Contents lists available at GrowingScience

International Journal of Industrial Engineering Computations

homepage: www.GrowingScience.com/ijiec

Research on workload balance problem of mixed model assembly line under parallel task strategy

Kang Wang[a,b], Yuwei Zhang[b*] and Zhenping Li[b*]

[a]*School of Management Engineering, Capital University of Economics and Business, Beijing, Republic of China*
[b]*School of Systems Science and Statistics, Beijing Wuzi University, Beijing, Republic of China*

C H R O N I C L E

Article history:
Received October 12 2024
Received in Revised Format December 29 2024
Accepted January 21 2025
Available online January 21 2025

Keywords:
Mixed-model assembly line
Mixed-integer programming
Parallel task
Load balancing
Improved Simulated Annealing Algorithm

A B S T R A C T

Aiming at the inefficiency caused by an unbalanced workstation load in the mixed-model assembly line (MMAL), we study the assembly line (AL) design and load balancing problem under parallel tasks. Considering the task configuration cost, workstation opening cost and penalty cost of unbalanced load on the assembly line, a mixed integer programming model with the workstation's space capacity constraint is established to formulate the mixed-model assembly line load balancing problem (MMALLBP), which is aiming at minimizing the total cost. In addition, the simulated annealing algorithm with an improvement strategy is proposed. Numerical experiments using the improved simulated annealing algorithm are superior to the solver in terms of solving time and stability, and the solving accuracy is higher than that of the traditional simulated annealing algorithm. Allowing parallel tasks can flexibly allocate tasks to the workstations, effectively use the idle time of the workstations, reduce the number of opened workstations, improve the production efficiency, reduce construction costs and the risk caused by the unbalanced load of AL.

1. Introduction

In recent years, the diversification and personalization of customer requirements have made it difficult for single-model assembly lines to respond quickly to market fluctuations. As a result, an increasing number of manufacturing companies are turning to mixed-model assembly line (MMAL) for its high efficiency and flexibility. The design and operation of MMAL revolves around two key considerations: assembly line design and product sequencing. Assembly line design involves allocating the necessary processing equipment to workstations for each task. On the other hand, product sequencing involves determining the order in which products are introduced into the assembly line (AL), with the aim of minimizing waiting time at each workstation and increasing production efficiency. In the aforementioned two issues, the design of MMAL plays a crucial role in influencing the production efficiency. In recent years, a considerable amount of research has been carried out by scholars on the design of MMAL. These studies can be categorized into two different optimization objectives: the first category focuses on optimizing the number of workstations given a cycle time, while the second category aims to optimize the cycle time given a fixed number of workstations. Workload balancing of individual workstations is a key factor influencing AL efficiency. Research on assembly line balancing design problems both domestically and internationally has primarily focused on two traditional categories, type I with a given cycle time and type II with a given number of workstations. However, the unique characteristics of MMAL introduce complexities and challenges to the balancing design problem.

The rest of this paper is organized as follows: Section 2 presents the literature review and the main contributions of this paper. Section 3 describes our problem, introduces assumptions and relevant parameters, and presents the mathematical model. Then, in Section 4, a solution method is developed, and in Section 5, the computational results based on the assembly line problem set are discussed and the performance of the algorithm is analyzed accordingly. The results of two strategies with and without parallel tasks are compared and evaluated, and some practical management insights are provided. Finally, Section 6 presents the conclusion and future directions for further research.

* Corresponding author
E-mail morbenvy@126.com (Y. Zhang) lizhenping@bwu.edu.cn (Z. Li)
ISSN 1923-2934 (Online) - ISSN 1923-2926 (Print)

doi: 10.5267/j.ijiec.2025.1.005

2. Literature review

The literature review in this manuscript primarily commences by addressing the conventional categorizations of assembly line balancing problem, encompassing considerations for load balancing. Subsequently, it enumerates and scrutinizes the pertinent challenges associated with flexible assembly line balancing.

Mei et al. (2023) considered a complex multi-manned assembly line balancing problem (CMALBP) for a given cycle time for minimizing the number of workstations and reducing the idle time of workstations. They set up a multi-objective optimization model with the goal of minimizing the number of workstations and workers, and designed a water wave algorithm to solve the problem by taking load balancing between stations as the performance index of the algorithm. Li et al. (2023) established a mixed integer programming model with the objectives of the number of workstations, load balancing among workstations and product waiting time, and designed a hybrid heuristic algorithm to solve it. Wang and Liu (2020) considered human factors, introduced mental and physical load models, established an AL balance model considering workers' load with the goal of minimizing cycle time and load balance index, and designed a fast non-dominated sorting genetic algorithm with elite strategy to solve this problem. Hou et al. (2014) established a model aiming at minimizing the number of workstations and the load of workstations, which was solved by a genetic algorithm. Tang et al. (2018) designed an improved iterative local search algorithm to balance workstation load with the goal of minimizing the number of workstations. Çil et al. (2020) considered AL under human-machine cooperation, combined the high efficiency of robots and the personalized production advantages of workers, established a mixed integer programming model with the goal of minimizing the cycle time, and obtained a satisfactory solution of AL layout through an improved artificial bee colony algorithm.

To the best of our knowledge, most current studies focus on minimizing the number of workstations and load balancing as optimization goals, with little consideration of the correspondence between tasks and workstations.

The mixed-model assembly line load balancing problem (MMALLBP) with a given number of workstations is mainly aimed at minimizing the cycle time and reducing the idle time of workstations. Decker et al. (1993) aim to minimize the working time of different products in the workstation and the average working time of processing different products, to balance the load of the MMAL. Chen et al. (2018) adopted tabu search based genetic algorithm to solve the balance problem with the objectives of minimizing the load imbalance rate between workstations, the fluctuation of assembly time within workstations, the number of workstations, obtaining the optimal workstation allocation scheme and production scheduling sequence. Yi et al. (2017) studied the workstation allocation model, taking into account the goals of minimizing the total assembly time, equalizing assembly time between workstations, smoothing assembly time within workstations, etc. Li et al. (2019) set up a multi-objective assembly line smoothing optimization model with the smooth coefficient of workstation and cycle time as targets, considering the assembly complexity, and adopted genetic algorithm to solve it.

At present, there are some studies on the load balancing problem with a given number of workstations. Due to the complex mathematical model structure of the problem, scholars mainly design heuristic methods to solve it.

The above research aims at minimizing workstation idle time and balancing workstation load from two perspectives. Some scholars study the design scheme of assembly lines from the Angle of improving the flexibility of the assembly line and balancing the load between workstations.

One of the effective ways to improve the flexibility of the assembly line is to introduce parallel tasks, allowing the same task to be assigned to different workstations. If any task can be assigned to any workstation, the flexibility of MMAL will reach the maximum, the solution space will increase, and increasing the difficulty of solving it too (Li et al., 2018).

In the research related to parallel tasks, Mosadegh et al. (2012) contemplated the feasibility of executing the same task associated with a product within a mixed-model assembly line at diverse workstations in a Manufacturing Planning and Control System (MPS). Their objective was to optimize the utilization of workstation time, diminish the overall task duration, and equitably distribute the workload across workstations. Anuar and Bukchin (2006) implemented a strategy wherein the same task could be concurrently processed at multiple workstations. They dynamically adjusted assembly tasks among stations during the operation of the assembly line, thereby reducing the cycle time and enhancing production efficiency. Guo et al. (2008) introduced concurrent task execution to minimize total idle time. Initially, they identified shared tasks and the allocation ratios of tasks among distinct workstations, followed by the establishment of task allocation rules. While these aforementioned studies achieved workload balance through the incorporation of parallel tasks, they did not account for the task configuration cost associated with such parallelization. Moreover, the intricate relationship between tasks and workstations remained unexplored in their analyses.

The incorporation of parallel tasks into assembly lines is inevitably associated with an escalation in equipment configuration costs, encompassing expenditures related to equipment acquisition and workstation setup. Furthermore, the expenses incurred due to identical task configurations across distinct workstations, such as processing time and raw material transportation time, may exhibit variations. Cakir et al. (2011) systematically examined the essential equipment purchase costs for task allocation

to workstations, aiming to enhance the smoothness of workstation loads and curtail the total cost of the assembly line. Their research primarily focused on the single-product assembly line balancing problem. Niroomand (2021) factored in both task allocation costs and workstation activation costs in their analytical framework. Salehi et al. (2020) endeavored to minimize the aggregate costs, including equipment configuration, workstation activation fees, and worker wages, by devising a hybrid simulated annealing algorithm tailored to address the intricacies of the assembly line balancing problem. Li et al. (2021) found the Pareto solution by bee swarm algorithm in the trade-off between assembly line cost and efficiency. Chen et al. (2018) undertook an exploration of the task and personnel allocation quandary in the context of parallel workstations, formulating a comprehensive mixed-integer programming model and employing heuristic algorithms and genetic algorithms for its resolution. Pearce et al. (2019) considered workstation characteristics, imposed constraints on task allocation, and developed a multi-constraint integer programming model specifically designed for addressing complex real-world assembly line balancing problems. Hazır and Dolgui (2013) established a mathematical model for assembly line balancing design problems, accounting for assembly tool constraints, and subsequently employed a solver for its resolution.

Although the above studies have considered the equipment configuration costs generated by setting up parallel tasks, most of them have aimed at minimizing the total cost or maximizing the efficiency of AL, few studies have considered load balancing between workstations, task-workstation correspondence, workstation space limitation and task setting costs.

One hand, allowing parallel tasks can increase the flexibility of AL and improve the load balance between workstations. On the other hand, setting multiple parallel tasks requires the configuration of additional equipment and parts, so additional costs will be generated. How to achieve a balance between improving the flexibility of AL and reducing the total cost is one of the problems urgently that the manufacturing industry needs to solve.

In this paper, the problem of assembly line design and load balancing allowing parallel tasks is studied. Considering workstation activating cost, task configuration cost and load difference penalty cost between workstations, the optimal assignment scheme among workstations and the actual processing scheme of each task of each product are found. The task configuration cost refers to the cost incurred when a task is assigned to a workstation and the equipment required to complete the task is installed on the workstation.

The main contributions of this paper are as follows:

(1) The load balancing problem of an assembly line with parallel tasks is investigated, where parallel tasks are introduced and multiple factors such as the inter-correspondence between tasks and workstations as well as the spatial constraints of workstations are considered;

(2) A mixed-integer planning model with the objective of cost minimization is constructed, which includes the fixed cost of workstation activation, the cost of process configuration, and the penalty cost of load unevenness;

(3) A simulated annealing algorithm with an improved strategy is proposed and its effectiveness is verified by case studies. Finally, the solution results under the two strategies are analyzed in depth and relevant management insights are provided.

3. Problem definition and mathematical formulation

3.1 Problem Description and Analysis

3.1.1 Problem Description

The typical assumption in MMAL design is that identical workstations will process the same or similar tasks across multiple products. An increase in the number or types of products could lead to uneven distribution of workload across different workstations, if each workstation is to complete the same or similar tasks in case of relying on a single workstation. In addition, the low utilization of the AL may be caused by the accumulation of products in front of the bottleneck point and the frequent change of product assembly modes at a single workstation, ultimately reducing production efficiency. Permitting the same task to be allocated for parallel performance on different workstations, challenges this assumption, and creates an opportunity to create a more flexible layout of AL. This approach will enable the efficient utilization of workstation cycle time, and ultimately, contribute to reduce or eliminate situations mentioned above.

The problem can be defined as follows: given a cycle time C, product types and their respective demand, the tasks i and their priority relation are known. The set of successor tasks S(i), the number of available workstations |J|, and the maximum number of tasks N that can be assigned to each workstation are also predetermined. Since the processing equipment required for each task is different, if a task is assigned to a workstation, the equipment required for the task must be installed at that workstation. Assuming that the same task of each product can only be completed in one workstation, and the same task of different products can be completed in different workstations, how to assign the task to the workstation and determine which workstation each task of each product is processed in, so that the load of each workstation can be balanced as much as possible and the total

cost of design and operation of the AL can be minimized.

For example, two products A and B both contain 10 task, and the task relationship of the two products is known (as shown in Fig 1(a)(b)). Suppose that the demand ratio of the two products is 0.6 and 0.4 respectively, and the cycle time is 9.

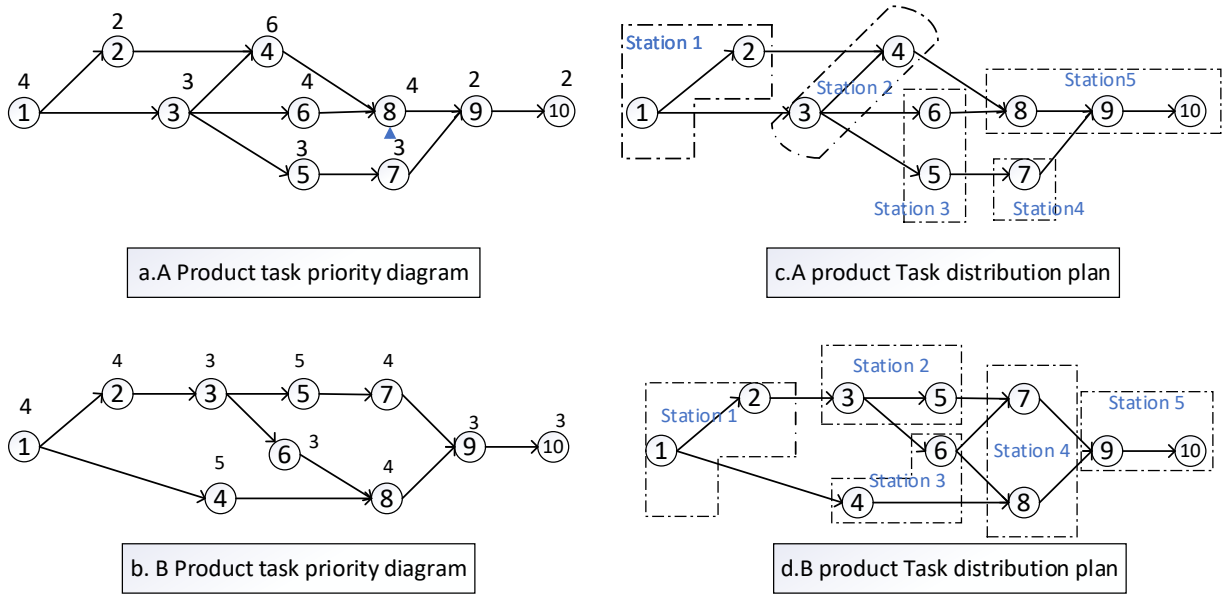

Fig. 1. Feasible scheme of parallel tasks assembly line

If each task can only be assigned to one workstation, the distribution results among workstations and the load of each workstation are shown in Table 1, where the difference between the maximum load and the minimum load is 4.6. If a task is allowed to be assigned to multiple workstations, Table 2 shows a feasible task assignment scheme, in which processes 4, 5 and 8 are assigned to two workstations respectively. Based on this task allocation scheme, the load of each workstation can be calculated according to the actual processing schemes of the two products described in Fig 1(c)(d), and the difference between the maximum load and the minimum load is reduced to 3.6.

Table 1
Allocation results without duplicate tasks

Station	1	2	3	4	5
Assigned task	1，2	3，4	6，8	5，7	9，10
Workload	6.8	8.6	7.6	7.2	4.8

Comparison of the allocation schemes in the two cases indicates that the distribution of workstation loads is more balanced in the feasible scheme which allows multiple workstations to be allocated for a task, as compared to the optimal scheme that prohibits sharing, thus improving the assembly line's productivity.

Table 2
Allocation results with duplicate tasks

Station	1	2	3	4	5
Assigned task	1，2	3，4，5	4，5，6	7，8	8，9，10
Workload	6.8	8.6	7.4	5	7.2

This paper comprehensively considers the MMALLBP under parallel task, taking into account the objectives of parallel task strategy, workstation capacity constraint and assembly line balancing. The mixed integer programming model is established to address the problem, and a simulated annealing algorithm with improved strategy is designed.

3.1.2 Problem Analysis

The optimization objective for the MMALBP with parallel tasks is to minimize the total cost, which includes the fixed cost of activating workstations, the task configuration cost and the penalty cost of uneven loads between workstations. The main constraints include task sequencing constraints to satisfy different products, and constraints on the maximum number of tasks that can be handled by each workstation.

To simplify the issue, this article assumes the following:

(1) Workstations are homogeneous, and each task can be allocated to any of them.
(2) The processing time for each task is contingent upon the specific product being manufactured, rather than the assigned workstation.
(3) Tasks belonging to the same product must be conducted at a singular workstation, whereas tasks from different products may be executed at separate workstations.

3.2 Mathematical Model

3.2.1 Notation

(1) Sets
S : Set of product types
I : Set of tasks
J : Set of workstations
$S(i)$: i Set of immediate successor task types for task i

(2) Index
s : Index of product category, $s \in S$
i : Index of task index, $i \in I$
j : Index of workstation, $j \in J$

(3) Parameters:
C : Cycle time
F : Fixed cost of workstation activation
M : Any large real number
N : Maximum number of tasks that can be allocated to a workstation
a_{ij} : The equipment configuration cost required for production of task i on workstation j
q_s : The proportion of the demand for product s to the total demand for product
t_{is} : Task time required for task i of product s
β : Penalty coefficient for unbalanced maximum workload difference

(4) Decision variables
$x_{ij} = \begin{cases} 1, & \text{The equipment required for task } i \text{ is allocated to workstation } j \\ 0, & \text{Otherwise} \end{cases}$
$w_j = \begin{cases} 1, & \text{Activate Workstation } j \\ 0, & \text{Otherwise} \end{cases}$
$y_{ijs} = \begin{cases} 1, & \text{The task } i \text{ of product } s \text{ in scene } d \text{ is assigned to workstation } j \\ 0, & \text{Otherwise} \end{cases}$
$T_{\min}$: The minimum workload of the workstation
$T_{\max}$: The maximum workload of the workstation

The MMALLBP can be expressed as the following mixed integer programming model.

$$\min Z = \sum_{i=1}^{I}\sum_{j=1}^{J} a_i x_{ij} + F * \sum_{j=1}^{J} w_j + \beta(T_{\max} - T_{\min}) \tag{1}$$

$$\sum_{j=1}^{J} x_{ij} \geq 1, \qquad \forall i \tag{2}$$

$$\sum_{i=1}^{I} x_{ij} \leq N, \qquad \forall j \tag{3}$$

$$y_{ijs} \leq x_{ij}, \qquad \forall i, j, s \tag{4}$$

$$\sum_{j=1}^{J} jy_{ijs} \le \sum_{j=1}^{J} jy_{ajs}, \qquad \forall s, a \in S(i) \tag{5}$$

$$\sum_{j=1}^{J} y_{ijs} = 1, \qquad \forall s, i \tag{6}$$

$$x_{ij} \le w_j, \qquad \forall i, j \tag{7}$$

$$\sum_{i=1}^{I} t_{is} y_{ijs} \le C, \qquad \forall j, s \tag{8}$$

$$\sum_{s=1}^{S} q_s \sum_{i=1}^{I} t_{is} y_{ijs} \le T_{\max}, \qquad \forall j \tag{9}$$

$$\sum_{s=1}^{S} q_s \sum_{i=1}^{I} t_{is} y_{ijs} \ge T_{\min} - M*(1-w_j), \qquad \forall j \tag{10}$$

$$T_{\max} \ge 0, T_{\min} \ge 0 \tag{11}$$

$$x_{ij}, y_{ijs}, w_j \in \{0,1\}, \qquad \forall i, j, s \tag{12}$$

The objective function (1) aims to minimize the total cost, comprising equipment configuration costs, workstation activation costs (the first and second terms, respectively), and penalty costs for workload imbalances among workstations (the third term). Constraints (2)-(12) specify: (2) each task must be assigned to at least one workstation; (3) the maximum tasks that can be assigned to each workstation must not exceed N; (4) a workstation can process a task only if allocated the requisite equipment； (5) tasks must follow a specified processing sequence; (6) tasks for the same product are processed at the same workstation; (7) if a task is assigned to a workstation, the workstation must be activated; (8) workstation processing time cannot exceed the cycle time; (9)-(10) the average workload of activated workstations must fall within specified maximum and minimum values; and (11)-(12) decision variable values are constrained.

3.3 Model Validation

To verify the correctness of the model, we employed the example illustrated in Fig 1. There are a total of two products, and the relationship between the priority of tasks and their respective process time is illustrated in Fig 1. Suppose that the activated cost of each workstation is $F = 10$, and the equipment configuration cost required by each process in each workstation is shown in Table 3.

Table 3
Setting costs of each tasks.

	Task									
Station	1	2	3	4	5	6	7	8	9	10
1	2	1	1	2	1	2	2	2	1	2
2	1	1	2	2	2	1	1	2	1	1
3	2	2	2	1	2	1	2	1	2	1
4	2	1	2	2	2	2	2	2	1	2
5	1	1	1	2	1	1	1	2	2	2

After substituting the data from Fig 1 and Table 3 into the model, Gurobi was used to derive the optimal solution which is displayed in Table 4. The minimum value of the objective function was determined to be 72. When parallel tasks were allowed, task 8 was assigned to workstation 3 and workstation 5, and the maximum load of the workstation was reduced from 8.6 to 7.2 compared to the solution in Table 1 which did not allow parallel tasks. The minimum load was increased from 5 to 6.6, and there was a significant reduction in process load difference from 3.6 to 0.6.

Through comparative analysis, it can be seen that using the optimal solution obtained by the model in this paper, the workstation load range is reduced by 83.3%, which indicates that using the model in this paper to design the AL and arrange the product processing plan can reduce the design and operation cost of the AL, reduce the waiting time and the idle time of the station during the product processing, and improve the productivity of AL.

The classical assembly line load balancing problem requires that a task can only be assigned to a single workstation, and the classical assembly line load balancing problem is a NP-hard problem.. The MMALLBP, examined in this paper, involves allowing one task to be assigned to several workstations - an increased level of complexity compared to the classical assembly

line load balancing problem. To obtain an approximate optimal solution rapidly, we devised a refined simulated annealing algorithm for the problem of balancing the load on mixed-flow assembly lines in parallel tasks.

Table 4
Optimal allocation scheme

Station	1	2	3	4	5
Assigned task	1，2	3，6	4，8	5，7	8，9，10
Workload	6.8	6.6	7.2	7.2	7.2

4. Simulated annealing algorithm with improved strategy

Simulated annealing algorithm(SA) is a local search algorithm based on neighborhood, which can be used to solve combinatorial optimization problems. In this section, a simulated annealing algorithm with improved strategy(ISA) is designed to solve the MMALLBP. Aiming at the defect that SA is easy to fall into the local optimal solution, three kinds of improvement operators are introduced to expand the search range, jump out of the local optimal solution, and improve the quality of the solution.

4.1 Encoding and decoding

The coding of the MMALLBP under parallel task is a random number sequence between 0 and 1 with length |I|. Each position of the sequence corresponds to a task, and the random number represents the weight of the task. In the decoding procedure, the tasks are sorted initially by their weights from highest to lowest. For every product, the highest weighted task is assigned to the relevant station based on priority, ensuring that the task sequence constraints are met. This process is repeated for each product until all tasks have been assigned. Each code can be decoded into exactly one feasible task assignment scheme, since the sequential constraints of the product processes have been considered in the decoding process.

Random weight sequence	0.64	0.52	0.82	0.68	0.27	0.2	0.36	0.01	0.48	0.03
Task	1	2	3	4	5	6	7	8	9	10
Weight sorting	3	4	1	2	7	8	6	10	5	9

Decoding ↓ Encoding ↑

Station	1	2	3	4	5
Product 1 task distribution	1, 2	3,4	5,6	7	8,9,10
Product 2 task distribution	1,2	3,5	4,6	7,8	9,10

Fig. 2. Schematic diagram of encoding and decoding

Taking the 10 tasks of two products in Fig. 1 as an example, a coding sequence of length 10 is randomly generated (as shown in Fig 2). According to the encoding sequence, the decoding steps are as follows:

Step 1: Order the tasks by weight from greatest to smallest. Puts all tasks into the unscheduled tasks list.

Step 2: For each product, from the list of unscheduled tasks, select the task whose immediate preceding tasks have been arranged (or no immediate preceding tasks) and the task with the greatest weight, and arrange a workstation with the smallest serial number that meets the constraints of task and workstation capacity (allowing the same tasks of different products to be arranged in different workstations) for this task. Remove the already scheduled task from the list and record the results of the scheduled tasks for each product in the solution matrix. Calculate the cumulative load of each workstation based on the tasks already scheduled for each workstation.

Repeat Step 2 until all tasks for all products have been scheduled to the workstation.

Step 3: Calculate the number of workstations opened according to the task arrangement results, and calculate the objective function value by Eq. (1).

The pseudo-code for the decoding process is as follows:

Decode procedure：

Input: Weight sequence, Product set S, Task set I, Assignable workstation set J_i, Set of tasks without immediate preceding tasks N_i, Set of immediate successor task types for task i F_i, Workstation w availability time T_w, Task i processing time t_i

Output: Task assignment result

1： ***While*** $|S| \neq 0$ ：

While $|I| \neq 0$:
  Select the task i with the most weight in N_i
  For j in J_i :
  If $T_w \geq t_i$
    Assign task i to the workstation j , $T_w = T_w - t_i$
    Update J_{F_i} of the element in F_i
  Else
    Skip this workstation
  End If
  Update task set (delete scheduled task) I
End While
Update product set (remove scheduled products) S
End While

4.2 Neighborhood transformation

The product task assignment results are influenced by the order of task weights, and varying weight sequences produce distinct achievable solutions. This study utilizes two approaches to generate neighborhood solutions. Firstly, two points swap positions, where two points in the weight sequence are randomly chosen for exchange, as displayed in Fig 3(a). Secondly, sequence swap is used. Randomly select a point in the sequence and exchange the fragments around that point, as illustrated in Fig 3(b).

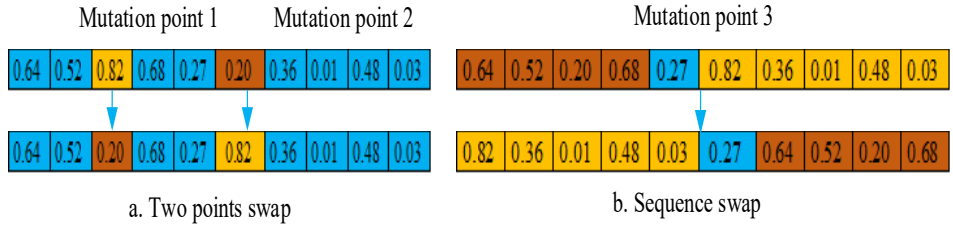

Fig. 3. Schematic diagram of neighborhood transformation

4.3 Improvement operator

To improve the quality of the feasible solution and speed up the convergence speed. In this paper, three improvement operators are designed to improve the generated feasible solutions while satisfying the product production process constraints and workstation cycle time constraints.

(1) The delete operator, which selects a parallel task assigned to more than one workstation and deletes the task from one of these workstations;
(2) The maximum load move operator, which selects the workstation with the largest load and moves a task from that workstation to another workstation;
(3) The minimum load move operator, which selects the workstation with the lowest load and moves a task from another workstation to that workstation.

For the feasible solution given in Fig 4(a), workstation 2 has the maximum load and workstation 4 has the minimum load. Assuming $F = 10$, $a_{ij} = 3$, $\beta = 10$,and substituting into the objective function (1) can be calculated as $Z_a = 125$. Fig 4(b)-(d) are obtained after performing three different operations on this feasible solution. Fig. 4(b) represents that the parallel task 8 of product a is removed from workstation 5; Fig. 4(c) represents that the task 3 of product a is moved from workstation 2, which has the highest load, to workstation 1; and Fig 4(d) represents that the task 5 of product a is moved from workstation 3 to workstation 4, which has the lowest load.

The value of the objective function after executing the deletion operator is $Z_b = 124$; executing the maximum load shift operator is $Z_c = 125$; executing the minimum load shift operator is $Z_d = 116$. It is evident that by adjusting all three improvement operators, better feasible solutions can be achieved.

Workstation	1	2	3	4	5
Assigned task	1,2	3,4,5	4,5,6	7,8	8,9,10
Workload	6.8	8.6	7.4	5	7.2

a. Original feasible solution

Workstation	1	2	3	4	5
Assigned task	1,2	3,4,5	4,5,6	7,8	9,10
Workload	6.8	8.6	7.4	7.4	4.8

b. The delete operator

Workstation	1	2	3	4	5
Assigned task	1,2,3	3,4,5	4,5,6	7,8	8,9,10
Workload	8.6	6.8	7.4	5	7.2

c. The maximum load move operator

Workstation	1	2	3	4	5
Assigned task	1,2	3,4,5	4,6	5,7,8	9,10
Workload	6.8	8.6	5.6	6.8	7.2

d. The minimum load move operator

Fig. 4. Schematic diagram of operation results of three improved operators

4.4 Algorithmic pseudo-code

Based on the above operations, a simulated annealing algorithm with improved strategy is designed. The pseudo-code of the algorithm is as follows:

A simulated annealing algorithm with improved strategy

Input: Initial temperature T_0, Final temperature T_f, Cooling Rate λ, Maximum number of iterations R、Initial weight sequence

Output: Optimal solution π_b and the corresponding objective function value $z(\pi_b)$

Generate initial solution π， Compute $z(\pi)$, $\pi_c = \pi, z(\pi_c) = z(\pi)$

Initialization： $\pi_b = \pi, z(\pi_b) = z(\pi)$， $T = T_0$

While $T \geq T_f$ or Reach terminal time

  While $r \leq R$:

    If $rand \leq p$ (p random generation)

    Obtaining a neighborhood solution by Performing Two-point swap operation π'

    else

    Obtaining a neighborhood solution by Performing Sequence swap operation π'

    End If

    Improved neighborhood solution π' : randomly select an improved operator to get a solution π''

    If $z(\pi'') < z(\pi_c)$

      $\pi_c = \pi'', z(\pi_c) = z(\pi'')$

    else

      Accept the new solution with a probability of $\exp(-(z(\pi_c) - z(\pi''))/T)$

    End If

    If $z(\pi_c) < z(\pi_b)$

      $\pi_b = \pi_c, z(\pi_b) = z(\pi_c)$

    End If

    $\tau = \tau \cdot \lambda$

  End While

End While

5. Numerical experiments

To examine the effectiveness of the algorithm, this section presents simulated examples based on the fundamental data in literature (Otto et al., 1964). Initially, the commercial solver Gurobi and the improved simulated annealing algorithm are used to solve the problem and the performance of the algorithm is analyzed. Then, the MMALLBP's results are compared and analyzed when parallel tasks are allowed and when they are not, to verify the superiority of allowing them.

5.1 Example Generation and Parameter Settings

Based on the fundamental data derived from literature. a tasks precedence graph is generated. The number of tasks $|I|$ and product types $|S|$ are set as parameters, with task numbers $|I|$ = 10, 15, 20, 25. Process times t_i are randomly generated from a uniform discrete distribution $U[3,7]$, and the cost of configuring tasks a_{ij} at the workstation is randomly generated from a

uniform discrete distribution $U[1,5]$. When $|I|$ = 10, the number of product types is set to $|S|$ = 2, 3, 4; for the remaining values of $|I|$, the number of product types $|S|$ is set to 2 and 3, generating a total of 9 sets of test cases.

The maximum number of tasks that can be arranged at each workstation is set to N = 3, and the cycle time is C = 9. The fixed cost of opening the workstation is F = 10, and the penalty coefficient for uneven load is $\beta = 10$. The demand ratio for two products is 0.58 and 0.42, and the demand ratio for three products is 0.4, 0.3, and 0.3, and the demand ratio for four products is 0.2, 0.3, and 0.3, and 0.2.

For each set of parameters, 15 examples are generated. For each example, the integer programming model is solved using Gurobi 9.0.2 to obtain an exact optimal solution, and an improved simulated annealing algorithm is used to find an approximate optimal solution. The average solution time and objective function value for each example under each set of parameters are recorded.

5.2 Simulation Calculation and Settlement Analysis

The given parameters for the improved simulated annealing algorithm are as follows: initial temperature $T_0 = 100$, termination temperature $T_f = 0.01$, cooling rate $\lambda = 0.9$, and maximum iteration number $R = 5\times|I|\times|S|$ (the number of iterations is related to the size of the example, and the larger the size of the example, the more internal iterations are required.)

For each example, both the Gurobi 9.0.2 solver and the improved simulated annealing algorithm were used for calculation using the Python programming language on a laptop with an Intel Core i5-6200U CPU @ 2.30GHz. As the size of the example increases, the solution time of the solver increases exponentially. In this paper, the maximum running time of the solver is set to 3600s.

The results obtained directly using the Gurobi solver and using the algorithm proposed in this paper are shown in Table 5. In terms of solution time, both methods increase with the size of the example, and as the number of product types increases for the same task size, the solution time also increases. However, for all examples, the solution time of the solver is much longer than that of the improved simulated annealing algorithm. When the number of tasks is greater than 20 and the number of products is greater than 3, for some examples the solver fails to obtain an optimal solution within 3600s (*the third column in Table 5 lists the number of examples where the solver fails to obtain an optimal solution within 3600s), while the improved simulated annealing algorithm takes no more than 10s.

Compared with the objective function values, it can be found that for small-scale examples with 10 tasks (Rows 1-3), the improved simulated annealing algorithm can quickly obtain an approximate optimal solution with a GAP of less than 2% from the optimal solution, and for medium-sized and large-scale examples, it can obtain a feasible solution with a GAP of less than 10% from the optimal solution. For examples with the same task size (e.g., examples with 10 tasks), increasing the number of product types increases the GAP between the approximate solution and the optimal solution, but it is still within 10%.

In summary, for small-scale examples, the improved simulated annealing algorithm can quickly obtain an accurate optimal solution or an approximate optimal solution. As the problem size increases, the speed advantage becomes more apparent and the gap between the approximate optimal solution and the exact optimal solution is within an acceptable range.

Table 5
Solution results of the test instances

I-S	The Gurobi Solver results				The Improved Simulated Annealing Algorithm Solution results				GAP (%)
	CPU time(s)		objective value		CPU time(s)		objective value		
	Average	Maximum	Average	Maximum	Average	Maximum	Average	Maximum	
10-2	2.28	3.07	93.06	95	0.808	0.97	93.46	95	0.43
10-3	5.75	5.88	102.64	108	2.23	3.51	103.36	110	0.71
10-4	9.89	19.12	108.40	113	3.10	3.39	110.10	116	1.57
15-2	28.58	48.13	154.78	159.6	1.53	1.74	162.60	176	5.05
15-3	107.07	304.36	139.52	148	3.36	3.73	149.09	154	6.86
20-2	272.01	881.53	201.66	228	2.81	3.79	214.20	237	6.22
20-3	1695.68	3600(1)*	195.69	204	5.49	5.70	209.69	216	7.15
25-2	2925.27	3600(5)	239.36	249	3.82	4.25	260.99	270	9.04
25-3	3406.98	3600(9)	218.00	238	8.89	9.55	239.00	247	9.63

To analyze the stability of the algorithm, the solution times for different-sized examples were normalized $I_t = (t - t_{\min}) / (t_{\max} - t_{\min})$ and a box plot was generated to compare the dispersion of the solution times. As shown in Fig 5, the solution time of the Gurobi solver is unstable across various example sizes, while the solution time of the improved simulated annealing algorithm is relatively stable.

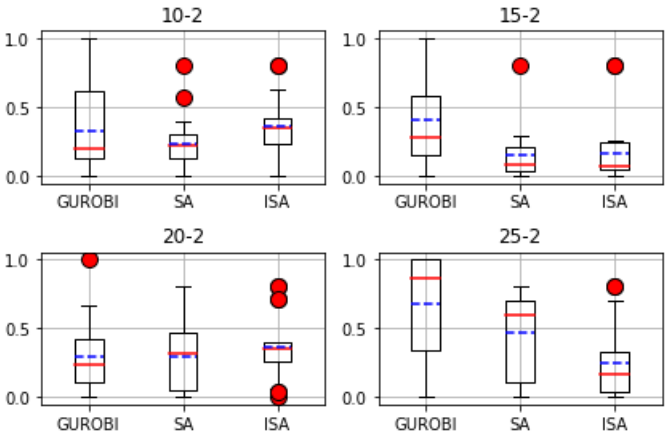

Fig. 5. Comparison of solving time

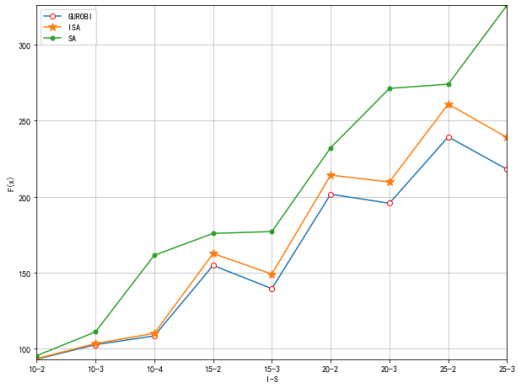

Fig. 6. Comparison of solution results

To verify the effectiveness of the improved strategy, examples of different sizes were solved using both the improved simulated annealing algorithm and the classical simulated annealing algorithm. The results are shown in Fig 6, which indicate that the computational results of the improved simulated annealing algorithm are significantly better than those of the classical simulated annealing algorithm. This indicates that the improved strategy can help to avoid local optima by jumping out of local optimal solutions.

5.3 Comparison of strategies for allowing and not allowing parallel tasks

To objectively evaluate the advantages of allowing parallel tasks, this section compares and analyses the mixed-model assembly line load balancing under the two strategies of allowing and not allowing parallel tasks. In the strategy of not allowing parallel tasks, the constraint (2) in the model should be an equality.

For the 15 examples generated by each set of parameters in Section 3.2, the integer programming model is solved using the Gurobi solver under both strategies to obtain the results. Fig 7 shows the comparison results.

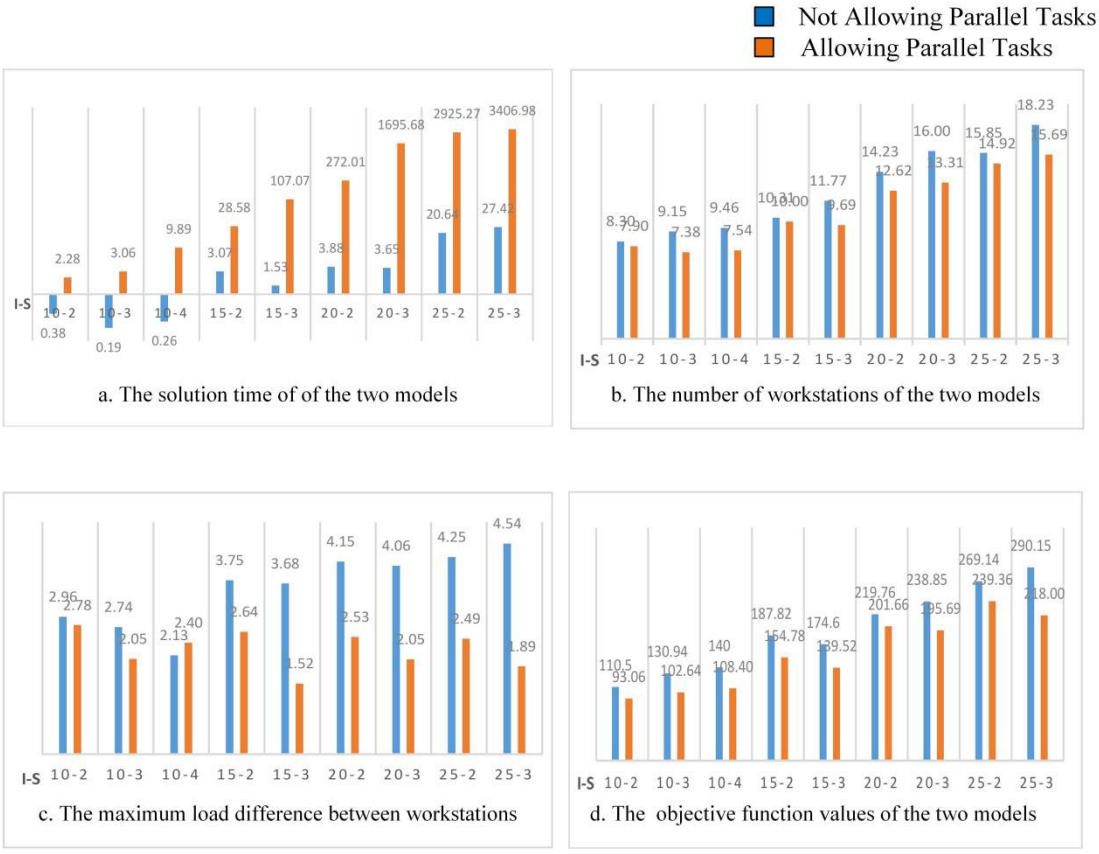

Fig. 7. Comparison of solution results with those of parallel tasks not allowed

Fig 7(a) demonstrates the solution time of the two models. It can be seen that the average solution time of the model without

allowing parallel tasks is less than 30 seconds, which is significantly faster than the solution time of the model with allowing parallel tasks. This is because the feasible region of the model without allowing parallel tasks is smaller, which indicates from another perspective that the strategy of allowing parallel tasks greatly increases the difficulty of the problem, so it is necessary to design fast and effective solution algorithms.

Fig 7(b) shows the number of workstations that need to be activated under the two strategies. Under the strategy of not allowing parallel tasks, more workstations need to be activated, and as the problem scale increases, the difference in the number of workstations activated between the two strategies becomes larger, indicating that allowing parallel tasks can better improve workstation utilization.

Fig 7(c) demonstrates the maximum load difference between workstations under the two strategies. For all examples, the maximum load difference between workstations under the strategy of not allowing parallel tasks is greater than that under the strategy of allowing parallel tasks, indicating that allowing parallel tasks can make workstation load more balanced, and as the problem scale increases, the advantage of allowing parallel tasks in load balancing becomes more significant.

Fig 7(d) shows the objective function values of the two models. It can be found that for all examples, the optimal solution of the model without allowing parallel tasks is inferior to that of the model with allowing parallel tasks. As the number of products increases, the GAP between the optimal values of the two models becomes larger, for example, for examples with 10 processes, as the number of products increases, the GAP values between the optimal solutions obtained under the two strategies are respectively from 18.74% to 27.58% and to 29.15%. This is because as the number of products increases, if parallel tasks are not allowed, each process has limited space for allocation, so more workstations need to be activated. At the same time, since workstation load differences cannot be eliminated by setting parallel tasks, both the fixed costs for activated workstations and penalties for load differences between workstations increase, resulting in increasingly large differences in objective function values between models without allowing and with allowing parallel tasks.

From the above comparative analysis, it can be seen that in the mixed-model assembly line load balancing problem, allowing parallel tasks can not only reduce the number of workstations turned on but also make workstation load more balanced. Therefore, allowing parallel tasks is an important way to improve the production efficiency of mixed-model assembly lines and reduce production costs.

When manufacturing companies carry out workshop layout, they can not only consider it from a balance perspective, but also set up parallel tasks for bottleneck and trouble-prone tasks to improve productivity and assembly line flexibility.

6. Conclusion

In this paper, we conducted research on the mixed-model assembly line load balancing problem. Under the strategy of allowing parallel tasks, we established a mixed integer programming model with the goal of minimizing the sum of fixed costs for turning on workstations, task configuration costs, and penalties for load imbalance between workstations. Considering the constraint of the number of tasks that each workstation can handle, we designed a simulated annealing algorithm with an improved strategy to solve the model.

Through simulated calculations of different-sized examples, we compared the solutions obtained using the simulated annealing algorithm with the improved strategy and the Gurobi solver to solve the mixed integer programming model, and analyzed the comprehensive effects of the proposed algorithm in terms of solution time and accuracy, verifying the fast and effective algorithm.

By comparing the results of solving the mixed-model assembly line load balancing problem under the strategies of allowing and not allowing parallel tasks, we analyzed the advantages of allowing parallel tasks in designing mixed-model assembly lines. Although allowing parallel tasks requires an increase in task configuration costs, this strategy can make the load distribution more balanced among workstations, reduce the number of workstations activated, and improve assembly line production efficiency.
We extended the traditional assembly line where each process can only be assigned to one workstation to allow one task to be assigned to multiple workstations. The research results showed that adopting the parallel task strategy can effectively reduce the number of workstations activated and reduce load differences between workstations. Therefore, the parallel task strategy is an effective method for improving the flexibility of mixed-model assembly lines.

Our assumption was that the types of products to be produced and their proportions were known with certainty in advance. However, in practice, due to long-term decision-making in assembly line design, the types of products produced or their proportion may frequently change every day. How to consider changes in product types and their proportion and design flexible mixed-model assembly lines that meet different requirements is a problem that can be further studied in the next step. In addition, although we established a mixed integer programming model for the load balancing problem in mixed-model assembly lines, we did not design an exact algorithm specifically for the model. In the future, we can combine the model

structure to study accurate algorithms for solving the model or designing more intelligent algorithms to improve solution efficiency and accuracy.

Acknowledgements

This research was jointly supported by the National Natural Science Foundation of China (NSFC) under Project No. 71771028, Beijing Natural Science Foundation under Project No. 9212004 and No. Z180005, Beijing Wuzi University Youth Scientific Research Fund (2024XJQN07) and Beijing Wuzi University Scientific Research Fund (2024XJKY33).

References

Anuar, R., & Bukchin, Y. (2006). Design and operation of dynamic assembly lines using work-sharing. *International Journal of Production Research, 44*(18-19), 4043-4065.

Cakir, B., Altiparmak, F., & Dengiz, B. (2011). Multi-objective optimization of a stochastic assembly line balancing: A hybrid simulated annealing algorithm. *Computers & Industrial Engineering, 60*(3), 376-384.

Chen, Y. Y., Cheng, C. Y., & Li, J. Y.(2018) Resource-constrained Assembly Line Balancing Problems with Multi-Manned Workstations. *Journal of Manufacturing Systems, 48*, 107-119.

Chen, Y.（2019） Optimization and Simulation Research on Balancing and Sequencing of Mixed-flow Assembly Line. Wuhan University of Technology.

Çil, Z. A., Li, Z., Mete, S., & Özceylan, E. (2020). Mathematical model and bee algorithms for mixed-model assembly line balancing problem with physical human–robot collaboration. *Applied soft computing, 93*, 106394.

Decker, M. (1993). Capacity smoothing and sequencing for mixed-model lines. *International Journal of Production Economics, 30*, 31-42.

Guo, Z. X., Wong, W. K., Leung, S. Y. S., Fan, J. T., & Chan, S. F. (2008). A genetic-algorithm-based optimization model for solving the flexible assembly line balancing problem with work sharing and workstation revisiting. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews), 38*(2), 218-228.

Gutjahr, A. L., & Nemhauser, G. L. (1964). An algorithm for the line balancing problem. *Management science*, *11*(2), 308-315.

Hazır, Ö., & Dolgui, A. (2013). Assembly line balancing under uncertainty: Robust optimization models and exact solution method. *Computers & Industrial Engineering, 65*(2), 261-267.

Hou, L., Wu, Y. M., Lai, R. S., & Tsai, C. T. (2014). Product family assembly line balancing based on an improved genetic algorithm. *The International Journal of Advanced Manufacturing Technology, 70*, 1775-1786.

Li, A. P., Zhao, Y. X., & Zhang, J. H. (2019). Multi-objective Assembly Line Balancing Optimization Method Considering Complexity of Assembly Relationship. *Computer Integrated Manufacturing Systems, 25*(07), 1665-1675.

Li, J. L. (2018) Design Mixd Model Assembly Lines with Adjustable Production Capacity. Xi'an Jiaotong University.

Li, Z. P., Shi, Y., & Wu, L. Y. (2022) Research on Mixed Flow Line Balancing and Scheduling Optimization with Multiple Constraints. *Journal of System Simulation,* 1-14.

Li, Z., Janardhanan, M. N., & Ponnambalam, S. G. (2021). Cost-oriented robotic assembly line balancing problem with setup times: multi-objective algorithms. *Journal of Intelligent Manufacturing, 32*, 989-1007.

Mei, Z. H. A. N. G., Yanxia, F. U., Jinhui, Z. H. U., & Shuaihang, D. E. N. G. (2024). Complex multi-manned assembly line balancing using improved water wave algorithm. *Computer Integrated Manufacturing System*, *30*(1), 129.

Mosadegh, H., Ghomi, S. F., & Zandieh, M. (2012). Simultaneous solving of balancing and sequencing problems in mixed-model assembly line systems. *International Journal of Production Research, 50*(18), 4994-5016.

Niroomand, S. (2021). Hybrid artificial electric field algorithm for assembly line balancing problem with equipment model selection possibility. *Knowledge-Based Systems, 219*, 106905.

Otto, A., Otto, C., & Scholl, A. (2013). Systematic data generation and test design for solution algorithms on the example of SALBPGen for assembly line balancing. *European Journal of Operational Research, 228*(1), 33-45.

Pearce, B. W., Antani, K., Mears, L., Funk, K., Mayorga, M. E., & Kurz, M. E. (2019). An effective integer program for a general assembly line balancing problem with parallel workers and additional assignment restrictions. *Journal of Manufacturing Systems, 50*, 180-192.

Salehi, M., Maleki, H. R., & Niroomand, S. (2020). Solving a new cost-oriented assembly line balancing problem by classical and hybrid meta-heuristic algorithms. *Neural Computing and Applications, 32*, 8217-8243.

Tang, Q. h., Rao, D., & Li, Z. X. (2018). Improved Iterated Local Search Algorithm for Type I Mixed-Model Two-Sided Assembly Line Balancing Problem[J]. *Computer Integrated Manufacturing Systems, 24*(02): 390-399.

Wang, C. J., & Liu, J. M. (2020). Research on Balance Problem of Mixed-Flow Assembly Line with Operator's Workload. *Journal of Chongqing University of Technology (Natural Science), 34*(07), 100-107.

Yi, Z. F. (2017). Modelling and Algorithm Research for Mixed Model Assembly Line Balancing and Sequencing Optimization. Dalian Maritime University.