Applied Mathematical Modelling 65 (2019) 256–270

Contents lists available at ScienceDirect

Applied Mathematical Modelling

journal homepage: www.elsevier.com/locate/apm

Metaheuristic algorithms for balancing robotic assembly lines with sequence-dependent robot setup times

Mukund Nilakantan Janardhanan[a,b,*], Zixiang Li[c,d], Grzegorz Bocewicz[e], Zbigniew Banaszak[e], Peter Nielsen[a]

[a] *Department of Materials and Production, Aalborg University, Aalborg, Denmark*
[b] *Department of Engineering, University of Leicester, Leicester, United Kingdom*
[c] *Engineering Research Center for Metallurgical Automation and Measurement Technology of Ministry of Education, Wuhan University of Science and Technology, Wuhan, Hubei, China*
[d] *Key Laboratory of Metallurgical Equipment and Control Technology, Wuhan University of Science and Technology, Wuhan, Hubei, China*
[e] *Department of Electronics and Computer Science, Koszalin University of Technology, Sniadeckich 2, 75-453 Koszalin, Poland*

ARTICLE INFO

Article history:
Received 18 December 2017
Revised 9 August 2018
Accepted 20 August 2018
Available online 29 August 2018

Keywords:
Assembly line balancing
Robotic assembly line
Sequence-dependent setup times
Metaheuristics

ABSTRACT

Industries are incorporating robots into assembly lines due to their greater flexibility and reduced costs. Most of the reported studies did not consider scheduling of tasks or the sequence-dependent setup times in an assembly line, which cannot be neglected in a real-world scenario. This paper presents a study on robotic assembly line balancing, with the aim of minimizing cycle time by considering sequence-dependent setup times. A mathematical model for the problem is formulated and CPLEX solver is utilized to solve small-sized problems. A recently developed metaheuristic Migrating Birds Optimization (MBO) algorithm and set of metaheuristics have been implemented to solve the problem. Three different scenarios are tested (with no setup time, and low and high setup times). The comparative experimental study demonstrates that the performance of the MBO algorithm is superior for the tested datasets. The outcomes of this study can help production managers improve their production system in order to perform the assembly tasks with high levels of efficiency and quality.

1. Introduction

Assembly lines have been extensively used in the consumer electronics and automobile industries for the assembly of different products [1,2]. Due to increasing human labor costs and customers' mounting demands for a variety of products, industries are utilizing robots in assembly lines in order to improve production flexibility and product quality [3]. Robotic assembly line balancing (RALB) problems have been receiving increased attention in the last few years. RALB is defined as assigning a set of tasks to workstations in balanced form by allocating the best robot to each task at each workstation. Some contributions on RALB focus on type I robotic assembly line balancing (RALB-I) problems to minimize the number of workstations, while type II robotic assembly line balancing (RALB-II) problems are concerned with the optimization of cycle

* Corresponding author at: Department of Engineering, University of Leicester, Leicester, United Kingdom
E-mail addresses: lizixiang@wust.edu.cn (M.N. Janardhanan), zixiangliwust@gmail.com (Z. Li), bocewicz@ie.tu.koszalin.pl (G. Bocewicz), zbigniew.banaszak@tu.koszalin.pl (Z. Banaszak), peter@mp.aau.dk (P. Nielsen).

time [4]. Simple assembly line balancing problems are classified as NP-hard; the RALB problem considered in this research is also categorized under NP-hard due to the added complexities [5,6].

Rubinovitz and Bukchin [7] introduce the RALB problem with the aim of minimizing the number of workstations (RALB-I); later Rubinovitz et al. [8] apply the branch and bound method to solve the same problem. Due to computational complexity, researchers have started to use heuristics [9] and metaheuristic algorithms to solve problems of this nature; e.g. Levitin et al. [3] utilize a genetic algorithm to solve the RALB-II problem, where it is assumed that all robot types are available without any limitations. Gao et al. [5] were the first to present the mathematical model for RALB-II with slight differences from the assumption in Levitin et al. [3]. It is assumed that the available robot types are predetermined and only one robot is available for each type. They incorporate an improved genetic algorithm with local search to solve the identified problem. Yoosefelahi et al. [10] present a RALB problem with multiple objectives, using three versions of multi-objective evolution strategies. Nilakantan et al. [4] utilize a set of bio-inspired algorithms (particle swarm optimization and cuckoo search algorithms) to solve the same RALB-II problem reported in Levitin et al. [3] and provide better solutions for the benchmark problems. Most of the abovementioned literature focuses on type I and type II RALB problems. Due to an increased awareness of energy conservation, Mukund Nilakantan et al. [11] were the first to deliver a contribution focusing on minimizing energy consumption in straight robotic assembly lines by using particle swarm optimization based on the assumptions in Levitin et al. [3]. Most of the literature focuses on straight robotic assembly lines and considers only single product assembly; however, recently, many contributions are being reported for different assembly line configurations and different product assemblies (mixed model). Çil et al. [12] solve a mixed-model RALB-II problem and utilize beam search to minimize cycle time. Rabbani et al. [13] solve a multi-objective mixed-model RALB-II problem by using a multi-objective genetic algorithm and particle swarm optimization. Çil et al. [14] report multi-objective optimization using a goal programming technique. Several objectives such as minimizing cycle time, number of workstations and robot cost are considered and tested in a case study. Çil et al. [15] were the first to propose a study on parallel robotic assembly line balancing problems, with the aim of minimizing cycle time. To solve the proposed problem, they utilize beam search approaches and compare them to other types of robotic assembly lines. Li et al. [16] were the first to propose simultaneously balancing and sequencing robotic mixed-model assembly lines. They propose a mixed-integer programming model to minimize makespan and utilize a CPLEX solver for solving small-sized problems, making use of two metaheuristics: a restarted simulated annealing algorithm and a co-evolutionary algorithm, to address this NP-hard problem. Nilakantan et al. [17] optimize carbon footprint and line efficiency simultaneously and utilize a multi-objective co-operative co-evolutionary algorithm. These types of problems are naturally suited to being solved by implementing metaheuristics, as can be seen from the solution strategies utilized in the literature listed above.

In most of the contributions, setup time is not given much attention and it is assumed that the setup times are either negligible or can be included in the task processing time. However, given today's advanced and flexible manufacturing systems, it is important that the various resources are utilized efficiently and that setup times are treated separately from processing times, allowing the assembly operations to be performed simultaneously, thus improving resource utilization. There are two types of setup time: sequence-independent and sequence-dependent [18]. Sequence-independent setup time considers the setup time for the current task, regardless of the preceding task. In the case of sequence-dependent setup time, setup time depends on both the current task and the preceding task. Özcan and Toklu [19] report that some consideration of sequence-dependent setup times is necessary for assembly lines. The first study which explores sequence-dependent setup times was performed by Andres et al. [20], who term the problem: General assembly line balancing problem with setups. They utilize eight different heuristic rules and a GRASP algorithm to solve this problem. Scholl et al. [21] propose a problem similar to that of Andres et al. [20], where sequence-dependent task time increments are introduced, along with several versions of a mixed-integer program, and various solutions to the problem are proposed. Yolmeh and Kianfar [22] propose solving assembly line balancing and scheduling problems with setup times (SUALBP) and utilize a hybrid genetic algorithm to solve them. The problem aims to assign and schedule tasks for each station. The operators and parameters involved are selected using designs of experiments. Yolmeh and Kianfar compared the solution obtained with the published solutions, and found that the proposed algorithm outperformed the reported solutions. Scholl et al. [23] propose a mathematical model for the assembly line balancing problem and scheduling problem which considers sequence-dependent setup times and propose a toolbox comprising a set of heuristic methods to test the proposed problem. Extensive computational studies demonstrate that proposed heuristics dominate former approaches in terms of solution quality and computation times.

Akpinar and Baykasoğlu [24] propose a model for solving mixed model assembly line balancing problems which utilizes setup times and applies multiple colony hybrid bee algorithms. They tested this model on a set of datasets and reported the superiority of their method compared to others in the literature. Seyed-Alagheband et al. [25] investigate the problem of balancing and sequencing tasks in an assembly line using sequence-dependent setup times with the aim of minimizing cycle time. They develop a mathematical model and propose a simulated annealing algorithm to solve the problem. Hamta et al. [26] propose a multi-objective assembly line balancing problem with flexible operation times, sequence-dependent setup times and learning effect. Three objectives are simultaneously optimized and they utilize particle swarm optimization algorithms with a variable neighborhood search to solve the problem.

To the best of the authors' knowledge, no contributions have been published which deal with straight robotic assembly lines with sequence-dependent setup times, aside from one recent paper by Aghajani et al. [27] on two-sided assembly lines. In this study, a new mixed-integer programming model is proposed for type II robotic mixed-model two-sided assembly line balancing. In the case of robotic assembly lines, it is essential to consider sequence-dependent setup times, since tasks might require some setup changes to the tools/robotic arms in order for the allocated tasks to be properly executed in the workstations. If a task is to be executed in the same workstation directly before another task, the preceding task may influence the following task because a setup/tool change might be necessary, and this setup time must be considered when calculating the end time of the task. Moreover, if a task is assigned to the workstation as the final task, a setup/tool change may be required before the first task assigned to that workstation can be carried out, since the tasks are performed cyclically [19].

Building on the significance and relevance of the above study to sequence-dependent setup times in robotic assembly lines, this study presents several contributions, as follows:

(1) A mixed-integer linear programming model is developed to solve robotic assembly line balancing problems using sequence-dependent setup times and with the aim of minimizing cycle times (type II RALB-S). Small-sized problems are solved using the CPLEX optimization package. Datasets are generated for creating problem instances in robotic assembly lines with setup times.
(2) A newly developed Migrating Birds Optimization (MBO) algorithm is employed to tackle large-sized problems within an acceptable CPU time. MBO was selected due to its superior performance in solving problems of a similar nature [28,29] and this is the first attempt to apply MBO to solve type II RALB with sequence-dependent setup times.
(3) A comprehensive comparative study is conducted to test the performance of the proposed algorithm. A set of well-known metaheuristic algorithms are selected and re-implemented to solve the proposed problem; from the computational study, it can be seen that the proposed MBO performs better for the majority of problem instances. Since it is an NP-hard problem, it is necessary to test it with different metaheuristics, perform a comparative study, and find the best performing metaheuristic.

The remainder of the paper is organized as follows: Section 2 presents the mathematical model and the problem assumptions. Section 3 presents a detailed methodology along with an illustrated example. Section 4 presents detailed comparative computational study analysis. Finally, the conclusion and future research directions are provided in Section 5.

2. Mathematical model formulation

This section first describes the problem and the problem assumptions, before presenting a mathematical model of the problem.

2.1. Problem description

Based on conclusions from the literature review, this paper will focus on proposing algorithms and approaches for solving type II robotic assembly line balancing problems with setup times. This paper tackles the problem by aiming to minimize the cycle time and integrate setup times into the cycle time. The assumptions listed here are based on the studies of Gao et al. [5] and Andres et al. [20]. It should be noted that Scholl et al. [23] categorize the setup times into two types: forward setup and backward setup. Forward setup refers to a situation where task j is operated directly after task i is completed in the same cycle. Backward setup occurs when task i is the last one operated at the workpiece in cycle p and the worker has to move on to the next workpiece to complete a task which is to be assembled in cycle p+1. In this study, only forward setup times are considered based on the work reported in Aghajani et al. [27], which is the only paper published to date which considers setup times in robotic assembly lines. Based on the authors' knowledge, the positions of robot and products are always fixed and hence no backward setup time is observed in automobile factories. Therefore, in this study, of robotic assembly lines with setup times, only forward setup times are considered. The following assumptions are considered in this study.

(1) This study considers a robotic assembly line where a single product is assembled.
(2) The task processing times of the robots, the sequence-dependent setup times matrix and the precedence relationships are known deterministically.
(3) The processing times and setup times of robots are independent of the workstation at which tasks are processed.
(4) A straight robotic assembly line layout is considered. In this layout, workstations are arranged in a straight line.
(5) A maximum of one robot can be allocated to each workstation.
(6) The number of available robots is greater than or equal to the number of workstations.
(7) Material handling, loading and unloading times are considered negligible or can be considered to be included in the processing time of the tasks.

In the considered robotic assembly line there is a set of workstations, and each of these workstations has a robot allocated to it. If there are *Nt* tasks and *Ns* workstations, the available *Nr* robots will be allocated to these *Ns* workstations to perform the allocated *Nt* tasks. The considered RALB problem aims to assign *Nt* tasks to *Ns* workstations and allocate the *Nr* robots with the objective of minimizing cycle time, where the cycle time includes the sequence-dependent setup times. There are two main sub-problems that are optimized simultaneously: assignment of tasks and allocation of robots. A task will be executed only if all preceding tasks have been completed and a robot has been allocated to the workstation to perform the allocated tasks. Fig. 1 shows a sample layout of a robotic assembly line; the assembly line consists of 4

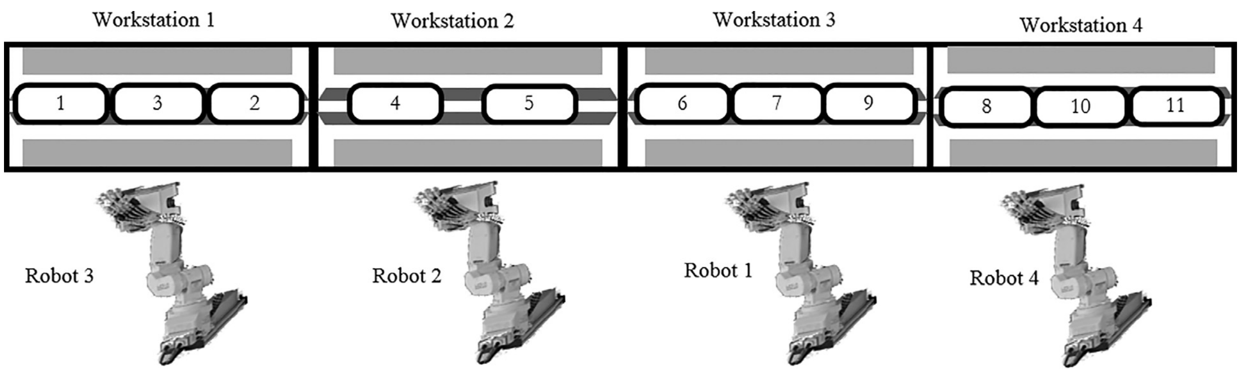

Fig. 1. Balanced robotic assembly line.

workstations and 11 tasks. Tasks are allocated to the workstations in a balanced manner and a best-fit robot is assigned to perform the allocated tasks.

2.2. Mathematical model – mixed integer programming model

The notations used in the mathematical formulations are given as follows:

- *Indices*

i, j: Index of tasks.
k: Index of workstations.
p: Index of position inside the schedule of a workstation.
r: Index of robots.
I: Set of tasks and $I = \{1, 2, \cdots, i, \cdots, Nt\}$, where Nt is the number of tasks.
K: Set of workstations and $K = \{1, 2, \cdots, k, \cdots, Nk\}$ where Nk is the number of workstations.
R: Set of robots and $R = \{1, 2, \cdots, r, \cdots, Nr\}$ where Nr is the number of available robots.

- *Parameters*

t_{ir}: Operation time of task i by robot r.
PT_i: Set of tasks, all of which precede task i.
Nm_k: Maximum number of tasks that can be assigned to workstation k.
N^{max}: Maximum number of tasks that can be assigned to any workstation. $N_{max} = \max_k\{Nm_k\}$.
s_{ijr}: Setup time when task j is performed just after task i at the same workstation operated by robot r.
ψ : A very large positive number.

- *Decision variables*

CT: Cycle time.
x_{irkp}: Binary variable. x_{irj} is equal to 1 when task i is operated by robot r at station k in position p of its schedule.
y_{irk}: Binary variable. y_{irk} is equal to 1 when task i is the last one operated by robot r in the sequence of tasks assigned to workstation k.
z_{ijrk}: Binary variable. z_{ijrk} is equal to 1 when task i is performed immediately before task j is operated by robot r at the workstation k in the same or in the next cycle.
w_{rk}: Binary variable. w_{rk} is equal to 1 when robot r is allocated to station k.
The model presented below is developed based on the mathematical model presented in Andres et al. [20].

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

Subject to:

$$\sum_{r\in R}\sum_{k\in K}\sum_{p=1}^{Nm_k} x_{irkp} = 1 \quad \forall i \in I \tag{2}$$

$$\sum_{k\in K} w_{rk} \le 1 \quad \forall r \in R \tag{3}$$

$$\sum_{r\in R}\sum_{k\in K} w_{rk} = Nk \tag{4}$$

$$\sum_{i\in I} \mathrm{x}_{irkp} \le 1 \;\; \forall \mathrm{r} \in \mathrm{R},\; \mathrm{k} \in \mathrm{K},\; \mathrm{p} = 1, \cdots, Nm_k \tag{5}$$

$$\sum_{i\in I} \mathrm{x}_{\mathrm{irk,p+1}} \le \sum_{i\in I} \mathrm{x}_{\mathrm{irkp}} \;\; \forall \mathrm{r} \in \mathrm{R},\; \mathrm{k} \in \mathrm{K},\; \mathrm{p} = 1, \cdots, Nm_k - 1 \tag{6}$$

$$\sum_{r\in R}\sum_{k\in K}\sum_{p=1}^{Nm_k} (N^{max} \cdot (k-1) + p) \cdot \mathrm{x}_{\mathrm{irkp}} \le \sum_{r\in R}\sum_{k\in K}\sum_{p=1}^{Nm_k} (N^{max} \cdot (k-1) + p) \cdot \mathrm{x}_{\mathrm{jrkp}} \;\forall (i, \mathrm{j}) | i \in PT_j \tag{7}$$

$$\sum_{i\in I}\sum_{r\in R}\sum_{p=1}^{Nm_k} t_{ir} \cdot \mathrm{x}_{\mathrm{irkp}} + \sum_{i\in I}\;\sum_{\mathrm{j}\in \mathrm{I}|(i\ne j)}\;\sum_{r\in R} s_{ijr} \cdot \mathrm{z}_{\mathrm{ijrk}} \;\le CT \;\; \forall k \in K \tag{8}$$

$$\mathrm{x}_{\mathrm{irkp}} + \mathrm{x}_{\mathrm{jrk,p+1}} \le 1 + z_{ijrk} \;\; \forall \mathrm{r} \in \mathrm{R},\; \mathrm{k} \in \mathrm{K},\; \mathrm{p} = 1, \cdots, Nm_k - 1,\; \forall (i, \mathrm{j}) | (\mathrm{i} \ne \mathrm{j}) \wedge (\mathrm{j} \notin PT_i) \tag{9}$$

$$\mathrm{x}_{\mathrm{irkp}} - \sum_{\forall \mathrm{j}\in \mathrm{I}|(i\ne j)\wedge(\mathrm{j} \notin PT_i)} \mathrm{x}_{\mathrm{jrk,p+1}} \le y_{irk} \;\; \forall \mathrm{i} \in \mathrm{I},\; \mathrm{r} \in \mathrm{R},\; \mathrm{k} \in \mathrm{K},\; \mathrm{i} \ne \mathrm{j},\; \mathrm{p} = 1, \cdots, Nm_k - 1 \tag{10}$$

$$y_{irk} + \mathrm{x}_{\mathrm{jrk1}} \le 1 + z_{ijrk} \;\; \forall \mathrm{r} \in \mathrm{R},\; \mathrm{k} \in \mathrm{K},\; \mathrm{i} \ne \mathrm{j},\; \mathrm{i} \notin PT_j \tag{11}$$

$$\sum_{i\in I} \mathrm{x}_{\mathrm{irkp}} \le \psi \cdot w_{rk} \;\; \forall r \in R, k \in K,\; \mathrm{p} = 1, \cdots, Nm_k \tag{12}$$

The objective function (1) aims to minimize the cycle time. Constraint (2) implies that each task must be operated by only one robot and must be assigned to only one position at only one workstation. Constraint (3) and constraint (4) ensures that each workstation is allocated a robot and the number of utilized robots is equal to the number of workstations. Constraint (5) implies that, in each position at each workstation, there should be no more than one task allocated to robot *r*. Constraint (6) ensures that the tasks are assigned by increasing positions in the schedule of every workstation by robot *r*. Constraint (7) addresses precedence constraints, which constrain the task assignment on both the positions inside the same workstation and between different workstations. Constraint (8) ensures that the global time in each workstation, including task durations and setup times, is less than or equal to cycle time. Constraints (9)–(11) restrict the value of z_{ijrk}. Specifically, Constraint (9) denotes that the variable z_{ijrk} is 1 when task *i* is allocated to position *p* and task *k* is allocated to position *p+1* in the schedule of workstation *k*. Constraints (10)–(11) denote that the variable y_{irk} is 1 when task *i* is the last one operated by robot *r* in the sequence of tasks assigned to workstation *k*, and the variable z_{ijrk} is 1 when task *j* is assigned to the first position and task *i* to the last position in the schedule of workstation *k*. Constraint (12) ensures that tasks found at the same workstation are operated by the same robot.

3. Proposed migrating birds optimization algorithm

The migrating birds optimization algorithm is a recently developed metaheuristic approach, based on the V-shaped flight formation of migrating birds, a strategy that has been proven to be effective in conserving energy while flying long distances during migration [30]. It is believed that each bird flies at a specific angle and distance relative to the lead bird. The lead bird in the flock expends the most energy while the other birds follow the flying pattern of the lead bird. MBO is an algorithm that includes a neighboring search technique. The algorithm starts with a set of initial solutions, which are improved at each step. There are four main steps and four parameters in MBO. After the initial solutions are generated, the subsequent steps are performed iteratively until the termination criteria are met: leader improvement, block improvement and leader replacement. In the loop, improvement of the leader is attempted by generating and evaluating *k* neighbor solutions. If the improvement is obtained, the best neighbor solution after evaluation of all neighbor solutions replaces the current leader. In addition to the abovementioned step, other potential individuals are also improved using the *x* unused best neighbor solutions that are in the front and also its neighbor (*k-x*) solutions, and this procedure is the block improvement stage of the algorithm. The sharing of neighbor solutions of other individuals helps promote communication between the individuals and also facilitates the evolution of the whole population. This is referred to as the benefit mechanism [31]. Leader improvement and block improvement are executed consecutively *m* times and then the leader replacement procedure is carried out where the leading individual is moved to the end and the best of the next individuals will be made the leader.

Recently, MBO has shown its superior performance in solving combinatorial problems such as quadratic assignment problems [30] and flow shop scheduling problems [32]. Despite this, attempts to use the algorithm to solve assembly line balancing problems have been minimal. This study aims to show why MBO should be used to improve robotic assembly line balancing. The following sections describe how the metaheuristic algorithm is implemented to solve the considered problem.

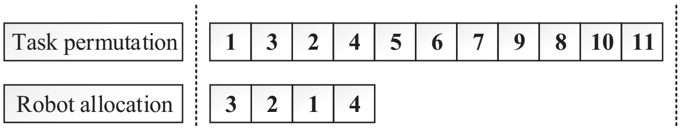

Fig. 2. Encoding scheme.

3.1. Encoding and decoding

This study uses the encoding procedure reported in Gao et al. [5], where both task permutation and robot allocation are utilized. Based on the example presented in Fig. 1, one possible encoding is illustrated in Fig. 2. In the task permutation, the tasks in the first positions have higher priority and are allocated first. For instance, tasks 1, 2 and 3 are in the first positions and hence they are allocated to the first workstation, as can be seen in Fig. 1. In the robot assignment procedure, the robots are allocated to workstations in a sequence. For instance, robot 3, robot 2, robot 1 and robot 4 are allocated to workstation 1, workstation 2, workstation 3 and workstation 4.

As two vectors are utilized in the encoding scheme, this study develops its decoding procedure based on Gao et al. [5], and utilizes the iterative mechanism for two-sided assembly lines used in Li et al. [33]. The decoding procedure is presented in detail below. In this procedure, robot allocation is firstly determined by the robot allocation vector, following which the workstation is assigned with as many as possible tasks in the former positions of the task permutation.

Algorithm Decoding procedure.

```
Input: Task permutation, robot allocation and instance
Start
1.  Evaluate the initial cycle time
2.  Open a new workstation
3.  Plan the allocation of tasks
4.  Assign the allocated robot to the current workstation.
5.  Select the tasks satisfying precedence constraint and cycle time constraint.
6.  If a situation with no assignable tasks exists
        a. Open a new workstation
        b. Or else
        c. Execute the previous step (delete the task)
7.  End if
8.  Allot the task in the former position of the task permutation to the current workstation
9.  Update the remaining capacity of the current workstation
10.  If all tasks have been allotted
11.  Terminate the decoding procedure
12.  Otherwise
13.  Open a new workstation
14.  End if
15.  Stop
Output: The corresponding cycle time achieved and the detailed task and robot assignment
```

In the proposed approach, the initial cycle time is set to a large value,$CT = 2 \cdot \sum_{i\in I} \sum_{r\in R} t_{ir} / (Nr \cdot Ns)$, where Nr is the number of available robots and Ns is the number of workstations. Assigning a large value ensures that all the problems will be able to obtain a good feasible solution. The procedure updates the initial cycle time when a new best cycle time is obtained using $CT = CT_{Best} - 1$. Once the new best cycle time is achieved, all the individuals in the solution are re-decoded using this new CT and their fitness values are updated. The initial cycle time in the algorithm evolution is updated using the following procedure.

Setting the initial cycle time to a large value ensures the discovery of a feasible solution in a faster computational time by gradually decreasing achieved cycle time. This method is referred to as an iterative mechanism in Li et al. [33], who used this procedure for type II two-sided assembly line balancing problems. Using the same initial cycle time for all individuals in the solution, it preserves minor improvements made to individuals. However, in the procedure reported in Mukund, Nilakantan and Ponnambalam [34], the initial cycle time is incremented gradually from a small value until a feasible solution is obtained. This naturally increases the computational time; however, the proposed method is computationally faster because it executes the decoding scheme only once.

3.2. MBO methodology

This study presents a modified MBO algorithm with specifically chosen improvements. The algorithm of the proposed MBO is illustrated in this section. In the standard MBO algorithm, there are four steps, starting with block initialization (population initialization) and followed by three steps (leader improvement, block improvement and leader replacement)

Algorithm Cycle time update.

```
Step 1. Set the initial cycle time as $CT = 2 \cdot \sum_{i\in I} \sum_{r\in R} t_{ir}/(Nr \cdot Ns)$
Step 2. Obtain the solutions using CT as cycle time
Step 3. If a new best cycle time is achieved, CT is updated to $CT = CT_{Best} - 1$ and all the
        individuals are re-decoded using this new CT as cycle time
Step 4. Obtain the solutions in each iteration using CT as the initial cycle time
Step 5. If the algorithm is still running, go to the 3rd step
```

which constitute a loop, and this loop is terminated when one predetermined termination criterion is satisfied. The procedure followed in this study is similar; however, several improvements have been made to adapt the procedure to the problem.

The improvements made are as follows: if the leader improvement or block improvement procedure shows an improvement, the incumbent individual is directly replaced with a new neighbor solution. Even when the same fitness value is achieved, the incumbent individual is replaced with the new neighbor solution. The fitness of the neighbor solution is set to a very large positive value if it shares the same fitness as the incumbent individual. The ideas behind these improvements are as follows: replacing the incumbent individual immediately after the improvement achieved helps to search for more areas that are promising and helps avoid unnecessary searching around a poor individual. Replacing the incumbent individual with a new neighbor with an equivalent fitness value means there could be many solutions, as many individuals could share the same fitness value. By implementing this modification, it is possible to explore more solutions, enhancing the exploration to some extent. The fitness of the neighbor solution that shares the same fitness value as the incumbent one is set to a very large positive value. This value assignment is done to avoid premature convergence of the proposed algorithm. The value set in this paper is 10,000. If this is not incorporated after a few iterations, all the individuals will have the same fitness value.

Algorithm Procedure of modified MBO.

```
Input: Algorithm parameters and one instance (task operation times and precedence relations)
1.  Generate initial individuals randomly;                    % Initialization
2.  While (Termination criterion is not met) do
3.    For $i=1$ to $m$ do
4.        For $j=1$ to $k$ do                                  % Leader improvement
5.            Generate a neighbor solution for the leader solution
6.            Replace the current leader solution with the neighbor solution when the same
              or better fitness is achieved.
7.        End for
8.        Replace the current leader solution when same or better fitness value is achieved
          with best individual from the neighbor solutions.
9.        Set a large value for the fitness of the individual whose fitness is the same as the
          incumbent one.
10.      For each individual on both (left and right) sides % Block improvement
11.        For j=1 to (k-x) do
12.          Generate a neighbor solution for this solution;
13.          Replace the current solution with the neighbor solution when the same or
          better fitness is achieved.
14.      End for
15.      Replace this individual with the best individual from its (k-x) neighbor solutions and the x unused best neighbor
         solutions of the solution in the front when the same or better fitness is achieved.
16.      Set a large value to the fitness of the individual whose fitness is the same as the
         incumbent one.
17.      End for
18.   End for
19.  Move the leading individual to the end. % Leader replacement
20.  Forward the immediately following individual to the leader position.
21.  Endwhile
Output: Best cycle time achieved so far and the detailed task and robot assignment
```

The proposed MBO shows fast convergence, as observed during preliminary experiments. This study also employs new acceptance criterion to enhance exploitation capacity and avoid getting trapped into local optima. Specifically, once the current best cycle time has remained unchanged for many iterations (set to 500), the greedy acceptance criterion is replaced with the acceptance criterion in a simulated annealing algorithm [27]. Namely, the new neighbor solution replaces the incumbent one when it achieves a better fitness or with a probability of $exp^{-(\mathrm{Fit}(S')-\mathrm{Fit}(S))/(T\times Fit(S))}$, where S and S' refer to the incumbent solution and the new neighbor solution, T is the temperature, and $Fit(S)$ is the cycle time by a solution. In this paper, T is initialized with an initial temperature (set to 0.2), and is updated with $T = T\times\alpha$ (α is the cooling rate, set to 0.95) in each iteration. If a new best cycle time is achieved, the original greedy acceptance criterion is again applied.

Table 1
Precedence relationships and operation times for the illustrative example.

Tasks	Successors	Operation times			
		Robot 1	Robot 2	Robot 3	Robot 4
1	2, 3, 4, 5	81	37	51	49
2	6	109	101	90	42
3	7	65	80	38	52
4	7	51	41	91	40
5	7	92	36	33	25
6	8	77	65	83	71
7	9	51	51	40	49
8	10	50	42	34	44
9	11	43	76	41	33
10	11	45	46	41	77
11	-	76	38	83	87

Table 2
Sequence-dependent setup times between tasks for the illustrative example.

Robot 1											**Robot 2**										
0	2	9	10	10	0	4	7	0	5	7	0	2	1	2	4	7	8	6	8	3	3
8	0	7	1	10	3	1	5	2	2	3	5	0	1	2	1	2	5	8	5	3	0
6	6	0	3	9	1	4	1	0	0	10	4	5	0	3	6	1	4	2	1	1	1
10	6	3	0	0	3	2	0	8	2	9	1	5	1	0	7	1	8	1	2	5	7
6	6	6	3	0	4	4	8	2	0	7	5	2	3	4	0	4	7	5	6	3	4
8	9	5	1	10	0	7	0	0	5	3	8	8	3	1	4	0	3	4	4	4	3
6	7	4	6	5	7	0	7	0	9	7	1	7	7	1	4	6	0	0	2	6	1
8	2	6	5	1	1	6	0	7	5	7	5	6	5	6	2	3	2	0	1	4	3
7	5	2	6	6	4	8	1	0	10	6	0	1	6	3	8	0	6	4	0	6	0
8	8	8	8	0	3	6	4	4	0	5	6	7	0	1	5	6	2	0	7	0	2
8	5	10	4	1	2	10	8	1	4	0	0	8	3	3	2	8	6	5	3	2	0
Robot 3											**Robot 4**										
0	0	5	1	3	7	5	5	3	7	5	0	4	0	4	4	4	0	0	0	4	0
3	0	6	1	7	0	2	4	1	3	0	5	0	0	1	4	5	4	2	3	5	0
3	7	0	4	7	2	3	0	8	4	1	1	3	0	0	2	0	0	2	5	3	1
1	0	2	0	5	5	7	2	2	4	6	4	3	2	0	0	0	1	1	2	5	3
0	6	8	2	0	5	0	4	2	3	5	1	0	3	3	0	4	5	5	3	0	2
1	3	4	1	0	0	0	5	2	6	3	4	3	2	0	4	0	5	5	6	1	4
0	7	7	6	4	1	0	4	3	5	5	2	5	4	5	5	2	0	3	0	5	4
3	1	5	1	7	7	7	0	5	4	1	1	4	2	5	1	6	0	0	0	6	5
8	1	5	1	4	6	1	1	0	2	0	0	3	2	5	2	4	0	3	0	3	5
6	4	3	2	6	6	5	4	4	0	2	4	3	1	3	0	4	1	4	3	0	2
1	0	7	7	6	2	0	5	1	1	0	4	5	0	0	2	4	3	0	4	2	0

Table 3
Detailed task assignment and robot selection for the illustrative example.

	Workstation 1	Workstation 2	Workstation 3	Workstation 4
Robot allocation	4	1	3	2
Task assignment	1, 2, 5	6, 4	3, 7, 9	8, 10, 11
Operation time	49, 42, 25	77, 51	38, 40, 41	42, 46, 38
Setup times	4, 4, 1	1, 3	3, 3, 5	4, 2, 5
Total time	125	132	130	**137**

3.3. Illustrative example

This section presents an example that illustrates how the encoding, decoding and cycle time calculation process works. The example considered is a problem made up of 11 tasks with 4 work stations and 4 robots. The precedence relationships and operation times of each robot's task are presented in Table 1. The first column presents the task numbers while the second column presents the precedence relationship. The remaining columns present the robots' operation times. The sequence-dependent setup times between tasks are shown in Table 2.

Based on the proposed procedure, the cycle time of the robotic assembly line is calculated by considering the sequence-dependent setup times between tasks; the allocation of robots and tasks are presented in detail in Table 3. In the table, the operation times of the allocated tasks, the sequence-dependent setup times and the total time are presented for each work-

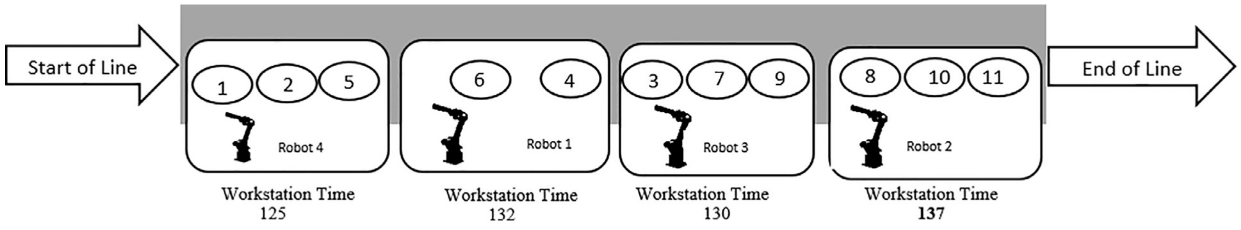

Fig. 3. Final tasks and robot allocation for the illustrative example.

Table 4
Tested algorithms for Type II RALB-S.

Algorithms	Description	Reference
GA	Genetic Algorithm - Elite strategy is applied by cloning the best individual to replace one of the offspring.	Kim et al. [36], Taha et al. [37]
PSO	Particle Swarm Optimization - Crossover, insert and swap operators are applied for population evolution.	Li et al. [35], Petropoulos and Nearchou [38]
ABC	Artificial Bee Colony - When no improvement on the best solution is achieved, a scout is applied to replace the worst individual or a duplicate individual with a neighbor solution.	Tang et al. [39], Saif et al. [40]
DCS	Discrete Cuckoo Search - Individuals with duplicate and worst solutions are discarded and replaced with the neighbor solutions of remaining individuals.	Li et al. [41]
SA	Simulated annealing algorithm - Proposes the task sequence vector, breakpoint vector and robot allocation vector for encoding. The other operators are set similarly to their settings in Aghajani et al. [27].	Aghajani et al. [27]
MBO	Migrating Bird Optimization	-

station. Fig. 3 also shows the final allocation of tasks to the workstations and the detailed robot allocation. The workstation completion times are also presented, and the cycle time is found to be 137.

4. Computational study

This section first presents the details of the experimental design, followed by the findings of the evaluation of the proposed models, and, finally, reports a comparison of the implemented algorithms. Since no research has been published concerning this problem, there are no benchmark problems available. This research utilizes and expands on nine sets of benchmark problems based on the ones reported in Gao et al. [5]. From each problem, the original precedence network and task performance robots are preserved. For every problem, two levels of setup time variability are set:

- For low *Variability,* the matrix of setup times is randomly generated based on uniform discrete distribution $U[0, 0.25^{*}min_{\forall i \in N}t_i]$.
- For high *Variability*, the matrix of setup times is randomly generated based on uniform discrete distribution $U[0, 0.75^{*}min_{\forall i \in N}t_i]$.

In total, the benchmark is composed of 33 problems with different combinations of task sizes and robots available. All 33 problems are tested with two levels of variability in setup times. Problems with task sizes ranging from 11 to 70 are classified as small-sized datasets and problems with task sizes ranging from 89 to 297 are classified as large-sized datasets. This paper re-implements some recent and high-performing metaheuristic methods by adapting them to the considered problem. The algorithms tested in this paper are summarized in Table 4. These algorithms are taken from the literature and have been reported to solve problems of a similar nature (references to relevant studies are summarized in Table 4). This paper adopts the same procedure followed in Li et al. [35] with regards to the termination criteria. Maximum elapsed CPU time is considered as the termination criterion in this paper and is equal to $Nt \times Nt \times \tau$ milliseconds, where it is tested at six levels (τ=10, 20, 30, 40, 50 and 60). By following this procedure, more computational time is allocated to large-sized problems and six termination criteria are used to analyze the performance of the algorithms for short CPU time and large CPU time. The algorithms were executed on a cluster of personal computers and were coded in C++. All the experiments were carried out on a tower type of server. This server had two Intel Xeon E5-2680 v2 processors (40 processor cores in total) running at 2.8 GHz and 64 GB of RAM memory.

4.1. Selected parameters for the metaheuristic algorithms

This research utilizes the full factorial design and Analysis of Variance (ANOVA) technique to determine the parameters following Li et al. [33,35] and many others. Specifically, all the combinations of the parameters are tested on one test problem with 111 tasks and 13 workstations, and this test problem is solved 10 times by each combination of the parameters,

Table 5
Parameters selected for the tested algorithms.

Algorithm	Parameters	Range	Selected value/procedure
GA	Population size	40, 80, 120, 160, 200	120
	Selection type	–	Binary tournament selection (the better of two randomly selected individuals is selected)
	Crossover type	–	Two-point crossover
	Crossover probability	0.4, 0.5, 0.6, 0.7, 0.8	0.5
	Mutation type	–	Swap operation and insert operation
	Mutation probability	-	1-crossover probability
ABC	Population size	40, 80, 120, 160, 200	40
	Neighborhood operator	–	Swap operation and insert operation
	Scout phase	–	Sending a scout when there is no improvement within an iteration to replace the worst individual in the population
PSO	Number of swarms	4, 8, 12	6
	Number of particles in a swarm	20, 40, 60	20
	c	0.5, 0.6	0.7
DCS	Population size	40, 80, 120	40
	Rate of abandoned individuals	0.1, 0.2, 0.3, 0.4	0.1
SA	Initial temperature	0.5, 1	0.5
	Cooling rate	0.9, 0.95, 0.98	0.9
	Number of iterations before a temperature change	50, 100, 500, 1000	500
MBO	Population size	5, 11, 21	5
	k	7, 11, 21	11
	x	3, 5, 10	5
	m	10, 20	20

Table 6
Optimal solutions obtained using CPLEX solver.

Instances	Nw	CPLEX solver						MBO			
		No setup		Low		High					
		Results	CPU	Results	CPU	Results	CPU	No setup	Low	High	CPU
P11	4	128	0.59	137	316.27	152	2693.95	128	137	152	1.21
P25	3	503	0.68	NA	3600	NA	3600	503	516	579	6.25
P25	4	327	2.44	NA	3600	NA	3600	327	346	380	6.25
P25	6	213	442.96	NA	3600	NA	3600	213	227	242	6.25
P25	9	134 (99.24)	2210.58*	NA	3600	NA	3600	121	131	142	6.25
P35	4	449	25.70	NA	3600	NA	3600	449	462	494	12.25
P35	5	344	78.18	NA	3600	NA	3600	344	355	392	12.25
P35	7	233 (213.66)	3355.58*	NA	3600	NA	3600	222	237	261	12.25
P35	12	148 (86.95)	2506.64*	NA	3600	NA	3600	112	118	131	12.25
P53	5	554	2310.04	NA	3600	NA	3600	559	574	619	28.09
P53	7	343 (270.11)	2563.51*	NA	3600	NA	3600	320	334	359	28.09
P53	10	322 (184.78)	3040.56*	NA	3600	NA	3600	239	256	276	28.09
P53	14	248 (116)	3600	NA	3600	NA	3600	162	170	185	28.09
P70	7	510(386.79)	1911.09*	NA	3600	NA	3600	448	469	507	49.00
P70	10	329(216.08)	3600	NA	3600	NA	3600	271	282	309	49.00
P70	14	346 (157)	3600	NA	3600	NA	3600	201	211	233	49.00
P70	19	600 (110)	3600	NA	3600	NA	3600	152	158	175	49.00

Note: In the third column, the number before the bracket is the upper bound and the number within the bracket is the lower bound, the CPU time is added with '*' when CPLEX terminates due to being out of memory, and NA means that no solution is achieved within the given CPU time.

and the achieved cycle times are regarded as the response variable in the ANOVA test. For reasons of space restrictions, this study does not present the detailed ANOVA results, but instead shows the parameters selected for the algorithms considered in Table 5. As there is one reported article addressing setup times in a two-sided robotic assembly line utilizing simulated annealing (SA)[27], this study also re-implements the SA algorithm using the reported encoding and decoding schemes.

4.2. Computational evaluation

This section presents a comparative study of the algorithms. Each algorithm is solved 30 times and the resulting best results for small-sized problems are compared with the optimal solution achieved by CPLEX solver and this is reported in Table 6. Three different criteria are tested in this paper (no setup time, low and high setup times). The first column in the table shows the problems tested denoted by their task size. The second column shows the number of workstations (*Nw*) for the problems. The next two columns present the cycle time obtained using a CPLEX solver as well as the computation

Table 7
Average RPD values for RALB problems with low setup times.

Instances	Nw	PSO	SA	GA	DCS	ABC	MBO
P11	4	0.00	0.00	0.00	0.00	0.00	0.00
P25	3	0.12	0.21	0.14	0.39	0.29	0.12
P25	4	0.00	0.00	0.00	0.00	0.00	0.00
P25	6	0.00	0.00	0.00	0.00	0.00	0.00
P25	9	1.15	0.53	0.61	0.23	0.15	0.31
P35	4	1.30	0.67	1.28	0.50	0.50	0.33
P35	5	8.34	4.17	5.38	2.00	0.85	0.00
P35	7	5.79	3.26	3.99	3.69	2.70	2.49
P35	12	9.58	2.29	2.80	1.27	0.93	1.10
P53	5	0.70	0.46	0.70	0.67	0.49	0.63
P53	7	1.08	0.75	0.78	0.45	0.00	0.00
P53	10	7.18	5.02	4.98	5.43	4.90	4.57
P53	14	8.49	4.82	5.12	3.13	3.19	3.43
P70	7	7.10	4.49	3.35	2.19	2.65	2.19
P70	10	10.60	5.48	4.09	3.84	1.42	1.00
P70	14	12.64	6.35	4.81	3.46	2.45	1.83
P70	19	13.86	7.15	3.92	1.90	1.01	0.89
P89	8	5.09	3.28	3.01	0.98	2.20	0.61
P89	12	6.77	3.46	3.54	2.11	1.72	1.15
P89	16	9.59	5.20	4.47	2.15	1.91	2.32
P89	21	15.00	8.80	5.00	3.46	2.16	2.55
P111	9	11.47	7.88	4.72	3.20	1.30	2.28
P111	13	15.84	8.88	6.38	4.19	2.04	2.22
P111	17	15.96	8.50	5.00	3.46	2.19	1.85
P111	22	18.96	10.95	4.68	3.43	1.79	1.29
P148	10	14.46	8.15	6.17	5.34	3.00	3.38
P148	14	16.89	10.16	4.95	4.82	2.90	3.06
P148	21	19.67	11.52	6.70	3.93	2.21	2.34
P148	29	21.08	12.75	5.98	3.14	1.76	1.18
P297	19	12.67	7.78	4.26	2.66	1.26	0.99
P297	29	16.42	9.06	4.17	2.51	0.87	0.39
P297	38	17.81	11.10	4.52	2.70	1.38	0.73
P297	50	18.88	12.95	5.75	2.76	1.87	0.37
Average RPD		9.83	5.64	3.67	2.42	1.58	**1.38**

time (CPU) when no setup times are considered. CPLEX is programmed in such a way that the program terminates when it reaches 3600 s. The next two columns show the results obtained for low setup times; however, only P11 with 11 tasks and 4 workstations could obtain a solution within the predetermined termination criteria. Similarly, for high setup times, CPLEX could only achieve a solution for the P11 problem. The results obtained by MBO are the best cycle times within 30 iterations with the termination criterion of $Nt \times NT \times 10$ milliseconds.

It is observed that CPLEX could only achieve 6 optimal solutions for no setup instances, and only one solution for both low setup times and high setup times. This situation proves the complexity of the considered problem. It is to be noted that the results obtained by CPLEX could be improved if we utilize more processors and larger RAM, but this research utilizes the same configuration for a fair comparison. It is also clear that MBO achieves the same or smaller cycle times for all the tested instances, and the results obtained by MBO are larger than or equal to the lower bounds. Notably, the results obtained using MBO outperform the results of CPLEX for all the instances in P53 and P70 with shorter computational times. This comparison demonstrates the superiority of the MBO algorithm in solving large-size instances and clarifies the reasons for utilizing some algorithms.

The selected metaheuristic algorithms for comparison were tested under different termination criteria to solve all the problem instances, as discussed earlier. Based on the results obtained, cycle times are transferred to relative percentage deviations (RPD) based on Eq. (12), where CT_{some} is the cycle time achieved by one combination and CT_{Best} is the cycle time achieved by all combinations.

$$RPD = 100 \cdot (CT_{some} - CT_{Best})/CT_{Best} \tag{11}$$

Based on the selected parameters, all algorithms are solved for 30 iterations and the RPD values are presented in Table 7. As there are 3 datasets, 6 termination criteria and 30 running times, there is a large amount of data and it is important to carefully analyze this data. Table 7 presents only the average RPD values obtained by algorithms for the considered problems with low setup times with τ=60 as an example. It is observed that MBO achieves the best performance with the overall RPD of 0.65, and ABC and DCS are the second and third best performers. The PSO and SA, on the contrary, are the worst and the second worst performers, with an overall RPD of 3.34 and 3.16. Surprisingly, MBO achieves the best performance for P111, P148 and P297, demonstrating the superiority of MBO in solving large-size instances.

Table 8
Average RPD values for RALB problems by algorithm.

No setup times						
Algorithms	$Nt \times Nt \times 10$	$Nt \times Nt \times 20$	$Nt \times Nt \times 30$	$Nt \times Nt \times 40$	$Nt \times Nt \times 50$	$Nt \times Nt \times 60$
PSO	10.83	10.15	9.81	9.53	9.36	9.15
SA	5.37	5.37	5.36	5.36	5.36	5.35
GA	4.65	4.24	3.95	3.79	3.67	3.54
DCS	3.60	3.13	2.90	2.71	2.62	2.50
ABC	2.82	2.22	1.98	1.82	1.71	1.64
MBO	2.16	1.69	1.43	1.22	1.07	0.97
Low setup time variability						
Algorithms	$Nt \times Nt \times 10$	$Nt \times Nt \times 20$	$Nt \times Nt \times 30$	$Nt \times Nt \times 40$	$Nt \times Nt \times 50$	$Nt \times Nt \times 60$
PSO	11.50	10.89	10.46	10.22	9.99	9.83
SA	5.65	5.65	5.64	5.64	5.64	5.64
GA	4.85	4.32	4.09	3.90	3.78	3.67
DCS	3.69	3.21	2.92	2.67	2.54	2.42
ABC	3.11	2.38	2.03	1.81	1.68	1.58
MBO	2.63	2.15	1.86	1.67	1.50	1.38
High setup time variability						
Algorithms	$Nt \times Nt \times 10$	$Nt \times Nt \times 20$	$Nt \times Nt \times 30$	$Nt \times Nt \times 40$	$Nt \times Nt \times 50$	$Nt \times Nt \times 60$
PSO	13.48	12.75	12.32	12.00	11.79	11.64
SA	6.66	6.65	6.65	6.65	6.65	6.65
GA	5.52	4.94	4.68	4.41	4.23	4.11
DCS	4.38	3.71	3.32	3.01	2.78	2.64
ABC	3.84	2.78	2.24	1.96	1.73	1.59
MBO	3.11	2.44	2.06	1.80	1.61	1.47

Table 8 presents the overall RPD values obtained for the 33 datasets for the three scenarios (no setup times, low and high setup time variability) for the six metaheuristics considered. The average RPD values are obtained for each algorithm by taking the average RPD values for all the problems. Different termination criteria (six in total) are tested to observe the performance of the considered metaheuristic algorithms. It should be noted that the detailed results (average RPD, best cycle times, average cycle times and standard deviation) for each instance are omitted for space reasons, but they are available upon request. From this table, it can be observed that the proposed MBO is able to perform better than the other metaheuristic algorithms for all termination criteria.

Notably, for no setup times, the MBO performed better than other algorithms for all the termination criteria. For termination criterion with $Nt \times Nt \times 10$ ms, MBO ranked first, followed by DCS, ABC, GA, SA and PSO. For other termination criteria, MBO also performs better than the other algorithms and ranks first among them. For low setup time variability and high setup time variability, MBO performs the best among these algorithms with the termination criterion of $Nt \times Nt \times 10$ milliseconds, and it remains the best performer across all the other five tested criteria. Again, PSO and SA are the two worst performers for all the termination criteria. In summary, this computational study suggests that MBO is the best performer among all the algorithms with regard to no setup times, low, and high setup time variability. Furthermore, the results show that setup times cannot be ignored in a robotic assembly line and researchers should include setup times when considering how to balance a robotic assembly line. We have omitted the detailed cycle times which were used to calculate the RPD values for each problem; they are available on request and will be archived in Research Gate. The superiority of the proposed MBO should be attributed to the combination of the problem-specific improvements and the strong local search ability of the original MBO. This method utilizes an improved decoding procedure and an iterative mechanism in order to preserve the minor improvements made to individuals. Moreover, having the MBO replace the incumbent individual immediately after the improvement is made, this allows searching for more promising areas and helps avoid unnecessary searching around a poor individual. Meanwhile, the proposed MBO utilizes two improvements to enhance exploration capacity. (1) The incumbent individual is replaced with the new neighbor solution when the same fitness value is achieved. (2) The fitness of the neighbor solution is set to a very large positive value if it shares the same fitness as the incumbent one. Without these two improvements, all the individuals will have the same fitness value after a few iterations and the algorithm might show premature convergence. In short, these problem-specific improvements are in favor of the MBO maintaining a proper balance between exploration and exploitation.

Although the difference is quite clear, carrying out statistical analysis to confirm that the observed difference is statistically significant is still advised. As the performances of algorithms are quite different when solving different instances, this research utilizes the average RPD of all instances in one run as the response variable. Subsequently, the ANOVA test is carried out where the algorithm type and termination criteria are regarded as two controlled factors. The ANOVA analysis demonstrates that there are statistically significant differences between these methods, the termination criteria and the interaction of the two factors. Fig. 4 depicts the means plots of algorithms for three scenarios, with Fig. 4a showing no setup times, Fig. 4b showing low setup time variability and Fig. 4c showing high setup time variability. For better readability, these

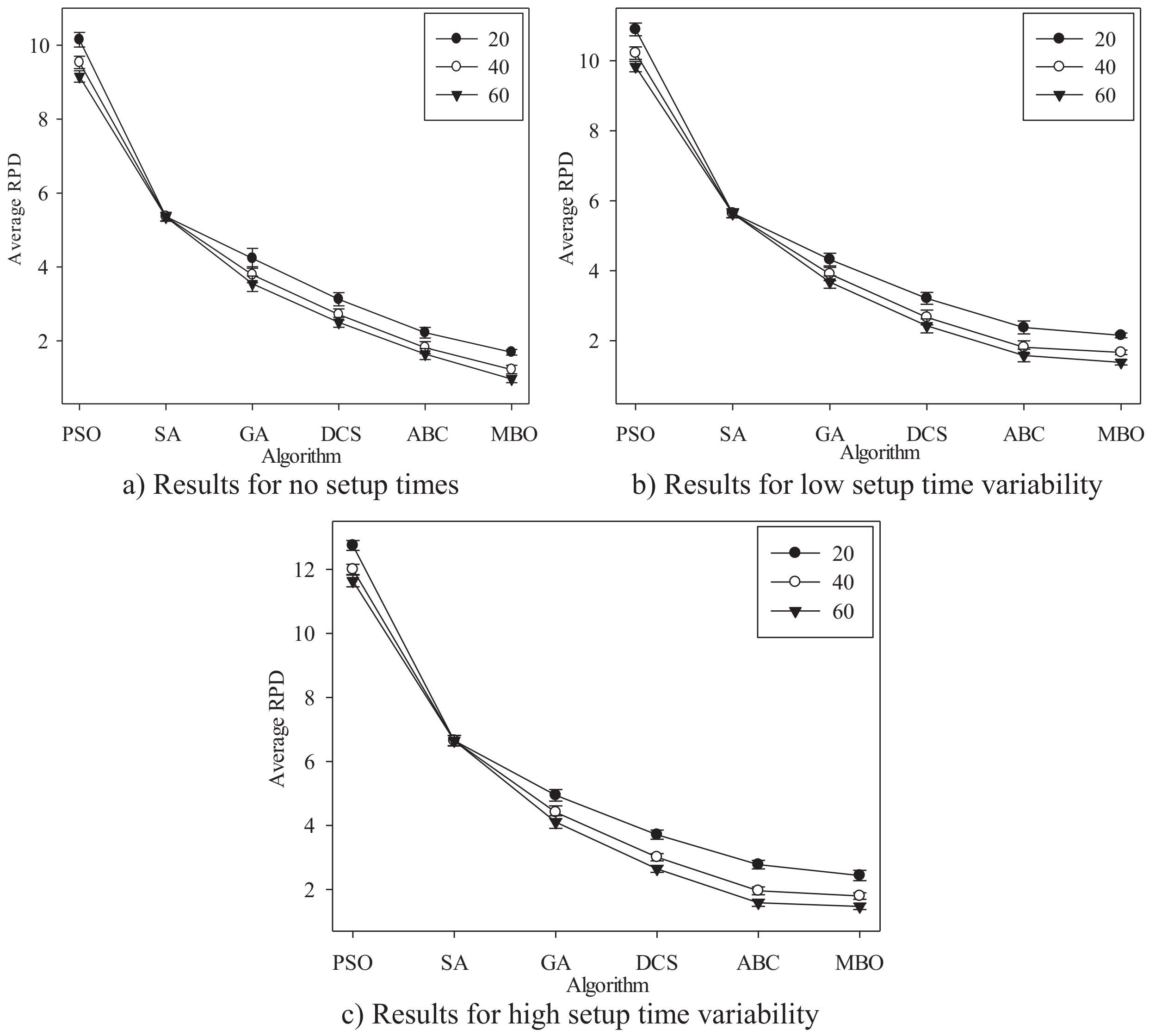

Fig. 4. Means plot and 95% Tukey HSD confidence intervals for the interactions between algorithms and termination criteria.

figures primarily present the results when τ=20, 40 and 60 are tested. It can be clearly seen from the figures that MBO is the best performer in all three scenarios based on these three termination criteria. It can also be seen that SA is trapped in the local optima and cannot find better results with increased CPU time. MBO ranks first, DCS and ABC rank second and third, followed by GA, SA and PSO in terms of their performance in solving the considered problem. The statistical analysis also concludes that the proposed MBO obtains superior results, as shown in Table 7 and Table 8.

5. Conclusion and future research

This paper presents the methodology used to address the problem of balancing a robotic assembly line with sequence-dependent setup times. The objective is to optimize the cycle time; the paper presents a mathematical model for the considered problem, and a CPLEX solver is used to solve small-sized problems. Three different scenarios are tested (no setup time, low and high setup times) and, since the considered problem falls under the NP-hard category, a recently developed metaheuristic Migrating Birds Optimization (MBO) algorithm, along with a set of four metaheuristics, is implemented to solve the problem. Average RPD (Relative Percentage Deviation) values for the three scenarios and all the considered metaheuristics are presented in detail. From an analysis of the results, it can be seen that the proposed MBO algorithm performed better for all three scenarios with different termination criteria. The study can be implemented in a decision support system and production managers can use the findings of this study and apply it in real-time scenarios. The results were obtained within a short computational time; hence, results can be quickly analyzed, facilitating the effective and high-quality design and implementation of a system.

This study's outcome will help production managers test different possible scenarios for balancing a robotic assembly line with sequence-dependent setup times and determine a feasible balanced solution within an acceptable computational time. This system can be integrated into decision support systems for real-time usage. In the future, different layouts of robotic assembly lines with setup times could be compared. It would also be interesting to examine the performance of hybrid metaheuristic algorithms for problems of this type. As the industry contexts are quite diverse, there might be backward setup times on some occasions and thus it would be of interest to address this gap. Researchers could consider including more realistic situations such as constraints on the allocation of tasks to the robots. Another interesting area of work would be to consider a semi-robotic assembly line situation where both robots and human workers are required to execute tasks.

Acknowledgement

The second author acknowledges the financial support from the National Natural Science Foundation of China under grant 61803287.

References

[1] A. Scholl, C. Becker, State-of-the-art exact and heuristic solution procedures for simple assembly line balancing, Eur. J. Oper. Res. 168 (2006) 666–693.
[2] M. Li, Q. Tang, Q. Zheng, X. Xia, C. Floudas, Rules-based heuristic approach for the U-shaped assembly line balancing problem, Appl. Math. Model. 48 (2017) 423–439.
[3] G. Levitin, J. Rubinovitz, B. Shnits, A genetic algorithm for robotic assembly line balancing, Eur. J. Oper. Res. 168 (2006) 811–825.
[4] J.M. Nilakantan, S. Ponnambalam, N. Jawahar, G. Kanagaraj, Bio-inspired search algorithms to solve robotic assembly line balancing problems, Neural Comput. Appl. 26 (2015) 1379–1393.
[5] J. Gao, L. Sun, L. Wang, M. Gen, An efficient approach for type II robotic assembly line balancing problems, Comput. Ind. Eng. 56 (2009) 1065–1080.
[6] A. Roshani, A. Roshani, A. Roshani, M. Salehi, A. Esfandyari, A simulated annealing algorithm for multi-manned assembly line balancing problem, J. Manuf. Syst. 32 (2013) 238–247.
[7] J. Rubinovitz, J. Bukchin, Design and balancing of robotic assembly lines, in: Proceedings of the Fourth World Conference on Robotics Research Pittsburgh, PA, SME 1991, 1991.
[8] J. Rubinovitz, J. Bukchin, E. Lenz, RALB–A heuristic algorithm for design and balancing of robotic assembly lines, CIRP Ann. Manuf. Technol. 42 (1993) 497–500.
[9] A. Sepahi, S.G.J. Naini, Two-sided assembly line balancing problem with parallel performance capacity, Appl. Math. Model. 40 (2016) 6280–6292.
[10] A. Yoosefelahi, M. Aminnayeri, H. Mosadegh, H.D. Ardakani, Type II robotic assembly line balancing problem: An evolution strategies algorithm for a multi-objective model, J. Manuf. Syst. 31 (2012) 139–151.
[11] J. Mukund Nilakantan, G.Q. Huang, S.G. Ponnambalam, An investigation on minimizing cycle time and total energy consumption in robotic assembly line systems, J. Cleaner Prod. 90 (2015) 311–325.
[12] Z.A. Çil, S. Mete, K. Ağpak, Analysis of the type II robotic mixed-model assembly line balancing problem, Eng. Optim. (2016) 1–20.
[13] M. Rabbani, Z. Mousavi, H. Farrokhi-Asl, Multi-objective metaheuristics for solving a type II robotic mixed-model assembly line balancing problem, J. Ind. Prod. Eng. 33 (2016) 472–484.
[14] Z.A. Çil, S. Mete, K. Ağpak, A goal programming approach for robotic assembly line balancing problem, IFAC-PapersOnLine 49 (2016) 938–942.
[15] Z.A. Çil, S. Mete, E. Özceylan, K. Ağpak, A beam search approach for solving type II robotic parallel assembly line balancing problem, Appl. Soft Comput. 61 (2017) 129–138.
[16] Z. Li, M.N. Janardhanan, Q. Tang, P. Nielsen, Mathematical model and metaheuristics for simultaneous balancing and sequencing of a robotic mixed–model assembly line, Eng. Optim. 50 (2018) 877–893.
[17] J.M. Nilakantan, Z. Li, Q. Tang, P. Nielsen, Multi-objective co-operative co-evolutionary algorithm for minimizing carbon footprint and maximizing line efficiency in robotic assembly line systems, J. Cleaner Prod. 156 (2017) 124–136.
[18] A. Allahverdi, H. Soroush, The significance of reducing setup times/setup costs, Eur. J. Oper. Res. 187 (2008) 978–984.
[19] U. Özcan, B. Toklu, Balancing two-sided assembly lines with sequence-dependent setup times, Int. J. Prod. Res. 48 (2010) 5363–5383.
[20] C. Andres, C. Miralles, R. Pastor, Balancing and scheduling tasks in assembly lines with sequence-dependent setup times, Eur. J. Oper. Res. 187 (2008) 1212–1223.
[21] A. Scholl, N. Boysen, M. Fliedner, The sequence-dependent assembly line balancing problem, OR Spectrum 30 (2008) 579–609.
[22] A. Yolmeh, F. Kianfar, An efficient hybrid genetic algorithm to solve assembly line balancing problem with sequence-dependent setup times, Comput. Ind. Eng. 62 (2012) 936–945.
[23] A. Scholl, N. Boysen, M. Fliedner, The assembly line balancing and scheduling problem with sequence-dependent setup times: problem extension, model formulation and efficient heuristics, OR Spectrum 35 (2013) 291–320.
[24] Ş. Akpinar, A. Baykasoğlu, Modeling and solving mixed-model assembly line balancing problem with setups. Part II: A multiple colony hybrid bees algorithm, J. Manuf. Syst. 33 (2014) 445–461.
[25] S. Seyed-Alagheband, S.F. Ghomi, M. Zandieh, A simulated annealing algorithm for balancing the assembly line type II problem with sequence-dependent setup times between tasks, Int. J. Prod. Res. 49 (2011) 805–825.
[26] N. Hamta, S.F. Ghomi, F. Jolai, M.A. Shirazi, A hybrid PSO algorithm for a multi-objective assembly line balancing problem with flexible operation times, sequence-dependent setup times and learning effect, Int. J. Prod. Econ. 141 (2013) 99–111.
[27] M. Aghajani, R. Ghodsi, B. Javadi, Balancing of robotic mixed-model two-sided assembly line with robot setup times, Int. J. Adv. Manuf. Technol. 74 (2014) 1005–1016.
[28] E. Ulker, V. Tongur, Migrating birds optimization (MBO) algorithm to solve knapsack problem, Proc. Comput. Sci. 111 (2017) 71–76.
[29] I. Benkalai, D. Rebaine, C. Gagné, P. Baptiste, Improving the migrating birds optimization metaheuristic for the permutation flow shop with sequence-dependent set-up times, Int. J. Prod. Res. (2017) 1–13.
[30] E. Duman, M. Uysal, A.F. Alkaya, Migrating Birds Optimization: A new metaheuristic approach and its performance on quadratic assignment problem, Inf. Sci. 217 (2012) 65–77.
[31] V. Tongur, E. Ülker, Migrating birds optimization for flow shop sequencing problem, J. Comput. Commun. 2 (2014) 142.
[32] A. Sioud, C. Gagné, Enhanced migrating birds optimization algorithm for the permutation flow shop problem with sequence dependent setup times, Eur. J. Oper. Res. 264 (2018) 66–73.
[33] Z. Li, I. Kucukkoc, J.M. Nilakantan, Comprehensive review and evaluation of heuristics and meta-heuristics for two-sided assembly line balancing problem, Comput. Oper. Res. 84 (2017) 146–161.
[34] J. Mukund Nilakantan, S.G. Ponnambalam, Robotic U-shaped assembly line balancing using particle swarm optimization, Eng. Optim. 48 (2016) 231–252.

[35] Z. Li, M.N. Janardhanan, Q. Tang, P. Nielsen, Co-evolutionary particle swarm optimization algorithm for two-sided robotic assembly line balancing problem, Adv. Mech. Eng. 8 (2016) 1687814016667907.
[36] Y.K. Kim, J.Y. Kim, Y. Kim, A coevolutionary algorithm for balancing and sequencing in mixed model assembly lines, Appl. Intel. 13 (2000) 247–258.
[37] R.B. Taha, A.K. El-Kharbotly, Y.M. Sadek, N.H. Afia, A Genetic Algorithm for solving two-sided assembly line balancing problems, Ain Shams Eng. J. 2 (2011) 227–240.
[38] D.I. Petropoulos, A.C. Nearchou, A particle swarm optimization algorithm for balancing assembly lines, Assemb. Autom. 31 (2011) 118–129.
[39] Q. Tang, Z. Li, L. Zhang, An effective discrete artificial bee colony algorithm with idle time reduction techniques for two-sided assembly line balancing problem of type-II, Comput. Ind. Eng. 97 (2016) 146–156.
[40] U. Saif, Z. Guan, W. Liu, C. Zhang, B. Wang, Pareto based artificial bee colony algorithm for multi objective single model assembly line balancing with uncertain task times, Comput. Ind. Eng. 76 (2014) 1–15.
[41] Z. Li, N. Dey, A.S. Ashour, Q. Tang, Discrete cuckoo search algorithms for two-sided robotic assembly line balancing problem, Neural Comput. Appl. (2017) 1–12.