by the developed models. The abbreviation hGA means the best cycle-times by a hybrid genetic algorithm in Gao et al. (2009), and the values of these cycle-times are taken from the literature directly. It should be noted that the cycle-times by hGA, to the authors' best knowledge, are the current best published results regarding solutions to the considered RALB-II problem.

From Table 6, it is seen that the cycle-times for 23 cases out of 32 cases are updated, and especially the cycle-times for all the 20 large-size cases are updated. Among the remaining nine cases, eight cases are solved optimally by hGA and hence no improvement can be achieved. To be specific, among these updated cycle-times, GA achieves two cases, PSO achieves one case, ABC1 achieves five cases, ABC2 achieves seven cases, SA achieves seven cases, and RSA achieves 16 cases. Clearly, RSA is again the best performer in terms of the updated cycle-times. To evaluate the performance of the algorithms in updating the cycle-time in all cases, this research calculates the average improvement rate of 32 cases, and the improvement in one case by an algorithm is calculated utilizing $100 \cdot (CT_{hGA} - CT_{Some})/CT_{hGA}$, where CT_{hGA} is the best cycle-time by hGA or current best cycle-time, and CT_{Some} is the yield cycle-time by one implemented algorithm for the same case. The calculated average improvement rates by the seven tested algorithms are as follows: -0.33% by GA, -6.26% by PSO, -1.56% by DCS, 0.57% by ABC1, 0.92% by ABC2, 1.20% by SA, and 1.34% by RSA. As we can see, RSA and SA achieve the largest and second-largest average improvement rates, and ABC2 and ABC1 achieve the third- and fourth-largest average improvement rates. The other three methods, cannot achieve positive average improvement rate, indicating, on average, worse results than the known best results. It should be noted that hGA utilizes a strong local search procedure whereas the implemented method utilizes no local search procedure and is much simpler. Despite the simplicity of the tested methods, four of them achieve positive average improvement rates. In summary, these computational results validate the superiority of the proposed RSA and SA in terms of the average improvement rate.

6. Conclusion and future research

Modern assembly line systems utilize robots by replacing human workers to improve quality and increase flexibility. This paper studies type II robotic assembly lines and the associated line-balancing problem with cycle-time minimization criterion. To solve this problem, this paper formulates four mixed-integer linear

programming models to tackle small-size problem instances for optimality and two metaheuristic methodologies: original simulated annealing algorithm (SA) and restarted simulated annealing algorithm (RSA) for solving large-size problem instances in an acceptable computational time. The restarted method utilizes the restart mechanism to replace the incumbent temperature with a new temperature to emphasize exploitation. The two methods employ iterative mechanisms for cycle-time update and new objective to preserve the solution with fewer critical workstations.

All the models achieve optimality for nine out of 16 cases within 3600s whereas the published non-linear model in Gao et al. (2009) achieves the optimality only for two cases within acceptable CPU time. Among the tested models (Model 1, Model 2, Model 3, and Model 4), Model 1 is the best performer in terms of the number of cases solved to optimality within the given CPU time. To evaluate the developed methods, five other metaheuristic methodologies are re-implemented: genetic algorithm, particle swarm optimization algorithm, cuckoo search algorithm, and two types of artificial bee colony algorithms. A comprehensive study is conducted to solve 32 benchmark instances using three termination criteria. Computational results along with statistical analysis using multifactor analysis of variance demonstrate that the proposed methods produce promising results and the proposed RSA is the best performer among those tested methods. The implemented algorithms are capable of achieving smaller cycle times that in turn helps in increasing the line efficiency, leading to increasing product output, by achieving 23 new upper bounds out of 32 benchmark cases. The proposed models will help production managers in the decision making and these models can be utilized for designing/redesigning robotic assembly lines that are efficient in terms of minimizing cycle time.

Future research avenues that stem out from the extensions of the solved problem include mixed-model robotic assembly line and mixed-model robotic assembly line balancing and sequencing. Since the real-world industrial contexts are much more complex than the typical problem addressed in literature, this research will assist in reducing the gap between research and real-world application. For instance, there might be constraints such as a task that cannot be operated by some robots or a robot cannot be allocated to some workstations. It would also be interesting to research the collaboration between humans and robots and its impact on the line-balancing problem, since this configuration is more relevant in the factories of the future.

Table 6 Best cycle-times by tested methodologies

Problem	Ns	OPT	hGA(Gao et al., 2009)	GA	PSO	DCS	ABC1	ABC2	SA	RSA
P25	3	503	503	503	503	503	503	503	503	503
	4	327	327	327	327	327	327	327	327	327
	6	213	213	213	213	213	213	213	213	213
	9	-	123	**121**	**121**	125	**121**	**121**	**121**	**121**
P35	4	449	449	449	449	450	449	449	449	449
	5	344	344	344	344	375	344	344	344	344
	7	222	222	222	222	222	222	222	222	222
	12	-	113	112	118	112	**111**	**111**	112	**111**
P53	5	554	554	558	560	560	556	556	556	554
	7	320	320	320	320	320	320	320	320	320
	10	-	230	230	248	239	230	239	239	238
	14	-	162	162	169	165	159	160	160	**158**
P70	7	448	449	454	455	454	**448**	**448**	454	**448**
	10	-	272	**271**	286	282	**271**	**271**	272	273
	14	-	204	202	215	211	201	200	**198**	**198**

	19	-	154	154	166	155	149	**148**	151	151
P89	8	-	494	494	494	497	**492**	494	494	494
	12	-	370	365	379	377	365	365	363	**362**
	16	-	236	237	252	238	236	236	236	**235**
	21	-	205	205	221	205	202	**201**	199	**201**
P111	9	-	557	562	588	579	559	**556**	559	560
	13	-	319	329	355	331	321	321	320	**319**
	17	-	257	253	280	262	251	250	**246**	248
	22	-	192	197	222	196	195	192	191	**190**
P148	10	-	600	624	668	617	612	611	614	**606**
	14	-	427	431	476	445	425	420	417	**416**
	21	-	300	299	337	296	292	287	**283**	**283**
	29	-	202	202	229	198	196	192	**189**	**189**
P297	19	-	646	654	711	662	639	638	632	**629**
	29	-	430	435	503	448	435	427	424	**422**
	38	-	344	339	392	340	341	333	**324**	328
	50	-	256	263	299	256	256	249	**244**	245

*Updated cycle-times in bold.

References

AGHAJANI, M., GHODSI, R. & JAVADI, B. 2014. Balancing of robotic mixed-model two-sided assembly line with robot setup times. *The International Journal of Advanced Manufacturing Technology,* 74**,** 1005-1016.

BATTAÏA, O. & DOLGUI, A. 2013. A taxonomy of line balancing problems and their solutionapproaches. *International Journal of Production Economics,* 142**,** 259-277.

BAYKASOGLU, A. 2006. Multi-rule Multi-objective Simulated Annealing Algorithm for Straight and U Type Assembly Line Balancing Problems. *Journal of Intelligent Manufacturing,* 17**,** 217-232.

BORBA, L. & RITT, M. 2014. A heuristic and a branch-and-bound algorithm for the Assembly Line Worker Assignment and Balancing Problem. *Computers & Operations Research,* 45**,** 87-96.

ÇIL, Z. A., METE, S. & AĞPAK, K. 2016. Analysis of the type II robotic mixed-model assembly line balancing problem. *Engineering Optimization***,** 1-20.

DAOUD, S., CHEHADE, H., YALAOUI, F. & AMODEO, L. 2014. Solving a robotic assembly line balancing problem using efficient hybrid methods. *Journal of Heuristics,* 20**,** 235-259.

DO, N. A. D., NIELSEN, I. E., CHEN, G. & NIELSEN, P. 2016. A simulation-based genetic algorithm approach for reducing emissions from import container pick-up operation at container terminal. *Annals of Operations Research,* 242**,** 285-301.

EREL, E., SABUNCUOGLU, I. & AKSU, B. A. 2001. Balancing of U-type assembly systems using simulated annealing. *International Journal of Production Research,* 39**,** 3003-3015.

FATHI, M., Á LVAREZ, M. J. & RODRÍGUEZ, V. 2016. A new heuristic-based bi-objective simulated annealing method for U-shaped assembly line balancing. *European Journal of Industrial Engineering,* 10**,** 145-169.

GAO, J., SUN, L., WANG, L. & GEN, M. 2009. An efficient approach for type II robotic assembly line balancing problems. *Computers & Industrial Engineering,* 56**,** 1065-1080.

HAMZADAYI, A. & YILDIZ, G. 2012. A genetic algorithm based approach for simultaneously balancing and sequencing of mixed-model U-lines with parallel workstations and zoning constraints. *Computers & Industrial Engineering,* 62**,** 206-215.

HAMZADAYI, A. & YILDIZ, G. 2013. A simulated annealing algorithm based approach for balancing and sequencing of mixed-model U-lines. *Computers & Industrial Engineering,* 66**,** 1070-1084.

JAYASWAL, S. & AGARWAL, P. 2014. Balancing U-shaped assembly lines with resource dependent task times: A Simulated Annealing approach. *Journal of Manufacturing Systems,* 33**,** 522-534.

KHORASANIAN, D., HEJAZI, S. R. & MOSLEHI, G. 2013. Two-sided assembly line balancing considering the relationships between tasks. *Computers & Industrial Engineering,* 66**,** 1096-1105.

LEVITIN, G., RUBINOVITZ, J. & SHNITS, B. 2006. A genetic algorithm for robotic assembly line balancing. *European Journal of Operational Research,* 168**,** 811-825.

LI, Z., DEY, N., ASHOUR, A. S. & TANG, Q. 2017a. Discrete cuckoo search algorithms for two-sided robotic assembly line balancing problem. *Neural Computing and Applications***,** 1-12.

LI, Z., JANARDHANAN, M. N., TANG, Q. & NIELSEN, P. 2016a. Co-evolutionary particle swarm optimization algorithm for two-sided robotic assembly line balancing problem. *Advances in Mechanical Engineering,* 8**,** 1-14.

LI, Z., KUCUKKOC, I. & NILAKANTAN, J. M. 2017b. Comprehensive review and evaluation of heuristics and meta-heuristics for two-sided assembly line balancing problem. *Computers & Operations Research,* 84**,** 146-161.

LI, Z., TANG, Q. & ZHANG, L. 2016b. Minimizing energy consumption and cycle time in two-sided robotic assembly line systems using restarted simulated annealing algorithm. *Journal of Cleaner Production,* 135**,** 508-522.

MIRALLES, C., GARCÍA-SABATER, J. P., ANDRÉS, C. & CARDÓS, M. 2008. Branch and bound procedures for solving the Assembly Line Worker Assignment and Balancing Problem: Application to Sheltered Work centres for Disabled. *Discrete Applied Mathematics,* 156**,** 352-367.

MOSADEGH, H., ZANDIEH, M. & GHOMI, S. M. T. F. 2012. Simultaneous solving of balancing and sequencing problems with station-dependent assembly times for mixed-model assembly lines. *Applied Soft Computing,* 12**,** 1359-1370.

NILAKANTAN, J. M., HUANG, G. Q. & PONNAMBALAM, S. 2015a. An investigation on minimizing cycle time and total energy consumption in robotic assembly line systems. *Journal of Cleaner Production,* 90**,** 311-325.

NILAKANTAN, J. M., LI, Z., TANG, Q. & NIELSEN, P. 2017. Multi-objective co-operative co-evolutionary algorithm for minimizing carbon footprint and maximizing line efficiency in robotic assembly line systems. *Journal of Cleaner Production,* 156**,** 124-136.

NILAKANTAN, J. M. & PONNAMBALAM, S. 2016. Robotic U-shaped assembly line balancing using particle swarm optimization. *Engineering Optimization,* 48**,** 231-252.

NILAKANTAN, J. M., PONNAMBALAM, S. G., JAWAHAR, N. & KANAGARAJ, G. 2015b. Bio-inspired search algorithms to solve robotic assembly line balancing problems. *Neural Computing & Applications,* 26**,** 1379-1393.

NOURMOHAMMADI, A., NOURMOHAMMADI, A., ESKANDARI, H. & ESKANDARI, H. 2017. Assembly line design considering line balancing and part feeding. *Assembly Automation,* 37**,** 135-143.

ÖZCAN, U. 2010. Balancing stochastic two-sided assembly lines: A chance-constrained, piecewise-linear, mixed integer program and a simulated annealing algorithm. *European Journal of Operational Research,* 205**,** 81-97.

ÖZCAN, U. & TOKLU, B. 2009. Balancing of mixed-model two-sided assembly lines. *Computers & Industrial Engineering,* 57**,** 217-227.

RABBANI, M., MANAVIZADEH, N. & HOSSEINI AGHOZI, N. S. 2015. Robust optimization approach to production system with failure in rework and breakdown under uncertainty: evolutionary methods. *Assembly Automation,* 35**,** 81-93.

RABBANI, M., MOUSAVI, Z. & FARROKHI-ASL, H. 2016. Multi-objective metaheuristics for solving a type II robotic mixed-model assembly line balancing problem. *Journal of Industrial and Production Engineering,* 33**,** 472-484.

RASHID, M. F. F., HUTABARAT, W. & TIWARI, A. 2012. A review on assembly sequence planning and assembly line balancing optimisation using soft computing approaches. *The International Journal of Advanced Manufacturing Technology,* 59**,** 335-349.

RELICH, M. & PAWLEWSKI, P. 2016. A Multi-agent Framework for Cost Estimation of Product Design. *In:* BAJO, J., ESCALONA, M. J., GIROUX, S., HOFFA-DĄBROWSKA, P., JULIÁN, V., NOVAIS, P., SÁNCHEZ-PI, N., UNLAND, R. & AZAMBUJA-SILVEIRA, R. (eds.) *Highlights of Practical Applications of Scalable Multi-Agent Systems. The PAAMS Collection: International Workshops of PAAMS 2016, Sevilla, Spain, June 1-3, 2016. Proceedings.* Cham: Springer International Publishing.

ROSHANI, A., FATTAHI, P., ROSHANI, A., SALEHI, M. & ROSHANI, A. 2012. Cost-oriented two-sided assembly line balancing problem: A simulated annealing approach. *International Journal of Computer Integrated Manufacturing,* 25**,** 689-715.

ROSHANI, A. & NEZAMI, F. G. 2017. Mixed-model multi-manned assembly line balancing problem: a mathematical model and a simulated annealing approach. *Assembly Automation,* 37**,** 34-50.

RUBINOVITZ, J. & BUKCHIN, J. 1991. *Design and balancing of robotic assembly lines*, Society of Manufacturing Engineers.

RUBINOVITZ, J., BUKCHIN, J. & LENZ, E. 1993. RALB – A Heuristic Algorithm for Design and Balancing of Robotic Assembly Lines. *CIRP Annals - Manufacturing Technology,* 42**,** 497-500.

SCHOLL, A. & BECKER, C. 2006. State-of-the-art exact and heuristic solution procedures for simple assembly line balancing. *European Journal of Operational Research,* 168**,** 666-693.

SITEK, P., NIELSEN, I. E. & WIKAREK, J. 2014. A hybrid multi-agent approach to the solving supply chain problems. *Procedia Computer Science,* 35**,** 1557-1566.

SITEK, P. & WIKAREK, J. 2016. A Hybrid Programming Framework for Modeling and Solving Constraint Satisfaction and Optimization Problems. *Scientific Programming,* 2016**,** 1-13.

TANG, Q., LI, Z. & ZHANG, L. 2016. 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,* 97**,** 146-156.

YOOSEFELAHI, A., AMINNAYERI, M., MOSADEGH, H. & ARDAKANI, H. D. 2012. Type II robotic assembly line balancing problem: An evolution strategies algorithm for a multi-objective model. *Journal of Manufacturing Systems,* 31**,** 139-151.

Appendixes

Table A1 Main operators of the implemented methods

Algorithm	Abbrev	Applied operators
Genetic algorithm (Gao et al., 2009)	GA	Binary tournament selection is applied for individual selection, and elitism strategy is utilized to clone the best individual to the offspring.
Particle swarm optimization (Li et al., 2016a)	PSO	The crossover operator is applied to simulate the moving to global best individual and local best individual, and neighbor operator is utilized to simulate the initial velocity. No restart mechanism or local search on the best individual is applied.
Discrete cuckoo search (Li et al., 2017a)	DCS	The duplicated individuals and the worst individuals are abandoned, and they are replaced with the neighbor solutions of the remained individuals.
Artificial bee colony (Tang et al., 2016)	ABC1	The incumbent solution is updated when the new one achieves the better or the same fitness. The scout replaces the duplicated individual or the worst individual in the swarm with a randomly generated individual when no improvement on the best fitness is achieved.
Artificial bee colony (Tang et al., 2016)	ABC2	The incumbent solution is updated when the new one achieves the better or the same fitness. The scout replaces the duplicated individual or the worst individual in the swarm with a neighbor solution of a randomly selected individual from the current swarm when no improvement on the best fitness is achieved.
Simulated annealing	SA	The proposed methodology in Section 3.2.
Restarted simulated annealing	RSA	The proposed methodology in Section 3.2.

Table A2 Selected parameter values of the implemented methodologies

Algorithm	Parameters	Range	Selected value
GA	Population size	80, 120, 160, 200	120
	Crossover rate (Mutation rate=1-crossover rate)	0.5, 0.6, 0.7, 0.8	0.5
	Selection type	Binary tournament selection	
	Crossover operator	Two-point crossover operator	
PSO	Swarm number	4, 6, 8	4
	Number of particles in a swarm	20, 40	40
	Leaner rate	0.4, 0.5, 0.6, 0.7	0.7
	Moving to global best or local best	Two-point crossover operator	
DCS	Population size	20, 40, 60, 80	20
	Abandon rate	0.1, 0.2, 0.3, 0.4	0.1
ABC1	Population size	20, 40, 60, 80	20
ABC2	Population size	20, 40, 60, 80	20
SA	Initial temperature	0.5, 1	1.0
	Ratio of temperature decreasing	0.9, 0.95, 0.98	0.9
	Iteration rate	100, 500, 1000	500
RSA	Initial temperature	0.5, 1.0	0.5
	Cooling rate	0.9, 0.95, 0.98	0.9
	Number of iterations before a temperature change	100, 500, 1000	100
	Restart temperature	0.1, 0.01, 0.001, 0.0001, 0.0	0.01 for small-size problems 0.001 for large-size problems
	Restart time (RT) before executing restart mechanism	100, 200	200

Table A3-1 Detailed results for Model 1 and Model 2

Tested model	Task number	Ns	Single equations	Single variables	Non-zero elements	Iterations	Nodes	Results	Time (s)
Model 1	P25	3	75	235	1,281	4359	65	503	0.85
		4	85	417	2,276	40067	611	327	2.28
		6	111	937	5,118	1303828	15991	213	55.46
		9	165	2,107	11,511	57300147	537022	123	3600
	P35	4	108	577	3,172	139989	2244	449	6.51
		5	120	901	4,955	260772	1847	344	12.40
		7	150	1,765	9,709	3858563	36812	222	224.69
		12	260	5,185	28,524	26725190	54749	130	3600
	P53	5	175	1,351	8,155	927120	7216	554	55.27
		7	205	2,647	15,981	18055777	118910	320	1461.13
		10	265	5,401	32,610	27379891	70791	345	3600
		14	373	10,585	63,910	13952637	21798	288	3600
	P70	7	226	3,480	18,872	31893875	250196	448	3091.40
		10	286	7,101	38,510	21156916	43502	363	3600
		14	394	13,917	75,474	12349626	13091	577	3600
		19	574	25,632	139,004	6453810	7156	1023	3600
Model 2	P25	3	763	194	5,414	55949707	16477823	503	3600
		4	990	226	8,110	31000819	7079287	327	3600
		6	1,444	296	14,864	43303111	7485976	214	3600
		9	2,125	416	28,400	27488794	5025133	125	3600
	P35	4	2,371	433	20,234	33946347	5147710	456	3600
		5	2,935	477	28,095	32543872	4370414	348	3600
		7	4,063	571	47,201	20965116	799343	236	3600
		12	6,883	841	114,706	11079698	1310261	128	3600
	P53	5	3,103	567	29,086	30053350	3144291	560	3600
		7	4,269	697	48,840	24080990	554717	329	3600
		10	6,018	907	87,216	21623950	366851	274	3600
		14	8,350	1,215	154,708	16664144	137086	182	3600
	P70	7	15,052	1,590	177,604	7356866	147177	507	3600
		10	21,406	1,851	317,200	4549834	25672	352	3600
		14	29,878	2,227	562,632	3286099	7507	296	3600
		19	40,468	2,742	964,732	1689243	61061	185	3600

Table A3-2 Detailed results for Model 3 and Model 4

Tested model	Task number	Ns	Single equations	Single variables	Non-zero elements	Iterations	Nodes	Results	Time (s)
Model 3	P25	3	232	82	759	4043	451	503	0.44
		4	449	113	1,452	244240	8198	327	5.03
		6	1,141	181	3,606	5780437	367636	213	243.59
		9	2,914	298	9,027	31466587	459797	130	3600
	P35	4	615	153	2,000	143537	13552	449	5.04
		5	1,020	196	3,275	500289	27924	344	21.97
		7	2,184	288	6,881	6603289	263998	222	466.32
		12	7,439	553	22,896	13138218	73719	125	3600
	P53	5	1,778	286	5,675	27235094	1182648	554	1453.93
		7	3,756	414	11,795	35108103	581730	347	3600
		10	8,253	621	25,550	11004986	46460	347	3600
		14	17,357	925	53,186	4343624	6654	529	3600
	P70	7	3,941	533	12,537	32194702	549926	458	3600
		10	8,630	791	26,970	10308079	41297	361	3600
		14	18,102	1,163	55,846	4340084	7090	1383	3600
		19	35,657	1,673	109,041	1771845	912	1395	3600
Model 4	P25	3	3,193	82	9,642	2351	68	503	1.23
		4	4,397	113	13,296	39710	978	327	14.13
		6	7,063	181	21,372	273109	3622	213	83.16
		9	11,797	298	35,676	11183732	155859	123	3600
	P35	4	7,967	153	24,056	178230	3634	449	70.53
		5	10,210	196	30,845	126301	1327	344	61.25
		7	15,050	288	45,479	1131004	15559	222	632.80
		12	29,495	553	89,064	2830002	8022	135	3600
	P53	5	75,168	286	225,845	747735	2037	560	3600
		7	106,502	414	320,033	506316	600	377	3600
		10	155,033	621	465,890	318338	144	407	3600
		14	222,849	925	669,662	263176	45	311	3600
	P70	7	103,208	533	310,338	551804	496	467	3600
		10	150,440	791	452,400	319926	149	308	3600
		14	216,636	1,163	651,448	235759	46	888	3600
		19	305,096	1,673	917,358	317372	8	706	3600