Journal of the Nigerian Association of Mathematical Physics
Volume27, (*July, 2014*), pp 549 – 562

Computer Software for Solving Assembly Line Balancing Problem

Edokpia, R.O.[1], Okonta, C.I.[2] and Audu, H.A.P.[3]

[1,2]Department of Production Engineering, University of Benin, Nigeria
[3]Department of Civil Engineering, University of Benin, Nigeria.

Abstract

The problem of assembly line balancing is a non-deterministic polynomial-time-hard optimization problem. We presented in this work computer software for solving SALBP based on genetic algorithm, using object oriented approach of Visual Basic.net. The methodology utilises three different priority-based heuristics and Genetic Algorithm (GA) in solving assembly line balancing problem. The GA also adopts a fitness function based on realized cycle time and a crossover based on fitness ranking.

Keywords: Object Oriented Programming; Assembly Line Balancing; Heuristic Encoded Genetic Algorithm.

Keywords: Diffraction, knife edge, Line of Sight

1.0 Introduction

Assembly lines are flow-line production systems, where a series of workstations, on which interchangeable parts are added to a product, are linked sequentially according to the technological restrictions [1]. The assembly line balancing problem is a decision problem on how best tasks are to be assigned to the various workstations in order to increase efficiency. This increase in efficiency can be approximated to the minimization of the number of workstations and the maximization of the production rate. There are three reasons why assembly lines were developed. They are for a cost-efficient mass-production of standardized products, designed to exploit a high specialization of labor and the associated learning effects [2].

The variable of interest for the ALB consists of number of tasks, processing time, precedence relationships, and the cycle time. The goals of the ALB are to minimize the number of workstations (m), minimize the workload variance, minimize the idle time, and maximize the line efficiency [3]. If a single product is produced on a line, then the problem is called simple assembly line balancing problem (SALBP). SALBP can be classified into three groups according to the objectives;

– SALBP-1: the objective is to minimise the number of stations on the line for a given cycle time.
– SALBP-2: the objective is to minimise the cycle time for a given number of stations on the line.
– SALBP-E: the objective is to maximise the line efficiency for variable cycle time and number of Stations [4].

Balancing the assembly line needs some constraints, as follows:

i. Precedence constraint should be satisfied.
ii. The cycle time is greater than or equal to the time of any work element.
iii. The workstation time should not exceed the cycle time [5].

Though SALBP is a class of NP-hard optimization problems; effective exact methods are available in solving small and medium-size problems. Approximate methods (heuristics and metaheuristics) have been developed in order to overcome the size limitation of the exact methods aiming at providing good solutions that are as near to the optimal solution as possible. Nevertheless, further algorithmic improvement is necessary for solving large-scale problems [6].

Genetic algorithms are adaptive heuristic search algorithm premised on the Darwin's evolutionary ideas of natural selection and genetic. The basic concept of genetic algorithms is designed to simulate processes in natural system necessary for evolution. As such they represent an intelligent exploitation of a random search within a defined search space to solve a problem [7]. GA is a global optimization algorithm derived from evolution and natural selection.

The basic thought of Genetic algorithm include

i. Randomly producing an original population whose number of individuals is a constant N.
ii. Producing next generation by crossing over and mutation among individuals.
iii. Forming the new population of N individuals from the generation of ii.

Corresponding author: Edokpia, R.O., E-mail: ralphedokpia@yahoo.com, Tel.: +2348023368811

iv. Producing the next population by repeating the step ii and iii until obtaining the individual which satisfies conditions [8].

While genetic algorithms have the advantage of not getting stuck in local optima, they have other problems. When the search space is very large then genetic algorithm methods generally take a long time to converge to good quality solutions [9].

In order to find optimal solution to the ALB problem via GA methods, four critical elements are required. First, an appropriate representation is required. This is accomplished by representing a task sequence in terms of chromosome. Second, a fitness function is required to evaluate the quality of different potential solutions. Third, a set of genetic operators (parent selection, crossover and mutation) which generate new chromosomes as a function of older chromosomes must be defined. Finally, algorithm parameters must be decided [10].

The purpose of this work is basically to develop and implement computer software that would greatly enhance an accurate evaluation of simple assembly line balancing problem using genetic algorithm. Thus the problem addressed was:

i. To provide very accurate and efficient solutions for ALBP.
ii. To reduce the computational complexity and the arduous task usually encountered in solving line balancing problems.
iii. To provide a visual display of the solution so as to easily and speedily interpret these solutions.

2.0 Design of Software Model

In designing the software, The Genetic Algorithm procedure followed is stated in [11].

Synopsis of the Genetic Algorithm

[Start] Generate random population of *n* chromosomes (suitable solutions for the problem)
[Fitness] Evaluate the fitness *f(x)* of each chromosome *x* in the population
[New population] Create a new population by repeating following steps until the new population is complete
[Selection] Select two parent chromosomes from a population according to their fitness
[Crossover] With a crossover probability, cross over the parents to form a new offspring
[Mutation] With a mutation probability mutate new offspring
[Accepting] Place new offspring in a new population
[Replace] Use new generated population for a further run of algorithm
[Test] If the end is satisfied, **stop**, and return the best solution in current population
[Loop] Go to step **2**

The Flow diagram of genetic algorithm is shown in Figure 1.

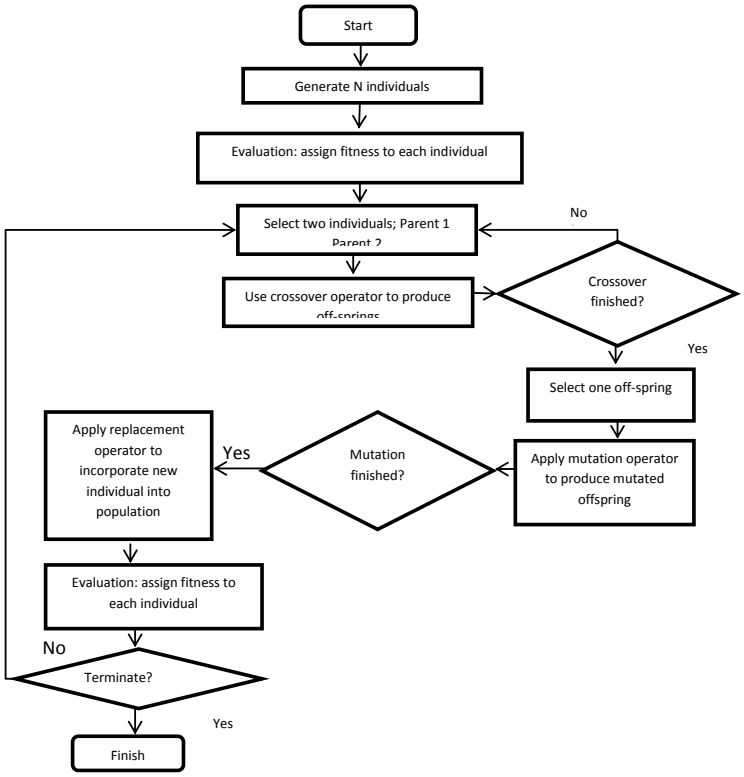


Figure 1: Flow diagram of genetic algorithm

The procedures were carefully programmed using visual basic.Visual Basic.Net is an object oriented programming language, comprehending the syntax and the effects of the programming language elements is obviously an essential part of learning a language, but appreciating how the language features work and how they are used is equally important. Rather than just using code fragments, it provides us with practical application to solving specific problems. These applications can then provide a basis for us to experiment and see the effects of changing the code in various ways. VB.Net is a powerful and compact computer language that allows you to write programs that specify exactly what you want your computer to do. You're in charge: you create a program, which is just instructions, and your computer will follow them. There are four fundamental stages in the creation of any VB.Net program, they are Editing, Compiling, Linking, Executing as shown in figure 2.

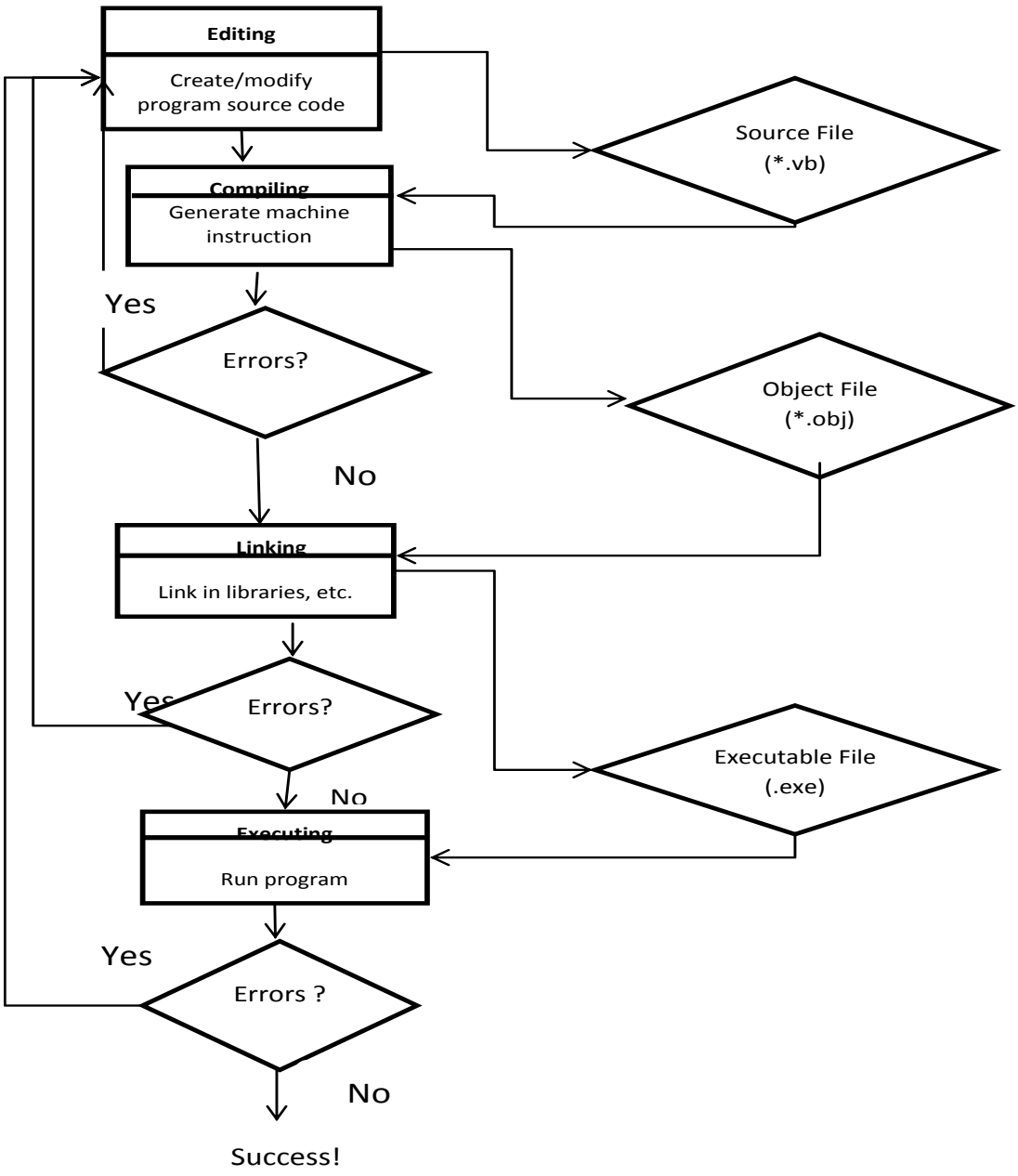


Figure 2: Creating and executing a program in visual basic.ne

In the design of the software, the relevant equations and models were assembled. Also visual representation of the solutions to problems identified was developed in the form of table solutions. The software was tested on a variety of problems and the

solutions obtained were accurate, thus validating the accuracy of the software. The flow chart for the software design is shown in figure 3. The following is the algorithm on which the software is based:

Start
If Form1.load = **True**, **Then**
If Button3. Click = **True**, **Then**
If Form2.load = **True**, **Then**
If Button2. Click = **True**, **Then**
ovalshapes.show =**True**
End if
Input known parameter and assign to Ovalshape from: task, time and cycle time
If Button3. Click = **True**, **Then**
Ovalshape.Highlight and Inputbox.Show
Input task precedences
End if
End if
If Button4. Click = **True**, **Then**
If Form4.load = **True**, **Then**
If button1.click = **True**, **Then**
If Inputbox.Show = **True, Then**
Input the value of *a*
End if
If DataGridView1.Show, DataGridView2.Show, DataGridView3.Show, DataGridView4 = **True Then**
Generate a table of the initial population
Compute the fitness of the population using equations 1 and 2
Input the value of *x1*, *x2* and *mp* to generate new population using equation 3
End if
End if
End if
If CheckBox1. Click = **True, Then**
Compute the fitness of the next generation
If Inputbox.Show =**True, Then**
Input the number of generations equations 1 and 2
End if
End if
If Button6.Click = **True, Then**
If Form5.load = **True**, **Then**
End if
End if
End if
End if
End

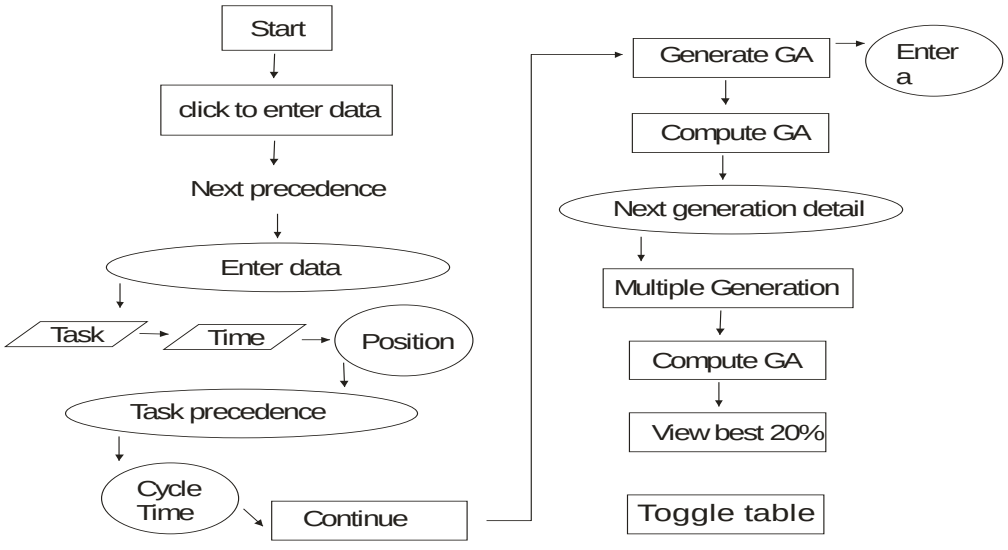


Figure 3:Flow chart of the computer program for evaluating assembly line balancing

2.1 Description of the Usage of the Software

The programme's structure basically consist of three stages namely the input stage, analysis, and output stage [12].The steps followed in using the program are explained using the example below. The example problem has 12 tasks and a cycle time of 10units. The precedence network of the presented example is graphically shown in Figure. 4 while the precedence table is shown in table 1.. A full detail of the problem is presented in [10]

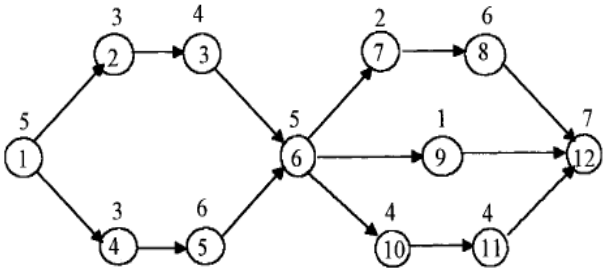
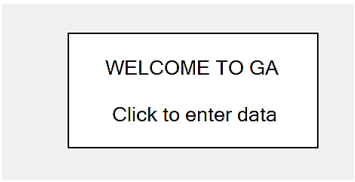


Figure 4: Precedence diagram of assembly network. [10].

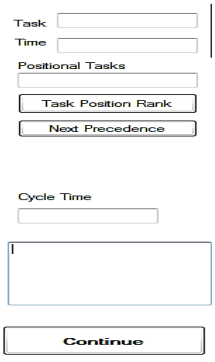
Table 1: The precedence table Cycle time (CT) = 10.

Task No.	Task Time	Immediate Predecessor task
1	5	-
2	3	1
3	4	2
4	3	1
5	6	4
6	5	3,5
7	2	6
8	6	7
9	1	6
10	4	6
11	4	10
12	7	8,9, 11

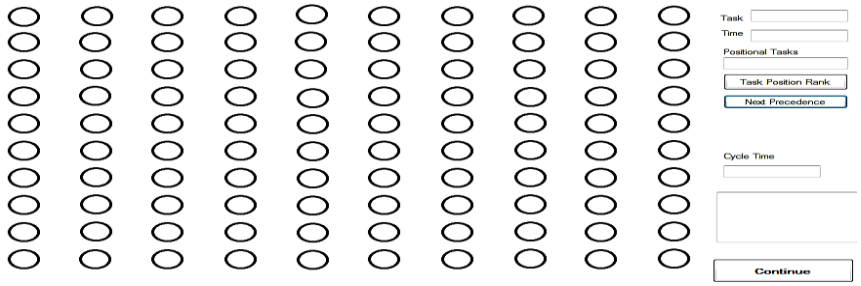
Step1. Launch the program. This will open the welcome window which has the enter button.



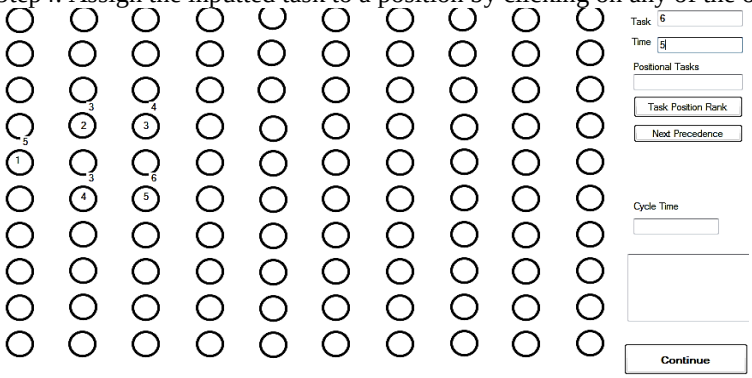
Step2. Click the enter data button to go the data input widow where the tasks, task time, task precedence and cycle time will be inputted.



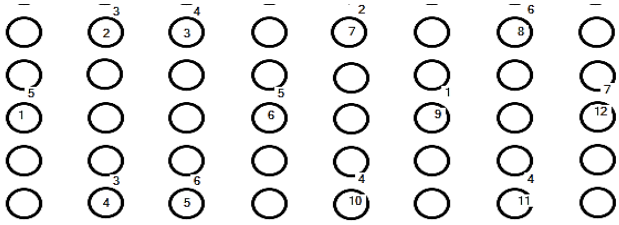
Step3. Click on next precedence and enter the task number and task time to labelled textbox.



Step4. Assign the inputted task to a position by clicking on any of the oval shape.

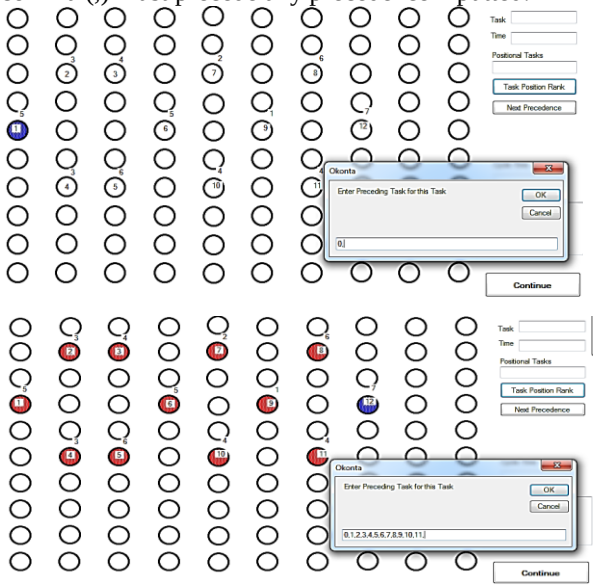


Step5. Repeat steps 3 and 4 for all the tasks.

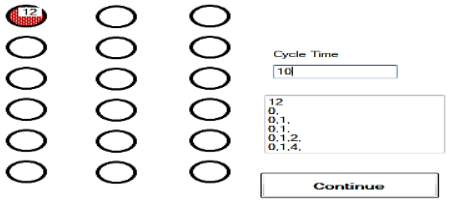


Step6. Click on enter precedence. The will pop up an input box and an oval shape will be highlighted.

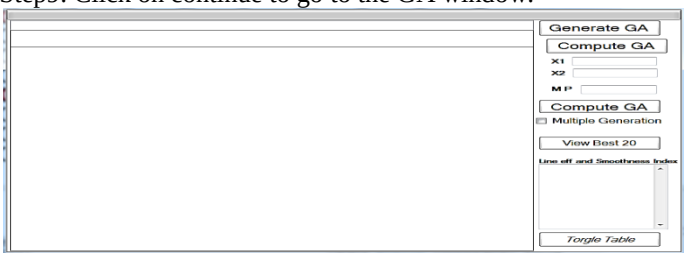
Step7. Enter the precedence to the task in the oval shape highlighted starting with 0, for the tasks with no precedence, 0,1, for the tasks that have task number 1 as precedence, 0,1,2, for tasks that have task number two as precedence, etc. NOTE a comma (,) must precede any precedence inputted.



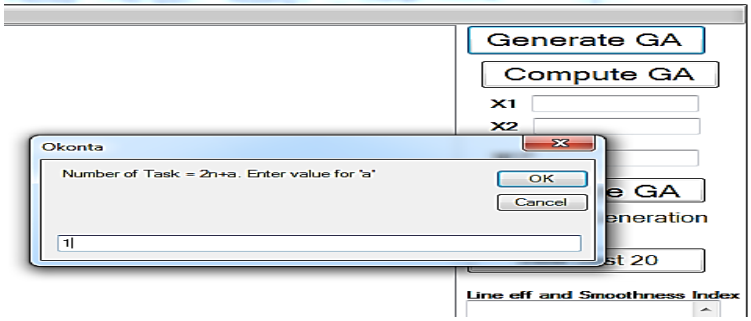
Step8. Enter the cycle time into the labelled textbox.



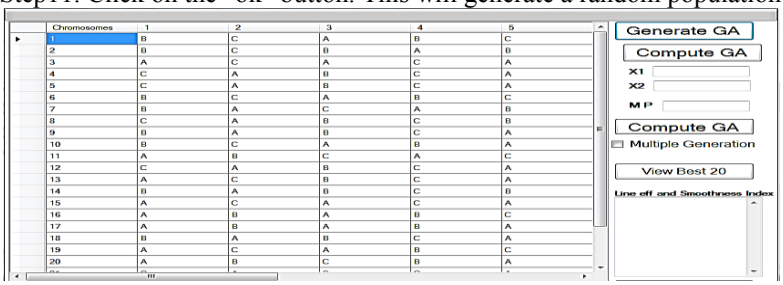
Step9. Click on continue to go to the GA window.



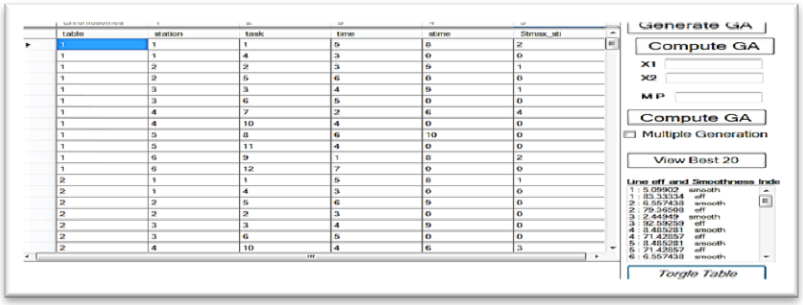
Step10. Click on generate GA. This will pop up an input box where the value of "a" will be inputted. "a" is the number of chromosome in excess of 2×n where "n" is the number of tasks.



Step11. Click on the "ok" button. This will generate a random population of chromosomes



Step12. Click on compute GA. this will give the solution to the first generation in table with the line efficiency and smoothness index in a list box. Click on toggle table to view the solutions.

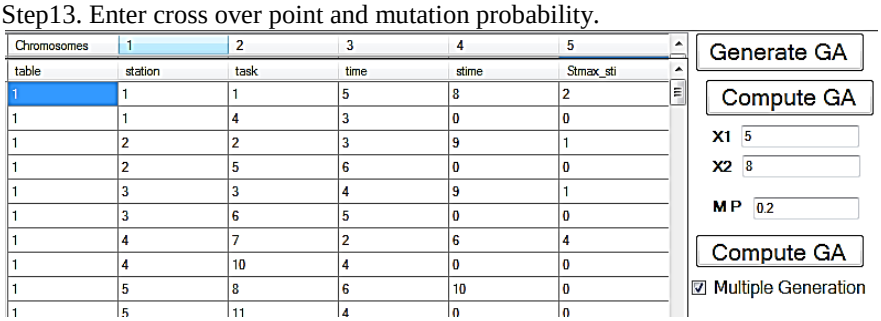


The summary of the solution in example 1 is shown in Table 2.

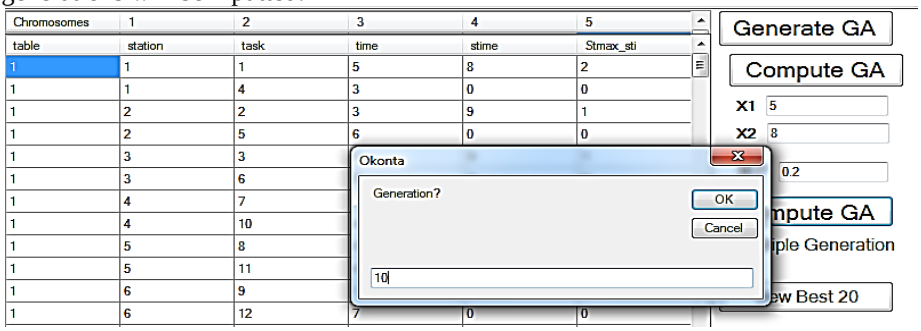
Table 2: Solution to the initial population of example 1

Chromosome	No of station	Manual Line efficiency	Program Line efficiency	Error	Manual Smoothness	Manual Smoothness	error
1	6	83.33333	83.33334	0.00%	2	5.09902	0.00%
2	7	79.36508	79.36508	0.00%	6.557438	6.557438	0.00%
3	6	83.33333	83.3333 4	0.00%	4.89898	4.89898	0.00%
4	6	83.33333	83.33334	0.00%	5.09902	5.09902	0.00%
5	7	71.42857	71.42857	0.00%	8.485281	8.485281	0.00%
6	6	92.59259	92.59259	0.00%	2.44949	2.44949	0.00%
7	7	79.36508	79.36508	0.00%	6.557438	6.557438	0.00%
8	6	83.33334	83.33334	0.00%	4.89898	4.89898	0.00%
9	7	71.42857	71.42857	0.00%	8.485281	8.485281	0.00%
10	7	79.36508	79.36508	0.00%	6.557438	6.557438	0.00%
11	7	79.36508	79.36508	0.00%	5.567764	5.567764	0.00%
12	6	83.33333	83.33334	0.00%	5.09902	5.09902	0.00%
13	6	92.59259	92.59259	0.00%	2.44949	2.44949	0.00%
14	7	79.36508	79.36508	0.00%	6.557438	6.557438	0.00%
15	6	83.33333	83.33334	0.00%	5.09902	5.09902	0.00%
16	6	83.33334	83.33334	0.00%	5.09902	5.09902	0.00%
17	7	71.42857	71.42857	0.00%	8.485281	8.485281	0.00%
18	6	83.33334	83.33334	0.00%	4.89898	4.89898	0.00%
19	6	83.33334	83.33334	0.00%	5.09902	5.09902	0.00%
20	6	92.59259	92.59259	0.00%	2.44949	2.44949	0.00%
21	6	92.59259	92.59259	0.00%	2.44949	2.44949	0.00%
22	7	79.36508	79.36508	0.00%	5.91608	5.91608	0.00%
23	6	83.33334	83.33334	0.00%	4.89898	4.89898	0.00%
24	7	71.42857	71.42857	0.00%	8.485281	8.485281	0.00%
25	6	92.59259	92.59259	0.00%	2.44949	2.44949	0.00%

Step13. Enter cross over point and mutation probability.



Step14. Click on the checkbox and then the next generation button. This will pop up an input box where the number of generations will be inputted.



Step15. Click on view best 20%. This will give the best solutions to the problem.

Final Result

1	A	C	A	C	A
2	A	B	C	B	A
3	B	C	A	B	A
4	B	C	A	B	A
5	A	B	C	B	A

View Best 20

Line eff and Smoothness Index

1 : 2.44949 smooth
1 : 92.59259 eff
2 : 2.44949 smooth
2 : 92.59259 eff
3 : 2.44949 smooth
3 : 92.59259 eff
4 : 2.44949 smooth
4 : 92.59259 eff
5 : 2.44949 smooth
5 : 92.59259 eff
6 : 2.44949 smooth

Torgle Table

table	station	task	time	stime	Stmax_sti
1	1	1	5	8	1
1	1	4	3	0	0
1	2	2	3	9	0
1	2	5	6	0	0
1	3	3	4	9	0
1	3	6	5	0	0
1	4	7	2	8	1
1	4	8	6	0	0
1	5	10	4	9	0
1	5	11	4	0	0
1	5	9	1	0	0
1	6	12	7	7	2
2	1	1	5	8	1
2	1	4	3	0	0
2	2	2	3	9	0
2	2	5	6	0	0
2	3	3	4	9	0
2	3	6	5	0	0
2	4	7	2	8	1
2	4	8	6	0	0
2	5	10	4	9	0
2	5	11	4	0	0
2	5	9	1	0	0
2	6	12	7	7	2
3	1	1	5	8	1
3	1	4	3	0	0
3	2	2	3	9	0
3	2	5	6	0	0
3	3	3	4	9	0
3	3	6	5	0	0
3	4	7	2	8	1
3	4	8	6	0	0
3	5	10	4	9	0
3	5	11	4	0	0
3	5	9	1	0	0
3	6	12	7	7	2
4	1	1	5	8	1
4	1	4	3	0	0
4	2	2	3	9	0
4	2	5	6	0	0
4	3	3	4	9	0
4	3	6	5	0	0
4	4	7	2	8	1
4	4	8	6	0	0
4	5	10	4	9	0
4	5	11	4	0	0
4	5	9	1	0	0
4	6	12	7	7	2
5	1	1	5	8	1
5	1	4	3	0	0
5	2	2	3	9	0
5	2	5	6	0	0
5	3	3	4	9	0
5	3	6	5	0	0
5	4	7	2	8	1
5	4	8	6	0	0
5	5	10	4	9	0
5	5	11	4	0	0
5	5	9	1	0	0
5	6	12	7	7	2

The best 20% corresponds to the pareto optimal solutions to the assembly line balancing problem.

Example 2. **Source:** [11]. Precendence diagram is shown in figure 5 and the precedence table is shown in table 3

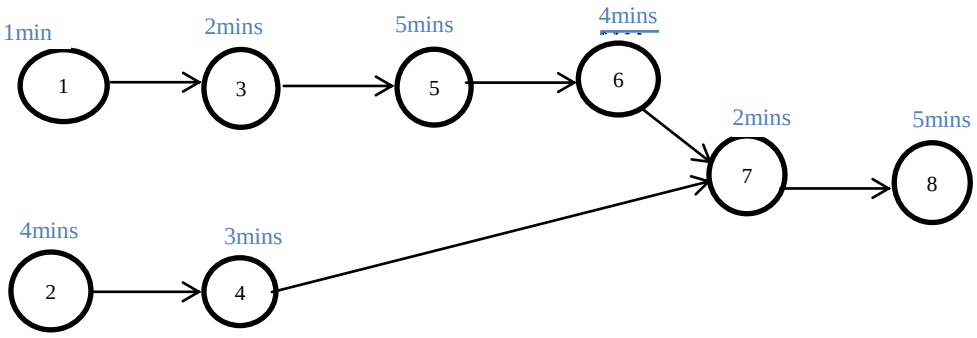
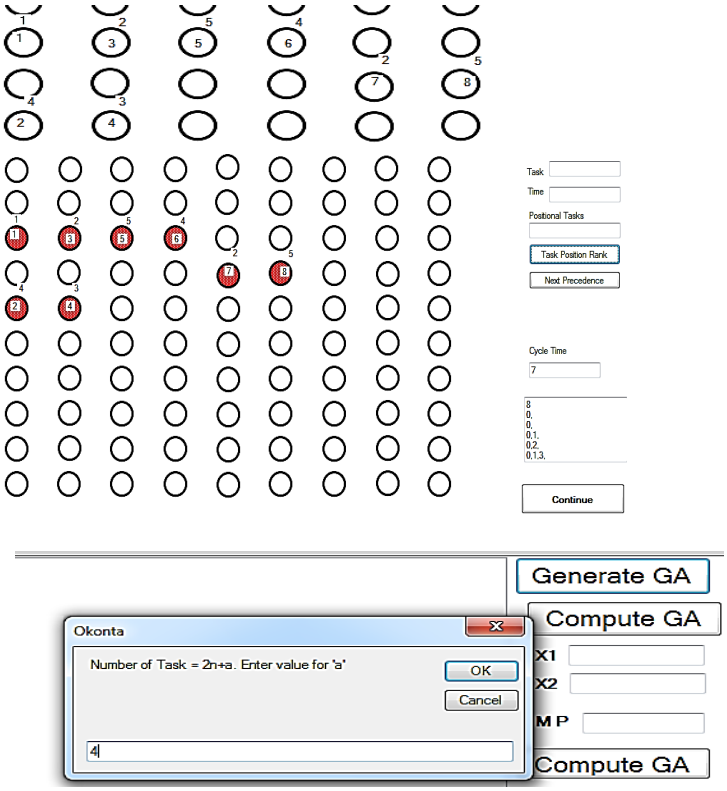


Figure 5:.Precedence Diagram

Table 3: The precedence table (CT) = 7 Minutes

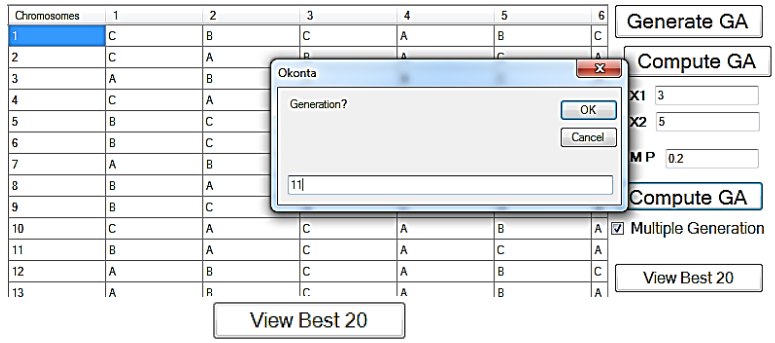
Task Number	Task Time (minutes)	Immediate Predecessor Task
1	1	-
2	4	-
3	2	1
4	3	2
5	5	1,3
6	4	1,3,5
7	2	1,2,3,4,5,6
8	5	1,2,3,4,5,6,7



Using the same staeps as in example 1, the following solution to the initial population as contained in table 1 was obtained.

Table 4: Solution to the initial population of example 2

chromosome	No of station	Manual Line efficiency	Program Line efficiency	error	Manual Smoothness index	Manual Smoothness index	error
1	5	86.66667	86.66666	0.00%	2	2	0.00%
2	4	92.85714	92.85714	0.00%	2	2	0.00%
3	4	92.85714	92.85714	0.00%	2	2	0.00%
4	5	86.66667	86.66666	0.00%	2	2	0.00%
5	5	74.28571	74.28571	0.00%	5	5	0.00%
6	5	86.66667	86.66666	0.00%	2	2	0.00%
7	5	86.66667	86.66666	0.00%	2	2	0.00%
8	5	74.28571	74.28571	0.00%	5	5	0.00%
9	5	86.66667	86.66666	0.00%	2	2	0.00%
10	5	86.66667	86.66666	0.00%	2	2	0.00%
11	4	92.85714	92.85714	0.00%	2	2	0.00%
12	4	92.85714	92.85714	0.00%	5.385165	5.385165	0.00%
13	5	86.66667	86.66666	0.00%	2	2	0.00%
14	5	86.66667	86.66667	0.00%	2	2	0.00%
15	5	74.28571	74.28571	0.00%	5	5	0.00%
16	4	92.85714	92.85714	0.00%	2	2	0.00%
17	4	92.85714	92.85714	0.00%	2	2	0.00%
18	4	92.85714	92.85714	0.00%	2	2	0.00%
19	5	86.66667	86.66667	0.00%	2	2	0.00%
20	5	86.66667	86.66667	0.00%	2	2	0.00%



1	B	A	C	B	C
2	B	C	A	B	C
3	B	C	A	B	C
4	B	C	A	C	A

table	station	task	time	stime	Stmax_sti
1	1	2	4	7	0
1	1	1	1	0	0
1	1	3	2	0	0
1	2	5	5	5	2
1	3	6	4	7	0
1	3	4	3	0	0
1	4	7	2	7	0
1	4	8	5	0	0
2	1	2	4	7	0
2	1	1	1	0	0
2	1	3	2	0	0
2	2	5	5	5	2
2	3	6	4	7	0
2	3	4	3	0	0
2	4	7	2	7	0
2	4	8	5	0	0
3	1	2	4	7	0
3	1	1	1	0	0
3	1	3	2	0	0
3	2	5	5	5	2
3	3	4	3	7	0
3	3	6	4	0	0
3	4	7	2	7	0
3	4	8	5	0	0
4	1	2	4	7	0
4	1	1	1	0	0
4	1	3	2	0	0
4	2	5	5	5	2
4	3	6	4	7	0
4	3	4	3	0	0
4	4	7	2	7	0
4	4	8	5	0	0

3.0 System Requirement and Operating System

For the assembly line balancing software to work perfectly, the computer must fulfil the following requirements:
Computer as from Pentium, at least 266MHz
-Windows XP (32 or 64 Bit) or
- Windows Vista (32 or 64 Bit) or
- Windows 7 (32 or 64 Bit) or
- Windows 8 (32 or 64 Bit)

4.0 Discussion

This software was tested for this paper on several assembly line balancing problems. Best 20% of the possible optimal solutions were displayed in table form with their corresponding line efficiency and smoothness index in the associated listbox. The software performance accuracy was tested rigorously through large number of different problems. The computational speeds for the various problems depend on the number of tasks and the number of successive generations. With the concept of the realized cycle time and parent selection based on fitness ranking, the GA undergoes less iteration to obtain optimum solutions for Assembly Line Balancing Problems.

5.0 Conclusion

Genetic Algorithm provides a comprehensive search methodology for optimization. When the search space is very large then genetic algorithm methods generally take a long time to converge to good quality solutions. In order to obtain optimum solutions within a reasonable time of its implementation, we have designed computer software for the evaluation of assembly line balancing problem using GA. The software performs accurately and efficiently.

References

[1] Pekin N (2006); M.Sc. thesis, Middle East Technical University: Multi Criteria Assembly Line Balancing Problem With Equipment Decisions

[2] Shtub, A. and E.M. Dar-El,(1989). A methodology for the Selection of Assembly Systems. International. Journal of. Production. Research., 27: 175-186. DOI: 10.1080/00207548908942537

[3] Suwannarongsri S., Limnararat S. and Puangdownreong, D (2007) Hybrid Tabu Search Methodfor Assembly Line Balancing, Proceedings of the 7th WSEAS International Conference on Simulation, Modelling and Optimization, Beijing, China. pp. 443-448.

[4] Rashid, M.F.F, Hutabarat, W and Ashutosh, T (2011). A Review on Assembly Sequence Planning and Assembly Line Balancing Optimisation using SoftComputing Approaches. International Journal of Advanced Manufacturing Technology, Volume 59, Issue 1-4, pp335-349

[5] Ariffin, M.K.A.M, Fathi, M and Ismail, N (2012). A New Heuristic Method to Solve Straight Assembly Line Balancing Problem. Pertanika Journal of Science and Technology. 20 (2) 355 – 369.

[6] Kriengkorakot N and Pianthong N (2007) The Assembly Line Balancing Problem: Review articles. KKU Engineering Journal, 34(2) 133 - 140

[7] Bajpai, P and Kumar, M (2008). Genetic Algorithm – an Approach to Solve Global Optimization Problems. Indian Journal of Computer Science and Engineering Vol 1 No 3 199-206)

[8] Guo, C and Yang, X (2011). A Programming of Genetic Algorithm in Matlab7.0 Modern Applied Science Vol. 5, No. 1;pp. 230-235

[9] Muhammad Z. M and Muhammad I. J (2010). Soft Computing in Optimizing Assembly Lines Balancing. Journal of Computer Science 6 (2): 141-162

[10] Edokpia R. O and Okonta C. I. (2013). On The use of heuristics and genetic algorithm for solving linebalancing problems. a comparative analysis. Journal of the Nigerian Association of Mathematical PhysicsVolume 25,pp267-280

[11] Edokpia R. O and Okonta C. I. (2013) Solving Assembly Line Balancing Problems:A case study of a manufacturing company**.** Journal of the Nigerian Association of Mathematical Physics Volume 25,pp. 251-266.

[12] Akpobi, J.A. and Lawani A.I. (2006). Computer-Aided-Design of flywheels. Advances in Engineering Software 37. 222–235

[13] Ponnambalam S. G, Aravindan P, Mogileeswar Naidu G (2000) Multi-objective genetic algorithm for solving assembly line balancing International Journal Advance Manufacturing 16(5), pp. 341–352