An immune genetic algorithm for simple assembly line balancing problem of type 1

Han-ye Zhang
School of Mechanical and Materials Engineering, Jiujiang University, Jiujiang, China

Abstract

Purpose – The purpose of this study is to develop an immune genetic algorithm (IGA) to solve the simple assembly line balancing problem of type 1 (SALBP-1). The objective is to minimize the number of workstations and workstation load for a given cycle time of the assembly line.

Design/methodology/approach – This paper develops a new solution method for SALBP-1, and a user-defined function named $\psi(\cdot)$ is proposed to convert all the individuals to satisfy the precedence relationships during the operation of IGA.

Findings – Computational experiments suggest that the proposed method is efficient.

Originality/value – An IGA is proposed to solve the SALBP-1 for the first time.

Keywords Assembly line balancing problem of type 1, Immune genetic algorithm, Precedence matrix

Paper type Research paper

1. Introduction

The assembly line balancing problem (ALBP) determines the assignment of tasks to an ordered sequence of stations, such that each task is assigned to exactly one station, no precedence constraint is violated and some selected performance measures are optimized (Sabuncuoglu *et al.*, 2000). The ALBP is an important issue in the manufacturing field, and it directly affects the production efficiency and the utilization of manufacturing resources.

The ALBP was first mathematically formulated by Salveson (1995). The ALBP (especially in the manufacturing industry) has attracted interest from many scholars and engineers in the past 60 years.

Generally speaking, the ALBP is divided into two classes:

1 simple assembly line balancing problem (SALBP); and
2 generalized assembly line balancing problem (GALBP).

The SALBP means that a single product is produced, the set of tasks are of deterministic operation times and the set of workstations are arranged in a straight line. Meanwhile, GALBP includes all of the problems that do not belong to the SALBP, such as a mixed-model or a multi-model line, parallel stations, U-shaped or two-sided lines and stochastic task times. According to the objectives, the SALBP can be categorized into three types:

1 *SALBP-1*: Minimize the number of stations for a given cycle time which mainly used in the design and installation phase of the assembly line.
2 *SALBP-2*: Minimize the cycle time by given the numbers of workstations which mainly used to optimize and adjust the existing assembly line.
3 *SALBP-E*: Balance the workload while both the workstation and the cycle time are optimized.

In this paper, only SALBP-1 is considered, because it is widely used in the industry to reduce number of workstations/machines/workers. These reductions lead to decreased production cost and improve the cost competitiveness of the plant (Pitakaso, 2015).

In summary, the methods for solving SALBP-1 discussed in the literature can be generally classified into three kinds: exact method, heuristic approach and artificial intelligence method.

Sprecher (1999) presented a branch-and-bound algorithm for solving SALBP-1. The computational results indicated that this algorithm can compete with the best algorithms currently available for solving SALBP-1. Easton *et al.* (1989) presented two network-based algorithms for solving the type 1 ALBPs (ALBP-1). Computational tests indicated that these algorithms were more efficient than previous network-based methods (including dynamic programming methods) and more favorably compared with the branch-and-bound methods. Peeters and Degraeve (2006) presented a new lower bound, namely, linear programming (LP) relaxation of an integer programming formulation based on Dantzig–Wolfe decomposition to solve SALBP-1. Computational results showed that the lower bound was equal to the best-known objective function value for the majority of the instances. Moreover, the new LP-based lower bound was able to prove optimality for an open problem. Liu *et al.* (2008) presented a branch-and-bound algorithm to solve SALBP-1. The proposed algorithms consisted of a constructive and two destructive algorithms. The computational results showed that the proposed algorithms were efficient. Bautista and Pereira (2009) presented a bounded dynamic programming to solve SALBP-1.

The current issue and full text archive of this journal is available on Emerald Insight at: **www.emeraldinsight.com/0144-5154.htm**

Assembly Automation
39/1 (2019) 113–123
 [ISSN 0144-5154]
[DOI 10.1108/AA-08-2017-101]

Received 12 May 2017
Revised 29 August 2017
26 October 2017
12 December 2017
12 January 2018
Accepted 17 January 2018

The results obtained by the procedure showed that the implementation was capable of obtaining an optimal solution of 267 out of the 269 instances found in the literature. As well, it also found the best-known solution for another instance. Sewell and Jacobson (2012) presented a branch, bound and remember (BB&R) algorithm for solving SALBP-1. The algorithm combined a variety of existing methods and created a novel approach that was computationally superior to all existing exact algorithms reported in the literature. In particular, for 269 known benchmark problems, the algorithm successively found the best solutions and proved their optimality for all these problems, typically in a fraction of a second. Vila and Pereira (2013) presented a new exact algorithm for solving SALBP-1. The algorithm was a station-oriented bidirectional branch-and-bound procedure based on a new enumeration strategy that explored the feasible solutions tree in a non-decreasing idle time order. The tests showed that the proposed algorithm outperformed the best existing exact algorithm for solving SALBP-1, verifying optimality for 264 of the 269 benchmark instances.

Because the SALBP-1 belongs to non-deterministic polynomial-hard class of combinatorial optimization problems, the exact methods are computationally inefficient in solving large-scale problems, and many researchers turn their interests toward develop efficient heuristic approaches to solve SALBP-1.

Helgeson and Birnie (1961) presented a method named ranked positional weight (PW) technique to solve SALBP-1. Hoffmann (1963) presented a procedure by applying the FORTRAN program to solve SALBP-1. The procedure led to optimal line balances by operating on a matrix of zeroes and ones called a "Precedence Matrix". Hackman *et al.* (1989) proposed a simple, fast and effective heuristic named immediate update first-fit (IUFF) for the SALBP-1. The algorithm introduced heuristic fathoming as a technique for reducing the size of the branch-and-bound tree. Fleszar and Hindi (2003) presented a new heuristic algorithm and new reduction techniques for the SALBP-1. The new heuristic was based on the well-known Hoffmann heuristic and built solutions from both sides of the precedence network to choose the best. The computational efficiency enables it to arrive at optimal or near-optimal solutions for large-scale problem instances in less than 2 seconds of computing time on a PC. Kilincci and Bayhan (2008) developed a heuristic algorithm, P-in, based on Pinvariants of PNs to solve SALBP-1. The algorithm was coded in MATLAB, and its efficiency was tested on the Talbot and Hoffmann benchmark data sets according to some performance measures and classifications. Based on a recent bidirectional approach and the famous critical path method widely used in project management, Yeh and Kao (2009) proposed a new efficient heuristic method to resolve the issue of task assignment for SALBP-1. The effectiveness of the proposed heuristic algorithm was verified by sample problems. Otto and Otto (2014) provided recommendations for *ad hoc*, *i.e.* without pretests or construction of priority rules, and illustrated their effectiveness on the example of ALBP with resource constraints. The authors showed that split-second priority rules could not only achieve better results than 20-100 s of an exact algorithm or a meta-heuristic but also significantly reduce the warming-up phase of algorithms.

Because of the differences among the ALBP, such as the numbers of tasks, and precedence relations between tasks, there was no heuristic rule to ensure that all of the ALBP-1 can get the optimal solution. In recent years, numerous researchers have employed artificial intelligence method to solve ALBP.

Ponnambalam *et al.* (2000) proposed a multi-objective genetic algorithm (GA) to solve SALBP-1. It was found that the proposed GA performs better in all the performance measures than the heuristics. However, the execution time for the GA is longer, because the GA searches for global optimal solutions with more iteration. To solve the deterministic and single-model ALBP, Sabuncuoglu *et al.* (2000) developed a new GA with a special chromosome structure that was partitioned dynamically through the evolution process. The results of extensive computational experiments indicated that the proposed GA approach outperforms the well-known heuristics in the literature. Goncalves and de Almeida (2002) presented a hybrid GA for the SALBP-1. The chromosome representation of the problem was based on random keys, and the assignment of operations to the workstations was based on a heuristic priority rule. The computation results validated the effectiveness of the algorithm. Yu and Yin (2010) presented an adaptive GA to solve SALBP-1. The computational results demonstrated that the proposed adaptive GA was an effective algorithm. Yolmeh and Kianfar (2012) proposed a hybrid GA to solve SALBP-1. The computational results validated the effectiveness of the algorithm. Blum (2008) proposed a hybrid ant colony optimization (ACO) approach called Beam-ACO for the SALBP-1. The results showed that Beam-ACO was a state-of-the-art meta-heuristic for the SALBP-1. Sulaiman *et al.* (2011) presented an approach based on the ACO technique to address the SALBP-1. It introduced an approach to dynamically assign the value of priority rule or heuristic information during the task selection phase by allowing the ant to look forward to its direct successors during the consideration in selecting a task to be assigned into a workstation. The computation results validated its effectiveness. Dou *et al.* (2013) proposed a direct discrete particle swarm algorithms (DDPSA) to solve SALBP-1. In the presented DDPSA, a particle is a permutation of N integers that represents the feasible task sequence directly, and a new multi-fragment crossover is proposed to update the particle and maintain the feasibility of the task sequence. To avoid the stagnation of search in local optimum, the fragment mutation is embedded into the DDPSA. The results showed that the DDPSA is effective and powerful for the SALBP-1. Lapierre *et al.* (2006) proposed a novel tabu search algorithm for solving both SALBP-1 and non-standard versions of this problem coming from real-life applications. Computational results on both showed that the method was efficient. Guden and Meral (2013) proposed an adaptive simulated annealing method to solve the real-life SALBP-1 of a dishwasher producer which consisted of approximately 300 tasks, 400 precedence relations per product model. The performance of the algorithm was tested on several problem instances from the literature and were found to be satisfactory. Then it was used to solve the real-life problem instance considered and a better solution than the current one was obtained. Zhang (2017) proposed an improved immune algorithm (IIA) to solve the SALBP-1.The results show that the proposed IIA is efficient.

Combining immune algorithm (IA) with GA, the detection and search ability near the optimal solutions are increased and the problem of premature convergence is avoided. In recent years, the IGA has been applied in the field of manufacturing, such as: job–shop scheduling problem (Chang *et al.*, 2011; Chen *et al.*, 2014; Wang *et al.*, 2010; Xu and Li, 2007) and the layout problem in the cellular manufacturing system (Ghosh *et al.*, 2016). But the study of ALBP by IGA has not been retrieved by the Engineering Village. In this paper, the IGA is used to study the SALBP-1.

Some more detailed literature review of the ALBP can be found in the literature (Battaia and Dolgui, 2013; Boysen *et al.*, 2008; Mz, 2010; Scholl and Becker, 2006; Sivasankaran and Shahabudeen, 2014; Tasan and Tunali, 2008).

The remainder of the paper is as follows: Section 2 provides the mathematical model of SALBP-1. Section 3 presents the IGA method for finding the solution. In Section 4, a set of benchmark assembly line problems taken from the literature are used to test and compare, and a real industrial case is applied. Section 5 concludes the paper with the recommendations of future work.

2. Mathematical model of SALBP-1

2.1 Notations

The following notations that will be used throughout the paper are described as follows:

- i = task identifier, where $i \in N$;
- j = task identifier, where $j \in N$;
- u = task identifier, where $u \in N$;
- v = task identifier, where $v \in N$;
- k = workstation identifier, where $k \in M$;
- n = number of tasks;
- m = number of workstations in the assembly line;
- N = set of tasks, where $N = \{1,2,\cdots,n\}$;
- M = set of workstations, where $M = \{1,2,\cdots,m\}$;
- C = cycle time;
- t_i = processing time of task i;
- $s(k)$ = the set of tasks which are currently assigned to workstation k;
- ST_k = processing time of the k^{th} workstation, $ST_k = \sum_{i \in s(k)} t_i$;
- P = precedence matrix, $P(i,j) = 1$, if task i is an immediate predecessor of task j; $P(i,j) = 0$, Otherwise;
- P' = precedence matrix, $P'(i,j) = 1$, if task i is an immediate or indirect predecessor of task j; $P'(i,j) = 0$, Otherwise.;
- SI = smooth index, where $SI = \sqrt{\frac{1}{m}\sum_{k=1}^{m}(C - ST_k)^2}$; and
- x_{ik} = binary variable, $x_{ik} = 1$, if task i is assigned to workstation k; $x_{ik} = 0$, otherwise.

2.2 Mathematical formulation

2.2.1 Objective function

Generally speaking, the objective function of SALBP-1 is to minimize the number of the workstations, e.g.:

$$f_1 = \min m$$

Sometimes, it might encounter such situations: some solutions have the same number of workstations, but these solutions have different task–workstation assignments. Hence, SI is adopted as the auxiliary objective function f_2, e.g.:

$$f_2 = \sqrt{\frac{1}{m}\sum_{k=1}^{m}(C - ST_k)^2}$$

Therefore, the objective function is:

$$\begin{aligned} \min f &= w_1 \times f_1 + w_2 \times f_2 \\ &= w_1 \times m + w_2 \times \sqrt{\frac{1}{m}\sum_{k=1}^{m}(C - ST_k)^2} \end{aligned} \tag{1}$$

Where w_1 and w_2 are the weighted coefficients. Compared with f_2, the value of f mainly depends on f_1, so the ratio of w_1/w_2 should be larger.

2.2.2 Constraint conditions

$$\sum_{k \in M} x_{ik} = 1, \forall i \tag{2}$$

$$\sum_{k \in M} k \cdot x_{ik} \le \sum_{k \in M} k \cdot x_{jk}, \forall P'(i,j) = 1 \tag{3}$$

$$\sum_{i \in N} x_{ik} \cdot t_i \le C, \forall k \tag{4}$$

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

Equation (2) ensures that every task is assigned to one and only one workstation. Constraint (3) defines precedence relationships between tasks i and j. Constraint (4) ensures that the processing time of every workstation does not exceed the cycle time. Constraint (5) defines the variable domains.

To guarantee that all the individuals satisfy the precedence relationships, a user-defined function named $\psi(\cdot)$ is defined as follows.

Definition. The formula that maps from A to A* is given as $\Psi(A) = A^*$, while satisfying:

- If $P'(a_{i,j}, a_{i+1,j}) = 0$, then change the place of $a_{i,j}$ and $a_{i+1,j}$ in Matrix A.
- If $P'(a_{i,j}, a_{i+1,j}) = 1$, then the place of $a_{i,j}$ and $a_{i+1,j}$ remains unchangeable in Matrix A.

Where:

$$A = \begin{bmatrix} a_{1,1} & \cdots & a_{1,j} & \cdots \\ \vdots & \ddots & \vdots & \vdots \\ a_{i,1} & \cdots & a_{i,j} & \cdots \\ a_{i+1,1} & \cdots & a_{i+1,j} & \cdots \\ \vdots & \ddots & \vdots & \ddots \\ a_{n,1} & \cdots & a_{n,j} & \cdots \end{bmatrix}$$

$$P' = \begin{matrix} \\ a_{i,j} \end{matrix} \begin{matrix} \begin{matrix} \quad & \quad & \quad & a_{i+1,j} & \quad \end{matrix} \\ \begin{bmatrix} p'_{1,1} & \cdots & p'_{1,v} & \cdots & p'_{1,n} \\ \vdots & \ddots & \vdots & p'_{a_{i,j},a_{i+1,j}} & \vdots \\ p'_{u,1} & \cdots & p'_{u,v} & \cdots & p'_{u,n} \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ p'_{n,1} & \cdots & p'_{n,v} & \cdots & p'_{n,n} \end{bmatrix} \end{matrix}$$

$a_{i,j} \in \{1,2,\cdots,n\}\ \forall i,j$;
n represents the number of tasks;

Each column represents a full permutation of n in Matrix A;

$$p'_{u,v} = P'(u,v) = \begin{cases} 1, & \text{if task } u \text{ is an immediate or indirect predecessor of task } v; \\ 0, & \text{otherwise} \end{cases}$$

A is a matrix which represents an initial population generated randomly or a new population updated with the increase of the number of iteration (such as crossover operation or mutation operation in GA).

3. Immune genetic algorithm for SALBP-1

Because the IGA consists of GA and IA; GA and IA are elaborated separately in the following sections.

3.1 Genetic algorithm

GA is mainly composed of two basic operations, namely, crossover and mutation. In this paper, the one-point crossover and swap mutation are used, and the detailed descriptions are as follows.

3.1.1 Single-point crossover operation

Let the two chromosomes which will participate in the single-point crossover operation be Parent 1 and Parent 2, respectively.

Step 1. Determine the total number of chromosomes selected in the crossover operation. If it is odd, a chromosome will be randomly deleted. Then, two chromosomes are selected randomly to mate, and the two mating chromosomes named Parent 1 and Parent 2 are operated as follows.

Step 2. Randomly select a positive integer as the crossover point from 1 to $n - 1$, where n is the gene number of chromosome, *e.g.*, the number of tasks.

Step 3. Maintain the left segment of the crossover point in Parent 1, and in Parent 2 delete these genes which are the same values in the left segment of the crossover point in Parent 1. Maintain the original order of the remaining genes in Parent 2 to replace the right segment of the crossover point in Parent 1, e.g. Offspring 1 is generated.

Step 4. Likewise, Offspring 2 is generated.

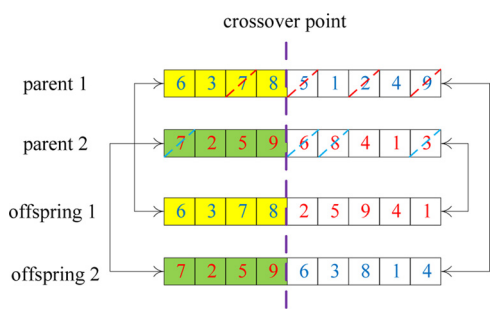
The principle is shown in Figure 1.

3.1.2 Swap mutation operation

Let the chromosome which will be participated in the swap mutation operation be the parent.

Step 1. Randomly select two positive integers as the mutation points from 1 to $n - 1$, where n is the gene number of chromosome, e.g. the number of tasks.

Figure 1 Process of single-point crossover



Step 2. The two genes in the mutation points are interchanged, and then the offspring is generated. The principle is shown in Figure 2.

3.2 Immune algorithm

3.2.1 Affinity

In the immune optimization algorithm, the antigens correspond to optimization problems, the antigens are usually an objective function and the antibodies correspond to the optimum solution to the problems.

The affinity between antigen and antibody corresponds to the matching between the feasible solution and the optimum solution which is called antibody fitness. The smaller the antibody fitness, the closer the feasible solution is to the optimum solution. The antibody fitness is calculated as shown in equation (1).

The affinity between antibodies corresponds to a similar degree between feasible solutions that is called similarity. The larger the similarity, the more similar the feasible solutions will be obtained.

Suppose M antibodies, each antibody is composed of N genes, and r species can be chosen in each genetic *locus* ranging from k_1 to k_r; $i(j)$ represents the values of antibody i on the jth genetic *locus* (Figure 3). Obviously, $i(j) \in k_t$, where $1 \le i \le M$, $1 \le j \le N$, $1 \le t \le r$.

The average information entropy of the M antibodies is defined as follows:

$$H(M) = \frac{1}{N}\sum_{j=1}^{N} H_j(M) \tag{6}$$

Where $H_j(M)$ represents the information entropy of the jth gene among the M antibodies, and it is defined as follows:

$$H_j(M) = \sum_{i=1}^{r}(-p_{ij} \cdot log_2 p_{ij}) \tag{7}$$

Therefore, equation (8) can be derived from equations (6) and (7):

$$H(M) = \frac{1}{N}\sum_{j=1}^{N}\sum_{i=1}^{r}(-p_{ij} \cdot log_2 p_{ij}) \tag{8}$$

Figure 2 Process of swap mutation

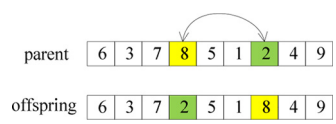
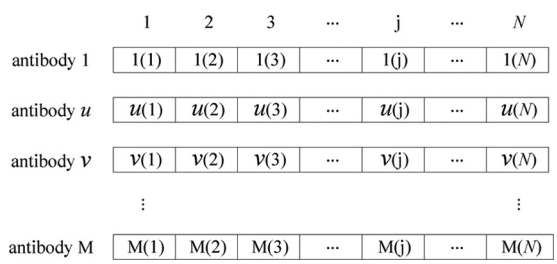


Figure 3 Allele schematic drawing



Where p_{ij} represents the probability that the value of the jth genetic *locus* is k_i among the antibody population. If (1) is the value of the jth genetic *locus* is k_i and (2) the frequency that k_i occurs is m_{ij}, then $p_{ij} = m_{ij}/M$.

The similarity S_{uv} between Antibodies u and v is defined as follows:

$$S_{uv} = \frac{1}{1 + H(2)} \tag{9}$$

Where $H(2)$ represents the average information entropy between Antibodies u and v. $H(2)$ can be calculated by equation (8), but the process of solving $H(2)$ is complicated. It can be simplified as follows.

$u(j) = v(j)$ means that the value of the jth genetic *locus* of Antibody u is equal to that of Antibody v, i.e. Antibody u and Antibody v have Allele j.

$|u(j) = v(j)|$ represents the total numbers of Allele j between Antibody u and Antibody v. Further, let $Neg = |u(j) = v(j)|$.

(1)$\forall j \in [1,N]$, if $u(j) = v(j)$, further let $u(j) = v(j) = k_1$, then $m_{1j} = 2$, $p_{1j} = 1$, and $(-p_{ij}log_2p_{ij}) = (-p_{1j}log_2p_{1j}) = 0$;

(2)$\forall j \in [1,N]$, if $u(j) \neq v(j)$, further let $u(j) = k_1$, $v(j) = k_2$, then $m_{1j} = 1$, $m_{2j} = 1$, $p_{1j} = 1/2$, $p_{2j} = 1/2$, and $(-p_{ij}log_2p_{ij}) = (-p_{1j}log_2p_{1j}) + (-p_{2j}log_2p_{2j}) = 1$.

According to equation (8), $H(2)$ can be derived as follows:

$$H(2) = \frac{N - Neg}{N} \tag{10}$$

According to equations (9) and (10), S_{uv} can be derived as follows:

$$S_{uv} = \frac{N}{2N - Neg} \tag{11}$$

3.2.2 Concentration

Antibody concentration c_i means the numbers of the antibodies similar to antibody i divide the antibody population M, and it is defined as follows:

$$c_i = \frac{1}{M}\sum_{j=1}^{M} Q_j \tag{12}$$

where $Q_j = \begin{cases} 1 & if\ S_{ij} \geq \lambda \\ 0 & if\ S_{ij} < \lambda \end{cases}$, and λ means similarity threshold ranging from 0.9 to 1.

3.2.3 Antibody survival expectancy

In order to preserve the diversity of the antibody population and avoid premature convergence, antibody survival expectancy is used to promote and inhibit the antibody population, and the antibody survival expectancy is defined as follows:

$$e_i = f_i \cdot c_i \tag{13}$$

Obviously, the antibody with low fitness or low concentration will be promoted, and that with high fitness or high concentration will be inhibited. As a result, this forms a new diversity-keeping strategy.

3.2.4 Memory vault

A memory vault is used to save the excellent antibody, and the original memory vault is usually designed by the engineers.

3.3 IGA algorithm flow

3.3.1 Step 1. Setup parameters

- PS: The population size.
- P_C: The crossover probability.
- P_m: The mutation probability.
- P_i: The immune vaccination probability.
- λ : The similarity threshold, $\lambda = 0.91$.

3.3.2 Step 2. Initialize

The PS individuals are generated randomly. All of these individuals make up the initial population named A. Then the definition function $\Psi(\cdot)$ is used to guarantee that all of the individuals meet the Constraint (3), e.g. $\Psi(A) = A^*$.

3.3.3 Step 3. Update the population

3.3.3.1 Crossover operation. Let $M_1 = \lceil PS * p_c \rceil$; the symbol of $\lceil x \rceil$ means that the minimal positive integer is greater than or equal to x and it has the same meanings as in the following paragraphs.

If M_1 is odd, then $M_1 - 1$ is assigned to M_1, e.g. $M_1 \leftarrow M_1 - 1$.

First, all of the M_1 individuals are selected randomly from $A^{(t)}$ to constitute a matrix which is named $A_1^{(t)}$, where $A^{(t)}$ means the t^{th} population of A. Second, any two individuals are selected randomly from $A_1^{(t)}$ to carry on the paired crossover operation until all the M_1 individuals are chosen, according to the principle of Figure 1. Third, the $A_1^{(t)}$ matrix is written as $\overline{A_1^{(t)}}$ after the crossover operation. Finally, $\overline{A_1^{(t)}}$ is executed $\Psi(\cdot)$ operation and let $\overline{A_1^{(t)*}} = \psi\left(\overline{A_1^{(t)}}\right)$.

3.3.3.2 Mutation operation. Let $M_2 = \lceil PS * p_m \rceil$. First, all of the M_2 individuals are selected randomly from $A^{(t)}$ to constitute a matrix which is named $A_2^{(t)}$. Second, each individual in $A_2^{(t)}$ carries out a mutation operation according to the principle of Figure 2. Third, the $A_2^{(t)}$ matrix is written as $\overline{A_2^{(t)}}$ after the mutation operation. Finally, $\overline{A_2^{(t)}}$ is executed $\Psi(\cdot)$ operation and let $\overline{A_2^{(t)*}} = \psi\left(\overline{A_2^{(t)}}\right)$.

3.3.3.3 Immune vaccination operation. Let $M_3 = \lceil PS * p_i \rceil$. P_g is chosen as the vaccine in the memory vault. First, all of the M_3 individuals are selected randomly from $A^{(t)}$ to constitute a matrix which is named $A_3^{(t)}$. Second, any individual in $A_3^{(t)}$ and P_g perform the crossover operation according to the principle of Figure 1. Third, the $A_3^{(t)}$ matrix is written as $\overline{A_3^{(t)}}$ after the crossover operation. Finally, $\overline{A_3^{(t)}}$ is the executed $\Psi(\cdot)$ operation and let $\overline{A_3^{(t)*}} = \psi\left(\overline{A_3^{(t)}}\right)$.

3.3.3.4 The selection of antibody survival expectancy. Let $B^{(t)} = \overline{A_1^{(t)*}} \cup \overline{A_2^{(t)*}} \cup \overline{A_3^{(t)*}}$. The antibody survival expectancy of all the $(PS + M_1 + M_2 + M_3)$ individuals in $B^{(t)}$ is calculated according to equation (13). Then the results of these calculations are arrayed in the ascending order. Finally, the first PS individuals are selected to constitute the next-generation population $A^{(t+1)}$.

3.3.4 Step 4. Update the memory vault

The objective function of all the M individuals in $A^{(t+1)}$ are calculated according to equation (1). Then the individual corresponding to the minimum value of the objective function

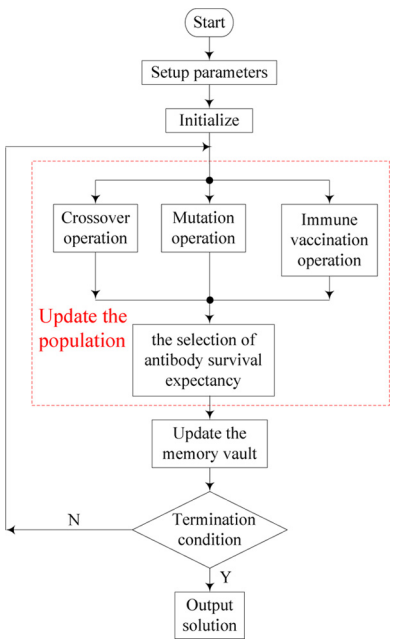
is denoted as $P_g^{(t+1)}$. If $f\left(P_g^{(t+1)}\right) \leq f\left(P_g^{(t)}\right)$, then $P_g = P_g^{(t+1)}$. Otherwise, $P_g = P_g^{(t)}$.

3.3.5 Step 5. Termination condition

Generally speaking, the algorithm is stopped when it satisfies one of the following three conditions:

1 The value of the best individual objective fitness is equal to the predefined threshold.
2 The value of the best individual objective fitness and the average value of the population's objective fitness converge with the increase of iterations.
3 The number of iterations reaches the predefined generations.

Figure 4 Flowchart of IGA



In this paper, condition 2 is selected as the termination condition of the IGA. Once condition 2 is satisfied, then the IGA will go to Step 6; otherwise, it will go to Step 3.

3.3.6 Step 6. Output

The flowchart of the IGA is shown in Figure 4.

4. Computational results

This section reports computational results for the proposed IGA described in Section 3. It compares two existing exact methods, two existing heuristic approaches and two existing artificial intelligence methods on the benchmark problems which can be obtained from http://alb.mansci.de/

The IGA was coded in MATLAB. The experimental results were obtained on a PC with 2 GHz CPU and 3 GB RAM. Referring to references (Chang *et al.*, 2011; Chen *et al.*, 2014; Wang *et al.*, 2010; Xu and Li, 2007), the IGA had the following configuration: PS =100, P_c = 0.6, P_m = 0.1, P_i = 0.3. The experiment was terminated when the optimal solution was found and the computational results were collected and reported.

4.1 Compare with exact method

In this section, the proposed IGA was compared with the BB&R-BFS with BINLB (Sewell and Jacobson, 2012) and Enumeration Procedure (Vila and Pereira, 2013). The BB&R-BFS with BINLB was coded in the C programming language and was executed on a 2.0 GHz, dual core and Intel T7200 processor with 3.25 GB of memory. The Enumeration Procedure was programmed in C++ and was executed on a Macintosh MacBook Pro with a 2.33 GHz Intel Core 2 Duo processor and 2 GB of RAM.

The computational results are shown in Figure 5. In Figure 5, there are ten columns. The first column lists the names of the used test instances. The second column shows the number of tasks. The third column shows values of cycle time, measured in second. The fourth column shows the optimal number of stations of the test instances found in the literature. The remaining columns report the best number of stations and the computational time by BB&R-BFS with BINLB,

Figure 5 Comparison with BB&R-BFS with BINLB and enumeration procedure

Problem				BB&R-BFS with BINLB		Enumeration procedure		IGA	
Name	*n*	*C*	Optimal	Best	Computational time	Best	Computational time	Best	Computational time
Wee-Mag	75	47	33	33	0.20	33	3,600	33	1.50
Wee-Mag	75	49	32	32	0.16	32	3,600	32	1.44
Wee-Mag	75	50	32	32	0.17	32	3,600	32	1.42
Arcus2	111	7,520	21	21	6.83	21	270	21	1.70
Barthol2	148	84	51	51	0.48	51	3	51	4.13
Barthol2	148	85	50	50	Mem	50∼51	3,600	50	4.09
Barthol2	148	89	48	48	0.81	48	12	48	4.06
Barthol2	148	91	47	47	0.53	47	2	47	3.93
Barthol2	148	93	46	46	0.48	46	2	46	3.90
Barthol2	148	95	45	45	Mem	45	4	45	3.70
Barthol2	148	97	44	44	0.64	44	4	44	3.65
Barthol2	148	99	43	43	0.73	43	2	43	3.59
Barthol2	148	109	39	39	0.92	39	4	39	3.43
Scholl	297	1,394	50	50	Mem	50	162	50	82.09
Scholl	297	1,483	47	47	3,600	47∼48	3,600	47	61.46
Scholl	297	1,515	46	46	3,600	46	1,059	46	57.25
Scholl	297	1,699	42	42	1.97	42	1,236	42	47.19

Note: Mem means program terminated as a result of exceeding memory limit

Enumeration Procedure, and IGA. The unit of computational time is seconds.

From the computational results shown in Figure 5, it can be seen that the IGA can also obtain the optimal solution, but some of the computational times are longer, which are marked by yellow shadow.

4.2 Compare with heuristic approach

In this section, the proposed IGA is compared with LRS (an algorithm defined by Lapierre *et al.*, 2006) and rules combination algorithm (RCA) (Li *et al.*, 2013).

The LRS was coded in the C++ programming language and was carried out on a Sun workstation UltraSparc 10 (100 MHz). The RCA was programmed in MATLAB and was executed on a PC with AMD SempronLE1250 processor and DDR2 667 MHz.

The results are shown in Figure 6. For the meanings of the columns in the Figure 6, please refer to Figure 5. The unit of computational time is seconds.

From the computational results shown in Figure 6, it can be seen that IGA can find 15 more optimal solutions compared with LRS and RCA which are marked by green shadow, but all the computational time is longer which are marked by yellow shadow.

4.3 Compare with artificial intelligence method

In this section, the proposed IGA was compared with HPSO (Dou *et al.*, 2012) and BACO (Blum, 2008).

The HPSO was coded in the VC++6.0 programming language and was carried out on a PC with PEV1.7 GHz CPU and 512 MB of memory. The BACO was programmed in C++ and was carried on a PC with an AMD64 × 2 4400 processor and 4 GB of memory.

The results are shown in Figure 7. For the meanings of the columns in the Figure 7, please refer to Figure 5. The unit of computational time is seconds.

From the computational results shown in Figure 7, it can be seen that IGA can find three more optimal solutions which are marked by green shadow, and the convergence rate of IGA is faster than that of HPSO marked by red shadow, but some of the computational time is longer which are marked by yellow shadow.

4.4 Testing on a practical application

The effectiveness of the IGA has been verified, and then it is used to solve a real-world application, namely, the APXV9R20B antenna assembly shop, and all the information is seen in Table I (Hu *et al.*, 2014). The cycle time is 38.40 s. The unit of t_i is seconds.

The IGA is implemented and compared with the basic particle swarm optimization algorithm (PSO), and the results of task-workstations assignment are seen in Table II.

According to the information in Table II, the performance comparison results of IGA and PSO are seen in Table III.

From the calculation in Table III, it can be seen that the number of workstations and the balance rate calculated by IGA are the same as those by PSO, but the smooth index calculated by IGA is smaller than that by PSO. The smaller smooth index means that the processing time among workstations is closer, and this will increase the coefficient of utilization of workers and equipment. Therefore, the proposed IGA can obtain a better solution.

5. Conclusions

In the paper, an IGA is proposed to solve the SALBP-1 for the first time that aims to minimize the number of workstations and workstation load for a given cycle time of assembly line. Combining IA with GA, the detection and search ability near the optimal solutions are increased and the problem of premature convergence is avoided. A user-defined function named $\psi(\cdot)$ is designed to convert all the individuals to satisfy the precedence relationships during the operation of IGA.

Figure 6 Comparison with LRS and RCA

Problem				LRS		RCA		IGA	
Name	*n*	*C*	Optimal	Best	Computational time	Best	Computational time	Best	Computational time
Arcus2	111	5,755	27	27	2.75	27	0.0928	27	1.90
Arcus2	111	8,847	18	18	4.01	18	0.0890	18	1.62
Arcus2	111	10,027	16	16	4.94	16	0.0821	16	1.54
Arcus2	111	10,743	15	15	3.78	15	0.0887	15	1.48
Arcus2	111	11,378	14	14	5.16	14	0.0863	14	1.44
Arcus2	111	17,067	9	9	8.31	9	0.0675	9	1.30
Scholl	297	1,422	50	50	*	50	1.5393	50	59.31
Scholl	297	1,452	48	49	*	49	1.5229	48	58.28
Scholl	297	1,584	44	45	*	45	1.5327	44	60.28
Scholl	297	1,834	38	39	*	39	1.5578	38	45.79
Scholl	297	1,883	37	38	*	38	1.5421	37	56.82
Scholl	297	1,935	36	37	*	37	1.5098	36	44.94
Scholl	297	2,049	34	35	*	35	1.5273	34	43.78
Scholl	297	2,111	33	34	*	34	1.5626	33	44.35
Scholl	297	2,177	32	33	*	33	1.4776	32	43.81
Scholl	297	2,247	31	32	*	32	1.4787	31	43.74
Scholl	297	2,322	30	31	*	31	1.4668	30	43.7
Scholl	297	2,402	29	30	*	30	1.5136	29	43.4
Scholl	297	2,488	28	29	*	29	1.4794	28	43.84
Scholl	297	2,580	27	28	*	28	1.4772	27	43.35
Scholl	297	2,680	26	27	*	27	1.5491	26	42.85
Scholl	297	2,787	25	27	*	26	1.5482	25	41.75

Note: *Indicates the literature has no corresponding results

Figure 7 Comparison with HPSO, BACO and HGA

Problem				HPSO			BACO			IGA		
Name	*n*	*C*	Optimal	Best	Gen no ofb est	Computational time	Best	Gen no ofb est	Computational time	Best	Gen no ofb est	Computational time
Mertens	7	7	5	5	1	0.21	*	*	*	5	1	0.80
Mertens	7	10	3	3	1	0.09	*	*	*	3	1	0.78
Mertens	7	18	2	2	1	0.09	*	*	*	2	1	0.73
Mitchell	21	14	8	8	1	2.74	*	*	*	8	1	0.81
Mitchell	21	26	5	5	1	2.70	*	*	*	5	1	0.80
Mitchell	21	39	3	3	1	2.82	*	*	*	3	1	0.78
Sawyer	30	25	14	14	5	8.75	*	*	*	14	1	0.87
Sawyer	30	30	12	12	1	8.88	*	*	*	12	1	0.86
Sawyer	30	41	8	8	9	8.90	8	*	0.064	8	1	0.84
Kilbrid	45	79	7	7	6	28.89	7	*	0.11	7	1	0.89
Kilbrid	45	92	6	6	24	26.57	6	*	0.096	6	1	0.87
Kilbrid	45	138	4	4	4	34.39	4	*	0.074	4	1	0.86
Tonge	70	176	21	21	24	173.97	*	*	*	21	1	1.21
Tonge	70	251	14	*	*	*	14	*	4.26	14	1	1.12
Tonge	70	320	11	*	*	*	11	*	0.21	11	1	1.07
Tonge	70	364	10	10	1	167.49	*	*	*	10	1	1.06
Tonge	70	468	8	8	1	165.93	*	*	*	8	1	1.03
Tonge	70	527	7	7	1	168.28	*	*	*	7	1	0.98
Arcus1	83	3,786	21	*	*	*	21	*	0.34	21	1	1.33
Arcus1	83	8,414	10	10	1	294.17	*	*	*	10	1	1.12
Arcus1	83	10,816	8	8	1	274.50	*	*	*	8	1	1.08
Arcus2	111	11,570	13	*	*	*	13	*	8.61	13	1	1.42
Arcus2	111	17,067	9	9	2	845.35	*	*	*	9	1	1.30
Barthold	148	403	14	*	*	*	14	*	0.01	14	1	1.95
Barthold	148	513	11	*	*	*	11	*	0.79	11	1	1.69
Barthold	148	626	9	*	*	*	9	*	0.70	9	1	1.60
Barthold	148	805	7	*	*	*	7	*	0.003	7	1	1.52
Barthold2	148	84	51	*	*	*	51	*	3.76	51	1	4.13
Barthold2	148	93	46	*	*	*	46	*	1.72	46	1	3.90
Barthold2	148	101	42	*	*	*	42	*	1.81	42	1	3.54
Barthold2	148	112	38	*	*	*	38	*	1.54	38	1	3.29
Barthold2	148	121	35	*	*	*	35	*	1.53	35	1	3.06
Barthold2	148	133	32	*	*	*	32	*	1.36	32	1	2.96
Barthold2	148	146	29	*	*	*	29	*	2.05	29	1	2.73
Barthold2	148	152	28	*	*	*	28	*	1.29	28	1	2.62
Barthold2	148	163	26	*	*	*	26	*	1.17	26	1	2.54
Barthold2	148	170	25	*	*	*	25	*	1.18	25	1	2.48
Scholl	297	1,422	50	*	*	*	50	*	0.62	50	2	59.31
Scholl	297	1,515	46	*	*	*	47	*	0.005	46	2	57.25
Scholl	297	1,834	38	*	*	*	38	*	34.51	38	1	45.79
Scholl	297	2,177	32	*	*	*	33	*	0.034	32	1	43.81
Scholl	297	2,322	30	*	*	*	31	*	0.051	30	1	43.7
Scholl	297	2,488	28	*	*	*	28	*	81.63	28	1	43.84
Scholl	297	2,680	26	*	*	*	26	*	4.17	26	1	42.85
Scholl	297	2,787	25	*	*	*	25	*	4.08	25	1	41.75

Note:*Indicates the literature has no corresponding results. Genno of best means in which generation can the best solution be obtained

Computational experience with the proposed IGA on a set of benchmark assembly line problems shows that the IGA can obtain the optimal solution. On the one hand, all individuals in each generation are effective individuals because of the user-defined function which can speed up the convergence rate of the IGA. On the other hand, it takes time to convert all the individuals to satisfy the precedence relationships during the operation of IGA which may lead to longer computational time, but the computational time is acceptable. As a result, the advantage of IGA is that it can not only obtain the optimal solution but also has faster convergence speed. The shortcoming of IGA is that it takes longer computational time.

Table I Information of APXV9R20B antenna assembly shop

Task node	Task description	Immediate successor task	t_i/s
1	Put the chassis on the assembly line and clean	2,3,5,6,7	3.4
2	Install the high frequency oscillator	10	6.96
3	Install the wall components	4,8	5.29
4	Install the input cable components	9	2.58
5	Install the high-frequency isolation	28	12.13
6	Install the low-frequency isolation	28	3.9
7	Install the high-frequency isolation block	28	4.2
8	Install low-frequency oscillator	9	10.46
9	Low-frequency cable wiring	11,14,15	6.12
10	High-frequency cable wiring	12,13	10
11	Install low-frequency ground plate	16	12.59
12	High-frequency oscillator cable welding to the oscillator	17	21.8
13	Install high-frequency ground plate	18	12.98
14	Install the low-frequency oscillator cable in place	16	9.29
15	Install low-frequency switching	16	4.38
16	Low-frequency cable welding and cleaning	19,21	28.89
17	Install the high-frequency oscillator cable in place	20	8.43
18	High-frequency cable welding and cleaning	20	22.66
19	Place the low-frequency phase shifter components	23	1.76
20	Place the high-frequency phase shifter components	22,24	1.63
21	Install low-frequency flat	23	6.14
22	Install high-frequency flat	25	4.97
23	Low-frequency plate welding	25	26.90
24	High-frequency plate welding	25	33.43
25	Install phase shift	26,27	11.63
26	Fixed the guide box and the front panel	28	5.23
27	The support installation	28	3.07
28	Install support		7.37

Table II Assembly balancing result of the proposed IGA versus PSO

IGA			PSO		
Workstation no.	Set of tasks	Total assembly time/s	Workstation no.	Set of tasks	Total assembly time/s
1	1,2,3,8,10	36.11	1	1,5,3,2,8	38.24
2	4,5,12	36.51	2	4,10,12	34.38
3	13,9,14,15	32.77	3	9,14,15,11	32.38
4	18,11	35.25	4	13,18	35.64
5	16,21,19	36.79	5	16,21,19	36.79
6	17,20,23	36.96	6	17,20,23	36.96
7	7,24	37.63	7	24,22	38.40
8	6,22,25,26,27,28	36.17	8	7,25,6,27,26,28	35.40

Table III Performance comparison of IGA and PSO

Algorithm	m	η(%)	SI
IGA	8	93.81	*2.7538*
PSO	8	93.81	3.0354

Note: η represents the balance rate, $\eta = \frac{\sum_{t=1}^{n} t_i}{C \cdot m}$

Future research should include the following:

- The algorithm should be extended to solve other types' ALBPs, such as two-sided, U-type and mixed-model assembly line.
- Other artificial intelligence algorithm should be developed to solve the SALBP-1, such as Bayesian algorithm.

References

Battaia, O. and Dolgui, A. (2013), "A taxonomy of line balancing problems and their solution approaches", *International Journal of Production Economics*, Vol. 142 No. 2, pp. 259-277.

Bautista, J. and Pereira, J. (2009), "A dynamic programming based heuristic for the assembly line balancing problem", *European Journal of Operational Research*, Vol. 194 No. 3, pp. 787-794.

Blum, C. (2008), "Beam-ACO for simple assembly line balancing", *Informs Journal on Computing*, Vol. 20 No. 4, pp. 618-627.

Boysen, N., Fliedner, M. and Scholl, A. (2008), "Assembly line balancing: which model to use when?", *International Journal of Production Economics*, Vol. 111 No. 2, pp. 509-528.

Chang, P.C., Huang, W.H. and Ting, C.J. (2011), "A hybrid genetic-immune algorithm with improved lifespan and elite antigen for flow-shop scheduling problems", *International Journal of Production Research*, Vol. 49 No. 17, pp. 5207-5230.

Chen, B.B., Gao, S.C., Wang, S.Q. and Aorigele, B. (2014), "Adaptive immune-genetic algorithm for fuzzy job shop scheduling problems", International Conference in Swarm Intelligence, Vol. 8794, pp. 246-257.

Dou, J.P., Su, C. and Li, J. (2012), "Discrete particle swarm optimization algorithm for assembly line balancing problems of type 1", *Computer Integrated Manufacturing Systems*, Vol. 18 No. 5, pp. 1021-1030.

Dou, J.P., Li, J. and Su, C. (2013), "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 Nos 9/12, pp. 2445-2457.

Easton, F., Faaland, B., Klastorn, T.D. and Schmitt, T. (1989), "Improved network based algorithms for the assembly line balancing problem", *International Journal of Production Research*, Vol. 27 No. 11, pp. 1901-1915.

Fleszar, K. and Hindi, K.S. (2003), "An enumerative heuristic and reduction methods for the assembly line balancing problem", *European Journal of Operational Research*, Vol. 145 No. 3, pp. 606-620.

Ghosh, T., Doloi, B. and Dan, P.K. (2016), "An immune genetic algorithm for inter-cell layout problem in cellular manufacturing system", *Production Engineering*, Vol. 10 No. 2, pp. 157-174.

Goncalves, J.F. and de Almeida, J.R. (2002), "A hybrid genetic algorithm for assembly line balancing", *Journal of Heuristics*, Vol. 8 No. 6, pp. 629-642.

Guden, H. and Meral, S. (2013), "An adaptive simulated annealing method for type-one simple assembly line balancing: a real life case study", *Journal of the Faculty of Engineering and Architecture of Gazi University*, Vol. 28 No. 4, pp. 897-908.

Hackman, S.T., Magazine, M.J. and Wee, T.S. (1989), "Fast, effective algorithms for simple assembly line balancing problems", *Operations Research*, Vol. 37 No. 6, pp. 916-924.

Helgeson, W.P. and Birnie, D.P. (1961), "Assembly line balancing using the ranked positional weight technique", *Journal of Industrial Engineering*, Vol. 12 No. 6, pp. 394-398.

Hoffmann, T.R. (1963), "Assembly line balancing with a precedence matrix", *Management Science*, Vol. 9 No. 4, pp. 551-562.

Hu, X.M., Zhang, Y.Y., Zeng, N. and Wang, D. (2014), "A novel assembly line balancing method based on PSO algorithm", *Mathematical Problems in Engineering*, Vol. 2014, pp. 1-10.

Kilincci, O. and Bayhan, G.M. (2008), "A P-invariant-based algorithm for simple assembly line balancing problem of type-1", *The International Journal of Advanced Manufacturing Technology*, Vol. 37 Nos 3/4, pp. 400-409.

Lapierre, S.D., Ruiz, A. and Soriano, P. (2006), "Balancing assembly lines with tabu search", *European Journal of Operational Research*, Vol. 168 No. 3, pp. 826-837.

Li, M., Li, S., Xia, X.H., Tang, Q.H. and Zheng, Q.X. (2013), "Rules combination algorithm of assembly line balancing problem for large-scale multiple stations", *Computer Integrated Manufacturing Systems*, Vol. 19 No. 11, pp. 2780-2787.

Liu, S.B., Ng, K.M. and Ong, H.L. (2008), "Branch-and-bound algorithms for simple assembly line balancing problem", *The International Journal of Advanced Manufacturing Technology*, Vol. 36 Nos 1/2, pp. 169-177.

Mz, M. (2010), "Soft computing in optimizing assembly lines balancing", *Journal of Computer Science*, Vol. 6 No. 2, pp. 141-162.

Otto, A. and Otto, C. (2014), "How to design effective priority rules: example of simple assembly line balancing", *Computers & Industrial Engineering*, Vol. 69, pp. 43-52.

Peeters, M. and Degraeve, Z. (2006), "An linear programming based lower bound for the simple assembly line balancing problem", *European Journal of Operational Research*, Vol. 168 No. 3, pp. 716-731.

Pitakaso, R. (2015), "Differential evolution algorithm for simple assembly line balancing type 1 (SALBP-1)", *Journal of Industrial and Production Engineering*, Vol. 32 No. 2, pp. 104-114.

Ponnambalam, S.G., Aravindan, P. and Naidu, G.M. (2000), "A multi-objective genetic algorithm for solving assembly line balancing problem", *International Journal of Advanced Manufacturing Technology*, Vol. 16 No. 5, pp. 341-352.

Sabuncuoglu, I., Erel, E. and Tanyer, M. (2000), "Assembly line balancing using genetic algorithms", *Journal of intelligent Manufacturing*, Vol. 11 No. 3, pp. 295-310.

Salveson, M.E. (1995), "The assembly line balancing problem", *The Journal of Industrial Engineering*, Vol. 6 No. 3, pp. 18-25.

Scholl, A. and Becker, C. (2006), "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.

Sewell, E.C. and Jacobson, S.H. (2012), "A branch, bound, and remember algorithm for the simple assembly line balancing problem", *Informs Journal on Computing*, Vol. 24 No. 3, pp. 433-442.

Sivasankaran, P. and Shahabudeen, P. (2014), "Literature review of assembly line balancing problems", *The International Journal of Advanced Manufacturing Technology*, Vol. 73 Nos 9/12, pp. 1665-1694.

Sprecher, A. (1999), "Competitive branch-and-bound algorithm for the simple assembly line balancing problem", *International Journal of Production Research*, Vol. 37 No. 8, pp. 1787-1816.

Sulaiman, M., Choo, Y.H. and Chong, K.E. (2011), *Ant Colony Optimization with Look Forward Ant in Solving Assembly Line Balancing Problem*, IEEE, New York, NY, pp. 115-121.

Tasan, S.O. and Tunali, S. (2008), "A review of the current applications of genetic algorithms in assembly line balancing", *Journal of intelligent Manufacturing*, Vol. 19 No. 1, pp. 49-69.

Vila, M. and Pereira, J. (2013), "An enumeration procedure for the assembly line balancing problem based on branching by non-decreasing idle time", *European Journal of Operational Research*, Vol. 229 No. 1, pp. 106-113.

Wang, X.J., Gao, L., Zhang, C.Y. and Shao, X.Y. (2010), "A multi-objective genetic algorithm based on immune and entropy principle for flexible job-shop scheduling problem", *The International Journal of Advanced Manufacturing Technology*, Vol. 51 Nos 5/8, pp. 757-767.

Xu, X.D. and Li, C.X. (2007), "Research on immune genetic algorithm for solving the job-shop scheduling problem", *The International Journal of Advanced Manufacturing Technology*, Vol. 34 Nos 7/8, pp. 783-789.

Yeh, D.H. and Kao, H.H. (2009), "A new bidirectional heuristic for the assembly line balancing problem", *Computers & Industrial Engineering*, Vol. 57 No. 4, pp. 1155-1160.

Yolmeh, A. and Kianfar, F. (2012), "An efficient hybrid genetic algorithm to solve assembly line balancing problem with sequence-dependent setup times", *Computers and Industrial Engineering*, Vol. 62 No. 4, pp. 936-945.

Yu, J.F. and Yin, Y.H. (2010), "Assembly line balancing based on an adaptive genetic algorithm", *The International Journal of Advanced Manufacturing Technology*, Vol. 48 Nos 1/4, pp. 347-354.

Zhang, H.Y. (2017), "An improved immune algorithm for simple assembly line balancing problem of type 1", *Journal of Algorithms & Computational Technology*, Vol. 11 No. 4, pp. 317-326.

Further reading

Rashid, M., Hutabarat, W. and Tiwari, A. (2012), "A review on assembly sequence planning and assembly line balancing optimisation using soft computing approaches", *The International Journal of Advanced Manufacturing Technology*, Vol. 59 Nos 1/4, pp. 335-349.

Corresponding author

Han-ye Zhang can be contacted at: tzgjzhy@163.com