Int J Adv Manuf Technol (2008) 36:582–588
DOI 10.1007/s00170-006-0861-3

ORIGINAL ARTICLE

Two-sided assembly line balancing using an ant-colony-based heuristic

Adil Baykasoglu · Türkay Dereli

Received: 23 June 2006 / Accepted: 24 October 2006 / Published online: 25 November 2006

Abstract Two-sided assembly line balancing (ALB) problems usually occur in plants which are producing large-sized high-volume products, such as buses, trucks, and domestic products. Many algorithms and heuristics have been proposed to balance the well known classical one-sided assembly lines. However, little attention has been paid to solve two-sided ALB problems. Moreover, according to our best knowledge, there is no published work in the literature on two-sided ALB problems with zoning constraints (2sALBz). In this study, an ant-colony-based heuristic algorithm is proposed for solving 2sALBz problems. This paper also makes one of the first attempts to show how an ant colony heuristic (ACH) can be applied to solve 2sALBz problems. In the paper, example applications are presented and computational experiments are performed to present the suitability of the ACH to solve 2sALBz problems. Promising results are obtained from the solution of several test problems.

Keywords Assembly line balancing · Two-sided assembly lines · Ant colony optimization

1 Introduction

Assembly lines can be categorized into one-sided and two-sided lines. Two-sided assembly lines use both (left and right) sides of the line in parallel. Two-sided assembly line balancing (ALB) problems usually occurs in plants which are producing large-sized high-volume products, such as buses, trucks, and domestic products. The two-sided assembly line has several features that are distinguished from those considered in traditional (one-side) line balancing problems:

- Large-sized products that require assembly operations on both sides of the products during assembly.
- Several facilities of the line are dedicated to certain tasks, due to specific requirements, such as size of the part, type of operation, etc. These types of restrictions are known as zoning or positional constraints, which are very common in two-sided assembly lines.
- Thirdly, once a line is designed and installed, the length of the line or the maximum number of workstations does not change during operation. Most of the traditional methods, however, presume that the line length can be varied at any time. Even if this presumption can be valid at the design time of the line, it may no longer be acceptable at operational time.

A considerable amount of literature is available on solving traditional ALB problems. Traditional ALB problems are known as NP-hard combinatorial optimization problems [1]. This is the main reason for the previous researches, including Talbot et al. [2], Ghosh and Gagnon [3], and Baykasoglu [4], in developing heuristic solution approaches. Some authors, including Arcus [5], Johnson [6], and Gunther et al. [7], also considered positional constraints. However, these approaches have invariably dealt with one-sided ALB problems. Although two-sided ALB problems are often encountered in the real world, little attention has been paid to them. Moreover, as discussed by Falkenauer [8], in most of the ALB studies, real-world situations are not truly taken into account.

Bartholdi [9] is the first researcher who addressed the two-sided ALB problems. He suggested a simple assignment rule, and a major focus is placed on the development and use of an interactive program assisting humans in building

A. Baykasoglu (✉) · T. Dereli
Department of Industrial Engineering, University of Gaziantep, 27310 Gaziantep, Turkey
e-mail: baykasoglu@gantep.edu.tr

solutions quickly and incrementally. After Bartholdi [9], Kim et al. [10] developed a genetic algorithm to solve two-sided ALB problems. Lee et al. [11] generated an assignment procedure for two-sided ALB problems, in order to maximize work relatedness and slackness. One of last papers which also considers the two-sided ALB problem is presented by Lapierre and Ruiz [12]. In Lapierre and Ruiz [12], an enhanced priority-based heuristic is developed to solve a specific two-sided ALB problem.

Different assembly tasks are carried out on the same product in parallel at both sides of a two-sided assembly line. A two-sided assembly line is illustrated in Fig. 1. A pair of two directly facing workstations, i.e., 1 and 2, is called a "mated station", and one of them calls the other a "companion". A two-sided assembly line, in practice, can offer a number of advantages over a one-sided assembly line. These include shorter line length, reduced throughput time, lower cost of tools and fixtures, and less material handling [9].

A two-sided assembly line is different to the traditional one-sided assembly lines. In a two-sided assembly line, tasks have restrictions on their operation directions. In a two-sided assembly line, some tasks can be performed on both sides of the assembly line, but some tasks can only be performed on one side of line (right or left). Therefore, assembly tasks are grouped into three types: L (left), R (right), and E (either). For example, in a car assembly line, such tasks as installing fuel tanks, air filters, and toolboxes are L-type tasks because these can be more easily performed from the left-hand side of the line; meanwhile, mounting batteries, air tanks, and mufflers are usually carried out from the right-hand side and, thus, are R-type tasks. E-type tasks include assembling axles, propeller shafts, and radiators, which do not have any preferred operation directions [10]. Similar examples can be seen in assembling large refrigerators, locomotives, etc.

In a two-sided assembly line, interference between tasks that are performed on the opposite sides of the line has to be taken into account. This is mainly because tasks on the opposite side can interfere with each other through precedence constraints, which might cause idle time (in cases where a workstation needs to wait for a predecessor task to be performed at the opposite side of the line). Therefore, balancing the line needs to take into account the sequence-dependent completion time of tasks, unlike a one-sided assembly line [10].

According to our best knowledge, there is no published work in the literature on two-sided ALB problems with zoning constraints (2sALBz). In this study, an ant colony optimization (ACO) [13] based heuristic algorithm is proposed for solving 2sALBz problems. This paper also makes one of the first attempts to show how an ant colony heuristic (ACH) can be applied to solve 2sALBz problems.

2 Two-sided assembly line balancing problems with zoning constraints (2sALBz)

In this study, a two-sided ALB problem is considered with zoning (positional) constraints. The objective is to minimize the number of workstations and, where possible, maximize the work relatedness. Work relatedness is an index which represents the degree of assignment of related tasks to the same workstations. Maximizing work relatedness where possible has some practical advantages. If related tasks are assigned to the same workstations, then work efficiency, quality, and worker satisfaction can improve. The index of work relatedness as given by Agrawal [14] is used in this study: $wr = n \Big/ \sum_{j=1}^{n} sn_j$, where n is the total number of workstations and sn_j is the number of connected networks representing the precedence relations of tasks assigned to station j. Moreover, it is assumed that the cycle time is predetermined. The constraints of the problem are precedence constraints, cycle time constraints, and zoning constraints. The precedence diagram that is shown in Fig. 2 illustrates an example of the two-sided ALB problem, which is obtained from Lee et al. [11]. A circle indicates a task. Each task is associated with a label (t_i, d), where t_i is the ith task processing time and d=(L, R, or E) denotes the preferred operation direction.

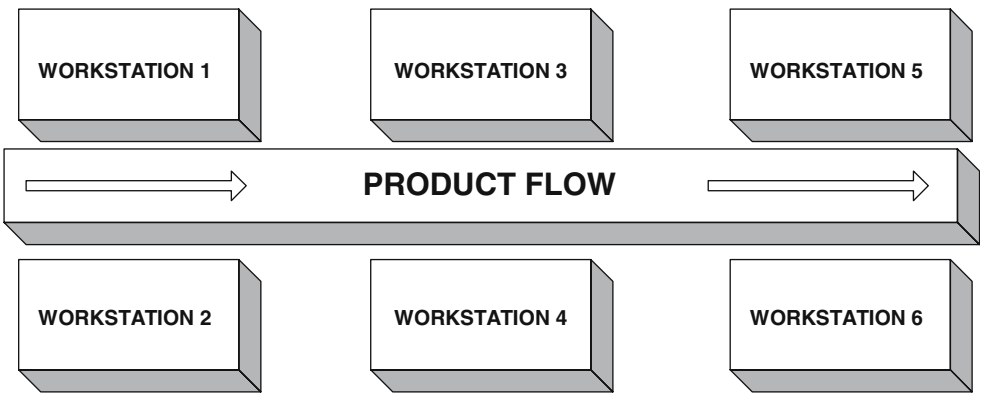

Fig. 1 A two-sided assembly line

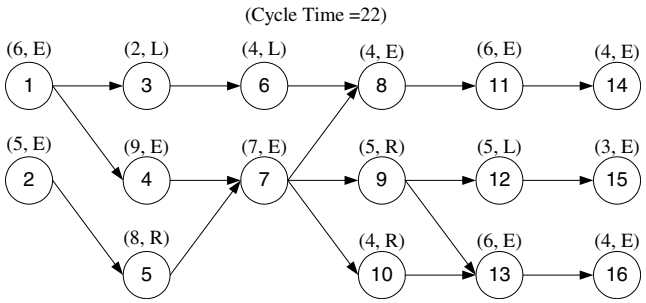

Fig. 2 Precedence diagram

Zoning constraints indicate which tasks must be assigned to the same workstation and which tasks must not be assigned to the same workstation. Tasks are defined to be self-containing, useful work elements. Nevertheless, because of safety, skill, or equipment requirements, we may wish to assign pairs of tasks to the same workstation. We let *ZS* be the set of task pairs that must be assigned to the same workstation. On the other hand, certain tasks may not be allowed to take place at the same workstation. We let *ZD* be the set of task pairs that cannot be performed at the same workstation.

Several assumptions are made when using positional constraints. Firstly, it is assumed that tasks which must be assigned to the same workstation follow each other. In the *ZS* set, all tasks' immediate precedence must be a member of this *ZS* set, except the first precedence task. Secondly, tasks that cannot be performed at the same workstation cannot be performed at the same workstation pair in the two-sided ALB problem. For example, welding and painting tasks cannot be performed at the same workstations because of the risk of fire danger. So, in a two-sided problem, these tasks cannot be performed at the same workstation pair due to the proximity of these stations. Consider the problem as shown in Fig. 3. Tasks 1, 3, 4 and tasks 6, 7, 8 can be a *ZS* set, but tasks 10, 12, 13 cannot be a *ZS* set, since one of the immediate predecessors of the 13th task (9th task) is not in the *ZS* set.

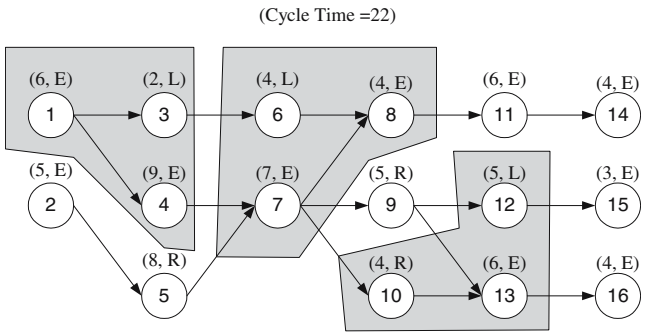

Fig. 3 Precedence diagram and positional constraints

3 The ant colony algorithm for 2sALBz problems

Notation

n	Number of tasks
cs	Ant colony size
m	Index for ant ($1 \le m \le cs$)
i	Index for task ($1 \le i \le n$)
k, R, L	Indexes for station ($1 \le k \le n$), R=right station index, L=left station index
q	Queue no ($1 \le q \le n$)
List A={}	Candidate task(s) list A
List B={}	Candidate task(s) list B
nc	Number of candidate tasks in candidate list B
$a(i)$	0 if the ith task is not assigned to a station; 1 if the ith task is assigned but not finished; 2 if the ith task is assigned and finished
$WQ(q)$	Number of the task assigned to the qth queue ($1 \le WQ(q) \le n$)
$f(m, i, q)$	Pheromone variable (1 if the ith task is assigned to the qth queue by the mth ant; 0 otherwise)
$gf(i, q)$	Global pheromone quantity for assigning the ith task in the qth queue
$pw(i)$	ith task positional weight
$wr(i)$	ith task work relatedness index
tf	Total pheromone quantity
tpw	Total positional weight value
twr	Total work related index value
$r(i)$	ith task selection probability
$p(l)$	Cumulative probability ($l \rightarrow i$)
$x(i, k)$	Binary decision variable (1 if the ith task is assigned to the kth station; 0 otherwise)
$xb(i, k)$	Binary decision variable (the best solution)
$of(m)$	mth ant's objective function value
aof	Average objective function value
α	Importance rate of global pheromone quantity of decision ($1 \le \alpha \le 1$)
β	Importance rate of decision without pheromone ($1 \le \beta \le 1$)
δ	Importance rate of positional weight ($1 \le \delta \le 1$)
θ	Importance rate of work related index ($1 \le \theta \le 1$)

3.1 The algorithm

The basis of the proposed solution procedure consists of four steps. In the first step, the opening of the station pairs (right and left) is performed. In the second step, determination of the available tasks list which can be assigned to the current stations pair without violating the constraints is performed. Afterwards, a task is selected randomly from this set by using

the ant colony algorithm. Finally, the selected task is assigned to an appropriate station according to the rules and constraints. In order to find the solution, these four steps are repeated until all of the tasks have been assigned to the stations:

Repeat

Step 1: opening of new stations
Step 2: determine available tasks which can be assigned
Step 3: select task by using the ant colony algorithm
Step 4: selected task is assigned to the station

Until (all tasks have been assigned)

These steps are graphically shown in Fig. 4.

The solution procedure which is given above will be repeated until the target solution is found or the ant number reaches the ant colony size. So, in this procedure, every ant builds a solution. After an ant has generated a solution, the global solution is updated according to the current solution:

Repeat

Solution procedure
Update the global solution

Until (any number reaches ant colony size or the optimum solution is found)

The ant procedure is repeated until the target solution is found or the iteration number reaches the iteration limit:

Repeat

Ant procedure
Evaporation of the pheromone

Until (the target solution is found or the iteration number reaches the iteration limit)

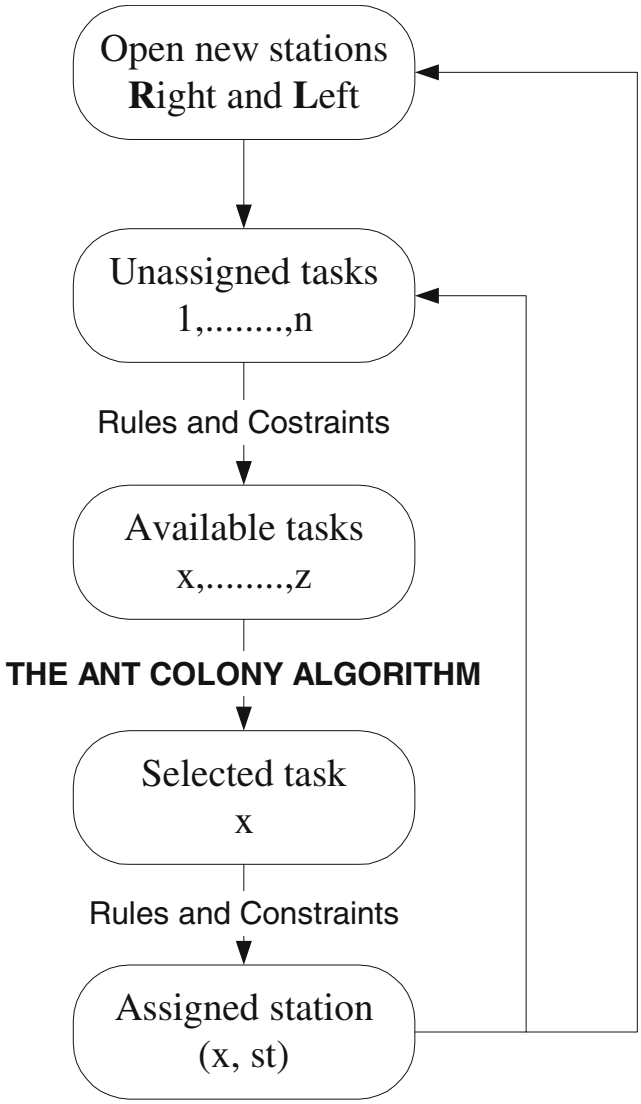

Fig. 4 Steps of the proposed algorithm

3.2 Pseudo code of the ant colony heuristic

START
Repeat

1. Set the initial values, start iteration.
Repeat

2. Create a new ant (m=m+1), set initial values (R=1, L=2, q=1).
Repeat

3. Form the candidate task(s) list A. Check all tasks. If a task isn't assigned (a_i=0), and its immediate precedence tasks are assigned (a_j>0 j∈IP_i) and it can be processed at the current station pair (positional constraint), then add the task to list A (List A={i}).

3.1. If list A is empty, then go to step 4.2 (open new stations).
4. Form the candidate task(s) list B. Check the tasks in list A for cycle time availability. If a task is eligible for assignment to a station in the current pair (right or left station) by satisfying the cycle time constraint, then add this task to list B. If a task has to be processed with other tasks (positional constraint), this task's processing time is assumed to be equal to the total of the tasks in the group.

4.1. If list B is empty, then go to step 4.2 (open new stations). Else, go to step 5.
4.2. Open new stations pair (R=R+2, L=L+2) and go to step 3.

5. Determine all tasks' selection probability in candidate list B according to the global pheromone quantities, work relatedness, and positional weight values of the tasks:

$$tf = \sum_{i \in \text{list B}} gf(i,\ q),\ twr = \sum_{i \in \text{list B}} wr(i),$$
$$tpw = \sum_{i \in \text{list B}} pw(i)$$

$$\underset{i \in \text{list B}}{r(i)} = \frac{((gf(i,\ q) * \alpha) + (pw(i) * \delta) + (wr(i) * \theta) + \beta)}{((\alpha * tf) + (\beta * nc) + (\delta * tpw) + (\theta * twr))}$$

6. Select a task randomly from list B according to the selection probabilities:

$$p(l) = p(l-1) + r(i) \quad l = 1, \ldots,\ nc \quad i \in \text{list B}$$
$$(p(0) = 0,\ p(nc) = 1)$$

Generate a random number, rand∈(0, 1). Choose the ith task whose cumulative probability satisfies p(l-1)≤ rand≤p(l).

7. Assign the selected task to the appropriate station. If the task assembly side is right (left), then assign the task to the right (left) station. If the task assembly side is either, then go to step 7.1 (select task) and assign the task to the selected station (x(i, k)=1, i=selected task, k=selected station).

7.1. Select the station which can finish the task earlier. If the finishing time of the stations are equal, then randomly select a station.

8. Deposit pheromone (f(m, i, q)=1. Update variables (WQ_q=i, a_i=1). Increase the queue number, q=q+1.

8.1. If the selected task has to be processed with some other tasks (a group of tasks), then select a task from the group according to the precedence order and go to step 7.

Until (all of the tasks are assigned to stations)

9. Update global pheromone quantities:

$$gf(i,\ q) = f(m,\ i,\ q) + gf(i,\ q)$$

10. Calculate the objection function value of the current ant's solution. If it is better than the global optimum, then update the best solution as the current ant's solution and update the global optimum as the current ant's solution objective function value. If of (m)<best solution, then best solution=of(m). For all tasks (i) and stations (k), xb(i, k)=x(i, k).

Until (ant number reaches ant colony size)

11. Determine the ant(s) which did not find improving solutions. Evaporate the pheromone which was deposited by these ants:

$$aof = \left[\left(\sum_{m=1}^{cs} of(m) \right) \Big/ cs \right]$$

If of(m)>aof, then for all tasks (i) and queues (q):

$$gf(i,\ q) = gf(i,\ q) - f(m,\ i,\ q)$$

Until (iteration number reaches iteration limit)

END

Table 1 Results of the computational study for two-sided assembly line balancing (ALB) problems without zoning constraints

Problem	Cycle time	GA	Group assignment	ACO	CPU time (s)
P9	3	6	–	6	<1
	4	5	–	5	<1
	5	4	–	4	<1
	6	3	–	3	<1
P12	5	6	–	6	<1
	6	5	–	5	<1
	7	4	–	4	<1
P24	20	8	–	8	<1
	25	6	–	6	<1
	30	5	–	5	<1
	35	5	–	5	<1
	40	4	–	4	<1
P65	326	–	17	17	<1
	381	–	15	15	<1
	435	–	13	13	<1
	490	–	12	12	<1
	544	–	10	10	2.48
P148	204	–	27	26	4.39
	255	–	21	21	15.64
	306	–	18	18	50.91
	357	–	15	15	3.78
	408	–	14	14	2.19
	459	–	13	12	180.76
	510	–	11	11	15.05
P205	1133	–	23	24	451.14
	1322	–	20	22	449.27
	1510	–	20	18	288.2
	1699	–	16	18	448.28
	1888	–	16	15	177.84
	2077	–	14	14	7.06
	2266	–	13	12	131.3
	2454	–	12	12	6.99
	2643	–	12	11	68.54
	2832	–	10	10	303.63

4 Computational study

Two types of computational work are carried out. In order to be able to make a comparison with the existing algorithms, the test problems are solved without any positional constraints in the first computational study. In the second part, positional constraints are added to the problem, and these problems are solved. The ant algorithm is programmed in Visual Basic 5.0 and tested on a Pentium III, 1.6 GHz, 256 MB RAM personal computer.

In the computational studies, six test problems, which are available in the literature, are used. The test problems P9, P12, P24 are taken from Kim et al. [10], P65, P205 are taken from Lee et al. [11], and P148 is taken from Bartholdi [9]. In these data sets, there are no positional constraints.

Table 2 Results of the computational study for two-sided ALB problems with zoning constraints (2sALBz)

Problem	Cycle time	Without zoning constraints	CPU time (s)	With zoning constraints	CPU time (s)
P9	3	6	<1	7	<1
	4	5	<1	6	<1
	5	4	<1	4	<1
	6	3	<1	3	<1
P12	5	6	<1	6	<1
	6	5	<1	5	<1
	7	4	<1	5	<1
P24	20	8	<1	8	<1
	25	6	<1	6	<1
	30	5	<1	5	<1
	35	5	<1	5	<1
	40	4	<1	4	<1
P65	326	17	<1	17	3.52
	381	15	<1	15	<1
	435	13	<1	13	2.78
	490	12	<1	12	<1
	544	10	2.48	10	1.85
P148	204	26	4.39	26	10.32
	255	21	15.64	21	3.64
	306	18	50.91	18	463.39
	357	15	3.78	18	2.06
	408	14	2.19	15	2.02
	459	12	180.76	13	465.92
	510	11	15.05	11	6.76
P205	1133	24	451.14	25	264.32
	1322	22	449.27	22	264.31
	1510	18	288.2	19	270.34
	1699	18	448.28	18	264.28
	1888	15	177.84	16	263.91
	2077	14	7.06	16	266.76
	2266	12	131.3	14	259.72
	2454	12	6.99	14	258.44
	2643	11	68.54	13	259.79
	2832	10	303.63	12	258.85

Table 3 The zoning constraints for the test problems

Test problem	Set of task pairs
P9	ZS: {6, 9} ZD: {3, 9}
P12	ZS: {1, 4} ZD: {3, 5}
P24	ZS: {1, 11}; {7, 10} ZD: {14, 24}
P65	ZS: {3, 23, 24}; {31, 32}; {36, 37} ZD: {10, 30}; {46, 56}
P148	ZS: {90, 111, 112}; {11, 12, 13}; {29, 31}; {37, 38}; {40, 41}; {50, 51} ZD: {70, 30}; {8, 145}; {55, 71}; {147, 143}; {108, 102}; {125, 122}; {48, 110}
P205	ZS: {7, 8, 9, 10, 11, 12}; {20, 21, 22, 23}; {30, 31, 32}; {37, 38, 39}; {2, 3} ZD: {25, 27}; {29, 33}; {35, 110}; {93, 109}; {114, 174}; {144, 154}; {156, 190}; {77, 88}; {87, 100}; {40, 70}

ZS=**Z**one **S**ame: the set of tasks pairs that must be assigned to the same workstation
ZD=**Z**one **D**ifferent: the set of tasks pairs that cannot be performed on the same workstation

Therefore, the positional constraints are generated in this work for the test problems, which are listed in Table 3.

The test problems are solved with various cycle times. In total, 34 tests are made. The solutions are compared with the published results. The results are tabulated in Table 1. In Table 1, a comparison is also given with the results of Kim et al.'s genetic algorithm (GA) [10] and Lee et al.'s group assignment procedure [11]. Comparisons are made in terms of the number of stations found by the algorithms. In Kim et al. [10] and Lee et al. [11], only the average results are given. Therefore, we also used the average number of stations for our comparisons. If the average number is not an integer value, then the maximum integer number which is smaller than the average station number is taken (e.g., 17.7→17, 12.1→12, etc.).

In the second part of the computational study, the positional constraints are generated hypothetically. Again, 34 tests are generated. The results are shown in Table 2. The zoning constraints are shown in Table 3.

For the two-sided ALB problem without positional constraints, the proposed algorithm found the same results for 25 problems, in six test problems it performed better, and only in 3 test problems slightly worse solutions are obtained.

5 Discussion and conclusions

There are many algorithms and solution procedures to balance the well known classical one-sided assembly lines.

However, little attention has been paid to solving two-sided ALB problems. This might be due to the complexity of the problem. Unlike one-sided assembly lines, the sequence-dependent finish times of tasks need to be taken into account in balancing two-sided assembly lines. There are just a few studies in the literature that have proposed solution approaches to the two-sided ALB problem. These studies are based on GAs and the group assignment heuristic. In the previous researches, there is also not enough consideration given to zoning and positional constraints.

In this study, an ant-colony-based heuristic algorithm is proposed for solving 2sALBz problems. This paper makes one of the first attempts to show how an ACH can be applied to solve 2sALBz problems. In this paper, example applications are presented and computational experiments are performed to present the suitability of the ACH to solve 2sALBz problems. Promising results are obtained from the solution of test problems without zoning constraints, which are collected from the literature. For the constrained case, no published results are available for direct comparison. Therefore, some of the test problems are converted into constrained problems and then solved. The present study can be extended in several ways. The problem can be modeled and solved as a multiple-objective optimization problem by taking into account several other criteria, such as load balancing and smoothing. An extensive parameter analyses study can be performed to compare the performance of several algorithms from different perspectives. Real-life case studies can provide very useful research contributions, and bus and truck assembly factories are very good candidates for this purpose.

References

1. Gutjahr AL, Nemhauser GL (1964) An algorithm for the line balancing problem. Manage Sci 11(2):308–315
2. Talbot FB, Patterson JH, Gehrlein WV (1986) A comparative evaluation of heuristic line balancing techniques. Manage Sci 32 (4):430–454
3. Ghosh S, Gagnon RJ (1989) A comprehensive literature review and analysis of the design, balancing and scheduling of assembly systems. Int J Prod Res 27(4):637–670
4. Baykasoglu A (2006) Multi-rule multi-objective simulated annealing algorithm for straight and U type assembly line balancing problems. J Intell Manuf 17(2):217–232
5. Arcus AL (1963) An analysis of a computer method of sequencing assembly line operations. PhD dissertation, University of California, Berkeley, California
6. Johnson RV (1983) A branch and bound algorithm for assembly line balancing problems with formulation irregularities. Manage Sci 29(11):1309–1324
7. Gunther RE, Johnson GD, Peterson RS (1983) Currently practiced formulations for the assembly line balance problem. J Oper Manag 3(4):209–221
8. Falkenauer E (2005) Line balancing in the real world. In: Bouras A, Gurumoorthy B, Sudarsan R (eds) Proceedings of the International Conference on Product Lifecycle Management PLM'05. Inderscience, Geneva, Switzerland, pp 360–370
9. Bartholdi JJ (1993) Balancing two-sided assembly lines: a case study. Int J Prod Res 31(10):2447–2461
10. Kim YK, Kim Y, Kim YJ (2000) Two-sided assembly line balancing: a genetic algorithm approach. Prod Plan Control 11(1):44–53
11. Lee TO, Kim Y, Kim YK (2001) Two-sided assembly line balancing to maximize work relatedness and slackness. Comput Ind Eng 40(3):273–292
12. Lapierre SD, Ruiz AB (2004) Balancing assembly lines: an industrial case study. J Oper Res Soc 55(6):589–597
13. Dorigo M, Di Caro G, Gambardella LM (1999) Ant algorithms for discrete optimization. Artif Life 5(2):137–172
14. Agrawal PK (1985) The related activity concept in assembly line balancing. Int J Prod Res 23(2):403–421