Cycle Time Enhancement by Simulated Annealing for a Practical Assembly Line Balancing Problem

Mai Huong Dinh*
Hanoi University of Science and Technology, Hanoi, Vietnam
Hanoi University of Industry, Hanoi, Vietnam
E-mail: huongdm@haui.edu.vn

Viet Dung Nguyen*, Van Long Truong, Phan Thuan Do*, Thanh Thao Phan and Duc Nghia Nguyen
Hanoi University of Science and Technology, Hanoi, Vietnam
E-mail: dungnv@soict.hust.edu.vn, long.tv166391@sis.hust.edu.vn,
{thuan.dophan, thao.phanthanh}@hust.edu.vn, *in the memory of Professor Duc Nghia Nguyen*

Keywords: assembly line balancing, simulated annealing, meta-heuristic

Received: March 15, 2020

In the garment industry, assembly line balancing is one of the most significant tasks. To make a product, a manufacturing technique called assembly line is utilized, where components are assembled and transferred from workstation to workstation until the final assembly is finished. Assembly line should always be as balanced as possible in order to maximize efficiency. Different types of assembly line balancing problems were introduced along with many proposed solutions. In this paper, we focus on an assembly line balancing problem where the upper bound of the number of workers is given, tasks and workers have to be grouped into workstations so that the cycle time is minimized, the total number of workers is minimized and balance efficiency is maximized. With unfixed number of workstations and many other constraints, our problem is claimed to be novel. We propose three different approaches: exhaustive search, simulated annealing and simulated annealing with greedy. Computational results affirmed that our simulated annealing algorithm performed extremely good in terms of both accuracy and running time. From these positive outcomes, our algorithms clearly show their applicability potential in practice.

Povzetek: Problem uravnoteženja proizvodne poti je predstavljen odprto, brez omejitev npr. števila delavcev, zato je izviren. Avtorji testirajo več algoritmov in predlagajo najboljšega.

1 Introduction

The assembly line consists of a set of workstations arranged along a material transport system. Parts of the clothes are assembled on workstations, which are transported from workstation to workstation in one direction until the final product is completed. Workers perform a number of tasks in each workstation to create the product. The assembly line rhythm, which is called the cycle time, is the average time for each workstation to complete its tasks before transferring the product's subassembly to the next workstation. After that, the workstations receive new subassembles and repeat the assigned work.

In the fashion industry, patterns of clothing are everchanging, each product has a set of tasks to assemble the details of clothes together. Each task has unique time to be performed on a specific type of machine or done manually. Among tasks there are precedence relationships, where some tasks must be performed before some other tasks to create a certain clothing product. Different products will have different numbers of tasks and the precedence relationships among tasks will be different. The order of execution between tasks is a directed graph with no cycles.

The assembly line balancing problem (ALBP) is to assign tasks to workstations so that one or more objectives are optimized while ensuring that the precedence relationship among tasks are satisfied. Bryton [3] was the first to propose the ALBP in 1954, while Salveson [24] first published it in a mathematical form in 1955.

There are many ways to classify an ALBP. It is classified into simple and generalized problems [2, 26, 19]. The simple assembly line balancing problem (SALBP) is for straight single, dedicated lines, the duration of the task is determined and the only goal is to optimize line efficiency. SALBP according to the research objectives has been classified by Scholl and Becker [26] into 4 categories: SALBP-1, SALBP-2, SALBP-E, SALBP-F. SALBP-1 objective is minimizing the number of workstations given the cycle time [27]. SALBP-2 goal is to minimize the cycle time given the number of workstations [21]. SALBP-E relates to maximizing line efficiency while simultaneously considering the relationship between the number of workstations and the cycle time [11]. SALBP-F is a feasibility study to

* Corresponding authors

determine if there is a balanced assembly line when fixing both the number of workstations and the cycle time. However researchers usually want to challenge with more realistic problems than SALBP. These problems consider relevant factors including equipment selection, parallel workstations, mixed-model production, processing alternatives, etc. All of them form a very large class of problems called generalized ALBP (GALBP) [1].

The ALBP is usually based on a number of assumptions such as: only one type of product is produced on the line, the processing time of each task is determined and given, the processing sequence of tasks is subject to priority constraints (precedence relations), each task is only assigned to one worker, the maximum processing time of a task is less than the cycle time, etc. [12, 1]. When we assume that the maximum processing time of a task is less than the cycle time, the production speed of the assembly line is limited by that maximum processing time. This problem is solved by creating parallel workstations, which consist of two or more copies of the workstation performing the same group of tasks. The purpose of creating parallel workstations is to speed up production and balance time between workstations. However the number of tasks performed by each worker increases then and they require higher skills from workers. In order to control this process, several studies have set conditions when forming parallel workstations. Sarker and Shanthikumari [25] limited the number of parallel workstations. Buxey [4] limits the number of tasks per worker. McMullen et al. [22] allowed to form parallel workstations so that the processing time of workstations is closest to the cycle time. Vilarinho and Simaria [29] allowed the creation of parallel workstations when the maximum processing time of the tasks did not exceed twice of the cycle time and there is no more than two parallel workstations.

Since an ALBP usually falls into the NP-hard class of combinatorial optimization problems [13], heuristic algorithms are constantly being developed to estimate optimal solutions. Methods such as Hoffmann [15], Helgeson-Birnie [14], Kilbridge-Wester [17] have been applied to ALBP and have certain results, but the solution may fall into the local optimization area. To explore the optimal solution area, meta-heuristic methods were applied. Chiang [7], Lapierre et al. [20] applied the Tabu search method to solve type 1 of SALBP. Ponnambalam et al. [23] applied the multi-target genetic algorithm (GA) to consider different criteria such as number of workstations, line efficiency, the maximum difference between the processing time of workstations and CPU.

ALBP studies in the garment industry have been developed with the goals and constraints of various actual production models. Kayar and Akyalçin [16] have applied a number of heuristic methods to balance the line given the cycle time. Chen et al. [6] used GA to solve the ALBP with the goal of minimizing the number of workstations given the cycle time, while taking into consideration the influence of the skill level of each worker. Eryürük utilized heuristic methods to balance the assembly line in the garment industry in articles [9, 10] and compared the line efficiency of the methods applied when the cycle time was constant.

In this paper, we solve a GALBP in economic production conditions of Dong Van Garment Factory of Hanoi Textile and Garment Corporation, Vietnam. This problem is close to the SALBP of type 2, since its main objective is to minimize the cycle time. In our problem, the upper bound of number of workers (operators) is fixed. The factory mainly produces knitting products, each worker is trained to be able to carry out up to 3 different tasks, which requires not too many skills for workers in the garment industry. We have built a model of ALBP where given the upper bound of number of workers, the minimum cycle time has to be determined while ensuring numerous conditions. Having the minimum cycle time, we also need to determine the minimum number of workers and the maximum balance efficiency achievable. We deal with this GALBP by applying the result of our existing paper [8] which solved another GALBP where the minimum number of workers needs to be determined given the cycle time. Our GALBP has many similarities with the ALBP mentioned in [21], where the minimum cycle time has to be determined given the number of workers and parallel workstations. However, our problem has many different points compared to the one in [21]. While they fixed the number of workers on the assembly line, we allow it to be not greater than a certain number. Moreover, we allow each workstation to complete its tasks in a longer period of time than the cycle time, but not longer than *the upper limit of the cycle time* which will be mentioned later. Specifically, our research contributions are consistent with actual production in the following characteristics:

- The number of workers is not fixed, but the upper bound is given. This constraint is a very new factor, which is much more flexible in real conditions when the factory has a fixed budget for hiring labors or when they do not need to use all of their workers.
- Each workstation is allowed to have up to three workers and perform up to three tasks.
- There are up to two devices under specific constraints in each workstation.
- The processing time R (or cycle time) of each workstation should deviate by approximately $\pm\Delta R$ from the cycle time. Depending on the level of organization of the line, one of the following values is assigned to Δ: 5%, 10% or 15%. This is a very different case compared to other studies. The value $R + \Delta R$ is called *the upper limit of the cycle time*, the cycle time of each workstation must not be greater than this value. The interval $[R - \Delta R, R + \Delta R]$ is called the balanced cycle time interval, and workstations having cycle time within the balanced cycle time interval are called balanced work-

stations. These balanced workstations are the key factor to increase balance efficiency.

Tasks are combined into groups and assigned to workstations with the primary objective of minimizing the cycle time, which means maximizing production speed. The secondary goal is to minimize the total number of workers on the assembly line in order to reduce labor costs and save labor for other jobs in the factory. The tertiary goal is to maximize the proportion of balanced workstations out of all workstations created, thus maximize balance efficiency. In our solutions, binary search is used and proven to be correct to find the optimal cycle time, total number of workers and balance efficiency. Within each iteration of the binary search process, a meta-heuristic method is applied for approximate calculations. We propose three different approaches: exhaustive search, simulated annealing (SA) and simulated annealing with greedy. The proposed algorithms have been evaluated on the actual data set of Dong Van Garment Factory, Hanoi Textile and Garment Joint Stock Corporation, Vietnam. Computational results affirmed that our SA algorithm performed extremely good in terms of both correctness and time. From positive outcomes of this paper, our algorithms clearly show their applicability potential in practice.

2 Problem formulation

2.1 Notations

Throughout the paper, the following notations listed in Table 1 are used.

Table 1: Notation list

$Tasks$	Set of all tasks
M	Number of tasks in $Tasks$
$\hat{N}$	Upper bound of number of workers
N	Total number of workers
t_i	Processing time of task i
R	Cycle time
R'	Upper limit of the cycle time
Δ	Deviation coefficient of cycle time
S_i	Set of all tasks in workstation i
T_i	Total processing time of all tasks in workstation i
n_i	Number of workers in workstation i
R_i	Cycle time of workstation i
k	Total number of workstations
k'	Number of balanced workstations
H	Balance efficiency

2.2 Problem statement

Our GALBP has the primary goal of minimizing the cycle time given the upper bound of number of workers. The secondary goal is minimizing the total number of workers on the assembly line. Then the last goal is determining the maximum balance efficiency. Since its main goal is minimizing the cycle time which is also the objective of SALBP type 2, we denote our problem as GALBP-2. Some particular characteristics of it are described below.

– There is a set $Tasks$ of M tasks. The i^{th} task consumes t_i processing time performed by a machine or by manual work. These M tasks need to be assigned to workstations. Each task will be done in only one workstation.

– There are 3 types of tasks: Type 1 includes tasks using common machines, Type 2 includes tasks using special machines which requires a large investment while having short processing time and Type 3 is manual work.

– On the assembly line of a factory, each workstation can have up to three tasks. In a workstation, if two or three tasks use the same kind of machine, it is counted as one machine only. Machines/manual works assigned into a workstation must match with one of the three cases below:

  + All are manual works.
  + There is exactly one machine and other machines/-manual works, if there are any, are all manual works.
  + There are exactly two special machines (from Type 2 tasks).

– The precedence relations are represented as a directed acyclic graph. This precedence graph is used to determine the execution order of tasks during the assembly process. Some tasks must be done before other tasks. If there is an edge from task u to task v, it means that task u must be done before task v. As a consequence, a task u must be done before task v if there is a path from u to v on the precedence graph. Furthermore, between two different workstations X and Y, if there is a task $u \in X$ and a task $v \in Y$ such that u must be done before v, then there must not exist a task $u' \in X$ and a task $v' \in Y$ such that v' must be done before u', otherwise the product cannot be made.

– Workstations run in parallel.

– In each workstations, all workers do the same task at the same time before moving to another task. Therefore, the processing time (or cycle time) of each workstation i to finish all of its tasks, denoted by R_i, is equal to sum of processing time of all of its tasks (T_i) divide by the number of workers in it (n_i). After all tasks in a workstation are done, the process is operated again with the same set of tasks.

– The total number of workers in all workstations, denoted by N, must not exceed $\hat{N}$.

- The cycle time (or rhythm), denoted by R, is the time limit for each workstation to complete the assigned tasks before transferring the product to another workstation. The sum of all tasks' processing time in a workstation must not exceed $3(R + \Delta R)$. The cycle time R_i of each workstation i must not be greater than $R + \Delta R$, where Δ is given and takes one of the following values: 5%, 10% or 15%.
- If R_i lies within the balanced cycle time interval $[R - \Delta R, R + \Delta R]$, then workstation i is called a ***balanced workstation***.
- Balance efficiency, denoted by H, is the percentage of the number of workstations having their cycle time lies within $[R - \Delta R, R + \Delta R]$.

2.3 Optimization formulation

In addition with the problem statement, here are some induced constraints and formulas which completes the definition of our problem:

GALBP-2 objectives:

Primary: minimize R

Secondary: minimize N

Tertiary: maximize H

Input: $Tasks, \hat{N}, \Delta$ ($\Delta \in \{5\%, 10\%, 15\%\}$)

Definitions:

$$M = |Tasks| = \sum_{i=1}^{k} |S_i| \tag{1}$$

$$N = \sum_{i=1}^{k} n_i \tag{2}$$

$$H = \frac{k'}{k}.100(\%) \tag{3}$$

$$R' = R + \Delta R \tag{4}$$

$$\forall i, 1 \le i \le k : T_i = \sum_{j \in S_i} t_j \tag{5}$$

$$\forall i, 1 \le i \le k : n_i = \begin{cases} 1, & T_i \le R' & \text{(6a)} \\ 2, & R' < T_i \le 2R' & \text{(6b)} \\ 3, & 2R' < T_i \le 3R' & \text{(6c)} \end{cases}$$

$$\forall i, 1 \le i \le k : R_i = \frac{T_i}{n_i} \tag{7}$$

Constraints:

$$\forall i, 1 \le i \le k : |S_i| \le 3 \tag{8}$$

$$\forall i, j, 1 \le i < j \le k : S_i \cap S_j = \emptyset \tag{9}$$

$$\forall i, 1 \le i \le k : T_i \le 3R' \tag{10}$$

$$N \le \hat{N} \tag{11}$$

Constraint (8) ensures a workstation always has no more than 3 tasks. Constraint (9) along with definition (1) guarantees that a task can only be assigned to exactly one workstation. Constraint (10) shows that the total processing time of all tasks in a workstation is always less than or equal to 3 times the upper limit of the cycle time. Constraint (11) shows that the total number of workers cannot be greater than the upper bound of the number of workers which is given as input.

2.4 Examples

As an example, in Table 2 we show technological indexes of a Polo-Shirt product at Dong Van Garment Factory, Hanoi Textile & Garment Joint Stock Corporation, Vietnam (Figure 1). The process of manufacturing such a Polo-Shirt includes 25 tasks. In the table, $t_i(s)$ denotes the processing time of Task i.

The precedence graph of the Polo-Shirt product in Table 2 is shown in Figure 2, along with a sample assignment of tasks into workstations. This sample assignment ensures that the constraints about machines in a workstation and precedence relations are satisfied.

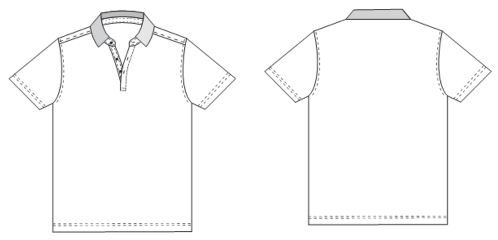

Figure 1: Model of Polo-Shirt.

Table 2: Product technological indexes of Polo-Shirt

No	Task	Machine	Type	t_i(s)
1	Check, mark placket	Check-table	1	32.0
2	Sew placket to front	Lockstitch machine	1	30.0
3	Topstitch placket	Lockstitch machine	1	118.5
4	Trim top of placket	Hand-made	3	12.0
5	Sew collar with collar band	Lockstitch machine	1	59.1

6	Trim bottom edge of collar band	Hand-made	3	12.0
7	Trim bottom edge of collar band	Hand-made	3	10.0
8	Sew shoulder	Overlock machine	1	23.2
9	Topstitch shoulder	1 needle - chainstitch machine	1	11.5
10	Sew collar band with 2 point top of placket	Lockstitch machine	1	27.3
11	Sew collar	Overlock machine	1	32.4
12	Topstitch collar band	Lockstitch machine	1	87.9
13	Attach sleeve set to armhole	Overlock machine	1	43.9
14	Topstitch armhole	1 needle - chainstitch machine	1	24.8
15	Side seam	Overlock machine	1	61.4
16	Hem bottom opening	2 needles - chainstitch machine	1	30.8
17	Hem sleeve opening	2 needles - chainstitch machine	1	41.6
18	Sew bottom of placket	Lockstitch machine	1	15.4
19	Sew bottom opening, sleeve opening	Lockstitch machine	1	23.0
20	Button hole on collar band	Button holing machine	2	9.5
21	Button hole on placket	Button holing machine	2	19.0
22	Button	Button machine	2	38.0
23	Bartack placket	Bartack machine	1	9.5
24	Bartack hem sleeve opening	Bartack machine	1	19.0
25	Trim thread	Hand-made	3	36.0

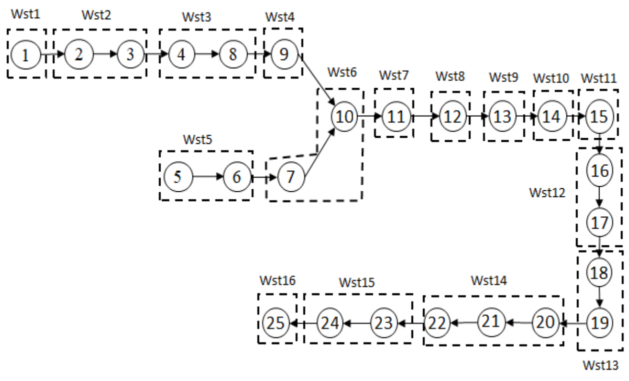

Figure 2: Precedence graph and a sample assignment of tasks into workstations.

3 Solution overview

3.1 Solution outline

To solve the GALBP-2 problem, we simply use binary search method to find the minimum cycle time. Because if there is a solution which consumes no more than $\hat{N}$ workers when $R = x$ then the minimum value of R is certainly not greater than x, and if such a solution does not exist it means that R must be greater than x (the correctness of this argument will be proven in section 3.2). To check for the existence of such a solution, we will have to solve a sub-problem which is also a GALBP: Given the set of all tasks, the values R and Δ, find a way to assign tasks into workstations in order to minimize the total number of workers.

The following **GALBP2** procedure is the framework for our solution. Given the set $Tasks$ of tasks, the upper bound of number of workers $\hat{N}$ and the deviation coefficient of cycle time Δ, **GALBP2** will produce an estimated optimal solution $wstSet$ which is a set of workstations, along with its corresponding values R, N, and H. The return value of **GALBP2** has the form $(R, N, H, wstSet)$. In this procedure, we assume that ϵ is a very small real positive number, α is the maximum processing time of a task in $Tasks$ and β is the sum of processing time of three tasks which have largest processing time in $Tasks$ (if $M \leq 3$ then β is the sum of processing time of all tasks in $Tasks$).

The procedure **GALBP1** inside **GALBP2** solves the sub-problem which is also a GALBP. It takes three parameters: $Tasks$, R and Δ. It generates a solution $wstSet$, which is set of workstations that first minimizes the total number of workers N and then maximizes the balance efficiency H. The return value of **GALBP1** is $(N, H, wstSet)$. This sub-problem is denoted as GALBP-1 because its primary objective is minimizing the total number of workers, close to the SALBP-1 which has the goal of minimizing the number of workstations where each workstation consists of only one worker. Since GALBP-1 is an NP-hard problem, **GALBP1** can only be approximately calculated. Therefore, several meta-heuristic methods are deployed in section 4 to increase the accuracy of

Procedure 1 Solve GALBP-2

Require: $Tasks$: set of all tasks,
$\hat{N}$: upper bound of total no. of workers,
Δ: deviation coefficient of the cycle time.

```
1: procedure GALBP2($Tasks$, $\hat{N}$, $\Delta$)
2:     $lowR \leftarrow \frac{\alpha}{3(1+\Delta)}$                ▷ min valid $R$
3:     $upR \leftarrow \frac{\beta}{1-\Delta}$                    ▷ max effective $R$
4:     while $upR - lowR > \epsilon$ do
5:         $R \leftarrow \frac{upR+lowR}{2}$
6:         $(N, H, wstSet) \leftarrow$ GALBP1$(Tasks, R, \Delta)$
7:         if $N > \hat{N}$ then
8:             $lowR \leftarrow R$
9:         else
10:            $upR \leftarrow R$
11:    $(N, H, wstSet) \leftarrow$ GALBP1$(Tasks, upR, \Delta)$
12:    if $N > \hat{N}$ then                                 ▷ no solution
13:        return $(\infty, \infty, 0\%, NULL)$
14:    else
15:        return $(upR, N, H, wstSet)$
```

this procedure.

3.2 Binary search correctness

Recall that in section 3.1, we have stated that if there is a solution which consumes no more than $\hat{N}$ workers when $R = x$ then the minimum value of R is certainly not greater than x, and if such a solution does not exist it means that R must be greater than x. The correctness of this argument is proven in Lemma 3.1 below.

Lemma 3.1. *Let $Tasks$ be any set of tasks and $\Delta \in \{5\%, 10\%, 15\%\}$. Let R_1, R_2 such that $\frac{\alpha}{3(1+\Delta)} \leq R_1 < R_2$, where α is the maximum processing time of a task in $Tasks$. Assume that procedure* $\mathbf{GALBP1}$ *can always produce an accurate result, if we set $(N_1, H_1, wstSet_1) = \mathbf{GALBP1}(Tasks, R_1, \Delta)$ and $(N_2, H_2, wstSet_2) = \mathbf{GALBP1}(Tasks, R_2, \Delta)$, then $N_1 \geq N_2$.*

Proof. First we need to show that for all $R \geq \frac{\alpha}{3(1+\Delta)}$, a valid solution for **GALBP1** always exists. Indeed, a solution where each workstation contains exactly one task would fit all the constraints mentioned in the problem statement.

Then, we consider an interesting observation here: $wstSet_1$ is also a solution when $R = R_2$ since all mentioned constraints are still satisfied. Moreover, if $R = R_2$, solution $wstSet_1$ will consumes not as many workers as itself when $R = R_1$, because of the way we calculate the number of workers in each workstations. Assume that when $R = R_2$, $wstSet_1$ consumes N workers, then we have $N_2 \leq N \leq N_1$ which is what we want to prove. □

Actually, when $R < \frac{\alpha}{3(1+\Delta)}$, there will be no solution. Because there exists at least one workstation i which has $T_i > 3(R + \Delta R)$, contradict with problem statement. Therefore setting $lowR = \frac{\alpha}{3(1+\Delta)}$ at the beginning of procedure **GALBP2** is indeed appropriate. Moreover when $R > \frac{\beta}{1-\Delta}$, the minimum number of workers stops to decrease further, so initializing $upR = \frac{\beta}{1-\Delta}$ is suitable too.

4 Methods to estimate the procedure GALBP1

With the application of **GALBP2** procedure, our original GALBP-2 is turned into solving another GALBP-1 in procedure **GALBP1**. GALBP-1 is very similar to the original problem GALBP-2, with all the constraints remain the same except that the number of workers is not bound anymore.

Since the GALBP-1 in procedure **GALBP1** is an NP-hard problem, it cannot be fully solved in polynomial time. Therefore, we tried to apply exhaustive search (brute-force search) along with different meta-heuristic methods such as simulated annealing (SA for short) and SA with greedy to produce answers as close as possible to the optimal ones. For a similar version of this GALBP-1 where H must not be less than 80%, we have already proposed an efficient SA algorithm [8] which performs excellently in terms of accuracy and speed. Therefore, with some reasonable modifications, we could expect our same methods to work well in this GALBP-1.

Throughout section 4, we introduce about our approaches in detail to cope with this GALBP-1. The following section 5 will contain a full evaluation of all methods when being applied to solve our original GALBP-2 based on experimental results on real data of the garment industry.

4.1 Exhaustive search

The exhaustive search finds the optimal result by considering all possible solutions. We design a simple exhaustive search algorithm for this GALBP-1 in the procedure 2.

In this procedure, wst is the current built workstation which consists of tasks, $curSol$ is the current solution which is a set of workstations and $bestSol$ is the current best solution. By initializing $bestSol$ as a random valid solution and calling $\mathbf{exhaustive}(1, 1, \emptyset, \emptyset)$, we will have $bestSol$ as our optimal solution when $\mathbf{exhaustive}$ terminates.

For our Polo-Shirt example, when the number of tasks is not too large, the exhaustive search can still return an optimal solution after a reasonable time. Nonetheless, when input is big enough, it takes hours to run until termination, which is infeasible in industrial environment. Therefore, better approaches should be applied to deal with this problem.

Procedure 2 Exhaustive search for GALBP-1

```
Require: $i$: $1^{st}$ task in current workstation,
         $j$: last added task in current workstation,
         $wst$: current workstation,
         $curSol$: current solution.

1:  procedure exhaustive($i$, $j$, $wst$, $curSol$)
2:      if $i > M$ then
3:          if $curSol$ is better than $bestSol$ then
4:              $bestSol \leftarrow curSol$
5:      else if $wst = \emptyset$ then
6:          if $task_i$ is marked then
7:              exhaustive($i+1, i+1, \emptyset, curSol$)
8:          else
9:              Push $task_i$ into $wst$
10:             exhaustive($i, i, wst, curSol$)
11:             Pop $task_i$ out of $wst$
12:     else
13:         if $wst$ is valid then
14:             Mark all tasks in $wst$
15:             Push $wst$ into $curSol$
16:             exhaustive($i+1, i+1, \emptyset, curSol$)
17:             Pop $wst$ out of $curSol$
18:             Unmark all tasks in $wst$
19:         if $|wst| < 3$ then
20:             for $k \leftarrow j+1$ to $M$ do
21:                 if $task_k$ is not marked then
22:                     Push $task_k$ into $wst$
23:                     exhaustive($i, k, wst, curSol$)
24:                     Pop $task_k$ out of $wst$
```

4.2 Simulated annealing

SA algorithm has been widely applied due to its feasibility in NP-hard problem classes through a randomized controlled process with reasonable calculation time. Therefore, the SA algorithm is a good tool for ALBP with a lot of constraints.

4.2.1 Motivation and idea

Simulated annealing (SA for short) was first applied to optimization problems by S. Kirkpatrick et al. [18] and V. Cerny [5]. In the book "Metaheuristics: From design to implementation" of El-Ghazali Talbi [28], the author described almost every aspect of SA in detail. It is a metaheuristic to approximate optimal solution in a large search space for an optimization problem. The idea of SA algorithm is derived from physical metallurgy. The metal is heated to high temperatures and cooled slowly so that it crystallizes in a low energy configuration.

SA is chosen to solve this ABLP because of its simplicity and efficiency. It allows for a more extensive search for the global optimal solution, and can even find a global optimal solution if it runs for enough amount of time.

We represent our SA approach in Procedure 3. This Procedure is a close edition of the general SA algorithm from Talbi's book [28].

Procedure 3 Simulated Annealing

```
Require: $s_0$: initial solution,
         $T_{max}$: starting temperature,
         $L$: neighbor generation loop time limit,
         $T_{dec}$: temperature drops after each step,
         $P$: probability to accept worse solution.

1:  procedure SA($s_0, T_{max}, L, T_{dec}, P$)
2:      $s \leftarrow s_0$
3:      $T \leftarrow T_{max}$
4:      while $T > 0$ do
5:          for $i \leftarrow 1$ to $L$ do
6:              Generate a random neighbor $s'$
7:              if $s'$ is better than $s$ then
8:                  $s \leftarrow s'$
9:              else
10:                 Assign $s \leftarrow s'$, probability $P(T)$
11:         $T \leftarrow T - T_{dec}$
12:     return Best solution found
```

There are five parameters that we need to decide for SA: s_0 as the initial solution; starting temperature T_{max}, L and T_{dec} for cooling schedule; and P as the acceptance probability of moving to a worse solution. Also we need to design a procedure to generate a random neighbor s' from a current solution s. All these factors will affect the quality of our algorithm.

4.2.2 Initial solution

In theory, the initial solution s_0 can be any valid solution and it does not affect the quality of SA. However, when the solution searching space is too large, a good initial solution can be a suitable approximation for the global optimum in a short amount of time. In section 4.2 we set a random solution as the initial solution for SA, and in section 4.3 we will assign a solution obtained from a greedy method to s_0. Result comparison between these two approaches shows a remarkable efficiency difference.

4.2.3 Neighbor generation

A neighbor of a solution s is generated simply by moving a task from a workstation to another workstation (including creating a new workstation consist of only that task) or swapping two tasks in two different workstations. There are at most M^2 valid neighbors of a solution.

Among all valid neighbors of s, we just consider its χ best neighbors and randomly choose one of them. The reason why we do not choose among all valid neighbors is to save computation cost without worsening the algorithm efficiency too much.

χ is set high at the beginning and decreases as the temperature decreases, so that when temperature is high a wide range of neighbor is considered and at the end only better solutions are chosen.

4.2.4 Move acceptance

Usually, the probability P that a worse solution is accepted depends on the current temperature T, the current solution s and the new solution s'. One of the most basic forms of P [28] can be written as:

$$P(T, s, s') = e^{-\frac{f(s')-f(s)}{T}} = e^{-\frac{\Delta E}{T}}$$

In which $\Delta E = f(s') - f(s)$ is the different of quality between the new and current solution. However in our SA algorithm, P depends only on T by a simple formula:

$$P(T) = \frac{T}{T_{max}}$$

ΔE is not used in our case since the quality of s and its chosen neighbor s' are not too different, they are even very close. Because s' is generated from s by just moving a task from a workstation to another or swapping two tasks in two workstations, and also s' is chosen among χ best neighbors of s. Therefore, ΔE tends to be very small and negligible. Also, it is very hard to find an ideal formula for calculating the quality of a solution. Any tuned formula for a solution's quality is just overfit to some set of tests and performs badly in other tests.

Computational results show that $P(T)$ works well compared to any tuned version of $P(T, s, s')$ that we design. Moreover, in our case $P(T)$ formula is much simpler and more reasonable.

4.2.5 Cooling schedule

In theory, the higher T_{max} and L are the higher chance for optimal solution to be discoverable. Similarly, the lower T_{dec} is, the better our final solution will be. However, to save computation energy, these three parameters should be carefully tuned.

4.2.6 Multiple execution

Since the solution search space for this GALBP-1 is very large, it is not guaranteed that when SA is applied on a unique input, a unique output will be produced. Therefore, given an input, SA algorithm will be repeated multiple times to provide multiple answers, then the best answer among them will be the solution for **GALBP1** procedure. By experimenting on actual data, we realize that 10 times of repetition is enough to stabilize our SA algorithm without taking too much of time.

4.3 Simulated annealing with greedy

A good initial solution provided by a greedy approach can always be a suitable approximation for the optimal result in a short amount of time. Also, when the solution search space is too large, it could help SA to find better final solution by focusing the process on a critical region only. With our GALBP-1, our initial solution s_0 for SA is constructed by a 5-step algorithm described below:

* **Step 1**: Choose a task u such that there is not any remaining task $v \neq u$ where v must be done before u is processed.

* **Step 2**: Create a workstation X which contains u and some of the remaining tasks so that X is valid and the following Ws_X value is maximize (Ws here stands for "worker saved"):

$$Ws_X = n'_X - n_X$$

Where n'_X is the total number of workers needed to complete all the tasks in workstation X if we divide these tasks into separated one-task-only workstations. If there are many workstations X with the same value Ws_X, choose any workstation which is balanced.

* **Step 3**: Add X to s_0.

* **Step 4**: Remove all tasks belong to X.

* **Step 5**: If there is some task remaining, go back to Step 1.

At step 2 of this algorithm, a greedy strategy is utilized: the best workstation which contains task u is added to the solution. Such a strategy efficiently exploits a signature property of an assembly line: Its precedence graph is almost identical to a tree with only a few number of branches. Therefore, a workstation tends to consist of connected tasks on the precedence graph, and removing them does not affect our future decisions so much. Indeed, experimental results which will be discussed in section 5 show that the SA with greedy solution's efficiency is usually better than that of exhaustive search and traditional SA, in terms of both accuracy and running time.

5 Computational results

If the exhaustive search procedure were allowed to run fully, it would take several hours or even days until termination which is infeasible in industrial environment. Therefore, for each test, we forced it to terminate when it is called more than 6×10^6 times recursively, and its best produced result is collected. Besides that, for all versions of SA, we set $T_{max} = 100, L = 20$ and $T_{dec} = 5$ to guarantee solution quality without consuming too much time. All algorithms are implemented in C++, and run on a computer which has 2.60GHz i7-8850H CPU (12 CPUs), NVIDIA Quadro P1000, 16GB RAM and 512GB SSD.

Our algorithms were tested on real data set related to the production of Polo-Shirt products at Dong Van Garment Factory, Hanoi Textile & Garment Joint Stock Corporation, Vietnam. There are 12 cases, where 6 tests are created from each of these cases by modifying Δ and $\hat{N}$. The values of Δ and $\hat{N}$ for each test are the combinations of three values of Δ $(5\%, 10\%$ and $15\%)$ and two different values of $\hat{N}$ where $\hat{N}_{high}$ the greater is about twice as $\hat{N}_{low}$ the smaller and $\hat{N}_{low} \leq 1.5M$. $\hat{N}_{high}$ and $\hat{N}_{low}$ are different among cases. Hence there are 72 tests in total. The number of tasks M spreads evenly among tests, from 15 to 60. The performance of each algorithm on all tests in terms of the cycle time R, number of workers N, balance efficiency H

Table 3: Results for tests having $\Delta = 5\%$ and $\hat{N} = \hat{N}_{low}$

M	N̂	R-Ex	R-SA	R-SA-gr	N-Ex	N-SA	N-SA-gr	H-Ex (%)	H-SA (%)	H-SA-gr(%)	T-Ex (s)	T-SA (s)	T-SA-gr(s)
15	18	31.429	31.429	31.429	18	18	18	54.545	54.545	54.545	0.055	10	10
20	18	30.952	31.157	30.953	18	18	18	55.556	55.556	50	37	28	28
25	24	41.857	41.857	41.857	24	24	24	29.412	29.412	29.412	11	57	55
30	28	41.81	41.81	41.81	28	28	28	23.81	23.81	19.048	110	89	90
32	26	84.762	72.621	69.02	26	26	26	23.81	38.889	29.412	843	125	128
33	36	93.016	92.245	91.429	36	36	36	33.333	25	37.5	952	125	127
35	32	42.857	37.871	37.66	32	32	32	28	52.381	44.444	911	155	162
47	33	No	35.05	32.273	No	32	33	No	34.783	54.545	1240	439	432
48	40	87.619	61.903	60.272	40	40	40	14.286	32	40.741	1368	343	347
50	40	47.143	34.822	33.225	40	40	40	11.111	33.333	46.154	1353	439	447
52	52	66.952	53.401	48.857	49	51	51	18.421	32	52	1552	531	531
60	40	No	71.011	59.831	No	38	40	No	32.143	58.333	1625	895	897

and running time in seconds is documented to make diagrams on Figure 3.

The top six diagrams on Figure 3 show the cycle time R obtained from exhaustive search, SA and SA with greedy algorithms, divide by a number R_0 which is calculated as:

$$R_0 = \frac{\sum_{i=1}^{M} t_i}{\hat{N}} \tag{12}$$

R_0 is used as an approximation for the lower bound of R, since if $\Delta = 0\%$ then R_0 is exactly the lower bound of R and actually Δ is quite small ($\Delta \leq 15\%$) which means the real lower bound of R is not so different from R_0. Therefore R_0 is used to normalize R. Among the top six diagrams, the upper three of them consist of tests having $\hat{N} = \hat{N}_{low}$ and the lower three consist of tests having $\hat{N} = \hat{N}_{high}$. Each column contains a pair of diagrams sharing a particular Δ value ($5\%, 10\%$ or 15%). The same order applies to diagrams of the balance efficiency H and running time.

For example, Table 3 shows results of 12 tests having $\Delta = 5\%$ and $\hat{N} = \hat{N}_{low}$. Here "Ex" is exhaustive search, "SA" is simulated annealing and "SA-gr" is simulated annealing with greedy. These results are used to built the top-left diagram in each set of six diagrams in Figure 3.

Since R_0 is an approximation for the lower bound of R, a value of R is a good answer if it is not so far from R_0. When $\hat{N} = \hat{N}_{low}$, based on Figure 3, we can see that both SA and SA with greedy results are as good as results of exhaustive search in small tests but much better than exhaustive search in medium and large tests. Even in some cases, due to early termination, exhaustive search does not provide any valid solution, as opposed to SA algorithms, which still produces quality answers for all tests. In case of $\hat{N} = \hat{N}_{high}$, the results of R may not be close to R_0 since $\hat{N}_{high} \approx 2\hat{N}_{low}$ can be a bit too high which made R_0 too much lower than the real lower bound of R. Nevertheless, SA algorithms still show that they are always not worse than exhaustive search. In addition, SA with greedy is usually slightly better than traditional SA in terms of R, which reveals the effectiveness of greedy initial solution.

For the balance efficiency H, SA algorithms can be slightly worse than brute force when the number of tasks M is small. However as M grows larger, SA algorithms clearly become superior to the exhaustive one. Moreover, H is usually higher than 40% and often fluctuates from 60% to 80% when SA is utilized which are quite satisfying outcomes. A point worth noting is that SA with greedy is remarkably better than exhaustive search and traditional SA in almost all test cases.

In case of running time, SA algorithms completely outperform exhaustive search as expected since they are polynomial time algorithms while exhaustive search theoretically runs in exponential time. Also, results are produced from SA in less than 20 minutes even for the largest test cases. With its fast processing speed, SA is perfectly suitable for real industrial environment.

With all the evaluation above, we can conclude that SA is an efficient meta-heuristic for our GALBP-2. In addition,

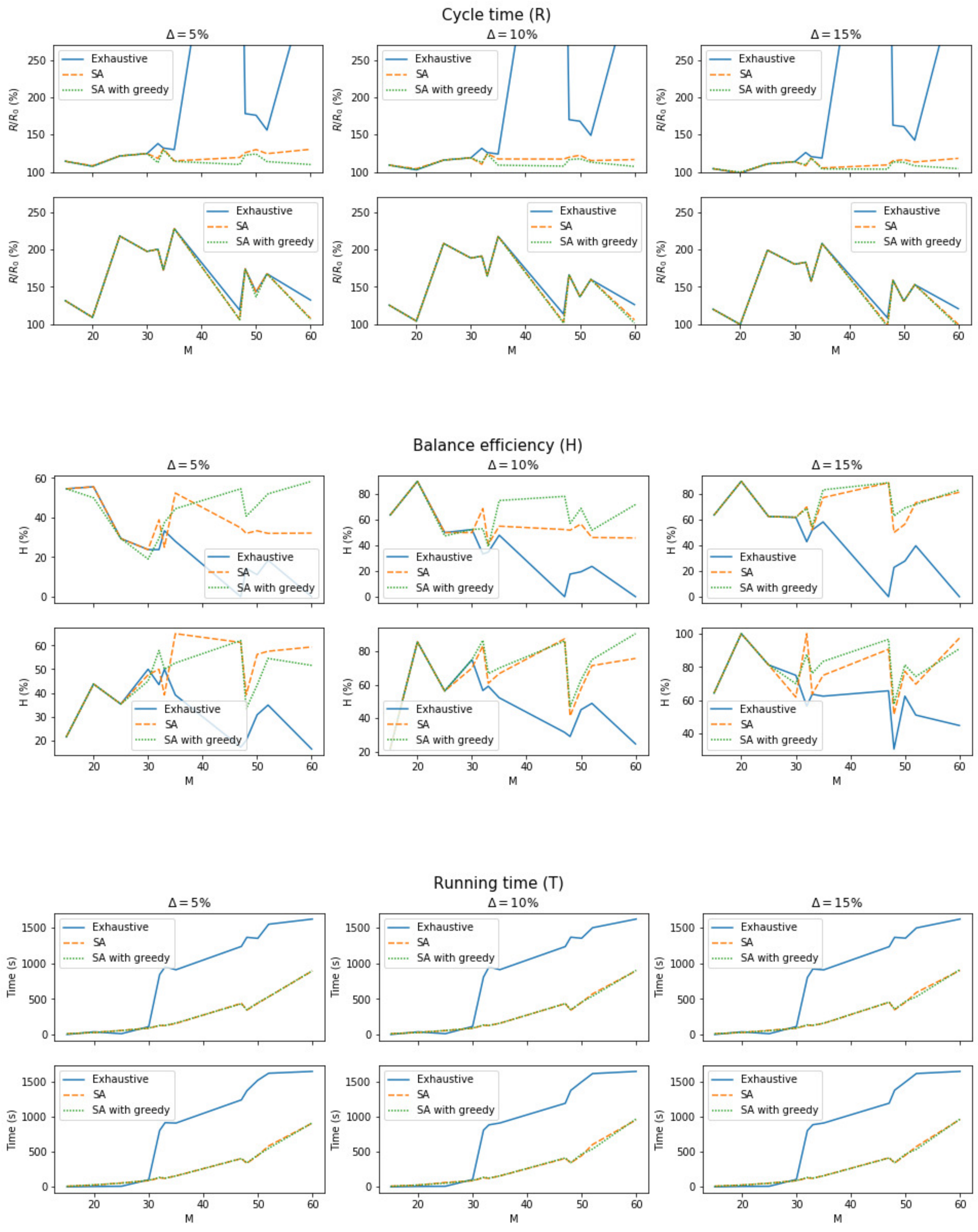

Figure 3: Diagrams of cycle time (R), balance efficiency (H) and running time of exhaustive search, SA and SA with greedy on 72 tests from Dong Van Garment Factory, Hanoi Textile & Garment Joint Stock Corporation, Vietnam.

the SA with greedy version is clearly the most excellent, compared to both exhaustive search and traditional SA.

6 Conclusion

In this paper, we represented a Simulated Annealing algorithm to solve a generalized assembly line balancing problem in the garment industry. Our GALBP-2 has the primary goal of minimizing the cycle time given the upper bound of number of workers. The secondary goal is minimizing the total number of workers on the assembly line. Then the last goal is determining the maximum balance efficiency. We efficiently utilized binary search to turn the original problem into a simpler problem GALBP-1, where the primary objective is minimizing the total number of workers and the secondary goal is maximizing the balance efficiency, given the cycle time. Then we introduced three methods to solve this GALBP-1: exhaustive search, SA and SA with greedy. All of them have their particular advantages in terms of accuracy and running time, depend on different test sizes. These algorithms are good supporting tools for garment factory managers to make plans before decisions. In other real assembly line balancing cases, our mentioned methods should also be considered as promising directions.

Acknowledgments

The authors would like to acknowledge the Dong Van Garment Factory, Hanoi Textile & Garment Joint Stock Corporation, Vietnam for supporting survey, experiment to complete this study.

References

[1] Ilker Baybars. A survey of exact algorithms for the simple assembly line balancing problem. *Management science*, 32(8):909–932, 1986. https://doi.org/10.1287/mnsc.32.8.909.

[2] Nils Boysen, Malte Fliedner, and Armin Scholl. A classification of assembly line balancing problems. *European journal of operational research*, 183(2):674–693, 2007. https://doi.org/10.1016/j.ejor.2006.10.010.

[3] Benjamin Bryton. *Balancing of a continuous production line*. PhD thesis, Northwestern University, 1954.

[4] GM Buxey. Assembly line balancing with multiple stations. *Management science*, 20(6):1010–1021, 1974. https://doi.org/10.1287/mnsc.20.6.1010.

[5] Vladimír Černỳ. Thermodynamical approach to the traveling salesman problem: An efficient simulation algorithm. *Journal of optimization theory and applications*, 45(1):41–51, 1985. https://doi.org/10.1007/bf00940812.

[6] James C Chen, Chun-Chieh Chen, Yi-Jhen Lin, CJ Lin, and TY Chen. Assembly line balancing problem of sewing lines in garment industry. In *Proceedings of the 2014 International Conference on Industrial Engineering and Operations Management Bali, Indonesia*, pages 7–9, 2014. https://doi.org/10.1109/icmlc.2009.5212600.

[7] Wen-Chyuan Chiang. The application of a tabu search metaheuristic to the assembly line balancing problem. *Annals of Operations Research*, 77:209–227, 1998.

[8] Mai Huong Dinh, Viet Dung Nguyen, Van Long Truong, Phan Thuan Do, Thanh Thao Phan, and Duc Nghia Nguyen. Simulated annealing for the assembly line balancing problem in the garment industry. In *Proceedings of the Tenth International Symposium on Information and Communication Technology*, pages 36–42, 2019. https://doi.org/10.1145/3368926.3369698.

[9] Selin Hanife ERYÜRÜK. Clothing assembly line design using simulation and heuristic line balancing techniques. *Journal of Textile & Apparel/Tekstil ve Konfeksiyon*, 22(4), 2012.

[10] SH Eryuruk, F Kalaoglu, and M Baskak. Assembly line balancing in a clothing company. *Fibres & Textiles in Eastern Europe*, 66(1):93–98, 2008.

[11] Rasul Esmaeilbeigi, Bahman Naderi, and Parisa Charkhgard. The type e simple assembly line balancing problem: A mixed integer linear programming formulation. *Computers & Operations Research*, 64:168–177, 2015. https://doi.org/10.1016/j.cor.2015.05.017.

[12] Waldemar Grzechca. Assembly line balancing problem with reduced number of workstations. *IFAC Proceedings Volumes*, 47(3):6180–6185, 2014. https://doi.org/10.3182/20140824-6-za-1003.02530.

[13] Allan L Gutjahr and George L Nemhauser. An algorithm for the line balancing problem. *Management science*, 11(2):308–315, 1964. https://doi.org/10.1287/mnsc.11.2.308.

[14] WB Helgeson and Dunbar P Birnie. Assembly line balancing using the ranked positional weight technique. *Journal of industrial engineering*, 12(6):394–398, 1961.

[15] Thomas R Hoffmann. Assembly line balancing with a precedence matrix. *Management Science*, 9(4):551–562, 1963. https://doi.org/10.1287/mnsc.9.4.551.

[16] Mahmut Kayar and Ö C Akyalçin. Applying different heuristic assembly line balancing methods in the apparel industry and their comparison. *Fibres & Textiles in Eastern Europe*, 2014. https://doi.org/10.5604/12303666.1191438.

[17] Maurice D Kilbridge and Leon Wester. A heuristic method of assembly line balancing. *Journal of Industrial Engineering*, 12(4):292–298, 1961.

[18] Scott Kirkpatrick, C Daniel Gelatt, and Mario P Vecchi. Optimization by simulated annealing. *science*, 220(4598):671–680, 1983. https://doi.org/10.1126/science.220.4598.671.

[19] N Kriengkorakot and N Pianthong. The assembly line balancing problem: Review problem. *J. Ind. Eng*, 6(3):18–25, 1955.

[20] Sophie D Lapierre, Angel Ruiz, and Patrick Soriano. Balancing assembly lines with tabu search. *European journal of operational research*, 168(3):826–837, 2006. https://doi.org/10.1016/j.ejor.2004.07.031.

[21] Yuchen Li, Honggang Wang, and Zaoli Yang. Type ii assembly line balancing problem with multi-operators. *Neural Computing and Applications*, 31(1):347–357, 2019. https://doi.org/10.1007/s00521-018-3834-1.

[22] Patrick R McMullen and GV Frazier. Using simulated annealing to solve a multiobjective assembly line balancing problem with parallel workstations. *International Journal of Production Research*, 36(10):2717–2741, 1998. https://doi.org/10.1080/002075498192454.

[23] SG Ponnambalam, P Aravindan, and G Mogileeswar Naidu. A multi-objective genetic algorithm for solving assembly line balancing problem. *The International Journal of Advanced Manufacturing Technology*, 16(5):341–352, 2000. https://doi.org/10.1007/s001700050166.

[24] M. E. Salveson. Induced matchings in intersection graphs. *The Journal of Industrial Engineering*, 6(3):18–25, 1955.

[25] Bhaba R Sarker and JG Shanthikumari. A generalized approach for serial or parallel line balancing. *THE INTERNATIONAL JOURNAL OF PRODUCTION RESEARCH*, 21(1):109–133, 1983. https://doi.org/10.1080/00207548308942341.

[26] Armin Scholl and Christian Becker. State-of-the-art exact and heuristic solution procedures for simple assembly line balancing. *European Journal of Operational Research*, 168(3):666–693, 2006. https://doi.org/10.1016/j.ejor.2004.07.022.

[27] Yuri N Sotskov, Alexandre Dolgui, Tsung-Chyan Lai, and Aksana Zatsiupa. Enumerations and stability analysis of feasible and optimal line balances for simple assembly lines. *Computers & Industrial Engineering*, 90:241–258, 2015. https://doi.org/10.1016/j.cie.2015.08.018.

[28] El-Ghazali Talbi. *Metaheuristics: from design to implementation*, volume 74. John Wiley & Sons, 2009. https://doi.org/10.1002/9780470496916.

[29] Pedro M Vilarinho and Ana Sofia Simaria. A two-stage heuristic method for balancing mixed-model assembly lines with parallel workstations. *International Journal of Production Research*, 40(6):1405–1420, 2002. https://doi.org/10.1080/00207540110116273.