Conclusions from Comparing Genetic Algorithms for U-shaped Assembly Line Balancing

Alexander Meltser
Yuval Cohen
Mireille Avigal

Department of Mathematics and Computer Science
The Open University, Raanana, Israel

Abstract

This paper compares several different Genetic Algorithm approaches for solving the Mixed Model U-Line Balancing and Sequencing. We first overview the Genetic Algorithms approach and the Assembly Line Balancing Problem in general, then we describe the Mixed Model U-shaped Assembly Line and the problems it presents. We proceed to applications of Genetic Algorithms to these problems, describing and comparing the various algorithms proposed in recent years. Several different algorithms are implemented and the results of comparative executions on benchmark problems follow. The comparisons are done on various combinations of parameter values. In particular, we investigate the behavior of the algorithms under different levels of crossover intensity, mutation intensity and elitism. Finally, we draw conclusions and present potential future research directions.

Keywords
Genetic algorithms, Assembly line, Line balancing, Mixed model, U-line, U-shape

1. Introduction

The general field of assembly line design and balancing has been intensively addressed during the last few decades. The variety and complexity of assembly line balancing problems have motivated many researchers to challenge these problems. Even the very simple assembly line balancing problems are known to be NP-hard [1, 2] (widely believed to not have polynomial time algorithms). Thus, many of the proposed solutions are based on heuristics such as Genetic Algorithms and other search techniques (e.g., simulated annealing (SA), tabu search (TS), ant colony (AC), and particle swarm optimization (PSO)).

In the last decades, the necessity of making production more versatile and flexible, has forced assembly line production systems to change from single model assembly lines to mixed-model assembly lines. That is, several different product models are assembled on the same line. Moreover, in various environments (e.g., JIT environment) the traditional straight line was replaced by a U-shaped assembly system [3]. These two characteristics (the U-shape and the many models) contribute to the complexity of the task assignment to stations so that exact optimization is not practical.

The mixed model assembly line balancing problem (MMALBP) can be stated as follows: given M models, the set of tasks and a cycle times associated with each model, the task times and the set of precedence constraints, the problem is to assign the tasks of all models to workstations so that the precedence constraints are not violated and some performance measure is optimized [4]. For U-shaped assembly lines the workstations can either be fully located on one side of the U shape or be located on both sides of the U shape, in which case the product visits such crossover workstation twice while moving along the assembly line.

The mixed model U-shaped line balancing problem was first addressed by [5]. U-shaped assembly lines are generally considered to be more effective than simple assembly lines, mainly due to the potential of crossover stations, facilitated by this layout. Every solution feasible for straight line is always feasible for a U-shaped line as well, since a U-shaped line does not necessarily have to contain crossover stations and can be viewed as a generalization of the straight line assembly system. However the optimal solution for a U-line may be
more efficient than the optimal solution for a straight line due to the wider range of valid task assignments made possible by the crossover workstations.

In this paper we compare three different genetic algorithm techniques from recent years that address the assembly line balancing problem of type I, where the goal is to minimize the number of required workstations for a given cycle time (or to maximize line efficiency).

A genetic algorithm meta-heuristic is a solution search strategy inspired by the principles of the evolution theory, practically mimicking the natural survival of the fittest process. It is often used to find solutions to combinatorial optimization problems, and particularly problems for which deterministic algorithms are computationally infeasible. An iterative process based on natural selection is used, starting with an initial population of possible solutions to the problem at question, and this population evolves during a series of iterations. At every iteration the population is assessed, and based on their fitness (a measure of how good the solution is), some of the fittest individuals are stochastically chosen to produce the next generation. New solutions are obtained through applying stochastic transformations to existing ones by means of two genetic (recombination) operators: mutation and crossover, which are applied with predefined probabilities (rates). Mutation makes changes in a single individual to create a slightly different individual, while crossover combines parts from two individuals to create one or more new individuals. The idea is that the offspring inheriting the better "qualities" (the fittest genetic material) from their parent have better chances to survive the natural selection and are more likely to be chosen for producing the next generation. This process is repeated until a certain stopping condition is reached, by which time the population hopefully contains the best possible solutions.

2. Compared algorithms

We compared three methods based on genetic algorithm approach. These three algorithms were published in recent years (since 2010) and address mixed-model U-shaped assembly line balancing problem of type I.

2.1 ASGA: Amelioration structure with a genetic algorithm
This algorithm is proposed by [1, 2].
Two performance criteria are considered simultaneously: the number of workstations (can also be expressed as line efficiency) and the variation of workload.
The amelioration structure first uses a genetic algorithm to optimize the number of workstations and the variation of workload for an initial cycle time. The optimal number of workstations is then fixed and the algorithm proceeds to ameliorating the variation of workload by repetitively executing the genetic algorithm, while decreasing the cycle time before every execution, until no solutions can be found with the fixed workstation number.
The genetic algorithm utilizes a priority-based chromosome encoding method (PBC), a position-based “weight mapping crossover” operator (WMX) and a simple “swap mutation” operator. Roulette-wheel selection is employed.

2.2 GARKM: Genetic algorithm by Rabbani, Kazemi and Manavizadeh
This algorithm using a new approach independent of product sequences is proposed by [3].
It aims to minimize the number of crossover workstations in the balanced line and maximize the line efficiency, as well as minimize the variation of workloads between stations. The idea behind minimizing crossover workstations is to overcome the complexities added by U-shaped layout without undermining the advantages it provides. In order to achieve this goal, the method assigns a “penalty” for crossover stations and also takes into account the time required by the operator to move between crossover station sides. However, decrease in the number of crossover stations will not come at the cost of increasing total number of stations. Additional difference from majority of researches in this field is that task execution times of tasks common between models are not assumed to be identical for different models. The chromosome encoding utilizes a set of task assignment rules introduced by [6] – this representation increases the flexibility of the chromosomes by easing crossover and mutation operations so that no repairing of chromosomes is required after the operations. The sampling mechanism employed is roulette-wheel selection.

2.3 NTSGA: Novel two-stage genetic algorithm
This algorithm is proposed by [4] and utilizes the duplicate tasks concept, which allows tasks shared between multiple models to be assigned to different stations, whereas the common supposition always assigns such tasks to the same station for all models. Such duplicated tasks lead to increased costs, since the tools or machines required for these
tasks might have to be duplicated and fitted at the multiple stations handling these tasks. However this approach also allows higher flexibility in line balancing, which can lead to overall savings. This method has two stages, both of which employ a genetic algorithm. The first stage aims to minimize the number of workstations, treating each model independently, i.e. executing genetic algorithm separately for each model. The second (and main) stage uses the results of the first stage as the initial population for an additional genetic algorithm execution, which aims to minimize the task duplication costs (that is proportional to the number of task duplicates). The chromosome encoding used is the same as for GARKM above. For selection mechanism, two well-known methods are considered: roulette-wheel and tournament selection.

3. Comparison process

We have implemented the three abovementioned algorithms in MATLAB and used the following four datasets:

- Thomopoulos 19-task problem [7],
- Kazemi 22-task problem [4],
- Hwang 49-task problem [1, 2] and
- Kim 61-task problem [8]


Additional data required was either obtained from the original articles [1-4] or generated arbitrarily:

- for ASGA: cycle time decrements
- for GARKM: crossover workstations coefficient (penalty) and operator's travel time
- for NTSGA: station cost and task duplication costs

In order to make the algorithms as similar as possible in terms of their input and output, we used the same cycle times for all models when running NTSGA (since ASGA and GARKM only support a single cycle time for all models) and calculated the workload variance for NTSGA (which does not originally use this measure). We introduced a new “bottleneck cycle time” measurement for all three algorithms. We also slightly modified ASGA by making the implementation employ the same elitism approach as in GARKM and NTSGA.

We performed numerous computational experiments to determine the optimal values for different genetic algorithms parameters. In particular we were interested in the values of the following parameters, that lead to best overall results across all three algorithms and four datasets:

- elitism rate,
- selection method (roulette-wheel vs. tournament selection),
- crossover rate,
- mutation rate,
- population size and
- termination condition

To do so, we executed the algorithms on all datasets, testing different values of one parameter at a time, keeping the rest of the parameters constant between executions.

The parameter values found to be optimal are: elitism rate of 10%, roulette-wheel selection, crossover rate of 0.8, mutation rate of 0.3, population size of 100 individuals, and termination condition of 100 generations with no fitness improvement. In addition, although this has been shown many times before, we also verified that the U-shaped layout (usually) yields better results than the simple straight line layout.

Once the optimal parameter values were found, we could compare the algorithms on equal terms, which still allow them to perform well. For lack of space, we only present here the results of this final comparison.

4. Comparison results

Comparing the results we focused on the following metrics: number of workstations, variance of workload, bottleneck cycle time, fitness value of the solutions and CPU requirements.
The experiments were performed using MATLAB v7.5.0 (R2007b) on a laptop with Intel® Core™ i5 CPU @ 2.60 GHz and 8GB of DDR3-SDRAM (1333MHz) memory. The results of the comparisons are summarized in Table 1 below. Each row in this table represents an average of 10 repetitions of an algorithm with the abovementioned optimal parameters on the specified dataset.

Table 1: A summary of the comparisons

<table>
<thead>
<tr>
<th>problem</th>
<th>algorithm</th>
<th>workstations number</th>
<th>workload variance</th>
<th>bottleneck cycle time</th>
<th>fitness*</th>
<th>CPU (sec.)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Thomopoulos-19</td>
<td>ASGA</td>
<td>4</td>
<td>0.035</td>
<td>1.86</td>
<td>1.754</td>
<td>765.82</td>
</tr>
<tr>
<td></td>
<td>GARKM</td>
<td>4</td>
<td>0.044</td>
<td>1.90</td>
<td>1.703</td>
<td>233.63</td>
</tr>
<tr>
<td></td>
<td>NTSGA</td>
<td>4</td>
<td>0.132</td>
<td>2.00</td>
<td>159.5</td>
<td>1209.62</td>
</tr>
<tr>
<td>Kazemi-22</td>
<td>ASGA</td>
<td>9</td>
<td>0.183</td>
<td>18.10</td>
<td>1.602</td>
<td>956.31</td>
</tr>
<tr>
<td></td>
<td>GARKM</td>
<td>9</td>
<td>0.176</td>
<td>18.20</td>
<td>1.544</td>
<td>457.77</td>
</tr>
<tr>
<td></td>
<td>NTSGA</td>
<td>8</td>
<td>0.146</td>
<td>18.90</td>
<td>252.9</td>
<td>2188.45</td>
</tr>
<tr>
<td>Hwang-49</td>
<td>ASGA</td>
<td>14</td>
<td>0.194</td>
<td>94.10</td>
<td>1.499</td>
<td>8834.73</td>
</tr>
<tr>
<td></td>
<td>GARKM</td>
<td>14</td>
<td>0.158</td>
<td>97.50</td>
<td>1.510</td>
<td>1183.49</td>
</tr>
<tr>
<td></td>
<td>NTSGA</td>
<td>13</td>
<td>0.118</td>
<td>100.00</td>
<td>443.8</td>
<td>3829.34</td>
</tr>
<tr>
<td>Kim-61</td>
<td>ASGA</td>
<td>7</td>
<td>0.052</td>
<td>14.94</td>
<td>1.785</td>
<td>3356.74</td>
</tr>
<tr>
<td></td>
<td>GARKM</td>
<td>8</td>
<td>0.080</td>
<td>14.48</td>
<td>1.688</td>
<td>1313.57</td>
</tr>
<tr>
<td></td>
<td>NTSGA</td>
<td>7</td>
<td>0.101</td>
<td>14.99</td>
<td>490.7</td>
<td>9559.71</td>
</tr>
</tbody>
</table>

* fitness is calculated differently in all three algorithms and the values for one algorithm should not be directly compared to values for another.
For each dataset the best values of each measure are marked by grey background.

As can be seen from Table 1, it's hard to proclaim a “winner”. Each algorithm has its own strengths. ASGA’s strength seems to be the bottleneck cycle time, while GARKM is the fastest (which is easy since it has the simplest structure – no amelioration or two-stage involved) and NTSGA's solutions typically require the lowest number of workstations.

5. Conclusions and future research

The results produced by the different algorithms cannot be directly compared one to one in terms of their main measure, the fitness value, as the algorithms aim at slightly different problem settings, require slightly different input and most importantly have different optimization criteria and thus the quality of the solutions cannot be measured on a comparable scale.

However we can compare additional important measures of the solutions found, such as number of required workstations, workload variance, the bottleneck cycle time and the required CPU time.
We can clearly see that NTSGA finds solutions requiring the lowest number of workstations on average, but it requires the longest CPU time to complete. ASGA often finds solutions leading to lowest bottleneck cycle time and typically completes significantly faster than NTSGA. GARKM has the simplest structure of the three algorithms and is the fastest to complete, but its solutions are usually not as good as the solutions found by the other two algorithms.
As one can expect, there is a tradeoff between solution quality and computation time.

Another aspect of our findings has to do with the optimal values of different genetic algorithm parameters. As has been shown before, the U shape is nearly always preferable to the straight layout. Using roulette-wheel as the selection method is generally more beneficial than tournament selection. Applying some elitism always leads to better results than without elitism at all. The preferred elitism rate and the preferred crossover and mutation rates are slightly different between the algorithms: ASGA gives better results with slightly higher values of all three rates than the averages presented above, while NTSGA “prefers” slightly lower rates.
Each algorithm has its strengths and can produce better results depending on the specific priorities of the designed assembly line. All three algorithms surveyed show better results and/or performance than previous heuristics and meta-heuristics, to which the authors compared their algorithms. All in all, genetic algorithms are a very promising approach for solving the assembly line balancing problem.

Interesting directions for future research include:

- further investigation of the influence the multifarious GA characteristics and parameters have on the quality of produced results,
- implementing and comparing more algorithms and
- looking for additional algorithm structures and combining some of them together, such as combining the two-stage approach of NTSGA with the amelioration structure of ASGA.

References