Research Article

A Heuristic-Mixed Genetic Algorithm for Type II Assembly Line Balancing with Multiple Workers in Workstations

Xiongwen Qian

School of Urban Transportation and Logistics, Shenzhen Technology University, Guangdong, China

Correspondence should be addressed to Xiongwen Qian; qianxiongwen@sztu.edu.cn

Received 6 December 2021; Accepted 2 February 2022; Published 27 March 2022

1. Introduction

The Assembly line balancing problem (ALBP) is a well-known optimization problem in manufacturing engineering. Production organized by assembly line is first divided into elementary tasks, which are then assigned to a sequence of workstations connected by a material handling device, e.g., a conveyor. A product (workpiece) starts its assembly from the first workstation and goes through workstations one by one before its manufacturing completes in the last workstation. The duration (cycle time) of each workpiece staying in each workstation is equal. ALBP aims to assign indivisible tasks to workstations so as to optimize certain assembly line performance metrics, such as cycle time or the total number of workstations, with the constraints of task precedence relationship and workstation capacity. As a variant of the classic bin packing problem, ALBP is NP-hard as well [1].

The classic ALBP assumes that each workstation is operated by only one worker; the capacity of one workstation does not depend on the number of workers but on the automatic equipment. However, in manual or semimanual manufacturing which produces small-sized products with no special requirement on equipment, multiple workers are possible to work in one workstation, performing the same set of tasks on workpieces in parallel. Workers can be seated along the conveyor, taking workpieces off the conveyor and working on them; after the workpieces get processed, they will be put back on the conveyor and transferred to the next workstation. Compared to single-manned workstations, multimanned stations help to achieve higher utilization (less workstation idle time), facilitate a higher production rate (shorter cycle time), and improve the reliability of the entire assembly line. For example, suppose one workstation is assigned a task with a duration of 90 seconds, but the required cycle time is 60 seconds. To meet the required cycle time, the workstation has to be assigned at least two workers, so each worker has in fact twice the cycle time to complete the task as shown in Figure 1.

However, the freedom of choosing the number of workers for workstations further complicates ALBP. The problem becomes twofold: not only assigning tasks but also allocating workers. The two subproblems are closely coupled, so they cannot be solved separately. We refer to the
assembly line balancing problem with multiple workers in workstations as mALBP. Following the traditional taxonomy of ALBP [3], mALBP is classified into two types in general: type I, which attempts to minimize the number of workers for a given cycle time, and type II, which tries to minimize the cycle time for a given number of workers. It is worth noticing that in either type, the number of workstations is not specified beforehand. In this paper, the type II mALBP (mALBP-II) is studied. For type I mALBP, readers may turn to [4–6] for reference.

Although ALBP has received intensive investigations (see reviews [3,7,8]), there is a paucity of literature dealing with mALBP-II. Basically, literature on type II simple assembly line balancing (SALB) can be found in [9–11] which does not take multiworker stations into consideration. In addition, some researchers address another more complicated problem, similar to mAPBP, which is called the assembly line balancing problem with parallel workstations (pALBP). McMullen and Frazier [5] consider a multiobjective mALBP with parallel workers in work centers, and they propose a simulated annealing heuristic to solve the problem. But their problem setting is more like a type I problem. Öztürk et al. [12] studied pALBP to minimize cycle time, but with the assumption that the number of parallel stations assigned to each stage is given.

Another perspective on studying mALBP is the assembly line balancing problem with worker allocation (wALBP). Zhang et al. [13] developed a multiobjective genetic algorithm for wALBP, including one objective to minimize cycle time. However, in their model, the total number of workstations must be predetermined as a parameter. Zaman et al. [14] considered an operator assignment problem within line balancing. They assumed that tasks were already assigned to workstations. Zacharia and Nearchou [15] solved a bicriteria assembly line worker assignment and balancing problem. Two objectives, the cycle time and the smoothness of the line are optimized. A smoothness index is proposed as a measure of workload balance among stations. Most recently, Walter et al. [16] considered the line balancing problem to be smoothing workloads via branch-and-bound with dynamic programming. Their primary goal is to level the workload rather than minimize cycle time.

In this paper, we propose a heuristic-mixed genetic algorithm (hGA) for mALBP. The algorithm minimizes cycle time as its first objective and balances workload among workstations as its second objective. A greedy heuristic together with a bisection search is embedded into the decoding process of the genetic algorithm so that the optimization of task assignment and worker allocation are achieved simultaneously. The rest of the paper is organized as follows: Section 2 presents a mathematical formulation of mALBP; a detailed description of the proposed hGA is in Section 3; numerical results are demonstrated in Section 4, followed by a real case study in Section 5. Section 6 concludes the paper.

2. Mathematical Formulation

In this section, a nonlinear integer programming model is formulated for mALBP-II. Notations are summarized as follows.

2.1. Indices

(i) $i$, $g$, $h$: index of task ($i, g, h = 1, 2, \ldots, n$)
(ii) $j$: index of station ($j = 1, 2, \ldots, m$)
(iii) $k$: number of workers in one station ($k = 1, 2, \ldots, K$)

2.2. Parameters

(i) $n$: number of tasks
(ii) $m$: estimated upper bound of the number of stations
(iii) $K$: maximum number of workers allowed in one station
(iv) $W$: given total number of workers
(v) $t_i$: duration of task $i$
(vi) $P$: precedence matrix = \{(g, h) task $g$ is an immediate predecessor of task $h$\}

2.3. Decision Variables

(i) $C$: cycle time
(ii) $x_{ij} = \begin{cases}1, & \text{if task } i \text{ is assigned to station } j, \\0, & \text{otherwise.}\end{cases}$
(iii) $y_{jk} = \begin{cases}1, & \text{if station } j \text{ is assigned } k \text{ workers,} \\0, & \text{otherwise.}\end{cases}$

2.4. Mathematical Model

\[\min C,\]
\[\text{st} \sum_{j=1}^{m} \sum_{k=1}^{K} k y_{jk} = W,\]
\[\sum_{j=1}^{m} \left(j x_{gj} - j x_{hj}\right) \leq 0, \forall (g, h) \in P,\]
\[\sum_{j=1}^{m} x_{ij} = 1, \forall i,\]
\[\sum_{j=1}^{m} t_i x_{ij} \leq \sum_{k=1}^{K} k y_{jk} C, \forall j,\]
The objective function (1) minimizes cycle time. Constraint (2) states that the total number of used workers equals the number of workers available. Constraint set (3) makes sure that no precedence relationship among tasks is violated. The constraint set (4) ensures that each task is performed exactly once. The capacity constraint (5), which is nonlinear, ensures that total task durations assigned to one station do not exceed its capacity. The constraint set (6) means that if a station is opened, the number of workers at this station is unique.

3. Proposed Heuristic-mixed Genetic Algorithm for mALBP-II

Genetic algorithm (GA), as a kind of metaheuristics, has been proved its effectiveness in solving ALBP [17]. In this section, a heuristic-mixed genetic algorithm is proposed to solve mALBP-II. The algorithm embeds a greedy heuristic and a bisection search into the decoding of feasible-sequence-based chromosomes (individuals), so task assignment and worker allocation are determined simultaneously.

The overall structure of hGA is given in Algorithm 1.

3.1. Encoding. Encoding maps a solution space to a coding space. In this study, following [13], feasible task sequences are adopted as chromosomes in the coding space. A feasible task sequence means following which all task precedence relationships have complied. Regarding feasible sequences as chromosomes enables hGA to search efficiently in the coding space, since every chromosome is always kept feasible during the entire evolution of hGA; no infeasible solution is generated.

Encoding happens at the beginning of hGA to generate the initial population. A feasible task sequence is built by repeatedly choosing a task without any predecessor in the sequence and deleting the selected task from the precedence graph until all tasks have been put into the sequence. If more than one task has no predecessor, i.e., eligible to be chosen, we select one at random. Figure 2 shows an example of a precedence graph. Each node corresponds to a task, and the numbers above nodes are task durations. One example of a generated feasible task sequence is shown in Figure 3.

3.2. Decoding. Decoding maps chromosomes (feasible task sequences) back to task and worker assignment solutions. For mALBP-II, we need to find out the minimum cycle time that each chromosome corresponds to, which becomes a suboptimization problem in itself. In order to solve the problem, we developed a heuristic-based bisection search algorithm that starts from the lower bound and upper bound of the cycle time. Given a cycle time, a greedy heuristic, inspired by [18], is used to compute the number of workers in need. The given cycle time is changed iteratively until the number of workers in need meets the number of workers available.

3.2.1. Maximum-Utilization Heuristic. Given a cycle time, the maximum-utilization heuristic allocates workers and assigns tasks to workstations to minimize the total number of workers in need. Workstations, one after another, are assigned tasks and workers. For one station, the basic procedure of assignment is composed of several stages. Each stage corresponds to a possible number of workers. At the first stage, tasks are assigned to the station according to the feasible task sequence until the sum of the durations of assigned tasks becomes almost equal to the station capacity (which equals C at the first stage). If any additional task was added, the capacity constraint would be violated. At the following stages, tasks are assigned with capacity constraints of $2 \ast C$, $3 \ast C$, until $K \ast C$. Then the best combination of the number of workers and tasks is determined according to the criterion that the workstation utilization, which equals the sum of the durations of tasks in the workstation divided by the station capacity, should be maximized. After tasks and number of workers are determined for the current workstation, then we delete the assigned tasks from the task sequence, let them go to the next workstation, and repeat the same procedure, until all tasks have been assigned. Table 1 records the detailed procedure of the heuristic working on the feasible task sequence shown in Figure 3. The cycle time is 40 seconds.

3.2.2. Bisection Decoding. Calling the maximum-utilization heuristic repeatedly, bisection decoding carries out a bisection search for the minimum cycle time corresponding to the chromosome to be decoded. The detailed steps of the bisection decoding are as follows:

(i) Step 1: let us calculate the lower bound CL and upper bound CU of the cycle time given the number of workers available W.
(i) Input: dataset of mALBP-II, GA parameters (popsize, maxgen, cr, mr);
(ii) Output: task and worker assignment solution;
(iii) set $t := 0$;
(iv) initialize sequence population $P(t)$;
(v) calculate fitness function value $f_p(P)$ by heuristic-mixed decoding;
(vi) while (not maxgen generation) do;
(vii) save the elite of $P(t)$;
(viii) select $Pr(t)$ from $P(t)$ by tournament selection; create $Pc(t)$ from $Pr(t)$ by order crossover; create $C(t)$ from $Pc(t)$ by insertion mutation;
(ix) calculate fitness function value $f_p(C)$;
(x) combine $C(t)$ with the saved elite;
(xi) set $P(t + 1) := C(t)$; set $t := t + 1$;
(xii) end while;
(xiii) decode the elite;
(xiv) return task and worker assignment.

**Algorithm 1: hGA for mALBP-II.**

| Station | No. of workers | Total duration | Capacity | Utilization | Tasks |
|---------|----------------|----------------|----------|-------------|-------|
| 1       | 1              | 21             | 40       | 0.53        | 1     |
|         | 2              | 50             | 80       | 0.63        | 1, 3  |
|         | 3              | 89             | 120      | 0.74*       | 1, 3, 6 |
|         | 1              | 37             | 40       | 0.93*       | 2     |
| 2       | 2              | 70             | 80       | 0.88        | 2, 4  |
|         | 3              | 106            | 120      | 0.88        | 2, 4, 5 |
|         | 1              | 33             | 40       | 0.83        | 4     |
|         | 2              | 69             | 80       | 0.86        | 4, 5  |
|         | 3              | 106            | 120      | 0.88*       | 4, 5, 7 |

3.3. Selection Operator. In this study, tournament selection [19] is used to select parents for crossover in the next process. Tournament selection helps to avoid high fitness individuals dominating the entire population. The tournament size is typically two, which means two individuals are selected randomly from the current population, and the one with the highest fitness is selected. An elite strategy is also used to make sure that the current best individual is passed onto the next generation.

3.4. Crossover Operator. Order crossover (OX) [19] is adopted as the crossover operator in this paper. The procedure guarantees the feasibility of task sequences after crossover. It is demonstrated by an example which is shown in Figure 4.

3.5. Mutation Operator. Feasible insertion is used as the mutation operator in hGA. First, a task is randomly selected. Second, a feasible range is determined by finding out the closest immediate predecessor and successor task of the selected task in the chromosome. Finally, a gene position within the feasible range is randomly chosen and the selected task is inserted. An example of an insertion mutation is shown in Figure 5.
3.6. Fitness Function. The fitness function $f_p(n)$ returns a fitness value for each individual $n$ and it is defined as

$$f_p(n) = -\left(\left\lceil \frac{C_T(n)}{\text{prc}} \right\rceil \times \text{prc} + \text{std}(n) \times \text{prc} \right),$$

(12)

where $C_T$ is the cycle time obtained after the bisection decoding of individual $n$; prc is the precision required for the cycle time, e.g., 0.1 seconds; std($n$) means the standard deviation of utilization among workstations, so it is between 0 and 1. $\lceil \cdot \rceil$ is a round-up function. The fitness function first considers the cycle time; when the cycle time is equal, the task-worker assignment with more even workload distribution is preferred. Since usually fitness value is the larger the better, a negative sign is put ahead.

4. Numerical Result

To test the performance of the proposed heuristic-mixed genetic algorithm in solving mALBP-II, a series of computational experiments are carried out. Precedence graphs of benchmark problems in [20] are used for the tests. Table 2 summarized the characteristics of the test problems.

Numerical results are shown in Table 3. The program is coded by MATLAB and runs on a 1.66 GHz laptop. The maximum number of workers allowed in one station is three. The crossover rate and mutation rate are set as 0.8 and 0.2, respectively. Cycle time precision is 0.1 seconds. The best, average, worst cycle time, and average computation time (ACT, unit in second) of ten runs for each combination of population size and maximum number of generations are provided. An attempt is also made to solve the nonlinear integer programming in Section 2 by Lingo, and the best solutions reported within an acceptable runtime (24 hours) are listed.

Based on the numerical results, it is concluded that hGA solves mALBP-II effectively and efficiently. Compared to the solutions returned by Lingo, hGA always outputs better (at least no worse) solutions in ten runs in much shorter computational time. Particularly, when the size of problems becomes larger (problem 8 and problem 9), the superiority of the solutions returned by hGA turns out to be more obvious. In addition, all solutions by hGA are very close to the lower bound of cycle time (which is calculated by equation (9)); the optimality gaps, in terms of the lower bounds, are within 5%.

5. Case Study

In this section, a case study resulting from an interphone manufacturing plant in China is carried out to demonstrate the practicality of hGA in solving real-life problems. The tasks and task durations of assembling an interphone are listed in Table 4. The corresponding precedence graph of tasks is shown in Figure 6. It is worth noticing that the majority of tasks are welding and installation, requiring no special expensive tools. The assembly of interphones is a kind of typical manual manufacturing. The problem is, given the number of workers available, how to group workers into workstations and assign tasks to them so as to make cycle time as short as possible within the constraints of task precedence relationship and workstation capacity.

Problem settings and hGA parameters are summarized as follows:

(i) Given number of available workers is 20.
(ii) Maximum number of workers in one station is 3 due to space limitation.
(iii) Crossover rate $cr = 0.8$.
(iv) Mutation rate $mr = 0.2$. 
We ran hGA ten times, and one of the best solutions is presented in Table 5. Lingo is also used to solve the problem. The returned solutions are compared by hGA and Lingo, although they output the same cycle time after optimization, the solution by hGA is superior because the workstation utilizations (worker workloads) are more even among workstations.

(v) Population size popsize = 10.

(vi) Maximum number of generation maxgen = 100.
6. Conclusion

In this paper, we consider a type II line balancing problem with multiple workers in workstations (mALBP). Unlike the manufacturing environment, which depends heavily on automatic equipment, manual or semimanual manufacturing enjoys much higher flexibility, allowing multiple workers to perform the same set of tasks on workpieces in the same workstation simultaneously and independently. The freedom of specifying the number of workers in workstations makes the classic NP-hard ALBP even harder to solve. A heuristic-mixed genetic algorithm (hGA) is therefore proposed to solve the problem. The algorithm minimizes cycle time as its first objective and balances workload among workstations as its second objective. A greedy heuristic and a bisection search are integrated into the decoding process of hGA so that the optimization of task assignment and worker allocation is accomplished at the same time. Numerical results and a real-life application demonstrate the efficiency and effectiveness of hGA. In future work, the impact of task duration uncertainty will be taken into consideration. A simulation-based optimization framework is also under development.

Data Availability

The data for the case study have been included in the manuscript.

Conflicts of Interest

The author declares that there are no conflicts of interest.

Acknowledgments

This work is supported by the National Natural Science Foundation of China (Grant no. 72101162).

References

[1] T. S. Wee and M. J. Magazine, "Assembly line balancing as generalized bin packing," Operations Research Letters, vol. 1, no. 2, pp. 56–58, 1982.

[2] X. Qian and Q. Fan, "Solving multi-manned assembly line balancing problem by a heuristic-mixed genetic algorithm," in Proceedings of the 2011 International Conference on Information Management, Innovation Management and Industrial Engineering, pp. 320–323, November 2011.
[3] C. Becker and A. Scholl, "A survey on problems and methods in generalized assembly line balancing," *European Journal of Operational Research*, vol. 168, no. 3, pp. 694–715, 2006.

[4] J. Bukchin and J. Rubinovitz, "A weighted approach for assembly line design with station paralleling and equipment selection," *IIE Transactions*, vol. 35, no. 1, pp. 73–85, 2003.

[5] 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, no. 10, pp. 2717–2741, 1998.

[6] Y. Ege, M. Azizoglu, and N. E. Ozdemirel, "Assembly line balancing with station paralleling," *Computers & Industrial Engineering*, vol. 57, no. 4, pp. 1218–1225, 2009.

[7] 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, no. 3, pp. 666–693, 2006.

[8] O. Battai and A. Dolgui, "A taxonomy of line balancing problems and their solution approaches," *International Journal of Production Economics*, vol. 142, pp. 259–277, 2013.

[9] E. J. Anderson and M. C. Ferris, "Genetic algorithms for combinatorial optimization: the assemble line balancing problem," *ORSA Journal on Computing*, vol. 6, no. 2, pp. 161–173, 1994.

[10] J. Rubinovitz and G. Levitin, "Genetic algorithm for assembly line balancing," *International Journal of Production Economics*, vol. 41, no. 1-3, pp. 343–354, 1995.

[11] K. Yeo Keun Kim, K. Yong Ju Kim, and Y. Kim, "Genetic algorithms for assembly line balancing with various objectives," *Computers & Industrial Engineering*, vol. 30, no. 3, pp. 397–409, 1996.

[12] C. Öztürk, S. Tunali, B. Hnich, and A. Örnek, "Balancing and scheduling of flexible mixed model assembly lines with parallel stations," *International Journal of Advanced Manufacturing Technology*, vol. 67, pp. 2577–2591, 2013.

[13] W. Zhang, M. Gen, and L. Lin, "A multiobjective genetic algorithm for Assembly Line Balancing problem with worker allocation," in *Proceedings of the 2008 IEEE International Conference on Systems, Man and Cybernetics*, pp. 3026–3033, October 2008.

[14] T. Zaman, S. K. Paul, and A. Azeem, "Sustainable operator assignment in an assembly line using genetic algorithm," *International Journal of Production Research*, vol. 50, pp. 5077–5084, 2011.

[15] P. T. Zacharia and A. C. Nearchou, "A population-based algorithm for the bi-objective assembly line worker assignment and balancing problem," *Engineering Applications of Artificial Intelligence*, vol. 49, pp. 1–9, 2016.

[16] R. Walter, P. Schulze, and A. Scholl, "SALSA: combining branch-and-bound with dynamic programming to smoothen workloads in simple assembly line balancing," *European Journal of Operational Research*, vol. 295, no. 3, pp. 857–873, 2021.

[17] 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, no. 1, pp. 49–69, 2008.

[18] F. Akagi, H. Osaki, and S. Kikuchi, "A method for assembly line balancing with more than one worker in each station," *International Journal of Production Research*, vol. 21, no. 5, pp. 755–770, 1983.

[19] M. Gen and R. Cheng, *Genetic Algorithms and Engineering Optimization Vol. 7*, John Wiley & Sons, 2000.

[20] A. Scholl, "Homepage for assembly line optimization research," 2021, https://www.assembly-line-balancing.de.