Mathematical models and migrating birds optimization for robotic U-shaped assembly line balancing problem

Zixiang Li[1, 2], Mukund Nilakantan Janardhanan[3*], Amira S. Ashour[4], Nilanjan Dey[5]

[1]Key Laboratory of Metallurgical Equipment and Control Technology, Wuhan University of Science and Technology, China. Email: zixiangliwust@gmail.com

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

[3]Mechanics of Materials Research Group, Department of Engineering, University of Leicester, Leicester, United Kingdom. Email: mukund.janardhanan@leicester.ac.uk

[4]Department of Electronics and Electrical Communication Engineering, Faculty of Engineering, Tanta University, Egypt. Email: amirasashour@yahoo.com

[5]Department of Information Technology, Techno India College of Technology, Kolkata, India. Email: neelanjandey@gmail.com

Abstract: Modern assembly line systems utilize robotics to replace human resources to achieve higher level of automation and flexibility. This work studies the task assignment and robot allocation in a robotic U-shaped assembly line. Two new mixed integer programming linear models are developed to minimize the cycle time when the number of workstations is fixed. Recently developed migrating birds optimization (MBO) algorithm is employed and improved to solve large-sized problems. Problem-specific improvements are also developed to enhance the proposed algorithm including modified consecutive assignment procedure for robot allocation, iterative mechanism for cycle time update, new population update mechanism and diversity controlling mechanism. An extensive comparative study is carried out to test the performance of the proposed algorithm, where seven high-performing algorithms recently reported in the literature are re-implemented to tackle the considered problem. The computational results demonstrate that the developed models are capable to achieve the optimal solutions for small-sized problems, and the proposed algorithm with these proposed improvements achieves excellent performance and outperforms the compared ones.

Keywords: Robotic U-shaped assembly line; Integer programming; Migrating birds optimization; Artificial intelligence

1. Introduction

Assembly lines have great applications in assembling standardized products and widely used across industries. Assembly lines can be categorized into three types based on their layouts: one-sided (straight) assembly line, two-sided assembly line and U-shaped assembly line. U-shaped assembly line differs from the other two assembly line types since there exists an entrance side and exit side. A task in U-shaped assembly line is assignable when all its predecessors or successors have been allocated to the same or earlier workstation (Miltenburg and Wijngaard, 1994; Scholl and Klein, 1999). When compared with the straight assembly line, U-shaped line has higher flexibility to meet the changes in demand with reduced cycle time and reduced cost (Nilakantan and Ponnambalam, 2016).

Robots play an essential role in modern manufacturing industry and it helps to reduce assembly cost and achieve higher flexibility. Robots can be programmed to operate different tasks and operate for 24 hours a day without fatigue (Gao et al., 2009). The assembly line equipped with robots to perform the tasks is referred to as robotic assembly line. Robotic assembly line balancing problem (RALBP) aims to assign tasks and allocate the best fit robots to workstations (Nilakantan et al., 2015a). If utilizing robots

in U-shaped assembly line, a new assembly line called robotic U-shaped assembly line arises. To increase the efficiency of robotic U-shaped assembly line, optimization methods are necessary to reduce the workstation number or the cycle time. This results in robotic U-shaped assembly line balancing problem (RUALBP), where tasks and best fit robots are allocated to workstations in U-shaped assembly line with one or several optimization criteria.

Regarding the literature reported on the U-shaped assembly line, Miltenburg and Wijngaard (1994) propose dynamic programming formulation to minimize the number of workstation numbers. Later, Urban (1998) present an integer programming formulation for the same problem. In addition, many researchers applied exact, heuristic and meta-heuristic methods for this problem. Scholl and Klein (1999) develop an exact method based on a branch and bound known as ULINO. Other studies on exact methods include those reported in Miltenburg (1998), Nakade and Ohno (1999), Gokcen and Agpak (2006) and Ogan and Azizoglu (2015). Regarding the literature with regards to heuristic and meta-heuristic methods, the applied methods include simulated annealing algorithms (Erel et al., 2001), a genetic algorithm (Hwang et al., 2008), ant colony optimizations (Baykasoglu and Dereli, 2009; Li et al., 2017d; Sabuncuoglu et al., 2009) and a critical path method (Avikal et al., 2013), to cite just a few. Many variants of U-shaped assembly line balancing problem have been solved using heuristics and meta-heuristics algorithms due to the NP-hard nature of the problem (Bagher et al., 2011; Chiang and Urban, 2006; Kim et al., 2006; Kim et al., 2000; Kucukkoc and Zhang, 2015; Rabbani et al., 2012).

The studies related to the robotic assembly line balancing are categorized based on the layout of the assembly lines and they are: general RALBP, RUALBP and robotic two-sided assembly line balancing problems (RTALBP). Rubinovitz and Bukchin (1991) report the seminal study related to the RALBP which is followed by another study that reports a heuristic method to solve the problem (Rubinovitz et al., 1993). Later, Levitin et al. (2006) and Gao et al. (2009) develop well-known genetic algorithm to tackle type II RALBPs. Yoosefelahi et al. (2012) address a multi-objective RALBP and Daoud et al. (2014) optimize the line efficiency. Nilakantan et al. (2015b) propose two bio-inspired methods for RALBP with the objective of minimizing cycle time and later they investigate energy consumption optimization in robotic assembly line (Nilakantan et al., 2015a). More recently, Çil et al. (2017a) propose the beam search to solve the type II mixed-model RALBP and Çil et al. (2017b) extend this method for parallel RALBP. Furthermore, type II RTALBP is tackled in Li et al. (2016a) and Li et al. (2017a). Li et al. (2016b) study the energy consumption using multi-objective simulated annealing algorithm, while Aghajani et al. (2014) tackle the mixed-model RTALB with simulated annealing algorithm. Nilakantan and Ponnambalam (2016) employ the particle swarm optimization to solve type II RUALBP. From the aforementioned literature, it is established that there is limited research on RUALBP. The only available nonlinear model proposed by Nilakantan and Ponnambalam (2016) is hard to be solved optimally using exact techniques for large-sized or even small-sized problems and it is also observed that only few evolutionary algorithms have been utilized in solving RUALBP, whereas literature reports extensive usage of evolutionary algorithms for solving RALBP problems.

Hence, this work presents two major contributions to the literature as follows: 1) Two new mixed-integer linear programming models are developed to minimize the cycle time in a U-shaped robotic assembly line. These two new linear models outperform the non-linear model proposed in Nilakantan and Ponnambalam (2016) in the comparative study conducted. 2) A newly developed metaheuristic algorithm, migrating birds optimization (MBO) algorithm is employed and improved to tackle the considered RUALBP in an acceptable computational time. It is to be noted that, this is for the first time MBO is applied to solve RUALBP. In addition, this research also improves and enhances the performance

of MBO by employing several problem-specific improvements such as modified consecutive assignment procedure for robot selection, iterative mechanism for cycle time update, new population update mechanism and diversity controlling mechanism. MBO is selected due to its superiority over others in solving problems similar to the considered problems such as planning and scheduling area as reported in Duman et al. (2012), Gao and Pan (2016) and Zhang et al. (2017). A comprehensive comparative study demonstrates that these improvements enhance the performance of MBO to a large extent, and the proposed MBO outperforms eight other algorithms statistically.

The outline of the remaining sections is presented as follows. Section 2 introduces the developed models. Section 3 details the proposed MBO. An example is provided in Section 4, and the proposed algorithm is evaluated in Section 5, where a comprehensive comparative campaign is conducted. Finally, Section 6 concludes this research.

2. Mathematical model

Although many researchers have presented the mathematical models for U-shaped assembly line such as Urban and Chiang (2006), Fattahi and Turkay (2015) and Li et al. (2017d). However, none of the models report the considered RUALBP or is able to tackle robot assignment. The only available model on RUALBP is a nonlinear model proposed by Nilakantan and Ponnambalam (2016) and this model might not be able to achieve the optimal solution even for small-sized instances. Hence, this section develops two new mixed-integer linear programming models to solve the considered problem.

2.1 Problem description and assumptions

Robotic U-shaped assembly line inherits the main features of U-shaped assembly line and robotic assembly line. On the basis of assumptions considered in Levitin et al. (2006) and Nilakantan and Ponnambalam (2016), the main assumptions in this paper are presented as follows:

- Each task is assigned to a workstation and operated by a robot.
- Each workstation is allocated with only one robot.
- The number of utilized robots is equal to the number of workstations.
- The processing time of a task depends on the robot assigned in the workstation.
- A task can be operated by any robot or the processing time by a robot is set to a very large positive number when this task cannot be operated by this robot.
- All types of robots are available without any limitations.
- Only one type of product is assembled in this robotic U-shaped line.
- The material handling, set-up, tool changing, loading and unloading factors are negligible.

RUALBP comprises of two interrelated sub-problems: task assignment and robot allocation. For task assignment, the assembly tasks are to be assigned to workstations without violating the precedence constraint and cycle time constraint. Specifically, a task is assignable when all its predecessors or successors have been assigned. The total operation time of the tasks on each workstation should be equal or smaller than the cycle time. On the other hand, robot allocation allocates the best robot to each workstation. A layout of the robotic U-shaped assembly line is illustrated in Figure 1, where 10 tasks are assigned to five workstations equipped with five robots.

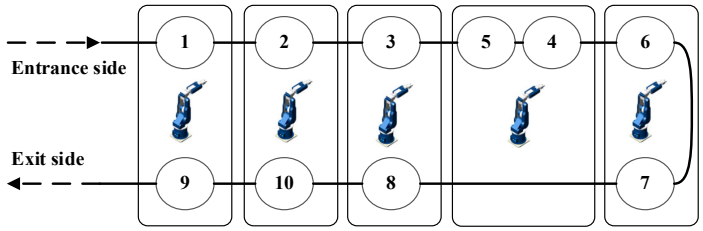

Figure 1. Layout of robotic U-shaped assembly line

2.2 Proposed mathematical models

This section presents two mathematical models to formulate the RUALB, notations used in the models are introduced first.

Indices:

i, p, q	A task, $i \in$ I.
j	A workstation, $j \in$ J.
r	A robot, $r \in$ R.

Parameters:

CT	Cycle time.
m	Upper bound on the number of workstations.
t_{ir}	Operation time of task i by robot r.

Decision Variables:

a_{irj}	Binary variable. a_{irj} is equal to 1 when task i is operated by robot r on workstation j; 0, otherwise.
x_{irj}	Binary variable. x_{irj} is equal to 1 when task i is operated by robot r on the entrance side of workstation j; 0, otherwise.
y_{irj}	Binary variable. y_{irj} is equal to 1 when task i is operated by robot r on the exit side of workstation j; 0, otherwise.
w_{rj}	Binary variable. w_{rj} is equal to 1 when robot r is allocated to workstation j; 0, otherwise.
u_i	Binary variable. u_i is equal to 1 when task i is allocated to the entrance side; 0, otherwise.

For RUALBP, the main characteristic is the precedence relationship which differentiates the RUALBP from the RALBP. Specifically, a task is assignable when all its predecessors or successors have been allocated in RUALBP. On the basis of Fattahi and Turkay (2015), the first model (Model 1) is built as follows. In this model, $\wp$ denotes the set of immediate precedence relationships ($\wp=\{(p, q)\}$, where p is the immediate predecessor of task q). Major difference from Fattahi and Turkay (2015) is that the developed Model 1 utilizes a_{irj} and w_{rj} to describe the allocation of tasks and robots

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

$$\sum_{r\in R}\sum_{j\in J} a_{irj}=1 \quad \forall i \in \text{I} \tag{2}$$

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

$$\sum_{r\in R}\sum_{j\in J} j \cdot a_{prj} - \sum_{r\in R}\sum_{j\in J} j \cdot a_{qrj} \le m \cdot (1+u_p-2\cdot u_q) \quad \forall (p,q) \in \wp \tag{4}$$

$$\sum_{r\in R}\sum_{j\in J} j \cdot a_{qrj} - \sum_{r\in R}\sum_{j\in J} j \cdot a_{prj} \le m \cdot u_p \quad \forall (p,q) \in \wp \tag{5}$$

$$\sum_{i\in I}\sum_{r\in R} t_{ir} \cdot a_{irj} \le CT \quad \forall j \in J \tag{6}$$

$$\sum_{i\in I} a_{irj} \le \psi \cdot w_{rj} \quad \forall r \in R, j \in J \tag{7}$$

The objective in expression (1) minimizes the cycle time. Constraint (2) ensures that each task is assigned to one workstation and operated by a robot. Constraint (3) guarantees that each workstation is equipped with a robot. Constraint (4) and (5) tackle precedence constraints. Constraint (4) makes sure that task q (the successor of task p) is allocated to the latter or the same workstation when both tasks are operated on the entrance side, whereas constraint (5) makes sure that task q is allocated to the former or the same workstation when both tasks are operated on the exit side. These two constraints also permit the allocation of task p to the entrance side and task q to the exit side and deny the allocation of task p to the exit side and task q to the entrance side. Constraint (6) deals with the cycle time constraint, demanding that the total operation time on each workstation is less than or equal to cycle time. Constraint (7) connects a_{irj} and w_{rj}, which guarantees that tasks on the same workstation are operated by the same robot.

The second model (Model 2) which proposes a different method to tackle the precedence constraint is developed based on the model reported in Urban and Chiang (2006). Different from Urban and Chiang (2006), the developed Model 2 utilizes x_{irj} and y_{irj} the task assignment, and w_{rj} to describe robot allocation.

$$\text{Minimize } CT \tag{8}$$

$$\sum_{r\in R}\sum_{j\in J}\left(x_{irj} + y_{irj}\right) = 1 \quad \forall i \in \mathrm{I} \tag{9}$$

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

$$\sum_{r\in R}\sum_{j\in J}(m - j + 1)\cdot\left(x_{prj} - x_{qrj}\right) \ge 0 \quad \forall (p,q) \in \wp \tag{11}$$

$$\sum_{r\in R}\sum_{j\in J}(m - j + 1)\cdot\left(y_{qrj} - y_{prj}\right) \ge 0 \quad \forall (p,q) \in \wp \tag{12}$$

$$\sum_{i\in I}\sum_{r\in R} t_{ir}\cdot\left(x_{irj} + y_{irj}\right) \le CT \quad \forall j \in J \tag{13}$$

$$\sum_{i\in I}\left(x_{irj} + y_{irj}\right) \le \psi \cdot w_{rj} \quad \forall r \in R, j \in J \tag{14}$$

Similarly, Equation (8) optimizes the cycle time. Constraint (9) guarantees that each task is allocated to the entrance side or exit side of a workstation and operated by a robot. Similar to Model 1, Constraint (10) also allocates a robot to each workstation. Constraint (11) and Constraint (12) deal with the precedence relationship. They achieve the same goal as the expression (4) and expression (5) in Model 1. Inequality (11) requires that task q (the successor of task p) is allocated to the latter or the same workstation when task p and task q are operated on the entrance side. And Constraint (12) guarantees that task q is allocated to the former or the same workstation when task p and task q are operated on the exit side. Constraint (13) makes sure that the total operation time of tasks allocated to entrance side and exit side of each workstation is smaller than or equal to the cycle time. Constraint (14) connects x_{irj}, y_{irj} and w_{rj}, ensuring that the tasks on the entrance side or exit side of the same workstation are operated by the same robot. Model 1 and Model 2 are capable to solve small-sized instances using Cplex solver for and the results obtained using these two models will be compared with non-linear model solution in Section 5.2.

3. Migrating birds optimization

Migrating birds optimization (MBO) is a bio-inspired meta-heuristic algorithm developed based on birds' migration behavior in a V-shaped flight (Duman et al., 2012). Since the first work in Duman et al. (2012), this method has attracted many researchers attention (Gao and Pan, 2016; Zhang et al., 2017). It has been reported that this method and has the capability of outperforming many other metaheuristic methods for solving combinatorial optimization problems. Along with the MBO's effectiveness, this work proposes the first attempt in employing this method to solve RUALBP. The proposed MBO algorithm is introduced in the following sections.

3.1 Solution representation

In order to tackle the RUALBP, this work utilizes a task permutation vector for encoding based on the procedure reported in Nilakantan and Ponnambalam (2016) and Nilakantan et al. (2015b). For instance, one possible encoding for 11 tasks is {1, 3, 2, 4, 5, 6, 7, 9, 8, 10, 11}. The former tasks in this task permutation are allocated at first, e.g. task 1 is firstly allocated. The task permutation is not a feasible solution and a decoding procedure is requisite. This work utilizes a decoding procedure inheriting the consecutive assignment procedure in Levitin et al. (2006) and the iterative mechanism for type II two-sided assembly line in Li et al. (2017c). The decoding procedure, referred as modified consecutive assignment procedure, is explained as follows. The main difference between the proposed method and the method proposed in Levitin et al. (2006) is with respect to assigning the task when it can be finished within cycle time by any robot. By making this difference, the proposed method allocates as many as possible tasks to this workstation. On the contrary, the original consecutive assignment procedure allocates maximal number of tasks that a robot can perform in the given task permutation. In other words, the proposed method allows the task allocation sequence to be conflicted for the given task permutation, whereas the original method requires the task allocation sequence to be same as the given task permutation.

Procedure: Proposed modified consecutive assignment procedure

***Step 1*:** Set the initial cycle time.

***Step 2*:** Open a new workstation.

***Step 3*:**

- ***Step 3.1*:** Add the tasks whose predecessors or successors have been allocated to the assignable task set.
- ***Step 3.2*:** Remove the task, whose possible finishing time is larger than cycle time when being operated by any robot, from the assignable task set for all the former workstations except for the last workstation.
- ***Step 3.3*:** If no assignable task exists, go to Step 2; else, execute Step 3.4.
- ***Step 3.4*:** Assign the task on the former position of the corresponding task permutation.
- ***Step 3.5*:** Update the remained capacity of the current workstation.

***Step 4*:** Allocate the best fit robot to the current workstation and assign the tasks operated by the best fit robot to the current workstation.

***Step 5*:** If all tasks have been allocated, terminate the decoding procedure. Otherwise, execute Step 2.

Another underlying issue is determining the initial cycle time. In this work, the initial cycle time is set to a big value as $CT = 2 \cdot \sum_{i \in I} \sum_{r \in R} t_{ir} / (Nr \cdot Ns)$, where *Nr* is the number of the available robot types and *Ns* is the number of workstations. This initial cycle time is updated when new best cycle time, CT_{Best}, is achieved using $CT = CT_{Best} - 1$. Furthermore, once a new best cycle time is achieved, all the individuals are re-decoded using this new CT and their objective values are updated accordingly. The initial cycle time update ensures that the achieved cycle time is decreasing gradually. This method, referred to as iterative mechanism, is developed based on a similar concept reported in Li et al. (2017c). It ensures that all the individuals are evaluated using the same initial cycle time and helps to preserve the tiny improvements on the individuals.

3.2 Classic migrating birds optimization algorithm

MBO algorithm is inspired by birds' migration behavior in a V-shaped flight, where one leading bird leads the whole flock containing two set of birds on left and right sides. The birds in each side fly in a line resulting in V-shape. The algorithm has been applied to solve quadratic assignment problem and solve combinatorial optimization problems successfully (Ulker and Tongur, 2017).

MBO consists of four main steps and have four parameters, where *n* represents the number of initial individuals, *k* is the number of neighbor solutions, *x* is the number of neighbor solutions to be shared with the next individual and *m* is the number of tours. MBO starts with population initialization, where *n* initial individuals are generated. Subsequently, leader improvement, block improvement and leader replacement consist of the main loop. These steps in this loop are repeated until a termination criterion is satisfied. Within the loop, the leader is tried for improvement by generating *k* neighbor solutions (leader improvement). If the improvement is achieved, the current leader is replaced with the best neighbor individual among all the neighbor solutions. In addition, the other individuals are updated when its (*k*-*x*) neighbor solutions (block improvement) or *x* unused best neighbor individuals of the front solution front achieves the better fitness. The sharing of neighbor solutions of other individuals (referred to as benefit mechanism) promotes the communication among individuals and the evolution of the whole population. After the leader improvement and the block improvement are conducted for *m* consecutive times, the leader replacement is carried out, which moves the leading individual to the end and forward one of the following individuals to the leading position.

As the considered problem is a discrete optimization problem, this research utilizes swap operation and insert operation following Li et al. (2016a) and Li et al. (2017a) in order to achieve high-quality neighbor solutions. Specifically, a number between 0.0 and 1.0 is randomly generated. If this number is less than 0.5, swap operation is performed; otherwise, insert operation is performed. The utilization of two neighbor operations helps to increase the search space.

Generate *n* initial individuals randomly; **% Initialization**
While (Termination criterion is not met) **do**
  For *i*=1 ***to*** *m* **do**
    Generate *k* neighbor solutions of the leader solution. **%Leader improvement**
    Replace the current leader solution with the best individual from the neighbor solutions when the same or better fitness is achieved.
    For each individual on the left and right sides **%Block improvement**
      Generate (*k*-*x*) neighbor solutions of this solution;
      Replace this individual when its (*k*-*x*) neighbor solutions or *x* unused best neighbor solutions of the front solution achieve the better fitness.
    Endfor
  Endfor
  Move the leading individual to the end. **%Leader replacement**
  Forward one of the following individuals to the leading position.
Endwhile

3.3 Improved migrating birds optimization algorithm

Initial experiments showed that the original MBO might get trapped into local optima due to fast convergence. Technique to enhance MBO utilized by Gao and Pan (2016) and Zhang et al. (2017) did not help in improving the performance of the MBO in solving the proposed problem. This could be mainly due to two reasons: 1) there are many solutions with the same cycle time and it is impossible to determine which one is better. 2) there are many sequences of tasks in one workstation and there are many different task permutations with the same task assignment. Hence, the task permutation is not enough to distinguish the solutions and the MBO gets trapped into local optima. Therefore, this section presents an improved migrating birds optimization algorithm.

Due to the aforementioned reasons, this research develops several problem-specific improvements. 1) When a better solution is achieved in the leader improvement or block improvement, the incumbent individual is replaced with the new neighbor solution immediately. 2) The incumbent individual is replaced with the new neighbor solution even when the same fitness is achieved. 3) The fitness of the neighbor solution is set to a very large positive value if it shares the same fitness as the incumbent one. The rationality of these improvements is explained as follows. The replacement of the incumbent individual immediately after achieving better solutions guides the search to the most promising area and avoids searching around a poor individual. The incumbent individual is replaced by the new neighbor solution with the same fitness since there are many solutions sharing the same fitness value. This modification makes it possible to explore more solutions and hence enhances the exploration. The fitness of a neighbor solution, which shares the same fitness value as the incumbent one, is set to a very large positive value (in this research it set to 10,000) to avoid the premature convergence of the proposed algorithm. In fact, if this modification is not conducted, all the individuals will have the same fitness value very soon.

Apart from the aforementioned modifications, this research also develops a diversity controlling mechanism when no improvement on the best cycle time is achieved before leader replacement. In general, two individuals might be regarded as two different solutions if they utilize two different task permutations. Nevertheless, this situation is not suitable for RUALBP, where there are many different task sequences within one workstation. Hence, this research utilizes the following procedure, where *Ns*

is the workstation number. The rationality of this diversity controlling mechanism is explained in this section. If solution *i* has the same task assignment as solution *j* (*m*=*Ns*), the solution *i* is abandoned by setting the fitness of solution *i* a very large positive value. If the solution *i* has a large similarity as the remained $n+1-j$ solutions in the swarm ($l \geq (Ns-2) \times (n+1-j)$), the solution i is also abandoned by setting the fitness of solution *i* to a very large positive value.

%Leader improvement

For *j*=1 **to** *k* **do**

Generate a neighbor solution of the leader solution.

Replace the current leader solution with the neighbor solution when the same or better fitness is achieved.

Endfor

Replace the current leader solution with the best individual from the neighbor solutions when the same or better fitness is achieved.

Set a large value to the fitness of the individual whose fitness is the same to that of the incumbent one.

For each individual in the on the left and right sides

%Block improvement

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

Generate a neighbor solution of this solution;

Replace the current solution with the neighbor solution when the same or better fitness is achieved.

Endfor

Replace this individual by the best individual from the (*k*-*x*) neighbor solutions of it and *x* unused best neighbor solutions of the solution in the front when the same or better fitness is achieved.

Set a large value to the fitness of the individual whose fitness is the same to that of the incumbent one.

Endfor

Procedure: Diversity controlling mechanism

% *i* and *j* refer to solutions, and *m* and *l* are utilized to check the similarity between solution *i* and solution *j* .

For *i*:=1 **to** *n*-1 **do**

$l := 0$;

For *j*:=*i*+1 **to** *n* **do**

$m := 0$;

For *k*:= 1 **to** *Ns* **do**

If (the task set on the former *k* workstations in solution *i* is the same to that on the former *k* workstations in solution *j***)**

$m := m + 1;\ l := l + 1;$

Endif

Endfor

If (*m*=*Ns***)**

Set the fitness of solution i a very large positive value;

Break;

Endif

Endfor

If $(l \geq (Ns-2) \times (n+1-j))$

Set the fitness of solution i a very large positive value;

Endif

Endfor

4. An illustrated example

This section provides a detailed example to present the solution representation. This instance with 25 tasks and 4 robots is taken from Gao et al. (2009). The precedence relationships and the operation times are presented in Table 1.

Table 1 Precedence relationships and operation times of tasks

Tasks	Successors	Operation times			
		Robot 1	Robot 2	Robot 3	Robot 4
1	2	85	42	38	81
2	3	47	74	48	43
3	4	87	107	63	53
4	5,8	50	55	57	41
5	6	60	58	39	38
6	7,10	70	47	49	74
7	11,12	44	47	33	37
8	9,11	113	47	84	39
9	10,13	68	35	43	32
10	-	81	131	73	63
11	13	124	70	47	52
12	15	51	90	32	36
13	14	117	89	44	45
14	16,19,20	34	71	42	67
15	17,22	65	79	36	84
16	18	109	115	60	103
17	18,23	59	36	31	35
18	25	82	61	50	54
19	22	39	84	65	53
20	21,25	66	150	52	69
21	22,24	49	36	36	44
22	-	46	63	32	43
23	25	71	70	58	57
24	-	54	48	24	52
25	-	77	62	63	90

Table 2 and Figure 2 present the detailed task assignment and robot selection procedure. Specifically, this table shows the detailed task assignment in the second row and the total operation times by robots in the remaining rows. Among the robots, the robot which has the smallest total operation time is selected as the best fit robot for each workstation. These selected best fit robots are shown in the second last row in the table and in bold in Figure 2. The cycle time of the considered problem is 278 units which is the maximum of the total operation times of workstations.

Table 2 Detailed task assignment and robot allocation

	Workstation 1	Workstation 2	Workstation 3	Workstation 4
Tasks	25, 24, 10, 1, 2, 22	3, 19, 4, 8, 9, 23	18, 5, 6, 16, 7,11	21, 17, 15, 13, 14, 12, 20
Robot 1	390	428	489	441
Robot 2	420	398	398	551
Robot 3	**278**	370	**278**	**273**
Robot 4	372	**275**	358	380
Selected robot	3	4	3	3
Cycle time	278			

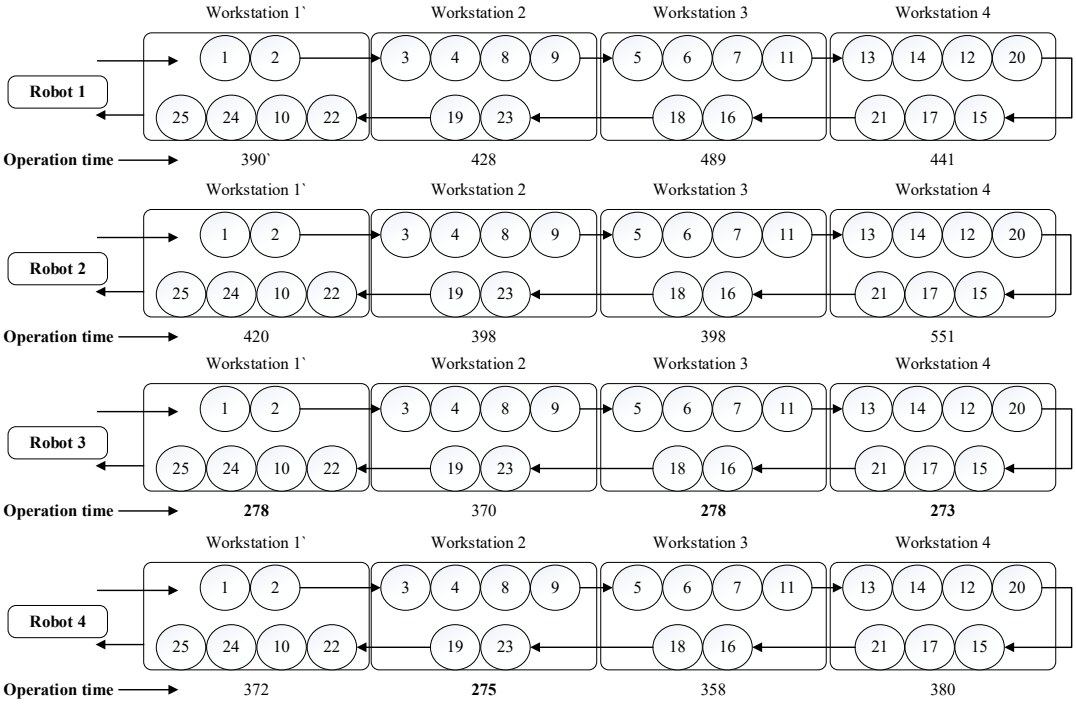

Figure 2. Task assignment and robotic allocation procedure

To highlight the main features of the proposed MBO, Figure 3 illustrates the evolution process of the proposed MBO without diversity controlling mechanism when solving the considered instance, where the initial cycle time is set to 278 units. Specifically, Figure 3 exhibits the minimum cycle time (Min), average cycle time (Avg) and maximum cycle time (Max) achieved by the whole flock during the iteration process (Iteration time). It can be seen clearly that the individuals converge to 278 when iteration time reaches about 96. It can also be observed that all the values of Min, Avg and Max decreases clearly with increased iteration time. This situation is attributed to the benefit mechanism and the greedy acceptance mechanism. Especially, the benefit mechanism makes the value of Max decrease quickly by sharing the neighbor solutions of high-quality individuals with the poor individuals. From Figure 3 it can be concluded that the proposed methodology has faster convergence speed and is capable of avoiding poor individuals by utilizing benefit mechanism.

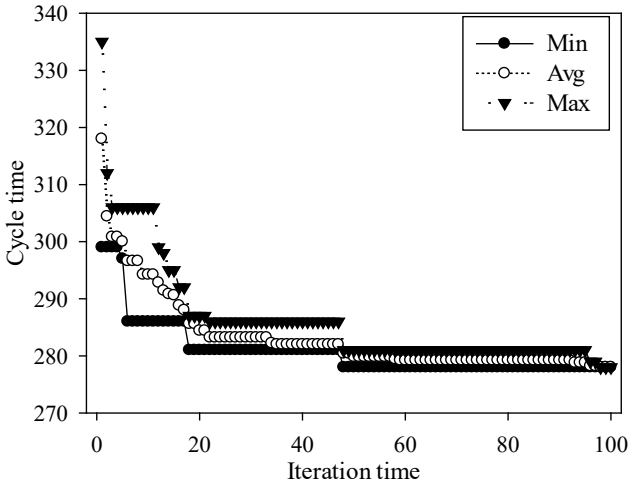

Figure 3. Evolution process of the proposed MBO

5. Experimental tests and results

This section presents the results obtained from the tests conducted using the developed models and a comparative study between MBO and other well-known algorithms.

5.1 Experimental design

To test the performance of the proposed MBO, MBO is compared with seven other published algorithms. Although there exist a number of optimization algorithms, they might not be able to solve the considered problem directly and hence this research mainly re-implements the methods applied to RUALBP and other assembly line balancing problems. The methods considered for comparative study are: simulated annealing algorithm (SA) (Khorasanian et al., 2013), particle swarm optimization (PSO1) (Hamta et al., 2013), particle swarm optimization (PSO2) (Li et al., 2016a), genetic algorithm (GA) (Levitin et al., 2006), teaching-learning-based optimization algorithm (TLBO) (Tang et al., 2015), artificial bee colony algorithm (ABC) (Saif et al., 2014), discrete cuckoo search (DCS) (Li et al., 2017a). The main operators of PSO1, PSO2, GA and ABC are selected based on the reported ones in Li et al. (2017c). Refer Appendix for the detailed pseudocodes of the implemented algorithms. The MBO without diversity controlling mechanism (OMBO) is also included in the comparison.

The benchmark problems summarized in Gao et al. (2009) are selected as the test instances, where eight sets of problems are solve: P25, P35, P53, P70, P89, P111, P148 and P297 (numbers represent the number of tasks) and each of them containing four cases. One small sized problem with 11 tasks reported in Nilakantan and Ponnambalam (2016) is also solved, resulting in a total of 33 cases. Termination criterion for each case is set as an elapsed CPU time which is set to $\mathrm{Nt} \times \mathrm{Nt} \times \tau$ milliseconds, where Nt is the number of tasks and τ is a parameter. In order to observe the performance of the algorithms from short to large computational time, τ is set to 10, 20, 30, 40, 50 and 60; respectively. Based on the parameter calibration method reported in Li et al. (2017c) , the full factorial design is proposed and the multifactor analysis of variance (ANOVA) technique is applied to select the parameter values. Specifically, the largest case with 297 tasks and 29 workstations is selected and is solved for ten times by any combination of the parameter levels, with the termination criterion of $\mathrm{Nt} \times \mathrm{Nt} \times 10$ milliseconds. Once all the experiments are conducted, the relative percentage deviation or RPD is selected as the response variable using$\mathrm{RPD} = 100 \cdot (CT_{Some} - CT_{Best})/CT_{Best}$, where CT_{Some} is the achieved cycle time by a combination and CT_{Best} is the smallest cycle time yield by all combinations. Subsequently,

the ANOVA technique is utilized to analyze these RPD values after checking the fulfillment of the normality, homogeneity of the variances and the independence of the residuals. Detailed ANOVA test is not presented due to space restrictions. But they are available upon request. All the algorithms are codes using C++ programming language and the experiments are conducted on a set of virtual machines in a tower type of server. The server has two Intel Xeon E5-2680 v2 processors (40 processor cores) at 2.8 GHz and 64 GB of RAM memory. All the virtual machines have one virtual processor and 2 GB of RAM. The mathematical models are solved using the Cplex solver in General Algebraic Modeling System 23.0.

5.2 Model evaluation

This section evaluates the performance of the developed models in Table 3, where Model-N is the developed model in Nilakantan and Ponnambalam (2016) and Model 1 and Model 2 refers to the two models developed in this paper. The models could only solve the small-size instances as they cannot achieve satisfying results for large-sized instances within acceptable running time. All the model terminates when the running time reaches 3600 second (s), and the achieved optimal cycle times or upper bounds (lower bounds) are reported. The results obtained by MBO are also reported and the reported ones are the best solution obtained in 10 runs with a termination criterion of $\mathrm{Nt} \times \mathrm{Nt} \times 10$ milliseconds.

Table 3 Results obtained from testing of proposed models and MBO algorithm

Cases	Ns	Model-N		Model 1		Model 2		MBO	
		Results	CPU(s)	Results	CPU(s)	Results	CPU(s)	Results	CPU(s)
P11	4	125	1.80*	115	1.20	115	2.01	115	1.21
P25	3	489	1.78*	468	3.88	468	4.90	468	6.25
P25	4	296	15.58*	278	208.00	278	138.18	278	6.25
P25	6	185	18.66*	175(163.25)	3600.00	179(162.45)	1495.50**	172	6.25
P25	9	104	3600.00	105(91)	1317.55**	102(91)	1566.94**	99	6.25
P35	4	341(340.25)	3600.00	341(339.59)	3600.00	341	256.25	341	12.25
P35	5	313	208.47*	302	3363.01	302	2265.18	302	12.25
P35	7	192(189.75)	3600.00	193(182)	1322.44**	193(182)	2303.19**	188	12.25
P35	12	98(89.94)	3600.00	98(86)	1927.40**	102(86)	3169.17**	89	12.25

Note: * refers to solution obtained due to termination on non-linear programming worsening; ** refers to the solution obtained after the model terminated due to out of memory.

From Table 3, it is clear that Model 1 and Model 2 achieves 4 and 5 optimal solutions; respectively. Model-N, on the contrary, cannot achieve the optimal solution as it terminates on non-linear programming (NLP) worsening. This is mainly due to Model-N being a non-linear model whereas the two developed models are linear model. Additionally, the proposed Model 1 and Model 2 outperforms Model-N for most cases, which further demonstrate the superiority of the proposed models. With respect to the proposed MBO, it could achieve solutions similar to optimal solution for all the cases with less running time, especially for P35. This finding reveals the reasons of utilizing the metaheuristics and suggest the superiority of the metaheuristics in solving large-sized instances.

5.3 Algorithm comparative study

This section provides the results obtained by the implemented algorithm and conducts a comparative study among the considered algorithms. All implemented methods are solved using the benchmark

problems for ten times iteratively. After conducting the experiments, the RPD is applied to transfer the obtained results. Table 4 exhibits the average RPD values for ten repeated runs for each problem under three termination criteria ($\tau = 20, 40, 60$). In the table, each cell reports the average RPD value of several cases in ten repeated runs. For example, each cell for P297 shows the average RPD of four cases: P297 with 19, 29, 38 and 50 workstations. Detailed results under other termination criteria are available upon request.

From Table 4, it can be established that MBO and OMBO are the two best performers when $\tau = 20$ with the overall RPD values of 1.04 and 1.21; respectively. ABC and DCS are the third and fourth best performers with the overall RPD values of 1.57 and 2.33. Based on the increasing order of the overall RPD values, MBO ranks the first, OMBO, ABC, DCS and SA rank the second, third, fourth and fifth and GA, PSO2, TLBO, and PSO1 rank the sixth, seventh, eighth and ninth. It can be also seen that OMBO and MBO outperform the other methods with smaller RPD values in solving large-sized instances such as P89, P111, P148 and P297. For the other two termination criteria, MBO obtains the smallest overall RPD values of 0.78 and 0.64, and OMBO obtain the second smallest overall RPD values of 0.95 and 0.79. MBO and OMBO also show clear superiority over other algorithms with smaller RPD values in solving large-sized instances, P89, P111, P148 and P297. When observing the performances of one algorithm under three terminate criterion, the overall RPD value reduces while increasing the running time for most algorithms, whereas PSO1 shows no clear improvement due to being trapped into local optima. However, the proposed MBO shows clear improvement although the obtain results are near to the optimal solutions, thereby demonstrating the strong exploration and exploitation capacity of the MBO method. All these computational results validate that the proposed MBO is quite effective for solving RUALBP, especially for large-sized problems, and also demonstrate the effectiveness of the diversity controlling mechanism.

Table 4 Average RPD values by implemented algorithms

Problem	Ns	Average relative percentage deviation									CPU(s)
		PSO1	TLBO	PSO2	GA	SA	DCS	ABC	OMBO	MBO	
$\tau = 20$											
P11	4	0.09	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	2.4
P25	3, 4, 6, 9	1.71	0.03	0.10	0.31	1.25	0.19	0.00	0.24	0.00	12.5
P35	4, 5, 7 ,12	4.22	1.68	1.73	1.67	2.08	1.09	0.43	0.63	0.14	24.5
P53	5, 7, 10, 14	5.33	2.38	2.97	2.08	1.74	1.52	0.80	1.09	0.62	56.2
P70	7, 10, 14, 19	9.73	5.74	4.93	3.78	3.05	2.91	1.65	1.33	1.25	98
P89	8, 12, 16, 21	8.21	4.84	4.68	2.94	3.68	2.55	1.54	1.35	1.16	158.4
P111	9, 13, 17, 22	11.51	8.01	6.62	4.18	2.48	3.21	2.33	1.65	1.79	246.4
P148	10, 14, 21, 29	12.49	11.29	7.99	4.83	2.62	3.81	2.93	1.78	1.80	438.1
P297	19, 29, 38, 50	11.72	9.37	7.93	4.60	2.55	3.92	3.25	1.89	1.79	1764.2
Overall RPD		7.87	5.25	4.48	2.95	2.36	2.33	1.57	1.21	**1.04**	-
$\tau = 40$											
P11	4	0.09	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	4.8
P25	3, 4, 6, 9	1.71	0.03	0.03	0.25	1.25	0.08	0.00	0.24	0.00	25
P35	4, 5, 7 ,12	4.22	1.56	1.46	1.45	2.08	0.84	0.40	0.57	0.10	49
P53	5, 7, 10, 14	5.33	2.11	2.63	1.85	1.72	1.29	0.74	1.09	0.60	112.4
P70	7, 10, 14, 19	9.73	5.42	4.59	3.31	2.91	2.60	1.48	1.07	1.16	196

P89	8, 12, 16, 21	8.21	4.57	4.46	2.65	3.60	2.34	1.31	1.26	0.97	316.8
P111	9, 13, 17, 22	11.51	7.59	6.30	3.83	2.34	2.78	1.74	1.23	1.29	492.8
P148	10, 14, 21, 29	12.49	10.42	7.55	4.37	2.17	3.31	2.07	1.28	1.23	876.2
P297	19, 29, 38, 50	11.71	8.17	7.62	3.78	1.73	3.13	2.23	1.07	1.08	3528.4
Overall RPD		7.87	4.83	4.20	2.60	2.16	1.98	1.21	0.95	**0.78**	-
$\tau = 60$											
P11	4	0.09	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	7.3
P25	3, 4, 6, 9	1.71	0.03	0.00	0.21	1.25	0.08	0.00	0.24	0.00	37.5
P35	4, 5, 7 ,12	4.22	1.40	1.42	1.38	2.08	0.79	0.32	0.57	0.09	73.5
P53	5, 7, 10, 14	5.33	2.06	2.22	1.50	1.72	1.09	0.74	1.05	0.51	168.5
P70	7, 10, 14, 19	9.73	5.27	4.27	3.10	2.91	2.49	1.40	1.00	1.09	294.0
P89	8, 12, 16, 21	8.21	4.52	4.22	2.52	3.58	2.15	1.22	1.15	0.91	475.3
P111	9, 13, 17, 22	11.51	7.14	5.98	3.54	2.23	2.57	1.48	1.02	1.07	739.3
P148	10, 14, 21, 29	12.49	10.13	7.33	4.07	1.93	3.07	1.66	0.94	0.95	1314.2
P297	19, 29, 38, 50	11.69	8.09	7.50	3.38	1.24	2.76	1.74	0.55	0.64	5292.5
Overall RPD		7.87	4.68	3.99	2.39	2.05	1.82	1.04	0.79	**0.64**	-

*Best overall RPD in bold

In order to have a better observation of the performance of algorithms under different termination criterion and ascertain the observed difference among the algorithms is statistically significant, this research conducts the statistical analysis, namely the multifactor ANOVA test. The average RPD value of one algorithm in one run is selected as the response variable following Li et al. (2017c) as the algorithms have diverse performance on different instances. Subsequently, the multifactor ANOVA test is also carried out with algorithm type and computational time as two factors after checking the fulfillment of the normality, homogeneity of the variances and the independence of the residuals. Detailed AVOVA table is omitted for space reasons, but it is sufficient to say that there are statistically significant differences between these algorithms, termination criteria and the interaction of the two factors. The means plot of the interactions between algorithms and termination criteria is illustrated in Figure 4, where only 5 best algorithms and three termination criteria ($\tau = 20, 40, 60$) are plotted for a better vision.

From Figure 4, it is clear that MBO is the best performer for all the three termination criteria, and OMBO is the second-best performer. It is to be noted that the overlapping interval means there is no significant statistical difference, and hence it is sufficient to state that MBO is statistically better than ABC, DCS and SA as there are no overlapping intervals between MBO and ABC, DCS or SA. OMBO is also statistically better than ABC, DCS and SA as there are no overlapping intervals between OMBO and ABC, DCS or SA. In summary, the statistical analysis demonstrates that the proposed MBO and OMBO outperforms other algorithms by a significant and larger margin under all the three termination criteria. From the statistical analysis and the results in Table 4, it further demonstrates the superiority of the proposed MBO methods and the effectiveness of the proposed diversity controlling mechanism.

The superiority of the proposed MBO should be attributed to problem-specific improvement such as modified consecutive assignment procedure, cycle time iterative mechanism, new population update mechanism and diversity controlling mechanism. The modified consecutive assignment procedure allocates more tasks to the former workstation, and hence reduces the total operation time in the last workstation or the cycle time. The iterative mechanism ensures that all the individuals are evaluated using the same initial cycle time and helps to preserve the tiny improvements on the individuals. These two improvements greatly improve the performance of the algorithms and hence help the algorithms to

update many upper bounds reported in the literature. In addition, improved MBO algorithm proposed in this paper utilizes new population update mechanism and diversity controlling mechanism to avoid the algorithm getting trapped into local optima and escape from being trapped into local optima. In summary, these improvements are in favor of the MBO having strong local search capacity, maintaining diversity of the population and having a proper balance between exploitation and exploration.

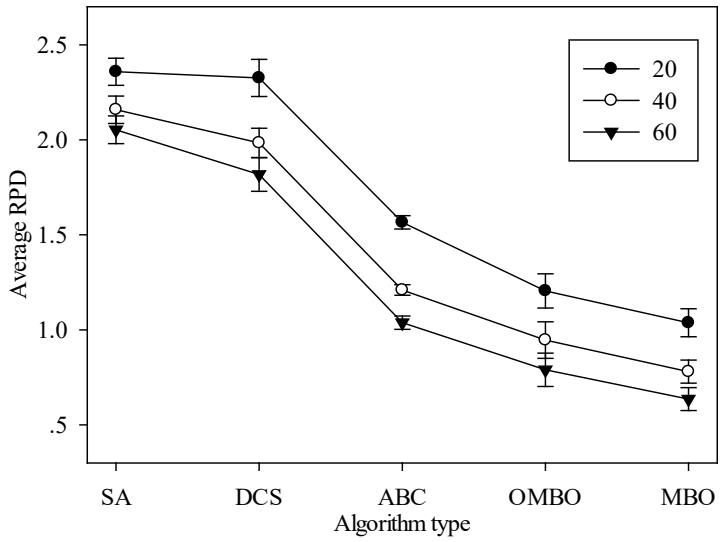

Figure 4 Means plot and 95% Tukey HSD confidence intervals for the interactions between algorithm types and termination criteria

As the proposed algorithm is capable to update the current upper bounds, Table 5 presents the comparison between the reported best results and the results obtained using the proposed MBO, where the result obtained by MBO when τ=10, 20 are illustrated. First and second column presents the problem size and number of workstations followed by the best cycle times and CPU times obtained using particle swarm optimization (PSO-N) as reported in Nilakantan and Ponnambalam (2016). It should be noted that PSO-N is the only method available which addresses the considered RUALBP. Table 5 reports the best cycle times (best), the percentage improvement rate (I.R.), the average cycle time (Avg.) for ten runs. The percentage improvement rate is calculated using $100 \times (CT_{PSO-N} - CT_{MBO})/CT_{PSO-N}$, where CT_{PSO-N} and CT_{MBO} are the best cycle time obtained using PSO-N and MBO. From Table 5, it can be observed that MBO updates the upper bounds for all the tested cases, especially for large-sized datasets. The average improvement rates by MBO are 14.80% and 15.06% when τ=10 and 20. It is also observed that MBO is able to obtain the better results in a shorter computation time. This comparative study further demonstrates the efficiency and effectiveness of the considered MBO for solving RUALBP.

Table 5 Comparison between the published best cycle times and the results by MBO

Problem	Ns	PSO-N*		MBO									
				τ = 10					τ = 20				
		Results	CPU(s)	Best	I.R.	Avg.	s.d.	CPU(s)	Best	I.R.	Avg.	s.d.	CPU(s)
P25	3	500	8.0	468	6.40	468.3	0.48	1.2	468	6.40	468.3	0.48	2.4
	4	318	9.2	278	12.58	278	0.00	6.3	278	12.58	278	0.00	12.5
	6	188	10.5	172	8.51	173.2	1.55	6.3	172	8.51	173.2	1.55	12.5

	9	114	13.5	99	13.16	99.2	0.63	6.3	99	13.16	99.2	0.63	12.5
P35	4	355	16.8	341	3.94	341	0.00	12.3	341	3.94	341	0.00	12.5
	5	332	19.5	302	9.04	304.1	2.02	12.3	302	9.04	304.1	2.02	24.5
	7	221	27.5	188	14.93	189.9	1.20	12.3	188	14.93	189.9	1.20	24.5
	12	103	31.5	89	13.59	89.7	0.95	12.3	89	13.59	89.7	0.95	24.5
P53	5	459	32.5	425	7.41	426.5	2.07	28.1	425	7.41	426.5	2.07	24.5
	7	286	34.8	256	10.49	257.7	2.41	28.1	256	10.49	257.7	2.41	56.2
	10	220	35.6	193	12.27	195.3	1.42	28.1	193	12.27	195.3	1.42	56.2
	14	148	41.2	125	15.54	126.3	0.95	28.1	125	15.54	126	0.94	56.2
P70	7	447	60.1	372	16.78	376.5	3.14	49.0	372	16.78	375.9	2.73	56.2
	10	272	66.8	222	18.38	223.4	1.17	49.0	222	18.38	223.1	0.99	98.0
	14	211	72.4	166	21.33	166.5	0.53	49.0	165	21.80	166	0.47	98.0
	19	144	82.2	117	18.75	117.9	0.57	49.0	116	19.44	117.1	0.74	98.0
P89	8	496	84.5	407	17.94	410.1	2.88	79.2	407	17.94	410.1	2.88	98.0
	12	326	87.1	275	15.64	277.6	2.37	79.2	275	15.64	277.5	2.27	158.4
	16	224	91.2	193	13.84	193.9	0.88	79.2	192	14.29	193.2	0.79	158.4
	21	174	95.3	147	15.52	147.6	0.52	79.2	146	16.09	147.1	0.88	158.4
P111	9	545	234.2	446	18.17	458.3	5.40	123.2	446	18.17	456.8	4.83	158.4
	13	320	253.7	265	17.19	267.9	2.08	123.2	264	17.50	266.3	1.95	246.4
	17	256	298.5	208	18.75	209.1	0.74	123.2	207	19.14	208.3	0.67	246.4
	22	186	348.9	151	18.82	151.9	0.57	123.2	150	19.35	151.1	0.74	246.4
P148	10	629	445.8	518	17.65	524.1	4.72	219.0	517	17.81	522.2	4.57	246.4
	14	421	519.2	339	19.48	344.1	3.14	219.0	337	19.95	341.6	2.72	438.1
	21	283	595.1	225	20.49	225.7	0.67	219.0	223	21.20	223.7	0.67	438.1
	29	187	655.3	157	16.04	157.1	0.32	219.0	155	17.11	155.6	0.52	438.1
P297	19	597	1573.2	502	15.91	506.1	3.07	882.1	500	16.25	502.4	2.32	438.1
	29	394	1693.8	334	15.23	334.9	0.99	882.1	330	16.24	332.1	1.37	1764.2
	38	293	1752.9	250	14.68	251.5	0.71	882.1	248	15.36	249.3	0.82	1764.2
	50	224	1802.3	190	15.18	191.2	0.63	882.1	189	15.63	189.6	0.70	1764.2

* Running using Intel core i5 processor (2.3 GHz).

6. Conclusion and future research

This work studied the robotic U-shaped assembly line balancing problems (RUALBP), where robots are utilized to assemble the products for achieving higher flexibility and reduced cost. Two new mixed-integer linear programming models are first developed with the cycle time minimization criterion. Due to the NP-hard nature of this considered problem, this work utilizes and improves the migrating birds optimization algorithm (MBO) for solving the large-sized problems for the first time. Meanwhile, several problem-specific improvements are also proposed to enhance the MBO, including modified consecutive assignment procedure for robot selection, iterative mechanism for cycle time update, new population update mechanism and diversity controlling mechanism. These improvements are in favor of the MBO maintaining diversity of the population and having a proper balance between exploitation and exploration.

Model comparison shows that the two new linear models outperform the published non-linear model and could achieve the optimal solutions for small-sized instances. A comprehensive computational and comparative study among the implemented algorithms on a set of 33 benchmark problems is also carried

out to test the performance of the proposed MBO. In the comparative study, seven other recent and effective algorithms reported in the literature are re-implemented and compared under six termination criteria from short to large computational times. The computational study, along with strong statistical analysis, demonstrates that the proposed improvements enhance the performance of the MBO algorithm by a significant margin and the proposed MBO outperforms all the compared algorithms statistically. Additionally, the proposed MBO is capable to update the upper bounds for 32 cases with less computational time in a comparison between reported best cycle times and the results by MBO.

Due to the superiority of the new proposed algorithm, it is suggested to utilize this new algorithm to solve different or more complex assembly line balancing problems, such as mixed-model robotic assembly line (Aghajani et al., 2014; Li et al., 2017b), worker assignment (Zacharia and Nearchou, 2016) and parallel workstations (Akpınar and Mirac Bayhan, 2011). Since the real industrial contexts are diverse and more complex, another interesting avenue is related to the robotic assembly line itself by involving more realistic features. For instance, in some occasions cooperative robots support the human worker, where workers and robots work in the same workstations simultaneously. In this regard, the research related to collaboration between humans and robots is beneficial and interesting. In addition, in some assembly lines only a fraction of workstations are assigned with robots, whereas the workers operate the tasks on other workstations.

Acknowledgement

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

Conflict of Interest

The authors declare that they have no conflict of interest.

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.

Akpınar, S., Mirac Bayhan, G., 2011. A hybrid genetic algorithm for mixed model assembly line balancing problem with parallel workstations and zoning constraints. Engineering Applications of Artificial Intelligence 24, 449-457.

Avikal, S., Jain, R., Mishra, P.K., Yadav, H.C., 2013. A heuristic approach for U-shaped assembly line balancing to improve labor productivity. Computers & Industrial Engineering 64, 895-901.

Bagher, M., Zandieh, M., Farsijani, H., 2011. Balancing of stochastic U-type assembly lines: an imperialist competitive algorithm. Int J Adv Manuf Tech 54, 271-285.

Baykasoglu, A., Dereli, T., 2009. Simple and U-Type Assembly Line Balancing by Using an Ant Colony Based Algorithm. Math Comput Appl 14, 1-12.

Chiang, W.-C., Urban, T.L., 2006. The stochastic U-line balancing problem: A heuristic procedure. European Journal of Operational Research 175, 1767-1781.

Çil, Z.A., Mete, S., Ağpak, K., 2017a. Analysis of the type II robotic mixed-model assembly line balancing problem. Engineering Optimization 49, 990-1009.

Çil, Z.A., Mete, S., Özceylan, E., Ağpak, K., 2017b. A beam search approach for solving type II robotic parallel assembly line balancing problem. Applied Soft Computing 61, 129-138.

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.

Duman, E., Uysal, M., Alkaya, A.F., 2012. Migrating Birds Optimization: A new metaheuristic approach and its performance on quadratic assignment problem. Information Sciences 217, 65-77.

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.

Fattahi, A., Turkay, M., 2015. On the MILP model for the U-shaped assembly line balancing problems. European Journal of Operational Research 242, 343-346.

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.

Gao, L., Pan, Q.-K., 2016. A shuffled multi-swarm micro-migrating birds optimizer for a multi-resource-constrained flexible job shop scheduling problem. Information Sciences 372, 655-676.

Gokcen, H., Agpak, K., 2006. A goal programming approach to simple U-line balancing problem. European Journal of Operational Research 171, 577-585.

Hamta, N., Fatemi Ghomi, S.M.T., Jolai, F., Akbarpour Shirazi, M., 2013. A hybrid PSO algorithm for a multi-objective assembly line balancing problem with flexible operation times, sequence-dependent setup times and learning effect. International Journal of Production Economics 141, 99-111.

Hwang, R.K., Katayama, H., Gen, M., 2008. U-shaped assembly line balancing problem with genetic algorithm. International Journal of Production Research 46, 4637-4649.

Khorasanian, D., Hejazi, S.R., Moslehi, G., 2013. Two-sided assembly line balancing considering the relationships between tasks. Computers & Industrial Engineering 66, 1096-1105.

Kim, Y.K., Kim, J.Y., Kim, Y., 2006. An endosymbiotic evolutionary algorithm for the integration of balancing and sequencing in mixed-model U-lines. European Journal of Operational Research 168, 838-852.

Kim, Y.K., Kim, S.J., Kim, J.Y., 2000. Balancing and sequencing mixed-model U-lines with a co-evolutionary algorithm. Production Planning & Control 11, 754-764.

Kucukkoc, I., Zhang, D.Z., 2015. Balancing of parallel U-shaped assembly lines. Computers & Operations Research 64, 233-244.

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., 2017b. Mathematical model and metaheuristics for simultaneous balancing and sequencing of a robotic mixed-model assembly line. Engineering Optimization 50, 877-893.

Li, Z., Kucukkoc, I., Nilakantan, J.M., 2017c. Comprehensive review and evaluation of heuristics and meta-heuristics for two-sided assembly line balancing problem. Computers & Operations Research 84, 146-161.

Li, Z., Nilakantan, J.M., 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.

Miltenburg, G.J., Wijngaard, J., 1994. The U-line Line Balancing Problem. Management Science 40, 1378-1388.

Miltenburg, J., 1998. Balancing U-lines in a multiple U-line facility. European Journal of Operational Research 109, 1-23.

Nakade, K., Ohno, K., 1999. An optimal worker allocation problem for a U-shaped production line.

International Journal of Production Economics 60–61, 353-358.
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., 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 Comput Appl 26, 1379-1393.
Ogan, D., Azizoglu, M., 2015. A branch and bound method for the line balancing problem in U-shaped assembly lines with equipment requirements. Journal of Manufacturing Systems 36, 46-54.
Rabbani, M., Kazemi, S.M., Manavizadeh, N., 2012. Mixed model U-line balancing type-1 problem: A new approach. Journal of Manufacturing Systems 31, 131-138.
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.
Sabuncuoglu, I., Erel, E., Alp, A., 2009. Ant colony optimization for the single model U-type assembly line balancing problem. International Journal of Production Economics 120, 287-300.
Saif, U., Guan, Z., Liu, W., Wang, B., Zhang, C., 2014. Multi-objective artificial bee colony algorithm for simultaneous sequencing and balancing of mixed model assembly line. The International Journal of Advanced Manufacturing Technology 75, 1809-1827.
Scholl, A., Klein, R., 1999. ULINO: Optimally balancing U-shaped JIT assembly lines. International Journal of Production Research 37, 721-736.
Tang, Q.H., Li, Z.X., Zhang, L.P., Floudas, C.A., Cao, X.J., 2015. Effective hybrid teaching-learning-based optimization algorithm for balancing two-sided assembly lines with multiple constraints. Chin J Mech Eng-En 28, 1067-1079.
Ulker, E., Tongur, V., 2017. Migrating birds optimization (MBO) algorithm to solve knapsack problem. Procedia Computer Science 111, 71-76.
Urban, T.L., 1998. Note. Optimal Balancing of U-Shaped Assembly Lines. Management Science 44, 738-741.
Urban, T.L., Chiang, W.-C., 2006. An optimal piecewise-linear program for the U-line balancing problem with stochastic task times. European Journal of Operational Research 168, 771-782.
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.
Zacharia, P.T., Nearchou, A.C., 2016. A population-based algorithm for the bi-objective assembly line worker assignment and balancing problem. Engineering Applications of Artificial Intelligence 49, 1-9.
Zhang, B., Pan, Q.-k., Gao, L., Zhang, X.-l., Sang, H.-y., Li, J.-q., 2017. An effective modified migrating birds optimization for hybrid flowshop scheduling problem with lot streaming. Applied Soft Computing 52, 14-27.

Appendix. Pseudocodes of the tested algorithms and utilized operators

The implemented and compared algorithms in this paper are: simulated annealing algorithm (SA), particle swarm optimization (PSO1), particle swarm optimization (PSO2), genetic algorithm (GA), teaching-learning-based optimization algorithm (TLBO), artificial bee colony algorithm (ABC), discrete cuckoo search (DCS). The psuedocdoes of the implemented algorithms are listed in sequence as follows.

A) SA algorithm

The main procedure of SA algorithm is presented in Algorithm 1.

Algorithm 1: Procedure of SA algorithm

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

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

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

Obtain a neighbor solution S' ;

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

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

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

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

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

Endfor

$T = T \times \alpha$

Endwhile

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

B) PSO1 algorithm

PSO1 utilizes the random-key method to tackle the considered discrete problem. Supposed that there are nine tasks and the original $x_{i,0}$ is [0.42, 0.68, 0.35, 0.01, 0.70, 0.25, 0.79, 0.59, 0.63], the floating-point vector is transferred into the task permutation in the following method. Here, the tasks with lowest value has the highest priority and thus task 4 with the lowest value 0.01 is allocated to the first position in the task permutation; the task 6 with the second lowest value 0.25 is allocated to the second position in the task permutation. Subsequently, the floating-point vector is transferred into the task permutation [4, 6, 3, 1, 8, 9, 2, 5, 7]. In short, by employing the random-key method, the original evolution techniques in PSO can be applied directly, where the task permutation is applied in decoding to obtain the fitness values.

Algorithm 2: Procedure of PSO1 algorithm

For i:=1 to PS **do** **%Initialization**

$x_{i,0} = x_{min} + random(x_{max} - x_{min})$;

$v_{i,0} = v_{min} + random(v_{max} - v_{min})$;

Obtain the fitness value of solution $x_{i,0}$;

Endfor

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

Select the global best particle $x_{i,k}^{gbest}$ and local best particle $x_{i,k}^{lbest}$;

For i:=1 to PS **do** **%Population evolution**

$v_{i,k+1} = c_1 \cdot r_1 \cdot \left(x_{i,k}^{gbest} - x_{i,k}\right) + c_2 \cdot r_2 \cdot \left(x_{i,k}^{lbest} - x_{i,k}\right) + w_k \cdot v_{i,k}$;

$x_{i,k+1} = x_{i,k} + v_{i,k+1}$;

Obtain the fitness value of solution $x_{i,k+1}$;

Endfor

Endwhile

%Parameter *PS* is population size; k denotes the iteration number of the algorithm.

C) PSO2 algorithm

PSO2 is a discrete PSO utilizing neighbor operator and crossover operator to update the population. The main procedure of PSO2 is presented in Algorithm 3.

Algorithm 3: Procedure of PSO2 algorithm
Initialize the *PS* individuals;
While (Termination criterion is not satisfied**)**
Select the global best particle x_{gbest} and local best particle x_{lbest};
Utilize the x_{gbest} as the first individual in the new population;
For *i*:=2 to *PS* **do**
If $(rand(0,1) < r)$
Obtain a new individual *ii* utilizing the crossover operator with individual *i* and x_{gbest} as parents;
Else
Obtain a new individual *ii* utilizing the crossover operator with individual *i* and x_{lbest} as parents;
Endif
Obtain a new individual *iii* utilizing neighbor operator on the individual *ii*;
Replace individual *i* with the new individual *iii*;
Endfor
Endwhile
%Parameter *PS* is population size; r is a parameter within (0,1).

D) GA algorithm

The main procedure of GA algorithm is presented in Algorithm 4.

Algorithm 4: Procedure of GA algorithm
Initialize the *PS* individuals;
While (Termination criterion is not satisfied**)**
%Selection and crossover
While (the number of offspring is not larger than $P_C \cdot PS$**)**
Select Parent 1 from the parent population utilizing tournament selection;
Select Parent 2 from the parent population utilizing tournament selection;
Obtain Child 1 and Child 2 utilizing two-point crossover operator;
Endwhile
% Mutation
While (the number of offspring is not larger than $PS - 1$)
Obtain one parent individual from the parent population utilizing tournament selection;
Obtain a new child individual utilizing neighbor operator;
Endwhile
% Elitist strategy

Select the best individual from the parent population;
Clone this best individual to the offspring as the last child individual;
Endwhile
%Parameter *PS* is population size; P_C is the crossover probability (the mutation probability = 1- crossover probability);

E) TLBO algorithm

TLBO utilizes the random-key method to tackle the considered discrete problem. Supposed that there are nine tasks, nine floating-point numbers between 0 and 1 are generated for one individual. For a vector ψ = (0.42, 0.68, 0.35, 0.01, 0.70, 0.25, 0.79, 0.59, 0.63), the floating-point vector is transferred into the task permutation in the following method. Here, the tasks with lowest value has the highest priority and thus task 4 with the lowest value 0.01 is allocated to the first position in the task permutation; the task 6 with the second lowest value 0.25 is allocated to the second position in the task permutation. Subsequently, the floating-point vector is transferred into the task permutation [4, 6, 3, 1, 8, 9, 2, 5, 7]. In short, by employing the random-key method, the original evolution techniques in TLBO can be applied directly, where the task permutation is applied in decoding to obtain the fitness values.

Algorithm 5: Procedure of TLBO algorithm
Initialize the *PS* individuals;
While (Termination criterion is not satisfied**)**
For *i*:=1 to *PS* **do** **% Teacher phase**
$T_F = round\big(1 + rand(0,1)\big)$;
% $rand(0,1)$ is random number within (0,1).
$X_{mean} \leftarrow$ Calculate mean value of the population;
$X_{tearcher} \leftarrow$ Best solution;
$X_i^{new} = X_i + r.(X_{tearcher} - T_F \cdot X_{mean})$;
Replace X_i with X_i^{new} when $\text{Fit}(X_i^{new}) \leq \text{Fit}(X_i)$;
Endfor
For *i*:=1 to *PS* **do** **% Learner phase**
Select a different individual *ii* randomly ($i \neq ii$);
If ($\text{Fit}(X_i) \geq \text{Fit}(X_{ii})$**)** $X_i^{new} = X_i + r.(X_{ii} - X_i)$;
Else $X_i^{new} = X_i + r.(X_i - X_{ii})$;
Endif
Replace X_i with X_i^{new} when $\text{Fit}(X_i^{new}) \leq \text{Fit}(X_i)$;
Endfor
EndWhile
%Parameter *PS* is population size; r is a random number within [0,1];
$\text{Fit}(p)$ is the fitness of solution *p*.

F) DCS algorithm

The main procedure of DCS algorithm is presented in Algorithm 5.

Algorithm 6: Procedure of DCS algorithm
Generate initial population with *PS* host nests; **%Initialization**
While (termination criterion is not met) **do**
For *p*=1 to ***PS*** do **%Population update**

Get a cuckoo (say a) randomly using neighbor operators on randomly selected individual;
Select a nest (say b) randomly from current population;
Replace solution b with solution a if $\mathrm{Fit}(a) \leq \mathrm{Fit}(b)$;
Endfor
Ranks the individuals based on the fitness;
Replace a fraction P_a of worse nests with the neighbor solutions of randomly selected solutions from the remained better individuals;
EndWhile
%Parameter *PS* is population size; $\mathrm{Fit}(p)$ is the fitness of solution p.

G) ABC algorithm

The main procedure of ABC algorithm is presented in Algorithm 6.

Algorithm 7: Procedure of ABC algorithm
Initialize the *PS* food sources;
While (Termination criterion is not satisfied**)**
For p:=1 to *PS* **do** **% Employed bee phase**
Generate a neighbor solution p' of the individual p;
Replace p with p' when $\mathrm{Fit}(p') \leq \mathrm{Fit}(p)$;
Endfor
For p:=1 to *PS* **do** **% Onlooker phase**
Obtain the probability value for each individual;
Select one individual S using roulette wheel selection scheme;
Generate a neighbor solution S' of the individual S;
Replace S with S' when $\mathrm{Fit}(S') \leq \mathrm{Fit}(S)$;
Endfor
% Scout phase
Select one duplicated solution or the solution with the worst fitness;
Replace selected individual with the neighbor solution of a randomly selected solution from the remained individuals;
Endwhile
%Parameter *PS* is population size; $\mathrm{Fit}(p)$ is the fitness of solution p.

H) Neighbor operator

The implemented algorithms employ the insert operator and swap operator as the neighbor operator, and the two-point crossover operator to combine two individuals. Specifically, the insert operator or swap operator is randomly selected and applied to obtain one new task permutation. Figure 5 depicts an example of insert operator and swap operator with nine tasks, and Figure 6 depicts an example of utilizing two-point crossover operator.

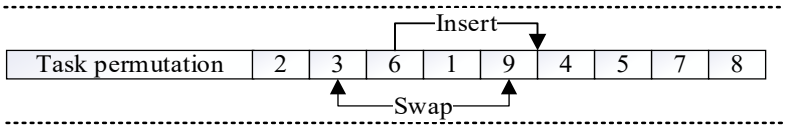

Figure 5 Insert operator and swap operator

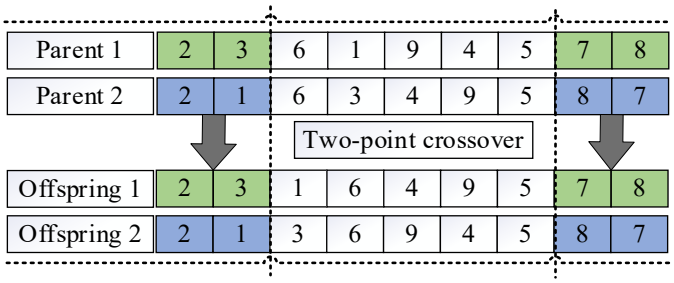

Figure 6 Two-point crossover operator