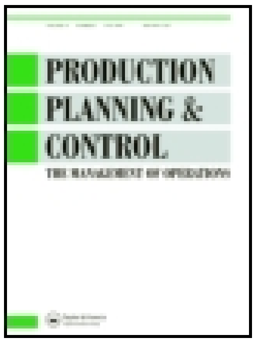

Production Planning & Control

ISSN: 0953-7287 (Print) 1366-5871 (Online) Journal homepage: https://www.tandfonline.com/loi/tppc20

Two-sided assembly line balancing: A genetic algorithm approach

Yeo Keun Kim , Yeongho Kim & Yong Ju Kim

To cite this article: Yeo Keun Kim , Yeongho Kim & Yong Ju Kim (2000) Two-sided assembly line balancing: A genetic algorithm approach, Production Planning & Control, 11:1, 44-53, DOI: 10.1080/095372800232478

To link to this article: https://doi.org/10.1080/095372800232478

Published online: 15 Nov 2010.

Two-sided assembly line balancing: a genetic algorithm approach

YEO KEUN KIM, YEONGHO KIM and YONG JU KIM

Keywords assembly line balancing, genetic algorithm, two-sided assembly lines, genetic operators, adaptation

Abstract. A two-sided assembly line balancing problem is typically found in plants producing large-sized high-volume products, e.g. buses and trucks. The features specific to the assembly line are described in this paper, which are associated with those of: (i) two-sided assembly lines; (ii) positional constraints; and (iii) balancing at the operational time. There exists a large amount of literature in the area of line balancing, whereby it has mostly dealt with one-sided assembly lines. A new genetic algorithm is developed to solve the problem, and its applicability and extensibility are discussed. A genetic encoding and decoding scheme, and genetic operators suitable for the problem are devised. This is particularly emphasized using problem-specific information to enhance the performance of the genetic algorithm (GA). The proposed GA has a strength

Authors: Yeo Keun Kim and Yong Ju Kim, Department of Industrial Engineering, College of Engineering, Chonnam National University, Kwangju 500-757, Republic of Korea, E-mail: kimyk@chonnam.chonnam.ac.kr, and Yeongho Kim, Department of Industrial Engineering, Seoul National University, Seoul 151-742, Republic of Korea.

YEO KEUN KIM is a Professor in the Department of Industrial Engineering at Chonnam National University at Kwangju, Korea. He received his BS, MS and PhD degrees in Industrial Engineering from Seoul National University at Seoul, Korea. He teaches courses in operations research, optimization theory, manufacturing systems and evolutionary algorithms. His areas of current interest are evolutionary algorithms, taboo search, simulated annealing and mathematical programming. His research topics include production planning and scheduling, assembly systems analysis and design, sequencing problems, and telecommunication network analysis. He has published articles in refereed national and international journals. He has been a referee for *Computers and Operations Research, Computers and Industrial Engineering,* and *European Journal of Operational Research.*

YEONGHO KIM is Assistant Professor at the Department of Industrial Engineering at Seoul National University. He received his PhD degree from North Carolina State University, and BS and MS degrees from Seoul National University. He is interested in the areas of manufacturing systems, AI applications in manufacturing, concurrent engineering and collaborative design, and Internet applications. His research topics cover a wide variety of areas including real-time assembly line scheduling, workflow management systems, distributed-open-integrated support of development processes, medical information systems, and computer supported collaborative work on the Internet. He is a member of the Editorial Board of *International Journal of Industrial Engineering—Applications and Practice,* and a member of the Editorial Board of *International Journal of Management Systems.*

YONG JU KIM has received BS, MS and PhD degrees in Industrial Engineering from Chonnam National University at Kwangju, Korea. He is a lecturer at the Department of Industrial Engineering at Dongshin University, Naju, Chollanamdo, Korea. He teaches computer programming, database management systems and CAD/CAM. His current interests are optimization theory, evolutionary computation, neural network, production planning and scheduling, computer-integrated manufacturing, and information engineering. He has published articles in national and international journals.

that it is flexible in solving various types of assembly line balancing problems. An experiment is carried out to verify the performance of the GA, and the results are reported.

1. Introduction

We consider an assembly line balancing problem which is typically found in plants producing large-sized high-volume products, e.g. buses and trucks. The assembly line has several features that are distinguished from those considered in traditional line balancing problems. First, lines to produce large-sized products are often two sided (left and right). In most previous approaches, however, it is assumed that only one side of the line is utilized. Secondly, an assembly line for large-sized products generally involves various types of special facilities, each of which is usually dedicated to a certain task. This restriction, commonly known as positional constraint, is not significantly considered in traditional approaches. Thirdly, once a line is designed and installed, the length of the line or the maximum number of workstations does not change when operating it in the real world. Most 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 the operational time. A new method of line balancing is proposed to cope with the above features of two-sided assembly lines.

A body of literature is available on assembly line balancing (ALB). The problems in general fall into an NP-hard class of combinatorial optimization problems (Gutjahr and Nemhauser 1964). This becomes a major reason why many previous researches, including Talbot *et al.* (1986), and Ghosh and Gagnon (1989), have proposed heuristic approaches. Some authors, including Arcus (1963), Johnson (1983), and Gunther *et al.* (1983), especially focus on dealing with positional constraints. A group of other researchers recognize that genetic algorithms have a great potential to ALB problems (Anderson and Ferris 1994, Leu *et al.* 1994, Suresh *et al.* 1996, Kim *et al.* 1998a). However, all these approaches have invariably dealt with one-sided assembly lines. Although two-sided assembly line balancing (two-ALB) problems are often encountered in the real world, little attention has been paid to the problems. To the best of our knowledge, Bartholdi (1993) is the first to address two-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 to build solutions quickly and incrementally.

We develop a new algorithm to solve two-ALB problems using a genetic algorithm (GA). GAs have been proven to be very efficient and powerful in a wide variety of applications (Goldberg 1989), particularly in combinatorial optimization problems. Another advantage is that GA has a strength that it is flexible in dealing with various types of optimization criteria and constraints. The distinct features of two-sided lines mentioned earlier add an additional dimension to the complexity of conventional ALB problems. We believe that GA is a proper strategy for solving the two-ALB problems. Not only does GA find good quality solutions quickly to such complex problems, but it is able to readily deal with constraints imposed on by the features of two-sided lines. In order to develop a new GA, a genetic encoding and decoding scheme, and genetic operators suitable for the problem are devised. The applicability of the GA is discussed, and extensive experiments are carried out to verify its performance.

2. Backgrounds

2.1. *Two-sided assembly line balancing*

In a two-sided assembly line, different assembly tasks are performed on the same product item in parallel at both sides of the line. A two-sided assembly line is illustrated in figure 1. A pair of two directly facing workstations, e.g. 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 provide several 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 (Bartholdi 1993).

Two-ALB is mainly distinguished from traditional one-sided assembly line balancing (one-ALB) in that tasks have restrictions on the operation directions. Some assembly operations prefer one of the two sides, while others can be performed at either side of the line. That is, the tasks are classified into three types: L (left); R

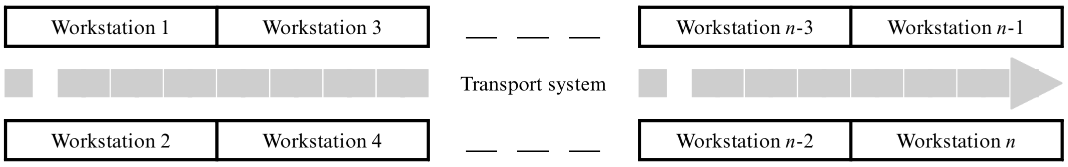

Figure 1. Two-sided assembly line.

(right); and E (either) type tasks. For example, in a truck assembly line, such tasks as installing fuel tanks, air filters and tool boxes are L-type tasks because these can be more easily performed at the left-hand side of the line; meanwhile mounting batteries, air tanks and mufflers are usually carried out at 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. Considering the operation directions is important to maximize the productivity of the assembly line. This is also important to the line configuration by assisting a better plan for laying out facilities and placing tools and fixtures.

The consideration of operation directions changes the traditional way of handling precedence and cycle time restrictions. Idle time is sometimes unavoidable even between tasks assigned to the same workstation. Consider, e.g. two tasks i and p such that p is an immediate predecessor of i. Suppose that i is assigned to workstation j and p to the companion of j. A worker at j cannot begin to work on task i unless task p is completed. Therefore, balancing the line needs to take into account the sequence-dependent finish time of tasks, unlike a one-sided assembly line. This notion of sequence dependency requires careful treatment for cycle time restriction.

The two-ALB problem considered in this paper is described as follows. The objective is to minimize the number of workstations to which tasks are allocated. It is assumed that the cycle time is predetermined, which is generally accepted in the literature (Ghosh and Gagnon 1989). Also given is the line configuration including the maximum number of available workstations and the location of facilities. Once the configuration is set up at the design time, it is usually maintained unless product design is significantly changed or the amount of demand is greatly varied. Also considered in this paper are positional constraints due to facility layout. A positional constraint states that a task must be assigned to a predetermined workstation. The constraints are particularly important for two-ALB because large, heavy facilities are very frequently involved in the line and their locations are permanently fixed. For example, once facilities for mounting tyres, cabs and engines in a truck assembly line are installed, then they are not usually relocated.

The precedence diagram shown in figure 2 illustrates an example two-ALB problem. A circle indicates a task. Each task is associated with a label of (t_i, d), where t_i is the i-th task processing time and d (= L, R or E) denotes the preferred operation direction. Tasks having positional constraints are enclosed in rectangles, and the predetermined workstations are shown in small rectangles.

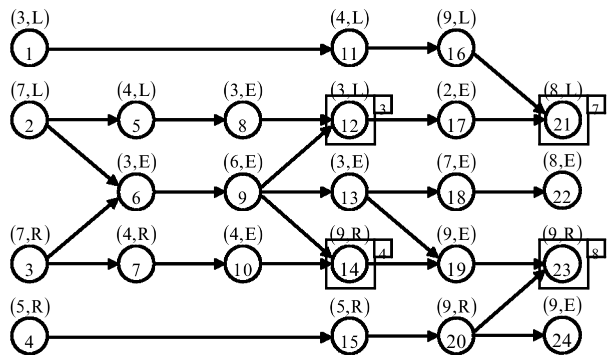

Figure 2. Precedence diagram.

2.2. *Genetic algorithms*

Genetic algorithms are a stochastic procedure which imitates the biological evolutionary process of genetic inheritance and the survival of the fittest. Although there are plenty of previous researches that use the algorithms, only the underlying GA framework adopted in this paper is presented due to space limitation. Those readers who are further interested in GA may refer to Goldberg (1989). The notation used in the GA procedure is first described.

$P(t)$	the t-th generation population ($t = 0, 1, 2, \dots$)
N_p	population size which is the number of individuals maintained in a population
P_c	crossover rate which is the proportion of individuals chosen for crossover
P_m	individual mutation rate which is the proportion of individuals chosen for mutation
P_g	gene mutation rate which is the proportion of genes actually mutated

The overall framework of the GA procedure is provided below.

Step 1. (Initial population) Let $t = 0$. Form the initial population $P(0)$.

Step 2. (Evaluation) Evaluate each individual in $P(0)$.

Step 3. (Selection) Select N_p individuals from $P(t)$. Copy them to $P(t+1)$.

Step 4. (Crossover) Randomly choose $0.5 \times P_c \times N_p$ pairs of parents in $P(t+1)$ for crossover. Apply crossover operation to each pair of parents to obtain $P_c \times N_p$ offspring. Replace the parents with the offspring in $P(t+1)$.

Step 5. (Mutation) Randomly choose $P_m \times N_p$ parents in $P(t+1)$ and apply mutation operation to the individuals according to the gene mutation rate of P_g.

Step 6. (Evaluation) Evaluate each individual in $P(t+1)$.

Step 7. (Elitist strategy) If the best individual in $P(t)$ is better than the best individual in $P(t+1)$, replace the worst individual in $P(t+1)$ with the best in $P(t)$.

Step 8. Let $t \quad t+1$. If the stopping criteria are met, then stop. Otherwise, go to step 3.

3. Genetic algorithm for two-ALB

A new genetic algorithm is developed. Important steps of the GA procedure provided in the previous section are further detailed. The method of using problem-specific information is particularly emphasized.

We first define the notation used in the genetic algorithm as follows.

CT	cycle time
R	the set of tasks which are not assigned yet or need to be reassigned (marked tasks)
$IP(i)$	the set of tasks immediately preceding task i
$S(i)$	the set of tasks following task i
$NR(i)$	$R \cap IP(i)$ (i.e. the set of tasks immediately preceding task i in R)
E_i	the latest workstation among those which the tasks in $IP(i)$ are assigned to. If $IP(i) = \varnothing$, then $E_i = 1$.
L_i	the earliest workstation among those which the tasks in $S(i) - R$ are assigned to. If $S(i) - R = \varnothing$, then L_i is set to the last workstation.
T_j	the total processing time of tasks assigned to workstation j
F_j	the completion time of workstation j
$AW(i)$	the set of workstations to which task i can be assigned, which is identified as follows: $AW(i) = \{2e-1, 2e+1, \ldots, 2l-1\}$ if i is an L-type task, $AW(i) = \{2e, 2e+2, \ldots, 2l\}$ if i is an R-type task, $AW(i) = \{2e-1, 2e, 2e+1, \ldots, 2l\}$ if i is an E-type task, where

$$e = \left[\frac{E_i}{2}\right]^+, \quad l = \left[\frac{L_i}{2}\right]^+$$

and $[x]^+$ is the smallest integer greater than or equal to x.

3.1. *Encoding*

In a genetic algorithm, an individual is an encoding of a potential solution. We use a 'group-number' encoding which allows the interpretation to be straightforward. An individual is a string of length m (the number of tasks), each element of which is an integer between 1 and n (the maximal number of workstations). If the i-th element is j, it represents that task i is assigned to workstation j. An example of group-number encoding for two-ALB is provided below.

Consider the precedence diagram in figure 2 and an assignment of tasks in table 1. Assume that the cycle time is 21, and the maximal number of workstations is eight.

The above assignment can be encoded by the following individual.

$$(3\ 1\ 2\ 2\ 1\ 1\ 2\ 4\ 1\ 3\ 7\ \underline{3}\ 3\ \underline{4}\ 2\ 7\ 5\ 3\ 5\ 4\ \underline{7}\ 5\ \underline{8}\ 8)$$

In this case, no worker is assigned to workstation 6. An underlined number indicates that the task assignment is predetermined by positional constraints.

With the group-number encoding, it is very natural to represent positional constraints as well as task assignment, so that task assignment to workstations is readily identified. This can be very useful when applying genetic operators addressed later in this paper.

3.2. *Decoding*

Decoding is a procedure to interpret an encoded individual into an actual task assignment. We need to determine the optimal sequence to each individual. An enumeration method, e.g. a branch-and-bound method may be used, but it requires a larger amount of computation time than the main genetic algorithm.

In this paper, a heuristic decoding method is developed. The technique uses a weight similar to the classical ranked positional weight (Helgeson and Birnie 1961), which is the sum of processing times for a task and all of its successors. In this paper, the weight computation is modified as follows. We first need to define the term

Table 1. Example of encoding.

Tasks assigned to left workstation	2, 5, 6, 9	10, 13, 18, 1, $\underline{12}$	19, 22, 17	11, 16, $\underline{21}$
Left workstation #	1	3	5	7
Mated-station #	1	2	3	4
Right workstation #	2	4	6	8
Tasks assigned to right workstation	3, 4, 15, 7	8, 20, $\underline{14}$		24, $\underline{23}$

'critical task'. A critical task is such a task that at least one of its immediate successors is assigned to the companion of the workstation to which the task is assigned. A critical task can cause delay in the operation of its immediate successors. Therefore, the critical task and its predecessors need to be sequenced as early as possible. For a critical task, a sufficiently large positive number is used for weight computation rather than its processing time. This gives a higher priority to the critical task and its predecessors. The weight computed in this manner is called 'critically ranked positional weight' (CRPW).

The overall procedure of the heuristic is as follows. For a mated-station, first find the set of such tasks that all of their immediate predecessors have already been sequenced. Among the tasks, select and sequence the task that has the earliest starting time. When ties occur, the one with the maximum CRPW is selected. This process is repeated until all the tasks in the mated-station are sequenced. The process is terminated when there is no critical task. The remaining tasks can be sequenced in any order if not violating the precedence relations. This procedure is applied to every mated-station. The heuristic provides a unique decoding for every individual.

3.3. *Initial population*

GA operates on a set of individuals. We develop a procedure to generate the initial population for two-ALB. In the procedure, we incorporate problem-specific information, e.g. precedence relations and workload of each workstation.

The procedure is as follows. An empty string of length m is created and each position is marked with * which means that the task is not assigned to any workstation yet. Then, tasks associated with positional constraints are first allocated to the predetermined workstations. For the example shown in figure 2, because tasks 12, 14, 21 and 23 are fixed to workstations 3, 4, 7 and 8, respectively, allocating the tasks to their relevant workstations gives the string of (* * * * * * * * * * * 3 * 4 * * * * * * 7 * 8 *).

Among those marked with *, tasks whose immediate predecessors are all assigned are identified. Assigning any of them does not violate precedence restrictions. Now, a task is selected from the candidate tasks using a single-pass heuristic rule which is randomly chosen among the following ones: (i) maximum ranked positional weight; (ii) maximum average ranked positional weight; (iii) maximum total number of follower tasks; (iv) maximum task time; (v) random. These heuristic rules are explained in Talbot *et al.* (1986). It is generally known that GA tends to show better performance with diverse individuals than biased ones. This indicates that, while forming the initial population, it is important to generate as many disparate individuals as possible. The use of single-pass heuristic rules is impregnated with this conception. Assigning a task that has a larger number of succeeding tasks would provide a greater diversity in future assignment. The first three rules have the common rationale that they calculate a large weight for a task that would bring about greater randomness. Let the selected task be i. The task, i, is allocated to the workstation that has the smallest workload among those of $AW(i)$. A tie can be broken by choosing the earliest workstation.

Filling in all the * marked positions in this manner creates one string, and the initial population is produced by repeating the procedure 'population size' times.

3.4. *Evaluation function and selection*

An evaluation function determines the fitness of potential solutions. This measure is used to select the more fit individuals for the next generation so that they have a chance to pass on their good characteristics to offspring. The following evaluation function is used in this research.

$$\text{Eval} = \sum_{j \in \mathcal{J}} WS_j,$$

where $\mathcal{J}$ is the set of workstations, and

$$WS_j = \begin{cases} 0, & \text{if } F_j = 0, \\ 1, & \text{if } 0 < F_j \le CT, \\ 1 + (F_j/CT + 1), & \text{if } F_j > CT. \end{cases}$$

An individual should be decoded prior to the evaluation. The function basically counts the number of workstations to which tasks are assigned. However, some individuals may turn out to be infeasible: i.e. the cycle time constraints are violated. Upon a workstation bearing the violation, a penalty of $F_j/CT + 1$ is imposed, so that the individual has a relatively high probability of disappearing at the next generation.

Among many selection techniques, we use a 'tournament' selection employed in messy genetic algorithms (Goldberg *et al.* 1989). Given a population, we first form a random permutation of the whole population. The permutation is divided into blocks of k (called tournament size) individuals. The best individual is selected from each block, and included in the next generation. If all the blocks are exhausted, a new permutation is formed. The process continues until the desired population size is obtained. Additionally, the 'elitist strategy' (Goldberg 1989), which keeps the best individual found so far, is used.

3.5. *Genetic operators*

Genetic algorithms in general employ two types of operators: crossover and mutation. A new crossover operator, called structured one-point crossover (SOX), is developed for two-ALB. This operator is modified from structural crossover proposed by von Laszewski (1991) for partitioning problems. The procedure of the SOX operator is as follows.

Step 1. Set r to an integer randomly generated from $[1, [(n-1)/2]^{-}]$, where $[x]^{-}$ is the largest integer less than or equal to x.
Step 2. The genes representing workstations $1-2r$ in parent P1 are passed on to the same positions in offspring O1. The remaining positions are copied from those of $2r+1$ to n in parent P2.
Step 3. Mark all the remaining empty positions with *.
Step 4. Compute the workload of each workstation. If the workload of a workstation is smaller than a $\alpha(\leq CT/2)$, then all the tasks assigned to the workstation are also * marked.
Step 5. To reassign the * marked positions, use the 'adaptation' procedure in section 3.6.

Another offspring, O2, can be produced by exchanging the role of parents P1 and P2.

The most important role of crossover operators is to transmit good characteristics of parents to offspring. Steps 1 and 2 partly provide an underlying basis for this role. A crossover point is chosen on the mated-station basis. That is, only an even number can be a crossover point. This is to prevent good genetic characteristics of parents from being destroyed during crossover operations. Also notice that precedence constraints and positional constraints are not violated by the crossover operation. For all the tasks assigned to $1, 2, \ldots 2r$ in P1 and those assigned to $2r, 2r+1, \ldots n$ in P2, the precedence constraints are satisfied, and each gene associated with a positional constraint is also preserved in the offspring.

In step 3, undetermined positions are all * marked. A task corresponding to such a position has been assigned to one of $[2r+1, n]$ in P1 and $[1, 2r]$ in P2. In step 4, some positions that have inherited certain tasks are switched into undetermined and * marked. If the workload of a workstation is small, then there exists a high chance that the number of workstations can be decreased by reassigning the tasks to other workstations. This can enhance the probability that the crossover produces high-quality individuals. The parameter α can be set by users. (In this study, $\alpha = CT/2$.) In step 5, the offspring produced undergoes the adaptation step aiming at improving the fitness while maintaining the precedence constraints. This is addressed in the next section.

$$\begin{array}{lc*{24}{c}c}
\mathrm{P1} = & (& 3&1&2&2&1&1&2&4&1&3&5&3&5&4&2&5&6&6&6&4&7&8&8&7 &) \\
 & & \downarrow&\downarrow&\downarrow&\downarrow&\downarrow&\downarrow&\downarrow&\downarrow&\downarrow&\downarrow& &\downarrow& &\downarrow&\downarrow& & & & &\downarrow& & & & & \\
\mathrm{O1} = & (& 3&1&2&2&1&1&2&4&1&3&7&3&*&4&2&7&*&*&*&4&7&5&8&8 &) \\
 & & & & & & & & & & & &\uparrow& & & & &\uparrow& & & & &\uparrow&\uparrow&\uparrow&\uparrow & \\
\mathrm{P2} = & (& 1&1&2&2&1&1&2&4&3&2&7&3&3&4&6&7&3&3&4&6&7&5&8&8 &)
\end{array}$$

Figure 3. Crossover example.

Consider parents P1 and P2 shown in figure 3. Suppose that the randomly generated cutover point is 2. Step 4 is skipped because no workstation meets the condition of the step. The resulting offspring is shown in figure 3.

A mutation operator acts on a single parent and produces an offspring by introducing small changes. The mutation is generally used to ensure the diversity of potential solutions and to prevent a premature convergence to local optima. The mutation operation used in this paper is as follows. An individual is first selected for mutation with a rate of P_{m}. Then, for the selected individual, it is decided for every gene except fixed position genes if it is to be changed with the rate of P_{g}. All the genes that are determined to undergo the mutation operation are marked with *, and the adaptation procedure in the next section is invoked to reassign new workstations.

3.6. *Adaptation procedure*

The genetic crossover and mutation operations described in the previous section are not complete because some positions of the resulting offspring are undetermined yet. An adaptation procedure is provided to fill up the positions that are * marked.

Step 1. Set R to contain all the marked tasks.
Step 2. Identify the set of candidate tasks such that $R' = \{i | i \in R \text{ and } NR(i) = \emptyset\}$.
Step 3. Among the tasks in R', select the tasks that have the maximum ranked positional weight. A tie is broken by random selection. Let i^* denote the selected task.
Step 4. Calculate E_{i^*} and L_{i^*}, and identify $AW(i^*)$.
Step 5. Compute F_j for each workstation $j \in AW(i^*)$, and determine a workstation j^* to which task i^* is assigned. There are two cases as follows:
  (i) If $A = \{j | T_j > 0$ and $F_j \leq CT$, $j \in AW(i^*)\} \neq \emptyset$, $j' = \min_{j \in A} j$. If i^* is an E-type task, j' is an odd number, and $T_{j+1} > 0$, then

$$j^* = \begin{cases} j', & F_{j'} \leq F_{j'+1} \\ j'+1, & F_{j'} > F_{j'+1} \end{cases}$$

  Otherwise, $j^* = j'$.

(ii) If, for all $j \in AW(i^*)$, $T_j = 0$ or $F_j > CT$, then the workstation having the smallest F_j becomes j^*. Ties are broken by selecting the smallest workstation ID.

Step 6. Assign task i^* to workstation j^*, and remove i^* from R. If R is empty, then stop; otherwise, go to step 2.

Step 5 plays the key role in the adaptation procedure. The step determines a workstation to which i^*, the task selected in step 3, will be assigned. To compute the finish time F_j, the decoding procedure explained earlier is applied only with the newly assigned task i^* and tasks that have already been assigned. T_j, the total processing time of tasks that are assigned to workstation j, does not include the task processing time of task i^*. First of all, consider that the candidates are such workstations that other tasks have already been assigned. A workstation having no task assignment is excluded from consideration. This is intended to minimize the number of workstations by skipping such workstations without assigning any task. Otherwise, only a small amount of workload would be allocated to some workstations. Consider the encoding example in table 1. Traditional heuristic rules will allocate task 24 to workstation 6, but this apparently increases the number of workstations. Assigning the task to workstation 8 by skipping workstation 6 can reduce the number of workstations. Though it seems to be a little *ad hoc*, it has not been noted in any literature. Although such an assignment could be very useful for line balancing at the operational time, no proper method is suggested yet. Indeed, mathematical programming, e.g. integer and goal programming, can represent the problem, but its use is limited to only small-sized problems. We provide a simple way which allows for skipping workstations without assigning any tasks. This can be an important advantage of the proposed method, in particular for lines involving many dedicated facilities.

Applying the adaptation procedure to the example in section 3.5 will give the following offspring

O1 = (3 1 2 2 1 1 2 4 1 3 7 3 3 4 2 7 5 3 5 4 7 5 8 8)

4. Experiments

The search capability of the proposed algorithm was analysed with five test-bed problems. Two relatively small-sized problems (P9 and P12) which, respectively, have nine and 12 tasks are shown in figure 4(a) and (b). A medium-sized problem having 24 tasks (P24) is presented in figure 2. Two large-sized practical problems (P65 and P148) are also used in the experiment. P65 is formulated by the authors of this paper for a truck assembly line of AAA automotive company. The problem can be found in Lee *et al.* (1997). P148 is constructed by Bartholdi (1993). In P148, the processing times of tasks 79 and 108 are much larger than those of others. Because these impose a limit on cycle time, the processing times are changed from 2.81 to 1.11 and from 3.83 to 0.43. Table 2 provides the settings of test-bed problems.

To run a genetic algorithm, it requires such parameters as population size, crossover and mutation rates, and termination criterion. The parameters are determined through a set of preliminary experiments. For every problem, the crossover rate is set to 0.5, the individual mutation rate is 0.5 and the gene mutation rate is 0.1. The tournament size is fixed at 2. The population size is varied with the problem size. The larger the problem size, the larger the population size. For the termination criterion, we used the maximal number of generations. As the problem size increases, the maximal number of generations is also increased. For every problem instance, the algorithm is applied 10 times, and the best solution is collected for each run. The average and standard deviation of the 10 solutions are reported. All the experiments are carried out on an IBM Pentium PC (CPU 166 MHz).

First, the performance of the proposed GA is compared to that of integer programming (Kim *et al.* 1998b) that can find the exact optimal solution. This comparison is performed on the small-sized problems (P9 and P12). The integer programming is solved using the CPLEX package. For GA, the population is fixed to 20, and the procedure is repeated for 50 generations. The fourth

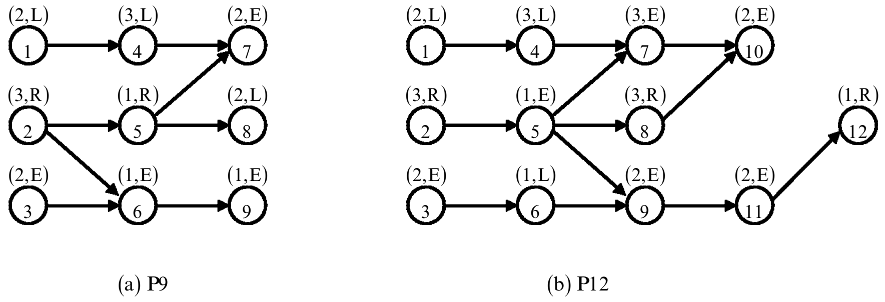

Figure 4. Small-sized test-bed problems: (a) P9 and (b) P12.

Table 2. Maximal number of workstations and positional constraints.

Problem	Maximal no. of workstations	Positional constraints: predetermined pairs of [workstation, task(s)]
P9	6	[3, 4], [4, 5]
P12	6	[3, 4], [4, 8]
	8	[3, 4], [6, 8]
P24	8	[3, 12], [4, 14], [7, 21], [8, 23]
P65	20	[5, (9, 10)], [6, (7, 8)], [15, (39, 58)], [16, (40, 55)]
P148	32	[5, (15, 18)], [6, (16, 19)], [7, 141], [9, 145], [10, 142], [12, 144], [17, (45, 46, 47)], [18, (38, 39, 40)], [23, (54, 133)], [24, (72, 134)]

Table 3. Performance comparison between GA and integer programming.

			Integer programming		Proposed GA		
Problem	# of stations	Cycle time	Optimal solution	CPU time	Average	SD	CPU time
P9	6	3	6	0.4	6	0.0	0.2
		4	5	1.4	5	0.0	0.2
		5	4	4.2	4	0.0	0.2
		6	3	6.0	3	0.0	0.2
P12	6	5	6	540.8	6	0.0	0.3
		6	5	347.8	5	0.0	0.3
		7	4	1274.9	4	0.0	0.3
	8	4	8	35.2	8	0.0	0.3
		5	6	461.3	6	0.0	0.3
		6	5	3315.6	5	0.0	0.3
		7	4	3501.1	4	0.0	0.3

and sixth columns of table 3 present the optimal solutions found by integer programming and the average of GA solutions, respectively. The fifth and eighth columns provide the CPU time required by each approach. For the small-sized problems, every run of GA finds the optimal solution. The power of GA can be clearly recognized from the fact that the computation time required is much smaller than that for integer programming.

For problems P24, P65 and P148, the integer programming could not find the optimal solutions within some reasonable amount of time. Thus, we compared the GA with the first-fit rule (FFR) that Bartholdi (1993) adopted for an interactive computer program. We modified it and implemented for a non-interactive version. In our experiment, 50 000 solutions are generated using the FFR heuristic. The 10 best solutions are selected, and their average, standard deviation and ratio of feasible solutions are presented. For the proposed GA, the population size is set to 50, and the maximum number of generations is fixed to 100.

To compare the performance between GA and FFR, the average of the best solutions is presented in table 4. For each problem instance, the proposed GA shows much better performance than the FFR heuristic. The improvement of the GA solutions is 16% on average that excludes the cases where the heuristic method does not find any feasible solution. In addition, the FFR heuristic has a major limitation that it sometimes fails to find feasible solutions. However, GA always finds feasible solutions for the test-bed problems. The average computation time required to create one solution with the single-pass heuristic for P24, P65 and P148 is all less than 0.01 s, whereas one run of GA requires 4, 15 and 60 s, respectively. However, this amount of computation time required by GA is still tolerable.

5. Summary and discussion

Balancing problems in two-sided assembly lines are addressed. Although the problems are often found in lines producing large-sized high-volume products, most previous researches deal with only one-sided lines. We develop a new method that can solve two-ALB problems. A genetic algorithm approach is adopted as the under-

Table 4. Comparison between GA and FFR.

Problem	Given # of stations	Cycle time	FFR Mean	FFR SD	FFR a	Proposed GA Mean	Proposed GA SD	Proposed GA b	Improved rate
P148	32	171	—	—	0.00	31.1	0.10	65.00	—
		200	—	—	0.00	26.9	0.10	83.40	—
		225	—	—	0.00	24.0	0.00	83.60	—
		250	24.0	0.00	0.00(2)*	21.0	0.00	83.60	12.50
		275	22.6	0.53	0.02	19.3	0.15	80.40	14.60
		300	22.0	0.00	0.11	18.0	0.00	76.80	18.18
		325	20.0	0.00	0.36	16.1	0.10	81.20	19.50
		350	19.5	0.53	1.62	15.0	0.00	79.80	23.08
		375	18.2	0.63	6.23	14.0	0.00	75.40	23.08
		400	18.2	0.63	28.93	13.2	0.13	80.20	27.47
P65	20	275	20.0	0.00	0.01(4)*	20.0	0.00	80.20	0.00
		300	19.7	0.48	0.15	18.1	0.14	86.50	8.12
		325	18.2	0.63	3.59	17.0	0.00	90.70	6.59
		350	16.2	0.63	16.98	15.5	0.16	86.20	4.32
		375	16.0	0.00	35.83	15.0	0.00	88.80	6.25
		400	16.0	0.00	56.30	13.8	0.13	90.20	13.75
		425	16.0	0.00	69.76	13.0	0.00	88.00	18.75
		450	15.1	0.32	77.83	12.1	0.10	91.20	19.87
		475	14.1	0.32	83.92	11.4	0.16	87.00	19.15
		500	14.1	0.32	83.80	11.0	0.00	89.11	21.99
P24	8	20	8.0	0.00	2.49	8.0	0.00	95.20	0.00
		25	7.5	0.53	22.55	6.0	0.00	94.20	20.00
		30	7.1	0.32	67.20	5.2	0.13	91.60	26.76
		35	6.0	0.00	91.66	5.0	0.00	81.60	16.67
		40	6.0	0.00	99.21	4.0	0.00	97.80	33.33

(a) The ratio of the number of feasible solutions over the 50 000 solutions obtained by FFR.
(b) The ratio of the number of feasible solutions contained in the last generation over the population size.
* Indicates that the number of feasible solutions is less than 10. The integer in parentheses is the number of feasible solutions. The average and standard deviation are taken over the feasible solutions.

lying framework of the new method. The major emphasis is placed on designing essential components of GA which include encoding and decoding scheme, and procedures of forming initial population, genetic operation and adaptation. We recognized that the self-adaptation capability of GA is very powerful in dealing with the problems.

We discuss the potential extensibility of the concept and technique proposed in this paper. First, because two-ALB is a generalized form of traditional one-ALB, the approach can be directly applied to this simpler problem. In this case, it is not necessary to consider preferred operation directions and mated-stations. The decoding procedure is unnecessary too. This procedure can be simply replaced by summing up task processing time for each workstation because it is not necessary to consider the sequence-dependent idle time.

Second, the proposed method can be used to minimize cycle time under a given number of workstations. We need some minor modifications to the proposed approach: (i) the encoding and decoding methods, and the procedure of generating initial population can be used without any change; (ii) because the goal is minimizing cycle time, F_j can be used as a measure for fitness evaluation; (iii) step 4 of the crossover operation can be simply removed; (iv) the calculated theoretical cycle time can be used for the cycle time (CT) required by the adaptation procedure because CT is not pre-specified in the problem. In addition, the proposed method can be used for other types of problems having different objectives. For example, consider workload smoothing which is often mentioned in previous researches dealing with ALB problems. This problem can be solved by replacing the evaluation function with a smoothness measure.

Third, the algorithm can be easily extended to accommodate real world complexities not explicitly addressed by conventional methods. For example, maximizing work relatedness can sometimes have practical significance. Related tasks are such tasks that are desirable to be assigned to the same workstation. If those are assigned to the same workstation, it can improve work efficiency, clarify the responsibility for the completion of the related tasks and give workers more job satisfaction. To improve work relatedness, a user can modify the original problem by adding supplementary positional constraints which force the movement and locking of some of the related tasks to one particular workstation. Then, the genetic algorithm can be applied once again. This subsequent application of GA respects the locked assignment. Another real world complexity is captured with restrictions on work zones. The work zone constraints are a general form of positional constraints. This specifies that a task must be assigned to one of several candidate workstations rather than just one candidate. The constraints can be met by using the intersection of the set of candidate workstations and the set of workstations between E_i and L_i in the initial population and adaptation procedure.

Fourth, a major advantage of GA is that it maintains a large variety of solutions. This pool of solutions provides line managers with a number of alternatives. This is especially important when criteria or requirements for good line balancing cannot be explicitly modelled. For example, variations of worker skill, space limitation due to large components, use of utility workers, etc. are widely present in practice, but these are usually difficult to be perfectly considered. However, with GA, line managers can just choose one promising solution from the pool while taking into account any of such considerations.

Fifth, the algorithm can be used as an engine for line balancing computer software. As Bartholdi (1993) points out, one major requirement for such software is an effective support of user interaction. Because task assignment in our algorithm is explicitly and easily represented, users' intention can be immediately conveyed to the software and vice versa.

References

Anderson, E. J., and Ferris, M. C., 1994, Genetic algorithms for combinatorial optimization: the assembly line balancing problem. *ORSA J. Computing* **6,** 161–173.

Arcus, A. L., 1963, An analysis of a computer method of sequencing assembly line operations. PhD dissertation, University of California, Berkeley.

Bartholdi, J. J., 1993, Balancing two-sided assembly lines: a case study. *International Journal of Production Research*, **31**, 2447–2461.

Ghosh, S., and Gagnon, R. J., 1989, A comprehensive literature review and analysis of the design, balancing and scheduling of assembly systems. *International Journal of Production Research*, **27**, 637–670.

Goldberg, D. E., 1989, *Genetic Algorithms in Search, Optimization & Machine Learning* (Reading, MA: Addison–Wesley).

Goldberg, D. E., Korb, B., and Deb, K., 1989, Messy genetic algorithms: motivation, analysis, and first results. *Complex Systems*, **3**, 493–530.

Gunther, R. E., Johnson, G. D., and Peterson, R. S., 1983, Currently practiced formulations for the assembly line balance problem. *Journal of Operations Management*, **3**, 209–221.

Gutjahr, A. L., and Nemhauser, G. L., 1964, An algorithm for the line balancing problem. *Management Science*, **11**, 308–315.

Helgeson, W. B., and Birnie, D. P., 1961, Assembly line balancing using the ranked positional weight technique. *Journal of Industrial Engineering*, **12**, 394–398.

Johnson, R. V., 1983, A branch and bound algorithm for assembly line balancing problems with formulation irregularities. *Management Science*, **29**, 1309–1324.

Kim, Y. J., Kim, Y. K., and Cho, Y., 1998a, A heuristic-based genetic algorithms for workload smoothing in assembly lines. *Computers & Operational Research*, **25**, 99–111.

Kim, Y. K., Kim, Y., and Lee, T. O., 1998b, Two-sided assembly line balancing models. Working paper, Department of Industrial Engineering, Chonnam National University, Korea.

von Laszewski, G., 1991, Intelligent structural operators for *k*–way group partitioning problem. In R. Belew and L. Booker (eds) *Proceedings of 4th International Conference on Genetic Algorithms*, San Diego, July 1991 (Morgan Kaufmann, San Mateo), pp. 45–52.

Lee, T. O., Kim, Y., and Kim, Y. K., 1997, An assignment procedure for balancing two-sided assembly lines. Working paper, Department of Industrial Engineering, Chonnam National University, Korea.

Leu, Y. Y., Matheson, L. A., and Rees, L. P., 1994, Assembly line balancing using genetic algorithms with heuristic-generated initial populations and multiple evaluation criteria. *Decision Science*, **25**, 581–606.

Suresh, G., Vinod, V. V., and Sahu, S., 1996, A genetic algorithm for assembly line balancing. *Production Planning & Control*, **7**, 38–46.

Talbot, F. B., Patterson, J. H., and Gehrlein, W. V., 1986, A comparative evaluation of heuristic line balancing techniques. *Management Science*, **32**, 430–454.