PLOS ONE

RESEARCH ARTICLE

# Two hybrid flow shop scheduling lines with assembly stage and compatibility constraints

**Rafael Muñoz-Sánchez[1], Iris Martínez-Salazar[1]*, José Luis González-Velarde[2], Yasmín Á. Ríos Solís[2]**

**1** Facultad de Ingeniería Mecánica y Eléctrica, Universidad Autónoma de Nuevo León, San Nicolás de los Garza, Nuevo León, México, **2** School of Engineering and Science, Tecnológico de Monterrey, Monterrey, Nuevo León, México

* iris.martinezslz@uanl.edu.mx





**Citation:** Muñoz-Sánchez R, Martínez-Salazar I, González-Velarde JL, Ríos Solís YÁ (2024) Two hybrid flow shop scheduling lines with assembly stage and compatibility constraints. PLoS ONE 19(6): e0304119. https://doi.org/10.1371/journal.pone.0304119

**Editor:** Mazyar Ghadiri Nejad, Cyprus International University Faculty of Engineering: Uluslararasi Kibris Universitesi Muhendislik Fakultesi, TURKEY

**Received:** February 6, 2024

**Accepted:** May 7, 2024

**Published:** June 21, 2024



**Data Availability Statement:** All the instances and the detailed results are in https://doi.org/10.6084/m9.figshare.24246547.

**Funding:** The author(s) received no specific funding for this work.

**Competing interests:** The authors have declared that no competing interests exist.

## Abstract

Two hybrid flow shop scheduling lines must be coordinated to assemble batches of terminated products at their last stage. Each product is thus composed of two jobs, each produced in one of the lines. The set of jobs is to be processed in a series of stages to minimize the makespan of the scheduling, but jobs forming a product must arrive at the assembly line simultaneously. We propose a mixed integer linear programming model. Then, based on the model, we propose a pull-matheuristic algorithm. Finally, we present two metaheuristics, a greedy randomized adaptive search procedure and a biased random key genetic algorithm, and compare all the methodologies with real-based instances of a production scheduling problem in the automobile manufacturing industry. The greedy algorithm yields high-quality solutions, while the genetic one offers the best computational times.

## 1 Introduction

The scheduling of flow shops with multiple parallel machines per stage, referred to in the literature as the hybrid flow shop (HFS), flexible flow shop, or multi-processor flow shop [1–3], is a complex combinatorial problem encountered in many real-world manufacturing processes.

In this article, two HFS must be coordinated to assemble batches of terminated products at the end of the lines, as shown in Fig 1. Each product is thus composed of two jobs, each produced in one of the HFS. In both HFS, the set of jobs are to be processed in a series of stages to minimize the time that elapses from the start of jobs to their end, that is, to minimize the maximal completion time or *makespan* of the scheduling. The jobs forming a product must arrive at the assembly line simultaneously, even if one HFS is, on average, faster than the other. Each stage of the HFS lines has several unrelated parallel machines. In each HFS, jobs pass through the stages of the shop floor in a flow sequence, but a job might skip some stages if they are not necessary for its manufacture. Each job $j$ requires a processing time $p_{jk}$ in stage $k$. We name this problem the 2-Hybrid Flow Shop with Assemble stage (2-HFSA).

As is usual in HFS [4], we suppose that all jobs and machines are available at time zero. Machines can process only one job operation at a time, and job operations can be processed by only one machine at a time. Moreover, operation preemption is not allowed.

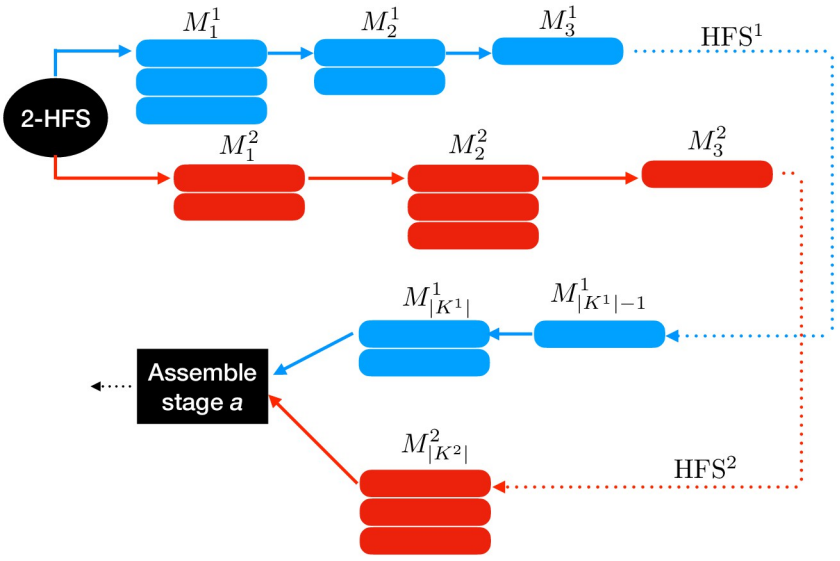


**Fig 1. 2-HSFA: Two HFS must be coordinated to assemble batches of terminated products at the end of the lines.**

https://doi.org/10.1371/journal.pone.0304119.g001

In [5], the shop problems are classified according to their configuration, constraints and assumptions, and objective function. In our case, each HFS denoted as HFS$^1$ and HFS$^2$, can be described as $FHm^l, (RM^{(k)})_{k=1}^{m^l} \mid prmu, M_j, S_{sd} \mid C_{\max}$, for $l = \{1, 2\}$. $FHm^l$ indicates a hybrid flow shop with $m^l$ stages. $(RM^{(k)})_{k=1}^{m^l}$ means that in all stages, $k = 1, \ldots, m^l$, there are any number of unrelated parallel machines. The term *prmu* indicates that the jobs are processed in every stage in the same order. The processing of job $j$ is restricted to the set of machines $M_j$ at stage $k$, known as eligibility. $S_{sd}$ indicates that the setup times depend on the sequence of operations. Finally, $C_{\max}$ is the maximum completion time.

Solving a single of our HFS problems is NP-hard. Indeed, our problem can be reduced to an HFS with two stages, even when one stage contains two machines and the other one a single machine, which is already NP-hard [6]. Notice that even if the objective functions of HFS$^1$ and HFS$^2$ correspond to the minimization of their makespan, a solution to the 2-HFSA problem may have long idle times because of the assembly constraint at the end of the production lines. This is where the main challenge of the 2-HFSA problem resides, in the coordination at the assembly stage that generates idle times.

This research's case study takes place in the automobile parts production system, more precisely, in a tractor-truck parts manufacturer. Their main products are tractor axles, and their essential elements are the crown-pinion assembly. The crown-pinion line's production times are significant and are the company's bottleneck since it is the hardest to optimize. The HFS1 makes the crowns, while the HFS2 produces the pinions. These two pieces form a product. Since the manufacturing process continues with the crown-pinion product, the batches of these two pieces should arrive simultaneously at the assembly stage.

This work focuses on solving the 2-HFSA, which has an assembly stage of two HFS. First, we propose a Mixed Integer Linear Programming model (MILP) to assert the exact assembly time at which both parts should be assembled, together with the machine eligibility and sequence-dependent setup times. The problem is NP-hard, so the size of the instances that can be solved optimally is relatively small. Then, we propose a matheuristic algorithm that takes advantage of our mathematical model. It first solves the sequence, and then the timing is

determined. Still, we propose two metaheuristics to face larger instances: a Greedy Randomized Adaptive Search Procedure (GRASP) and a Biased Random Key Genetic Algorithm (BRKGA). An extensive calibration of the different parameters and operators using experimental designs was executed. To evaluate the proposed algorithms, we conducted several experiments with 20 random instances based on real data from the case study company.

The rest of the paper is organized as follows. A literature review is presented in Section 1. The problem description and its mathematical model are in Section 2.1. Section 2.2 describes the matheuristic, while the metaheuristics algorithm GRASP and BRKGA are in Sections 2.3 and 2.4. Section 3 includes the experimental results, and Section 4 concludes the work.

## 1.1 Literature review

The HFS literature is large, but excellent reviews show the main tendencies in the field [1, 7, 8]. For the $FHm, (RM^{(k)})_{k=1}^{m} \mid\mid C_{\max}$ problem, [9] propose branch-and-bound algorithms. Also, [10, 11] determine lower bounds, while heuristics and metaheuristics are proposed in [12–15]. Moreover, neural networks are presented in [16], and ant-colony algorithms in [17]. The authors include sequence-depending setup times in [18–21], propose mathematical formulations and metaheuristics. For the $FHm, (RM^{(k)})_{k=1}^{m} \mid M_j \mid C_{\max}$ problem, which is more related to ours because of the restricted machines per stage, [22, 23] propose genetic algorithms. Nevertheless, authors do not consider an assembly stage of the coordination of two lines.

Some applications require more specific characteristics. [24] deal with $FHm_l, (RM^{(k)})_{k=1}^{m} \mid reentry, batch, S_{sd} \mid C_{\max}$ problem in a cardboard company where the system required re-entrant flows, external operations, and transfer batches between stations. [25] solve the $FHm_l, (RM^{(k)})_{k=1}^{m} \mid M_j, S_{sd}, buffer \mid C_{\max}$ problem with a genetic algorithm that takes into account the sequence-dependent setup time, availability constraints, and limited buffers. [26] solve the $FHm_l, (RM^{(k)})_{k=1}^{m} \mid M_j, S_{sd} \mid C_{\max}$ problem in the textile industry and the synthetic paint business with mathematical programming and genetic algorithms. In [27], the authors study an agricultural product scheduling problem. They introduces a mixed integer linear programming mathematical model to minimize work in process and maximize average machine cell utilization. However, none of these works considers the coordination of two different lines as the one proposed in this research.

[28] were among the first to consider assembly lines. They solve the $FH2, ((2^{(1)}, P2^{(2)})_{k=1}^{m} \mid assembly^{(2)} \mid \bar{F}$ problem with only two stages and only one of two machines per stage. The processing of the second stage cannot begin until the processing of the first stage is finished. The authors propose a lower bound, a branch-and-bound algorithm, and a simple heuristic algorithm. [29] study an HFS with assembly operations. The parts are produced in a hybrid flow shop, and an assembly stage produces the final products. A hierarchical branch-and-bound algorithm is presented. [30] determine the assembly scheduling and transportation allocation to minimize the waiting times. The main difference with our problem is that [30] consider only one HFS.

[31] present a two-stage HFS problem followed by a single assembly machine as the one we study in this article. As in our case, parts must be processed on the HFS stages and joined in the assembly stage to produce the final product. The authors propose two metaheuristic techniques. [32] propose a hybrid solving method that combines improved extended shifting bottleneck procedure and genetic algorithm for the assembly job shop scheduling problem, which differs from the one we propose here. [33] analyze a production line in an automobile assembly plant, using simulation and dispatching rules, to define a production planning strategy for the company. [34] focus on the green scheduling problem in a flexible job shop system. The

authors formulate a mixed integer linear multiobjective optimization model. Recently, [35] tackle the hybrid flow shop scheduling problem with a heterogeneous graph neural network to learn an optimal scheduling policy.

## 2 Materials and methods

### 2.1 Mixed integer linear programming for the 2-HFSA problem

Let set $J^l$ be the set of jobs manufactured by the HFS$^l$, and $J = J^1 \cup J^2$ be the set of jobs of the 2-HFSA problem. Let $N = \{(i, j) | i \in J^1, j \in J^2\}$ be the set of final products to be assembled. The stages of the 2-HFSA problem are the set $K = K^1 \cup K^2 \cup a$ where $a$ corresponds to the assembly stage (see Fig 1). Similarly, we define the machine set as $M = \cup_{l \in L, k \in K^l} M_k^l$.

The processing time of job $j \in J^l$ in stage $k \in K^l$ on machine $m \in M_k^l$ is $p_{jkm}^l$. Note that every job's processing time in the assembly stage $a$ is null. Thus, $p_{ja}^l = 0$, for $j \in J$. Jobs pass through the stages of the shop floor in a flow sequence, but a job might skip some stages if they are not necessary for its manufacture. Thus, eligibility is represented by the parameter $e_{jkm}^l$, which takes a value of one when job $j \in J^l$ must be executed with machine $m \in M_k^l$ of stage $k \in K^l$ for component $l \in L$. Moreover, let $\tau_{(i,j)km}^l$ represent the setup time between the job pair $(i, j)$ in machine $m \in M_k^l$, for $i, j \in J^l$ and component $l \in L$.

To model the 2-HFSA problem with a mixed integer linear programming formulation, let binary decision variables $y_{jkm}^l$ take a value of one if job $j \in J^l$ is assigned to machine $m \in M_k^l$ in stage $k \in K^l$, for component $l \in L$, and zero, otherwise. To determine the sequence of the jobs, let binary variables $x_{(i,j)km}^l$ take a value of one if job $i \in J^l$ is scheduled immediately before job $j \in J^l$ in machine $m \in M_k^l$, at stage $k \in K^l$, component $l \in L$, and zero, otherwise. Auxiliary variables $C_{jk}^l$ represent the completion time of job $j \in J^l$ in stage $k \in K^l \cup a$ for component $l \in L$. Similarly, $S_{(i,j)km}^l$ denotes the starting time of job $j \in J^l$, scheduled immediately after job $i$ in stage $k \in K^l$ in machine $m \in M_k^l$, and component $l \in L$. Finally, $C_{max}$ represents the *makespan* of the 2-HFSA problem. The mathematical model for the 2-HFSA is as follows.

$$\min \ C_{\max} \tag{1}$$

$$C_{ja}^l \leq C_{\max} \quad j \in J^l, \ l \in L \tag{2}$$

$$y_{jkm}^l \leq e_{jkm}^l \quad i \in J^l, k \in K^l, m \in M_k^l, l \in L \tag{3}$$

$$\sum_{m \in M_k^l} e_{jkm}^l y_{jkm}^l = 1 \quad k \in K^l, j \in J^l, l \in L \tag{4}$$

$$x_{(i,j)km}^l - x_{(j,i)km}^l \leq 1 \quad i, j \in J^l, k \in K^l, m \in M_k^l, l \in L \tag{5}$$

$$x_{(i,j)km}^l \leq y_{jkm}^l \quad i, j \in J^l, k \in K^l, m \in M_k^l, l \in L \tag{6}$$

$$\sum_{j \in J^l} e_{jkm}^l x_{(0,j)km}^l \leq 1 \quad k \in K^l, m \in M_k^l, l \in L \tag{7}$$

$$\sum_{i \in J^l \cup 0} x_{(i,h)km}^l - \sum_{j \in J^l \cup 0} x_{(h,j)km}^l = 0 \quad h \in J^l, m \in M_k^l, k \in K^l, \ \ l \in L \tag{8}$$

$$\sum_{j\in J^l} e^l_{jkm} x^l_{(j,0)km} \leq 1 \quad m \in M^l_k, k \in K^l, l \in L \tag{9}$$

$$C^l_{jk} \geq S^l_{(i,j)km} + p^l_{jkm} y^l_{jkm} + \tau^l_{(i,j)km} x^l_{(i,j)km} - B(1 - x^l_{(i,j)km}) \quad i,j \in J^l, k \in K^l, m \in M^l_k, l \in L \tag{10}$$

$$S^l_{(i,j)km} \geq C^l_{jk-1} \quad i,j \in J^l, k \in K^l, k > 1, l \in L \tag{11}$$

$$C^1_{ia} - C^2_{ja} = 0 \quad (i,j) \in N \tag{12}$$

$$x^l_{(i,j)km}, y^l_{ikm} \in \{0,1\} \quad i,j \in J^l, m \in M^l_k, k \in K^l, l \in L$$
$$C^l_{jk}, S^l_{(i,j)km}, C_{\max} \geq 0 \quad i,j \in J^l, m \in M^l_k, k \in K^l, l \in L$$

Expression (1) represents the makespan objective function. Constraints (2) define the makespan as the completion time of the last job processed in the assembly stage *a*. Expressions (3) assign a job $j \in J^l$ to a machine $m \in M^l_k$ in stage $k \in K^l$ only if the machine is eligible, $l \in L$. Eq (4) imply that each job $j \in J^l$ must be assigned to a machine $m \in M^l_k$ in every stage $k \in K^l$, for $l \in L$. Constraints (5) prevent cycling between two consecutive jobs and define a unique sequence. Constraints (6) state that a job $j \in J^l$ cannot be sequenced in machine $m \in M^l_k$ in stage $k \in K^l$ if not assigned to that machine. Constraints (7)–(9) ensure flow conservation in each machine by considering a dummy job 0 at the beginning and the end of the sequence. Constraints (10)) follow the idea proposed by [36] for sub-tours elimination purposes and establish the completion time of the jobs by considering the setup and the processing times; *B* is a big number that can be easily bounded. Constraints (12) indicate the simultaneous arrival of a product's components at the assembly stage. Finally, the last two constraints establish the nature of the variables.

This mixed integer linear programming can solve small instances, as shown in Section 3. In the following, we propose a matheuristic, a GRASP, and a BRKGA to solve larger instances.

## 2.2 Matheuristic for the 2-HFSA problem

Matheuristics are hybrid methods that combine exact approaches with metaheuristic strategies that aim to acquire the accuracy of mathematical programming and the benefits of computational time consumption that heuristics offer. According to [37], matheuristics often exhibit a "master-slave" scheme, where one of the elements (exact or approximated approach) takes the master's place and controls the other element. In the 2-HFSA problem, the job assignment to the machine stages corresponds to the master problem, while the scheduling of the jobs in the different machines is the slave problem. Both sub-problems are solved with a MILP but linked with a pull-planning heuristic method.

A pull system is a scheduling technique that fixes the project makespan of the jobs and then works backward to outline the steps to achieve the planned makespan quickly and efficiently [38]. In this manner, in our pull-planning matheuristic we aim for the simultaneous arrival of both product components at the assembly stage by finding feasible complying schedulings with our assignment subproblem. Unfeasible assignments are stored on a forbidden list to avoid cycling.

**Algorithm 1** Pull-matheuristic

1: Forbidden assignment solutions $\Gamma = \emptyset$
2: $\bar{C}_{\max} = |J| (\max_{j\in J^1, k\in K^1, m\in M_k^1, l\in L} p^1_{jkm} + \max_{j\in J^1, k\in K^1, m\in M_k^1, l\in L} \tau^1_{jkm})$
3: **while** maximum iterations without improvement not reached **do**
4: solve AS(Γ) to obtain a non forbidden assignment $\bar{Y}$
5: solve $SS(\bar{Y}, \bar{C}_{\max})$ to obtain a job scheduling $\gamma$
6: **while** time limit is not reached **do**
7: **if** $SS(\bar{Y}, \bar{C}_{\max})$ is feasible **then**
8: **if** $SS(\bar{Y}, \bar{C}_{\max})$ objective function value > 0 **then**
9: $\Gamma \leftarrow \Gamma \cup \gamma$, $\bar{C}_{\max} \leftarrow (1+\beta)\bar{C}_{\max}$
10: **else** $\bar{C}_{\max} \leftarrow (1-\beta')\bar{C}_{\max}$, update $Y^*, C^*_{\max}$
11: **end if**
12: **else if**
13: **then** $\bar{C}_{\max} \leftarrow (1+\beta)\bar{C}_{\max}$
14: **end if**
15: **end while**
16: **end while**

Algorithm 1 presents the pull-planning matheuristic pseudocode. First, the forbidden assignment set Γ is empty, and the trivial bound $\bar{C}_{\max}$ on the optimal makespan is set. While a maximum number of iterations without improvement has not been reached, the assignment subproblem AS(Γ) is solved, yielding a job-stage-machine unforbidden assignment $\bar{Y}$. This assignment is a parameter when solving the $SS(\bar{Y}, \bar{C}_{\max})$ model as well as $\bar{C}_{\max}$. While the $SS(\bar{Y}, \bar{C}_{\max})$ model yields a positive solution (step 8), the algorithm increases the $\bar{C}_{\max}$ by a percentage of $\beta$ and the algorithms iterates again after adding the job sequence *gamma* to the forbidden in set Γ. Indeed, an objective function value greater than zero means that the current solution presents some differences between the arrival times of the same product components. If the feasible solution has a value of 0 (step 11), then a feasible solution for the 2-HSFA has been found because the current solution has all the product components arriving simultaneously at the assembly stage. Thus, the $\bar{C}_{\max}$ is reduced by a percentage $\beta'$ to explore a tighter makespan. If no feasible solution is found, then the $\bar{C}_{\max}$ is increased (step 14). The algorithm iterates until a maximum number of iterations without improvement is reached or a time limit is reached (verified in the inner while loop).

The assignment stage of the pull-matheuristic determines the job assignment to the machines at the different stages by minimizing the completion time of the jobs at each stage. In this parametric model, we do not consider the job sequence but only ensure that the proposed assignment is not forbidden and, thus, is not in the Γ set. For this, we solve the following parametric MILP named AS(Γ) where the variable $C^l_{km}$ corresponds to the maximum accumulated processing times in stage $k \in K^l$, calculated considering all batches of component $l \in L$ assigned to any machine.

$$\min \sum_{l\in L}\sum_{k\in K^l}\sum_{m\in M^l_k} C^l_{km} \tag{13}$$

$$\sum_{j\in J^l}\sum_{m\in M^l_k} p^l_{jkm} y^l_{jkm} \le C^l_{km} \quad k \in K^l, l \in L \tag{14}$$

(3)–(4)

$$\sum_{l\in L}\sum_{j\in J^l}\sum_{k\in K^l}\sum_{m\in M^l_k} y^l_{jkm} \le |\gamma| - 1 \quad \gamma \in \Gamma \tag{15}$$

$$y^l_{jkm} \in \{0,1\} \quad j \in J^l, m \in M^l_k, k \in K^l, l \in L$$
$$\mathcal{C}^l_{mk} \geq 0 \quad m \in M^l_k, k \in K^l, l \in L$$

In the AS(Γ) mathematical model, the objective function (13) goal is to bound the processing time used in each stage by minimizing the sum of the maximum processing times for every component and every stage. Constraints set (14) establish the value of the $\mathcal{C}^l_{mk}$ variables; these constraints do not consider the setup times because this model does not perform a schedule. Constraints set (15) make the connection between the AS(Γ) and the scheduling models in the matheuristic by forbidding all the unfeasible job sequences $\gamma \in \Gamma$. The last constraint sets establish the nature of the variables.

The scheduling stage pull-matheuristic is also a parametric MILP that determines the schedule of jobs for each machine at each stage of the 2-HFSA. For this, we consider the assignment solution $\bar{(Y)} = \{\bar{y}^l_{jkm}\}$ yielded by the AS(Γ) model and the actual bound on the makespan $\bar{C}_{\max}$. This model is named SS($\bar{Y}, \bar{C}_{\max}$) and requires excess variables $D(i, j)$ that represent the difference in the completion time of a pair of jobs forming a product at the assembly stage, $(i, j) \in N$.

$$\min \sum_{l \in L} \sum_{(i,j) \in N} D_{(i,j)} \tag{16}$$

$$C^l_{ja} \leq \bar{C}_{\max} \quad j \in J^l, l \in L \tag{17}$$

(5)–(9), (11)

$$x^l_{(i,j)km} \leq \bar{y}^l_{jkm} \quad i, j \in J^l, m \in M^l_k, k \in K^l, l \in L \tag{18}$$

$$C^l_{jk} \geq S^l_{(i,j)km} + p^l_{jkm}\bar{y}^l_{jkm} + \tau^l_{(i,j)km}x^l_{(i,j)km} - B(1 - x^l_{(i,j)km}) \quad i, j \in J^l, k \in K^l, m \in M^l_k, l \in L \tag{19}$$

$$C^1_{ia} - C^2_{ja} \leq D_{(i,j)} \quad (i, j) \in N \tag{20}$$

$$C^2_{ja} - C^1_{ia} \leq D_{(i,j)} \quad (i, j) \in N \tag{21}$$

$$x^l_{(i,j)km} \in \{0,1\} \quad i, j \in J^l, m \in M^l_k, k \in K^l, l \in L \tag{22}$$

$$C^l_{jk}, S^l_{(i,j)km} \geq \quad i, j \in J^l, m \in M^l_k, k \in K^l, l \in L \tag{23}$$

$$D_{(i,j)} \geq 0 \quad (i, j) \in N \tag{24}$$

In the $SS(\bar{Y}, \bar{C}_{\max})$ model, the objective function (16) minimizes the sum of the differences between the completion times of the pair of jobs forming a product at the assembly stage. The completion time of the jobs at the assembly stage is bounded by parameter $\bar{C}_{max}$ in the constraint set (17). Constraint sets (5)–(9) amd (11) are the ones presented previously, defining the job sequence in the stages of the 2-HFSA problem. Constraints (18) define the sequence variables with the parameter values $\bar{y}^l_{jkm}$ obtained by solving the AS(Γ) model. Constraints (19) determine the starting and completion times of the jobs in the different stages, taking into

account the already computed jog assignment $\bar{Y}$. Eq sets (20) and (21) define the time difference $D_{(i,j)}$ between each pair of jobs forming a product in the assembly stage. The rest of the equations are the variable's nature.

Experimental results of the pull-matheuristic will be presented in Section 3.

## 2.3 GRASP for the 2-HFSA problem

The greedy randomized adaptive search procedure (GRASP) is a metaheuristic consisting of constructive iterations of greedily built randomized solutions that are then improved by a local search stage. GRASP was introduced in, and there are several surveys about this topic [39, 40].

Algorithm 2 displays our GRASP procedure to solve the 2-HFSA problem. While a maximum number of iterations is not reached, our GRASP generates a partial solution for $HFS^2$ by adding jobs to the problem's solution from a list of elements ranked by a greedy function according to the partial makespan per stage ($GreedySolutionHFS^2$). Then, with $GreedySolutionHFS^1$, we sequence the jobs in $HFS^1$. As mentioned in the introduction, it is usual that one line is faster than the other. In our case study, $HFS^1$ is the fastest, so we start the greedy solution with the $HFS^2$ and then set the other line accordingly. Combining the two greedy functions forms a feasible solution for the 2-HFSA that is then improved by a local search procedure.

**Algorithm 2** GRASP Algorithm for 2-HFSA

1: $Y^*, X^* \leftarrow \emptyset$, $C^*_{\max} \leftarrow \infty$
2: **for** $i \leq MaxIter$ **do**
3: $\bar{Y}, \bar{X} \leftarrow GreedySolutionHFS^2$
4: $\bar{Y}, \bar{X} \leftarrow GreedySolutionHFS^1\ (\bar{Y}, \bar{X})$
5: $C_{\max} \leftarrow LocalSearch(\bar{Y}, \bar{X})$
6: **if** $C^*_{\max} > C_{\max}$ **then** $Y^* = \bar{Y}, X^* = \bar{X}$, $C^*_{\max} = C_{\max}$
7: **end for**
8: return $Y^*, X^*, C^*_{\max}$

Algorithm 3 details the $GreedySolutionHFS^2$ procedure that corresponds to the greedy construction of a partial solution by scheduling jobs to line $HFS^2$. First, the candidate list $C$ comprises the jobs in $J^2$, and an empty solution is set. While the list is not empty, for each stage, each job $j$ is greedily assigned to the best machine by evaluating its partial completion time in this stage (step 5). Then, we denote by $C^{J^2\setminus C\cup j}_{\max}$ the partial makespan of the elements in $J^2\setminus C$ union job $j$. To obtain variability in the candidate set of greedy solutions, the ranked jobs according to their corresponding $C^{J^2\setminus C\cup j}_{\max}$ are placed in the restricted candidate list RCL (step 9), where $\alpha \in (0, 1)$ is responsible for the size of the RCL. A job $j^*$ is chosen randomly when building the solution (step 10). The completion time of the assembly stage for the $HFS^1$ is then equal to the one obtained in the $HFS^2$.

**Algorithm 3** $GreedySolutionHFS^2$

1: Candidate list $C = J^2$, current solution $\bar{Y} = \emptyset$ and $\bar{X} = \emptyset$
2: **while** $C \neq \emptyset$ **do**
3: **for** $j \in C$ **do**
4: **for** $k \in K^2$ **do**
5: Determine compatible machine $m \in M^2_k$ minimizing $C^2_{jk}$
6: **end for**
7: Compute partial makespan of $HFS^2$: $C^{J^2\setminus C\cup j}_{\max}$
8: **end for**
9: $RCL$ composed by $j \in C$ such that
$C^{J^2\setminus C\cup j}_{\max} < \min_{j'\in C}\{C^{J^2\setminus C\cup j'}_{\max}\} + \alpha\left(\max_{j'\in C}\{C^{J^2\setminus C\cup j'}_{\max}\} - \min_{j'\in C}\{C^{J^2\setminus C\cup j'}_{\max}\}\right)$
10: Randomly select $j^*$ in the $RCL$
11: Update $\bar{Y}, \bar{X}$ with $j^*$ sequenced in the chosen machines of the stages

```
12:   C^2_{j*a} = C^1_{ia}, for (i, j*) ∈ N
13:   C = C\j
14: end while
15: Return Ȳ, X̄
```

Production line one must end simultaneously with production line two; therefore, the constructive algorithm GreedySolutionHFS$^1$ $(\bar{Y}, \bar{X})$ (pseudocode in Algorithm 4) takes the partial solution $(\bar{Y}, \bar{X})$ together with its completion time as input. The greedy scheduling construction of HSF$^1$ is performed from the assembly stage to the first stage, assigning jobs to machines following the sequence order of HFS$^2$ obtained by Algorithm 3. Algorithm 4 assigns each batch to a machine, using the best greedy insertion value in every stage. Note that this procedure may lead to completion time unfeasibility, which is solved by adjusting (increasing) the completion times in line HFS$^2$.

**Algorithm 4** GreedySolutionHFS$^1$ $(\bar{Y}, \bar{X})$

```
1: for i ∈ J^2 ordered with respect to (X̄) do
2:   for k = a, ..., 1 do
3:     Determine compatible machine m ∈ M^1_k minimizing C^1_{jk}
4:     Update Ȳ, X̄ with the assignment of i
5:     if resulting partial solution is unfeasible then
6:       adjust completion times of (i, j) in HSF^2
7:     end if
8:   end for
9: end for
10: Return Ȳ, X̄
```

The obtained solution after the two greedy construction phases is given as input to the Local Search (see Algorithm 2) as an attempt to reduce its objective function. In our GRASP, the local search takes the sequence of the jobs and their scheduling from HFS$^2$ since it is the bottleneck line. Then, we apply the remove/insert operator for every job: we remove the job from its sequence and insert it at the end of the sequence in both lines HFS$^1$ and HFS$^2$ in its best compatible machine. We make their completion time coincide at stage $a$ and then shift all the other jobs and update the total completion time. After obtaining all the possible neighbors of the current solution, we keep the best one and continue with the iterations of the GRASP algorithm.

Fig 2 represents a solution to the 2-HFSA problem after applying our GRASP procedure. The instance has nine pairs of jobs, four stages in each HFS, and unrelated parallel machines. Note that the completion times of each pair at the end of the last stage $k = 4$ of each HFS are simultaneous. This way, the pair of jobs can go into the assembly stage.

## 2.4 BRKGA for the 2-HFSA problem

Genetic Algorithms have been used to solve scheduling problems [41–43]. [44] present the Biased Random Key Genetic Algorithm, BRKGA. In this section, we present a BRKGA to solve the 2-HFSA problem.

As mentioned by [45, 46], BRKGAs start with a population of $p$ chromosomes that will evolve with the next generations until *MaxG*. Each chromosome has $R$ random keys for several generations until a total number of generations is reached. In a given generation $r$, the evolutionary process is as follows.

1. Each chromosome is decoded, and its objective value (or fitness) is computed.
2. The population is partitioned into an elite set containing $p_e$ individuals with the best fitness values and the non-elite set with the rest of the individuals.

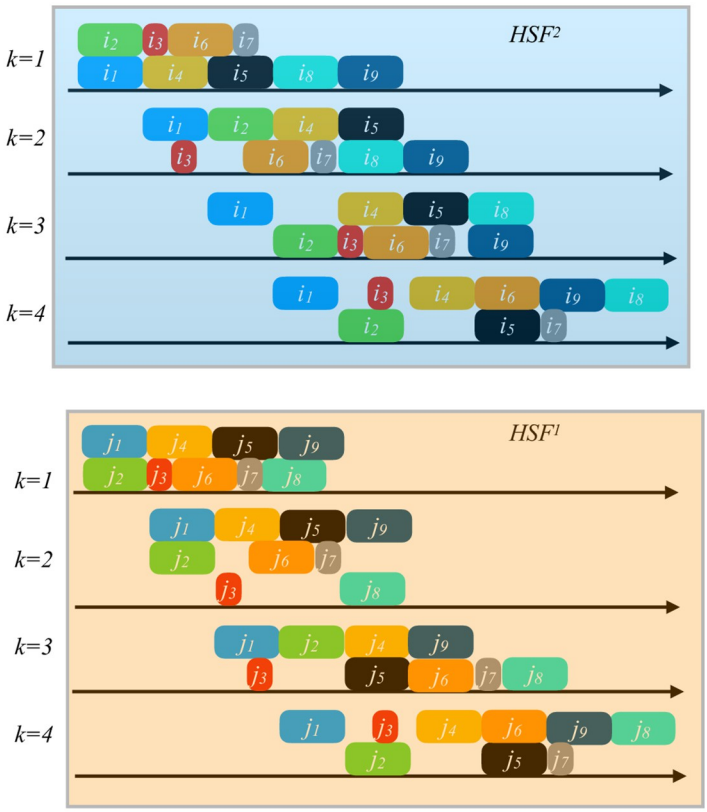


**Fig 2. Solution of the 2-HFSA problem after applying our GRASP procedure to an instance with nine pairs of jobs, four stages in each HFS, and unrelated parallel machines.**

https://doi.org/10.1371/journal.pone.0304119.g002

3. The population at generation $g + 1$ consists of $p_e$ elite individuals of the current generation, $p_m$ mutants that are randomly generated chromosomes, and $p_o = p - p_m - p_e$ offspring chromosomes produced by crossover operation between random individuals with replacement, one from the elite and another from the non-elite set. In the crossover, the $v$-th allele of offspring $u$ is defined by a uniform random number in [0, 1); if this value is larger than the biased probability $\rho > 0.5$, then offspring $u$ inherits the $v$-th allele of its elite parent; otherwise, it inherits the $v$-th allele of the non-elite parent.

The encoding and decoding algorithms are the only problem-dependent stages of the BRKGA. For the 2-HFSA problem, each chromosome of the population will be a vector of size $R = |N|(1 + |K^2| + |K^1|)$: the first $|N|$ alleles represent the sequence of the jobs in both lines, the second $|K^2||N|$ alleles indicate the job assignment to the machines of each stage of $HFS^2$, and the final $|K^1||N|$ alleles represents the job machine assignment for each batch for $HFS^1$.

We use the decoding method to transform the [0,1) random key of size $R$ and obtain a solution related to the 2-HFSA problem with its total completion time. Fig 3 shows an example of our decoding algorithm for an instance with three pairs of jobs and two stages per line. Note that $HSF^1$ has three machines in stage 2. To obtain the sequence of the jobs in $HFS^2$, we sort the first $|N|$ alleles in non-decreasing order. In our example, the sequence would be (3,1,2). Then, the following $|k^2N|$ alleles indicate the machine to which the job will be assigned in the $HFS^2$: the [0,1) interval is divided into the number of machines in the stage. Each machine is then associated with one of the subdivided intervals. In our example, stage 1 has two machines;

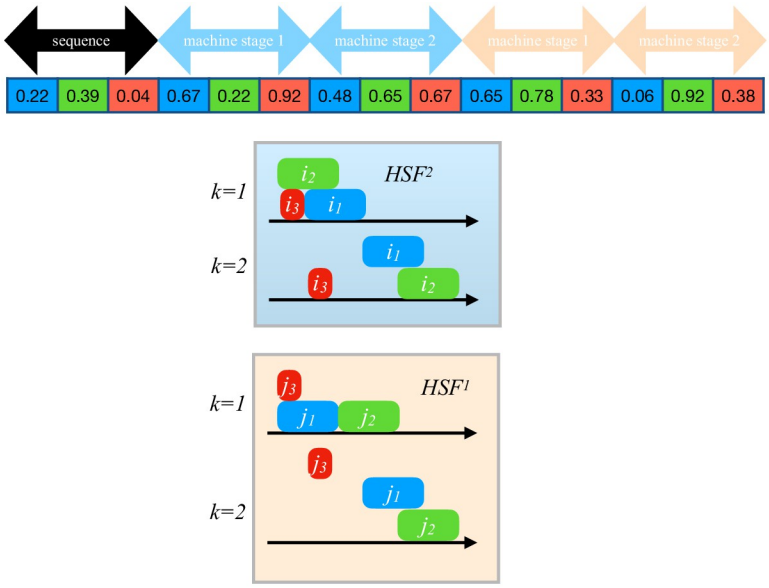


**Fig 3. Random key decoding example for an instance with three pairs of jobs and two stages per line.**

https://doi.org/10.1371/journal.pone.0304119.g003

thus, job $i_2$ is in Machine 1 and $i_3$ and $i_1$ in Machine 2. Similarly, the final $|k^2N|$ alleles are for the HFS[1]. Note that for Stage 2, there are three machines. Once the jobs are sequenced and forced to end simultaneously at the end of the last stage of the lines, the makespan of the individual can be computed.

## 3 Results and discussion

This section describes the computational experiments to validate and compare the mathematical model with the proposed heuristic algorithms. First, we describe the instance's characteristics. Then, we present a parameter-tuning procedure for the different algorithms. Finally, we compare the algorithm's performance.

We used an Intel (R) Core™ processor with 2.9GHz and 8GB RAM speed for computational experimentation. The codes were implemented in C++, using Microsoft Visual Studio 2012 and ILOG CPLEX 12.5 as a solver for the mathematical model. All the instances and the detailed results are in https://doi.org/10.6084/m9.figshare.24246547.

Experimentation includes four types of instances. Each instance type's size depends on the number of jobs, stages, and machines per stage. We create five instances of each type for a total of twenty instances. In Table 1, we present the values used for each type of instance: number of jobs $|N|$, number of stages for HSF[1] and HFS[2] ($|K^1|$ and $|K^2|$, respectively), maximum number

**Table 1. Description of the instance benchmark.**

| Type | $\|N\|$ | $\|K^1\|$ | $\|K^2\|$ | $\max\|M_k^1\|$ | $\max\|M_k^2\|$ | $p_{ikm}^1$ | $p_{jkm}^2$ | $\tau_{(i,j)km}^l$ |
|---|---|---|---|---|---|---|---|---|
| 1 | [10,12] | 2 | 2 | 4 | 3 | [40, 70] | [10, 12] | [25, 45] |
| 2 | [29, 35] | 3 | 3 | 5 | 5 | [40, 70] | [15, 25] | [25, 45] |
| 3 | [50, 60] | 5 | 6 | 10 | 12 | [60, 90] | [20, 30] | [25, 45] |
| 4 | [90, 125] | 12 | 14 | 25 | 29 | [40, 65] | [10, 12] | [25, 45] |

https://doi.org/10.1371/journal.pone.0304119.t001

**Table 2. Parameter tuning for the GRASP and the BRKGA algorithms obtained by Calibra.**

| GRASP | | | BRKGA | | |
|---|---|---|---|---|---|
| **Parameter** | **Range** | **Value** | **Parameter** | **Range** | **Value** |
| $\alpha$ | [0,1) | 0.3 | $p$ | [100,1000] | 887 |
| *MaxIter* | [100, 1000] | 763 | $p_m$ | $[0, 0.3p]$ | $\lceil 0.3p \rceil$ |
| | | | $p_e$ | $[0, 0.7p]$ | $\lceil 0.6p \rceil$ |
| | | | $\rho$ | [0.5,1] | 0.9 |
| | | | *MaxG* | [10, 100] | 78 |

https://doi.org/10.1371/journal.pone.0304119.t002

of machines per line ($\max|M_k^1|$ and $\max|M_k^2|$), processing times for HSF[1] and HFS[2], and setup times values in the last column. In each instance, the last stage is considered as the assembly stage. Type 3 instances emulate the usual production line of our case study, while Type 4 instances are rare but may sometimes occur. Since no similar article exists in the literature, comparing our approach with other approaches is difficult without removing strong components of our problem, like the assembly stage or the two related HFS. Nevertheless, we compare exact and approximated methods to evaluate our methodologies' performance in terms of quality and time.

The algorithm parameters for our GRASP and the BRKGA were tuned using Calibra [47], a software based on a Taguchi Fractional design of experiments enhanced with a local search procedure. Table 2 shows the test values and the results obtained by Calibra for our algorithms. For each parameter, the *Range* is the interval tested by Calibra, and column *Value* is the yielded parameter. For the GRASP, only two parameters must be tuned: the size of the restricted candidate list controlled by $\alpha$ and the maximum number of iterations.

For the pull-matheuristic, at each iteration $\beta$, $\beta'$ are uniformly drawn $\in [0, 0.5)$, the maximum number of iterations without improvements is set to five, and a total of three hours time limit of the inner loop.

Table 3 presents the results obtained with the mathematical model and the three approximated solution approaches that we proposed. The first column corresponds to the instance type. Columns 2 to 4 show the results obtained with the mathematical model. Columns 5 to 9 are for the pull-matheuristic, the next four columns are for the GRASP, and the last four are for the BRKGA. For the mathematical model, we present the objective value ("Obj."), the percentage gap ("Gap") computed as 100(best dual—best feasible solution)/(best feasible solution), and the time in seconds ("T."). We executed the algorithm 20 times for the three approximated algorithms because of their random components. We present the best objective value (columns"Obj."), the average objective function values (columns "Ave."), the average time in seconds ("T."), and the percentage difference ("Dif.") with the best feasible solution found by all the methods computed as 100(Obj.—best Obj.)/(best Obj.). The best objective function values are marked in bold.

For small instances of Type 1, the mathematical model solved with the integer linear solver obtains solutions with a gap average of less than 61%. The solver could not find feasible solutions for many Type 2 instances, moreover, the average gaps are almost 100%. It is important to highlight that the relevance of the model lies in guaranteeing the optimal solution, even if it is only for smaller instances.

The pull-matheuristic finds high-quality solutions for Type 1 instances and better solutions than the exact mathematical model, considering that the gaps were large. Note that the pull-matheuristic resides on exact models; therefore, as expected, it presents more computational time than the GRASP and the BRKGA ones. The pull-matheuristic yields feasible solutions for

**Table 3. Experimental results obtained with the mathematical model, the pull-matheuristic, the GRASP, and the BRKGA.**

| Type | Mathematical model | | | Pull-matheuristic | | | | GRASP | | | | BRKGA | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Obj. | T. | Gap | Obj. | Ave. | T. | Dif. | Obj. | Ave. | T. | Dif. | Obj. | Ave. | T. | Dif. |
| 1 | **1365.8** | 3600 | 68 | 1389.1 | 1396.3 | 2117 | 1 | 1420.3 | 1420.3 | 0 | 2 | 1402.9 | 1411.6 | 0 | 1 |
| | 891.6 | 3600 | 24 | **880.9** | 885.4 | 1176 | 0 | 949.81 | 1033.2 | 0 | 8 | 881.6 | 929.1 | 1 | 0 |
| | 1297.9 | 3600 | 73 | **1289.8** | 1300.9 | 2287 | 0 | 1320.3 | 1320.3 | 0 | 2 | 1300.2 | 1308.2 | 1 | 1 |
| | 1106.7 | 3600 | 68 | **1132.6** | 1137.2 | 2227 | 0 | 1200.5 | 1200.5 | 0 | 6 | 1154.3 | 1165.1 | 0 | 2 |
| | 1194.0 | 3600 | 70 | **1197.2** | 1213.2 | 2241 | 0 | 1229.0 | 1236.1 | 0 | 3 | 1212.0 | 1218.5 | 0 | 1 |
| 2 | - | 3600 | - | 6589.0 | 6698.5 | 8145 | 38 | **4775.2** | 4826.7 | 1 | 0 | 4778.1 | 4799.2 | 1 | 0 |
| | 3411.0 | 3600 | 97 | 4693.7 | 5017.5 | 7645 | 71 | 2779.0 | 2829.5 | 1 | 1 | **2749.8** | 2924.0 | 8 | 0 |
| | 4722.1 | 3600 | 100 | 5131.7 | 5205.0 | 7294 | 26 | **4061.0** | 4067.9 | 1 | 0 | 4062.1 | 4067.1 | 1 | 0 |
| | 4180.9 | 3600 | 100 | 5368.2 | 5368.2 | 7585 | 46 | 3674.5 | 3700.3 | 2 | 0 | **3665.9** | 3678.7 | 4 | 0 |
| | - | 3600 | - | 6388.9 | 6408.2 | 8121 | 32 | 4824.7 | 4863.1 | 1 | 0 | **4824.2** | 4835.2 | 2 | 0 |
| 3 | - | 3600 | - | 3754.3 | 3754.3 | 7958 | 5 | 3564.1 | 3667.8 | 14 | 0 | 3638.2 | 3917.2 | 24 | 2 |
| | - | 3600 | - | 5310.2 | 5310.2 | 13225 | 41 | **3755.3** | 4199.9 | 19 | 0 | 3805.7 | 4228.3 | 28 | 1 |
| | - | 3600 | - | 3590.6 | 3590.6 | 9243 | 7 | **3363.4** | 3857.1 | 14 | 0 | 3457.5 | 3964.6 | 23 | 3 |
| | - | 3600 | - | 5271.0 | 5271.0 | 7117 | 76 | **2990.0** | 3229.9 | 11 | 0 | 3047.8 | 3309.1 | 22 | 2 |
| | - | 3600 | - | 4037.5 | 4037.5 | 7659 | 1 | **3983.1** | 4199.8 | 15 | 0 | 3985.1 | 4127.7 | 24 | 0 |
| 4 | - | 3600 | - | 33730.5 | 33730.5 | 3490 | 161 | **12900.4** | 14201.2 | 291 | 0 | 14241.4 | 18393.7 | 102 | 10 |
| | - | 3600 | - | - | - | 40300 | - | **11085.0** | 12486.8 | 320 | 0 | 14103.2 | 20660.2 | 107 | 27 |
| | - | 3600 | - | - | - | 45242 | - | **12893.4** | 14377.9 | 338 | 0 | 16528.4 | 25785.5 | 110 | 28 |
| | - | 3600 | - | - | - | - | - | **14888.0** | 16095.6 | 493 | 0 | 22467.8 | 45194.9 | 134 | 51 |
| | - | 3600 | - | - | - | - | - | **11638.5** | 12854.1 | 360 | 0 | 14443.6 | 19757.2 | 113 | 24 |
| **Average** | | | **75** | | | | **32** | | | | **1** | | | | **8** |

https://doi.org/10.1371/journal.pone.0304119.t003

Type 2 and 3 ones. However, it could only solve one for Type 4 instances within the time limit. The complexity of the instances for the pull-matheursitic resides mainly in the number of jobs in the flow shops.

Regarding the GRASP and the BRKGA, both algorithms are very efficient in terms of time compared with the mathematical model and the pull-matheuristic. The GRASP algorithm finds the best solutions in the Type 3 and 4 instances, while the BRKGA performs well for the Type 2 instances. Nevertheless, the computational time of the BRKGA is very efficient, even compared to the GRASP for the larger instances. When solved with GRASP or the BRKGA, the complexity of an instance is in the number of jobs that must be processed and assembled in the system. Nevertheless, note that the BRKGA yields solutions in less than two minutes for real-life applications.

## 4 Conclusions

This article studies the 2-Hybrid Flow Shop with Assemble stage, where two HFS must be coordinated to assemble batches of terminated products at the end of the lines. Each product comprises two jobs, each produced in one of the HFSs. In both HFSs, the jobs will be processed in multiple stages with multiple parallel machines to minimize the scheduling time span. The jobs forming a product must arrive at the assembly line simultaneously.

To the best of our knowledge, there is no similar problem in the literature, and since it is based on a real case study, we aimed to solve it exactly and heuristically. This paper proposes a mathematical model for the 2-HFSA and three approximated solution approaches: a

matheuristic based on the decomposition of the exact mathematical model and two metaheuristic algorithms, a GRASP and a BRKGA.

The computational experiments include up to 14 stages and 29 machine instances, including a stage of assembly. According to a literature review, no studies include experimentation with the size of Type 4 instances, which emulates some of the real case scenarios. The computational results give evidence of the suitability of that GRASP-based methodology for company implementation due to the computation elapsed time required by the algorithm for quality solutions.

## Author Contributions

**Conceptualization:** Rafael Muñoz-Sánchez, Iris Martínez-Salazar.

**Investigation:** Rafael Muñoz-Sánchez.

**Methodology:** Iris Martínez-Salazar, José Luis González-Velarde.

**Project administration:** Iris Martínez-Salazar.

**Validation:** Rafael Muñoz-Sánchez, Iris Martínez-Salazar, Yasmín Á. Ríos Solís.

**Writing – original draft:** Rafael Muñoz-Sánchez, Iris Martínez-Salazar, Yasmín Á. Ríos Solís.

**Writing – review & editing:** Iris Martínez-Salazar, Yasmín Á. Ríos Solís.

## References

1. Ruiz R, Vázquez-Rodríguez JA. The hybrid flow shop scheduling problem. European Journal of Operational Research. 2010; 205(1):1–18. https://doi.org/10.1016/j.ejor.2009.09.024
2. Pinedo M. Scheduling Theory, Algorithms, and Systems. 4th ed. Springer; 2012. p. 151–160.
3. Hu Z, Liu W, Ling S, Fan K. Research on multi-objective optimal scheduling considering the balance of labor workload distribution. Plos one. 2021; 16(8):e0255737. https://doi.org/10.1371/journal.pone.0255737 PMID: 34352014
4. Johnson SM. Optimal two-and three-stage production schedules with set up times included. Naval Research Logistics Quarterly. 1954;(1):61–68. https://doi.org/10.1002/nav.3800010110
5. Vignier A, Billaut JC, Proust C. Les problèmes d'ordonnancement de type "flow-shop" hybride: état de l'art. RAIRO-Operations Research-Recherche Operationnelle. 1999; 33(2):117–183. https://doi.org/10.1051/ro:1999108
6. Gupta JN. Two-stage, hybrid flowshop scheduling problem. Journal of the operational Research Society. 1988; 39(4):359–364. https://doi.org/10.1057/jors.1988.63
7. Ribas I, Leisten R, Framiñan JM. Review and classification of hybrid flow shop scheduling problems from a production system and a solutions procedure perspective. Computers & Operations Research. 2010; 37(8):1439–1454. https://doi.org/10.1016/j.cor.2009.11.001
8. Tosun Ö, Marichelvam MK, Tosun N. A literature review on hybrid flow shop scheduling. International Journal of Advanced Operations Management. 2020; 12(2):156–194. https://doi.org/10.1504/IJAOM.2020.108263
9. Néron E, Baptiste P, Gupta JN. Solving hybrid flow shop problem using energetic reasoning and global operations. Omega. 2001; 29(6):501–511. https://doi.org/10.1016/S0305-0483(01)00040-8
10. Jin Z, Yang Z, Ito T. Metaheuristic algorithms for the multistage hybrid flowshop scheduling problem. International Journal of Production Economics. 2006; 100(2):322–334. https://doi.org/10.1016/j.ijpe.2004.12.025
11. Hidri L, Haouari M. Bounding strategies for the hybrid flow shop scheduling problem. Applied Mathematics and Computation. 2011; 217(21):8248–8263. https://doi.org/10.1016/j.amc.2011.02.108
12. Santos D, Hunsucker J, Deal D. On makespan improvement in flow shops with multiple processors. Production Planning & Control. 2001; 12(3):283–295. https://doi.org/10.1080/095372801300107824
13. Negenman EG. Local search algorithms for the multiprocessor flow shop scheduling problem. European Journal of Operational Research. 2001; 128(1):147–158. https://doi.org/10.1016/S0377-2217(99)00354-9

14. Fernandez-Viagas V, Perez-Gonzalez P, Framinan JM. Efficiency of the solution representations for the hybrid flow shop scheduling problem with makespan objective. Computers & Operations Research. 2019; 109:77–88. https://doi.org/10.1016/j.cor.2019.05.002

15. Liu Y, Yin M, Gu W. An effective differential evolution algorithm for permutation flow shop scheduling problem. Applied Mathematics and Computation. 2014; 248:143–159. https://doi.org/10.1016/j.amc.2014.09.010

16. Tang L, Liu W, Liu J. A neural network model and algorithm for the hybrid flow shop scheduling problem in a dynamic environment. Journal of Intelligent Manufacturing. 2005; 16:361–370. https://doi.org/10.1007/s10845-005-7029-0

17. Alaykỳran K, Engin O, Döyen A. Using ant colony optimization to solve hybrid flow shop scheduling problems. The international journal of advanced manufacturing technology. 2007; 35:541–550. https://doi.org/10.1007/s00170-007-1048-2

18. Kurz ME, Askin RG. Comparing scheduling rules for flexible flow lines. International Journal of Production Economics. 2003; 85(3):371–388. https://doi.org/10.1016/S0925-5273(03)00123-3

19. Kurz ME, Askin RG. Scheduling flexible flow lines with sequence-dependent setup times. European Journal of Operational Research. 2004; 159(1):66–82. https://doi.org/10.1016/S0377-2217(03)00401-6

20. Hasani A, Hosseini SMH. A bi-objective flexible flow shop scheduling problem with machine-dependent processing stages: Trade-off between production costs and energy consumption. Applied Mathematics and Computation. 2020; 386:125533. https://doi.org/10.1016/j.amc.2020.125533

21. Yang X, Zeng Z, Wang R, Sun X. Bi-objective flexible job-shop scheduling problem considering energy consumption under stochastic processing times. PloS one. 2016; 11(12):e0167427. https://doi.org/10.1371/journal.pone.0167427 PMID: 27907163

22. Wu Y, Liu M, Wu C. A genetic algorithm for solving flow shop scheduling problems with parallel machine and special procedure constraints. In: Proceedings of the 2003 International Conference on Machine Learning and Cybernetics (IEEE Cat. No. 03EX693). vol. 3. IEEE; 2003. p. 1774–1779.

23. Costa A, Cappadonna FA, Fichera S. A novel genetic algorithm for the hybrid flow shop scheduling with parallel batching and eligibility constraints. Int J Adv Manuf Technol. 2014; 75:833–847. https://doi.org/10.1007/s00170-014-6195-7

24. Alfieri A. Workload simulation and optimisation in multi-criteria hybrid flowshop scheduling: a case study. International Journal of Production Research. 2009; 47(18):5129–5145. https://doi.org/10.1080/00207540802010823

25. Yaurima V, Burtseva L, Tchernykh A. Hybrid flowshop with unrelated machines, sequence-dependent setup time, availability constraints and limited buffers. Computers & Industrial Engineering. 2009; 56(4):1452–1463. https://doi.org/10.1016/j.cie.2008.09.004

26. Ruiz R, Maroto C. A genetic algorithm for hybrid flowshops with sequence dependent setup times and machine eligibility. European Journal of Operational Research. 2006; 169(3):781–800. https://doi.org/10.1016/j.ejor.2004.06.038

27. Khalid QS, Azim S, Abas M, Babar AR, Ahmad I. Modified particle swarm algorithm for scheduling agricultural products. Engineering Science and Technology, An International Journal. 2021; 24(3):818–828. https://doi.org/10.1016/j.jestch.2020.12.019

28. Sung CS, Kim HA. A two-stage multiple-machine assembly scheduling problem for minimizing sum of completion times. International Journal of Production Economics. 2008; 113(2):1038–1048. https://doi.org/10.1016/j.ijpe.2007.12.007

29. Fattahi P, Hosseini SMH, Jolai F, Tavakkoli-Moghaddam R. A branch and bound algorithm for hybrid flow shop scheduling problem with setup time and assembly operations. Applied Mathematical Modelling. 2014; 38(1):119–134. https://doi.org/10.1016/j.apm.2013.06.005

30. You PS, Hsieh YC. A heuristic approach to a single stage assembly problem with transportation allocation. Applied Mathematics and Computation. 2012; 218(22):11100–11111. https://doi.org/10.1016/j.amc.2012.04.066

31. Komaki G, Teymourian E, Kayvanfar V. Minimising makespan in the two-stage assembly hybrid flow shop scheduling problem using artificial immune systems. International Journal of Production Research. 2016; 54(4):963–983. https://doi.org/10.1080/00207543.2015.1035815

32. Shi F, Zhao S, Meng Y. Hybrid algorithm based on improved extended shifting bottleneck procedure and GA for assembly job shop scheduling problem. International Journal of Production Research. 2020; 58(9):2604–2625. https://doi.org/10.1080/00207543.2019.1622052

33. Agnetis A, Pacifici A, Rossi F, Lucertini M, Nicoletti S, Nicolò F, et al. Scheduling of flexible flow lines in an automobile assembly plant. European Journal of Operational Research. 1997; 97(2):348–362. https://doi.org/10.1016/S0377-2217(96)00203-2

34. Li Z, Chen Y. Minimizing the makespan and carbon emissions in the green flexible job shop scheduling problem with learning effects. Scientific Reports. 2023; 13(1):6369. https://doi.org/10.1038/s41598-023-33615-z PMID: 37076558

35. Zhao Y, Luo X, Zhang Y. The application of heterogeneous graph neural network and deep reinforcement learning in hybrid flow shop scheduling problem. Computers & Industrial Engineering. 2024; 187:109802. https://doi.org/10.1016/j.cie.2023.109802

36. Miller CE, Tucker AW, Zemlin RA. Integer programming formulation of traveling salesman problems. Journal of the ACM (JACM). 1960; 7(4):326–329. https://doi.org/10.1145/321043.321046

37. Caserta M, Voß S. Metaheuristics: intelligent problem solving. In: Matheuristics: Hybridizing metaheuristics and mathematical programming. Springer; 2009. p. 1–38.

38. Bonney M, Zhang Z, Head M, Tien C, Barson R. Are push and pull systems really so different? International journal of production economics. 1999; 59(1-3):53–64. https://doi.org/10.1016/S0925-5273(98)00094-2

39. Ribeiro CC, Hansen P, Festa P, Resende MG. GRASP: An annotated bibliography. Essays and surveys in metaheuristics. 2002; p. 325–367.

40. Resende MG, Ribeiro CC. GRASP: Greedy randomized adaptive search procedures. In: Search methodologies: introductory tutorials in optimization and decision support techniques. Springer; 2013. p. 287–312.

41. Tian X, Liu X, Zhang H, Sun M, Zhao Y. A DNA algorithm for the job shop scheduling problem based on the Adleman-Lipton model. Plos one. 2020; 15(12):e0242083. https://doi.org/10.1371/journal.pone.0242083 PMID: 33264317

42. Shi X, Long W, Li Y, Deng D. Multi-population genetic algorithm with ER network for solving flexible job shop scheduling problems. PloS one. 2020; 15(5):e0233759. https://doi.org/10.1371/journal.pone.0233759 PMID: 32470077

43. Peng C, Wu G, Liao TW, Wang H. Research on multi-agent genetic algorithm based on tabu search for the job shop scheduling problem. PloS one. 2019; 14(9):e0223182. https://doi.org/10.1371/journal.pone.0223182 PMID: 31560722

44. Gonçalves JF, Resende MGC. Biased random-key genetic algorithms for combinatorial optimization. Journal of Heuristics. 2011; 17:487–525. https://doi.org/10.1007/s10732-010-9143-1

45. Resende MG, Ribeiro CC. Biased ranom-key genetic algorithms: an advanced tutorial. In: Proceedings of the 2016 on Genetic and Evolutionary Computation Conference Companion; 2016. p. 483–514.

46. Toso RF, Resende MG. A C++ application programming interface for biased random-key genetic algorithms. Optimization Methods and Software. 2015; 30(1):81–93. https://doi.org/10.1080/10556788.2014.890197

47. Adenso-Díaz B, Laguna M. Fine-tuning of algorithms using fractional experimental designs and local search. Law and Society Review. 2006; 54(1):99–114.