Hindawi
Mathematical Problems in Engineering
Volume 2018, Article ID 2481435, 17 pages
https://doi.org/10.1155/2018/2481435

Research Article

Assembly Line Balancing Based on Beam Ant Colony Optimisation

Jiage Huo,[1] Zhengxu Wang,[2] Felix T. S. Chan,[1] Carman K. M. Lee,[1] and Jan Ola Strandhagen[3]

[1] *Department of Industrial and Systems Engineering, The Hong Kong Polytechnic University, Hong Kong, China*
[2] *School of Business Administration, Institute of Supply Chain Analytics, Dongbei University of Finance and Economics, Dalian 116025, China*
[3] *Department of Production and Quality Engineering, Norwegian University of Science and Technology, NO-7491 Trondheim, Norway*

Correspondence should be addressed to Zhengxu Wang; wangzhengxu@dufe.edu.cn

Received 7 March 2018; Revised 21 August 2018; Accepted 12 September 2018; Published 3 October 2018

Academic Editor: Akhil Garg

We use a hybrid approach which executes ant colony algorithm in combination with beam search (ACO-BS) to solve the Simple Assembly Line Balancing Problem (SALBP). The objective is to minimise the number of workstations for a given fixed cycle time, in order to improve the solution quality and speed up the searching process. The results of 269 benchmark instances show that 95.54% of the problems can reach their optimal solutions within 360 CPU time seconds. In addition, we choose order strength and time variability as indicators to measure the complexity of the SALBP instances and then generate 27 instances with a total of 400 tasks (the problem size being much larger than that of the largest benchmark instance) randomly, with the order strength at 0.2, 0.6 and 0.9 three levels and the time variability at 5-15, 65-75, and 135-145 levels. However, the processing times are generated following a unimodal or a bimodal distribution. The comparison results with solutions obtained by priority rule show that ACO-BS makes significant improvements on the quality of the best solutions.

1. Introduction

An assembly line is a continuous production line consisting of materials and workstations combined with conveyor belts, and it can link men and machines closely and efficiently [1]. Assembly lines are flow-oriented systems that are indispensable for both the production of high quantity standardized products and low volume production of customized products [2]. Effective design and redesign of assembly lines require high investment and running costs and are essential in manufacturing industries [3]. In addition, assembly planning and control play important roles in managing expanding product ranges, reducing delivery time and costs, and increasing profitability [4].

The Assembly Line Balancing Problem (ALBP) is a well-studied classic problem [5] and can be seen as a generalization of the Bin Packing problem where precedence constraints are added [6]. It focuses on assigning tasks to workstations with the aim of satisfying the precedence relationships among the tasks, the workload limitation of the workstations, and optimizing performance measures [7]. According to Becker and Scholl [8], there are four types of ALBP: SALBP-I aims to minimise the number of workstations with a given fixed cycle time; SALBP-II minimises the cycle time with a given number of workstations; SALBP-E aims to minimise the cycle time and the number of workstations at the same time by considering their relation with the total idle time or the inefficiency of the line; SALBP-F determines the feasibility of the problem with given the number of workstations and the cycle time.

ALBP is a well-known NP-hard problem, and it has been researched for more than sixty years. It was first studied by Salveson [5] who constructed a mathematical model of ALBP and suggested a solution procedure. For decades, the core problem has been extended to meet robotic, machining, and disassembly contexts, but even the simple version is still challenging [3].

Exact methods and approximate methods have been used to solve ALBP. According to Baybars [9], if n denotes the total number of tasks, there are n! possible sequences of tasks in SALBP; if there are r precedence constraints, then there are approximately n!/2r distinct, feasible sequences. Consequently, the required computational time for obtaining an optimal solution with an exact method for most of ALBP increases exponentially with the instance size considered [3]. This limits the performance of exact methodologies especially when the problem size is extremely large. Therefore, exploring efficient heuristic methodologies to cope with large scale ALBP within an acceptable time period is clearly necessary.

Recently, meta-heuristic algorithms such as the genetic algorithm, particle swarm optimisation and the ant colony optimisation algorithm have been used to deal with ALBP due to these algorithms' good performance on optimisation [1].

Swarm intelligence algorithms are based on the collective behavior in decentralized, self-organized systems and consist of agents interacting with each other and the environment. There is no centralized control structure. This kind of algorithms can be scalable since the number of agents can be easily added or removed. Besides, each agent is simple to design, and reliance on individual agents is small. Although each agent is not sophisticated, complex tasks can be solved in cooperation. As to ant colony algorithm, its main novel idea is the synergistic use of cooperation among many relatively simple agents which communicate by distributed memory implemented as pheromone deposited on edges of a graph [10]. The colony as a whole coordinates the activities without a direct communication between individual ants, as an isolated ant basically moves at random [11]. Each ant can build a solution step by step, and information left by other ants is used during the solution generation process. Good solutions can be obtained eventually without direct communication.

ACO has good performance on solving combinatorial optimisation problems. To effectively address the assembly line balancing problem with complicating factors such as parallel workstations, stochastic task durations, and mixed-models, McMullen and Tarasewich [12] proposed an approach based on ant techniques and, in comparison with other heuristics, showed that the proposed method is competitive with other heuristic methods in terms of the performance measures used in the study. Bautista and Pereira [13] used an ant algorithm incorporating some ideas that have offered good results with SALBP to solve the time and space constrained ALBP and get much better results than those by Tabu search. Kucukkoc and Zhang [14] and Zhong and Ai [1] also explored ALBP with ant colony based approaches. Therefore, ACO based methodologies show promising performance in coping with ALBP, and ACO is sufficiently flexible to be combined with other algorithms to achieve better performance.

With the development of products, the problem size is increasing and the complexity in the assembly process is greatly increasing to a large extent. Although many explorations have been undertaken by researchers, the development of methods to suit the complex assembly context is urgent, with the increasing of complexity of ALBP. In this study, we focus more on the performance of the algorithm on the large scale ALBP. In order to deal with ALBP in a large problem size, we hybridized ACO with Beam Search (ACO-BS) to improve the efficiency and improve the computational performance of the algorithm so that satisfactory results can be obtained within an acceptable computation time. The paper is organized as follows: Section 2 presents works solving ALBP by ant techniques; Section 3 shows the problem description with the mathematical model; the algorithm of ACO-BS is comprehensively developed in Section 4; the proposed algorithm is tested with benchmark instances at first and then the larger scale problems are generated to further explore the performance of the algorithm. The computational results are given in Section 5, and Section 6 gives contributions of the work and the future directions.

2. Literature Review

The ant colony algorithm has been applied to solve ALBP, and the traditional ant colony algorithms have been adapted to deal with the complex models of ALBP. Furthermore, researchers have also validated the effectiveness of the ant colony heuristic in solving ALBP. Baykasoğlu and Dereli [15] integrated the computer method of sequencing operations for assembly lines, ranked the positional weight heuristic, and the ant colony heuristic to deal with the simple and U-shaped ALBPs. Additionally, Fattahi et al. [16] developed a heuristic approach based on the ant colony optimisation approach to solve the medium- and large-size scales of this problem, since the problem is NP-hard. The experimental results validate the effectiveness and the efficiency of the proposed algorithm.

SALBP, which belongs to a class of intensively studied combinatorial optimisation problems known to be NP-hard, has attracted the attention of researchers and practitioners of operations research for almost half a century [2]. With the development of ALBP, the core problem has been extended from a manual assembly background to robotic, machining, and disassembly contexts; thus there are various industrial environments and line configurations [3]. Researchers try to narrow down the gap between academic research and the reality faced by practitioners, with more constraints considered in order to explore more realistic problems. To effectively address the assembly line balancing problem with complicating factors such as parallel workstations, stochastic task durations, and mixed-models, McMullen and Tarasewich [12] proposed an approach based on ant techniques, and comparison with other heuristics showed that the proposed method is competitive with other heuristic methods in terms of the performance measures used in this study. Simaria and Vilarinho [11] presented an method to solve the two-sided mixed-model ALBP with an ant colony optimisation algorithm, where two ants "work" simultaneously to build a balancing solution which verifies the precedence, zoning, capacity, side, and synchronism constraints. The superior performance of the approach was demonstrated by the results of a computational experiment. AkpıNar et al. [17] presented a hybrid algorithm combining an ant colony algorithm with a genetic algorithm for type I mixed-model ALBP with features such as parallel workstations, zoning constraints and sequence dependent setup times between tasks. To carry out assembly sequence planning and assembly line balancing simultaneously, Lu and Yang [18] proposed an ant colony algorithm based on the searching mechanism and the pheromone updating mechanism. In addition, the assembly task time, the time for changing assembly directions, changing assembly tools, and the time for moving heavy parts in the workstation were also considered.

There are many situations in which multiple objectives are taken into account and these objectives are sometimes conflicting. Therefore, methods to solve multiobjective ALBP are valuable to guide practitioners. McMullen and Tarasewich [19] simultaneously addressed the objectives of crew size, system utilization, the probability of jobs being completed within a certain time frame, and system design costs, and the superiority of the modified ant colony optimisation technique was shown in comparative results. Zha and Yu [20] presented a new hybrid algorithm of ant colony optimisation and filtered beam search to solve the U-line rebalancing problem with two objectives. In the process of constructing a path, each ant explores several nodes for one step and chooses the best one by global and local evaluation at a given probability. The proposed algorithm was shown to be good at solving the U-line rebalancing problem. Kucukkoc and Zhang [14] introduced a type-E parallel two-sided ALBP and proposed a new ant colony optimisation method with optimised parameters for solving the problem and found promising ways to simultaneously minimise two conflicting objectives, namely, cycle time and number of workstations.

Some researchers used one colony of ants to update the pheromone values and guide the searching process, while other researchers used multiple colonies of ants in the searching process so as to make the searching process more efficient. Multiple ants can be applied to the searching process in multiple objective problems. Agrawal and Tiwari [21] utilized collaborative ant colony optimisation, which maintained bilateral colonies of ants which independently identified the two sequences but utilized the information obtained by their collaboration to guide the future path in solving a balancing problem in mixed-model disassembly, and the effectiveness and robustness of the proposed approach were well demonstrated. Ozbakir et al. [22] studied parallel assembly lines with a novel multiple-colony ant algorithm, and the effective algorithm was examined with benchmark instances and compared with other algorithms.

In conclusion, there are many approaches related to ACO and ALBP, and developing approaches that can solve ALBP within an acceptable time were critical in real industrial applications, since ALBP is a NP-hard problem and ALBP of complex products brings new challenges. More advances in methods to solve ALBP are necessary to suit the dynamic and changing industrial environment.

3. Problem Description

3.1. Problem Description. ALBP is about assigning tasks to workstations, while optimizing certain criterion and not violating a number of possible restrictions, and can be divided into Simple Assembly Line Balancing Problem (SALBP) and General Assembly Line Balancing Problem (GALBP) [9].

For SALBP, the cumulative constraints associated with the available work time at workstations, and the precedence constraints established by the order in which the tasks must be executed need to be taken into account [23]. Nevertheless, GALBP problems contain additional considerations, such as the restricted assignment of tasks [24], or the assignment in a block of certain tasks [25]. Large scale SALBP is considered in this study.

According to Baybars [9], there are five main assumptions in SALBP:

(A-1) a task cannot be split among two or more stations, and all tasks must be processed.

(A-2) tasks cannot be processed in arbitrary sequences due to technological precedence requirements.

(A-3) all stations under consideration are equipped and manned to process any one of the tasks, and any task can be processed at any station.

(A-4) the task process times are independent of the station at which they are performed and of the preceding or following tasks.

(A-5) the assembly system is assumed to be designed for a unique model of a single product.

3.2. Mathematical Model

Notation

n: total number of tasks;

UB: upper bound of the total number of workstations;

$LB = \lceil \sum_{i=1}^{n} t_i / C \rceil$: lower bound of the total number of workstations, for $i = 1, \dots, n$;

t_i: processing time of task i, for $i = 1, \dots, n$;

C: cycle time;

P: set of pairs of tasks (i, k) such that i immediately precedes k;

$P_i(S_i)$: set of tasks that precede (succeed) i, for $i = 1, \dots, n$;

$E_i = \lceil (t_i + \sum_{k \in P_i} t_k)/C \rceil$: lower bound on the number of the workstation to which task i can be assigned, for $i = 1, \dots, n$;

$L_i = UB + 1 - \lceil (t_i + \sum_{k \in S_i} t_k)/C \rceil$: upper bound on the number of the workstation to which task i can be assigned, for $i = 1, \dots, n$;

Variables

$x_{ij} \in \{0, 1\}$ 1, if and only if task i is assigned to workstation j; otherwise, 0 ($\forall i$; $j = E_i, \dots, L_i$);

$y_j \in \{0,1\}$ 1, if and only if any task is assigned to workstation j ($j = LB+1,\dots,UB$); otherwise, 0.

The mathematical model of SALBP-I is as follows:

$$\min \quad z = \sum_{j=LB+1}^{UB} j \cdot y_j \tag{1}$$

$$\sum_{j=E_i}^{L_i} x_{ij} = 1 \quad i = 1,2,\dots,n \tag{2}$$

$$\sum_{i=1}^{n} t_i \cdot x_{ij} \le C \quad j = 1,2,\dots,LB \tag{3}$$

$$\sum_{i=1}^{n} t_i \cdot x_{ij} \le C \cdot y_j \quad j = LB+1,\dots,UB \tag{4}$$

$$\sum_{j=E_i}^{L_i} j \cdot x_{ij} \le \sum_{j=E_k}^{L_k} j \cdot x_{kj} \quad \forall (i,k) \in P \tag{5}$$

The objective function (1) minimises the total number of workstations; constraint (2) suggests that every task is assigned to one and only one workstation; workload constraints (3) and (4) imply that the total processing time of each workstation does not exceed the cycle time; constraint (5) ensures that all the precedence relations are satisfied.

3.3. Reversibility of ALBP. One ALBP instance can transfer to its reverse version after all the precedence relationships are reversed. If $S_{rev} = \{S_1,\dots,S_m\}$ is a solution for the reverse problem, then a solution for the original problem can be obtained by inverting the workstation orders of S_{rev}. Thus, a solution for the original problem can be $S = \{S_m,\dots,S_1\}$. Following Bautista and Pereira [13], we solve the original problem and the reverse problem, respectively, and then choose a better solution from the solutions obtained.

Before comparing the solutions of the two versions, solutions of the reverse problem are transferred to those of the original problem. There are two criteria for selecting the best solution: (1) number of workstations and (2) idle time in the last workstation. The second criterion is added due the fact that there are always large plateaus when only the first criterion is used, and more idle time in the last workstation means better resource utilization of the previous workstations. When there are several solutions with the same number of workstations, preference goes to the solution with more idle time in the last workstation, and then the solution will be chosen randomly if there are still ties after the above two steps.

4. The Algorithm of ACO-BS

For the classical ant colony algorithm, many ants search for solutions separately in one iteration. According to Dorigo et al. [28], there are three kinds of classical ant colony algorithm: ant system, ant-density, and ant-quantity, and for the latter two models, each ant lays pheromone at each step, while for ant system, ants lay pheromone after the end of the tour. Thus, pheromone values are updated by global information in the ant system model, while local information is used to update the pheromone values in the other two models. Not surprisingly, the results of ant system model are better since global information rather than local information is used to guide the solution searching process.

However, with the ant system model, although ants may start from different starting points, there will still be large amounts of repetition when the algorithm progresses step by step during the searching process. For SALBP-I, even if different tasks are chosen by different ants to assign the current workstation, there is still a large possibility that the task sets generated by some ants for one workstation are the same. When there is no rule to prevent this kind of repetition, the searching process will not be effective when compared to other methods. This motivates us to add rules to prevent the repetition of assignment for each workstation.

Beam search is an adaptation of the branch and bound method in which only certain nodes are evaluated, and only promising nodes are kept for further branching and the remaining nodes are pruned permanently [29]. Beam width and the number of extensions are two important parameters in beam search, which progresses level by level, and moves downward from the best B_{wid} promising nodes at each level. B_{wid} is defined to be the beam width [29]. Meanwhile, the running time of beam search is polynomial of the problem size; thus an efficient searching process must involve more searching constraints. The extension number allowed for each node can be restricted to further speed up the searching process.

Since the beam search algorithm progresses level by level and ALBP can be seen as task assignment workstation by workstation, the searching framework of the beam search can be easily applied to ALBP. Of course, repetition of task assignment for every workstation can be controlled when using the beam search structure. On the other hand, for each workstation, selection rule in the ant colony algorithm can be used for task selection, and the quality of solution in one iteration can be used in the next iteration to guide the searching direction.

The general logic of the ACO-BS algorithm is as follows: At first, due to the reversibility of SALBP-I, a better solution is chosen after solving the original problem and the reverse problem once respectively, following the priority rule. The chosen solution is used to initialize the best-so-far solution (S_{bsf}). In addition, the results obtained by priority rule are compared with those obtained by ACO-BS to show the extent that ACO-BS improves the quality of solutions. Next, within a certain time, solutions to original problem and the reverse problem are obtained by using ACO-BS for each iteration. The pheromone values are updated corresponding to the solutions obtained so as to guide the searching process of the next iteration. If there is a better solution than S_{bsf} (determined by the abovementioned criteria in Section 3.3), S_{bsf} is updated by a better one.

4.1. Priority Rule. Before assigning tasks, the priority values of tasks are computed as follows:

$$\eta_j = \frac{t_j}{C} + \frac{|S_j|}{\max_{1\le i\le n} |S_i|}, \quad j = 1,\dots,n \tag{6}$$

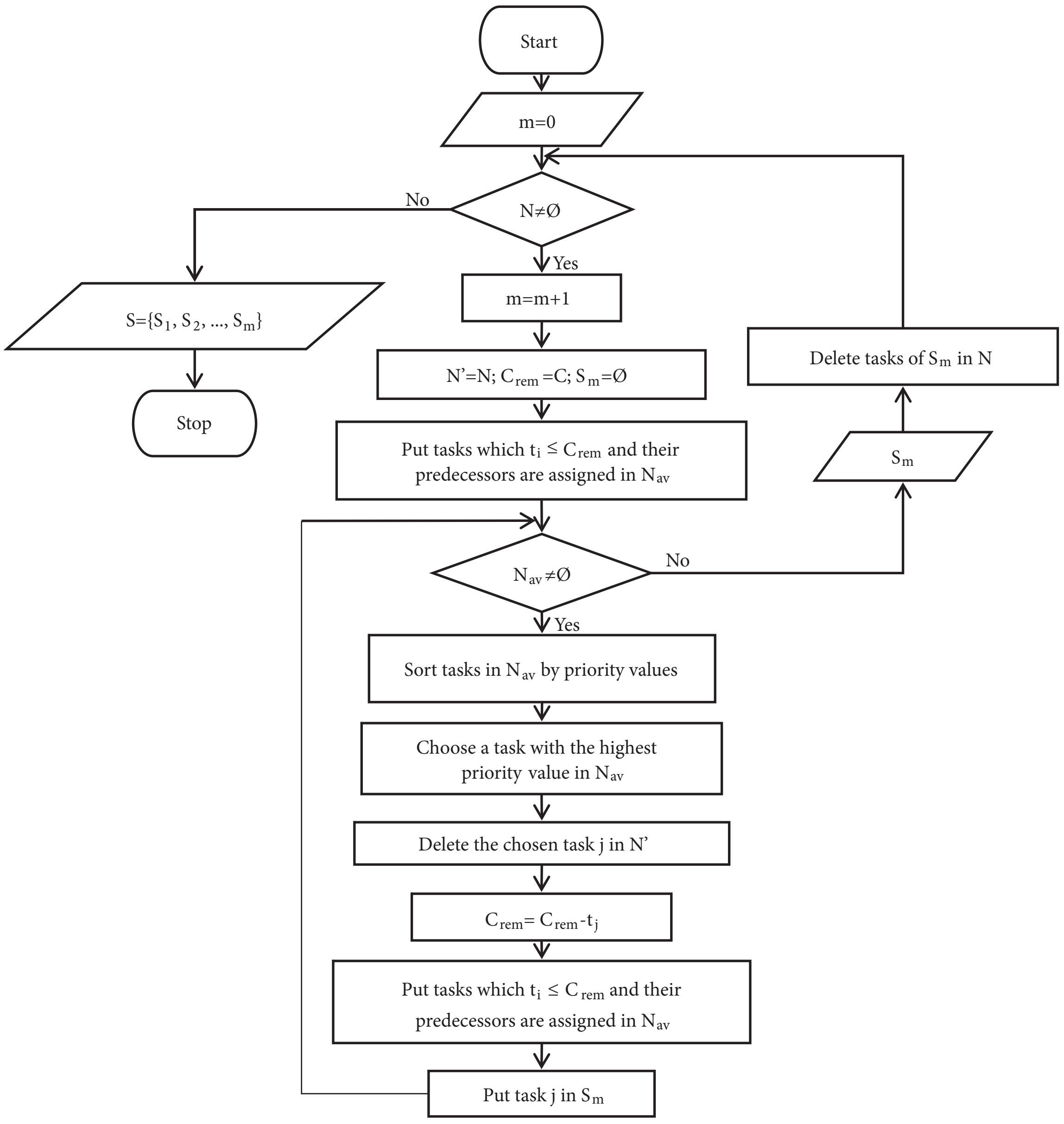
Start
m=0
N≠Ø
No
Yes
m=m+1
S={S1, S2, ..., Sm}
Stop
N'=N; Crem =C; Sm=Ø
Put tasks which ti ≤ Crem and their predecessors are assigned in Nav
Delete tasks of Sm in N
Sm
Nav≠Ø
No
Yes
Sort tasks in Nav by priority values
Choose a task with the highest priority value in Nav
Delete the chosen task j in N'
Crem= Crem-tj
Put tasks which ti ≤ Crem and their predecessors are assigned in Nav
Put task j in Sm

Figure 1: Flowchart for priority rule.

Starting from the first workstation, put all the tasks in a set N and set the idle time to be C. $|S_j|$ is the number of successors of task j. Also, tasks whose predecessors have been assigned are put into the set N_{nopre}. Figure 1 shows the flowchart for priority rule. The assignment is implemented by the following steps:

Step 1. Determine the available tasks. Examine tasks in N_{nopre} and put all tasks with processing time equal to the idle time into the available task set N_{av}, since saturating the time resource of a workstation is preferable in order to improve the utilization of resources. If N_{av} is empty, tasks with no predecessor and processing time less than the idle time are put into set N_{av}.

Step 2. If N_{av} is not empty, choose a task with the highest priority value from N_{av} (if there is more than one task with the highest priority value, choose one from them randomly), and go to Step 3. If the set is empty, go to Step 4.

Step 3. Delete the chosen task from N, set N_{nopre} and N_{av} to be ϕ, and the idle time will decrease by the processing time of the assigned task. Go to Step 1.

Step 4. Close the current workstation. If N is not empty, open a new one and set the cycle time to be C and then go to Step 1; if N is empty, end the procedure.

4.2. ACO-BS Algorithm. There are four steps in the ACO-BS algorithm. The first step is used to initialize the parameters,

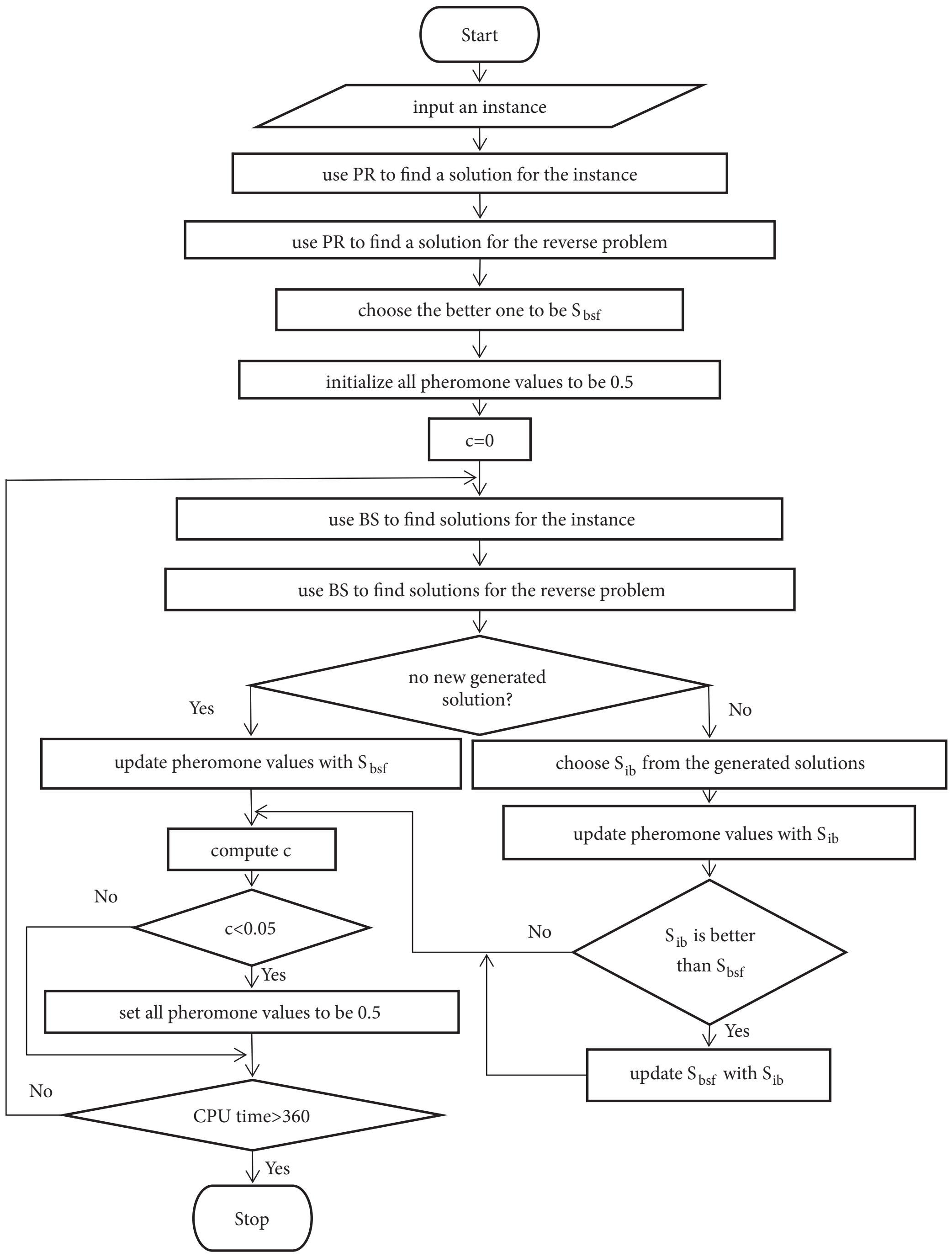
Start
input an instance
use PR to find a solution for the instance
use PR to find a solution for the reverse problem
choose the better one to be S_{bsf}
initialize all pheromone values to be 0.5
c=0
use BS to find solutions for the instance
use BS to find solutions for the reverse problem
no new generated solution?
Yes
No
update pheromone values with S_{bsf}
choose S_{ib} from the generated solutions
update pheromone values with S_{ib}
compute c
No
c<0.05
Yes
set all pheromone values to be 0.5
No
S_{ib} is better than S_{bsf}
Yes
update S_{bsf} with S_{ib}
No
CPU time>360
Yes
Stop

Figure 2: Flowchart for ACO-BS. c denotes the convergence value.

and Steps 2 to 4 are repeated within a certain time. The flowcharts for ACO-BS and BS are shown in Figures 2 and 3.

Step 1 (initialization). Generate one solution by using the priority rule described in Section 4.1 for the original problem and its reverse version. Then two criteria that are introduced in Section 3.3 are used to choose a better solution, which is used to initialize the best-so-far solution.

Additionally, the pheromone value is one important concept in the ant colony algorithm, and it is used to guide the searching process for good solutions. Let τ_{ij} ($i = 1, \ldots, UB$; $j = 1, \ldots n$) be the pheromone value between task j and workstation i, and all the pheromone values are initialized to be 0.5.

Step 2 (generate solutions from the original problem and the reverse one respectively by BS). Unlike the priority rule

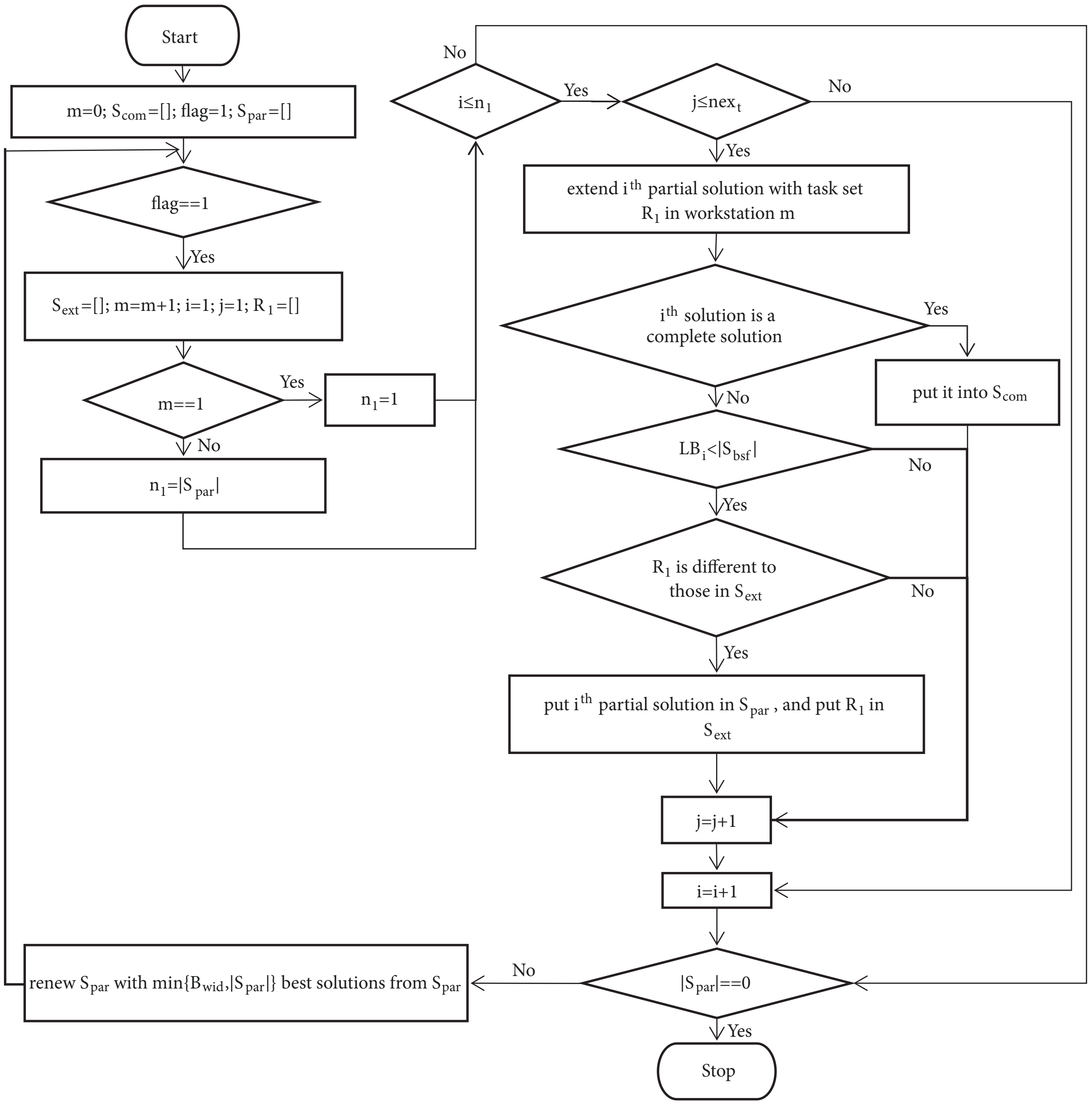
Start
m=0; S_com=[]; flag=1; S_par=[]
flag==1
Yes
S_ext=[]; m=m+1; i=1; j=1; R_1=[]
m==1
Yes
n_1=1
No
n_1=|S_par|
No
i≤n_1
Yes
j≤next_t
No
Yes
extend i^th partial solution with task set R_1 in workstation m
i^th solution is a complete solution
Yes
put it into S_com
No
LB_i<|S_bsf|
No
Yes
R_1 is different to those in S_ext
No
Yes
put i^th partial solution in S_par, and put R_1 in S_ext
j=j+1
i=i+1
renew S_par with min{B_wid,|S_par|} best solutions from S_par
No
|S_par|==0
Yes
Stop

Figure 3: Flowchart for BS.

which is used for task selection, the selection rule here makes use of the pheromone values and priority values of the tasks. Unlike the computation of priority values used in priority rule, the priority values used in ACO-BS are processed as follows [30] after they are computed using (6):

$$\eta_j' = \frac{\eta_j - \eta_{\min} + 1}{\eta_{\max}}, \quad j = 1, \ldots, n \tag{7}$$

where $\eta_{\min} = \min_{1\leq j\leq n}\eta_j$ and $\eta_{\max} = \max_{1\leq j\leq n}\eta_j$.

When choosing tasks from the set N_{av}, the probability p_j that task j is chosen by workstation k is used. Choose a task by maximizing the probability p_j or by roulette-wheel method is determined randomly with the same probability. p_j is calculated by the summation rule as follows [Choose a task by maximizing the probability p_j or by roulette-wheel method is determined 29]:

$$p_j = \frac{\left(\sum_{i=1}^{k} \tau_{ij}\right) \cdot \eta_j'}{\sum_{q\in N_{av}} \left(\sum_{i=1}^{k} \tau_{iq}\right) \cdot \eta_q'} \tag{8}$$

Partial solutions are extended by a set of tasks assigned to one workstation. In order to better illustrate the procedure of the algorithm, we illustrate the situation for the first workstation at first, and then the next steps are given.

At first, task assignment for the first workstation is explored for n_{ext} times, and the procedure is the similar as that by priority rule, but task selection rule here contains pheromone values and the priority values of tasks. Let S_{par} be the initial empty partial solution set and S_{ext} be the set that stores the task sets of the last workstation of all the partial solutions. For the first workstation, the two sets are the same. After each exploration, the task assignment for the first workstation, which is different to those in S_{ext} and its lower bound (will be described later) is less than $|S_{bsf}|$, which is the number of workstations needed in the best-so-far solution, is put into S_{par} and S_{ext}, because the task assignment for the first workstation is also a partial solution.

After the assignment of the first workstation, there will be at most n_{ext} partial solutions in S_{par}. One partial solution is picked one time, and then the following steps are repeated until the partial solution is extend for n_{ext} times (Let m denotes the workstation which is currently considered; $S_{ext} = \phi$):

(S1) $ext = 1$; $R_1 = \phi$ stores the task set for workstation m.

(S2) implement task assignment for workstation m, and get the task set R_1 for the workstation.

(S3) extend the partial solution considered by the task set R_1 for workstation m. If the extended solution is a complete solution, go to Step 4, else go to Step 5.

(S4) put the extended partial solution to S_{com} which stores the complete solutions.

(S5) If the lower bound (described below) of the workstation needed after the assignment for the partial solution is less than $|S_{bsf}|$ and R_1 is different from all the factors in S_{ext}, the partial solution will be put into S_{par}.

(S6) if $ext = n_{ext}$, end this procedure; else ext $= ext+1$ and $R_1 = \phi$ and go to Step 2.

There are criteria to select the partial solutions generated, and only partial solutions which will not lead to solutions worse than S_{bsf} can be extended in the next round. When choosing extensions after filling one workstation, two criteria are used. First, let N_{rem} be the set of tasks not assigned according to one partial solution s, and the lower bound on the workstations needed is as follows [2]:

$$LB_s = \left\lceil \frac{\sum_{j \in N_{rem}} t_j}{C} \right\rceil \tag{9}$$

Partial solutions are ranked by increasing the lower bound defined above. If there are ties after ranking by the first criterion, our preference goes to partial solutions with less idle time in the last workstation (further ties are broken randomly). Finally, for each workstation considered, there will be $\min\{B_{wid}, |S_{par}|\}$ generated, and B_{wid} denotes the width of beam and $|S_{par}|$ denotes the number of partial solutions obtained.

This step ends where there is no partial solution that can be extended. The partial solution set is empty when it is about to open a new workstation, and the extended partial solution, which is the complete solution, is put into S_{com}.

Step 3 (choose the iteration best solution and update pheromone values). Since in Step 2, if the lower bound for a partial solution is no less than $|S_{bsf}|$, it is aborted. Thus, there may be a situation in which there is no solution obtained in Step 2. If so, the best-so-far solution is used to update the pheromone values.

If there is a solution obtained in Step 2, an iteration best solution S_{ib} is chosen with the two criteria introduced in Section 3.3. S_{ib} is then used to update the pheromone values. Pheromone values τ_{ij} between task j and workstation i ($i = 1, \ldots, |S_{ib}|$; $j = 1, \ldots, n$) are updated. There are two updating processes, (1) pheromone evaporation, for each τ_{ij} needs to be updated, and there is $(1 - \rho) \cdot \tau_{ij}$ left after evaporation. $\rho \in (0, 1]$ is the evaporation rate, assigned as 0.1 in this study. (2)τ_{ij} increases ρ when task j is assigned to workstation i in S_{ib}.

When a pheromone value τ_{ij} is too small, task j tends never to be assigned to workstation i; when the value is too large, task j tends always to be assigned to workstation i. Consequently, the solution space is small, and this may lead to bad quality of the solutions generated. Thus, the pheromone values are restricted to the interval $[\tau_{\min}, \tau_{\max}]$ to prevent stagnation [32], and $\tau_{\min} = 0.01$ and $\tau_{\max} = 0.99$. If a pheromone value is larger than $\tau_{\max}$ after updating, it will be replaced by $\tau_{\max}$; if the value is smaller than $\tau_{\min}$ after updating, it is replaced by $\tau_{\min}$.

If S_{ib} is better than S_{bsf} (by using criteria in Section 3.3), S_{bsf} is updated by S_{ib}.

Step 4 (calculating the convergence value). After the initialization of the pheromone values in Step 1, the convergence value is 1. All the pheromone values are initialized to be 0.5 when the convergence value is less than 0.05. According to Kong et al. [33], pheromone reinitialization is an important strategy to avoid premature convergence by preventing the algorithm searching around a local optima continuously with low effectiveness. The convergence value is calculated as follows [30]:

$$\begin{aligned} &\textit{convergence} \\ &\quad = 2 \cdot \left(\frac{\sum_{j=1}^{n} \sum_{i=1}^{|S_{bsf}|} \min \left\{ \tau_{\max} - \tau_{ij}, \tau_{ij} - \tau_{\min} \right\}}{n \cdot \left| S_{bsf} \right| \cdot \left(\tau_{\max} - \tau_{\min} \right)} \right) \end{aligned} \tag{10}$$

5. Computational Results

The ACO-BS algorithm was implemented in MATLAB and was run on all the instances using an Intel Core i7-6700 (3.40 gigahertz) processor, with 32 gigabytes of available memory. The computation times spent on obtaining the best solutions and the standard deviations are reported, and all running time reported is given in CPU time seconds.

5.1. Results of Benchmark Instances of SALBP-I

5.1.1. Results by ACO-BS. In order to exhibit the superior performance of the algorithm developed in this paper, we tested the algorithm with benchmark instances (SALBP-I) published on https://assembly-line-balancing.de/.

There are 269 benchmark instances of SALBP-I. Optimal solutions can be obtained for 170 instances by using the priority rule only. After ten runs of ACO-BS (360 CPU time seconds for each run, there are 87 more instances whose optimal solutions can be obtained by the ACO-BS algorithm. There are 12 instances whose optimal solutions cannot be found by ACO-BS ($n_{ext} = 10$, $B_{wid} = 20$), however, when the time limit increases, the results are better. For example, for the instance Warnecke (with task number of 58 and cycle time of 60), the optimal result can be found, with the average solution found to be 27.7 (standard variation is 0.483); there are three runs in ten in which the optimal solution can be found, with the running times of 2931.790, 2687.077, and 3570.333. Of course, due the increased searching space, the results can be better when the width of the beam and the number of extensions increase. However, this will increase the running time.

Table 1 show the results of the benchmark instances. For each instance, the given cycle time, best solution ever found, solution found by priority rule, the best solution found by ACO-BS, the difference between solution found by priority rule, and the best solution found by ACO-BS are reported. Besides, the average and standard variation of solutions found in ten runs and the running times are also reported. The running time here is the computational time to find the best solution by ACO-BS for the first time. We can see from Table 1 that the algorithm performs well in most instances, but there are some instances that are a little tricky (tricky instances here refer to those whose optimal results cannot be found or cannot be found in every run). Such tricky instances are marked in Table 1. We explore their characteristics in order to have a better understanding of the complex instances and lay the foundation for the generation of such tricky instances.

5.1.2. Comparative Results with ACO, Genetic Algorithm, and Particle Swarm Algorithm. According to the results shown in Table 1, ACO-BS can achieve significantly better results than those obtained by the priority rule. However, further comparative experiments are needed to show the superiority of ACO-BS. The Ant Colony Optimization (ACO), which has similar framework with ACO-BS except the beam search part, Genetic Algorithm (GA) in Leu et al. [26] and Particle Swarm Optimization (PSO) in Dou et al. [27], are used to compare with the ACO-BS.

Since ACO-BS begins with a solution obtained by the priority rule, the other algorithms will also use the priority rule to get the initial solutions. Specifically, ACO will initialize the best-so-far solution by the priority rule, and the number of ants is set to be 20 to compare with ACO-BS which has the beam width of 20; GA with population size of 50 has initial solutions obtained by the priority rule and four heuristic methods in Leu et al. [26] (except the third one in [26]), and 10 initial solutions are obtained with these five methods applied to the original problem and the reverse one; PSO has 30 initial solutions, with 10 obtained the same way as the 10 initial solutions obtained in GA, and the other initial solutions are randomly generated. Thus, the best solutions

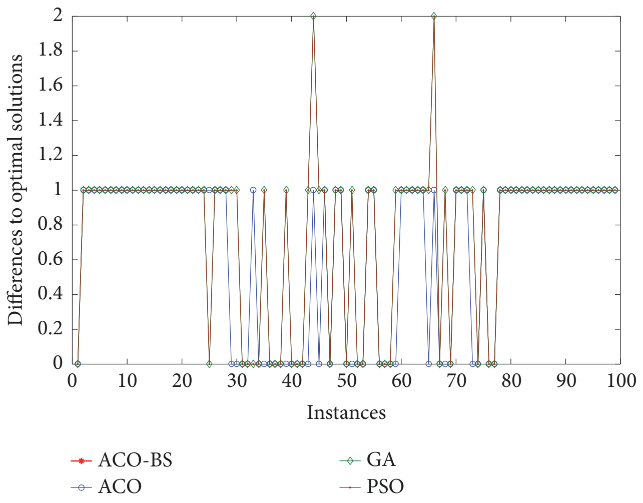

Figure 4: Comparison results between ACO, GA, in Leu et al. [26], and PSO in Dou et al. [27].

found by ACO, GA, and PSO will not be worse than those found by the priority rule as well. The other parameters in GA and PSO are the same as those in the corresponding two published papers.

The numbers of instances that can reach optimal solutions by ACO, GA, and PSO are 33, 24, and 23, respectively. Figure 4 shows the comparative results between ACO, GA, and PSO on the 99 benchmark instances whose optimal solutions cannot be reached by the priority rule. On the x axis are the 99 instances, and on the y axis are the differences between the best solutions found in 10 runs within 360 CPU time seconds and the corresponding optimal solutions. For GA and PSO, the difference between the best solution found and the corresponding optimal one ranges from 0 to 2. However, the largest difference between the best solution found by ACO and the corresponding optimal solution is 1. What is more, there are significantly more points related to ACO falling on the x axis, which means that among the three algorithms considered, ACO has comparatively better ability in searching solutions for ALBP.

In order to highlight the integration of beam search to ACO, ACO is compared with ACO-BS. With the increase of beam width in ACO-BS from 20 to 100, the number of ants used in ACO is increased to be 100, and the running time limit is also set to be 720 CPU time seconds. The numbers of instances whose optimal solutions can be reached by ACO and ACO-BS are 33 and 87, respectively. The comparative result is shown in Figure 5. From the points falling on x axis, we can see that there are many instances, whose optimal solutions cannot be reached by ACO but they can be reached by ACO-BS. Thus, with the integration of beam search, ACO-BS achieves significantly better solutions than those obtained by ACO. Thus, ACO-BS improves a lot in solving ALBP, compared with ACO.

Therefore, ACO shows superiority compared with GA in Leu et al. [26] and PSO in Dou et al. [27], and ACO-BS improves ACO by integrating beam search in ACO.

Table 1: Results of benchmark instances.

Instances	C	Optimal	Priority rule	ACO-BS	Difference	Solution		Running time [s]	
						avg.	std.	avg.	std.
Arcus1	3786	21	22	21∗	1	21	0	3.437	0.077
Arcus2	11570	13	14	13∗	1	13	0	65.417	60.831
Barthol2	84	51	52	51∗	1	51	0	7.575	0.127
Barthol2	85	50	51	51	0	51^T	0	0.254	0.022
Barthol2	87	49	50	49∗	1	49	0	7.818	0.161
Barthol2	89	48	49	48∗	1	48	0	8.277	0.182
Barthol2	91	47	48	47∗	1	47	0	8.333	0.257
Barthol2	93	46	47	46∗	1	46	0	8.034	0.201
Barthol2	95	45	46	45∗	1	45	0	8.193	0.153
Barthol2	97	44	45	44∗	1	44	0	8.013	0.175
Barthol2	99	43	44	43∗	1	43	0	9.656	3.344
Barthol2	101	42	43	42∗	1	42	0	11.605	7.768
Barthol2	104	41	42	41∗	1	41	0	8.338	0.182
Barthol2	106	40	41	40∗	1	40	0	10.094	3.506
Barthol2	109	39	40	39∗	1	39	0	8.675	0.203
Barthol2	112	38	39	38∗	1	38	0	8.583	0.197
Barthol2	115	37	38	37∗	1	37	0	8.758	0.174
Barthol2	118	36	37	36∗	1	36	0	8.864	0.127
Barthol2	121	35	36	35∗	1	35	0	11.590	4.274
Barthol2	125	34	35	34∗	1	34	0	8.886	0.153
Barthol2	129	33	34	33∗	1	33	0	9.145	0.144
Barthol2	133	32	33	32∗	1	32	0	9.107	0.175
Barthol2	137	31	32	31∗	1	31	0	9.190	0.115
Barthol2	146	29	30	29∗	1	29	0	10.081	2.874
Barthol2	152	28	29	28∗	1	28	0	9.309	0.130
Barthol2	157	27	28	27∗	1	27	0	9.233	0.117
Barthol2	163	26	27	26∗	1	26	0	9.232	0.118
Barthol2	170	25	26	25∗	1	25	0	9.306	0.109
Barthold	403	14	15	14∗	1	14	0	8.093	0.255
Barthold	434	13	14	13∗	1	13	0	8.597	0.108
Barthold	470	12	13	12∗	1	12	0	8.691	0.130
Barthold	513	11	12	11∗	1	11	0	8.618	0.131
Barthold	626	9	10	9∗	1	9	0	8.423	0.103
Buxey	41	8	9	8∗	1	8	0	0.812	0.014
Buxey	47	7	8	7∗	1	7	0	16.977	4.762
Gunther	54	9	10	9∗	1	9	0	1.035	0.011
Heskiaoff	205	5	6	5∗	1	5	0	1.050	0.031
Jackson	10	5	6	5∗	1	5	0	0.081	0.015
Kilbridge	69	8	9	8∗	1	8	0	1.784	0.065
Kilbridge	79	7	8	7∗	1	7	0	1.758	0.022
Kilbridge	92	6	7	6∗	1	6	0	1.736	0.050
Kilbridge	138	4	5	4∗	1	4	0	1.567	0.035
Lutz2	11	49	50	49∗	1	49	0	8.561	6.649
Lutz2	12	44	47	44∗	3	44.5^T	0.527	99.253	84.952
Lutz2	13	40	42	40∗	2	40	0	7.149	6.477
Lutz2	14	37	38	37∗	1	37.7^T	0.483	61.574	105.705
Lutz2	15	34	35	34∗	1	34	0	7.567	4.152
Lutz2	16	31	33	31∗	2	31	0	8.813	5.541
Lutz2	17	29	30	29∗	1	29	0	14.619	6.265

Table 1: Continued.

Instances	C	Optimal	Priority rule	ACO-BS	Difference	Solution		Running time [s]	
						avg.	std.	avg.	std.
Lutz2	18	28	29	28∗	1	28	0	27.185	11.602
Lutz2	19	26	27	26∗	1	26	0	8.493	5.250
Lutz2	20	25	26	25∗	1	25	0	2.688	0.081
Lutz2	21	24	25	24∗	1	24	0	2.748	0.078
Lutz3	110	15	16	15∗	1	15.8^T	0.422	31.939	42.531
Lutz3	118	14	15	14∗	1	14.4^T	0.516	82.378	107.850
Mansoor	62	3	4	3∗	1	3	0	0.130	0.007
Mukherje	201	22	23	22∗	1	22	0	4.698	0.094
Sawyer	41	8	9	8∗	1	8	0	0.959	0.020
Sawyer	47	7	8	7∗	1	7.1^T	0.316	225.300	145.082
Tonge	251	14	15	14∗	1	14.7^T	0.483	17.315	36.100
Tonge	320	11	12	11∗	1	11	0	3.575	0.079
Warnecke	54	31	32	31∗	1	31	0	50.685	47.707
Warnecke	56	29	30	29∗	1	29	0	20.740	14.768
Warnecke	60	27	29	28	1	28^T	0	7.998	5.720
Warnecke	62	27	28	27∗	1	27	0	21.701	16.420
Warnecke	65	25	27	25∗	1	25	0	4.152	2.172
Warnecke	68	24	25	24∗	1	24	0	13.874	11.894
Warnecke	71	23	24	23∗	1	23	0	14.076	13.767
Warnecke	82	20	21	20∗	1	20	0	3.393	2.080
Warnecke	92	17	18	17∗	1	17	0	4.433	3.124
Warnecke	104	15	16	15∗	1	15	0	2.266	0.037
Warnecke	111	14	15	14∗	1	14	0	3.053	1.544
Wee-mag	30	62	63	62∗	1	62	0	6.444	2.356
Wee-mag	46	34	35	34∗	1	34	0	8.894	6.803
Wee-mag	47	32	33	33	0	33^T	0	10.019	5.176
Wee-mag	52	31	32	31∗	1	31	0	4.056	0.040
Wee-mag	56	30	31	30∗	1	30	0	4.113	0.034
Scholl	1394	50	51	51	0	51^T	0	7.531	9.124
Scholl	1452	48	49	48∗	1	48.9^T	0.316	24.050	7.363
Scholl	1483	47	48	48	0	48^T	0	12.256	19.041
Scholl	1515	46	47	47	0	47^T	0	13.600	10.953
Scholl	1584	44	45	44∗	1	44.9^T	0.316	75.227	118.097
Scholl	1659	42	43	43	0	43^T	0	3.568	0.032
Scholl	1742	40	41	40∗	1	40.8^T	0.422	47.947	66.518
Scholl	1787	39	40	39∗	1	39.8^T	0.422	174.619	130.467
Scholl	1834	38	39	38∗	1	38.2^T	0.422	200.751	111.410
Scholl	1883	37	38	38	0	38^T	0	149.149	160.662
Scholl	1935	36	37	37	0	37^T	0	185.443	149.573
Scholl	1991	35	36	35∗	1	35.1^T	0.316	200.308	113.994
Scholl	2049	34	35	35	0	35^T	0	45.256	18.299
Scholl	2111	33	34	34	0	34^T	0	50.165	74.052
Scholl	2177	32	33	32∗	1	32.7^T	0.483	156.656	103.493
Scholl	2247	31	32	32	0	32^T	0	47.580	24.497
Scholl	2322	30	31	30∗	1	30.9^T	0.316	75.187	89.942
Scholl	2402	29	30	29∗	1	29.4^T	0.516	141.810	110.054
Scholl	2488	28	29	28∗	1	28.7^T	0.483	60.101	69.595
Scholl	2580	27	28	27∗	1	27.2^T	0.422	142.472	102.649
Scholl	2680	26	27	26∗	1	26	0	35.804	16.163
Scholl	2787	25	26	25∗	1	25	0	49.832	28.863

Note. ∗indicates that an optimal solution is found; ^{T}beside the average of solution indicates trickiness.

Table 2: Characteristics of tricky instances.

Instances	C	Sum	Mean	Var	t_{min}	t_{max}	TV	t_{min}/C	t_{max}/C	Mean/C	OS
Barthol2	85	4234	28.608	356.716	1	83	83	0.012	0.977	0.337	0.258
Lutz2	12	485	5.449	8.205	1	10	10	0.083	0.833	0.454	0.776
Lutz2	14	485	5.449	8.205	1	10	10	0.071	0.714	0.389	0.776
Lutz3	110	1644	18.472	184.525	1	74	74	0.009	0.673	0.168	0.776
Lutz3	118	1644	18.472	184.525	1	74	74	0.009	0.627	0.157	0.776
Sawyer	47	324	10.800	36.855	1	25	25	0.021	0.532	0.230	0.448
Tonge	251	3510	50.143	1505.400	1	156	156	0.004	0.622	0.200	0.200
Warnecke	60	1548	26.690	206.077	7	53	7.571	0.117	0.883	0.445	0.591
Wee-mag	47	1499	19.987	46.419	2	27	13.5	0.043	0.575	0.425	0.227
Scholl	1394	69655	234.529	38911.047	5	1386	277.2	0.004	0.994	0.168	0.582
Scholl	2247	69655	234.529	38911.047	5	1386	277.2	0.002	0.617	0.104	0.582

Note. For instances of Scholl, only the statistical information of tricky instances with the smallest and largest cycle time is reported in order to show the tendency character of the problem; "Sum" in the second column is the sum of processing times; and "t_{min}" and "t_{max}" denote the minimum and maximum task time, respectively.

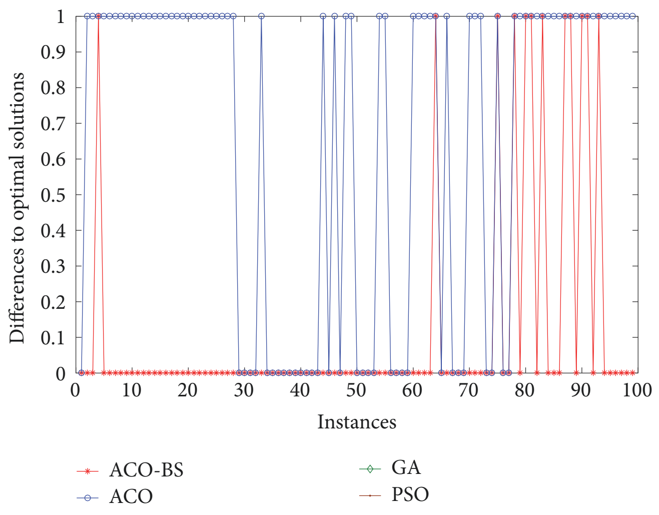

Figure 5: Comparative results between ACO and ACO-BS.

5.2. Results of Randomly Generated Instances. According to Scholl [34], the following three indicators can be used to measure the complexity of the ALBP instances:

Order Strength (OS): OS is defined as the number of arcs in the transitive closure of the precedence graph divided by $n \cdot (n-1)/2$, that is, the maximal number of arcs in an acyclic graph with n nodes. The middle values of OS seem to be harder than the low or high order strength values [35]. Nevertheless, when OS is 1 there is only one task sequence feasible; when OS is 0, SALBP-I becomes the bin packing problem, which is also NP-hard [34].

Time Variability (TV): TV is measure by t_{max}/t_{min}, which reflects the time structure of one instance. t_{max} and t_{min} denote the longest and shortest processing time, respectively. A smaller TV suggests a higher complexity.

Time Interval (TI): The interval is defined as $[t_{min}/C, t_{max}/C]$, which indicate the relation between the cycle time and the processing times. Instances with a time interval that is small and near to the right border of $[0,1]$ is expected to be relatively complicated.

Thus, we consider OS, TV, and time interval to measure the complexity of instances. We can see from Table 2 that the minimum processing time tends to be quite small (usually 1, with 5 and 7 as larger values). The OS seems to be around 0.2, 0.6, and 0.8. The smallest TV is 7.571, while the largest one can be 277.2. Additionally, the ratio of the minimum processing time to cycle time tends to be less than 0.1, and the maximum processing time ranges from 0.5 to a figure that is close to 1. The ratio of mean processing time to cycle time varies from 0.157 to 0.445. The minimum variation of processing time is 8.205; however, the largest is approximately 5000 times the minimum one.

Finally, we choose OS and TV as the main measurements for the tricky level of instances, and set three levels of OS (0.2, 0.6, and 0.9) and three levels of TV (5-15, 65-75, and 135-145). We pay attention to the minimum processing time, and give priority to the instance with a smaller minimum processing time. As we want to explore the larger scale instance, we choose the problem size to be 400. As the tricky level and the problem size level are high, we enlarge the time limit of one

Table 3

(a) Statistical description of precedence graphs of random instances.

OS level	OS	Number of stages	Number Of beginning nodes	Number Of ending nodes	Number of pairs precedence relations
0.2	0.208	40	3	2	551
0.6	0.607	40	3	4	862
0.9	0.904	50	2	7	1079

(b) Statistical description of processing times of random instances.

	Statistical information for processing times							Information for normal distributions used		
TV level	Mean	Var	t_{min}	t_{max}	TV	t_{min}/C	t_{max}/C	Type	mean	std.
5-15	253.902	5899.542	35	457	13.057	0.035	0.457	Bottom	250	80
	501.938	22247.106	63	855	13.572	0.063	0.855	central	500	150
	825.698	19925.720	69	999	14.478	0.069	0.999	bimodal	250 (750)	100 (250)
65-75	251.145	6878.485	8	539	67.375	0.008	0.539	Bottom	250	80
	502.035	21738.550	14	985	70.357	0.014	0.985	central	500	150
	816.795	23440.324	14	999	71.357	0.014	0.999	bimodal	250 (750)	100 (250)
135-145	253.125	6818.070	4	577	144.25	0.004	0.577	Bottom	250	80
	500.297	25632.059	7	978	139.714	0.007	0.978	central	500	150
	832.472	19895.062	7	999	142.714	0.007	0.999	bimodal	250 (750)	100 (250)

Note. Figures in the brackets of last two columns are means and standard deviations of one normal distribution, which is used to generate a bimodal distribution.

run from 360 CPU time seconds to 720 CPU time seconds and the width of beam increases to be 100 and the number of extensions increases to 30.

5.2.1. Generation of Random Instances. The random instances generation consists two parts: arc generation and task times generation.

Arc generation: According to Otto et al. [36], the concept of stages allows for a direct manipulation of stages characteristics. Following Otto et al. [36] and Kolisch et al. [37], we use three steps to generate precedence arcs. Firstly, the average number of tasks per stage is selected, and then the number of tasks per stage is generated following a truncated normal distribution (that is, the number is generated following a normal distribution iteratively until the number is no less than 1). Next, each beginning node (nodes have no predecessor) is assigned one successor, and each other node is assigned one predecessor. After assignments for all the nodes, one successor is chosen randomly for those having no successor. Finally, the second step is repeated until the expected complexity is reached.

During the abovementioned procedure, the following aspects should be taken into account. First, there should be redundant arcs. According to Kolisch et al. [37], let $N = (V, A)$ be a network with node set V and arc set A, and an arc (i_0, i_s) is called redundant if there are arcs $(i_0, i_1), \ldots, (i_{s-1}, i_s) \in A$ and $s \geq 2$. Second, predecessors and successors of nodes can only be chosen from the previous stage and the next stage, respectively. Last, tasks are always considered in the order of increasing order, and the added precedence relationships follow the topological rule.

Task times generation: Kilbridge and Wester [38] found that task times usually follow a unimodal or a bimodal distribution. Following Morrison et al. [35], processing times of tasks are generated randomly according to some prespecified normal distribution. Three kinds of task times are used: peak at the bottom: tasks times are drawn from a normal distribution with the mean centered around small times; peak in the middle: task times are drawn from a normal distribution with the mean of C/2; bimodal: task times are drawn from a combination of two normal distribution with means centered around small and large times.

Besides, task times are rounded to the next integer and possible rounding effects are compensated by setting the default cycle time to 1000, which is large enough to allow flexible time structures [36].

5.2.2. Results of Randomly Generated Instances. Tables 3(a) and 3(b) show the statistical description of the precedence graph and processing times of randomly generated instances, respectively. For the OS levels of 0.2 and 0.4, the number of stages is 40, while for the OS level of 0.9, the number of stages is 50. The number of stages is tuned by hand, and we find that when the number of stages is large, the number of iterations to add arcs so as to increase OS is less.

As to processing times, there are three types of distribution to generate the processing times for each TV level and the three TV values for one TV level are close in order to control the impact of TV, if impacts of OS or distribution types of processing times are expected to be examined.

Surprisingly, the solutions found by PR are the same with those found by ACO. The column of difference refers to the differences between the solutions obtained by priority rule and those obtained by ACO-BS, and this can indicate the extent to which ACO-BS improves the quality of solutions. Figure 6 shows the comparative results of ACO and ACO-BS. For 11 instances, the solutions found by ACO are the same with those by ACO-BS, while for the other 16 instances, there

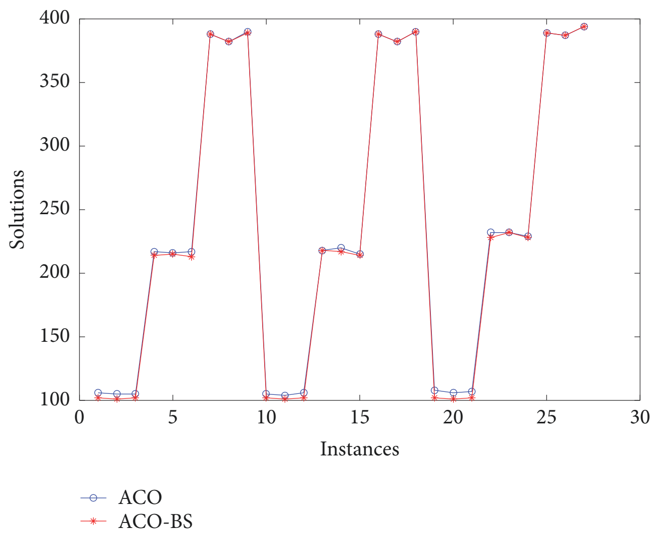

Figure 6: Comparative results between ACO and ACO-BS on 27 randomly generated instances.

are significant improvements in solution quality by ACO-BS compared with ACO.

We can see from Table 4 that the most significant tendency is that instances whose processing times follow the bimodal distribution are more difficult, consistent with Morrison et al. [35], since for all OS levels, there are 11 instances where there is no improvement after using ACO-BS with the time limit of 720 CPU time seconds, with solution quality of only one such kind of instance improved (OS level is 0.2, TV level is 135-145). Besides, there are two instances where there is no improvement on solution quality, with OS levels of 0.6 and 0.9 respectively and TV levels of 5-15 and 65-75, respectively. However, the similarity is that their processing times are generated following the distribution peak in the middle.

As to the standard deviation of the solutions obtained and improved by ACO-BS, it seems that the standard deviations for instances with processing times generated following the normal distribution (peaking at the bottom) tend to be 0. This implies that these kinds of instances are easiest to be solved.

Although the time limit to run ACO-BS is not quite large, some tendencies have already been shown. This can be useful to explore the characteristics of large scale instances.

6. Conclusions

A method based on the priority rule is used at first to generate the first best so far solution. After using this method once on the original problem and the reverse problem respectively, 63.20% of the total benchmark instances can reach the optimal results. Based on the best so far solution obtained by priority rule, ACO-BS searches for larger solution space in order to reach more optimal results. After ten runs (360 CPU time seconds for each run), 95.54% of the total benchmark instances can reach the optimal results. What is more, these results are better when increases in the width of the beam or the number of extensions are allowed, or by increasing the time limit for one run. We can conclude that the algorithm of ACO-BS is good in solving SALBP-I.

Premature convergence is a challenging problem for ACO, and several strategies are used to deal with this problem. First, the pheromone values are restricted to the interval of [0.01, 0.99] to prevent stagnation, so that there will not be too large pheromone values that some tasks tend to be assigned to the same workstation, and there will not be too small pheromone values that some tasks tend to avoid being assigned to a workstation. Thus, stagnation can be prevented [32]. Also, there is an evaporation process when update the pheromone values, so it is discouraged to assign one task to the same position. Second, when choosing task from the available task set, the probability that each task in the set is chosen will be calculated. But with equal probability, one task will be chosen from the set by maximizing the probability, or by the roulette-wheel method. Thus, in this way, tasks with a higher probability have a larger chance to be selected, but tasks with a lower probability still have chances to be selected. Third, the convergence value is calculated in every iteration. Since the pheromone values are initialized to be 0.5, the convergence value is 1 at the beginning. When all the pheromone values are close to 0.99 or 0.01, the convergence value will be close to zero. The pheromone values will be reinitialized to be 0.5 when the convergence value is less than 0.05, and this will prevent the stagnation [33]. Therefore, the strategies above always try to search for alternative solutions rather than staying at the stagnation state. The good results of ACO-BS compared with the results of ACO, GA, and PSO also demonstrate that the algorithm has the ability of jump out of local optimum and prevent premature convergence.

With the development of the manufacturing industry and the transformation from mass production to customization, assembly of complex products within an acceptable period has become urgent. Thus, we are concerned more about large

Table 4: Results of randomly generated instances.

OS level	TV level	Type	PR/ACO	ACO-BS	Difference	Solution		Running time [s]	
						avg.	std.	avg.	std.
0.2	5-15	bottom	106	102	4	102	0	466.411	10.825
	65-75	bottom	105	101	4	101	0	569.084	2.131
	135-145	bottom	105	102	3	102	0	559.794	6.917
	5-15	middle	217	214	3	215	1	466.249	318.705
	65-75	middle	216	215	1	215.4	0.548	369.356	155.936
	135-145	middle	217	213	4	213.8	1.304	305.772	106.824
	5-15	bimodal	388	388	0	388	0	14.027	0.055
	65-75	bimodal	382	382	0	382	0	13.947	0.067
	135-145	bimodal	390	389	1	389.6	0.548	1048.546	946.530
0.6	5-15	bottom	105	102	3	102	0	535.052	2.433
	65-75	bottom	104	101	3	101	0	546.670	5.939
	135-145	bottom	106	102	4	102	0	535.687	4.069
	5-15	middle	218	218	0	218	0	111.478	133.562
	65-75	middle	220	217	3	217.6	0.548	435.175	135.772
	135-145	middle	215	214	1	214.8	0.447	452.758	102.803
	5-15	bimodal	388	388	0	388	0	12.368	0.929
	65-75	bimodal	382	382	0	382	0	13.208	1.178
	135-145	bimodal	390	390	0	390	0	12.976	1.496
0.9	5-15	bottom	108	102	6	102.8	0.447	543.545	236.263
	65-75	bottom	106	101	5	101	0	482.360	115.202
	135-145	bottom	107	102	5	102	0	275.540	1.023
	5-15	middle	232	228	4	230.2	1.643	558.983	261.882
	65-75	middle	232	232	0	232	0	14.075	0.045
	135-145	middle	229	228	1	228.8	0.447	264.277	348.637
	5-15	bimodal	389	389	0	389	0	14.141	0.0592
	65-75	bimodal	387	387	0	387	0	1343.375	743.422
	135-145	bimodal	394	394	0	394	0	13.1387	1.227

scale ALBP in this study. In order to further examine the performance of the algorithm in more complicated instances, we generate large scale SALBP-I instances randomly and explore solutions for them with ACO-BS. OS and TV are chosen to measure the complexity of the random instances. Compared with solutions obtained by priority rule, there are significant improvements in the quality of the best solutions after applying ACO-BS, which shows that ACO-BS is efficient for small scale instances as well as large scale instances. Therefore, ACO-BS is a promising tool for solving SALBP-I, especially for those of complex products.

We browse further improvements, based on the current ACO-BS, since there are a number of extensions for the assignment of the workstation considered and the algorithm can be greatly improved by parallel computing. When taking advantage of this parallel feature, the algorithm will be able to perform better.

Data Availability

All the benchmark instances (SALBP-I) used in this study can be found on https://assembly-line-balancing.de/.

Conflicts of Interest

The authors declare that there are no conflicts of interest regarding the publication of this paper.

Acknowledgments

The work described in this paper was supported by a grant from the Natural Science Foundation of China (Grant no. 71471158). The authors also would like to thank the Hong Kong Polytechnic University Research Committee for financial and technical support and a grant under student account code RURR.

References

[1] Y.-G. Zhong and B. Ai, “A modified ant colony optimization algorithm for multi-objective assembly line balancing,” *Soft Computing*, vol. 21, no. 22, pp. 6881–6894, 2017.

[2] A. Scholl and C. Becker, “State-of-the-art exact and heuristic solution procedures for simple assembly line balancing,” *European Journal of Operational Research*, vol. 168, no. 3, pp. 666–693, 2006.

[3] O. Battaïa and A. Dolgui, "A taxonomy of line balancing problems and their solutionapproaches," *International Journal of Production Economics*, vol. 142, no. 2, pp. 259–277, 2013.

[4] G. Q. Huang, Y. F. Zhang, X. Chen, and S. T. Newman, "RFID-enabled real-time wireless manufacturing for adaptive assembly planning and control," *Journal of Intelligent Manufacturing*, vol. 19, no. 6, pp. 701–713, 2008.

[5] M. E. Salveson, "The assembly line balancing problem," *Journal of Industrial Engineering*, vol. 6, pp. 18–25, 1955.

[6] T. S. Wee and M. J. Magazine, "Assembly line balancing as generalized bin packing," *Operations Research Letters*, vol. 1, no. 2, pp. 56–58, 1982.

[7] E. Celik, Y. Kara, and Y. Atasagun, "A new approach for rebalancing of U-lines with stochastic task times using ant colony optimisation algorithm," *International Journal of Production Research*, vol. 52, no. 24, pp. 7262–7275, 2014.

[8] C. Becker and A. Scholl, "A survey on problems and methods in generalized assembly line balancing," *European Journal of Operational Research*, vol. 168, no. 3, pp. 694–715, 2006.

[9] I. Baybars, "A survey of exact algorithms for the simple assembly line balancing problem," *Management science*, vol. 32, no. 8, pp. 909–932, 1986.

[10] M. Dorigo and L. M. Gambardella, "Ant colony system: a cooperative learning approach to the traveling salesman problem," *IEEE Transactions on Evolutionary Computation*, vol. 1, no. 1, pp. 53–66, 1997.

[11] A. S. Simaria and P. M. Vilarinho, "2-ANTBAL: an ant colony optimisation algorithm for balancing two-sided assembly lines," *Computers & Industrial Engineering*, vol. 56, no. 2, pp. 489–506, 2009.

[12] P. R. McMullen and P. Tarasewich, "Using ant techniques to solve the assembly line balancing problem," *Institute of Industrial Engineers*, vol. 35, no. 7, pp. 605–617, 2003.

[13] J. Bautista and J. Pereira, "Ant algorithms for a time and space constrained assembly line balancing problem," *European Journal of Operational Research*, vol. 177, no. 3, pp. 2016–2032, 2007.

[14] I. Kucukkoc and D. Z. Zhang, "Type-E parallel two-sided assembly line balancing problem: Mathematical model and ant colony optimisation based approach with optimised parameters," *Computers & Industrial Engineering*, vol. 84, pp. 56–69, 2015.

[15] A. Baykasoğlu and T. Dereli, "Simple and u-type assembly line balancing by using an ant colony based algorithm," *Mathematical and Computational Applications*, vol. 14, no. 1, pp. 1–12, 2009.

[16] P. Fattahi, A. Roshani, and A. Roshani, "A mathematical model and ant colony algorithm for multi-manned assembly line balancing problem," *The International Journal of Advanced Manufacturing Technology*, vol. 53, no. 1–4, pp. 363–378, 2011.

[17] S. Akpınar, G. M. Bayhan, and A. Baykasoglu, "Hybridizing ant colony optimization via genetic algorithm for mixed-model assembly line balancing problem with sequence dependent setup times between tasks," *Applied Soft Computing*, vol. 13, no. 1, pp. 574–589, 2013.

[18] C. Lu and Z. Yang, "Integrated assembly sequence planning and assembly line balancing with ant colony optimization approach," *The International Journal of Advanced Manufacturing Technology*, vol. 83, no. 1-4, pp. 243–256, 2016.

[19] P. R. McMullen and P. Tarasewich, "Multi-objective assembly line balancing via a modified ant colony optimization technique," *International Journal of Production Research*, vol. 44, no. 1, pp. 27–42, 2006.

[20] J. Zha and J.-J. Yu, "A hybrid ant colony algorithm for U-line balancing and rebalancing in just-in-time production environment," *Journal of Manufacturing Systems*, vol. 33, no. 1, pp. 93–102, 2014.

[21] S. Agrawal and M. K. Tiwari, "A collaborative ant colony algorithm to stochastic mixed-model U-shaped disassembly line balancing and sequencing problem," *International Journal of Production Research*, vol. 46, no. 6, pp. 1405–1429, 2008.

[22] L. Ozbakir, A. Baykasoglu, B. Gorkemli, and L. Gorkemli, "Multiple-colony ant algorithm for parallel assembly line balancing problem," *Applied Soft Computing*, vol. 11, no. 3, pp. 3186–3198, 2011.

[23] J. Bautista, C. Batalla-García, and R. Alfaro-Pozo, "Models for assembly line balancing by temporal, spatial and ergonomic risk attributes," *European Journal of Operational Research*, vol. 251, pp. 814–829, 2016.

[24] A. Scholl, M. Fliedner, and N. Boysen, "Absalom: Balancing assembly lines with assignment restrictions," *European Journal of Operational Research*, vol. 200, no. 3, pp. 688–701, 2010.

[25] O. Battaïa and A. Dolgui, "Reduction approaches for a generalized line balancing problem," *Computers & Operations Research*, vol. 39, no. 10, pp. 2337–2345, 2012.

[26] Y. Leu, L. A. Matheson, and L. P. Rees, "Assembly Line Balancing Using Genetic Algorithms with Heuristic-Generated Initial Populations and Multiple Evaluation Criteria," *Decision Sciences*, vol. 25, no. 4, pp. 581–605, 1994.

[27] J. Dou, J. Li, and C. Su, "A novel feasible task sequence-oriented discrete particle swarm algorithm for simple assembly line balancing problem of type 1," *The International Journal of Advanced Manufacturing Technology*, vol. 69, no. 9-12, pp. 2445–2457, 2013.

[28] M. Dorigo, V. Maniezzo, and A. Colorni, "Ant system: optimization by a colony of cooperating agents," *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, vol. 26, no. 1, pp. 29–41, 1996.

[29] I. Sabuncuoglu and M. Bayiz, "Job shop scheduling with beam search," *European Journal of Operational Research*, vol. 118, no. 2, pp. 390–412, 1999.

[30] C. Blum, "Beam-ACO for simple assembly line balancing," *INFORMS Journal on Computing*, vol. 20, no. 4, pp. 618–627, 2008.

[31] D. Merckle and M. Middendorf, "An ant algorithm with a new pheromone evaluation rule for total tardiness problems," in *Real-World Applications of Evolutionary Computing: Proceedings of the EvoWorkshops 2000: EvoIASP, EvoSCONDI, EvoTel, EvoSTIM, EvoRob, and EvoFlight Edinburgh, Scotland, UK, April 17, 2000*, vol. 1803 of *Lecture Notes in Computer Science*, pp. 290–299, Springer, Berlin, Germany, 2000.

[32] M. López-Ibáñez, T. Stützle, and M. Dorigo, "Ant colony optimization: A component-wise overview," in *Handbook of Heuristics*, pp. 1–37, 2016.

[33] M. Kong, P. Tian, and Y. Kao, "A new ant colony optimization algorithm for the multidimensional knapsack problem," *Computers & Operations Research*, vol. 35, no. 8, pp. 2672–2683, 2008.

[34] A. Scholl, "Data of assembly line balancing problems," *Schriften zur Quantitativen Betriebswirtschaftslehre, 16/, TH Darmstadt*, 1993.

[35] D. R. Morrison, E. C. Sewell, and S. H. Jacobson, "An application of the branch, bound, and remember algorithm to a new simple assembly line balancing dataset," *European Journal of Operational Research*, vol. 236, no. 2, pp. 403–409, 2014.

[36] A. Otto, C. Otto, and A. Scholl, "Systematic data generation and test design for solution algorithms on the example of SALBPGen for assembly line balancing," *European Journal of Operational Research*, vol. 228, no. 1, pp. 33–45, 2013.

[37] R. Kolisch, A. Sprecher, and A. Drexl, "Characterization and Generation of a General Class of Resource-Constrained Project Scheduling Problems," *Management Science*, vol. 41, no. 10, pp. 1693–1703, 1995.

[38] M. Kilbridge and L. Wester, "The Balance Delay Problem," *Management Science*, vol. 8, no. 1, pp. 69–84, 1961.