Biologically Inspired Genetic Algorithm to Minimize Idle Time of the Assembly Line Balancing

Noraini Mohd Razali
School of Mechanical & Manufacturing Engineering
Dublin City University
Dublin 9, Ireland
norainimbr@ump.edu.my

John Geraghty
Enterprise Research Process Centre
Dublin City University
Dublin 9, Ireland
john.geraghty@dcu.ie

Abstract—**Assembly line balancing (ALB) is a well-known combinatorial optimization problem in production and operations management area. Due to the NP-hard nature of the ALB problem, many attempts have been made to solve the problem efficiently. In this study, biologically inspired evolutionary computing tool which is genetic algorithm (GA) is adopted to solve the ALB problem with the objective of minimizing the idle time in the workstation. The key issue in solving ALB is how to generate a feasible task sequence which does not violate the precedence constraints. This task sequencing is a vital work to be solved prior assigning tasks to workstation. In order to generate only feasible solution, a repairing strategy based topological sort is integrated in the GA procedure. The ALB test problems benchmarked from the literature are used in the study and the computational results show that the proposed approach is capable to obtain feasible solution with minimum idle time for a simple model assembly line.**

Keywords-genetic algorithm; topological sort; assembly line balancing; task sequencing; idle time

I. Introduction

Assembly lines are the most commonly used method in a mass production environment, because they allow the assembly of complex products by workers with limited training, by dedicated machines or by robots. An assembly line comprises a series of successive workstations connected together by a material handling system in which the components are consecutively assembled into a final product. A workstation is a physical area where a worker with tools and machines, or an unattended machine like robot performs a particular set of tasks. The components are processed depending on a set of tasks (operations), and they are performed at each workstation during a fixed time called as cycle time. The assembly line balancing (ALB) problem is to assign a set of tasks to workstation according to given precedence relationships among tasks and specific restrictions which aim to optimize one or more objectives, such as minimizing the number of stations and minimizing the cycle time [1]. Assembly line balancing occurs whenever an assembly line is reconfigure, redesigned or adjusted to match the demand of new product or capacity. The main focus of ALB problem is to obtain a task sequence which is feasible, minimizes workstation and ensures minimum idle times in the workstation so that the efficiency of the line is maximized.

Due to computational complexity of the problem, ALB is known to be an NP-hard problem [2]. The complexity of the ALB problem makes the optimum seeking methods impractical for problems of more than a few tasks or workstations. The number of the possible sequences is usually a factorial function of the number of tasks. This fact obviously leads to a combinatorial explosion of the number of alternatives to analyze for checking and selecting the best assembly sequence and, consequently, to unacceptable computation time. Many attempts have been made in the literature to solve the ALB problem using exact seeking methods, such as linear programming [3], integer programming [4], dynamic programming [5], and branch and bound approaches [6]. However, none of these methods has proven to be of practical use for large problems due to their computational inefficiency. Hence, numerous research efforts have been directed towards the development of heuristics and metaheuristics such as simulated annealing [7], tabu search [8] and genetic algorithms [9].

Genetic algorithms (GA) are a well-known optimization technique which appears to have been inspired by biology. The GA is designed to mimic nature's problem solving strategies. The techniques have powerful performance for such combinatorial optimization problems, especially for sequencing problems such as TSP, flow-shop scheduling and so on. However, when apply GA to synthesis practical sequencing problems, the infeasible chromosomes are often produced during crossover and mutation operations. Therefore, keeping feasibility of chromosomes might be considered as important issue when applying genetic algorithm. In this paper, a repairing method based on topological sort is integrated with GA in order to handle precedence constraints and generate only feasible solution during the evolutionary process. The paper consists of 5 sections. Following the introduction is the review of the ALB problem in section 2. Section 3 is dedicated to the design of the genetic algorithms to solve ALB problem. The results of the experiments are reported in section 4. Finally, conclusions are drawn.

II. REVIEW OF ASSEMBLY LINE BALANCING

Assembly line balancing (ALB) problem was first introduced by Bryton [10] and the first scientific study was published by Salveson [4]. The problem is to assign a set of tasks to workstation with some measure of performance to be optimized under the following restrictions: (i) each task is assigned to one and only one workstation, (ii) the precedence relationship among the tasks cannot be violated, and (iii) the sum of the task times of any workstation should not exceed the cycle time [11]. Since the task times allotted to workstations may be unequal, parts are produced at different speeds on the line. Accordingly, stations may either be starved or a queue may build up in front of a station. To regulate the flow of parts, assembly lines are often paced. In a paced line, each workstation is given a fixed amount of time called cycle time. The cycle time of an assembly line is predetermined by a desired production rate. Such production rate is set so that the desired amount of end product is produced within a certain time period. Material handling systems are designed so that after every certain cycle times, the system indexes, advancing the part to the next station. If a workstation finishes in less than cycle time given, it is idle for the remaining period. The difference between the time required by any station to complete its operations and the cycle time is called the idle time of the station. It is conventional to take the sum of all station idle times (called total idle time) as a measure of the efficiency of the design of a line.

The ALB is usually presented by the precedence graph. Consider a precedence graph in Figure 1 which specifies the order or sequence in which the task must be performed. The number in each circle refers task number, and the number above the circle refers the duration of the operation (task). The arrow represents directions of flow of operation.

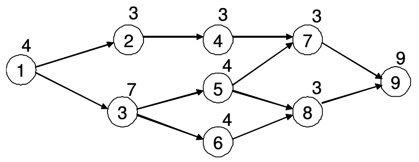

Figure 1. The precedence graph of simple assembly line.

The variable of interest for the ALB consists of number of tasks (*n*), processing time, precedence relationships, and the cycle time (*CT*). The goals of the ALB problems are to minimize the number of workstations (*m*), minimize the idle time (T_{id}), and maximize the line efficiency (*E*). Salveson [4] was the first to give a mathematical form to the problem and proposed a linear integer programming model to solve it. Since then a large number of articles have been published, detailing the advances in this field, and articles reviewing the models proposed are published periodically. Formulations of workstation, idle time, and line efficiency are given in (1) to (3) respectively, where *W* is the total processing time and T_i is the processing time of the *i*th workstation.

$$m = W / CT \tag{1}$$

$$T_{id} = \sum_{i=1}^{m} (CT - T_i) \tag{2}$$

$$E = \sum_{i=1}^{m} T_i / (mCT) \tag{3}$$

III. GENETIC ALGORITHM PROCEDURE FOR ALB PROBLEM

The genetic algorithm (GA) introduced by J. Holland in 1975 [12] with the inspiration of biology evolution is a stochastic techniques which become more and more popular for searching the optimal or near optimal solutions to large-scale optimization problems. The biological system's robustness, flexibility, efficiency encouraged scientists led by John Holland to search for artificial systems. The GA starts with a set of random solutions called population, and each individual in the population is called chromosome. The chromosomes evolve through successive iterations, called generations, during which the chromosomes are evaluated by means of measuring their fitness. To create the next generation, new chromosomes called offspring are formed by two genetic operators, i.e., crossover (merging two chromosomes) and mutation (modifying a chromosome). A new generation is formed by selection, which means that chromosomes with higher fitness values have higher probability of being selected to survive. After several generations, the algorithm converges to the best chromosome, which hopefully represents the optimal or near optimal solution to the problem.

In order to find optimal solution to the ALB problem via GA methods, five critical elements are required. First, an appropriate representation is required. This is accomplished by representing a task sequence in terms of chromosome. Second, a repair method is needed to restore infeasible chromosome generated in the initial population. Third, a fitness function is required to evaluate the quality of different potential solutions. Forth, a set of genetic operators (i.e., parent selection, crossover and mutation) which generate new chromosomes as a function of older chromosomes must be defined. Finally, algorithm parameters must be decided. Below are the steps in the GA procedure for ALB problem;

1. Chromosome representation
2. develop method to obtain feasible sequence
3. develop the fitness function
4. develop a reproduction method
5. set up GA parameters

A. *Chromosome Representation*

The first step in constructing a genetic algorithm for ALB problem is to define a genetic representation (encoding). Before a genetic algorithm can be put to work on any problem, a method is needed to represent potential solutions to that problem in a form that a computer can process. One common approach is to encode solutions as binary strings:

sequences of 1's and 0's. Another similar approach is to represent solutions as arrays of integers or decimal numbers, or to represent chromosomes as strings of letters. The *path representation* also called *permutation representation* is probably the most natural representation for sequencing problem. The length of a chromosome is the number of tasks to be processed in the assembly line. Since the chromosomes are generated randomly, the possibility of which generated chromosomes are infeasible due to breaking of one or more precedence constraint is greater. The procedure of repairing infeasible chromosome is necessary to convert the infeasible chromosomes into feasible ones. The chromosomes in the initial population are all randomly generated. The individual will then be repaired by topological sort procedure in order to obtain feasible sequence.

B. *Repair Method for Feasible Solution*

The key issue of the sequencing problem is to find an appropriate order of tasks or operations. This is a permutation problem in nature. Due to the existence of precedence constraints among tasks, an arbitrary permutation may yield an infeasible order. The approach to overcome generating invalid sequence is based on a topological sort, which allows the GA to generate only valid solutions in each generation. The topological sort is a node ordering in a directed graph such that if there is a path from a node v_i and a node v_j, then v_j appears after v_i in the ordering [13]. In a directed graph, the nodes represent tasks and the edges represent the precedence relations between tasks. More than single sequence of tasks can be derived from a directed graph using the topological sort technique. The procedure to sort nodes consists of selecting and storing any node that has no incoming edges. Then the nodes and all the edges leading out from the node are removed from the graph. Thus the path (v_i, v_j) in the directed graph shows that node v_i must be executed or scheduled before node v_j. If there is more than one node that has no incoming edges, a few ways can be performed in order to select the node such as by random selection, comparison of lower number of nodes and comparison of higher number of nodes. Then, the edges that start from the selected nodes are removed. This procedure is continuous until all nodes are selected. The procedure of the repair mechanism using topological sort can be summarized as below;

Step 1: form an initial available set of nodes having no predecessors, and create an empty string.

Step 2: terminate, if the available set is empty. Otherwise, go to step 3.

Step 3: select a node from the available set at random, and append it to the string.

Step 4: update the available set by removing the selected node and by adding every immediate successor of the node if all the immediate predecessors of the successor are already in the string. Go to step 2.

The infeasible chromosomes in the initial population as well as the offspring's chromosomes created from the reproduction process need to be repaired before going through the evaluation process. Consider Figure 1 and infeasible chromosome (1-6-5-3-2-4-7-9-8). TABLE I shows the step-by-step results after fixing this chromosome.

TABLE I. REPAIR MECHANISM WITH RANDOM SELECTION OF NODES (TASKS)

Available nodes(tasks)	**Infeasible chromosome**	**Updated sequence**
{1}	**1** 6 5 3 2 4 7 9 8	1
{2,3}	1 6 5 **3** 2 4 7 9 8	1 3
{2,5,6}	1 6 5 3 **2** 4 7 9 8	1 3 2
{4,5,6}	1 6 **5** 3 2 4 7 9 8	1 3 2 5
{4,6}	1 6 5 3 2 **4** 7 9 8	1 3 2 5 4
{6,7}	1 6 5 3 2 4 **7** 9 8	1 3 2 5 4 7
{6}	1 **6** 5 3 2 4 7 9 8	1 3 2 5 4 7 6
{8}	1 6 5 3 2 4 7 9 **8**	1 3 2 5 4 7 6 8
{9}	1 6 5 3 2 4 7 **9** 8	1 3 2 5 4 7 6 8 9

C. *Fitness Evaluation Function*

Fitness evaluation is to check the solution value of the objective function subject to the problem constraints. In this case, the objective function is simply to minimize the total idle time.

$$\text{Minimize } T_{id} = \sum_{i=1}^{m} (CT - T_i)$$

Subject to precedence constraint

D. *Genetic Operators*

The genetic operators mimic the process of heredity of genes to create new offspring at each generation. The operators are used to alter the genetic composition of individuals during representation. In essence, the operators perform a random search, and cannot guarantee to yield an improved offspring. There are three common genetic operators: selection, crossover and mutation.

Selection for Reproduction – In keeping with the ideas of natural selection, we assume that stronger individuals, that is, those with higher fitness values, are more likely to mate than the weaker ones. One way to simulate this is to select parents with a probability that is directly proportional to their fitness values. This method is called the roulette wheel method. The idea behind the roulette wheel selection technique is that each individual is given a chance to become a parent in proportion to its fitness. The chances of selecting a parent can be seen as spinning a roulette wheel with the size of the slot for each parent being proportional to its fitness. Obviously, those with the largest fitness (slot sizes) have more chance of being chosen. By repeating this each time an individual needs to be chosen, the better individuals will be chosen more often than the poorer ones, thus fulfilling the requirements of survival of the fittest. The basic advantage of roulette wheel selection is that it discards none of the individuals in the population and gives a chance to all of them to be selected [14]. Therefore, diversity in the population is preserved.

Crossover – The crossover operator is develops to change the order between the task in a chromosome solution. Through crossover, two parents chromosomes obtained from selection process are mated at a given probability, and produce two offspring (new chromosomes). Linear order crossover (LOX), a modified version of Order crossover (OX) is used in this study. The LOX operator treats the chromosome as a linear entity. For this operator, the swap occurs in the same fashion as it occurs in the OX operator, but when sliding the parent values around to fit in the remaining open slots of the child chromosome, they are allowed to slide to the left or right. This allows the chromosome to maintain its relative ordering and at the same time preserve the beginning and ending values. In the below example, after the values are swapped, there are two open spaces in the front of the chromosome and three open spaces at the end. The algorithm then goes through Parent 1 and finds the first two values that were not part of the swap, in this example they are 5 and 4. These values are shifted left to fill the first two chromosome locations. The final three locations are filled in a similar manner.

Parent 1 = 3 9|5 4 6 2|7 1 8
Parent 2 = 7 4|3 8 9 2|1 5 6
Offspring 1 = * * 3 8 9 2 * * *
Offspring 2 = * * 5 4 6 2 * * *
Offspring 1 = 5 4 3 8 9 2 6 7 1
Offspring 2 = 7 3 5 4 6 2 8 9 1

Mutation – In order to avoid from getting stuck onto a local minimum, population diversity is required to be kept up to some extent. In genetic algorithms, this is achieved by the help of a mutation mechanism, which causes some sudden changes on the traits of individuals according to a predefined mutation probability parameter. Mutation operation is performed in a single chromosome to create a single new offspring by different ways either by flipping, inserting, swapping or sliding the allele values at two randomly chosen gene positions. For this study, inversion mutation (i.e., flipping) is applied to an individual after went through crossover process. The inversion mutation operator randomly selects two cut points in the chromosome, and it reverses the subtour between these two cut points. Suppose that the first cut point is chosen between task 9 and task 5, and the second cut point between the 6th and 7th task. For example, consider the chromosome

Parent: 3 9 **5 4 6 2** 7 1 8
This result in
Offspring: 3 9 **2 6 4 5** 7 1 8

E. Genetic Parameters

One of the main difficulties in building a practical GA is in choosing suitable values for parameters such as population size, crossover rate and mutation rate. The selections of parameter values are very depend on the problem to be solved.

Population size – The population size is the number of candidate solutions in any one generation. In principle, the population size should be sufficiently large such that the population associated with the solution space can be adequately represented. A larger population, however, needs larger computation cost in terms of memory requirement and computation time. The population size in this study is set to be approximately half the total number of tasks.

Crossover rate, Pc – Crossover probability or crossover rate is how often will be crossover performed. If there is no crossover, offspring is exact copy of parents, but this does not mean that new generation (population) consists of the same old individual. If there is a crossover, offspring is made from parts of parents' chromosome. Most GA literature suggests that crossover rate should be set between 0.5 and 1.0.

Mutation rate, Pm – Mutation rate is how often will be parts of chromosome mutated. If there is no mutation, offspring is taken after crossover or copy without any change. If mutation is performed, part of chromosome is changed. Generally, mutation acts as a background operator that provides a small amount of random search. Its purpose is to maintain diversity within the population and inhibit premature convergence due to loss of information. Mutation rate is often small which is in the range of 0.001 to 0.1.

Stopping condition – Since GA is a stochastic search model, it is quite difficult to formally specify a convergence criterion, as it is often observed that the fitness of a population may remain static for a number of generations before a superior string is found. If the GA has been correctly implemented, the population will evolve over successive generations so that the fitness of the best and average chromosome in each generation decreases/increases towards the global optimum. Commonly, the algorithm terminates when either a maximum number of generations has been produced, or a satisfactory fitness level has been reached for the population. If the algorithm has terminated due to a maximum number of generations, a satisfactory solution may or may not have been reached. In this study, numerical simulation experiments will be conducted based on a fixed maximum number of generations.

Generational gap – It is possible that the chromosome with the highest fitness value in a generation may not survive selection process. A parameter called the generation gap was defined to control the fraction of the population to be replaced in each generation. Therefore 10% of the best fitness values in the population is kept and preserve for the crossover process.

IV. EXPERIMENTAL RESULTS & DISCUSSIONS

In this section, we present computational experiment results to evaluate the effectiveness of our proposed approach. For all experiments, the GA procedure utilize roulette wheel selection, linear order crossover and inversion mutation to generate new candidate solutions in every generation. The proposed algorithms are coded in Matlab 2009b and run on a Pentium Core 2 Duo PC with 2.0 GHz and 2.0 GB RAM under Windows XP operating system. For each experiment, 10 trials of simulation are carried out and the minimum idle time is taken as an optimal solution. As the computational results were based on different computer platforms or hardware, we do not conduct performance comparison between the proposed GA and the other algorithms.

The proposed GA was tested on a two benchmark single-model ALB problems. The first experiment involve a simple ALB problem from Scholl and Becker [15] which consists of 10 tasks and 10 precedence constraints as depicted in Figure 2. The total tasks time and the predetermined cycle time are 48 min and 11 min, respectively. The parameter setting of the GA which is *Pc* and *Pm* is 0.6 and 0.1 respectively. The population size is set to be half of the instance size and maximum number of generation is set as a termination criteria. TABLE II shows the optimal solution obtained from the proposed GA in which the minimum idle time and number of station are 7 min and 5 stations respectively. These results are similar with the one obtained by [15] which used heuristic technique in their work. This confirms that the proposed GA procedure used in this study is well adapted to solve ALB problem.

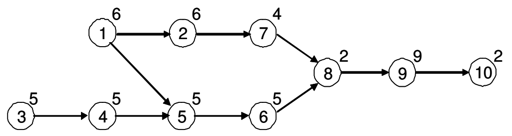

Figure 2. Precedence graph of 10 tasks

TABLE II. OPTIMAL SOLUTION FOR PROBLEM 1

Optimal task sequence: 1,3,2,4,5,6,7,8,9,10			
Station	**Assigned Task**	**Processing Time (min)**	**Idle Time (min)**
1	1,3	11	0
2	2,4	11	0
3	5,6	9	2
4	7,8	6	5
5	9,10	11	0
		Total Idle Time, T_{id}	**7**

The line efficiency, *E*=87.27%

The second experiment has use a benchmark data sets from Scholl [16] called Gunther problem which consists of 35 tasks and 45 precedence constraints as shown in Figure 3. The total tasks time and the predetermined cycle time of the Gunther problem are 483 min and 60 min, respectively. The GA parameter setting used are as follows; *Pc*=0.9, and *Pm*=0.01. The population size is set to be half of the instance size. The results in terms of the number of stations, total idle time, and the line efficiency are compared with the results obtained by S. Suwannarongsri and D. Puangdownreong [8] which used Tabu search to simulate the same problem. TABLE III shows the results obtained from the proposed GA used in this study. The utilization of 9 workstations gives a great reduction in idle time as well as improving the line efficiency. The idle time reduces 51.28% and line efficiency improves about 10% of the results reported from [8]. Therefore the proposed GA procedure use in this study outperforms the optimal solution obtained from the previous work. The performance graph in Figure 4 shows the best idle time found by the algorithm in each generation. The idle time reduced towards optimal solution as the generation increased and finally converged at a certain generation. This indicates that the genetic operators such as crossover and mutation used in the proposed algorithm are effective.

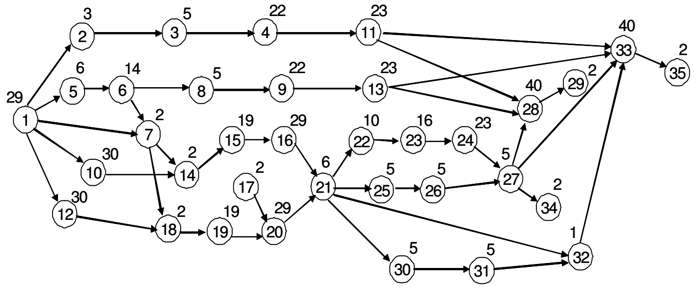

Figure 3. Precedence graph of Gunther problem

TABLE III. OPTIMAL SOLUTION FOR PROBLEM 2

Optimal task sequence: 1,10,5,6,8,9,13,2,3,4,7,14,15,16,12,18,19,11,17,20,21,30,22,23,31,32,24, 25,26,27,34,28,29,33,35			
Station	**Assigned Task**	**Processing Time (min)**	**Idle Time (min)**
1	1,10	59	1
2	5,6,8,9	47	13
3	13,2,3,4,7,14	57	3
4	15,16	48	12
5	12,18,19	51	9
6	11,17,20,21	60	0
7	30,22,23,31,32,24	60	0
8	25,26,27,34,28,29	59	1
9	33,35	42	18
		Total Idle Time, T_{id}	**57**

The line efficiency, *E*=89.44%

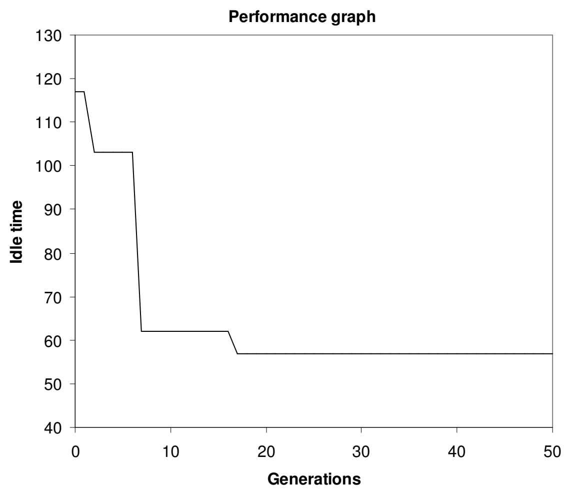

Figure 4. Performance graph for Gunther problem.

V. CONCLUSIONS

In this study, we have demonstrated how the genetic algorithm approach combined with a topological sort procedure can be applied for solving ALB problems with the objective that minimizes the total idle time in the workstation. The role of topological sort is to generate feasible task sequence while the GA is to further improve the quality of the solution. We presented two benchmark problems taken from the previous work and compare our results with them. The numerical results revealed that our approach is superior in terms of quality of the solution. The proposed GA well address the number of tasks assigned for each workstation giving a minimum idle time in the workstation as well as minimizes the number of stations for a given cycle time. The result of such solution would be increased production efficiency. The presented GA approach can thus be seen as the ideal compromise of optimizing complex and large problem and is thus highly recommendable for practical applications.

Our study is only concern on a simple model of an assembly line with a single objective optimization. Future research direction could be to further develop the proposed approach for solving different assembly line such as mixed-model production, two-sided lines and U-shaped lines, and might also to consider a multi-objective optimization for various ALB problems.

ACKNOWLEDGMENT

This study was supported by the Universiti Malaysia Pahang (UMP), Malaysia in collaboration with Dublin City University, Ireland.

REFERENCES

[1] S. Ghosh and R. J. Gagnon, "A comprehensive literature review and analysis of the design, balancing and scheduling of assembly systems," International Journal of Production Research, vol. 27, Issue 4, 1989, pp. 637-670.

[2] I. Baybars, "A survey of exact algorithms for the simple assembly line balancing problem," Management Science, 1986, 32(8): 909-932.

[3] E. H. Bowman, "Assembly line balancing by linear programming," Operations Research, 8(3), 1960, pp. 385-389.

[4] M. E. Salveson, "The assembly line balancing problem," Journal of Industrial Engineering, 6, 1955, pp. 18-25.

[5] M. Held, R. M. Karp, and R. Shareshian, " Assembly line balancing-Dynamic programming with precedence constraints," Operations Research, 11, 1963, pp. 442-459.

[6] S. B. Liu, K. M. Ng, and H. L. Ong, "Branch-and-bound algorithms for simple assembly line balancing problem," Int. Journal Adv. Manufacturing Technology, 2008, 36:169-177, doi:10.1007/s00170-006-0821-y.

[7] P. McMullen and G. Frazier, "Using simulated annealing to solve a multiobjective assembly line balancing problem with parallel work stations," International Journal of Production Research, vol. 44, 2006, pp. 27-42.

[8] S. Suwannarongsri and D. Puangdownreong, "Optimal assembly line balancing using tabu search with partial random permutation technique," International Journal of Management Science and Engineering Management, vol. 3, 2003, No. 1, pp. 3-18.

[9] J. Rubinovitz and G. Levitin, "Genetic algorithm for line balancing," International Journal of Production Economics, 41, pp. 343-354.

[10] B. Bryton, "Balancing of a continuous production line," M.S. Thesis, Northwestern University, Evanson, ILL, 1954.

[11] U. Ozcan nd B. Toklu, "Multiple-criteria decision-making in two-sided assembly line balancing: A goal programming and a fuzzy goal programming models," Computers & Operations Research, vol. 36, 2009, pp. 1955-1965.

[12] J. H. Holland, "Adaptation in natural and artificial system," Ann Arbor, Michigan: The University of Michigan Press, 1975

[13] C. Moon, J. Kim, G. Choi, and Y. Seo, "An efficient genetic algorithm for the traveling salesman problem with precedence constraints," European Journal of Operational Research, vol. 140, 2002, pp. 606-617.

[14] D.E. Goldberg and K. Deb, A comparative analysis of selection schemes used in genetic algorithms, in: G.J.E. Rawlins (Ed.), Foundations of Genetic Algorithms, Morgan Kaufmann, Los Altos, 1991, pp.69–93.Hdgjhd

[15] A. Scholl and C. Becker, "State-of-the-art exact and heuristic solution procedures for simple assembly line balancing," European Journal of Operational Research, vol. 168, 2006, pp. 666-693.

[16] A. Scholl, "Data of assembly line balancing problems", TH Darmstadt, http://www.assembly-line-balancing.de/