Journal of Mechanical Engineering and Automation 2012, 2(2): 25-35
DOI: 10.5923/j.jmea.20120202.05

A Cellular-rearranging of Population in Genetic Algorithms to Solve Assembly-Line Balancing Problem

Hossein Rajabalipour Cheshmehgaz[1,*], Mohammd Ishak Desa[1] , Farahnaz Kazemipour[2]

[1]Faculty of Computer Science and Information Systems, Universiti Teknologi Malaysia, Skudai, 81310, Johor, Malaysia
[2]Faculty of Management and Human Resources Development, Universiti Teknologi Malaysia, Skudai, 81310, Johor, Malaysia

Abstract Assembly line balancing problem (ALBP) is the allocating of assembly tasks to workstations with consideration of some criteria such as time and the number of workstations. Due to the complexity of ALB, finding the optimum solutions in terms of the number of workstations in the assembly line needs suitable meta-heuristic techniques. Genetic algorithms have been used to a large extent. Due to converging to the local optimal solutions to the most genetic algorithms, the balanced exploration of the new area of search space and exploitation of good solutions by this kind of algorithms as a good way can be sharpened with some meta-heuristic. In this paper, the modified cellular (grid) rearranging-population structure is developed. The individuals of the population are located on cells according to the hamming distance value among individuals as neighbours before regenerations and a family of cellular genetic algorithms (CGAs) is defined. By using the cellular structure and the rearrangements, some of the family members can find better solutions compared with others in the same iterations, and they behave much more reasonably in order to acquire the solution in terms of the number of workstations and the smoothly balanced task assignment on criteria conditions.

Keywords Cellular Genetic Algorithms, Assembly-line Balancing, Hamming Distance

1. Introduction

From ancient times to the modern day, assembly lines (ALs) have been modified as long as the long-term optimal design of the lines has surely been the most important milestone in the manufacturing process. Whereas designer of ALs (mostly) deal with many assembly tasks (around 400 tasks in a typical car ALs) and some critical limitations in design, the optimal solution can be achieved via a variety of heuristic ideas that must be massively computerized. Most of the work related to ALs concentrates on the assembly-line balancing (ALB) problem. The ALB problem deals with the assignment of the tasks (as duties) among workstations (or operators) so that the precedence relations are not violated, the total time for tasks in each workstation does not exceed the cycle time and a given objective function is considered to be optimized. The ALB problem falls into the NP-hard class of combinatorial optimization problems[1]. If there are *n* tasks and *r* preference constraints, then there are n!/2r possible task sequences[2]. Therefore, it can be time consuming for optimum-seeking methods to gain an optimal solution within this extremely large search space for some manufacturing operations, such as car assembly lines with more than 100 workstations.

* Corresponding author:
hrajabalipour@uk.ac.ir (Hossein Rajabalipour Cheshmehgaz)
Published online at http://journal.sapub.org/jmea

Despite the vast search space, many studies have tried to solve the ALB problem using optimum-seeking methods, such as linear programming[e.g. 3], integer programming[e.g. 4], dynamic programming[e.g. 5] and branch- and- bound approaches[e.g. 6]. However, none of these methods has proven to be of practical use for large assembly lines due to their computational inefficiency. Hence, the next research efforts have been directed towards the development of heuristics[e.g. 7, 8] and meta-heuristics such as simulated annealing[e.g. 9], tabu search[e.g. 10] and genetic algorithms[e.g. 11].

Due to the complexity of the ALB problem, a growing number of researchers have employed genetic algorithms (GAs), and most industrial engineers also use them to optimize problems which are difficult to find an optimal solution for in a reasonable time. GAs provides an alternative to traditional optimization techniques by using directed random searches to locate optimum solutions in complex search spaces. Hence, because of the popularity of GAs' application to the ALB problem, some papers exist which review the subject, including Dimopoulos and Zalzala[12], Scholl and Becker[13] and Tasan and Tunali[14] have all tried to modify GA using modified selection techniques, individual representation, crossover techniques etc. in order to improve the algorithms.

As having different priorities and objectives that GAs are trying to optimize as fitness functions, the ALB problem can be classified into some classes with different objectives. Additionally, the GAs' setting in chromosomes representa-

tions, initial population, and operation mechanisms were considered by GA designers to be like challenge and benchmarking.

1.1. ALB Problem Classes and their Objectives

Before making any decisions about assembly line design, the ALB problem must be classified according to the objectives and what needs to be developed. Figure 1 illustrates the classification of ALBP based on the objective function and problem structure[2,13,15-16].

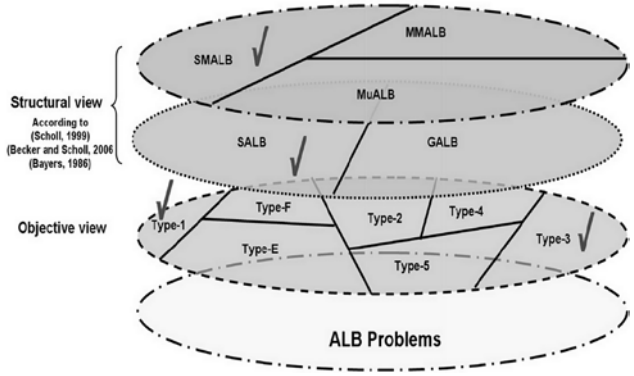


Figure 1. Classification of assembly line balancing problems and our research boundaries.

Scholl[16] has defined several versions of the ALB problem which have arisen by varying the objective function. Type-F is an objective-independent problem, which is to establish whether or not a feasible line balance exists for a given combination of n (number of workstations) and c (cycle time). Type-1 and Type-2 have a dual relationship; the first one tries to minimize the number of workstations for a given cycle time, and the second one tries to minimize the cycle time for a given number of workstations. Type-E is the most general problem version, which tries to maximize the line efficiency by simultaneously minimizing the cycle time and the number of workstations. Finally, Kim et al.[15] has explained that Type-3, 4 and 5 correspond to the maximization of workload smoothness, the maximization of work relatedness and multiple objectives with Type-3 and Type-4, respectively.

Based on the problem structure, ALB problems can be classified into two groups. While Becker and Scholl[16-17] have categorized that the first group includes single-model assembly-line balancing (SMALB), multi-model assembly-line balancing (MuMALB), and mixed-model assembly-line balancing (MMALB), the second group illustrated by Baybars[2], includes simple assembly-line balancing (SALB) and general assembly-line balancing (GALB). The SMALB problem involves only one product. The MuMALB problem involves more than one product produced in batches. The MMALB problem refers to assembly lines which are capable of producing a variety of similar product models simultaneously and continuously (not in batches). Additionally, the SALB problem, the simplest version of the ALBP and the special version of the SMALB problem, involves the production of only one product, where the assembly line has features such as a paced line with a fixed cycle time, deterministic independent processing times, no assignment restrictions, serial layout, one-sided workstations, equally equipped workstations and fixed-rate launching[14]. The GALB problem includes all of the problems that are not SALB, such as the balancing of mixed model, parallel, U-shaped and two-sided lines with stochastic dependent processing times; thereby, more realistic ALBPs can be formulated and solved.

Due to setting up expenditure, minimizing the number of workstations is frequently important than other criteria and, on the other side, the utility improvement of each workstation in terms of balancing the job among workstations in order to reduce the total idle time in manufacturing, is considerably more important. As a consequence, in this paper, as shown by the tickets in Figure 1, a combination of Type-1 and Type-3 based on objective, and; SMALB and SALB based on problem structure is considered and used.

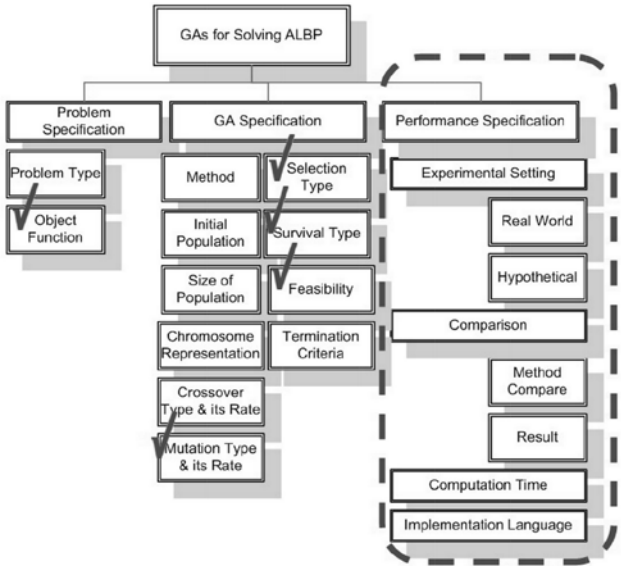


Figure 2. GAs Parameters and setting for ALBPs- sources:[14].

Depending on the problem to be solved by GAs, some GA parameters and structures can be influenced. As Tasan and Tunali[14] mentioned in their paper, they have reviewed the active literature based on specifications of problem, GA and performance. As seen in Figure 2, problem specifications contain the main features of the problems studied, GA specifications summarize information about the GA methods developed, initialization of the population, chromosome (individual) representation, fitness function, genetic operators, selection and survival schemes (elitism techniques), feasibility issues, and termination criteria (final conditions that specify the end of iteration in GA)- and finally, performance specifications include information about the data sets used to test GA, other solution methods to, which the performance of GA was compared, the computation time and the implementation language.

In[14], the authors have surveyed a lot of researchers' work that each has introduced and modified the GA for the ALB problem. Some of them worked on chromosome rep-

resentation[e.g. 18,19-20]; some of them have presented modified GAs on fitness function as the basis of objective function[e.g. 11, 21]. Some of the customized crossover operators utilized include the modified bin-packing crossover (modified BPCX)[11], the modified partially mapped crossover (modified PMX)[22], the heuristic structural crossover (HSX)[15], the informed order-based crossover[23], and the order crossover[19].

Tasan and Tunali[14] have reviewed the published literature based on the experimental settings in which the proposed GAs have been implemented, the other solution methods to which the GAs performance has been compared, the computation time required and, finally, the implementation language employed. However their paper has not tried to compare the performances of the reviewed GAs; it only presents the findings of the comparative studies reported in each work.

Genetic selection operation is a vital aim to find suitable parents to reproduce offspring and make a new population. The initial step always starts with a simple question "How important and influential is it that the selection of parent mechanism depends on similarities and diversities between parents". Based the literature, maintaining the population diverted on a cellular structure called cellular automaton (CA), was first introduced by Manderick and Spiessen[24]. They presented a cellular automata genetic algorithm (CAGA) as a kind of decentralized GA in which the population is arranged in a grid (usually two dimensions). CAGAs have been successfully implemented on a parallel platform used as a computational machine[25] and have also been used for optimization and simulation problems[26]. then Cao and Wu[27] firstly brought the hamming distance idea to CAGA in order to make desirable neighborhoods in CAGA. The individuals in the population were mapped onto a CA to make the locality and neighborhood. The mapping was based on the individuals' fitness and the hamming distances between individuals. The selection of individuals was control based on the structure of the CA, to avoid the fast population diversity loss and improve the convergence performance during the genetic search. The effectiveness of the CAGA was illustrated with two typical mechanical design optimization problems (Cao and Wu, 1998). Cheshmehgaz et al.[28] have employed the proposed CAGA to solve multi-model optimization problem, channel assignment problem in cellular mobile networks, the graph-coloring problem. The work has shown the effectiveness of CAGA.

In this paper, the grid (cellular) structure for the GA's population is used that makes the GA's selection operation restricted to find a mat for each individual. The structure can define a local selection for individuals of the population, so that individuals must be arranged on nodes of the grid (and sometimes rearranged in fixed iterations) and each arranged individual in the node can only match (do a crossover) with its neighbor's individuals (in neighbor nodes). The arrangement can be done on minimum hamming distance and maximum hamming distance value based upon its neighbors; and with regards to combination of it. Different kinds of neighbor definition are compared for an assembly line balancing case study with 83 tasks. Also, new mechanisms for mutation technique, objective function as fitness function, feasibility and survival type are modified and used.

The paper is organized as follows. In section 2, the typical GA and some structural frameworks used are presented and the grid structure applied for making the GA's population structure and for defining neighborhoods is explained and the cellular genetic algorithm (CGA) family is presented. Some individuals' arrangement and rearrangement techniques on the hamming distance basis are performed on a grid structure in order to make the hamming distance cellular GA family in section 3. In section 4, all kinds of arrangement mechanisms are tested and compared with the typical (conventional) genetic algorithm by using one large-size benchmark data set from Scholl[29]: data of assembly-line balancing problems.

2. Typical Genetic Algorithms for ALB

The structure of typical GA (TGA) is explained below as it performs one generation initial population, crossover and mutation operation in iteration.

Generate initial population
    Repeat
        Choose two individuals as parents for recombination
        Apply crossover with R_c probability
        Apply mutation with R_m probability
        Replace parents with offspring
    Until stopping condition is reached
Take the best chromosome of the population as the solution

GA specification is an initial step. Following subsections introduce task-based representation, initial population and crossover operation as long as new fitness function and mutation technique are introduced.

2.1. Representation

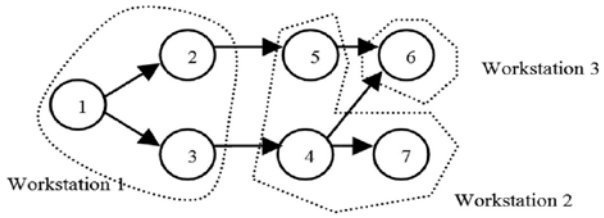
Sabuncuoglu et al.[19] has introduced task-based representation (TBR) where each task is represented by a number that is placed on a string (i.e. individual) with the string size equal to the number of tasks. The tasks are ordered by the individual relative to their order of processing. The tasks are allocated to workstations so that the sum of the task times in each station does not exceed the cycle time. This coding scheme is demonstrated in Figure 3 through a 7-task problem example as follows.

Example: we have seven tasks with own times shown in Figure 3.a, the precedence graph (matrix) are presented in Figure 3.b and the cycle time is equal to 20. Figure 3.c illustrates a string with an order of tasks that means tasks 1, task3 and task 2 are assigned to workstation 1, tasks 4, 5 and 6 are assigned to workstation 2 and finally task 7 is allocated to workstation 3. Figure 3.d shows another con-

figuration for the task balancing equaled with the string representation that is illustrated in Figure 3.e.

Task	1	2	3	4	5	6	7
Time	5	5	9	4	6	20	9

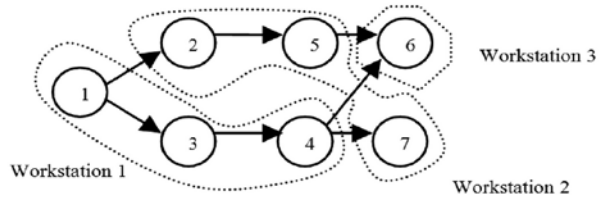
a) Tasks and their Times



b) A Precedence graph for tasks shown in a

1	3	2	5	4	7	6

c) a String in TBP form and giving Cycle Time=20
(Tasks assignment solution shown by dash line in b)



d) a Precedence graph for tasks shown in a with different task assignment

1	3	4	2	5	7	6

e) a String in TBP form and giving Cycle Time=20
(Tasks assignment solution shown by dash line in d)

Figure 3. An example shows Task-Base Representation for two different optimum solutions.

2.2. Objectives and Fitness Function Definition

There are many solutions for ALB but a few of them are better than others based on some objectives. However, the important objective of the ALB problem is to minimize the number of workstations but, in practical view, the balancing algorithm should also balance the total idle times among workstations too and provide a smooth balanced solution. In the previous example, the first configuration of task balancing, Figure 3.b, needs 3 workstations. The first workstation has tasks with total time: 5+5+9=19. The second workstation with total task time: 4+6+9, needs 19 units of time, and the third workstation needs 20 units of time. In the second configuration, Figure 3.d, the first workstation needs 18 units of time, the second one needs 20 and the third one needs 20 units of time. Although the total idle times for both cases is the same and equal to 2 units of time, the first configuration is more smoothly balanced based on idle time than the second.

Hence, we have defined a utility function for fitness function that consists of two objectives, i.e. minimizing the number of workstations and maximizing smoothness among workstations. The given and decision variables are introduced as follows.

Parameters:

- m: Number of tasks in AL (a given variable)
- n: Number of needed workstations (a decision variable)
- T_i: Task identity $0 < i \le m$
- W_j: Workstation identity $0 < j \le n$
- $A_{m,n}$: A binary matrix where its rows indicate the tasks and its columns indicate the workstations: $A(i, j)$'s value is 1 or 0. The value of 1 means that task i is assigned to workstation number j and 0 means not (a decision variable)
- $Time(T_i)$: T_i time (units of time) (a given variable)
- CT: Cycle time in AL(a given variable)
- $TotalTime(W_j)$: Total time the workstation j_{th} is busy, $TotalTime(W_j) \le CT$ (a decision variable)

$$\left[TotalTime(W_1), TotalTime(W_2), .., TotalTime(W_n)\right] = \left[Time(T_1), Time(T_2), ..Time(T_m)\right] \times A_{m,n} \quad (1)$$

- $IdleTime(W_j)$: Idle time in workstation $j_{th,}$ (a decision variable)

$$IdleTime(W_j) = CT - TotalTime(W_j) \quad (2)$$

- Root Sum Square (RSS): the most important parameter to specify smoothness in task assignment

$$\text{RSS (individual)} = \left(\sum_{k=1}^{n} \left(IdleTime(W_k) \right)^2 \right)^{1/2} \quad (3)$$

- Fitness function for each individual:

$$\text{Fitness (individual)} = \left(n \times RSS(Individual) \right)^{-1} \quad (4)$$

Equation (1) calculates the total time for each workstation that it needs according to $A_{m,n}$. Equation (2) specifies the idle time left in each workstation. Equation (3) illustrates a measure of balance and smoothness in the line by a solution and Equation (4) specifies the fitness value of a solution (used in this research).

2.3. Initial Population

The initial population is generated randomly assuring feasibility according to precedence relations. So, all individuals in the population in all generational steps will be feasible.

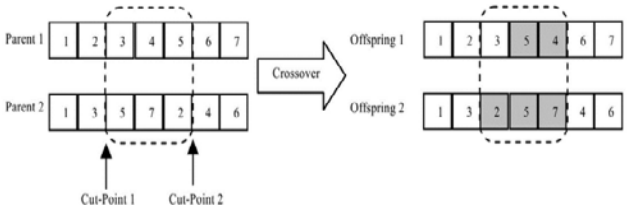


Figure 4. The crossover operation (gained from Sabuncuoglu et al., 2000).

2.4. Crossover Technique

The two parents that are selected are cut at two random cut-points. The offspring takes the same genes outside the cut-points at the same location as its parent and the genes in between the cut-points are scrambled according to the order that they have in the other parent. Sabuncuoglu et al.[19] presented the applicable crossover technique that we follow.

It is demonstrated in Figure 4. The major reason that makes the crossover operator important is that it assures feasibility of the offspring. Since both parents are feasible, both children must also be feasible. Keeping a feasible population is a key to the ALB problem since preserving feasibility drastically reduces computational effort.

2.5. Mutation Technique

In the mutation operation, a cut-point is randomly selected and the gene (bit) in this point is replaced with next gene (right-side) if it is possible (according to precedence limitations) and then all following genes will be exchanged to right-side gen as the same till there is no possibility to exchange. Figure 5 shows one mutation in the individual shown in Figure 3.e. As it is obvious, due to feasibility, gene 4 can be replaced with gene 2 and then with 5 till it cannot be exchanged.

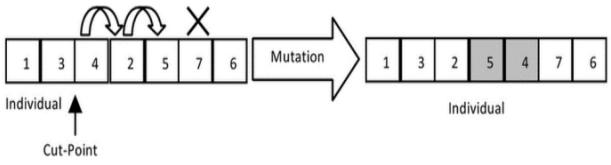


Figure 5. The mutation operation.

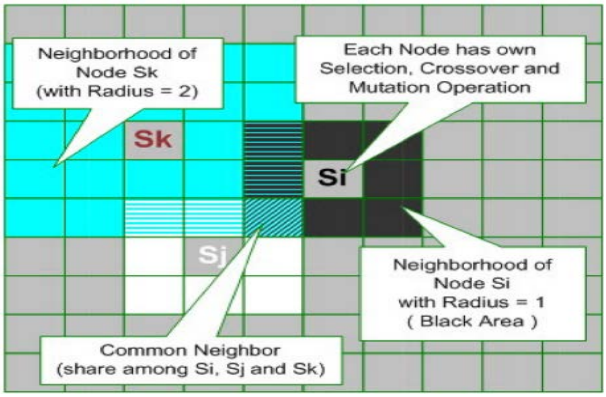


Figure 6. A 2-dimensional grid (cellular) structure with 3 nodes (cells) with different radius (1 and 2) and their neighbours.

3. Cellular-Rearranging of Population

3.1. Cellular (grid) Structure

The cellular structure (CS) used in this paper is a simplified version of cellular automaton (CA). CA is a collection of cells (nodes) on a grid of a specified shape that evolves through a number of discrete time steps according to a set of rules based on the states of neighboring cells. The rules are then applied iteratively for as many time steps as desired. Von Neumann is one of the first people to consider such a model, and incorporated a cellular model into his 'universal constructor' Cellular automata were studied in the early 1950s as a possible model for biological systems[30]. In this paper terminology, a CS comprises three components: $CS(N,R,O)$ which is explained below.

- N : size of CS (for instance, N =10 means a grid with 10*10 nodes – Figure 6 shows a 10*10 CS).
- R : radius of neighborhoods (figure 6 shows the two different radiuses with 1 and 2 – but in one CS, a unique radius amount must be used).
- O : a set of rules in each node and can be done simultaneously and separately.

The rules specify the states of nodes in the next time step. The state of node can be Boolean as active or inactive, or be a digit number, or even be binary strings. Let $Value^t(i,j)$ show the state of a node with location in i_{th} row and j_{th} column in CS at time t and $f_R(i,j)$ is a rule function that depends on the values of all nodes around node (i,j) and the neighbourhood with radius R and one can calculate the value of the node in the next time step. Equation (5) illustrates that $f_R(i,j)$ is calculated by function g which depends on all values of neighbors in the previous time step.

$$f_R(i,j) = g\left(\underset{\substack{i-R\le a\le i+R \\ j-R\le b\le j+R \\ 0<i<N, 0<j<N}}{Value^{t-1}}(a,b) \right) \quad (5)$$

The rules for GA used with CS include genetic selection, crossover and mutation operations that can be done concurrently in all nodes. One extra operation in nodes is 'replacement' that makes a new population by replacing new ones with olds and they must be done only after finishing one run of all genetic operations in all nodes. In the next subsection, the proposed genetic rule is going to be defined and presented.

3.2. Cellular Genetic Algorithms (CGA)

Sivanandam and Deepa[31] have classified GAs into five groups: simple GA, parallel & distributed GA, master-slave GA, coarse grained GA, and cellular GA. In the last group, the grid or fine-grained model individuals are placed on a large doughnut-shaped (the ends wrap around) one- or two-dimensional grid, one individual per grid location. The model is also called cellular because of its similarity with cellular automata with stochastic transition rules. Fitness evaluation is done simultaneously for all individuals and selection, reproduction and mating takes place locally within a small neighborhood. In time, semi-isolated niches of genetically homogeneous individuals emerge across the grid as a result of slow individual diffusion. This phenomenon is called isolation by distance and is due to the fact that the probability of interaction of two individuals is a fast decaying function of their distance. Recently Alba and Dorronsoro[32] have surveyed all conditions in CGA as a completed survey.

The following is a conventional the cellular genetic algorithmic ($P_{Crossover}$ and $P_{Mutation}$ are the parameters which show the probability of performing crossover and mutation operations set by a GA designer)[31-32]:

For each cell (node) *j* in the grid do in parallel
    Generate a random individual *j (feasible solution)*
End parallel for
While not termination condition do
    For each cell *j* do in parallel

Evaluate individual j (fitness function calculation)

Select a neighboring individual k

Produce offspring from j and k with probability $P_{Crossover}$

Mutate j with probability $P_{Mutation}$

Assign one of the best offspring to j

End parallel for

End while

In a more sophisticated view of CGAs, the new capsulated definition is presented. The capsulated definition helps to understand GAs in CS. First, the nodes must comprise two parts of information saved in the nodes in two successive time steps: $String^{t}(i,j)$ and $String^{t+1}(i,j)$ illustrate the genetic individuals (string/chromosome) at time steps t and $t+1$; and $Value^{t}(i,j)$ and $Value^{t+1}(i,j)$ are as previously defined in Section 3.1, but in two successive time steps. Secondly, the rule that calculating the values of the nodes comprises all genetic operation- selection, crossover and mutation - at once. The following pseudo codes show the modified rule function components and their order according to the genetic algorithms steps.

$$f_R(i,j)$$

$$\left\{\begin{array}{l} if\ RandomNumber(0,1) < P_{Crossover} \\ \quad Crossover(Selection_R(i,j)); \\ if\ RandomNumber(0,1) < P_{Mutation} \\ \quad Mutation(i,j); \\ \text{Re}\,placement(i,j); \end{array}\right\}$$

$$Selected-Neighbour \quad Selection_R(i,j)$$

$$\left\{\begin{array}{l} Sum = \sum_{a=i-R}^{i+R}\sum_{b=j-R}^{j+R} Value^{t}(a,b); \\ \qquad 0 < i \le N \\ \qquad 0 < j \le N \\ R = RandomNumber(0, Sum); \\ Sum = 0; \\ for\,(a = i-R; a \le i+R; a++) \\ \quad \left\{\begin{array}{l} for(b = j-R; b \le j+R; b++) \\ \quad \left\{\begin{array}{l} Sum = Sum + Value^{t}(a,b); \\ \quad if\ 0 < i \le N\ \&\ 0 < j \le N \\ \quad if\,(Sum > R) \\ \quad return(a,b); \end{array}\right\} \\ if\,(Sum > R) \\ \quad return(a,b); \end{array}\right\} \end{array}\right\}$$

$RandomNumber(a,b)$ function generates a random number between a and b, and $Selection_R(i,j)$ works by a local selection mechanism and neighborhood; radius R reveals a location of the neighbor which has already been selected for crossover operation in node (i,j). Fitness proportionate selection, also known as the roulette-wheel selection method, is employed by *Selection* operation[33]. The crossover and mutation operations that are mentioned in the pseudo code work as they are designed in TGA locally (see Section 2).

An issue in CGA is elitism strategy which means the best individuals in the old population should survive to the next population. Due to local selection in CGA, it cannot apply the strategy directly. In the final steps of CGA, replacement is considered and the best string (individual) among offspring based on fitness value would be a substitute for the old individual in node. The $replacment(i,j)$ function replaces the old value in the node with the best value gained from the last two time steps (see pseudo code following). And the value of individual fitness that is saved as $Value^{t}(i,j)$ in the node must be calculated again by $Calculation()$.

$$replacment(i,j)$$

$$\left\{\begin{array}{l} if\,(Value^{t}(i,j) < Value^{t-1}(i,j)) \\ \quad String^{t}(i,j) = String^{t-1}(i,j); \\ Calculation(Value^{t}(i,j)); \end{array}\right\}$$

In this paper, there are also some modifications in CGA as follows.

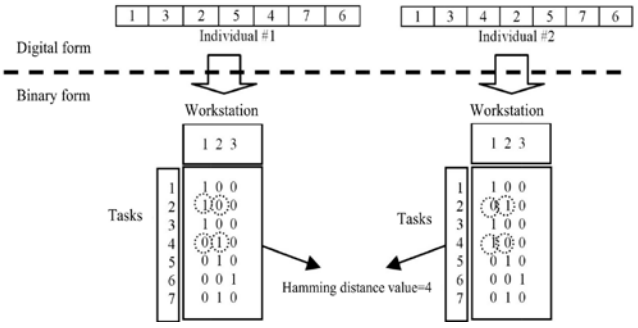


Figure 7. Transfer process to binary form and hamming distance calculation.

3.3. Hamming distance (HD)

Making the selection operation restricted is well-known strategy to prevent genetic drifting[34]. The grid structure can be usefully used to make the restriction[32]. In order to have a variety mates to be selected by individuals, in this research, the hamming distance parameter is considered specifying the similarity value among individuals. Each individual might be transferred to binary form as the assignment matrix (see Figure 7 – the columns specify the workstations number and the rows specify the tasks number in the assignment matrix, and the '1' or '0' symbol in each element illustrates which task is or is not assigned to which workstation) and then the value of the hamming distance can be calculated by counting the different 0s and 1s in genotype of individuals. Figure 7 illustrates the transfer process from phenotype (digital) form (task-based representation) to genotype (binary) form (assignment matrix) and the concept of similarity between individuals by using the hamming distance value. The example (in Figure 7) shows the value of HD between individual #1 and #2 is 4

or, in another definition, $HD(\#1,\#2)=4$.

It is assumed that if mates are similar (or dissimilar) to each other they would be better parents to make offspring. To put this idea to the test, some steps in CGA need to be modified. The next subsection shows a new change in CGA in the new step, 'cellular rearrangement' and it seems to work like pre-processing before doing any genetic operation in each generation or frequently.

3.4. Rearrangements and CGA family

Some kinds of arrangement are selected to locate the individual on a grid's nodes. By the arrangements we try to test the effectiveness of making the neighborhood. For instance, what if one individual could have a greater chance to have crossover with another individual that is so similar to or different to it.

In this research, three kinds of arrangements techniques are used: max-hamming distance, min-hamming distance and max-min-hamming distance, and then all are mixed with CGA as an added step and make the CGA family (Figure 8 shows the new framework of the CGA family).

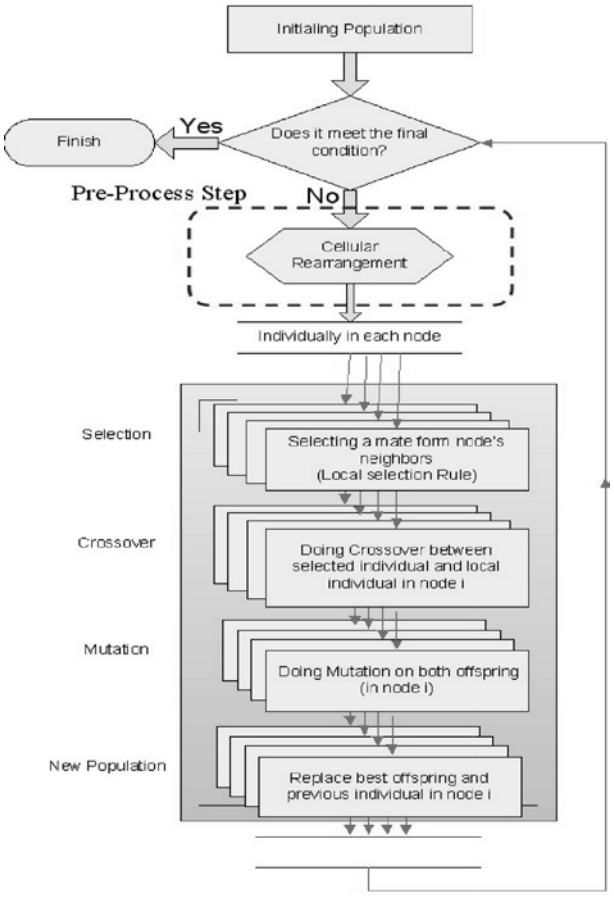


Figure 8. Flowchart Abstract of CGA Family.

Max-Hamming distance cellular genetic algorithm (MaxHCGA): it makes a variety of dissimilarities between the neighboring nodes. Actually, in the max-hamming distance rearrangement, the objective in individual arrangements can be formulated by (6).

$$Maximizing\left(\sum_{i=1}^{N}\sum_{j=1}^{N}\sum_{\substack{a=i-R\\0<a\le N}}^{i+R}\sum_{\substack{b=j-R\\0<b\le N}}^{j+R} HD(String^{t}(i,j),String^{t}(a,b))\right) \quad (6)$$

To make the simplest and most heuristic way to follow the objective, all steps in MaxHCGA are the same as CGA steps but only one step must be added: cellular rearrangement. After making the initial population, according to Section 2.3, one individual from the population is randomly chosen. The chosen individual is allocated into the node called pivot and then for all empty neighbors of the pivot, the rearrangement method continues to find individuals from the rest of the population that have the maximum hamming distance value with the individual in the pivot. The rearrangement method would work for all nodes that have an individual in and at least one empty neighbor. The rearrangement can be repeated each generation or different frequently. In this paper, the rearrangement is performed before each genetic generation to realize the effectiveness of the arrangement.

Min-Hamming distance cellular genetic algorithm (MinHCGA): as it is compared with MaxHCGA, only the rearrangement method's objective should be changed. In spite of using maximum hamming distance value, MinHCGA uses minimum hamming distance value for the rearrangement method. As a consequence, all individuals have more chance to meet individual who are more similar to. The objective considered in the rearrangement is formulated by (7).

$$Minimizing\left(\sum_{i=1}^{N}\sum_{j=1}^{N}\sum_{\substack{a=i-R\\0<a\le N}}^{i+R}\sum_{\substack{b=j-R\\0<b\le N}}^{j+R} HD(String^{t}(i,j),String^{t}(a,b))\right) \quad (7)$$

Max-Min-Hamming distance cellular genetic algorithm (MMHCGA): a combination of rearrangements of Max-HCGA and Min-HCGA makes MMHCGA which means that, alternately, Max-HCGA rearrangement and Min-HCGA rearrangement can be used. Let $MaxNeighbours(i,j)$ represent a set of all the neighbors around node (i,j) that are supposed to have maximum hamming distance valued with the individual allocated in node (i,j), and $MinNeighbors(i,j)$ embody a set of the neighbors who are supposed to have minimum hamming distance value with the individual in node (i,j). The objective of rearrangement in MMHCGA can be presented by two formulations (8) and (8).

$$Maximizing\left(\sum_{i=1}^{N}\sum_{j=1}^{N}\sum_{S\in MaxNeighbours(i,j)} HD(String^{t}(S),String^{t}(a,b))\right) \quad (8)$$

$$Minimizing\left(\sum_{i=1}^{N}\sum_{j=1}^{N}\sum_{S\in MinNeighbors(i,j)} HD(String^{t}(S),String^{t}(a,b))\right) \quad (9)$$

For example, Figure 9 illustrates four stages of such a simple rearrangement method in MMHCGA explained by formulas (8) and (9). First, one node in CA as a pivot cell and one individual as a pivot individual randomly are chosen and the individual will be assigned to the cell. The arrangements will be continued for the pivot's empty neighbors as following. For the pivot cell's neighbors- the

nodes in black in Figure 9.a- the maximum hamming distance value is considered, and for the others –the nodes in white –the minimum hamming distance value is used. To continue, in Figure 9.b, one of the grid's none-empty nodes is randomly chosen, and the rearrangement is repeated until all nodes have one individual.

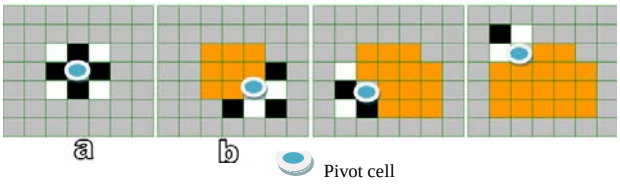


Figure 9. Initial four steps of MMHCGA Rearrangement.

4. Simulation and Results

To analyse the CGA algorithms family, a case with 83 assembly tasks were selected[from 29]. The given cycle time for all workstations is 5000 units of time. To make the results quite clear and also finding the differences between the CGA family and typical (ordinary non-cellular) GA, the following parameters values were fixed, but the values could be changed to different values also.

The time of each task has been shown in Table 1. Each algorithm executed from the initial population created randomly, by 100 executions with 300 iterations in each. The best individual and the worst individual (based on fitness value) in new populations generated by the algorithms, were recorded at the end of iterations of 10, 20, 30… and 300. And the average fitness value at the end of the iterations for all CGA family algorithms were also calculated and recorded.

An individual that has used the minimum number of workstations and has minimal RSS (root sum square) of idle times was identified as a relative best task balancing solution here. We compared the all member of CGA family with each other and the typical GA (TGA) on these circumstances.

As the GA parameters part involved, to set the parameters for GA, the selection rule that is used is the roulette selection technique, and other parameters are fixed as follows.

Genetic parameters setting:

- Population size: 100
- Selection rule: roulette wheel selection
- Crossover rate: 0.9
- Mutation rate: 0.2
- Elitism rate: 0.05 (only for TGA)
- Iteration (reproduction of population) number: 300
- Number of Executions (for sampling): 100

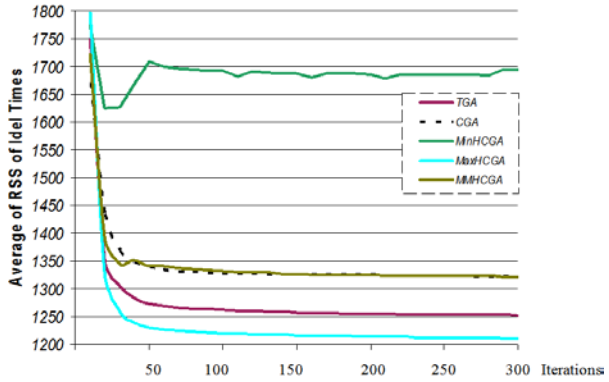
As mentioned before, the best solution would be the one (set of) individual(s) that has(have) minimum number of workstation (that it needs) and minimum RSS of idle times (maximum smoothness of tasks) by giving CT=5000.

Table 1. Precedence table of the case study.

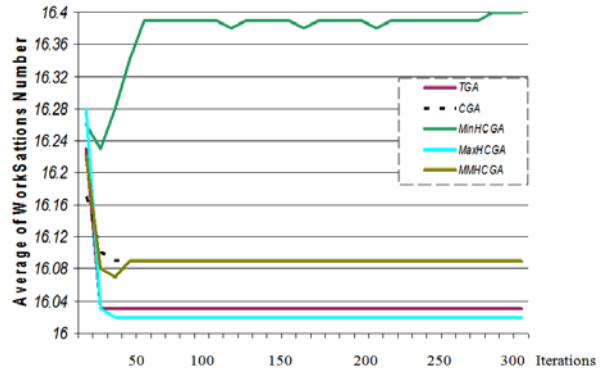
Tasks T_i	Time	Precedent Tasks	Tasks T_i	Time	Precedent Tasks	Tasks T_i	Time	Precedent Tasks
1	1673	2	29	714	30	57	578	58
2	985	3, 4, 5	30	1004	31	58	578	59
3	1836	6	31	713	39	59	578	60
4	973	6, 7	32	642	33, 34, 35,36	60	578	61
5	1700	8	33	629	37	61	578	62
6	2881	9, 10	34	1234	77, 78	62	578	63
7	2231	11	35	1143	77, 78	63	578	64
8	1040	77, 78	36	1266	38, 39	64	578	65
9	1793	10, 12	37	792	40	65	578	66
10	1250	11, 13, 14, 25	38	1251	41	66	578	67
11	700	12, 15	39	1310	42, 43, 44, 75	67	578	68
12	464	13, 16	40	663	77, 78	68	578	74, 75
13	500	14, 17, 18, 20	41	494	45	69	467	70, 71
14	1133	15, 19	42	1288	77, 78	70	887	72
15	577	16, 20, 39,	43	792	77, 78	71	396	73
16	483	17, 77, 78	44	578	46	72	1296	73
17	880	18, 21, 22, 28	45	594	47	73	1100	74
18	667	19, 23	46	578	48	74	2543	76
19	600	24	47	622	49	75	764	76
20	233	26	48	578	50	76	357	76
21	408	27	49	564	69	77	701	77, 78
22	849	27	50	578	51	78	1164	79
23	767	74	51	578	52	79	286	80, 81
24	850	28,29	52	578	53	80	2100	82
25	780	32	53	578	54	81	450	83
26	912	74	54	578	55	82	1300	83
27	748	69	55	578	56	83	3691	-
28	1863	32	56	578	57			

Figure 10, illustrates behaviors of five considered genetic algorithms including TGA (conventional non-cellular GA) and CGA family employed for the case study. As Figure 10.a shows, MaxHCGA has best behavior in decreasing RSS of idle time average as long as the number of iterations is going up. In contrast to MaxHCGA, MinHCGA behaves in worst manner which means that there is no decline in the average of RSS that is expected as the number of iterations is increasing.

Considering the average of number of workstations, Figure 10.b illustrates that MaxHCGA notably attain a better solution compared with others (16.02 as workstations average needed).



a) The changes of the RSS average values along the iterations in TGA and CGA family (after 100 runs for each).



b) The changes of the average numbers of workstations needed along the iterations in TGA and CGA family (after 100 runs for each).

Figure 10. CGS family behaviors in different iterations (on average values of RSS and needed workstations number basis).

In addition, Figure 11 shows the minimal RSS and maximal RSS of idle times, for all executions, in different iterations (for each algorithm). The values of both the parameters show that MaxHCGA works as well as it has been expected. The minimum RSS of idle times reached by MaxHCGA is about 1148.5 in the last iteration, 300 iterations (Figure11.a) and also the maximum RSS for MaxHCGA is better than others (Figure 11.b – about 2409).

Figure 12 presents the two examples of the best solutions attained by MaxHCGA and TGA at the final iteration. Each colored-row in Figure 12 is nominated as one workstation, ordered from number 1 to 16, and tasks have been marked by colored bars and the task times underneath. The idle time bars of workstations are specified in thick dash-rectangles and their values are in front.

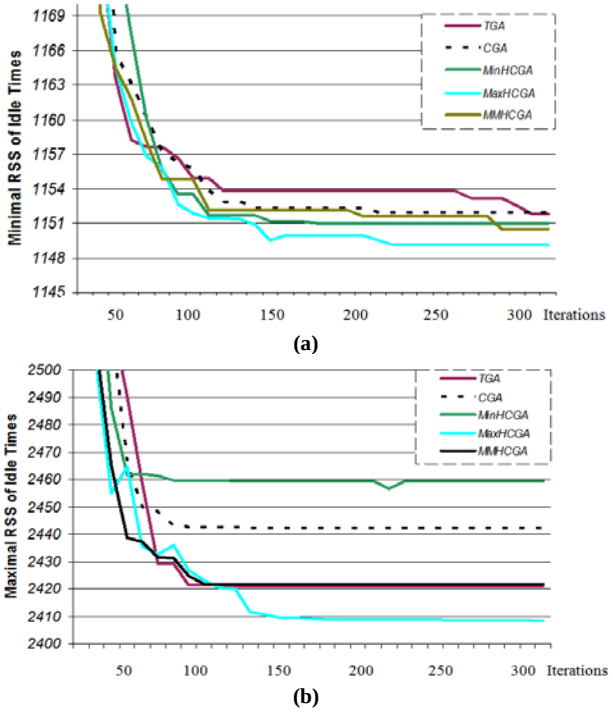
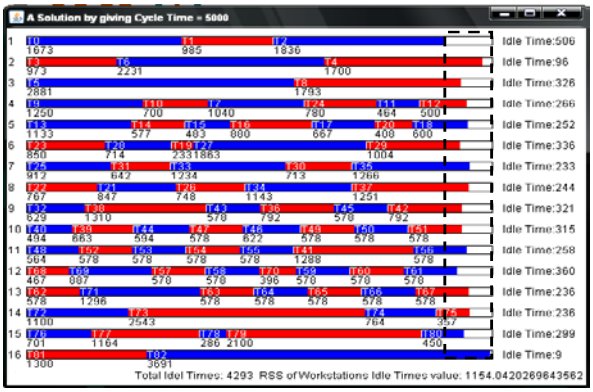
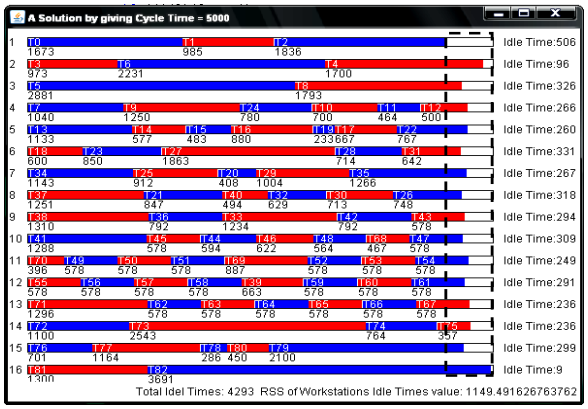


Figure 11. CGS Family behaviors in different iterations (on minimum and maximum value of RSS basis).



a) Best solution gained at iteration while running TGA by 100 runs



b) Best solution gained at final iteration while running MaxHCGA by 100 runs

Note: None Colored rectangles inside the dash-rectangle show the idle time for each workstation

Figure 12. Two best solutions of MaxHCGA and TGA by iteration 100.

In spite of TGA, MaxHCGA has found the solution with the greater smoothness level. However both solutions present the same number of workstations (equal to 16 workstations) and total idle times (= 4293), but they have reached a different RSS of idle times as smoothness (RSS for MaxHCGA is 1149.49 and for TGA is 11.54.04). It means that MaxHCGA has found a solution which is more smoothed.

In this research, a modified cellular GA was developed to solve the single-model ALB problem comprised of Type-1, Type-3, SALB and SMALB. Some heuristic rearrangement techniques based on hamming distance values were utilized and then the CGA family was defined. We showed that each member of the CGA family initiates a chance to individuals put on cells to find a particular mate from their neighbors (by a local selection mechanism), and then tries to do all genetic operations: crossover, mutation and replacement, locally and individually. With the aim of the rearrangements techniques, the concept of exploration/exploitation has been sharpened and examined for the ALB problem case. In the ALB case, the similarity between individuals calculated by hamming distance value illustrates how much similarity exists between task assignments to the workstations proposed by individuals. The main point of this paper showed how much the similarity and dissimilarity between parents is useful for making offspring for the next generation. TGA as conventional non-cellular GA, CGA, MinHCGA, MaxHCGA, MaxMinHCGA were all put on a test given by a test case, and then the comparison results show how much the similarity and dissimilarity in selection operations have been influenced. We concluded that MaxHCGA mostly achieves the solutions that have the minimal root sum square of idle times with comparing to other CGA family members and TGA.

5. Conclusions

Although two conventional criteria; minimum workstations and idle times, were considered here, other pragmatic criteria such as material handling constraints, resource limitations, human factors and multiple or mixed-model assembly manufacturing issues can be considered as further research directions.

REFERENCES

[1] R. M. Karp, "Reducibility among combinational problems. Complexity of computer applications," R. E. Miller and J. W. Thatcher., Eds., ed New York: Plenim Press, 1972, pp. 85-104

[2] I. Baybars, "A survey of exact algorithms for the simple assembly line balancing problem," *International Journal of Management Science,* vol. 32, pp. 909–932, 1986

[3] M. Peeters and Z. Degraeve, "An linear programming based lower bound for the simple assembly line balancing problem," *European Journal of Operational Research,* vol. 168, pp. 716-731, Feb 1 2006

[4] V. V. Shkurba and S. A. Beletskii, "Numerical methods for assembly-line balancing (survey) " *Cybernetics and Systems Analysis,* vol. 1, pp. 96–108, 1977

[5] J. Bautista and J. Pereira, "A dynamic programming based heuristic for the assembly line balancing problem," *European Journal of Operational Research,* vol. 194, pp. 787-794, May 1 2009

[6] A. Sprecher, "A competitive branch-and-bound algorithm for the simple assembly line balancing problem," *International Journal of Production Research,* vol. 37, pp. 1787-1816, May 20 1999

[7] K. Fleszar and K. S. Hindi, "An enumerative heuristic and reduction methods for the assembly line balancing problem," *European Journal of Operational Research,* vol. 145, pp. 606-620, Mar 16 2003

[8] R. Gamberini, *et al.*, "A new multi-objective heuristic algorithm for solving the stochastic assembly line rebalancing problem," *International Journal of Production Economics,* vol. 102, pp. 226-243, Aug 2006

[9] P. R. McMullen and G. V. Frazier, "Using simulated annealing to solve a multiobjective assembly line balancing problem with parallel workstations," *International Journal of Production Research,* vol. 36, pp. 2717-2741, Oct 1998

[10] S. D. Lapierre, *et al.*, "Balancing assembly lines with tabu search," *European Journal of Operational Research,* vol. 168, pp. 826-837, Feb 1 2006

[11] E. Falkenauer and A. Delchambre, "A Genetic Algorithm for Bin Packing and Line Balancing," presented at the the 1992 IEEE International Conference on Robotics and Automation, Nice, France, 1992

[12] C. Dimopoulos and A. M. S. Zalzala, "Recent developments in evolutionary computation for manufacturing optimization: Problems, solutions, and comparisons," *Ieee Transactions on Evolutionary Computation,* vol. 4, pp. 93-113, Jul 2000

[13] 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, pp. 666-693, Feb 1 2006

[14] S. O. Tasan and S. Tunali, "A review of the current applications of genetic algorithms in assembly line balancing," *Journal of Intelligent Manufacturing,* vol. 19, pp. 49-69, Feb 2008

[15] Y. J. Kim, *et al.*, "A heuristic-based genetic algorithm for workload smoothing in assembly lines," *Computers & Operations Research,* vol. 25, pp. 99-111, Feb 1998

[16] A. Scholl, *Balancing and Sequencing of Assembly Lines* Physica-Verlag HD; 2nd edition, 1999

[17] C. Becker and A. Scholl, "A survey on problems and methods in generalized assembly line balancing," *European Journal of Operational Research,* vol. 168, pp. 694-715, Feb 1 2006

[18] D. A. Ajenblit and R. L. Wainwright, "Applying genetic algorithms to the U-shaped assembly line balancing problem," presented at the the 1998 IEEE international conference on evolutionary computation, Anchorage, Alaska,

USA, 1998

[19] I. Sabuncuoglu, *et al.*, "Assembly line balancing using genetic algorithms," *Journal of Intelligent Manufacturing*, vol. 11, pp. 295-310, May 2000

[20] D. J. Stockton, *et al.*, "Use of genetic algorithms in operations management - Part 1: applications," *Proceedings of the Institution of Mechanical Engineers Part B-Journal of Engineering Manufacture*, vol. 218, pp. 315-327, Mar 2004

[21] E. C. Brown and R. T. Sumichrast, "Evaluating performance advantages of grouping genetic algorithms," *Engineering Applications of Artificial Intelligence*, vol. 18, pp. 1-12, Feb 2005

[22] Y. Tsujimura, *et al.*, "Solving Fuzzy Assembly-Line Balancing Problem with Genetic Algorithms," *Computers & Industrial Engineering*, vol. 29, pp. 543-547, Sep 1995

[23] K. C. C. Chan, *et al.*, "Handling the assembly line balancing problem in the clothing industry using a genetic algorithm," *International Journal of Clothing Science and Technology*, vol. 10, pp. 21 - 37, 1998

[24] B. Manderick and P. Spiessen, "Fine-Grained Parallel Genetic Algorithms," presented at the Proceedings of the 3rd International Conference on Genetic Algorithms, Fairfax, Virginia, USA, 1989

[25] E. Alba and B. Dorronsoro, "The exploration/exploitation tradeoff in dynamic cellular genetic algorithms," *Ieee Transactions on Evolutionary Computation*, vol. 9, pp. 126-142, Apr 2005

[26] S. Janson and M. Middendorf, "A hierarchical particle swarm optimizer and its adaptive variant," *Ieee Transactions on Systems Man and Cybernetics Part B-Cybernetics*, vol. 35, pp. 1272-1282, Dec 2005

[27] Y. J. Cao and Q. H. Wu, "A cellular automata based genetic algorithm and its application in mechanical design optimization," presented at the UKACC International Conference, Swansea, UK, 1998

[28] H. R. Cheshmehgaz, *et al.*, "The Improved Genetic Algorithm for Assignment Problems," *Proceedings of the 2009 International Conference on Signal Processing Systems*, pp. 187-191, 1016, 2009

[29] A. Scholl, "Data of assembly line balancing problems," *Schriften zur Quantitativen Betriebswirtschaftslehre 16/93, TU Darmstadt*, 1993

[30] S. Wolfram, *A New Kind of Science*, 1 ed. USA: Wolfram Media, 2002

[31] S. N. Sivanandam and S. N. Deepa, *Introduction to Genetic Algorithms*, 1 ed.: Springer, 2007

[32] E. Alba and B. Dorronsoro, *Cellular Genetic Algorithms*. USA: New York: Springer Science+Business Media, LLC, 2008

[33] T. Bäck, *Evolutionary Algorithms in Theory and Practice Evolution Strategies, Evolutionary Programming, Genetic Algorithms* New York: Oxford University Press, 1998

[34] C. A. C. Coello, *et al.*, *Evolutionary Algorithms for Solving Multi-Objective Problems.* New York, USA: Springer, 2007