Journal of Advanced Manufacturing Systems
Vol. 13, No. 2 (2014) 113–131
© World Scientific Publishing Company
DOI: 10.1142/S0219686714500085

Study and Analysis of GA-Based Heuristic Applied to Assembly Line Balancing Problem

P. Sivasankaran
Department of Mechanical Engineering, ARS College of Engineering
Maraimalai Nagar, Chennai 603209, Tamil Nadu, India
sivasankaran_p@yahoo.in

P. Shahabudeen
Department of Industrial Engineering, College of Engineering
Anna University, Chennai 600 025, Tamil Nadu, India
psdeen@gmail.com

Balancing assembly line in a mass production system plays a vital role to improve the productivity of a manufacturing system. In this paper, a single model assembly line balancing problem (SMALBP) is considered. The objective of this problem is to group the tasks in the assembly network into a minimum number of workstations for a given cycle time such that the balancing efficiency is maximized. This problem comes under combinatorial category. So, it is essential to develop efficient heuristic to find the near optimal solution of the problem in less time. In this paper, an attempt has been made to design four different genetic algorithm (GA)-based heuristics, and analyze them to select the best amongst them. The analysis has been carried out using a complete factorial experiment with three factors, viz. problem size, cycle time, and algorithm, and the results are reported.

Keywords: Assembly line balancing; cycle time; genetic algorithm; crossover operation; factorial experiment.

1. Introduction

Mass production gives the advantage of reduced setup times and improved production volume of a product. Industries follow line layout for the mass production system. In the auto industry, engine production involves the assembly of different components, viz. crankcase, cylinder, head, connecting rod, piston, valves, cam, silencer, different bearings, etc. as per a given precedence assembly sequence. This assembly process involves manual operations. This is an example of line balancing problem in which the objective is to group the tasks of assembling the product into a minimum number of workstations without violating the precedence relationships as well as the cycle time constraint such that the balancing efficiency of the assembly line is maximized.

The inputs of assembly line balancing problem are as listed below.

(1) Precedence network of tasks along with their timings.
(2) Production volume per shift of the product.

Cycle time is computed based on the given production volume per shift using the following formula,[1] which is the maximum time allocated to each workstation.

$$\text{Cycle Time (CT)} = \frac{\text{Effective time available per shift}}{\text{Production volume per shift}}.$$

The balancing efficiency of solution of the line balancing problem is given by the following formula.[1]

$$\begin{aligned}\text{Balancing efficiency} &= \left(\frac{\text{Sum of all task times}}{\text{Number of workstations} \times \text{Cycle time}}\right) \times 100 \\ &= \left(\frac{\sum_{i=1}^{n} t_i}{N \times CT}\right) = \times 100,\end{aligned}$$

where

n is the number of tasks,
t_i is the required time of the task i,
CT is the cycle time,
N is the number of workstations.

If the objective is to minimize the number of workstations for a given production rate, it is usually referred in literature as the SALBP-1 problem. In this paper, the single model assembly line balancing problem (SMALBP) type 1 is considered.

2. Literature Review

This section presents the review of literature of the single model straight type assembly line balancing problem.

Based on the methods used to solve the single model straight type assembly line balancing problem, the literature under this category is further classified as listed below.

(1) Mathematical models;
(2) Petri net;
(3) Heuristics;
(4) Genetic algorithms (GAs);
(5) Simulated annealing algorithms;
(6) Tabu search algorithms;
(7) Ant colony optimization (ACO) algorithms;
(8) Shortest path algorithm;
(9) Critical path method (CPM).

2.1. *Mathematical models*

Thangavelu and Shetty[2] developed 0-1 integer programming model to solve the simple assembly line balancing problem such that the number of workstations is minimized for a given cycle time. Deckro and Rangachari[3] developed a goal programming model for the simple assembly line balancing problem with the objective of minimizing the number of workstations, in which zoning, sequencing, idle time, cycle time, and costs are considered simultaneously. Pastor[4] developed an improved mathematical model for SALBP type 1 problem as well as SALBP type 2 problem. The type 1 problem minimizes the number of workstations whereas the type 2 problem minimizes the cycle time. Based on a computational experiment, they proved that their models give superior results.

2.2. *Petri-net*

Kilincci and Bayhan[5] considered the assembly line balancing problem, in which the objective is to minimize the number of workstations for a given cycle time. They developed a Petri net approach for this problem.

2.3. *Heuristics*

Panneerselvam and Sankar[6] considered the SMALBP in which the objective is to minimize the number of stations for a given cycle time. They have considered six heuristics of Dar-EL's[7] research and proposed six new heuristics. Through a carefully designed experiment, they concluded that three heuristics of Dar-EL's[7] contribution and all the six newly proposed heuristics by them are selected as the best set of heuristics to solve the problem. Future researchers can use the best solution of this set of heuristics as the seed of a simulated annealing algorithm. Ponnambalam *et al.*[8] compared six assembly line balancing heuristics, viz. rank positional weight, Kilbridge and Wester, Moodie and Young, Hoffman Precedence matrix, immediate update first fit and rank and assign heuristics, based on a number of excess stations, line efficiency, smoothness index and CPU time.

Genikomsakis and Tourassis[9] proposed a new novel measure called task proximity index for each pair of tasks of the assembly line balancing problem. This is the total number of followers for the first task in the pair. This can be used to assign tasks to form workstations. As per this rule, from the list which contains the set of qualified tasks for the current station, the task which has the maximum number of followers is to be assigned to the current workstation. Moon *et al.*[10] considered the assembly line balancing problem with the objective of minimizing the total annual workstation costs and annual salary of the assigned workers for a given cycle time. In this problem, the workers with variety of skills are assumed. They developed a mixed integer liner programming model with a GA for this integrated assembly line balancing problem with resource restrictions. They reported that the proposed algorithm gives efficient solution based on experimentation with numerical problems. Mutlu and Ozgormus[11] considered the assembly line balancing problem in which the objective is to assign the tasks to workstations such that the total number of workstations is minimized for a given cycle time with physical workload constraints. The physical workload is defined as the cost incurred by an individual, given their capacities, while achieving a particular level of performance on a task with specific demands.[12] They developed a linear programming model for this problem. Mahto and Kumar[13] compared the performance of Kilbridge–Wester heuristic approach and Helgeson–Brine approach for the assembly line balancing problem of type 1 with the objectives of optimizing crew size, system utilization, the probability of jobs being completed within a certain framework and system design costs.

2.4. *Genetic algorithms*

Rubinnovitz and Levitin[14] considered the SMALBP with deterministic processing time. They developed a GA and compared its results with that of the MUST algorithm suggested by Dar-EL and Rubinovitch.[15] Gen *et al.*[16] developed GA for the assembly line balancing problem with fuzzy processing times with the objectives of minimizing the number of workstations and the total idle time under the precedence constraints. *The authors did not compare the performance of their method with existing best method.* Ponnambalam *et al.*[17] developed a multi-objective GA for solving simple assembly line balancing problem for a given cycle time. The objectives include the number of workstations, line efficiency, and the smoothness index. They compared its performance with six existing heuristics. Sabuncuoglu *et al.*[18] developed a GA for the simple assembly line balancing problem with the objective of minimizing the number of workstations. The GA uses a special chromosome structure that is partitioned dynamically through the evolution process. Elitism is also implemented in the model by using some concepts of simulated annealing. Chong *et al.*[19] considered the assembly line balancing problem with realized cycle time which is the maximum of the loads of the stations and proposed a GA with heuristic generated initial population. Later, they compared its performance with that of the GA with randomly generated initial population. They found that their algorithm gives better results. Yu and Yin[20] developed an adaptive GA to determine the minimum number of workstations with workload balance between the workstations for a given cycle time.

Goncalves and Almeida[21] developed a hybrid GA for the type 1 assembly line balancing problem to maximize the balancing efficiency for a given cycle time. Tsujimura *et al.*[22] developed a GA to design the SMALBP with fuzzy processing times to minimize balancing delay and number of workstations. They have used a repair algorithm to rearrange the chromosomes to maintain the precedence constraints among the tasks.

2.5. *Simulated annealing algorithms*

Hong and Cho[23] considered the problem of generation of robotic assembly sequences with considerations of line balancing using simulated annealing algorithm, in which an energy function is derived in consideration of the satisfaction of assembly constraints and the minimization of assembly cost and the idle time. The energy function is iteratively minimized and occasionally perturbed by a simulated annealing until no further change in energy occurs to obtain a solution of line balancing. Narayanan and Panneerselvam[24] developed a new efficient set of heuristics (NESHU) to assign the tasks to minimum number workstations such that the balancing efficiency is maximized. The initial solution is generated using a heuristic for assembly line balancing (HAL) and composite weight factor, and then it is improved using global search heuristic, which is similar to simulated annealing algorithm.

2.6. *Tabu search algorithms*

Lapierre *et al.*[25] developed a new tabu search algorithm to form workstations such that the balancing efficiency is maximized for a given cycle time. They tested their algorithm with industrial data set and reported the results. Annarongsri and Limnararat[26] developed a hybrid tabu search method for the simple assembly line balancing problem with the objective of minimizing the number of workstations for a given cycle time. They combined the tabu search with the GA to find the solution of this problem. *They compared the proposed meta-heuristic with COMSOAL, which is a primitive method and reported that it performs well. Hence, it may be compared with the recent best algorithm.* Ozcan *et al.*[27] developed a tabu search algorithm for simple parallel assembly line balancing (PALB) problem with deterministic task times, which maximizes the line efficiency and minimizes the variation of workload among the workstations.

2.7. *ACO algorithms*

Chica *et al.*[28] considered the 1/3 variant of the time and space assembly line balancing problem with the objectives of minimizing the number of stations and the station area, given a fixed value for the cycle time. They developed two algorithms, viz. multi-ant-colony-system algorithm (MACS) and multi-objective random greedy search algorithm (MORGA) to solve this problem. The results of these algorithms have been compared with that of Non-dominated Sorting Genetic Algorithm II (NSGA-II) using 10 well-known problem instances and found that both the proposed algorithms show good performance. Ozbakir *et al.*[29] proposed ACO for PALB problem with bi-objectives of minimizing the idle time of workstations and maximizing the line efficiency. Through an experiment, they found that this algorithm performs better than existing approaches. McMullen and Tarasewich[30] developed a modified ACO technique for the assembly line balancing problem, in which the components of the objective function are crew size, system utilization, the probability of jobs being completed within a certain time frame and system design costs.

2.8. *Shortest path algorithm*

Kao *et al.*[31] developed a shortest route algorithm for the assembly line balancing problem with resource constraint such that the number of workstations for a given cycle time and the sum of the number of resource types required in different stations are minimized. Boysen and Fliedner[32] proposed a versatile assembly line balancing algorithm which is able to solve instances of SMALBP as well as several generalized assembly line balancing problems (GALBP) with relevant constraints such as parallel workstations and tasks, cost strategies, processing alternatives, zoning constraints and probabilistic processing times.

2.9. *CPM*

Fathi *et al.*[33] developed a new heuristic method based on CPM for simple assembly line balancing problem. Yeh and Kao[34] developed a bidirectional heuristic to group the tasks into a minimum number of workstations such that the balancing efficiency is maximized. In this approach, the bidirectional approach is coupled with CPM, which is used in project management.

From these literatures, it is clear that attempts have been made earlier on the development of meta-heuristics and some specific heuristics to solve the assembly line balancing problem of type 1, which is known as SALBP-1, in which the objective is to group the tasks into different workstations such that the balancing efficiency is maximized for a given cycle time. Among the algorithms, the GA is considered in this paper to solve SALBP-1. The authors feel that the performance of the GA is affected by the crossover method. Hence, in this paper, GA is tried with crossover variations. Results obtained with these variations are compared with a complete factorial experiment.

3. GA-Based Heuristic

In this section, four different GA-based heuristics are proposed in terms of different crossover methods and variation by way of assigning tasks to different workstations. The performances in terms of balancing efficiency of these GA-based heuristics are analyzed in the next section.

The GA mimics the mechanism of selection and evaluation. It generates successive population of alternate solutions until a solution is obtained that yields acceptable result. It is based on the fundamental processes that control the evolution of biological organisms, namely natural selection and reproduction.

The GA has three important phases, which are as listed below.

(1) Generation of initial population.

(2) Performing crossover and mutation on a subpopulation.
(3) Carrying out the above two phases for specified number of times and selecting the best solution at the end of the iterative process.

In this paper, four GA-based heuristics are proposed, which are as listed below.

- GA-based heuristic with cyclic crossover method (GAH1),
- GA-based heuristic with forward crossover method (GAH2),
- GA-based heuristic with reverse crossover method (GAH3),
- GA-based heuristic with cyclic crossover method and modified workstation formation (GAH4).

The objective is to compare these GA-based heuristics and select the best among them. Construction of chromosomes: Consider the precedence network of assembling a product, which has eight tasks, as shown in Fig. 1. Each chromosome in the original population is generated by randomly assigning the tasks to different gene positions in the chromosome. Two sample chromosomes for the tasks in Fig. 1 are shown in Tables 1 and 2.

3.1. *Different crossover methods*

The performance of GA-based heuristic may be affected by the crossover method. Hence, in this paper, the following three different crossover methods are

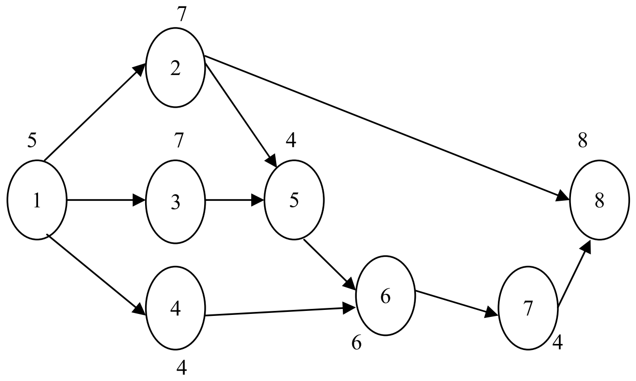

Fig. 1. Precedence network along with task times.

Table 1. Chromosome 1.

Position	1	2	3	4	5	6	7	8
Chromosome 1	1	5	6	3	7	4	8	2

Table 2. Chromosome 2.

Position	1	2	3	4	5	6	7	8
Chromosome 2	5	1	6	2	4	8	3	7

considered:

(1) Cyclic crossover method;
(2) Forward crossover method;
(3) Reverse crossover method.

3.1.1. *Cyclic crossover method*

The cyclic crossover method to create two offspring from the chromosome 1 and the chromosome 2 as shown in Tables 1 and 2, respectively, is explained below.

Step 1: Generate two crossover points randomly. Let them be 3 and 6.
Step 2: Identify the values of the genes in the chromosome 1 between the two crossover points inclusive of the crossover points. These values are 6, 3, 7 and 4.
Step 3: Form the offspring 2 whose gene positions are filled with the genes of the chromosome 2 as shown in Table 3.
Step 4: In the offspring 2, wherever the gene value is equal to each of the gene values identified between the two crossover points of the chromosome 1, set it to 0 as shown in Table 4.
Step 5: Read the nonzero values of the genes from left to right of the offspring 2 and write them in cyclic order as shown below starting from the next position of the second crossover point, except the points between the two crossover points as shown in Table 5.
Step 6: Write the values of the genes between the two crossover points of the chromosome 1 in the empty positions of the offspring 2 starting from the first vacant position from left as shown in Table 6 to get the final offspring 2.

Similarly, the offspring 1 to replace the chromosome 1 is created using the cyclic crossover method, which is as shown in Table 7.

Table 3. Offspring 2 with initial genes.

Position	1	2	**3**	4	5	**6**	7	8
Offspring 2	5	1	6	2	4	8	3	7

Table 4. Modified partial offspring 2.

Position	1	2	3	4	5	6	7	8
Offspring 2	5	1	0	2	0	8	0	0

Table 5. Rearranged partial offspring 2.

Position	1	2	3	4	5	6	7	8
Offspring 2	2	8					5	1

Table 6. Final offspring 2 using cyclic crossover method.

Position	1	2	3	4	5	6	7	8
Offspring 2	2	8	6	3	7	4	5	1

Table 7. Offspring 1 using cyclic crossover method.

Position	1	2	3	4	5	6	7	8
Offspring 1	3	7	6	2	4	8	1	5

3.1.2. *Forward crossover method*

The Steps 1–4 of this method are same as that of the cyclic crossover method.

Step 5: Read the nonzero values of the genes from left to right of the offspring 2 in Table 4 and write them in forward direction starting from the first available gene position from left to right, except the points between the two crossover points as shown in Table 8.

Step 6: Write the values of the genes between the two crossover points of the chromosome 1 in the empty positions of the offspring 2 starting from the first vacant position from left as shown in Table 9 to get the final offspring 2.

Similarly, the offspring 1 to replace the chromosome 1 is created using the forward crossover method, which is as shown in Table 10.

3.1.3. *Reverse crossover method*

The Steps 1–4 of this method are same as that of the cyclic crossover method.

Step 5: Read the nonzero values of the genes from left to right of the offspring 2 in Table 4 and write them in reverse direction starting from the last available

Table 8. Rearranged partial offspring 2.

Position	1	2	3	4	5	6	7	8
Offspring 2	5	1					2	8

Table 9. Final offspring 2 using forward crossover method.

Position	1	2	3	4	5	6	7	8
Offspring 2	5	1	6	3	7	4	2	8

Table 10. Offspring 1 using forward crossover method.

Position	1	2	3	4	5	6	7	8
Offspring 1	1	5	6	2	4	8	3	7

Table 11. Rearranged partial offspring 2.

Position	1	2	3	4	5	6	7	8
Offspring 2	8	2					1	5

Table 12. Final offspring 2 using reverse crossover method.

Position	1	2	3	4	5	6	7	8
Offspring 2	8	2	6	3	7	4	1	5

gene position from right to left, except the points between the two crossover points as shown in Table 11.

Step 6: Write the values of the genes between the two crossover points of the chromosome 1 in the empty positions of the offspring 2 starting from the first vacant position from left as shown in Table 12 to get the final offspring 2.

Similarly, the offspring 1 to replace the chromosome 1 is created using the reverse crossover method, which is as shown in Table 13.

3.2. *Construction of ordered vector*

If the workstations are formed by serially assigning the genes from the left of the chromosomes into workstations for a given cycle time, it may not be possible to assign the tasks, because the precedence constraints as shown in the Fig. 1 may not be satisfied. Hence, the genes of each chromosome are to be ordered (rearranged) such that the serial assignment of the genes from the ordered vector to workstations does not violate the precedence constraints as shown in the Fig. 1. Its steps are presented below.

Step 1: Input the chromosome.

(1) Let the chromosome I be as shown in Table 14 along with the values for the STATUS row as zero. If the value of the STATUS for a gene (task) is zero, then it signifies that it is not assigned to any workstation; otherwise, it signifies that it is assigned to some workstation.

(2) Form the immediate predecessor(s) matrix of the tasks shown in the Fig. 1 as shown in Table 15. The maximum of the number of immediate predecessors in the Fig. 1 is 2.

Table 13. Offspring 1 using reverse crossover method.

Position	1	2	3	4	5	6	7	8
Offspring 1	7	3	6	2	4	8	5	1

Table 14. Chromosome I.

Gene position J	1	2	3	4	5	6	7	8
STATUS	0	0	0	0	0	0	0	0
Chromosome 1	1	5	6	3	7	4	8	2

Table 15. Immediate predecessors matrix [IP_{pq}].

Task (p)	Immediate Predecessor(s) (q)	
1	Nil	
2	1	
3	1	
4	1	
5	2	3
6	4	5
7	6	
8	2	7

(3) Number of tasks (genes of the chromosome), N.
(4) Initialize the gene position of the ordered vector, $K = 1$.

Step 2: Set the gene position of the chromosome, $J = 1$.
Step 3: If the STATUS of the gene J of the chromosome I is equal to 1, then go to Step 9; else, go to Step 4.
Step 4: If all the immediate predecessors values in the Table 15 with respect to the task at the gene position J of the chromosome I are zero, then go to Step 5; otherwise, go to Step 9.
Step 5: Assign the task at the gene position J of the chromosome I to the gene position K of the ordered vector.

$$\mathrm{OV}_K = C_{IJ}$$

Step 6: Set the status of the gene position J of the chromosome I to 1.

$$\mathrm{STATUS}_J = 1$$

Step 7: Change the value of C_{IJ} in immediate predecessor matrix to zero, wherever it appears in it.
Step 8: Increment the gene position (K) of the ordered vector by 1 and go to Step 2.
Step 9: Increment the gene position (J) of the chromosome by 1.
Step 10: If $J \leq N$, then go to Step 3; otherwise go to Step 11.
Step 11: Stop.

The application of the above steps to the chromosome 1 shown in Table 1 gives an ordered vector as shown in Table 16.

Table 16. Ordered vector of chromosome 1 shown in Table 1.

Gene position J	1	2	3	4	5	6	7	8
Chromosome 1	1	3	4	2	5	6	7	8

Table 17. Solution for the ordered vector 1–3–4–2–5–6–7–8.

Workstation	Assigned tasks	Assigned time	Idle time
I	1, 3	12	0
II	4, 2	11	1
III	5, 6	10	2
IV	7, 8	12	0
Total		45	3
		Balancing efficiency = 93.75%	

3.3. *Evaluation of fitness function of chromosome*

The fitness function of the chromosome 1, namely balancing efficiency is obtained by assigning the tasks serially from left to right from its ordered vector 1–3–4–2–5–6–7–8 into workstations for a given cycle time of 12 units as shown in Table 17.

3.4. *GA-based heuristic with cyclic crossover method (GAH1)*

The steps of GA-based heuristic with cyclic crossover method to group the tasks into a minimum number of workstations such that the balancing efficiency is maximized are presented below.

Step 1: Input the following.

Number of tasks (n).
Processing times of tasks, T_J, $J = 1, 2, \ldots, n$.
Number of immediate predecessor(s) of tasks, NIP_p, $p = 1, 2, \ldots, n$.
Immediate predecessor(s) matrix of tasks, IP_{pq}, $p = 1, 2, \ldots, n$, $q = 1, 2, \ldots, \text{NIP}_p$
Cycle time, CT.
Mutation probability, $\alpha(0.3)$.
Best balancing efficiency, BBE $= 0$

Step 2: Set the GA parameters.

- Size of population, N.
- Size of subpopulation, M (30% of N).
- Number of iterations to be carried out, Q.

Creation and evaluation of population

Step 3: Construct N chromosomes of the population by randomly assigning the tasks to different gene positions in each of the chromosomes, $\text{C}_{K,J}$, $K = 1, 2, \ldots, N$ and $J = 1, 2, \ldots, n$.

Step 4: For each chromosome, $K = 1, 2, \ldots, N$, perform the following.

4.1: Find ordered vector of each chromosome.
4.2: Design workstation for the given cycle time using the processing times of the tasks and Immediate predecessor(s) matrix.
4.3: Find the balancing efficiency, BE_K.

Step 5: Set the iteration number q to 1.

Sorting the population

Step 6: Sort the chromosomes in the descending order of their balancing efficiencies.
Let the sorted chromosomes be, $\mathrm{SC}_{K,J}$, $K = 1, 2, \ldots, N$, $J = 1, 2, \ldots, n$ and the array of their sorted balancing efficiency be SBE_K, $K = 1, 2, \ldots, N$.

Step 7: Copy the sorted chromosomes, $\mathrm{SC}_{K,J}$, $K = 1, 2, \ldots, N$, $J = 1, 2, \ldots, n$ into $C_{K,J}$, $K = 1, 2, \ldots, N$, $J = 1, 2, \ldots, n$ along with their sorted balancing efficiencies SBE_K, $K = 1, 2, \ldots, N$ into balancing efficiency array, BE_K, $K = 1, 2, \ldots, N$.

Step 8: Perform the following steps to update the balancing efficiency if applicable.

8.1: If $\mathrm{BE}_1 \leq \mathrm{BBE}$, then go to Step 9; otherwise go to Step 8.2.
8.2: Update the following.

Best balancing efficiency, $\mathrm{BBE} = \mathrm{BE}_1$.
Best Chromosome, $\mathrm{BC}_J = \mathrm{SC}_{1,J}$, $J = 1, 2, \ldots, n$.

Step 9: If $q > Q$, then go to Step 20; otherwise, go to Step 10.

Workings on subpopulation

Step 10: Treat the topmost 30% of the population $(0.3N = P)$ of the new population after sorting as the subpopulation for the crossover operation. If the size of the subpopulation is not an even number, then add 1 to the size of the subpopulation.

Step 11: Set chromosome number, $P = 1$.

Step 12: Perform two point cyclic crossover between the chromosomes P and $P+1$ as listed below and obtain the offspring whose numbers are P and $P+1$.

Crossover between: $C_{P,J}$, $J = 1, 2, \ldots, n$ and $C_{P+1,J}$, $J = 1, 2, \ldots, n$.
Offspring obtained: $\mathrm{OS}_{P,J}$, $J = 1, 2, \ldots, n$ and $\mathrm{OS}_{P+1,J}$, $J = 1, 2, \ldots, n$.

Step 13: Perform mutation in each of the offspring for a mutation probability of α.

Step 14: For each of the offspring (offspring P and offspring $P+1$), perform the following.

14.1: Find the ordered vector of the offspring.
14.2: Design the workstation for the given cycle time using the processing times, T_J, $J = 1, 2, \ldots, n$ and immediate predecessor(s) matrix, $\mathrm{IP}_{pq}, p = 1, 2, \ldots, n$ and $q = 1, 2, \ldots, \mathrm{NIP}_p$ as per the ordered vector.
14.3: Find the balancing efficiency (BE_K and BE_{K+1}).

Step 15: Increment chromosome number by 2, $P = P + 2$.

Step 16: If $P \leq M$ then go to Step 12; otherwise, go to Step 17.

Step 17: Copy the new offspring $\mathrm{OS}_{R,J}$, $R = 1, 2, \ldots, M$ and $J = 1, 2, \ldots, n$ to the respective chromosome vectors, $C_{P,J}$, $P = 1, 2, \ldots, M$ and $J = 1, 2, \ldots, n$, respectively.

Step 18: Increment the iteration number by 1 $(q = q + 1)$.

Step 19: Go to Step 6.

Step 20: Perform the following.

Find the ordered vector of the best chromosome, BC_J, $J = 1, 2, \ldots, n$.
Design the workstation for the given cycle time using the processing times, T_J, $J = 1, 2, \ldots, n$ and immediate predecessor(s), IP_{pq}, $p = 1, 2, \ldots, n$ and $q = 1, 2, \ldots, \mathrm{NIP}_p$ and print them.
Print the best balancing efficiency, BBE.

Step 21: Stop.

3.5. *GA-based heuristic with forward crossover method* (*GAH*2)

The steps of the GA-based heuristic with *forward crossover* method are same as that of the GA-based heuristic with cyclic crossover method, except the Step 12. The required Step 12 of this heuristic is given below.

Step 12: Perform two point *forward crossover* between the chromosomes P and $P + 1$ as listed below and obtain the offspring whose numbers are P and $P + 1$.

Crossover between: $C_{P,J}$, $J = 1, 2, \ldots, n$ and $C_{P+1,J}$, $J = 1, 2, \ldots, n$.
Offspring obtained: $\mathrm{OS}_{P,J}$, $J = 1, 2, \ldots, n$ and $\mathrm{OS}_{P+1,J}$, $J = 1, 2, \ldots, n$.

3.6. *GA-based heuristic with reverse crossover method* (*GAH*3)

The steps of the GA-based heuristic with *reverse crossover method* are same as that of the GA-based heuristic with cyclic crossover method, except the Step 12. The required Step 12 of this heuristic is shown below.

Step 12: Perform two point *reverse crossover* between the chromosomes P and $P + 1$ as listed below and obtain the offspring whose numbers are P and $P + 1$.

Crossover between: $C_{P,J}$, $J = 1, 2, \ldots, n$ and $C_{P+1,J}$, $J = 1, 2, \ldots, n$.
Offspring obtained: $\mathrm{OS}_{P,J}$, $J = 1, 2, \ldots, n$ and $\mathrm{OS}_{P+1,J}$, $J = 1, 2, \ldots, n$.

3.7. *GA-based heuristic with cyclic crossover method and modified workstation formation* (*GAH*4)

The steps of the GA-based heuristic with *cyclic crossover method and modified workstation formation* are same as that of the GA based heuristic with cyclic crossover method, except the Steps 4.2 and 14.2. The required Steps 4.2 and 14.2 of this heuristic are shown below.

Step 4.2: Design the workstation for the given cycle time using the processing times, T_J, $J = 1, 2, \ldots, n$ and immediate predecessor(s) matrix, IP_{pq}, $p = 1, 2, \ldots, n$ and $q = 1, 2, \ldots, \mathrm{NIP}_p$ as per the ordered vector with the following addition.

While moving from one workstation to another workstation, if there is idle time in the current workstation, then look for alternate succeeding task(s) which can best fit into the current workstation that will result with either zero idle time or least idle time in that workstation.

Step 14.2: Design the workstation for the given cycle time using the processing times, T_J, $J = 1, 2, \ldots, n$ and immediate predecessor(s) matrix, IP_{pq}, $p = 1, 2, \ldots, n$ and $q = 1, 2, 3, \ldots, \mathrm{NIP}_p$ as per the ordered vector with the following addition.

While moving from one workstation to another workstation, if there is idle time in the current workstation, then look for alternate succeeding task(s) which can best fit subject to precedence constraints, into the current workstation that will result with either zero idle time or least idle time in that workstation.

4. Comparison of GA-Based Heuristics

In this section, the four different GA-based heuristics, viz. GAH1, GAH2, GAH3 and GAH4, are compared using a complete factorial experiment.

The factors considered in this experiment are problem size (Factor A), cycle time (Factor B) and algorithm, alternatively known as Heuristic (Factor C).

The number of levels of the problem size (Factor A) is 8, which varies from 15 tasks to 50 tasks with an equal increment of five tasks. For each problem size, a feasible cycle time is assumed and it acts as the first level of the Factor B and 125% of that cycle time is assumed as the level 2 of the Factor B. The four different GA-based heuristics (GAH1, GAH2, GAH3 and GAH4) are assumed as the levels of the Factor C. For each of the experimental combinations, two replications have been generated by keeping the number of tasks as a constant and varying the tasks times. This experiment resulted with 128 observations (balancing efficiencies).

The ANOVA model of this three factor experiment is

$$Y_{ijkl} = \mu + A_i + B_j + AB_{ij} + C_k + AC_{ik} + BC_{jk} + ABC_{ijk} + e_{ijkl},$$

where Y_{ijkl} is the balancing efficiency w.r.t. lth replication under ith problem size, jth cycle time and kth algorithm.

μ	is the overall mean of the balancing efficiency.
A_i	is the effect of the ith problem size on the balancing efficiency.
B_j	is the effect of the jth cycle time on the balancing efficiency.
AB_{ij}	is the interaction effect of the ith problem size and jth cycle time on the balancing efficiency.
C_k	is the effect of kth algorithm on the balancing efficiency.
AC_{ik}	is the interaction effect of the ith problem size and kth algorithm on the balancing efficiency.
BC_{jk}	is the interaction effect of the jth cycle time and kth algorithm on the balancing efficiency.

ABC_{ijk} is the interaction effect of the ith problem size, jth cycle time and kth algorithm on the balancing efficiency.

e_{ijkl} is the random error associated with the lth replication under ith problem size, jth cycle time and kth algorithm.

The result of the replication under each experimental combination is shown in Table 18. The results of the ANOVA of the complete factorial experiment are summarized in Table 19. From Table 19, it is clear that the Factor A, the Factor B and the interaction between the Factors A and B are significant and the remaining components of this model are insignificant at a significance level of 0.05.

Specifically, the calculated F value for the Factor C (0.1562) is less than the corresponding Table F value(2.770) at a significant level of 0.05. This means that

Table 18. Results of replications under different experimental combinations.

Problem Size	Cycle Time	Replication	GAH1	GAH2	GAH3	GAH4
15	50	1	87.50	87.50	87.50	77.78
		2	82.00	82.00	82.00	82.00
	65	1	89.74	89.74	89.74	89.74
		2	90.11	90.12	90.11	90.11
20	40	1	87.96	87.96	87.96	87.96
		2	80.88	80.88	80.88	80.88
	50	1	86.00	86.00	86.00	86.00
		2	78.57	78.57	78.57	78.57
25	60	1	74.25	74.25	74.25	74.25
		2	86.67	86.67	86.67	86.67
	75	1	79.20	74.25	79.20	79.20
		2	86.67	86.67	86.67	86.67
30	80	1	76.77	81.56	81.56	76.77
		2	85.07	85.07	85.07	85.06
	100	1	87.00	87.00	80.31	87.00
		2	87.50	87.50	87.50	87.50
35	85	1	80.12	80.12	80.12	80.12
		2	87.58	82.97	82.97	82.97
	110	1	82.55	82.55	82.55	82.55
		2	87.01	87.01	93.71	87.01
40	75	1	78.61	78.61	82.35	78.61
		2	79.47	82.78	82.78	79.47
	95	1	85.33	91.02	85.33	85.33
		2	87.17	87.13	87.13	87.13
45	95	1	79.31	82.14	79.31	79.31
		2	81.29	81.29	81.29	81.29
	120	1	86.71	82.77	82.77	86.71
		2	86.88	86.88	82.74	86.88
50	100	1	81.94	81.94	81.94	81.94
		2	79.84	82.42	82.42	79.84
	125	1	81.28	81.28	81.28	81.28
		2	81.76	81.76	81.76	81.76

Table 19. Results of ANOVA.

Source of Variation	Sum of Squares	Degrees of Freedom	Mean Sum of Squares	F_{cal}	F_{tab}	Remark
Problem Size (A)	317.5625	7	45.3661	2.8589	2.172	Significant
Cycle Time (B)	316.3125	1	316.3125	19.9264	4.010	Significant
AXB	262.7500	7	37.5357	2.3646	2.172	Significant
Algorithm (C)	7.4375	3	2.4792	0.1562	2.770	Insignificant
AC	59.0625	21	2.8125	0.1772	1.750	Insignificant
BC	18.9375	3	6.3125	0.3977	2.770	Insignificant
ABC	70.0000	21	3.3330	0.2100	2.770	Insignificant
Error	1015.9380	64	15.8740			
Total	2068.0000	127				

there is no significant difference between the algorithms (GAH1, GAH2, GAH3 and GAH4). So, any of the four GA-based heuristics can be selected for implementation. However, one can select the heuristic, which has the maximum mean balancing efficiency. As per this guideline, the GA-based heuristic 2 (GAH2) has the maximum mean balancing efficiency of 83.70031%. Hence, the GA-based heuristic with forward crossover method is selected as the best heuristic for solving the assembly line balancing problem considered in this paper.

5. Conclusion

In this paper, the SMALBP in which the balancing efficiency is to be maximized for a given cycle time, has been considered. Since, this problem comes under combinatorial category, development of an efficient heuristic is warranted. In this paper, four different GA-based heuristics, viz. GAH1, GAH2, GAH3 and GAH4 have been proposed. These are based on different crossover methods, viz. cyclic crossover method, forward crossover method and reverse crossover method and cyclic crossover method with modified workstation formation. The last three crossover methods are newly introduced in this paper. The procedure presented in Sec. 3.2 to obtain the ordered vector for each chromosome in the initial population as well as for each offspring is another contribution in this paper, which is very simple to implement.

These heuristics are compared using a complete factorial experiment with three factors, viz. "problem size", "cycle time" and "algorithm". Using the experimentation, it is found that the factors "problem size", "cycle time" and the interaction between the "problem size" and "cycle time" have effect on the balancing efficiency. The other components of the ANOVA model do not have effect on the balancing efficiency.

Since, the factor "algorithm" does not have effect on the balancing efficiency, there is no significant difference between the GA-based heuristics in terms of the balancing efficiency. This in turn implies that the crossover methods have no effect on the balancing efficiency of the assembly line balancing problem considered in this paper. Based on this inference, though any one of the GA-based heuristics may be selected for implementation, one may select the GA-based heuristic which has the maximum mean balancing efficiency. The GA-based heuristic with the forward crossover method has the maximum balancing efficiency and hence it is recommended to use this heuristic to design the minimum number of workstations for a given cycle time in the SMALBP.

References

1. R. Panneerselvam, *Production and Operations Management*, 3rd edn. (PHI Learning Private Limited, New Delhi, 2012).
2. S. R. Thangavelu and C. M. Shetty, Assembly line balancing by zero-one integer programming, *IIE Transactions* **3** (1971) 61–68.
3. R. F. Deckro and S. Rangachari, A goal approach to assembly line balancing, *Computers and Operations Research* **17** (1990) 509–521.
4. R. Pastor, LB-ALBP: The lexicographic bottleneck assembly line balancing problem, *International Journal of Production Research* **9** (2011) 2425–2442.
5. O. Kilincci and G. M. Bayhan, A petrinet approach for simple assembly line balancing problems, *International Journal of Advanced Manufacturing Technology* **30** (2006) 1165–1173.
6. R. Panneerselvam and C. O. Sankar, New heuristics for assembly line balancing problem, *International Journal of Management and Systems* **9** (1993) 25–36.
7. E. M. Dar-El, Solving large single-model assembly line balancing problems — a comparative study, *AIIE Transactions* **7** (1975) 302–310.
8. S. G. Ponnambalam, P. Aravindan and G. M. Naidu, A comparative evaluation of assembly line balancing heuristics, *Advanced Manufacturing Technology* **15** (1999) 577–586.
9. K. N. Genikomsakis and V. D. Tourassis, Task proximity index: A novel measure for assessing the work-efficiency of assembly line balancing configurations, *International Journal of Production Research* **50** (2012) 1624–1638.
10. I. Moon, R. Logendran and J. Lee, Integrated assembly line balancing with resource restrictions, *International Journal of Production Research* **47** (2009) 5525–5541.
11. O. Mutlu and E. Ozgormus, A fuzzy assembly line balancing problem with physical workload constraints, *International Journal of Production Research* **50** (2012) 5281–5291.
12. A. T. DiDomenico, An investigation on subjective assessment of workload postural stability under conditions of joint mental and physical demands, thesis (PhD), Virginia Polytechnic Institute and State University (2003).
13. D. Mahto and A. Kumar, An empirical investigation of assembly line balancing techniques and optimized implementation approach for efficiency improvements, *Global Journal of Researches in Engineering Mechanical and Mechanics Engineering* **12** (2012) 1–14.
14. J. Rubinnovitz and G. Levitin, Genetic algorithm for assembly line balancing, *International Journal of production Economics* **41** (1995) 343–354.
15. E. M. Dar-El and Y. Rubinovitch, MUST — a multiple solutions technique for balancing single model assembly lines, *Management Science* **25** (1979) 1105–1113.
16. M. Gen, Y. Tsujimura and Y. Li, Fuzzy assembly line balancing using genetic algorithms, *Computers and Industrial Engineering* **31** (1996) 631–634.
17. S. G. Ponnambalam, P. Aravindan and G. M. Naidu, A multi-objective genetic algorithm for solving assembly line balancing problem, *International Journal of Advanced Manufacturing Technology* **16** (2000) 341–352.

18. I. Sabuncuoglu, R. Erel and M. Tanyer, Assembly line balancing using genetic algorithm, *Journal of Intelligent Manufacturing* **11** (2000) 295–310.
19. K. E. Chong, M. K. Omar and N. A. Baker, Solving assembly line balancing problem using genetic algorithm with heuristic treated initial population, in *Proc. World Congress on Engineering* (2008). ISBN:978-988-17012-3-7.
20. J. Yu and Y. Yin, Assembly line balancing based on an adaptive genetic algorithm, *International Journal of Advanced Manufacturing Technology* **48** (2010) 347–354.
21. J. F. Goncalves and J. R. D. Almeida, A hybrid genetic algorithm for assembly line balancing, *Journal of Heuristics* **8** (2002) 629–642.
22. Y. Tsujimura, M. Gen and E. Kubota, Solving fuzzy assembly-line balancing problem with genetic algorithm, *Computers and Industrial Engineering* **29** (1995) 543–547.
23. D. S. Hong and H. S. Cho, Generation of robotic assembly sequences with consideration of line balancing using simulated annealing, *Robotica* **15** (1997) 663–674.
24. S. Narayanan and R. Panneerselvam, New efficient set of heuristics for assembly line balancing, *International Journal of Management and Systems* **16** (2000) 287–300.
25. S. D. Lapierre, A. Ruiz and P. Soriano, Balancing assembly lines with tabu search, *European Journal of Operational Research* **168** (2006) 826–837.
26. S. Annarongsri and S. Limnararat, A hybrid tabu search method for assembly line balancing, in *Proc. 7th Int. Conf. Simulation, Modelling and Optimization*, China, (2007).
27. U. Ozcan, H. Cercioglu, H. Gokcen and B. Toklu, A tabu search algorithm for the parallel assembly line balancing problem, *G.U. Journal of Science* **22** (2009) 313–323.
28. M. Chica, O. Cordon, S. Damas and J. Bautista, Multi-objective constructive heuristics for the 1/3 variant of the time and space assembly line balancing problem: ACO and random greedy search, *Information Sciences* **180** (2010) 3465–3487.
29. L. Ozbakir, A. Baykasoglu, B. Gorkemli and L. Gorkemli, Multiple-colony ant algorithm for parallel line balancing problem, *Applied Soft Computing* **11** (2011) 3186–3198.
30. P. R. McMullen and P. Tarasewich, Multi-objective assembly line balancing via a modified ant colony technique, *International Journal of Production Research* **44** (2006) 27–42.
31. H.-H. Kao *et al.*, An optimal algorithm for type-I assembly line balancing problem with resource constraint, *African Journal of Business Management* **4** (2010) 2051–2058.
32. N. Boysen and M. Fliedner, A versatile algorithm for assembly line balancing, *European Journal of Operational Research* **184** (2008) 39–56.
33. M. Fathi, A. Jahan, M. K. A. Ariffin and N. Ismail, A new heuristic method based on CPM in SALBP, *International Journal of Industrial Engineering* **7** (2011) 1–11.
34. D. H. Yeh and H.-H. Kao, A new bidirectional heuristic for the assembly line balancing problem, *Computers and Industrial Engineering* **57** (2009) 1155–1160.