APPLYING GENETIC ALGORITHMS TO THE U-SHAPED ASSEMBLY LINE BALANCING PROBLEM

Debora A. Ajenblit
Roger L. Wainwright

Mathematical and Computer Sciences Department
The University of Tulsa
600 South College Avenue
Tulsa, OK 74104-3189, USA
rogerw@penguin.mcs.utulsa.edu

ABSTRACT

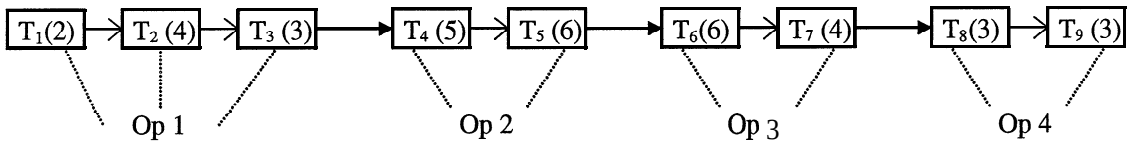
The traditional assembly line balancing problem considers the manufacturing process of a product where production is specified in terms of a sequence of tasks that need to be assigned to workstations. Each task takes a known number of time units to complete. Also, precedence constraints exist among tasks: each task can be assigned to a station only after all its predecessors have been assigned to stations. The U-shaped assembly line balancing problem is a relatively new problem derived from the traditional assembly line balancing problem. In the U-shaped assembly line balancing problem tasks can be assigned to stations either after all its predecessors or all of its successors have been assigned to stations. This paper presents a genetic algorithm (GA) solution to the Type I U-shaped assembly line balancing problem. Our research provides a global framework which can be used to deal with the two possible variations of this problem, minimizing total idle time, and balance of the workload among stations, or a combination of both. We developed six different assignment algorithms as a means for interpreting a chromosome and assigning tasks to workstations. The results show the GA to be an excellent technique for this problem. In the 61 standard test cases from the literature, our GA obtained the same results as previous researchers in 49 cases, superior results in 11 cases, and in only one case did worse. Moreover, the GA proved to be computationally efficient.

1. Introduction

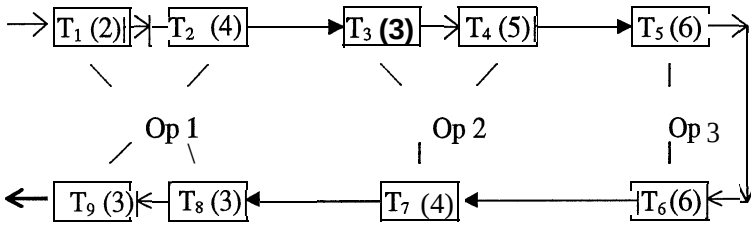
The traditional assembly line balancing problem considers the manufacturing process of a single product as a sequence of tasks which need to be assigned to different workstations. The distribution of tasks among the workstations is based on the required time units to complete each task as well as the precedence constraints that exist among these tasks [2]. Historically, the traditional assembly line balancing problem scenario leads to two types of optimization problems, Type I and Type II [2,10,12]. In the Type I problem, the cycle-time (maximum amount of time units that can be spent at each workstation) is fixed and the objective is to minimize the required number of workstations. The Type II problem attempts to minimize the maximum cycle-time given a fixed number of workstations.

In 1994, Miltenburg and Wijngaard [12] presented a new problem derived from the traditional assembly line balancing problem where production lines are arranged as U-shaped lines instead of straight lines. The U-shaped assembly line is an attractive alternative for assembly production systems since operators (workers) become multi-skilled by performing tasks located on different parts of the assembly line. Moreover, since the U-shaped line disposition allows for more possibilities on how to assign tasks to workstations, the number of stations needed for a U-shaped line layout is *never* more than the number of stations needed for the traditional straight line [12]. The reason for this is that in the traditional assembly line balancing problem, for a given workstation, the set of possible assignable tasks is conformed by those tasks whose *predecessors* have already been assigned to workstations, whereas in the U-shaped line problem the set of assignable tasks is determined by all those tasks whose *predecessors* or *successors* have already been assigned.

0-7803-4869-9/98 $10.0001998 IEEE



(a) Traditional Linear Assembly Line



(b) U-shaped Assembly Line

Figure 1. Example Traditional Assembly Line and U-shaped Assembly Line where the cycle time is 12, task time is shown in parenthesis and tasks are ordered 1 to 9.

Consider Figure 1 where each workstation has a cycle time of 12 and the predecessors of the tasks are such that task 1 must precede task 2, which must precede task 3, etc. to task 9. The time units for each task are given in parentheses. Figure l(a) shows a schedule for the 9 tasks for a traditional (linear) assembly line. Note four stations (or four operators) are required each with efficiency of 9/12,11/12,10/12 and 6/12, respectively. In Figure l(b) the same tasks are scheduled using a U-shaped assembly line where three workstations (operators) are required each with 100% efficiency. Notice in Figure l(b) a higher level of operator skill is required.

It is also possible to consider both Type I and Type II problems when dealing with the U-shaped assembly line problem. In this paper we limit our research to the Type I U-shaped assembly line problem, i.e., minimizing the number of stations given a fixed cycle-time. Note within this context two factors can be considered for the distribution of tasks to stations [10]: (a) total idle time, and (b) work balance among the workstations. Taking these two factors into account, the Type I U-Shaped assembly line balancing problem developed by Miltenburg and Wijngaard[12] can be rewritten and formally stated as follows:

Given
- asetofntasks, $T = \{T_1, T_2, \ldots . T_n\}$
- a function $t: T \rightarrow N$, where $t(T_k)$ is the necessary time units to complete T_k
- a fixed cycle time C,
- a binary precedence relation on elements of T

$P = \{ (T_x, T_y) / T_x$ *must be done before* $T_y \}$,

the objective is to find Π that satisfies:

(1) $\Pi = [S_1, S_2, \ldots S_W]$, a partition of T.

(2) $\sum_{T_k \in S_j} t(T_k) \leq C$ for all $S_j \in \Pi$

(3) For each $T_y \in T$:
for all T_x such that $(T_x, T_y) \in P$, $T_x \in S_i, T_y \in S_j$, then $i \leq j$ or
for all z such that $(T_y, T_z) \in P$, $T_y \in S_j, T_z \in S_k$, then $k \leq j$

a function f is minimized.

Condition (1) requires Π to be a partition of set T. This ensures that all tasks will be assigned to a station and a given task will not be assigned to more than one station. At each workstation, the total time units to complete all tasks assigned to it will not exceed cycle-time C; this is guaranteed by Condition (2). Precedence constraints are expressed by Condition (3): a task is assigned to a workstation either after all its predecessors have been assigned or before its predecessors have been assigned.

Finally, a function f is defined to evaluate the goodness of an assignment of tasks to workstations. Based on the above factors, we considered three possible definitions for f specified as f_1, f_2, *and* f_3 specified below.

$$f_1 = WC - \sum_{i=1}^{W} \sum_{T_k \in S_i} t(T_k)$$

$$f_2 = \sum_{i=1}^{W} \left(\sum_{T_k \in S_i} t(T_k) - C \right)^2 / W$$

$$f_3 = af_1 + b\sqrt{f_2}$$

where *f1* minimizes total idle time, *f2* minimizes mean-squared idle time, and *f3* is a combination of *f1* and *f2* where ***a*** and ***b are*** arbitrary constants each between 0 and 1, and $a + b = 1$.

2. Previous Work

The traditional (linear) assembly line balancing problem is known to be NP-hard [9]. If there are ***m*** tasks and *r* ordering constraints then there are $m!/2^r$ possible tasks sequences [2]. With such a vast search space it is nearly impossible to obtain an efficient solution using a deterministic algorithm. However, many attempts have been made [3,7,8,11,15,16]. None of these methods have proven to be of practical use for large problems due to their computational inefficiency.

Several heuristic-based methods for the traditional assembly line balancing have been developed. Baybars [1] and Talbot et ***al.*** [14] review and evaluate these different approaches. Another proposed solution considers a five-phase method using task elimination, decomposition and heuristics [2]. Also, a genetic algorithm was used to obtain near optimal solutions to the traditional assembly line balancing problem in combination with heuristic-based methods [10].

Miltenburg and Wijngaard adapted heuristics developed for the traditional assembly line and applied it to the U-shaped line balancing problem [12]. They tested their algorithm using previously studied examples and test cases found throughout the assembly line literature. They also devised a dynamic programming procedure. However, because of computational costs of the dynamic programming technique results were reported for small datasets only (those consisting up to 11 tasks).

3. A GA Solution for the U-Shaped Assembly Line Balancing Problem

A genetic algorithm operates on a collection of candidate solutions. Each solution is encoded as a finite length string, typically a string of bits or integers. The string is called a chromosome, and each element in the string is called a gene. Each chromosome represents a possible solution to the problem. A fitness function is applied to each chromosome in a population of chromosomes to determine how fit each chromosome is. Chromosomes are probabilistically selected based on their fitness. Genetic recombination operators take the selected chromosomes, exchange genetic material and produce children chromosomes. Davis [5], Goldberg [6], and Mitchel[13] provide an excellent in-depth study of the fundamentals of genetic algorithms. It is assumed the reader is familiar with the fundamental concepts of genetic algorithms.

Our GA implementation for the U-shaped assembly line balancing problem is ***order-based,*** i.e., a chromosome of length ***n*** is a permutation of $1,2,\ldots,n$. For a problem consisting of ***n*** tasks, each chromosome of length ***n*** represents a specific task ordering. Notice that in the U-shaped assembly line balancing problem, not every permutation will constitute a valid solution. Only those permutations in which all predecessors of a task precede that task in the permutation are considered feasible chromosomes. We dealt with only feasible chromosomes. Thus, it was necessary for us to generate only valid chromosomes in our initial population. Each gene of a chromosome in the initial population was obtained by randomly choosing the next task among unselected tasks whose predecessors had already been chosen. We also used a feasibility preserving crossover operator that ensures feasible chromosomes.

We used the crossover operator developed by Leu ***et al. in*** their GA solution for the traditional assembly line balancing problem [10]. We call this the "ordered two-point crossover". Given two parent chromosomes, two random crossover points are selected partitioning them into a left, middle and right portion. The ordered two-point crossover behaves in the following way: child 1 inherits its left and right section from parent 1, and its middle section is determined by the genes in the middle section of parent 1 in the order in which the values appear in parent 2. A similar process is applied to determine child 2.

Parent 1:	**4**		**2**	**1**	**1**		**3**	**1**	**6**		**5**
Parent 2:	**2**		**3**	**1**	**1**		**4**	**1**	**5**		**6**
Child 1:	4		2	1	3		1	1	6		5
Child 2:	**2**		**3**	**1**	**4**		1	**|**	5		**6**

Figure 2. **Example of Ordered Two-Point Crossover where crossover points are indicated by | symbol.**

Given two chromosomes than encode feasible solutions, then the ordered two-point crossover operator yields two new feasible chromosomes. Consider Figure 2. It is obvious that there are no violations of precedence constraints in either the left or right portion of Child 1, since they are exact copies of the left and right portions of Parent 1, which is known to be feasible. The placement of 1 and 3 in the middle portion of Child 1 does not violate any precedence constraint since they are located after 4 and 2 and before 6 and 5 in Parent 1. Finally, positioning task 3 before task1 does not violate any precedence constraint since task 3 preceded task1 in Parent 2 which is also assumed to be a feasible solution.

We did not use mutation in our GA implementation. The traditional mutation operators for order-based GAs are not appropriate for this problem since mutating a feasible chromosome can easily result in an infeasible chromosome. In order to use mutation, a new mutation operator would have had to be developed. Some alternatives were considered, including the one used by Leu et ***al.*** [10]. This operator did not enhance any of our solutions since it replicated work done by our initialization algorithm. Note that unlike Leu's GA implementation for the traditional assembly line problem [10], our GA implementation for the U-shaped assembly line problem starts with a randomly generated, initial population of 100 feasible solutions. This population size appeared to contain sufficient "genetic material" to allow the GA to evolve to a solution. No improvements were observed by increasing population size.

In order to compute the fitness value of a chromosome, it is necessary to determine how a particular order of tasks can be assigned to workstations. Given a chromosome depicting an ordering of the tasks, we developed as part of this research six different algorithms to assign tasks to stations. We denote this six assignment algorithms as assignment algorithms (a) through (f). The fitness value of a chromosome is determined by applying all six assignment algorithms (a)-(f) to it, and using the lowest fitness value. Assignment algorithm (a) attempts to always assign alternatively unassigned tasks at the beginning and the end of the sequence of tasks; assignment (b) does the same as (a) but giving priority to tasks located at the end of the sequence. In assignment (c), all possible unassigned tasks located at the beginning of the sequence are assigned and then all possible unassigned tasks at the end of the sequence are assigned. Similarly, assignment (d) attempts to assign all possible tasks at the end of the sequence and then to assign all possible tasks at the beginning of the sequence. Assignments (e) and (f) give alternate priority to the task at the beginning and end of the sequence, changing this priority at each iteration. Initially assignment (e) gives priority to the task at the beginning of the sequence, while assignment (f) does so to the task at the end of the sequence. Although these differences might seem trivial, they can lead to different assignments producing a different number of stations. Notice that all these variations in the assignment procedures respect the constraint that a task must be assigned to a station after all its predecessors or successors have been assigned to stations. It is possible that different assignment procedures end up with the same disposition of the assembly line, even though the order in which stations are determined is different.

4. Computational Results

Our GA solution was implemented using LibGA [4]. Our GA was applied to the 12 well known datasets from the literature for the traditional assembly line balancing problem. When one considers all possible cycle times used in these 12 datasets, this results in 61 different problems. The 61 problems include six problems from Merten, one problem from Bowman, five problems from Jaeschke, and so forth ending with six problems from Arcus. All 61 problems with results are listed in Table 1. Note Table 1 is depicted in two parts due to its size, but should be considered as one continuous table. Table 1 lists the problem source in the first column with the number of tasks listed below the problem source name. Next the various cycle times for each problem are given. The ideal (optimal) for each problem is given. Under the column, Others, solutions from previous researchers are given. For each of the 61 cases, N_{DP} represents the optimal results obtained using dynamic programming, and N_{Heur} represents the best known results to date using various heuristic algorithms as reported by Miltenburg and Wijngaard [12]. We tested each of these 61 problems using our three GAs whose fitness functions dealt with: total idle time (GA_1), balance of the workload among stations (GA_2), and both criteria with an equal weight, $\alpha = \beta = 0.5$, (GA_3). Note GA_1, GA_2, GA_3, correspond to the fitness functions f_1, f_2, f_3 described earlier. Our GA results are shown in Table 1. The number of stations obtained with each of the GAs is listed under columns GA_1, GA_2 and GA_3. Also for each of these results, a letter (a-f) follows representing which of the six assignments (a)-(f) that was used in the evaluation of the solution chromosome.

The dynamic programming technique produces optimal solutions. These results are shown (column

Table 1. Computational results comparing our genetic algorithms (GA1, GA2, GA3) to Others [12], which includes N_{DP}: (optimal results obtained with Dynamic Programming) and N_{Heur}: (best known results to date using heuristic algorithms).

Problem		Ideal	Others		GA_1	GA_2	GA_3
Merten	*Cycle*	*N*	N_{DP}	N_{Heur}	N	N	N
7	6	4.8	6	6	6a	6c	6c
	7	4.1	5	5	5a	5a	5a
	8	3.6	5	5	5a	5a	5a
	10	2.9	3	3	3a	3a	3a
	15	1.9	2	2	2b	2b	2a
	18	1.6	2	2	2a	2a	2a
Bowman							
9	20	3.7	4	5	***4d***	***4d***	***4d***
Jaeschke							
9	*6*	*6.2*	*8*	*8*	8a	8a	8a
	7	5.3	7	7	7a	7a	7a
	8	4.6	6	6	6e	6a	6a
	10	3.7	4	4	4a	4c	4a
	18	2.1	3	3	3a	3c	3c
Jackson							
11	7	6.6	7	8	***7a***	***7a***	***7a***
	9	*5.1*	*6*	*6*	6a	6c	6c
	10	4.6	5	5	5a	5a	5a
	13	3.5	4	4	4a	4c	4c
	14	3.3	4	4	4a	4a	4a
	21	2.2	3	3	3a	3d	3d
Dar-N							
11	48	3.8	4	4	4a	4d	4d
	62	3	3	4	***3b***	***3b***	***3b***
	94	2	2	3	***2d***	***2d***	***2d***
Mitchell							
21	*14*	*7.5*		*8*	8b	8c	8c
	15	7		8	8a	8b	8b
	21	5		6	***5a***	***5a***	5a
Heskiaoff							
28	*138*	*7.4*		*8*	8a	8d	8d
	205	5		6	6a	6c	6c
	216	4.7		5	5b	5e	5b
	256	4		4	4b	5d	5d
	324	3.2		4	4a	4c	4c
	342	3		3	3c	3a	3a

Problem		Ideal	Others			GA_1	GA_2	GA_3
Sawyer	*Cycle*	*N*	N_{DP}	N_{Heur}	N	N	N	N
30	25	13		15	***14d***	***14e***	***14e***	
	27	12		14	***13d***	***13d***	***13d***	
	30	10.8		12	12a	12d	12f	
	36	9		10	10b	10a	10b	
	41	7.9		9	***8c***	***8d***	***8c***	
	54	6		7	7a	7d	7c	
	75	4.3		5	5a	5f	5b	
Kilbridge&								
Webster	*57*	*9.7*		10	10c	10b	10d	
45	79	7		8	8a	8c	8c	
	92	6		7	7a	***6d***	7c	
	110	*5.1*		*6*	6a	6b	6c	
	138	4		5	4c	4c	4c	
	184	3		4	***3a***	***3a***	***3a***	
Tongue								
70	*176*	19.9		21	21f	21e	21e	
	364	9.6		10	10c	10e	10b	
	410	8.6		9	9b	9c	9c	
	468	7.5		8	8a	8a	8f	
	527	6.7		7	7a	7c	7c	
Arcus-83	5048	15		16	16b	16a	16a	
83	5853	12.9		14	14b	14e	14a	
	6842	11.1		12	12a	12f	12f	
	7571	10		11	11a	11c	11c	
	8412	9		10	10a	10d	10d	
	8898	8.5		9	9a	9c	9c	
	10816	7		8	8a	8c	8c	
Arcus - 111								
111	5755	26.1		27	28c	28d	28d	
	8847	17		18	18a	18d	18c	
	10027	15		16	16d	16d	16d	
	10743	14		15	15a	15d	15d	
	11378	13.2		14	14b	14d	14d	
	17067	8.8		9	9d	9a	9c	

N_{DP}) for the smaller sized problems *Merten* through *Dar-El* in Table 1 (first 21 test problems). In each of these problems all three of our GA algorithms also obtained the optimal solution. For the larger problems, *Mitchell* through *Arcus the* dynamic programming technique was too costly and was not attempted [12]. Next consider our GAs as a group versus N_{Heur}, which represents the best known results to date using various heuristic algorithms. In the 61 test problems our GA obtained the same result as the heuristic algorithms in 49 cases, obtained superior results in 11 cases, and did worse in only one case. GAs solutions that are better than those provided by the heuristic procedure are shown in bold and italics.

In Table 1 notice all of the assignment algorithms (a)-(f) were used as the algorithm that represented the best solution. In fact among the 11 test cases in which our GA outperformed all other heuristics, each of the assignment algorithms (a)-(f) was used at least once lending merit to each of our six assignment algorithms. Furthermore, the GAs proved to be competitive in terms of computational costs. For example, a solution to the largest of the datasets, (Arcus 111 tasks) was found in less than 10 minutes running on a Sun Sparcstation 10.

5. Conclusions

A GA solution for the Type I U-shaped assembly line balancing problem was developed. To the authors' knowledge this is the first time a genetic algorithm solution has been developed for the U-shaped problem. One of the main assets of this research is that it provides a global framework which can be used to deal with the two possible variations of this problem, minimizing total idle time, and balance of the workload among stations, or a combination of both. As part of this research we developed assignment algorithms (a) through (f) as a means for interpreting a chromosome and assigning tasks to workstations. In our results each of these assignment algorithms proved to have merit. The results show our GA to be an excellent technique for this problem. In the 61 standard test cases, our GA obtained the same results as previous researchers in 49 cases, superior results in 11 cases, and in only one case did worse. Moreover, the GA proved to be computationally efficient.

References

[1] I. Baybars. ***A Survey of Inexact Algorithms for the Simple Assembly Line Balancing Problem. GSIA WP-64-82-83,*** Carnegie-Mellon University, Pittsburgh, PA, July 1983 (revised October 1984).

[2] I. Baybars. ***An Efficient Heuristic Method for the Simple Assembly Line Balancing Problem.*** Int. Journal of Production Research, vol. 24, No. 1, 1986.

[3] J.M. Charlton, C.C. Death. ***A general method for machine scheduling.*** Int. Journal of Production Research No. 7, 1969.

[4] A.L. Corcoran,, R.L. Wainwright. ***Using LibGA to develop Genetic Algorithms for Solving Combinatorial Optimization Problems.*** Lance Chambers, Editor, Practical Handbook of Genetic Algorithms, Applications Volume I, pages 143-172, CRC Press, 1995.

[5] L. Davis, editor, ***Handbook of Genetic Algorithms,*** Van Nostrand Reinhold, 1991

[6] D. Goldberg. ***Genetic Algorithms in Search, Optimization, and Machine Learning,*** Addison Wesley, 1989.

[7] M. Held, R.M. Karp, R. Sharesian. ***Assembly line balancing-Dynamic programming with precedence constraint.*** Operations Research, No. 11, 1963.

[8] J.R. Jackson. ***A computing procedure for a line balancing problem.*** Management Science, No. 2, 1956.

[9] R.M. Karp. ***Reducibility among combinatorial problems.*** Complexity of Computer Applications, New York: Plenum, 1972.

[10] Y. Leu, L. Matheson, L. Rees. ***Assembly Line Balancing Using Genetic Algorithms with Heuristic-Generated Initial Populations and Multiple Evaluation Criteria.*** Decision Sciences, vol. 25, No. 4.

[11] P. Mertens. ***Assembly line balancing by partial enumeration.*** Ablauf- und Planung-forschung Vol. 8, 1967.

[12] G.J. Miltenburg, J. Wijngaard. ***The U-line Line Balancing Problem.*** Management Science, vol. 40, No. 10, October 1994.

[13] Melaline Mitchell, ***An Introduction to Genetic Algorithms, The MIT Press, 1996.***

[14] F.B. Talbot, J. H. Patterson. ***A comparative evaluation of heuristic line balancing techniques.*** GBS, University of Michigan, Ann Arbor, Michigan, Working Paper 125, 1981.

[15] F. Van Assche, W.S. Herroelen. ***An optimal procedure for the single-model deterministic assembly line balancing problem.*** European Journal of Operational Research, Vol. 3, 1979.

[16] T.S. Wee, M.J. Magazine. ***An efficient branch and bound algorithm for assembly line balancing- Part 1: minimize number of workstations.*** University of Waterloo, Waterloo, Ontario, Working Paper 151, 1981.