Proceedings of the 4th World Congress on Mechanical, Chemical, and Material Engineering (MCM'18)
Madrid, Spain – August 16 – 18, 2018
Paper No. ICMIE 102
DOI: 10.11159/icmie18.102

Hybridizing Ant Colony Optimization by Beam Search for the Assembly Line Balancing Problem

Jiage Huo[1], Felix T.S. Chan[1], Carman K. M. Lee[1], Jan Ola Strandhagen[2], Ben Niu[3]
[1]Department of Industrial and Systems Engineering, the Hong Kong Polytechnic University
Hung Hom, Kowloon, Hong Kong, China
jiage.huo@connect.polyu.hk; f.chan@polyu.edu.hk; ckm.lee@polyu.edu.hk
[2]Department of Production and Quality Engineering, Norwegian University of Science and Technology
NO-7491, Trondheim, Norway
ola.strandhagen@ntnu.no
[3]College of Management, Shenzhen University
Nanshan, Shenzhen, China
drniuben@163.com

Abstract **-** The type-I Assembly Line Balancing Problem (ALBP) focuses on the task assignment process with the objective of minimizing the number of workstations for a given cycle time. With the development of complex products, the problem size and the complexity in the assembly process is increasing. In this study, we hybridize the ant colony optimization algorithm via beam search (ACO-BS) in order to solve the type-I ALBP, and we focus more on the large scale ALBP in order to suit to the industrial requirements. We test ACO-BS with benchmark instances with a time limit of 360 seconds for one run, and the results show that 95.54% of the problems can reach their optimal solutions. In addition, since we want to explore the large scale ALBP, we generate 27 instances with a total of 400 tasks (the largest number of tasks in the benchmark instances of type-I ALBP is 297) randomly basing on the complexity indicators of order strength and processing time variation. There are three levels of order strength, 0.2, 0.6 and 0.9, and the time variation is set to be at 5-15, 65-75 and 135-145 levels. Meanwhile, the processing times of the tasks usually follow a unimodal or bimodal distribution, and we generate task times to follow three kinds of distribution respectively, unimodal distribution peaking at the bottom, unimodal distribution peaking in the middle and bimodal distribution. The comparison results with solutions obtained by the priority rule demonstrate the superiority of ACO-BS in solving large scale ALBP.

Keywords**:** Assembly Line Balancing, Beam Search, Ant Colony Algorithm, Benchmarking Data Set, Computational Efficiency.

1. Introduction

An assembly line is a continuous production line consisting of materials and workstations combined with conveyor belts, linking 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 of assembly lines requires high investment and running costs[3], and assembly planning and control play important roles in reducing the delivery time and costs and in increasing profitability [4].

The Assembly Line Balancing Problem (ALBP) is a well studied classic problem, and can be seen as a generalization of the bin packing problem where precedence constraints are added [5]. 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 the performance measures [6]. According to Becker and Scholl [7], there are four types of ALBP: SALBP-I aims to minimize the number of workstations with a given fixed cycle time; SALBP-II minimizes the cycle time with a given number of workstations; SALBP-E aims to minimize 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 the given number of workstations and the cycle time.

Exact methods and approximate methods have been used to solve ALBP. The required computational time for obtaining an optimal solution with an exact method for most cases 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 optimization and the ant colony optimization (ACO) algorithm have been used to deal with ALBP due to these algorithms having good performance in optimization [1]. Many researchers explored ALBP with ant colony based approaches, and found ACO has good performance in solving combinatorial optimization problems [1, 8]. For example, to effectively address ALBP with complicating factors such as parallel workstations, stochastic task times and mixed-models, McMullen and Tarasewich [8] proposed an approach based on ant techniques and, in comparison with other heuristics, showed that the proposed method is competitive in terms of the performance measures used in the study.

With the development of the manufacturing industry and the transformation from mass production to customization, the complexity of the assembly process is increasing, and ALBP of complex products within an acceptable time period becomes critical. Although much exploration has been done by researchers, the exploration of methods suiting to the complex assembly context is critical, with the increasing 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 hybridize the ACO with Beam Search (ACO-BS) to improve the efficiency and improve the computational performance of the algorithm so that satisfactory results can be achieved within an acceptable computation time.

2. Problem Description

n : total number of tasks;

UB: upper bound of the total number of workstations;

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

t_i : processing time of task i , for $i = 1,\mathrm{K}\ ,n$;

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 $i = 1,\mathrm{K}\ ,n$;

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

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

Variables:

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

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

The mathematical model of SALBP-I is as follows:

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

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

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

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

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

The objective function (1) minimizes 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.

One ALBP can transfer to its reverse version after all the precedence relationships are reversed. If $S_{rev} = \{S_1, \mathrm{K}, S_m\}$ is a solution for the reverse problem, then a solution for the original problem can be $S = \{S_m, \mathrm{K}, S_1\}$. Following Bautista and Pereira [9], we solve the original problem and the reverse problem respectively, and then two criteria are used for selecting the best solution after solution transformation of the reverse problems: (1) number of workstations; (2) idle time in the last workstation. The second criterion is added due the fact that there are large plateaus when only the first criterion is used, and more idle time in the last workstation means better resource utilization for the previous workstations.

3. The Algorithm of ACO-BS

The general logic of the ACO-BS algorithm is as follows: At first, a better solution is chosen after solving the original problem and the reverse problem respectively by the priority rule, and it is used to initialize the best-so-far solution (S_{bsf}).

3.1. Priority Rule

Before assigning tasks, the priority values of the tasks are computed as follows:

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

Starting from the first workstation, put all the tasks not assigned in a set N and set idle time to be C. Also, put tasks with no predecessor into a set N_{nopre}. The assignment is implemented as follows: (1) determine 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 remained of a workstation means better resource utilization. If N_{av} is empty, tasks with no predecessor and processing time less than the idle time are put into set N_{av}. (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 highest priority value, choose one randomly), and go to step (3); If the set is empty, go to step (4). (3) delete the chosen task from N and N_{nopre}, $N_{av} = \phi$ and the idle time will decrease by the processing time of the latest assigned task. Go to step (1). (4) close the current workstation. If N is not empty, open a new one and set the cycle time to be C, then go to step (1); if N is empty, end the procedure.

3.2. ACO-BS

The first step is used to initialized the parameters, and steps 2 to 4 are repeated within a certain time period.

Step 1: Initialization

Generate one solution by using the priority rule for the original and reverse problem. Then the two criteria are used to choose a better solution 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} $(i = 1,\mathrm{K}\ ,UB; j = 1,\mathrm{K}\ n)$ 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 ACO-BS

Unlike the priority rule which is used for task selection, the selection rule here makes use of the pheromone values and priority values of the tasks. Differing from the computation of the priority values used in the priority rule, the priority values used in ACO-BS are processed as follows [10], after computing using equation (6):

$$\eta'_j = \frac{\eta_j - \eta_{\min} + 1}{\eta_{\max}},\ j = 1,\mathrm{K}\ ,n \tag{7}$$

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

When choosing tasks from the set N_{av}, the probability p_j that a task is chosen by workstation k is calculated using equation (8) by the summation rule as follows [11]. Choose a task by maximizing p_j or by roulette-wheel method which is determined with the same possibility.

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

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, and then the next steps are given.

At first, task assignment for the first workstation is explored n_{ext} times, and the procedure is similar to by the priority rule, but the 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 $|S_{bsf}|$, are put into S_{par} and S_{ext}.

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 by one, and then the following steps are repeated until extending for n_{ext} times (let m denotes the workstation which is currently considered; $S_{ext} = \phi$): (1) $ext = 1$; $t_m = \phi$ stores the task set for workstation m. (2) Implement task assignment for workstation m, and get the task set t_m for the workstation. (3) Extend the partial solution considered by the task set t_m for workstation m. If the extended solution is a complete solution, go to step 4, else go to step (5). (4) Put the extended partial solution to set S_{com} which stores the complete solution. (5) If the lower bound (described below) of the workstation needed for the partial solution is less than $|S_{bsf}|$ and t_m is different from all the factors in S_{ext}, the partial solution will be put into S_{par}. (6) If $ext = n_{ext}$, end this procedure; else $\text{ext} = ext + 1$ and $t_m = \phi$, and go to step (2).

When choosing extensions after filling one workstation, two criteria are used. First, let N_{rem} be the set of tasks not assigned for one partial solution, and the lower bound on the workstations needed is computed 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 $\min\left\{B_{wid}, \left|S_{par}\right|\right\}$ generated, and B_{wid} is the width of beam.

This step is ended when 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 the pheromone values

Since in step 2, if the lower bound of a partial solution is no less than $\left|S_{bsf}\right|$, the partial solution is aborted. Thus there may be no solution obtained in step 2, and then the best-so-far solution is used to update the pheromone values.

If the solution set obtained in step 2 is not empty, a iteration best solution S_{ib} is chosen with the two criteria. S_{ib} is then used to update the pheromone values. Pheromone values τ_{ij} between the task j and workstation i ($i = 1,\mathrm{K}, \left|S_{ib}\right|$; $j = 1,\mathrm{K}, n$) are updated. There are two updating processes: (1) pheromone evaporation: for each τ_{ij} to be updated, 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 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, and $\tau_{\min} = 0.01$ $\tau_{\max} = 0.99$. If a pheromone value is larger than $\tau_{\max}$ after updating, it is set to be $\tau_{\max}$; if the value is smaller than $\tau_{\min}$ after updating, it is set to be $\tau_{\min}$.

If the iteration best solution is better than the best-so-far solution, the latter is updated by the former.

Step 4: Calculating the convergence factor

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 convergence value is less than 0.05. The convergence value is calculated as follows [10]:

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

4. 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 are given in CPU seconds.

4.1. Results of Benchmark Instances of SALBP-I

In order to exhibit the superior performance of ACO-BS, we tested ACO-BS with benchmark instances published on https://assembly-line-balancing.de/. There are 269 benchmark instances in SALBP-I. By using the priority rule only, optimal solutions can be obtained for 170 instances (with 99 instances whose optimal solutions cannot be reached). After ten runs of

ACO-BS ($n_{ext} = 10, B_{wid} = 20$) with a time limit of 360 seconds, there are 87 more instances whose optimal solutions can be obtained. We can see from figure 1 that the values of solutions obtained by ACO-BS equal those of the optimal solutions (see 87 positions where '☆' are on the line), except for 12 instances (12 positions where the '☆' are above the line, with one isolated and 11 overlapping with '*').

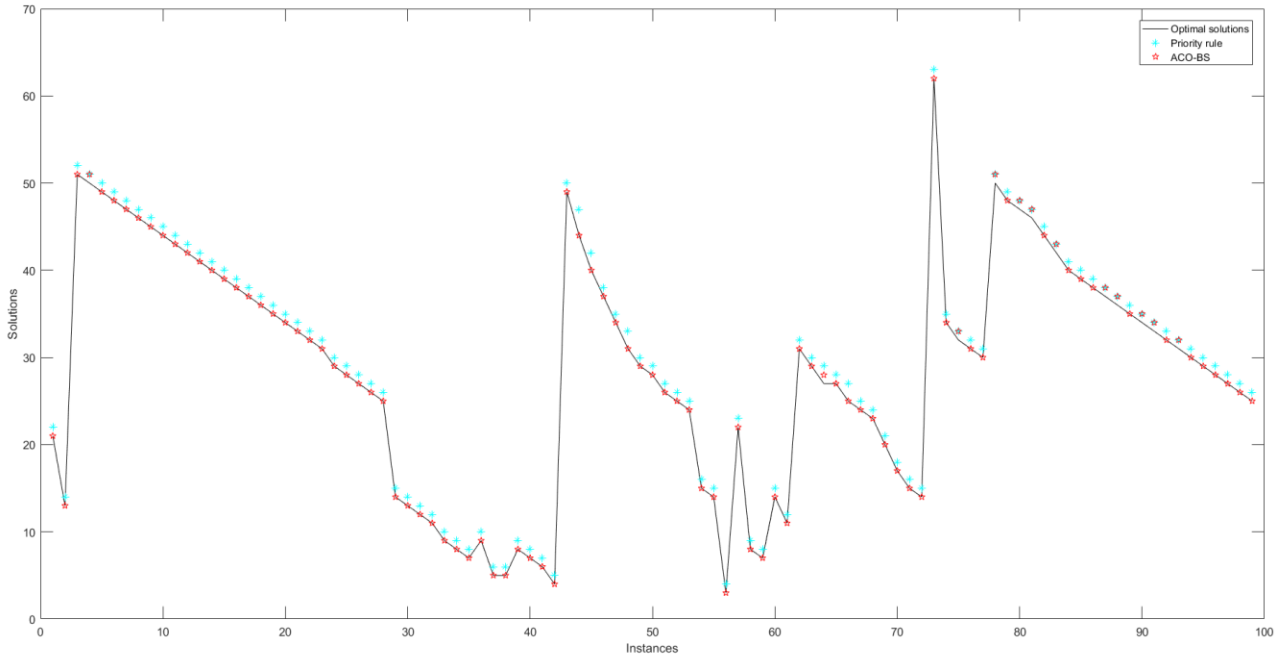

Fig. 1: Comparison of optimal solutions, solutions obtained by priority rule and solutions obtained by ACO-BS in 99 instances.

4.2. Generation and Results of Randomly Generated Instances

According to Scholl [12], the following three indicators can be used to measure the complexity of the ALBP instances: (1) Order Strength (OS). OS is defined as the number of arcs in the transitive closure of the precedence graph divided by $\left[n \cdot (n-1)\right]/2$, 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 [13]. But when OS is 1 there is only one task sequence feasible; when OS is 0, SALBP-I reverts to the bin packing problem which is also NP-hard [12]. (2) Time Variability (TV). TV is measured by $t_{\max}/t_{\min}$, which reflects the time structure of one instance. A smaller TV suggests a higher complexity. We set three levels of OS (0.2, 0.6, 0.9) and three levels of TV (5-15, 65-75, 135-145). The problem size is set to be 400. We enlarge the time limit of one run to 720 seconds, and the width of beam and number of extensions increased to be 100 and 30.

The random instances generation consists two parts: arc generation and task times generation.

Generation of arcs. According to Otto et al. [14], the concept of stages allows for a direct manipulation of the graph characteristics. Following Otto et al. [14] and Kolisch et al. [15], 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 normal distribution. Next, each beginning node (with 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 above-mentioned procedure, the following aspects should be taken into account: First, there may be redundant arcs which should be deleted. According to Kolisch et al. [15], 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), ..., $(i_{s-1}, i_s) \in A$ 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 increasing order, and the added precedence relationships follow the topological rule.

Generation of tasks times. Kilbridge and Wester [16] found that task times usually follow a unimodal or bimodal distribution. Following Morrison et al. [13], three kinds of task times are generated: (1) peak at the bottom: tasks times are drawn from a normal distribution with the mean centered around small times; (2) peak in the middle: task times are drawn from a normal distribution with a mean of C/2; (3) bimodal: task times are drawn from a combination of two normal distribution with means centered around small and large times. Task times are rounded to the next integer and possible rounding effects are compensated for by setting the default cycle time to 1000 to allow flexible time structures [14].

The OS values for three levels are 0.208, 0.607 and 0.898. Table 1 shows information of task times of randomly generated instances. The figures in brackets are the means and standard deviations to generate a bimodal distribution.

Table 1: Statistical description of task times of random instances.

Statistical information for processing times						Information for normal distributions		
TV level	Mean	Var	t_{min}	t_{max}	TV	Distribution	Mean	Std.
5-15	253.902	5899.542	35	457	13.057	bottom	250	80
	501.938	22247.106	63	855	13.572	central	500	150
	825.698	19925.720	69	999	14.478	bimodal	250 (750)	100 (250)
65-75	251.145	6878.485	8	539	67.375	bottom	250	80
	502.035	21738.550	14	985	70.357	central	500	150
	816.795	23440.324	14	999	71.357	bimodal	250 (750)	100 (250)
135-145	253.125	6818.070	4	577	144.250	bottom	250	80
	500.297	25632.059	7	978	139.714	central	500	150
	832.472	19895.062	7	999	142.714	bimodal	250 (750)	100 (250)

The column of difference in table 2 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. We can see from table 2 that the most significant tendency is that instances whose processing times follow the bimodal distribution are more difficult, consistent with Morrison et al. [13], since for all OS levels, there are 11 instances where there is no improvement after using ACO-BS with the time limit of 720 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.

Table 2: Results of randomly generated instances.

OS level	TV level	Distribution	Priority rule	ACO-BS	Difference	Solution		Running time	
						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	0.059
	65-75	bimodal	387	387	0	387	0	1343.375	743.422
	135-145	bimodal	394	394	0	394	0	13.138	1.227

5. 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 the priority rule, ACO-BS searches for a larger solution space in order to reach more optimal results. Since all the constraints of ALBP have been satisfied during the solution searching process and then the best solution is selected, the accuracy of this method can be guaranteed. After ten runs (360 seconds limit 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. 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 the solutions obtained by the 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 shown to be an effective tool for solving SALBP-I of complex products.

Acknowledgements

The work described in this paper was supported by grants from The Natural Science Foundation of China (Grant No. 71471158); The Hong Kong Polytechnic University (Project No. 4-BCCM); and The Hong Kong Polytechnic University under student account code RURR. The authors also would like to thank The Hong Kong Polytechnic University Research Committee for financial and technical support.

References

[1] Y. G. Zhong and B. Ai, "A modified ant colony optimization algorithm for multi-objective assembly line balancing," *Soft Comput*., vol. 21, pp. 6881-6894, 2017.

[2] A. Scholl and C. Becker, "State-of-the-art exact and heuristic solution procedures for simple assembly line balancing," *Eur. J. Oper. Res.*, vol. 168, pp. 666-693, 2006.

[3] O. Battaïa and A. Dolgui, "A taxonomy of line balancing problems and their solutionapproaches," *Int. J. Prod. Econ*., vol. 142, pp. 259-277, 2013.

[4] G. Q. Huang, Y. Zhang, X. Chen, and S. T. Newman, "RFID-enabled real-time wireless manufacturing for adaptive assembly planning and control," *J. Intell. Manuf.*, vol. 19, pp. 701-713, 2008.

[5] T. Wee and M. J. Magazine, "Assembly line balancing as generalized bin packing," *Oper. Res. Lett.*, vol. 1, pp. 56-58, 1982.

[6] E. Celik, Y. Kara, and Y. Atasagun, "A new approach for rebalancing of U-lines with stochastic task times using ant colony optimisation algorithm," *Int. J. Prod. Res*., vol. 52, pp. 7262-7275, 2014.

[7] C. Becker and A. Scholl, "A survey on problems and methods in generalized assembly line balancing*," Eur. J. Oper. Res*., vol. 168, pp. 694-715, 2006.

[8] P. R. McMullen and P. Tarasewich, "Using ant techniques to solve the assembly line balancing problem," *IIE Trans., vol. 35, pp. 605-617, 2003.*

[9] J. Bautista and J. Pereira, "Ant algorithms for a time and space constrained assembly line balancing problem," *Eur. J. Oper. Res*., vol. 177, pp. 2016-2032, 2007.

[10] C. Blum, "Beam-ACO for simple assembly line balancing," *INFORMS J. Comput*., vol. 20, pp. 618-627, 2008.

[11] D. Merkle and M. Middendorf, "An ant algorithm with a new pheromone evaluation rule for total tardiness problems," in *Workshops on Real-World Applications of Evolutionary Computation*, 2000, pp. 290-299.
[12] A. Scholl, *Data of assembly line balancing problems.* Techn. Hochsch., Inst. für Betriebswirtschaftslehre, 1995.
[13] 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," *Eur. J. Oper. Res*., vol. 236, pp. 403-409, 2014.
[14] 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," *Eur. J. Oper. Res*., vol. 228, pp. 33-45, 2013.
[15] R. Kolisch, A. Sprecher, and A. Drexl, "Characterization and generation of a general class of resource-constrained project scheduling problems," *Manage. Sci*., vol. 41, pp. 1693-1703, 1995.
[16] M. Kilbridge and L. Wester, "The balance delay problem," *Manage. Sci.*, vol. 8, pp. 69-84, 1961.