Complex & Intelligent Systems
https://doi.org/10.1007/s40747-020-00166-z

ORIGINAL ARTICLE

An estimation of distribution algorithm with branch-and-bound based knowledge for robotic assembly line balancing

Bin-qi Sun[1] · Ling Wang[1]

Received: 5 May 2020 / Accepted: 3 June 2020

Abstract
Robotic assembly lines are widely used in manufacturing industries. The robotic assembly line balancing (RALB) problem aims to balance the workloads among different workstations and optimize the assembly line efficiency. This paper addresses a particular type of RALB problem, which minimizes the assembly line cycle time by determining the task and robot assignment in each workstation under precedence constraints. To solve the problem, we present an effective hybrid algorithm fusing the estimation of distribution algorithm and branch-and-bound (B&B) based knowledge. A problem-specific probability model is designed to describe the probabilities of each task being assigned to different workstations. Based on the probability model, an incremental learning method is developed and a sampling mechanism with B&B based knowledge is proposed to generate new feasible solutions. The fuse of B&B based knowledge is able to reduce the search space of EDA while focusing the search on the promising area. To enhance the exploitation ability, a problem-specific local search is developed based on the critical workstation to further improve the quality of elite solutions. The computational complexity of the proposed algorithm is analyzed, and the effectiveness of the B&B based knowledge and the problem-specific local search is demonstrated through numerical experiments. Moreover, the performance of the proposed algorithm is compared with existing algorithms on a set of widely-used benchmark instances. Comparative results demonstrate the effectiveness and efficiency of the proposed algorithm.

Keywords Estimation of distribution algorithm · Sampling mechanism · Branch-and-bound based knowledge · Problem-specific local search · Robotic assembly line balancing

Introduction

During recent years, advanced automation techniques have been applied to assembly systems to improve flexibility and productivity. One of the most successful applications is the robotic assembly line, which uses robots to replace human labors. Compared to manual assembly line, the robotic assembly line can speed up the assembly process and reduce the work stress of workers [1]. Moreover, it provides more favorable conditions for decision-makers to design and manage the assembly line.

Assembly line balancing (ALB) problem is a typical problem in the design and management of assembly lines. It aims to balance the workload among different workstations by appropriately determining the task assignments subject to operational constraints. The ALB problem can be classified into four types according to the objectives, i.e. ALB-1, ALB-2, ALB-E and ALB-F [2, 3]. The objective of ALB-1 is to minimize the number of workstations with a given cycle time, while ALB-2 minimizes the cycle time with a predetermined workstation number. ALB-E attempts to maximize the line efficiency by minimizing the cycle time and workstation number simultaneously. ALB-F is a feasibility problem to figure out whether a feasible solution exists with a given combination of cycle time and workstation number. The most basic version of ALB is known as simple assembly line balancing (SALB) problem [2], which considers straight manual assembly lines with deterministic task time. The robotic assembly line balancing (RALB) problem can be regarded as the extension of SALB. Different

✉ Ling Wang
wangling@tsinghua.edu.cn

Bin-qi Sun
sbq17@mails.tsinghua.edu.cn

1 Department of Automation, Tsinghua University, Beijing 100084, China

Published online: 12 June 2020

from SALB, the decisions of RALB include not only the task assignment but also the robot assignment, which makes the problem more challenging.

So far, some mathematical models and model-based optimization algorithms have been proposed for RALB with different objectives and problem assumptions. In Rubinovitz and Bukchin [4], the RALB was considered as a RALB-1 to minimize the workstation number with a given cycle time. They proposed a frontier-search based branch-and-bound (B&B) algorithm to solve the problem. Although some heuristic rules were incorporated to reduce the search space, the storage space and computation time of the B&B based method are still too large to solve large scale problems. Bukchin and Tzur [5] considered a flexible assembly line designing problem to minimize total equipment costs with a given cycle time. They developed a B&B algorithm to solve small scale problems and proposed a B&B based heuristic to solve large scale problems. However, the performance of the heuristic is sensitive to the algorithm parameter, which is hard to calibrate when solving practical problems with different scales. Tsai and Yao [6] developed a line-balance-based capacity planning heuristic, which determines the type and number of robots required in each workstation to minimize the deviation of the workstation output rate. In the heuristic, an integer programming model is developed to generate an initial solution. Then, special heuristic rules are designed to identify the bottleneck workstation and improve the initial solution by making task adjustments to the bottleneck workstation. However, the proposed heuristic was only tested on ten small scale instances. Kim and Park [7] developed an integer programming model for a special type of RALB and solved it using a cutting plane algorithm. However, the assignment of robots was considered to be fixed for each workstation instead of decision variables in the model.

Due to the NP-hardness of RALB problem [8], it is difficult to obtain satisfied solutions by using the traditional model-based methods for large scale problem in acceptable time. So many researchers attempt to design efficient data-driven metaheuristics to solve large scale problems. Levitin et al. [9] proposed a genetic algorithm (GA) for RALB to minimize the cycle time with given workstation number. Two different decoding procedures were developed for task and robot assignments, including a recursive procedure and a consecutive procedure. The algorithm was tested on randomly generated instances, the results showed that the consecutive procedure is better than the recursive procedure. Gao et al. [10] proposed a hybrid GA for RALB-2. However, the problem assumptions in the two papers are different. In [9], one type of robot can be assigned to different workstations. But in [10], each robot can only be assigned to one workstation. Recently, a particle swarm optimization (PSO) and a hybrid of cuckoo search and PSO (CS-PSO) were proposed by Nilakantan et al. [11]. The problem assumptions in [11] are the same as [9]. The consecutive procedure is adopted to determine the task and robot assignment. The computational results show that the PSO and the CS-PSO are better than the hybrid GA. Following [10], Li et al. [12] developed four mixed-integer linear programming models to find optimal solutions of small scale problems and proposed two simulated annealing (SA) algorithms to solve large scale problems. The experiments showed that the SAs are superior to other algorithms under the same problem assumptions. In addition to the study of one-sided straight assembly line, Li et al. proposed a discrete cuckoo search algorithm [13] and a migrating birds optimization [14] to solve the two-sided and U-type RALB problems, respectively. Moreover, there are also some papers considering the reduction of energy consumption brought by robots in robotic assembly lines [15–17].

From the literature review, it can be seen that the existing RALB algorithms are either model-based approaches or data-driven metaheuristics. Both of them have their own merits and limitations. Model-based approaches can obtain the optimal solution for small scale problems, but it is hard to tackle large scale problems. Data-driven metaheuristics are able to solve large scale problems with satisfactory results, while their random nature makes it difficult to guarantee the robustness. In this paper, we propose a hybrid algorithm fusing these two kinds of approaches together to solve the RALB-2 following the assumptions in [9, 11]. It employs the estimation of distribution algorithm (EDA) [18] as the search framework due to its successful applications in many optimization problems [19–23] and fuses B&B strategy as knowledge into the sampling process of EDA to enhance search efficiency. With the reasonable integration, the proposed EDA with B&B based knowledge (EDA-BB) outperforms other existing metaheuristics in solving the RALB-2 with different scales.

The remaining of the paper is organized as follows. The next section provides the description and formulation of the RALB-2. The following section presents the details and analysis of the proposed EDA-BB. Computational results and comparisons are reported in the next section. Finally, the paper is ended with some conclusions and future work in the last section.

Problem description

An assembly line consists of a conveyer belt and a sequence of workstations as shown in Fig. 1. The components are consecutively launched down the assembly line through the conveyer belt and move from workstation to workstation until they reach the end of the line [2]. When a component passes through a workstation, a certain amount of tasks will be performed by a robot. The processing time of a task is

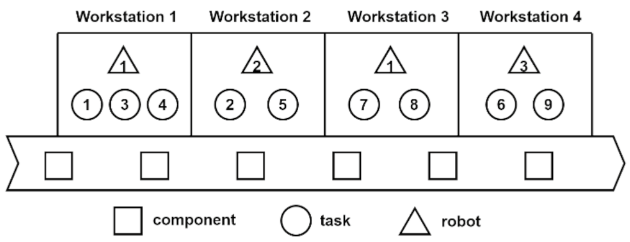


Fig. 1 The layout of the robotic assembly line

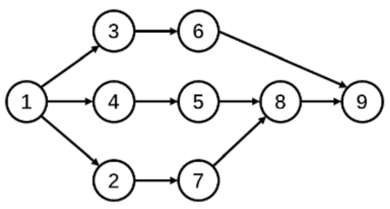


Fig. 2 The illustration of precedence constraints

called the task time, the sum of the task time in a workstation is called the workstation time. The workstation time needs to be smaller than the duration of a component passing through the workstation, which is referred as the cycle time. The tasks need to be executed in a specified order given by the precedence constraints. The precedence constraints can be illustrated as a directed acyclic graph (DAG), where the nodes represent the tasks and the arcs represent the precedence relationships between the tasks. An example with 9 tasks is shown in Fig. 2.

The objective of the RALB-2 is to minimize the cycle time by determining the task and robot assignment of each workstation under the precedence constraints. The basic assumptions of the RALB-2 are given as follows.

- A single type of product is assembled on the assembly line.
- The assembly line is paced, i.e. the cycle time of all workstations equals to the same value.
- The task time is deterministic and depends on the type of operating robot.
- A task can only be operated in one workstation and all tasks should be performed without violating the precedence constraints.
- Only one robot can be assigned to a workstation. Any task can be operated by any robot.
- Different workstations can use the same type of robot. There is no number limit for each type of robot.
- The setup time and tool changing time are included in the task time.

According to the same assumptions as [11], the 0–1 integer programming model is presented with the following notations.

Parameters:

N: number of tasks.
i,j: indexes of tasks, $i,j = 1, 2, \ldots, N$.
M: number of workstations.
k: index of workstations, $k = 1, 2, \ldots, M$.
R: number of robot types.
r: index of robot types, $r = 1, 2, \ldots, R$.
$t_{i,r}$: processing time of task i by robot r.
P_i: the set of predecessors of task i.

Decision variables:

c: cycle time.
$x_{i,k}$: 1, if task i is assigned to station k; 0, otherwise.
$y_{r,k}$: 1, if robot r is assigned to station k; 0, otherwise.

$$\min c = \max_{1 \le k \le M} \sum_{i=1}^{N} \sum_{r=1}^{R} t_{i,r} \cdot x_{i,k} \cdot y_{r,k} \tag{1}$$

$$\text{s.t.} \quad \sum_{k=1}^{M} k \cdot x_{i,k} - \sum_{k=1}^{M} k \cdot x_{j,k} \le 0, \forall i \in P_j, \forall j \tag{2}$$

$$\sum_{k=1}^{M} x_{i,k} = 1, \forall i \tag{3}$$

$$\sum_{r=1}^{R} y_{r,k} = 1, \forall k \tag{4}$$

$$x_{i,k} \in \{0, 1\}, \forall i, k \tag{5}$$

$$y_{r,k} \in \{0, 1\}, \forall r, k \tag{6}$$

The objective function (1) is to minimize the cycle time, which is defined as the maximum workstation time. Constraints (2) ensure the precedence relationships cannot be violated. Constraints (3) and (4) imply that each task must be assigned to one single workstation and each workstation must be assigned a robot, respectively. Constraints (5) and (6) indicate the type of the decision variables.

EDA-BB for RALB-2

Owing to the NP-hard nature of RALB-2, only small scale problems can be solved using mathematical programming in acceptable time [11]. Therefore, it is important and necessary to develop efficient metaheuristics to solve large scale problems. In this section, an estimation of distribution algorithm with B&B based knowledge will be presented for RALB-2.

Outline of the EDA-BB

EDA [18] is a typical population-based metaheuristic with good performances in solving complex optimization problems. The major difference between EDA and other metaheuristics like GA is the means of generating new solutions. EDA builds a probability model to estimate the distribution of elite solutions and generates new solutions by sampling the probability model. The advantage of the probability model is that it can adapt itself towards superior structural elements during the evolution [24]. Hence, the exploration ability of EDA is usually stronger than other metaheuristics. However, owing to the lack of problem-specific design, the exploitation ability of EDA needs to be enhanced. In this paper, the B&B strategy is fused as knowledge into the sampling of the EDA to improve the search efficiency and a problem-specific local search is developed to improve the quality of elite solutions. The flowchart of the proposed EDA-BB is illustrated in Fig. 3.

In the initialization phase, the probability model is initialized uniformly and the upper bound of cycle time is initialized using simple heuristic rules. Then, the probability model is sampled to generate a population of candidate solutions. The B&B based knowledge and the upper bound of cycle time are used in the sampling process to guarantee the quality of generated solutions. After that, the candidate solutions are sorted according to their objective values and some better solutions are selected as elite solutions. Then, the problem-specific local search is performed to further improve the elite solutions, which are used to update the probability model and the upper bound. Finally, the algorithm outputs the best solution when the stopping criteria are met.

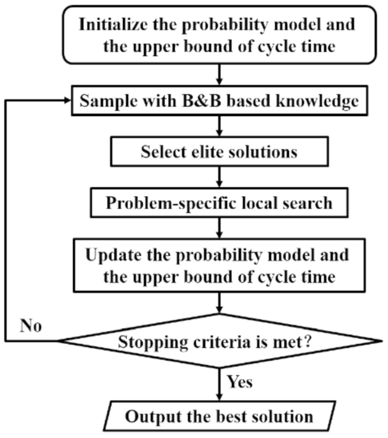

Initialize the probability model and the upper bound of cycle time
Sample with B&B based knowledge
Select elite solutions
Problem-specific local search
Update the probability model and the upper bound of cycle time
Stopping criteria is met?
No
Yes
Output the best solution

Fig. 3 Flowchart of EDA-BB

More details of the proposed EDA-BB will be presented in the following sections.

Probability model

The probability model plays an important role in the design of EDA since it is used to describe the distribution of the solution space and generate new solutions. A good probability model should be able to reflect the characteristics of the problem and also easy to implement.

There are two classes of decision variables in the RALB-2: the assignment of tasks and the assignment of robots. Since the number of robots is not limited, we can first determine the assignment of tasks and then choose the best-fit robot for each workstation. So the probability model can be designed to describe the distribution of tasks assigned to different workstations. We use a matrix $P(g)$ to represent the probability model in generation g as follows:

$$P(g) = \begin{bmatrix} \rho_{11}(g) & \rho_{12}(g) & \dots & \rho_{1N}(g) \\ \rho_{21}(g) & \rho_{22}(g) & \dots & \rho_{2N}(g) \\ \vdots & \vdots & \ddots & \vdots \\ \rho_{M1}(g) & \rho_{M2}(g) & \dots & \rho_{MN}(g) \end{bmatrix},$$

where element $\rho_{ki}(g)$ denotes the probability that task i is assigned to workstation k in the gth generation. Since a task must be assigned to one workstation, it is clear that $\sum_{k=1}^{M} \rho_{ki}(g) = 1, \forall i, g$.

Initialization

The initialization of the EDA-BB includes two parts: the initialization of the probability model and the initialization of the upper bound of cycle time. As there is no prior knowledge of the elite solution distribution, the probability model is initialized uniformly, i.e. $\rho_{ki}(g) = 1/M, \forall k, i, g$.

The upper bound of cycle time is important for the sampling of EDA-BB. A common way to obtain an upper bound of a problem is to calculate the objective value of a feasible solution. Hence, we develop a heuristic to construct good initial solutions and use the best objective of the initial solutions as the initial upper bound.

The heuristic includes into two steps. The first step is to generate task sequences and the second step is to decode the task sequences into feasible solutions. There have been a lot of priority rules proposed in the literature to construct task sequences. The most widely-used priority values are computed based on the task time and the precedence relations [8]. In this paper, we select two rules (maximum task time and maximum immediate follower number) from [8] to generate the task sequences. The second step is to decode the task sequences into feasible solutions. The consecutive procedure proposed in [9] is used due to its superiority to other methods according to [9]. Then, the cycle time of each feasible solution is calculated and the lower one is used to initialize the upper bound.

Sampling mechanism with B&B based knowledge

The population of EDA is generated by sampling the probability model. Due to the fuse of the B&B based knowledge, the sampling process of EDA-BB is divided into M stages, where M is the number of workstations. The procedure of each sampling stage is explained as follows.

In the first stage, PS partial solutions are generated. Each partial solution contains the task and robot assignments of the first workstation. The tasks are assigned by selecting from the *task list* using the Roulette Wheel Method [25] iteratively. The *task list* records the tasks which can be assigned to workstation m without violating the precedence constraints, and the proportions in the Roulette Wheel Method are taken from the probability model. Once a task is selected from the *task list*, the best-fit robot that performs the assigned tasks in the least time is assigned to the workstation and the related workstation time is calculated. The assigning procedure continues until the workstation time reaches the upper bound of the cycle time. Specific procedures are given in Algorithm 1.

Algorithm 1 Procedures of assigning tasks and robot to workstation m

```
1:  Initialize an empty workstation $W_m$; initialize the time list $WT_r \leftarrow 0, r = 1,2,\dots,R$ recording
    the workstation time operated by robot $r$.
2:  while $WT_{r_b} < UB$ do
3:      Select a task $s$ using roulette wheel method: $s \leftarrow RouletteWheelMethod(task\ list, P)$.
4:      for $r = 1$ to $R$ do
5:          $WT_r \leftarrow WT_r + t_{s,r}$.
6:      end for
7:      if $WT_{r_b} < UB$ do
8:          Add task $s$ to $W_m$: $s \rightarrow W_m$.
9:          $r_b \leftarrow \arg\min_r WT_r$.
10:         Update the task list.
11:     end if
12: end while
13: Assign the best-fit robot $r_b$ to workstation m.
```

In each stage $m \in [2, \mathrm{M}-1]$, it first calculates the lower bounds of the PS partial solutions from the previous stage by (7), where A_k denotes the set of tasks assigned in workstation k and U denotes the set of unassigned tasks. Then, the PS partial solutions are sorted according to their lower bounds, and the $\mathrm{PS}/2$ partial solutions with larger lower bounds are deleted from the population. After that, each of the remaining partial solutions is expanded into two new branches by generating the task and robot assignments for workstation m using Algorithm 1. Each branch is a new partial solution with the task and robot assignments of the first m workstations. To guarantee the diversity of the two branches, their task assignments of workstation m are compared with each other. If the task assignments of the two branches are the same, a replace or delete operator will be performed on one of them. Specific procedures of the sampling process in stage $m \in [2, \mathrm{M}-1]$ are given in Algorithm 2.

$$LB = \sum_{k=1}^{m} \min_{1 \le r \le R} \sum_{i \in A_k} t_{i,r} + \sum_{i \in U} \min_{1 \le r \le R} t_{i,r} \tag{7}$$

Algorithm 2 Sampling procedures in stage $m \in [2, \mathrm{M}-1]$

```
Input: $PS$ partial solutions in stage $m-1$: $S_{m-1}(n)$ , $n = 1,2,\ldots,PS, 2 \le m \le M-1$
1:   Calculate the lower bound of each partial solution by (7).
2:   Rank the partial solutions $S_{m-1}$ in ascending order of cycle time.
3:   Delete the last $PS/2$ solutions from $S_{m-1}$.
4:   for $n = 1$ to $PS/2$ do
5:     for $l = 1$ to $2$ do
6:       Generate $W_m$ (the assignment of workstation $m$) using Algorithm 1.
7:       Add $W_m$ to $S_{m-1}(n)$ to generate a new branch in stage $m$: $S_m(2 \cdot n - 2 + l)$.
8:     end for
9:     if the assignment in $S_m(2 \cdot n - 1)$ is the same with $S_m(2 \cdot n)$ do
10:      Find the last task $p$ which is assigned to workstation $m$ in $S_m(2 \cdot n)$.
11:      if task $p$ can be replaced by other unassigned task $q$ do
12:        Replace task $p$ with task $q$ with the least task time; update the task list; assign the best-fit robot.
13:      end if
14:      else do
15:        Delete task $p$ from $S_m(2 \cdot n)$; update the task list; assign the best-fit robot.
16:      end else
17:    end if
18:  end for
Output: $PS$ partial solutions $S_m(n)$ in stage , $1 \le n \le PS, 2 \le m \le M-1$.
```

In the last stage, each of the PS partial solutions from stage $M-1$ is expanded into a complete solution by assigning all the unassigned tasks into the last workstation and allocating the robot with the shortest workstation time.

In this way, PS complete solutions are generated. The whole procedures of the sampling mechanism are given in Algorithm 3.

Algorithm 3 Sampling with B&B based knowledge

```
Input: the probability model $P$, the upper bound of cycle time $UB$
1:    for $n = 1$ to $PS$ do
2:      Generate a partial solution $S_1(n)$ by assigning tasks and best-fit robot
3:      to the first workstation using Algorithm 1.
4:    end for
5:    for $m = 2$ to $M-1$ do
6:      Generate $PS$ partial solutions $S_m$ using Algorithm 2.
7:    end for
8:    for $n = 1$ to $PS$ do
9:      Generate a complete solution $S_M(n)$ by assigning all unassigned tasks
10:     and the best-fit robot to the last workstation.
11:   end for
Output: $NP$ complete solutions $S_M(n), n = 1,2,\ldots,PS$
```

For clarity, an illustration of the whole sampling process is shown in Fig. 4. The grey circles represent the partial solutions which are dismissed in the sampling process and the white ones represent the partial solutions to be expanded in the next stage. From the figure, it can be seen that the sampling process is guided by the B&B based knowledge. The promising partial solutions with better lower bounds are expanded to generate more branches and the inferior ones are dismissed to avoid wasting much computation. In this way, the search space of the sampling is reduced and the search is concentrated on the promising area to generate more high-quality solutions.

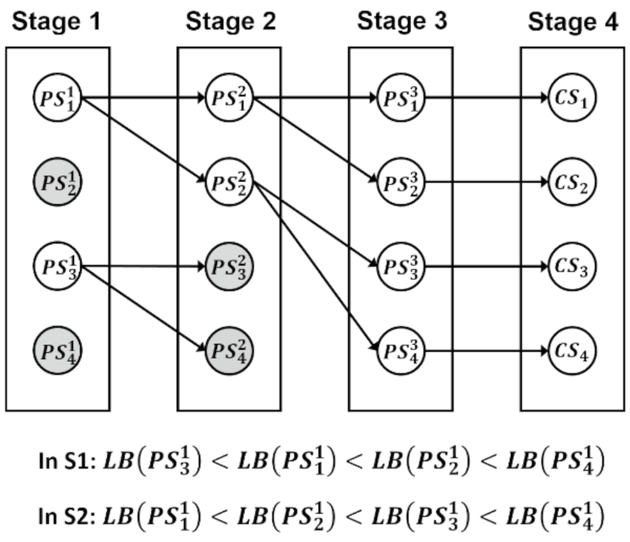


Fig. 4 An illustration of sampling process with B&B based knowledge

Elite selection and local search

After a population is generated by sampling the probability model, we rank the solutions in the population in the ascending order of their cycle time. The first $\lfloor PS \cdot \alpha \rfloor$ solutions are selected as elite solutions, where α is a user-determined parameter. Then, a problem-specific local search is performed on the elite solutions to further reduce their cycle time.

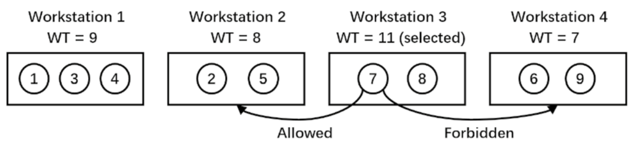
The local search is an iterative process with neighborhood structure. It is clear that the solution can be improved only if the workload of the fullest workstation is reduced and the workloads of other workstations do not exceed the original cycle time. Thus, we select the workstation with the maximum workload as the critical workstation. Then it uses two operators on the tasks assigned to the critical workstation and its adjacent workstations.

The first one is the insert operator, which attempts to move one task from the fullest workstation to its left or right neighbor workstation (if any). Owing to the precedence constraints, one task can be inserted to the left (right) workstation only if it has no predecessors (followers) in the selected workstation.

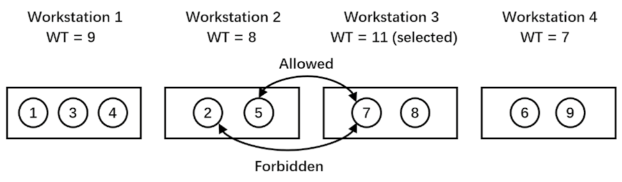
The second one is the swap operator, which attempts to exchange one task in the fullest workstation with another task in its left or right neighbor workstation (if any). Similar to the insert operator, task A in the selected workstation can be exchanged with task B in the left (right) workstation only if the following conditions hold: (1) task A has no predecessors (followers) in the selected workstation; (2) task B has no followers (predecessors) in the left (right) workstation; (3) task A is not the follower (predecessor) of task B.

Figure 5 shows an illustration of the insert and swap operator. In Fig. 5, there are nine tasks assigned to four workstations. The task assignments and the workstation time (WT) are given in the figure and the precedence relations are supposed to be the same as in Fig. 2. It can be seen from the figure that the WT of workstation 3 is the biggest among all workstations, so it is selected as the critical workstation. According to Fig. 2, task 7 is the predecessor of task 8, so task 7 can be inserted to workstation 2, however, it cannot be inserted to workstation 4 or exchanged with the tasks assigned in workstation 4. Moreover, as task 7 is the follower of task 2, they are not allowed to be exchanged by the swap operator. While task 7 and task 5 can be exchanged since they do not have precedence relations.

At each iteration of the local search, it first selects the critical workstation. Then the tasks assigned to the critical workstation are explored by the insert and swap operators sequentially. The variation of the objective value is calculated for each possible move. Once the solution is improved, the operator is implemented and the algorithm goes to the next iteration. The algorithm terminates if no improvements



(a) Illustration of insert operator.



(b) Illustration of swap operator.

Fig. 5 Illustration of insert and swap operator

are made throughout the iteration. The procedure of the above local search is given in Algorithm 4.

Algorithm 4 Procedures of local search

```
Input: a complete solution $S$
1:  Select critical workstation $W_t$ $(1 \le t \le M)$.
2:  $s \leftarrow t-1$.
3:  for each task $p \in W_t$ do
4:    for $n = 1$ to $2$ do
5:      if $1 \le s \le M$ and $p \to W_s$ is feasible do
6:        evaluate insert $(p \to W_s)$.
7:        if $S$ is improved do
8:          implement insert $(p \to W_s)$.
9:          go to line 1.
10:       end if
11:       for each task $q \in W_s$ do
12:         evaluate swap $(p \to W_s,\ q \to W_t)$.
13:         if $S$ is improved do
14:           implement swap $(p \to W_s,\ q \to W_t)$.
15:           go to line 1.
16:         end if
17:       end for
18:     end if
19:     $s \leftarrow t+1$.
20:   end for
21: end for
Output: the improved solution $S$.
```

Update probability model and upper bound

After the problem-specific local search, the elite solutions are used to update the probability model and the upper bound. The probability model is updated based on the population-based incremental learning (PBIL) method [26]. The basic idea is to increase the probability of using construction steps that are contained in high quality solutions via positive feedback [24, 27]. The specific update rule in EDA-BB is given as follows:

$$\rho_{ki}(g+1) = (1-\beta)\cdot\rho_{ki}(g) + \beta\cdot\frac{1}{\lfloor PS\cdot\alpha\rfloor}\cdot\sum_{h=1}^{\lfloor PS\cdot\alpha\rfloor} I_{ki}^{h}(g), \forall i,k, \tag{8}$$

where $\beta \in (0,1)$ represents the learning rate and $I_{ki}^{h}(g)$ is the following indicator function corresponding to the hth elite solution:

$$I_{ki}^{h}(g) = \begin{cases} 1, & \text{if task } i \text{ is assigned to workstation } k \text{ in the } h\text{th elite solution} \\ 0, & \text{otherwise} \end{cases}. \tag{9}$$

The upper bound is updated using the objective value of the best solution found in each generation as (11). In this way, the best-so-far objective value is recorded as the upper bound and will be used as a reference value when generating new solutions in the following generation:

$$UB(g+1) = \min\left\{UB(g), C_{\text{best}}\right\}, \tag{10}$$

where $UB(g)$ is the upper bound in generation g, C_{best} is the cycle time of the best solution.

Table 1 Parameter values

Parameters	Factor level			
	1	2	3	4
α	0.04	0.16	0.28	0.40
β	0.01	0.08	0.15	0.22

Table 2 Orthogonal array and ARVs

Experiment number	Factor level		ARV (%)
	α	β	
1	1	1	1.13
2	1	2	0.80
3	1	3	0.79
4	1	4	0.88
5	2	1	1.12
6	2	2	0.73
7	2	3	0.84
8	2	4	0.90
9	3	1	1.14
10	3	2	0.69
11	3	3	0.75
12	3	4	0.87
13	4	1	1.13
14	4	2	0.78
15	4	3	0.79
16	4	4	0.78

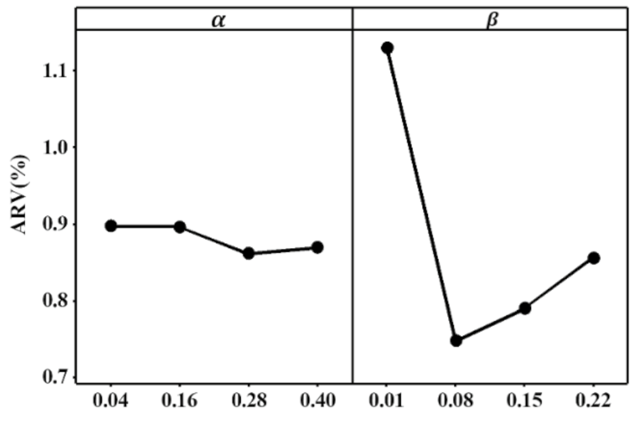


Fig. 6 Factor level trend

Complexity analysis

The time complexity of EDA-BB is analyzed as follows. According to the flowchart in Fig. 3, the time is mainly spent in running the searching loop in the proposed EDA-BB. In each generation of the loop, the algorithm needs to execute sampling, elite selecting, local search and update. The time complexities of elite selecting and update are easily derived as $O(PS \cdot \log PS)$ and $O(PS \cdot N + M \cdot N)$, respectively. The time complexity of the sampling can be calculated by Algorithms 1–3. In Algorithm 1, there are at most N tasks assigned to workstation m and for each task assignment, the time complexity is $O(N)$. So the time complexity of the task assignment is $O\left(N^2\right)$. Additionally, by introducing the *time list* WT_r, the time complexity of assigning the best-fit robot is $O(N \cdot R)$. Thus, the time complexity of generating the assignment for a workstation is $O\left(N^2 + N \cdot R\right)$. In Algorithm 2, the time complexity of selecting $PS/2$ partial solutions is $O(PS \cdot \log PS)$ and the time complexity of expanding $PS/2$ partial solutions into PS branches in the next stage is $O\left(PS \cdot N^2 + PS \cdot N \cdot R\right)$. Based on the analysis of Algorithm 1 and 2, the total time complexity of the sampling procedures in Algorithm 3 can be derived as $O\left(M \cdot PS \cdot \log PS + PS \cdot N^2 + PS \cdot N \cdot R\right)$. For the local search, the time complexity of one iteration is $O\left(N^2 + N \cdot R\right)$. As the local search terminates until no improvements can be made in one iteration, the number of iterations is at most $\Delta C = (UB - LB)$, where UB and LB are the upper bound and lower bound of the cycle time, respectively. Therefore, the total time

Table 3 Comparisons of EDA-BB with EDAnBB and EDAnLS

N	M	EDA-BB	EDAnBB		EDAnLS	
		$\overline{RPD}$	$\overline{RPD}$	p value	$\overline{RPD}$	p value
25	3	**0.000**	**0.000**	–	**0.000**	–
	4	**0.000**	**0.000**	0.171	0.003	0.000
	6	**0.000**	**0.000**	–	**0.000**	–
	9	**0.000**	**0.000**	–	0.001	0.081
35	4	**0.000**	**0.000**	–	0.002	0.000
	5	**0.000**	**0.000**	–	0.013	0.000
	7	**0.000**	0.003	0.003	0.010	0.000
	12	0.010	**0.000**	0.998	0.029	0.001
53	5	**0.000**	**0.000**	0.081	0.013	0.000
	7	**0.000**	**0.000**	–	0.005	0.000
	10	**0.000**	0.033	0.000	0.001	0.081
	14	**0.000**	0.029	0.000	0.012	0.000
70	7	**0.000**	0.002	0.002	0.014	0.000
	10	**0.000**	0.002	0.171	0.020	0.000
	14	**0.000**	0.016	0.000	0.004	0.021
	19	**0.000**	0.014	0.000	0.003	0.310
89	8	**0.000**	0.016	0.000	0.005	0.000
	12	**0.000**	0.001	0.295	0.009	0.000
	16	**0.000**	0.001	0.208	0.009	0.000
	21	**0.000**	0.009	0.000	0.006	0.013
111	9	**0.000**	0.018	0.000	0.011	0.000
	13	**0.000**	0.038	0.000	0.001	0.151
	17	**0.000**	0.029	0.000	0.011	0.000
	22	**0.000**	0.041	0.000	0.006	0.065
148	10	**0.000**	0.037	0.000	0.026	0.000
	14	**0.000**	0.067	0.000	0.006	0.027
	21	**0.000**	0.080	0.000	0.006	0.002
	29	**0.000**	0.075	0.000	0.005	0.023
297	19	**0.000**	0.040	0.000	0.004	0.040
	29	**0.000**	0.065	0.000	0.005	0.025
	38	**0.000**	0.070	0.000	0.003	0.070
	50	**0.000**	0.065	0.000	0.008	0.028

Bold values indicate the best results

complexity of the local search is $O\left(PS \cdot \Delta C \cdot \left(N^2 + N \cdot R\right)\right)$. Accordingly, the time complexity of the EDA-BB is $O\left(Gen \cdot \left(PS \cdot \left(\Delta C \cdot \left(N^2 + N \cdot R\right) + M \cdot \log PS\right) + M \cdot N\right)\right)$.

The space complexity of the EDA-BB can be analyzed as follows. In EDA-BB, each solution needs an array of size $(N + M)$ to store the task assignment and an array of size M to store the robot assignment. Moreover, for each solution, the size of the *time list* and *task list* is R and at most N, respectively. Thus, the space complexity of the EDA-BB is $O(PS \cdot (N + M + R))$.

Computational results

Experimental settings

To test the performance of the proposed EDA-BB, we carry out numerical experiments on the RALB-2 dataset with 32 benchmark instances which were used in [10, 11]. The instances are generated by adding the robot task times to the widely-used SALB dataset prepared by Scholl [28], where the precedence graphs can be found at https://www.assemblyline-balancin.de/. There are 8 problem scales in the dataset varying from 25 to 297 tasks. For each scale, there are four instances with different workstation and robot numbers. While for each instance, the numbers of workstations and robots are supposed to be the same.

The EDA-BB is coded in C++ and tested on a PC with an Intel Core i5-6500 CPU @ 3.2-GHz. As our comparative algorithms, the PSO and CS-PSO proposed in [11] are also coded in C++ but tested on Intel Core i5 CPU @ 2.3 GHz. For fair comparisons with PSO and CS-PSO, we use Passmark Performance Test on https://www.cpubenchmark.net to test the performance of both processors. The test results show that both processors are comparable. So our experimental results are compared with the results reported in [11] directly in the following comparisons.

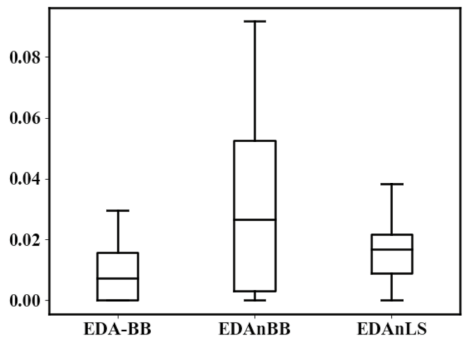


Fig. 7 Boxplot of RPD with or without B&B based knowledge and local search

Parameter settings

The EDA-BB has three key parameters: the population size PS, the rate of elite solutions α, and the learning rate β. The population size is a parameter determining the number of solutions generated in each generation. A small value will result in insufficient search in each generation, while a large value will result in long convergence time. We carry out a large amount of preliminary experiments to determine a moderate value of PS and a suitable stopping criterion of the algorithm. Through the preliminary experiments, the population size is set as 500, and the stopping criterion of the algorithm is set as 300 generations.

In order to investigate the effect of the other two parameters, the Taguchi method of design-of-experiments [29] is used. Four factor levels are considered for each parameter, as listed in Table 1. Accordingly, an orthogonal array $L_{16}\left(4^2\right)$ is selected, which includes 16 different (α, β) combinations. The 32 instances are used for the investigation. For each instance, the EDA-BB with each combination n is run ten times independently to obtain the average objective value $\overline{C}_n$. Then the best average objective value $\overline{C}_{\text{best}}$ among 16 combinations is derived as the reference objective value and the relative percentage deviation (RPD) of each combination n is calculated as $\text{RPD}_n = \left(\overline{C}_n - \overline{C}_{\text{best}}\right)/\overline{C}_{\text{best}} \times 100$. Finally, the average of RPD_ns in 32 instances is calculated as the average response value (ARV) for combination n. The orthogonal array and the ARVs are listed in Table 2.

Furthermore, the trend of each factor is shown in Fig. 6. It can be seen that β is more significant than α. The value of β controls the learning speed of the probability model. The bigger the value, the larger the learning speed. However, too big value will result in the premature convergence of the algorithm. The value of α determines the size of elite solution set in each generation. A small value leads the algorithm to learn from a small number of best solutions, while a large value allows more solutions to be learned. According to the results of DOE, moderate values are recommended for the two parameters: $\alpha = 0.28,\ \beta = 0.08$. These values are used for the following tests and comparisons.

Effect of B&B based knowledge and local search

To demonstrate the effectiveness of the B&B based knowledge and the problem-specific local search, we compare the EDA-BB with the EDA without B&B based knowledge (EDAnBB) and the EDA without local search (EDAnLS). In EDAnBB, the lower bound will not be used to evaluate the partial solutions in each stage of the sampling process. Thus, all the partial solutions will enter the next stage without selecting and branching. In EDAnLS, the elite solutions will be directly used to update the probability model without

Table 4 Comparisons between EDA-BB and other algorithms

N	*M*	PSO		CS-PSO		EDA-BB			
		RPD^b	CPU time (s)	RPD^b	CPU time (s)	RPD^w	RPD^m	RPD^b	CPU time (s)
25	3	**0.00**	2.7	**0.00**	3.6	**0.00**	**0.00**	**0.00**	**1.7**
	4	12.37	2.9	12.37	3.9	**0.00**	**0.00**	**0.00**	**2.2**
	6	3.09	3.0	7.22	4.2	**0.00**	**0.00**	**0.00**	**2.8**
	9	0.92	3.3	4.59	4.5	**0.00**	**0.00**	**0.00**	**4.1**
35	4	**0.00**	4.9	0.88	5.2	**0.00**	**0.00**	**0.00**	**3.1**
	5	0.91	5.4	2.13	6.3	**0.00**	**0.00**	**0.00**	**3.5**
	7	4.98	6.9	6.47	6.9	1.00	**0.00**	**0.00**	**4.1**
	12	8.42	8.5	10.53	8.9	4.21	0.02	**0.00**	**6.7**
53	5	**0.00**	13.1	1.11	13.5	**0.00**	**0.00**	**0.00**	**3.8**
	7	3.89	14.9	6.36	16.8	**0.00**	**0.00**	**0.00**	**5.3**
	10	8.87	16.2	10.34	17.9	**0.00**	**0.00**	**0.00**	**6.9**
	14	5.97	19.9	8.96	20.0	2.24	**0.00**	**0.00**	**9.4**
70	7	6.70	29.0	6.95	32.9	0.50	**0.00**	**0.00**	**7.1**
	10	10.46	32.5	12.55	35.8	1.67	0.01	**0.00**	**8.6**
	14	12.14	39.1	15.61	43.3	3.47	0.02	**0.00**	**11.6**
	19	13.82	43.4	19.51	47.8	4.88	0.03	**0.00**	**15.5**
89	8	4.78	41.9	5.47	45.7	1.14	**0.00**	**0.00**	**7.9**
	12	6.67	50.4	18.33	51.6	1.00	**0.00**	**0.00**	**10.1**
	16	3.79	59.6	10.90	63.3	0.95	**0.00**	**0.00**	**13.0**
	21	6.25	75.3	10.00	80.5	3.13	0.02	**0.00**	**17.3**
111	9	9.87	82.3	10.50	85.5	1.47	0.01	**0.00**	**12.2**
	13	14.64	89.5	12.86	92.5	0.36	**0.00**	**0.00**	**14.7**
	17	10.60	98.5	17.05	107.4	3.23	0.02	**0.00**	**18.2**
	22	13.75	110.8	15.63	114.5	3.75	0.02	**0.00**	**23.9**
148	10	8.21	179.8	10.04	183.5	2.37	0.01	**0.00**	**16.1**
	14	18.70	205.5	18.98	207.9	3.12	0.01	**0.00**	**19.5**
	21	19.21	215.9	20.96	219.5	3.06	0.01	**0.00**	**27.7**
	29	18.87	230.3	19.50	242.2	2.52	0.01	**0.00**	**40.6**
297	19	11.44	891.8	14.07	1118.3	2.25	0.02	**0.00**	**34.7**
	29	15.20	997.6	16.08	1331.3	3.51	0.02	**0.00**	**54.2**
	38	19.61	1269.9	15.69	1593.5	4.31	0.02	**0.00**	**77.6**
	50	13.33	1390.8	25.64	1664.3	3.08	0.01	**0.00**	**119.5**

Bold values indicate the best results

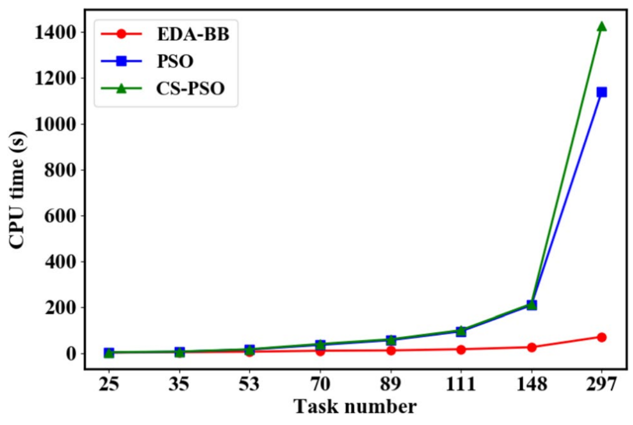


Fig. 8 Comparison of average CPU time

further improvement by the problem-specific local search. Because of the random factors in the EDA-based algorithms, each instance is run 20 times. For each run, the RPD of objective value obtained by each algorithm is calculated as $\mathrm{RPD}_{\mathrm{a}} = \left(C_{\mathrm{a}} - C_{\mathrm{best}}\right)/C_{\mathrm{best}} \times 100$, where C_{a} is the cycle time obtained by algorithm a, and C_{best} is the best cycle time among the three algorithms. The average RPD values of 20 runs are summarized in Table 3.

From Table 3, it can be seen that the average RPD obtained by EDA-BB is smaller than those obtained by the other two algorithms on most instances. Regarding the only exception, the small-scale instance with 35 tasks and 12 workstations, the EDAnBB performs better than EDA-BB. However, as the problem scale increases, the advantage of the EDA-BB becomes more apparently. This implies the effectiveness of the B&B based knowledge to reduce the search space for large scale problems. To analyze whether the advantages of the EDA-BB over the other two algorithms are significant, we carry out the nonparametric Mann–Whitney test with 95% confidence level. The p values are listed in Table 3, where "–" means the objective values obtained by the two tested algorithms are all same. The p values show that the EDA-BB is significantly better than EDAnBB on all large scale instances ($N > 89$). The advantage of EDA-BB over EDAnLS is also testified according to their average RPD and p values.

Figure 7 shows the boxplots of all RPD values obtained by the three algorithms. From Fig. 7, it can be seen that the box sizes of EDA-BB and EDAnLS are smaller than EDAnBB. This implies that the B&B based knowledge fused in the EDA is able to enhance the algorithm's stability in solving different scales of problems. Compared with EDAnLS, the objective values achieved by EDA-BB are smaller, which demonstrates the effectiveness of the local search in improving the elite solutions. From the above investigation, it can be concluded that the B&B based knowledge and the problem-specific local search are both effective in improving the performance of EDA.

Comparisons with other algorithms

Next, we compare EDA-BB with the PSO and CS-PSO which showed good performances in solving RALB-2 [11]. The results of PSO and CS-PSO are taken from [11] directly, and the results of EDA-BB are obtained by running each instance 20 times. Comparative results are reported in Table 4.

In Table 4, the RPD values of each algorithm are calculated as $\mathrm{RPD}^{b} = \left(C^{*} - C^{*}_{\mathrm{best}}\right)/C^{*}_{\mathrm{best}} \times 100$, where C^{*} denotes the best cycle time found by the certain algorithm and C^{*}_{best} denotes the best cycle time found by the three tested algorithms. In addition, another two RPD values are calculated for the proposed EDA-BB: $\mathrm{RPD}^{w} = \left(C^{w} - C^{*}_{\mathrm{best}}\right)/C^{*}_{\mathrm{best}} \times 100$ and $\mathrm{RPD}^{m} = \left(C^{m} - C^{*}_{\mathrm{best}}\right)/C^{*}_{\mathrm{best}} \times 100$, where C^{w} and C^{m} denote the worst and mean cycle time found by EDA-BB among 20 runs, respectively. In Table 4, the best results are displayed in bold font.

It can be seen that EDA-BB obtains the best objective values among the three algorithms for all instances. By contrast, PSO and CS-PSO only obtain four best objective values on small scale instances. Furthermore, the RPD^{w} of EDA-BB is smaller than the RPD^{b} of PSO and CS-PSO. This implies the worst solutions obtained by EDA-BB are even better than the best solutions obtained by the other two algorithms. In addition, EDA-BB achieves $\mathrm{RPD}^{w} = 0$ on nine small scale instances, which shows the stability of EDA-BB in solving small scale problems.

Moreover, the average CPU time of the three algorithms is also reported in Table 4. It can be seen that the CPU time cost of EDA-BB is much shorter than PSO and CS-PSO. To further investigate the relationship between the computational time and the problem scale, the trend of CPU time with the increase of task number is plotted in Fig. 8. It is clear that the CPU time of PSO and CS-PSO grows dramatically when the task number is larger than 89, while the CPU time of EDA-BB grows slightly.

From the computational results, it can be concluded that EDA-BB is of better performances than the existing algorithms in solving the RALB-2 in terms of both the solution quality and computational time.

Conclusion

In this paper, we propose a hybrid algorithm by fusing the B&B strategy as knowledge into the estimation of distribution algorithm to solve the RALB-2. The computational complexity of the proposed algorithm is analyzed and the performances are tested. Numerical comparison shows that the proposed EDA-BB performs more effective and efficient than the existing algorithms. The B&B based knowledge employs the lower bound with mathematical model analysis, while the EDA employs the data-driven search via probability model sampling. EDA-BB is the fuse of the model-based analysis and data-driven metaheuristics. The effectiveness of the proposed hybrid algorithm mainly owes to the probability model based exploration, the problem-specific local search based exploitation, and the B&B knowledge based sampling. Therefore, exploration and exploitation are well balanced in EDA-BB, and promising search area can be focused in sampling process.

The future work is to study the multi-objective RALBs by considering several objectives simultaneously like the cycle time, equipment cost, and energy consumption. It is also important to study the mixed-model RALBs by considering different product assembly line. In addition, adaptive swarm intelligence algorithms will be studied in the future research.

Acknowledgements This study is supported by the National Science Fund for Distinguished Young Scholars of China (No. 61525304) and the National Natural Science Foundation of China (No. 61873328).

Compliance with ethical standards

Conflict of interest The authors declare that we have no conflict of interest.

References

1. Boysen N, Fliedner M, Scholl A (2008) Assembly line balancing: which model to use when? Int J Prod Econ 111(2):509–528
2. Boysenaab N, Fliedner M, Scholl A (2007) A classification of assembly line balancing problems. Eur J Oper Res 183(2):674–693
3. Becker C, Scholl A (2006) A survey on problems and methods in generalized assembly line balancing. Eur J Oper Res 168(3):694–715
4. Rubinovitz J, Bukchin J, Lenz E (1993) RALB—a heuristic algorithm for design and balancing of robotic assembly lines. CIRP Ann Manuf Technol 42(1):497–500
5. Bukchin J, Tzur M (2000) Design of flexible assembly line to minimize equipment cost. IIE Trans 32(7):585–598
6. Tsai D-M, Yao M-J (1993) A line-balance-based capacity planning procedure for series-type robotic assembly line. Int J Prod Res 31(8):1901–1920
7. Kim H, Park S (1995) A strong cutting plane algorithm for the robotic assembly line balancing problem. Int J Prod Res 33(8):2311–2323
8. Levitin G, Rubinovitz J, Shnits B (2006) A genetic algorithm for robotic assembly line balancing. Eur J Oper Res 168(3):811–825
9. Scholl A, Becker C (2006) State-of-the-art exact and heuristic solution procedures for simple assembly line balancing. Eur J Oper Res 168(3):666–693
10. Gao J, Sun L, Wang L, Gen M (2009) An efficient approach for type II robotic assembly line balancing problems. Comput Ind Eng 56(3):1065–1080
11. Nilakantan JM, Ponnambalam SG, Jawahar N, Kanagaraj G (2015) Bio-inspired search algorithms to solve robotic assembly line balancing problems. Neural Comput Appl 26(6):1379–1393
12. Li Z, Janardhanan MN, Nielsen P, Tang Q (2018) Mathematical models and simulated annealing algorithms for the robotic assembly line balancing problem. Assem Autom 38(4):420–436
13. Li Z, Dey N, Ashour AS, Tang Q (2018) Discrete cuckoo search algorithms for two-sided robotic assembly line balancing problem. Neural Comput Appl 30(9):2685–2696
14. Li Z, Janardhanan MN, Ashour AS, Dey N (2019) Mathematical models and migrating birds optimization for robotic U-shaped assembly line balancing problem. Neural Comput Appl. https://doi.org/10.1007/s00521-018-3957-4
15. Nilakantan JM, Li Z, Tang Q, Nielsen P (2017) Multi-objective co-operative co-evolutionary algorithm for minimizing carbon footprint and maximizing line efficiency in robotic assembly line systems. J Clean Prod 156:124–136
16. Zhang Z, Tang Q, Li Z, Zhang L (2019) Modelling and optimisation of energy-efficient U-shaped robotic assembly line balancing problems. Int J Prod Res 57(17):5520–5537
17. Zhang Z, Tang Q, Zhang L (2019) Mathematical model and grey wolf optimization for low-carbon and low-noise U-shaped robotic assembly line balancing problem. J Clean Prod 215:744–756
18. Larrañaga P, Lozano JA (2001) Estimation of distribution algorithms: a new tool for evolutionary computation. Springer Science & Business Media, Berlin
19. Fang C, Kolisch R, Wang L, Mu C (2015) An estimation of distribution algorithm and new computational results for the stochastic resource-constrained project scheduling problem. Flex Serv Manuf J 27(4):585–605
20. Wang Y, Chen W (2019) A decomposition-based hybrid estimation of distribution algorithm for practical mean-cvar portfolio optimization. In: International conference on intelligent computing, Nanchang, pp 38–50
21. Wu C, Wang L (2018) A multi-model estimation of distribution algorithm for energy efficient scheduling under cloud computing system. J Parallel Distrib Comput 117:63–72
22. Wang S-Y, Wang L (2016) An estimation of distribution algorithm-based memetic algorithm for the distributed assembly permutation flow-shop scheduling problem. IEEE Trans Syst Man Cybern 46(1):139–149
23. Wang L, Wang S, Zheng X (2016) A hybrid estimation of distribution algorithm for unrelated parallel machine scheduling with sequence-dependent setup times. IEEE/CAA J Autom Sin 3(3):235–246
24. Grahl J (2008) Estimation of distribution algorithms in logistics: analysis, design, and application. Dissertation, University of Mannheim, Mannheim
25. Lipowski A, Lipowska D (2012) Roulette-wheel selection via stochastic acceptance. Phys A 391(6):2193–2196
26. Baluja S (1994) Population-based incremental learning. A method for integrating genetic search based function optimization and

competitive learning. Carnegie-Mellon University, Pittsburgh, Pennsylvania

27. Cheng R, He C, Jin Y, Yao X (2018) Model-based evolutionary algorithms: a short survey. Complex Intell Syst 4:283–292
28. Scholl A (1995) Data of assembly line balancing problems. Darmstadt Technical University, Department of Business Administration, Economics and Law, Institute for Business Studies (BWL)
29. Montgomery DC (2017) Design and analysis of experiments. Wiley, New York