Article

Makespan Minimization for the Two-Stage Hybrid Flow Shop Problem with Dedicated Machines: A Comprehensive Study of Exact and Heuristic Approaches

Mohamed Karim Hajji [1,*], Hatem Hadda [2] and Najoua Dridi [2]

[1] College of Engineering and Technology, American University of the Middle East, Egaila 54200, Kuwait
[2] Oasis, Ecole Nationale d'Ingénieurs de Tunis, Université de Tunis El Manar, Tunis 1068, Tunisia; hatem.hadda@esti.rnu.tn (H.H.); najoua.dridi@enit.rnu.tn (N.D.)
* Correspondence: mohammad.hajji@aum.edu.kw

Abstract: This paper presents a comprehensive approach for minimizing makespan in the challenging two-stage hybrid flowshop with dedicated machines, a problem known to be strongly NP-hard. This study proposed a constraint programming approach, a novel heuristic based on a priority rule, and Tabu search procedures to tackle this optimization problem. The constraint programming model, implemented using a commercial solver, serves as the exact resolution method, while the heuristic and Tabu search explore approximate solutions simultaneously. The motivation behind this research is the need to address the complexities of scheduling problems in the context of two-stage hybrid flowshop with dedicated machines. This problem presents significant challenges due to its NP-hard nature and the need for efficient optimization techniques. The contribution of this study lies in the development of an integrated approach that combines constraint programming, a novel heuristic, and Tabu search to provide a comprehensive and efficient solution. The proposed constraint programming model offers exact resolution capabilities, while the heuristic and Tabu search provide approximate solutions, offering a balance between accuracy and efficiency. To enhance the search process, the research introduces effective elimination rules, which reduce the search space and simplify the search effort. This approach improves the overall optimization performance and contributes to finding high-quality solutions. The results demonstrate the effectiveness of the proposed approach. The heuristic approach achieves complete success in solving all instances for specific classes, showcasing its practical applicability. Furthermore, the constraint programming model exhibits exceptional efficiency, successfully solving problems with up to $n = 500$ jobs. This efficiency is noteworthy compared to instances solved by other exact solution approaches, indicating the scalability and effectiveness of the proposed method.

Keywords: scheduling; two-stage flowshop problem; dedicated machines; constraint programming; tabu search; heuristic; Johnson's rule (Johnson's algorithm)

check for updates

Citation: Hajji, M.K.; Hadda, H.; Dridi, N. Makespan Minimization for the Two-Stage Hybrid Flow Shop Problem with Dedicated Machines: A Comprehensive Study of Exact and Heuristic Approaches. *Computation* **2023**, *11*, 137. https://doi.org/10.3390/computation11070137

Academic Editor: Francesco Cauteruccio

Received: 15 June 2023
Revised: 5 July 2023
Accepted: 6 July 2023
Published: 10 July 2023

1. Introduction

In the current economic context, the optimization of operations (such as purchasing, production, transportation, and even financial operations) has found a significant place in the concerns of industries. Making money and/or saving time is no longer an option but rather a necessity that enables the company to survive and face an open and highly competitive international market on one hand, and increasingly demanding customers with a multitude of choices on the other hand. Several approaches have been designed and developed by specialists with the aim of better utilizing resources and eliminating any kind of waste while maintaining an acceptable level of quality. Their scope of action has particularly focused on tactical and operational levels where everything can change at any moment, and where the overall performance of the company depends on the level of mastery of functions such as production and planning. This is where the scheduling function comes into play, serving as a function "that consists of organizing the timing of interdependent operations using limited available resources to carry out a production plan" [1].

Indeed, scheduling is considered one of the most studied branches of operations research in specialized literature, with a significant portion dedicated to flow shop problems. These problems present an intermediate level of complexity, making them a favorable ground for testing new solution approaches. However, researchers aim to approach real-life cases as closely as possible, which has motivated recent work recommending the consideration of increasingly realistic industrial models.

As a result, different configurations of these problems have been considered. Among these configurations is the case of scheduling problems with dedicated machines, which exist in various industries such as the construction industry [2], electronic systems manufacturing [3], paper industry [4], and many others including painting and printing workshops, chemical laboratories, and pharmaceutical industries. Although common in different industries, the problem of scheduling workshops with dedicated machines has not been extensively studied and has been limited to cases with a single machine on one of the stages.

In this paper, we considered the two stages flow shop problem with dedicated machines. We assumed that the dedicated machines are located on both manufacturing stages. These problems are generally NP-hard, which raises several questions: are there any special polynomial cases? Can we find sets of dominant solutions? Up to what sizes can we achieve exact solutions? Beyond this size, what can be proposed?

Firstly, we conducted a literature review on flow shop problems with dedicated machines. Three main observations can be mentioned. Firstly, we noticed the scarcity of works exclusively dedicated to dedicated machines. Secondly, almost all of these works considered the configuration where one of the stages consists of a single machine. We also highlight the lack of specific resolution techniques that exploit the specificity of dedicated machines. This literature review motivated us to undertake more in-depth analyses on both theoretical and empirical levels.

On the theoretical analysis front of the problems considered in this paper, we have put forward elimination rules. These rules were designed to effectively decrease the size of the search space, resulting in substantial time savings during calculations, enabling more efficient and faster computations. Regarding the approximate solution, a metaheuristic has been introduced, which has yielded highly competitive results, often reaching optimality. This work also focused on developing a constraint programming model, which was tested using the Cp Optimizer (IBM). The results were highly satisfactory, and instances of significantly larger sizes (over 500 jobs) were successfully solved.

The motivation behind addressing the two-stage hybrid flow shop problem with dedicated machines using both exact and approximate methods arises from its practical significance in real-life industrial settings. However, limited literature exists on this specific problem configuration, particularly with a focus on cases involving dedicated machines on both stages. Our aim was to fill this research gap and provide insights and solutions that can optimize scheduling and resource allocation in industrial environments effectively. By employing a combination of exact and approximate methods, we can explore the problem's complexity, identify special cases, and develop efficient solution approaches that strike a balance between computational tractability and solution quality. Ultimately, our goal was to offer practical and effective scheduling strategies for industries dealing with two-stage hybrid flow shop problems with dedicated machines, leading to improved operational efficiency and resource utilization.

Throughout this paper, we emphasize the scarcity of research dedicated to dedicated machines and the lack of specific resolution techniques for this configuration. Our comprehensive literature review highlights the need for more in-depth analysis, both theoretically and empirically. We contribute to the field by proposing elimination rules that reduce the search space and improve computational efficiency. Additionally, we introduced a metaheuristic approach that yields highly competitive results. Furthermore, we present a constraint programming model capable of successfully solving instances with significantly larger sizes. In summary, our work not only advances the understanding of scheduling problems with dedicated machines in real-life industrial scenarios but also provides novel solution methods to tackle these challenges.

This paper is organized as follows: In Section 2, we conduct a literature review to explore the existing research and related works in the field and we present the studied problem in Section 3. Then, we introduce certain fundamental features, namely the elimination rules in Section 4 and Lower bounds in Section 5 . Section 6 focuses on the constraint programming modeling of the problem. In Sections 7 and 8, we also present approximated solution approaches by establishing new heuristics and by adapting meta heuristic procedures. Section 9 focuses on the experimental results and Section 10 concludes the paper.

2. Literature Review

Many researchers have focused on flow shop scheduling. The interest in this problem is largely motivated by the particularity of its configuration, which is increasingly found in real workshops, and by the wealth of work on these problems. The objective of this section is to review the existing literature on the flow shop scheduling problems and to identify the gaps between the different works in order to properly place our study in relation to the existing literature.

The two-stage hybrid flow shop problem with dedicated machines is considered as a generalization of other more basic problems such as the serial flow shop with two machines ($F2||C_{max}$). The latter is considered one of the most studied problems in the literature for which an optimal solution is known. Therefore, we present the works and results that are related to our research and particularly those concerning the two-machine flow shop and the two-stage hybrid flow shop with dedicated machines.

We note that this problem is also known in the specialized literature under the names "flow shop with multiple processors" and "flexible flow line" [5]. We can represent the hybrid flow shop problem as a workshop with several stages, each containing one or more machines. If we have only one stage, then the problem is reduced to a parallel machine problem, and if we have only one machine on each stage, then we return to a serial flow shop problem.

Hybrid two-stage flow shop problems are known to be NP-hard in the strong sense as soon as the number of machines on one of the two stages exceeds one, even if preemption is allowed. The research on these problems can be distinguished according to the interest it provides. In what follows, we propose to present results of theoretical interest such as lower bounds, polynomial cases and worst case analyses in the first place, and resolution results (exact and approximate) in the second place.

The techniques used for the exact resolution are in the majority of the cases the branch and bound algorithms (B&B) and mathematical programming. In addition, constraint programming and dynamic programming are used.

A B&B for scheduling problems consists of an implicit enumeration of the solutions by creating partial sequences of jobs one by one and creating a tree structure that expands into complete solutions [6]. The B&B method for scheduling problems was initially introduced in [7], making it the earliest known instance of this approach. Despite the relative success of B&B, they are still unable to solve medium and large size problems, and seem too complex for real world problems. On the other hand, B&B can be used implicitly in mathematical models, in particular to solve sub-problems such as assignment problems. In this context, a mathematical program is proposed for the FSH problem with availability constraints while implicitly using a B&B. In [8] a mathematical model is presented for the $2FHD|s_j, m_j|C_{max}$ that was implemented with Clpex and was only able to solve small instances (25 jobs). In [9], the authors give a mixed programming formulation (MILP) and three heuristics for the m machine flow problem with the objective of minimizing the makespan and maximizing the total net revenue. In [10], the authors proposed an MILP for the non-permutation flow shop problem with availability constraints.

Another approach that frequently draws comparisons to MILP is constraint programming. This approach focuses on formulating and solving problems by defining a set of constraints that must be satisfied. Over the 2010s, some papers have put forward the utilization of Constraint Programming (CP) as a means to tackle flowshop problem. In [11–13] a constraint programming approaches are proposed and compared to MILP formulations in terms of effeciency. According to [6], to this day, no comparison has yet been made between CP modelling and the B&B algorithm, and the techniques have not been applied together. An exact alternative method to modeling and solving the Nurse Scheduling Problem (NSP) using the Constraint Satisfaction Problem (CSP) framework is proposed in [14]. The Weighted Constraint Satisfaction Problem (WCSP) was employed to capture working requirements and nurses' preferences over shift patterns. A variant of the Branch and Bound (B&B) algorithm, enhanced with constraint propagation and ordering heuristics, was developed to solve the WCSP. Experimental results demonstrate the time efficiency of the proposed algorithm, which returns optimal schedules within acceptable running times for several NSP instances. In [15], the authors considered the makepsan minimization for the construction scheduling projects with temporal and renewable resource constraints. Experimental analysis using the IBM ILOG CP Optimizer on generated test instances shows that solving the problem with converging material flows requires significantly more computation time compared to solving the problem with diverging material flows. In [16], the authors studied the task scheduling problem on multicore architectures. They proposed scheduling methods that utilized constraint programming to determine the number of threads for each task and schedule them on the multicore system, with the objective of minimizing the overall schedule length. Through experimental results, the authors demonstrated the superiority of their proposed methods compared to existing approaches based on integer linear programming. In [17] , the authors investigated the performance of different constraint programming solvers for the flexible job shop scheduling problem. They compared five solvers, including commercial and non-commercial software, and analyzed their relative performance on various problem instances. The results show that the IBM ILOG CPLEX CP Optimizer and Google's OR-Tools perform the best, with complementary strengths in finding optimal solutions and quickly determining high-quality feasible solutions. Based on this complementarity, the authors proposed algorithm selection approaches using machine learning techniques. Computational analysis confirms the superiority of these approaches compared to using a single solver. The multi-resource-constrained unrelated parallel machine scheduling problem was considered in [18], aiming to minimize the maximum completion time of jobs. The study incorporated various operational constraints and develops an exact solution approach based on constraint programming (CP). The proposed CP model, enriched with lower bound restrictions and redundant constraints, outperforms existing solutions with an average gap of 15.52%. Computational experiments validate the effectiveness of the proposed approach in handling real-world manufacturing environments.

Approximate resolution is certain cases the only way to solve the NP-hard problems, especially when the instances are very large. For this reason, most specialists have oriented their research towards the development of heuristic methods. We present in what follows a non-exhaustive review of the methods that are related to our work.

We obviously start with the heuristics based on Johnson's rule. We note that this rule is considered the most used in this field of research. We also present Herrmann's heuristic and Lee [19], which present a very interesting priority rule.

S.M. Johnson [20] proposed a decision rule allowing for a solution to the two-machine problem $F2||C_{max}$ in a polynomial time. It can be described as follows: let a_i and b_i be the process times of job i, respectively, on the first and second machines. If for two jobs i and j we have $min(a_i, b_j) \leq min(a_j, b_i)$ then there exists an optimal solution where the job i precedes j. We note that this article has motivated scheduling researchers to accept the makespan as an optimization criterion. [21].

An alternative approach to solving this type of problems is to use metaheuristic methods. Meta-heuristic resolution of the flow shop problems often involves the Tabu Search (TS), Simulated Annealing (SA), the Genetic Algorithms (GA), and the Ant Colony Algorithms. We will briefly present some works dealing with GA and then discuss works involving TS and SA.

A genetic algorithm was proposed in [22] to deal with the m stage flow shop problem. A three-stage hybrid flow shop problem inspired by a real problem in the electronic circuit manufacturing industry was studied in [3] by presenting an AG.

The results have been satisfactory considering the relative complexity of the addressed problems. It is important to mention that implementing a genetic algorithm is not as complicated as it may seem, as it involves only a few rules for crossover and selection. However, we believe that this procedure may not allow for an ideal exploration of the search space, which limits the discovery of new points that could contain global minima especially for the flow shop with dedicated machines.

Regarding TS and SA, a two-stage hybrid flow shop problem with m parallel machines in each stage was considered in [23]. The authors presented two meta-heuristics, a tabu search procedure, and a simulated annealing procedure. A similar comparative study was also presented in [24,25] for the problem $F2(D^m, 1) \mid r_i, q_i \mid C_{max}$. A tabu search method for $F_m||C_{max}$ that uses an insertion operator to generate the neighborhood was also proposed in [26]. In [27], the authors proposed a constructive method in a Tabu search procedure. For this same problem, two simulated annealing procedures are presented in [28] with alternative methods for assigning jobs to separate stages.

In [29], the authors presented a tabu search for the problem $F_m||C_{max}$, where the generation of the neighborhood relies on the computation of lower bounds to evaluate the movements. In this procedure the tabu list had a dynamic length that is modified cyclically. The authors claimed that it provides significantly better results than other procedures in the literature. A tabu search for the two-stage flexible flow shop was also implemented in [30]. Another tabu search was also presented in [2] for a hybrid no-wait flow shop problem inspired by a real industrial problem. In [31], the authors presented a novel hybrid algorithm that combines tabu search with a genetic algorithm for flow shop scheduling to minimize makespan. The algorithm incorporates a new population initialization technique, balancing global and local search. Empirical results demonstrate its superior performance, outperforming six other hybrid algorithms, showcasing its effectiveness in achieving higher-quality solutions. In a similar context, a comprehensive approach to the general distributed flexible job shop scheduling problem (DFJSP) in [32] by considering operation outsourcing and multiple optimization objectives. The proposed hybrid genetic algorithm and tabu search (H-GA-TS) outperforms simple GA and TS algorithms, combining global search capabilities with efficient local search. Experimental comparisons confirm the effectiveness and performance advantages of the hybrid algorithm, demonstrating its ability to improve solution quality. In [33] the authors addressed a proactive resource-constrained project scheduling problem with skill-switching resources to maximize schedule robustness. An integer linear programming model was formulated and a tabu search algorithm embedded with two specialized algorithms (TS-IM1 and TS-IM12) was developed. Computational experiments confirmed the effectiveness of the designed algorithms and improvement measures, with TS-IM12 outperforming a commercial solver and demonstrating higher efficiency in solving the problem. In [34], the authors considered a material transportation optimization in mining operations by utilizing autonomous mining trucks. The study proposed a mixed-integer programming model to jointly optimize truck trips and speeds, aiming to minimize energy consumption. A novel tabu search algorithm was developed, consisting of an improved flow allocation model and a guided tabu search procedure. The effectiveness of the proposed model and real-time scheduling approach was validated through a case study in a coal mine in Inner Mongolia, China. The results demonstrated the acceleration of the tabu search procedure and the computational feasibility of the real-time scheduling system for short-term decisions.

The literature review we have carried out tells us about the paucity of work on hybrid flow shops containing several dedicated machines on both stages at the same time. The aim of our study was to somewhat reduce this gap and to offer a contribution to the study and resolution of the two-stage problem with dedicated machines, including the proposition of elimination rules.

Indeed, our interest in tabu search was mainly motivated by the rarity of this type of approach, given that the vast majority of research focuses on genetic algorithms. We also believe that we have not come across a study that presents a tabu search that is accurately and suitably configured.

Therefore, we believe that there is an opportunity to improve the current state-of-the-art by conducting a thorough analysis of the effects of different parameter settings on the performance of tabu search. This would provide insights into how to effectively apply tabu search to various optimization problems and make it a more widely used technique in the field.

3. Problem Description $F2(D^P, D^M)||C_{max}$

In scheduling problems, various codifications have been proposed to standardize notation and facilitate problem description. One widely used notation is the three-field coding scheme: $\alpha \mid \beta \mid \gamma$. The α field describes workshop characteristics, such as the type of machines and their arrangement. The β field captures job-specific details, including constraints and properties such as preemption, deadlines, and release dates. The γ field represents the chosen optimization criterion, such as minimizing completion times, delays, or penalties. This notation was proposed by Graham [35], providing a standardized approach for representing scheduling problems.

The two-stage hybrid flow shop problem with dedicated machines denoted by $F2(D^P, D^M)||C_{max}$ describes a workshop that consists of two stages, where there are P machines on the first stage and M machines on the second stage. The problem consists in scheduling a set $J = \{J_1, J_2, ..., J_n\}$ of n jobs on the different machines of the workshop in order to minimize the maximum completion time "makespan". The machines of the workshop are dedicated in two stages, which means that each machine executes only a given type of job. Each job has two operations to complete (one on each stage) before leaving the workshop; it can only start the second operation if the first operation is completed. We note that a machine can only execute one job at a time. Figure 2 describes an example of a workshop for the $F2(D^P, D^M)||C_{max}$. In what follows, we used the following notations.

$n > 0$	Number of jobs to schedule;
$J = \{J_1, J_2, \ldots, J_n\}$	Set of all the jobs;
$C_{max}\pi$	Makespan of the solution π;
π^*	an optimal solution ;
C^*_{max}	Optimal makespan;
P	Number of machines on the first stage ($p \in \{1 \ldots P\}$);
M	Number of machines on the second stage ($m \in \{1 \ldots M\}$);
$a_j > 0$	Process time of the job j on the first stage;
$b_j > 0$	Process time of the job j on the second stage;
$C^1_{j,p}$	completion time of job j on machine p on the first stage, $p \in \{1 \ldots P\}$;
$C^2_{j,m}$	completion time of job j on machine m on the second stage, $m \in \{1 \ldots M\}$;
T^1_p	Set of jobs to be executed on the machine p, $p \in \{1 \ldots P\}$;
T^2_m	Set of jobs to be executed on the machine m, $m \in \{1 \ldots M\}$;
$n_{1,p} = \lvert T^1_p \rvert$	Number of jobs to be executed on the machine p, $p \in \{1 \ldots P\}$;
$n_{2,m} = \lvert T^2_m \rvert$	Number of jobs to be executed on the machine m, $m \in \{1 \ldots M\}$;
π^p_u	The job at position u ($1 \leq u \leq n_{1,p}$) on machine p, $p \in \{1 \ldots P\}$;
σ^m_v	The job at position v ($1 \leq v \leq n_{2,m}$) on machine m, $m \in \{1 \ldots M\}$;

We denote by $R_{p,m}$ the set of jobs passing on machine p at the first stage and then on machine m at the second stage, $R_{p,m}$ is called the jobs route. We also denote by $n_{p,m}$ the number of jobs following the route $R_{p,m}$. Note that we have a total of $P \times M$ routes for the $F2(D^P, D^M)||C_{max}$.

For a given solution, let J_u and J_v be two jobs of the same route possibly separated by other jobs of different types on a dedicated machine, we denote by t_{uv} the sum of the process times of the jobs (belonging to the other types) which are scheduled between J_u and J_v on this dedicated machine. For a given type T_k^α, $k' \in \{1, \ldots, P\}$ (resp. $k' \in \{1, \ldots, M\}$) and $\alpha = \{1,2\}$ for the first and second stage, we note by $\overline{T_k^\alpha} = \{J_i \in T_k^\alpha | a_i \leq b_i\}$ and $\underline{T_k^\alpha} = \{J_i \in Q | a_i > b_i\}$. Moreover, for any given set of jobs $Q \subseteq J$ we denote by $a(Q) = \sum_{J_i \in Q} a_i$ and $b(Q) = \sum_{J_i \in Q} b_i$. Recall that the two-machine flow shop problem ($F2||C_{max}$) can be solved by applying Johnson's rule (JR) [20]: J_i precedes J_j if

$$min\{a_i, b_j\} \leq min\{a_j, b_i\}$$

Moreover, for a given type T_k^α, $k \in \{1, \ldots, m\}$, we denote by z^k the optimal makespan if we consider the $F_2||C_{max}$ problem for the jobs of T_k^α. We note that $z^k \leq C^*_{max}$, $\forall\ k \in \{1, \ldots, m\}$.

In Figure 1, we present a visual representation of the workshop under consideration.

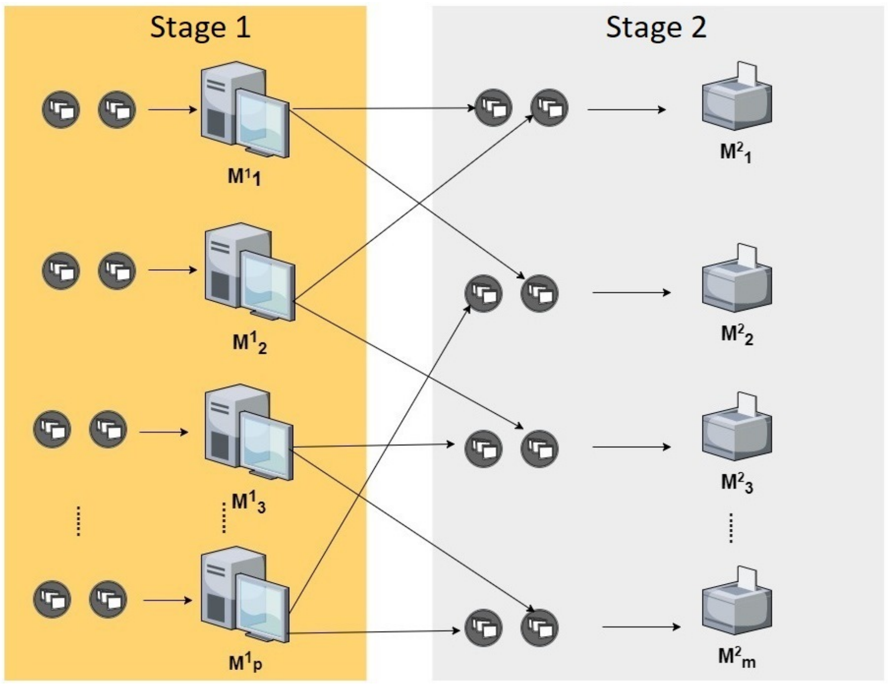

Figure 1. Example of a workshop for the $F2(D^P, D^M)||C_{max}$.

Structure of an $F2(D^P, D^M)||C_{max}$ Solution

A solution for the $F2(D^P, D^M)||C_{max}$ problem is determined by giving several permutations which specify the order of appearance of the jobs, respectively, on P machines on the first stage and on M machines on the second stage. It is then necessary to describe the solution in terms of sub-permutations on each of the machines. Thus, the relevant matter in the π permutation, for example, is the order in which the jobs of the same type appear, independently of their positions in relation to the jobs of other types. As an illustration, let us consider the sequence $\pi = \prec J_1J_2J_3J_4 \succ$ for an instance with $P = 2$ machines on the first stage where J_1 and J_2 will pass on M_1 and J_3 and J_4 on M_2. The permutation π leads to the sub-sequence $\pi^1 = \prec J_1J_2 \succ$ on M_1 and $\pi^2 = \prec J_3J_4 \succ$ on M_2. Note that any other permutation π' in which J_1 precedes J_2 and J_3 precedes J_4 will return the same subsequences π^1 and π^2.

Therefore, it is convenient to represent a solution of the $F2(D^P, D^M)||C_{max}$ problem by giving P permutations $\pi^\alpha(\alpha \in 1..P\})$ of the jobs on each machine of the first stage, and M permutations $\sigma^\beta(\beta \in\in 1..M\})$ of the jobs on the second stage. We will also retain the combination of these permutations $\pi = \pi^1\pi^2....\pi^P$ and $\sigma = \sigma^1\sigma^2...\sigma^M$ and the solution will be denoted (π, σ).

The makespan C_{max} for a given solution is calculated by finding a critical path on both stages. This path is defined as the longest path on the set of possible paths. A path can be defined by giving a job $\pi_u^p = \sigma_v^m$ with $1 \leq u \leq n_{1,p}$ which will define a sequence of operations of the jobs on the machines M_p and M_m. The formula of calculation of the length of a path $l(u, v, p, m)$ is given by Equation (1).

$$C_{l(u,\ v,\ p,\ m)} = (\sum_{i=1}^{u} a_{\pi_i^p} + \sum_{i=v}^{n_{km}} b_{\sigma_i^m}) \tag{1}$$

with

$$\begin{array}{l} 1 \leq p \leq P \\ 1 \leq u \leq n_{1,p} \\ \pi_u^p = \sigma_v^m \,. \end{array}$$

The makespan of a given solution (π, θ) is defined as the maximum on all possible paths as given by Equation (2).

$$C_{max}(\pi, \theta) = \max_{p \in \{1..P\}, m \in \{1..M\}, u \in \{1..n_{1,p}\}, \pi_u^p = \sigma_v^m} C_{l(u,\ v,\ p,\ m)}. \tag{2}$$

Note that, for the critical path (see Figure 2), the finishing execution time of π_u^p on M_p coincides with the start time of its execution on M_m.

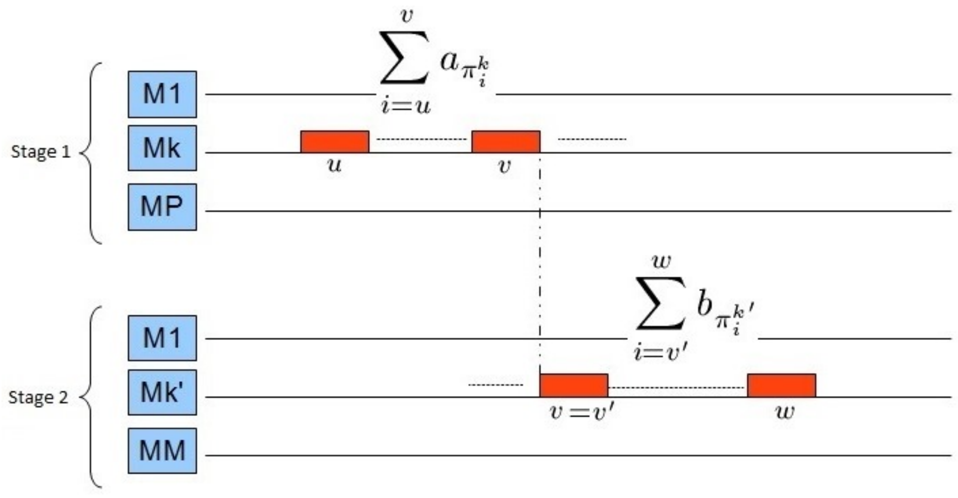
Stage 1
M1
Mk
MP
$\sum_{i=u}^{v} a_{\pi_i^k}$
u
v
Stage 2
M1
Mk'
MM
$\sum_{i=v'}^{w} b_{\pi_i^{k'}}$
$v = v'$
w

Figure 2. Makespan calculation.

4. Dominant Solutions for the $F2(D^P, D^M)||C_{max}$)

In this section, we identify a set of dominant solutions for $F2(D^P, D^M)||C_{max}$.

As we have several machines on each stage, we cannot accurately talk about permutation solutions (see Section 3). We keep the same output order of the jobs on the first stage; the jobs are thus executed according to the first-in-first-out (FIFO) rule. The following result establishes the dominance of these solutions.

Theorem 1. *For $F2(D^P, D^M)||C_{max}$, there exists an optimal solution that follows the FIFO rule at the second stage.*

Proof. By fixing the jobs on the dedicated machines of the first stage, the minimization of the makespan on the second stage comes down to consider M problems of $1|r_i|C_{max}$ where the availability date r_i of J_i, $\forall J_i \in T_k^2$ and $k \in 1 \ldots M\}$, corresponds to its completion date on the first stage. The optimal solution of $1|r_i|C_{max}$ is given by the scheduling of the jobs in the increasing order of their availability dates [36,37]. □

This result reduces the effort of scheduling to the first stage only since the FIFO rule is dominant for the second stage. Thus, the θ permutation (the second stage order) is entirely defined by the FIFO rule. In what follows, we will only consider this type of solution and we will only refer to π (sequence of jobs on the first stage) to designate a given solution (instead of (π, θ)).

Theorem 1 outlines an efficient approach for achieving a comprehensive solution using the provided sub-permutations P on the first stage machines. This method effectively reduces the search space, enabling significant compression.

Theorem 1 proposes an alternative approach where, rather than seeking the optimal solution for the n jobs across both stages, the focus is placed on identifying the best sub-permutation specifically for the dedicated machines in the first stage (with a total of $\Pi_{1 \le k \le P} n_{1,k}!$ solutions).

Note also that, given a π solution, it is always possible to construct a π' solution by rearranging the jobs on the second stage according to Theorem 1 so that $C_{max}\pi' \le C_{max}\pi$.

We now introduce the Corollary 1 for $F2(D^P, D^M)||C_{max}$.

Let u and v be two π jobs with the same route and consecutive on the first stage. Let π_{uv} (respectively, θ_{uv}) be the set of jobs of the other routes that separates u and v on the first stage (respectively, on the second stage) (see Figure 3) and let $t_{\pi_{uv}}$ and $t_{\theta_{uv}}$ be their respective process times.

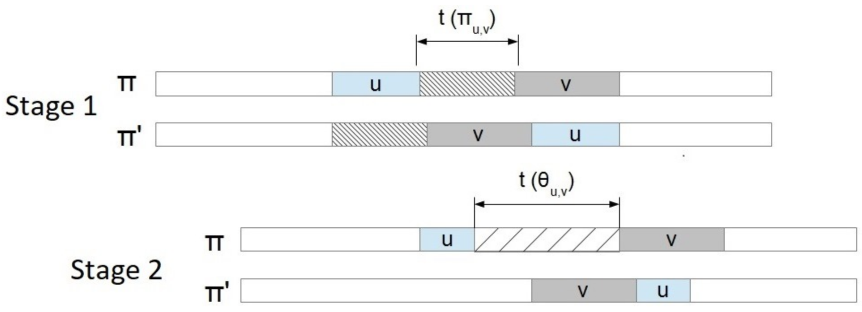

Figure 3. Construction of π' based on π.

Corollary 1. *Given a solution π to the problem $F2(D^P, D^M)||C_{max}$, if the relation*

$$min\{t_{\pi_{uv}} + a_v; b_u + t_{\theta_{uv}}\} \le min\{a_u; b_v\} \quad (3)$$

is satisfied, then the solution π' obtained from π in which $\prec \pi_{uv}, v \succ$ precedes u at the first stage, and the job v precedes $precu, \theta_{uv} \succ$ on the second stage, has a makespan value less than or equal to that of π (i.e., $C_{max}(\pi') \le C_{max}(\pi)$).

Proof. Consider the problem $F2||C_{max}$ with the two machines that constitute the route of u and v, with the fictive jobs u' and v' where $a_{u'} = a_u$, $b_{u'} = b_u + t_{\theta_{uv}}$, $a_{v'} = t_{\pi_{uv}} + a_v$, and $b_{v'} = b_v$.

Equation (3) corresponds to Johnson's rule applied to the jobs u' and v'. Hence, their inversion will not deteriorate the makespan. On the other hand, the forward scheduling of the π_{uv} will not affect the execution dates of the other jobs. Hence, $C_{max}(\pi') \le C_{max}(\pi)$. □

Note that, by using the Corollary 1, it is possible to demonstrate the existence of an optimal solution in which the couples of consecutive jobs of the same type satisfy the relation (3).

5. Lower Bounds

In this section, we present five lower bounds as a point of reference with which to assess the efficiency of heuristic solutions.

For the first lower bound, it is clear that we cannot avoid the maximum of the sums of the process times of each type (T_p^1) on the first stage with the minimum value of the process time b_i ($i \in T_p^1$) on the second stage. This bound is given by the following formula:

$$LB_1 = \max_{i \in T_p^1, p \in \{1 \dots P\}} (\sum a_i + \min b_i).$$

Symmetrically, we considered the following lower bound:

$$LB_2 = \max_{i \in T_m^2, m \in \{1 \dots M\}} (\sum b_i + \min a_i).$$

We also established two lower bounds based on Johnson's rule (JR) [20]. We first separated each subgroup of jobs with the same route. The application of Johnson's rule gives for each route $R_{p,m}$ a subpermutation $\pi_{p,m}$ whose makespan value is $C_{max}^{Joh}(\pi_{p,m})$. We retained the maximum value over all routes.

$$LB_3 = \max_{1 \le p \le P,\ 1 \le m \le M} (C_{max}^{joh}(\pi_{p,m})).$$

For the LB_4 bound, we considered the last route of jobs that appears for the first time on a given machine on the second stage. These jobs were preceded by at least one job from each other route that shares that machine. We then considered the following bound:

$$LB_4 = \min_{1 \le p \le P, 1 \le m \le M} (C_{max}^{joh}(\pi_{p,m}) + \sum_{1 \le k \le M, k \ne m} \min_{R_{pk}} a_i).$$

Symmetrically, we considered the following lower bound:

$$LB_5 = \min_{1 \le p \le P, 1 \le m \le M} (C_{max}^{joh}(\pi_{p,m}) + \sum_{1 \le k \le P, k \ne p} \min_{R_{km}} b_i).$$

We finally retained

$$LB = \max_{1 \le i \le 5} \{LB_i\}$$

6. Modeling and Exact Resolution

In this section, we proposed an exact resolution approach using constraint programming, which has been less commonly utilized compared to other techniques such as linear programming models. This relative scarcity of research employing constraint programming as the primary approach may be attributed to the extensive exploration and familiarity with other methods, such as linear programming or heuristic algorithms.

The utilization of constraint programming offers several advantages. Firstly, it has proven highly effective in optimizing diverse domains and handling intricate optimization challenges. Furthermore, the modeling process in constraint programming is simplified, as the problem can be expressed as a set of constraints. This simplification reduces the problem's complexity, making it more comprehensible and solvable.

Through further research and exploration of constraint programming, new insights and more efficient solutions can be discovered for such complex scheduling problems. Therefore, we have chosen constraint programming as our preferred approach for solving the two-stage hybrid flowshop problem with dedicated machines, leveraging its strengths to contribute to the advancement of the field.

It is crucial to highlight that formulating the problem as a Constraint Satisfaction Problem (CSP) is imperative for effectively leveraging constraint programming. A CSP is a specific type of mathematical problem that involves finding a solution satisfying a given set of constraints. According to Baptiste et al. [38], a CSP is described by the triplet {Variables, Values, Constraints }, where each variable relates to a set of possible values called "domain of variable", and each constraint designates the possible and allowed combinations of the various values of these variables. For constraint programming, we recognize different types of variables including intervals and sequences. An interval for a scheduling problem is characterized by the triplet { Process time, Start date, End date} (Size, Start value, End Value). A sequence is a set of non-overlapping intervals. The constraints involve the start value (startOf) and the end value (endOf) of the domain of each variable. The satisfaction of these constraints depends on the respect of these start and end times. In other words, a job can only start its second operation if its first operation is finished. Moreover, given the constraint of the capacity of the machines, a job can only be executed if its predecessor has finished its execution, hence the constraint of non-overlapping of the intervals in a sequence.

CSP Model

Variables

Intervals

$C^1[p, j, a_j]$: interval of job j on the machine p ($j \in T^1_p,\ p \in \{1, \ldots, P\}$) of size a_j;
$C^2[m, j, b_j]$: interval of job j on the machine m ($j \in T^2_m,\ m \in \{1, \ldots, M\}$) of size b_j;

Sequences

α_p : Sequence of size $n_{1,p}$ containing the completion times of the jobs $j \in T^1_p$, $p \in \{1, \ldots, P\}$;
β_m : Sequence of size $n_{2,m}$ containing the completion times of the jobs $j \in T^2_m$, $m \in \{1, \ldots, M\}$;

Model

$min\ C_{max}$

Under constraints

$$noOverlap\ \ (\alpha_p\ ),\quad p \in \{1, \ldots, P\} \tag{4}$$

$$noOverlap\ \ (\beta_m\ ),\quad m \in \{1, \ldots, M\} \tag{5}$$

$$startOf\ \ (\ C^1[p, j, a_j]\ ) \geq 0\ ,\quad j \in T^1_p,\ p \in \{1, \ldots, P\} \tag{6}$$

$$startOf\ \ (\ C^2[m, j, b_j]\ ) \geq endOf\ \ (\ C^1[p, j, a_j]\ )\ ,\quad j \in T^1_p \cap T^2_m,\ p \in \{1, \ldots, P\},\ m \in \{1, \ldots, M\} \tag{7}$$

$$C_{max} \geq endOf\ \ (\ C^2[m, j, b_j]\ )\ ,\quad j \in T^2_m,\ m \in \{1, \ldots, M\}. \tag{8}$$

The constraints (4) (resp. (5) prevent the overlapping of the intervals containing the process times of the jobs on the first (resp. second) stage. The constraint (6) indicates that all the starting dates of the intervals on the first stage are positive. The constraint (7) indicates that a job can only start its execution on the second stage once its operation on the first stage is finished. The constraint (8) defines the makespan to be minimized.

7. Heuristic Resolution

In this section, we present a new heuristic named the Johnson-Indexed Priority Algorithm (JIPA) for the $F2(D^P, D^M)||C_{max}$.

The idea is to Initially schedule the jobs of the same route according to Johnson's rule to obtain π_{pm}. We then calculate a PSI (Priority Scoring Index) that will define the appearance order of these jobs on the first stage. In other words, the PSI will define the priority on the first stage between two jobs of different routes. Similar ideas have been proposed for similar problems in [19,39].

We considered two possibilities for computing the PSI. Let u be a job in the subsequence $\pi_{p,m}$, the first PSI is given by $PSI_u = \sum_{i=w}^{n_{p,m}} b_{\pi_{p,m}(i)}$, where $\pi_{p,m}(i)$ is the i th job in $\pi_{p,m}$ and w represents the position of job u in $\pi_{p,m}$. This index only considers the remaining workload on the second stage.

Note that when the machines on the second stage are heavily loaded, Johnson's rule tends to sort the jobs in ascending order of their process times on the first stage. To better adjust and use this rule in such cases, we considered the inclusion of the process times of both stages in the calculation of the PSI. We thus introduced a second index $PSI'_u = \sum_{i=w}^{N_{p,m}} \frac{a_{Pi_{p,m}(i)}}{b_{Pi_{p,m}(i)}}$.

The jobs were then scheduled according to the decreasing order of their indexes PSI_u or PSI'_u. The Johnson-Indexed Priority is described by Algorithm 1.

Algorithm 1 Johnson-Indexed Priority Algorithm

```
Start
Step 1. Apply $JR$ for all sub-sequences $\pi_{p,m}$, $p \in \{1 \dots P\}$, $m \in \{1 \dots M\}$.
Step 2. Calculate the indexes $PSI$ and $PSI'$.
Step 3. Schedule the jobs in the decreasing order of $PSI$
Step 4. Return $\pi^{PSI}$ and $\pi^{PSI'}$
Step 5. Return $C_{max} = min(C_{max}^{\pi^{PSI}}, C_{max}^{\pi^{PSI'}})$.
END.
```

The application of the Johnson-Indexed Priority Algorithm is illustrated by the following instance.

Let there be an instance of $F2(D^2, D^2)||C_{max}$ with $n = 8$ jobs, $P = 2$ machine on the first stage and $M = 2$ machines on the second stage. Let $T_1^1 = \{J_1, J_3, J_5, J_7\}$ and $T_2^1 = \{J_2, J_4, J_6, J_8\}$ be the sets of jobs passing respectively on the first and second machine of the first stage. Let $T_1^2 = \{J_2, J_3, J_6, J_7\}$ and $T_2^2 = \{J_1, J_5, J_4, J_8\}$ be the sets of jobs passing, respectively, on the first and second machines of the second stage. The process times are presented in Table 1.

Table 1. Example of instance for the $F2(D^2, D^2)||C_{max}$.

Jobs	J_1	J_2	J_3	J_4	J_5	J_6	J_7	J_8
a_i	4	7	2	5	3	1	4	6
b_i	6	3	5	3	4	4	6	2

According to the types of jobs on the different stages, we can obtain the routes which are as follows: $R_{11} = \{J_3, J_7\}$, $R_{12} = \{J_1, J_5\}$, $R_{21} = \{J_2, J_6\}$ and $R_{22} = \{J_4, J_8\}$.

Applying the Johnson rule to these routes separately gives the following sub-sequences: $\prec J_3, J_7 \succ$, $\prec J_5, J_1 \succ$, $\prec J_6, J_2 \succ$ and $\prec J_4, J_8 \succ$.

To schedule the sets $\prec J_3, J_7 \succ$ and $\prec J_5, J_1 \succ$ on M_1^1, then $\prec J_6, J_2 \succ$ and $\prec J_4, J_8 \succ$ on M_2^1, we calculate the index $PSI_u = \sum_{i=w}^{n_{p,m}} b_{\pi_{p,m}(i)}$. For the job J_3, for example, the calculation of the index is based on $\prec J_3, J_7 \succ$ giving $PSI_3 = b_3 + b_7 = 11$ and $PSI_7 = b_7 = 6$. For the set $\prec J_5, J_1 \succ$,we have $PSI_5 = 10$ and $PSI_1 = 6$. The decreasing order of the indices returns the sequence $\prec J_3, J_5, J_7, J_1 \succ$ to be scheduled on M_1^1. Similarly, for M_2^1, we get the sequence $\prec J_6, J_4, J_2, J_8 \succ$.

As a result, the application of the $JIPA$ returns the solution $\pi_H = \prec J_3, J_5, J_7, J_1 \succ, \prec J_6, J_4, J_2, J_8 \succ$ with $C_{max}^{JIPA} = 21$, which coincide with the lower bound $LB_1 = \max_{i \in T_p^1, p \in \{1 \dots P\}} (\sum a_i + \min b_i)$, i.e., that an optimal solution is obtained.

It should be noted that this index does not modify the order given by the Johnson rule at each sub-sequence $\pi_{p,m}$, it only sets the priority order between the jobs not belonging to the same route.

8. Metaheuristic Resolution

To tackle this complex problem, we turn to metaheuristics, which are global optimization techniques that aim to find high-quality solutions by exploring a large solution space. One promising metaheuristic for this problem is the tabu search algorithm. Specifically tailored for combinatorial optimization problems, the tabu search algorithm utilizes a search strategy that avoids revisiting previously explored solutions, which can help increase the efficiency of the search process and improve the likelihood of finding a good quality solution. By applying tabu search to the two-stage hybrid flowshop problem, we can more effectively navigate the large solution space and find solutions that are close to optimal.

To assess the effectiveness of tabu search in solving the two-stage hybrid flowshop problem, we introduced and tested five different variants of this metaheuristic. Each variant was applied to various classes of problem instances, and the generated solutions were later compared to an optimal solution if one was available, or to a lower bound if an optimal solution cannot be found. By using these measures, we can assess the effectiveness of each variant of the tabu search algorithm and identify which one(s) perform best.

Additionally, comparing the solutions generated by tabu search to an exact or lower bound solution will give us insight into the quality of the tabu search solutions and how close they are to optimal. Through these tests, we hope to gain a comprehensive understanding of the relative competitiveness of tabu search in solving the two-stage hybrid flowshop problem.

In order to effectively cover the research space of the two-stage hybrid flowshop problem, the tabu search algorithm relies on two key components: neighborhood operators and tabu list management. Neighborhood operators are used to generate new candidate solutions by making incremental changes to the current solution. These operators allow tabu search to explore the solution space more thoroughly, and can be designed to prioritize certain types of changes over others. The management of tabu lists is another critical component of tabu search. The tabu list is a memory structure that keeps track of recently visited solutions and prevents the algorithm from revisiting those solutions in the immediate future. By managing the tabu list, tabu search can effectively balance exploration and exploitation, avoiding getting stuck in local optima and continuing to search for better solutions. These two components work together to allow tabu search to systematically and effectively search the solution space of the two-stage hybrid flowshop problem.

8.1. Neighborhood Operators

For our proposed tabu search method, we carefully selected and implemented three different neighborhood operators:

8.1.1. Operator ($NeighborSwap$)

The first operator is the swap operator; the idea is to swap two adjacent jobs. The set of neighbor solutions generated by this operator, and which constitutes the neighborhood of the starting solution S_0, is bounded by $(n_{1,p} - 1)$ neighbors.

Example

For $n = 4$ and $S_0 = \prec J_2J_3J_1J_4 \succ$, we have a neighborhood $V_{S_0} = \{\ S'_1 = \prec J_3J_2J_1J_4 \succ$, $S'_2 = \prec J_2J_1J_3J_4 \succ, S'_3 = \prec J_2J_3J_4J_1 \succ \}$.

The advantage of this operator is that it allows for the exploration of a large portion of the search space while preserving the initial structure of the starting solution. This is because it only switches the position of two jobs within the same machine, thus it prevents the destruction of the starting solution's structure by slightly modifying it at each step.

However, it should be noted that the swap operator may not be sufficient on its own to fully explore the search space and identify the optimal solution. Therefore, it is often used in conjunction with other neighborhood operators, such as the insertion and inversion operators, to ensure a more comprehensive search of the solution space.

8.1.2. Operator ($NonNeighborSwap$)

The idea of this operator is to swap two jobs which are not necessarily neighbors. the $NonNeighborSwap$ operator is designed to work with non-adjacent elements. This approach allows the algorithm to explore a wider range of potential solutions that may be overlooked by more traditional operators that are limited to adjacent jobs. This operator gives a neighborhood of $\frac{n_{1,p}(n_{1,p}-1)}{2}$ feasible solutions.

Example :

For $n = 4$ and $S_0 = \prec J_2J_3J_1J_4 \succ$ we have a neighborhood $V_{S_0} = \{\ S'_1 = \prec J_3J_2J_1J_4 \succ,\ S'_2 = \prec J_1J_3J_2J_4 \succ,\ S'_3 = \prec J_4J_3J_1J_2 \succ,\ S'_4 = \prec J_2J_1J_3J_4 \succ,\ S'_5 = \prec J_2J_4J_1J_3 \succ,\ S'_6 = \prec J_2J_3J_4J_1 \succ \}$.

Note that the neighborhood generated by the $NeighborSwap$ operator is entirely contained in the neighborhood generated by the operator $NonNeighborSwap$.

8.1.3. Operator ($InsrtShift$)

The main idea of this operator is to insert a job at a given position and to shift the other jobs while keeping their starting order, i.e., this operator involves removing a job from its current position and inserting it into a different position within the same machine. This can be useful for exploring new solutions and potentially improving the overall quality of the generated sequence.

The $InsrtShift$ operator gives a neighborhood of $n_{1,p}(n_{1,p} - 2) + 1$ feasible solutions.

Example

For $n = 3$ and $S_0 = \prec J_1J_2J_3 \succ$, the neighbourhood of S_0 is :
$V_{S_0} = \{\ S'_1 = \prec J_2J_1J_3 \succ,\ S'_2 = \prec J_3J_1J_2 \succ,\ S'_3 = \prec J_1J_3J_2 \succ,\ S'_4 = \prec J_2J_3J_1 \succ \}$.

Given the particularity of our problem and the fact that a starting solution contains P sub-sequences, the number of solutions generated by these different change operators will decrease since we only swap jobs that belong to the same type on the first stage.

We note that one of the key advantages of using neighborhood operators in the tabu search algorithm is that they allow for small modifications to be made to the starting solution at each step, without completely destroying its underlying structure. This is particularly important in the context of the two-stage hybrid flowshop problem, where the starting solution is often already a good approximation of the optimal solution.

Another advantage of these operators is that they are relatively easy to implement and computationally efficient. They can be applied to any pair of jobs within the same machine, and the resulting solutions can be quickly evaluated to determine their makespan. This makes these operators a good choice for use in large-scale optimization problems such as our studied problem, where computational efficiency is critical. Table 2 represents the number of generated solutions by operators.

Table 2. Number of solutions generated per operator.

Operator	Number of Solutions
$NeighborSwap$	$\sum_{p=1}^{P}(n_{1,p} - 1)$
$NonNeighborSwap$	$\sum_{p=1}^{P} \frac{n_{1,p}(n_{1,p}-1)}{2}$
$InsrtShift$	$\sum_{p=1}^{P} n_{1,p}(n_{1,p} - 2) + 1$

8.2. Management of Tabu Lists

In addition to the neighborhood operators, our tabu search method also places great importance on the management of tabu lists. The tabu list is a critical component of the algorithm, as it stores information about previously visited solutions and helps guide the search towards new and unexplored regions of the solution space. Specifically, our approach employed two distinct methods for managing the tabu list and it involves capturing and storing the recent movements and objective function values.

8.2.1. Movement Restriction Mode

In order to efficiently manage the search space, our algorithm records the jobs whose permutation led to the current solution at each iteration. This information is then used to update the tabu list, which keeps track of recent movements to avoid revisiting previously explored solutions. By capturing this trace of movements, our algorithm can more effectively explore the search space and avoid getting stuck in local optima. We illustrate the approach with the following example.

Given an initial solution $S_0 = \prec J_3 J_2 J_1 J_4 \succ$, if we execute the change operator $NeighborSwap$ (swap two neighboring jobs) and retain the neighboring solution $S_1' = \prec J_2 J_3 J_1 J_4 \succ$, the tabu list of length $L_{liste} = 3$ being initially empty will be filled as follows :

$$tabu_{mvt}\begin{pmatrix} 3 & . & . \\ 2 & . & . \end{pmatrix}.$$

For the generation of the next neighbors, if the two jobs to be swapped are present in the list then the change is prohibited. Once all the cells of the list are filled, the oldest element is overwritten. Note that, for the operator $InsrtShift$, the couple (job, position) is registered.

8.2.2. Objective Function Restriction Mode

Our tabu search method updates the tabu list by taking into account objective function values. As multiple solutions can have the same makespan value, we used the makespan as a parameter to prohibit the adoption of a neighbor if its makespan is already in the tabu list. By doing so, we effectively prevent the adoption of neighboring solutions that have makespan values included in the tabu list. This approach helps in exploring new solutions while avoiding cycling through solutions with similar makespan values.

As illustration, let S_1' be a selected neighbor with a makespan $= 560$. The tabu list is then:

$$tabu_{makespan} = (560\,.\ .\ .\ .).$$

In order to prevent the algorithm from revisiting previously explored solutions, any neighbor with a makespan equal to 560 will be excluded from consideration as long as this value is present in the tabu list. To update the tabu list, we utilized a simple overwrite approach where the oldest element is replaced with the newest element as new solutions are generated.

By combining these techniques, our proposed tabu search method is able to more effectively navigate the search space and converge to high-quality solutions in a computationally efficient manner.

8.3. Diversification

During the initial tests for our tabu search, we encountered a recurring issue every 40 to 60 iterations, where the same solutions would reappear, accompanied by a stagnation of the makespan, such that the value of C_{max} remained constant from the initial iterations onwards. To tackle this issue, we attempted to introduce a diversification strategy after a predetermined number of iterations, aimed at discovering previously unexplored regions of the search space and thereby avoiding getting trapped in local optimum. This was achieved through the introduction of random permutations of the current solution, which added an element of stochasticity to the algorithm and allowed for a more thorough exploration of the solution space. By incorporating this diversification strategy, we were able to mitigate the issue of the algorithm cycling through previously visited solutions and improve the overall quality of the solutions obtained.

8.4. Selected Versions of the Tabu Search

The following Algorithm 2 describes the retained tabu search procedure.

Algorithm 2 Tabu Search Procedure

```
Let $S_{current}$ be a starting solution.
Set $Nbr_{iteration}$;
Initialize $Best_{C_{max}}$, $Best_{Solution}$, $tabulist$ ;
$i \leftarrow 1$;
while ($i \leq Nbr_{iteration}$) do
    Find $S_{next}$ such that ( $C_{max}(S_{next}) \leq C_{max}(S_{neighbor})$ ) $\forall\ S_{neighbor} \in V_{S_{current}} \setminus tabulist$;
    Update $tabulist$;
    $S_{courante} \longleftarrow S_{next}$ ;
    if $C_{max}(S_{next}) \leq C_{max}(S_{current})$ then
        $Best_{C_{max}} \longleftarrow C_{max}(S_{next})$;
        $Best_{Solution} \longleftarrow S_{next})$;
        $i \leftarrow i + 1$ ;
    end if
end while
return $Best_{Solution}$.
```

We implement five distinct versions of the tabu search algorithm, each incorporating a unique combination of neighborhood operators and tabu list management strategies. An overview of these five versions is presented in Table 3.

Table 3. Tabu search versions.

Tabu List/Operators	$NeighborSwap$	$NonNeighborSwap$	$InsrtShift$
Movement restriction mode	TS1	TS2	TS4
Makespan restriction mode	.	TS3	TS5

For example, the neighborhood of the first version ($TS1$) is generated using the change operator $NeighborSwap$ and the tabu list is managed using the method of restriction of the movements already made.

Stop Criteria

The stop criteria in tabu search is a crucial aspect of the algorithm and must be well defined to achieve the desired results. In our study, we set two stop criteria for the algorithm. The first criterion is based on the number of iterations $Iter$, where we stop the process if it reaches a predefined limit of 200 iterations. This limit is determined by empirical analysis and ensures that the algorithm does not run indefinitely, especially in cases where it is unable to find an optimal solution within a reasonable time.

The second stop criterion is based on the quality of the obtained solution. Specifically, we compared the makespan value of the obtained solution to a known lower bound. If the makespan value is equal to the lower bound, the algorithm is stopped since it has achieved an optimal solution. This approach was motivated by the fact that finding the optimal solution is often the primary objective of optimization problems, and the lower bound provides a reference point to evaluate the quality of the solution.

By combining these two stop criteria, we can ensure that the algorithm terminates within a reasonable time while also still allowing for sufficient opportunity to obtain optimal solutions.

9. Computational Results

Before presenting the computational results of the exact and heuristic solutions, we will provide a detailed description of the classes of instances and the various configurations of the workshops that were considered, such as problem size, number of machines, and other relevant factors.

9.1. Classes of Instances

In order to test the different approaches already presented, we generated several random instances belonging to different classes, and sizes ranging from $n = 20$ jobs to $n = 500$ jobs. For each size, 20 instances were generated and the jobs were (except for some classes) equally distributed between the different types of jobs.

We tested essentially the configuration where we had two machines on each stage (i.e., $P = M = 2$). For the exact and meta heuristic simulation, we tested a second configuration with $P = 3$ and $M = 4$. The operating times of the different jobs are randomly generated following a uniform distribution.

For the classes Cl_1 and Cl_2, the process times a_i and b_i were generated in the same range: $[1, 20]$ for Cl_1 and $[1, 100]$ for Cl_2.

For Cl_3, the process times a_i and b_i were generated in $[1, 100]$ and $[1, m \times 100]$, respectively.

For the class Cl_4, the process times were generated in the $[1, 20]$ but the jobs are not equally distributed on the different machines of the two stages. Indeed, and in order to create a bottleneck route, we tried to load a machine on each stage by making $\frac{n}{2}$ jobs pass on two randomly chosen machines (one on the first and the other on the second stage).

The idea of the Cl_5 class is similar to that in Cl_4, except that it loads a route not with a large number of jobs but with very large process times compared to the others. These process times are generated in $[1, 100]$ for $\frac{n}{2}$ jobs, and in $[1, p \times 100]$ and $[1, m \times 100]$, respectively, for the rest of the jobs on two machines randomly chosen on the first and second stages.

The generated classes are summarized in Table 4.

Table 4. Classes of instances.

Class	a_i	b_i
Cl_1	[1, 20]	[1, 20]
Cl_2	[1, 100]	[1, 100]
Cl_3	[1, 100]	$[1, m \times 100]$
Cl_4	[1, 20]	[1, 20]
Cl_5	[1, 100], $[1, p \times 100]$	[1, 100], $[1, m \times 100]$

In order to maintain consistency and accuracy throughout the experimentation process, all procedures were implemented in the programming language C++. The use of this programming language allowed for efficient and robust implementation of the various metaheuristics and algorithms used in this study. Additionally, all tests and computations were performed on a processor with a clock speed of 3.20 GHz. It is important to note that the choice of hardware can have a significant impact on the performance and results obtained in computational experiments. Therefore, all tests were performed on the same hardware to ensure that the results obtained were consistent and could be accurately compared.

9.2. Exact Resolution Results

In this study, we implemented a constraint programming technique using the Clpex Cp Optimizer solver. This solver was specifically designed for solving constraint satisfaction problems and provides efficient and powerful algorithms for finding optimal solutions.

Table 5 presents a summary of the average, minimum, and maximum computation time for the 20 generated instances, for each configuration and problem size. Additionally, it indicates the percentage of instances that were successfully solved. The tests were conducted using instances from the Cl_1 class. It is worth mentioning that a time limit of 60 s is set, and any instance that remains unsolved beyond this limit is deemed unsolvable.

Table 5 shows that the constraint programming model was very efficient and was able to solve problems up to $n = 500$ jobs, which is a significant size compared to instances solved by other exact solution approaches. Nevertheless, we notice that, for sizes of $n = 200$ and $n = 500$ jobs, our model failed to solve some instances.

Table 5. Resolution results for constraint programming.

		$F2(D^2, D^2)\|\|C_{max}$	
n	**Average Computation Time in *s***	**{Min, Max} Times in *s***	**% Resolution**
20	0.5	{0.1 , 1}	100%
50	0.5	{0.1 , 2}	100%
100	4	{3 , 6}	100%
200	10	{5 , 17}	94%
500	20	{18 , 52}	95%
		$F2(D^3, D^4)\|\|C_{max}$	
20	0.5	{0.1 , 1}	100%
50	0.5	{0.1 , 2}	100%
100	3	{3 , 5}	100%
200	8	{5 , 14}	98%
500	22	{18 , 43}	98%

It is interesting to note that the solver is able to solve larger problems when there are more machines available. This is because the number of possible solutions decreases with an increase in the number of machines, which is given by $(\frac{n}{max(P,M)}!)^{max(P,M)}$ for P machines on the first stage and M machines on the second stage for jobs equally distributed on the different machines. This reduction in the search space makes it easier for the solvers to find an optimal solution.

9.3. Heuristics Testing

We have proposed basic lower bounds and other lower bounds based on the Johnson's rule. These lower bounds have been presented for this same problem in [40]. The tests conducted on these lower bounds have shown great promise, making them suitable as a reference for evaluating the performance of the heuristics and metaheuristics proposed in this article.

Table 6 presents the deviations observed for the heuristic (PSI and PSI'-based variants) from the best lower bound. Table 7, on the other hand, presents the number of cases in which the heuristic approach achieves optimality, as indicated by its ability to return a solution that matches with the lower bound.

Table 6 shows that the heuristic approach achieved complete success in solving all instances for some classes. Even in cases where optimal solutions were not available, the PSI-based variant displayed an average worst-case deviation of 0.50% from the best lower bound, while the PSI'-based variant exhibited an average worst-case deviation of 2.45%. Moreover, the difference was less than 0.01% for the majority of tested instances in both versions. Notably, for the Cl_3 class, the PSI'-based variant , which incorporates the load of the first stage, outperformed the other heuristic. This supports the conclusion that, for this type of class, the computation of the priority index should include the process times of the first stage.

In contrast, both heuristic versions tended to become more efficient as the problem size increased. These results corroborate the findings reported in [41,42] for a simpler configuration.

Table 6. JIPA results, Workshop $F2(D^2, D^2)||C_{max}$.

n	20	50	100	150	200	500
			Cl_1			
PSI	0.50	0.05	0.00	0.00	0.00	0.00
PSI'	2.00	0.30	0.00	0.05	0.00	0.00
			Cl_2			
PSI	0.40	0.00	0.00	0.00	0.00	0.00
PSI'	1.85	0.30	0.00	0.00	0.00	0.00
			Cl_3			
PSI	0.15	0.00	0.00	0.00	0.00	0.00
PSI'	0.00	0.00	0.00	0.00	0.00	0.00
			Cl_4			
PSI	0.00	0.00	0.00	0.00	0.00	0.00
PSI'	0.45	0.00	0.00	0.00	0.00	0.00
			Cl_5			
PSI	0.35	0.00	0.00	0.00	0.00	0.00
PSI'	1.35	0.35	0.00	0.00	0.00	0.00

Table 7. JIPA's Optimal solutions, Workshop $F2(D^2, D^2)||C_{max}$.

n	20	50	100	150	200	500
			Cl_1			
PSI	19	19	**20**	**20**	19	**20**
PSI'	11	18	**20**	19	16	**20**
			Cl_2			
PSI	16	17	19	**20**	19	20
PSI'	4	14	12	19	17	17
			Cl_3			
PSI	11	16	18	19	19	11
PSI'	15	16	16	17	17	13
			Cl_4			
PSI	19	19	**20**	**20**	**20**	**20**
PSI'	16	19	19	19	**20**	**20**
			Cl_5			
PSI	16	19	**20**	17	18	19
PSI'	9	11	14	15	16	14

Table 7 shows that both variants of the Priority Score Index (PSI and PSI') are able to achieve optimal solutions most of the time. Indeed, the PSI gives an average of 15.6 optimal solutions for $n = 20$ jobs and 18.2 for $n = 500$ jobs on the 20 instances tested for all classes of instances. The heuristic $PSI4$ gives an average of 10.4 optimal solutions for $n = 20$ jobs and 16.6 for $n = 500$ jobs.

The results presented in Table 7 indicate that both variants of the Priority Score Index (PSI and PSI') demonstrate a high level of effectiveness in generating optimal solutions. Specifically, for all classes of instances tested, the average number of optimal solutions generated by PSI is 15.6 for 20 jobs and 18.2 for 500 jobs across the 20 instances tested. Meanwhile, the PSI′-based variant generates an average of 10.4 optimal solutions for 20 jobs and 16.6 optimal solutions for 500 jobs. The results of the experiment indicate that the JIPA heuristic is a dependable and effective approach for solving the given problem. These findings imply that JIPA can be considered a promising solution for scheduling problems, which can potentially be implemented in real-world applications.

9.4. Meta Heuristic Tests

Our study evaluated different versions of tabu search on five classes of instances with four different problem sizes ($n \in \{20, 50, 150, 300\}$). To assess their performance, we used random starting solutions and a tabu list of length 10 to avoid revisiting recently explored solutions. We also implemented a diversification mechanism to prevent the algorithm from getting stuck in a suboptimal solution. The diversification process was triggered when we observed a stagnation of the makespan value for 15 consecutive iterations.

It is important to note that these testing protocols were designed to ensure the reliability and robustness of the results. The reported findings regarding the effectiveness of the tabu search heuristics were obtained through a thorough experimental evaluation. Therefore, these results provide valuable insights into the potential applications of tabu search for solving complex scheduling problems.

In the first configuration, where $P = M = 2$, we evaluated the performance of the $TS2$ version (which employs the $NonNeighborSwap$ change operator and the movement-based mode of the tabu list management, as shown in Table 3) across 20 instances. Tables 8–12 present the results of this evaluation, indicating that the $TS2$ version produced 20 optimal solutions with 0% deviation from the lower bound on all the 20 instances tested ($C_{max}(\pi^{TS2}) = LB$).

The procedures $TS3$ and $TS5$ performed equally well with a maximum deviation recorded that does not exceed 0.2% from the lower bound. On the other hand, procedure $TS1$ did not show the same performance with a deviation that amounts to 9.3% from the lower bound and 0 optimal solution obtained. This can be explained by the nature of the neighborhood operator used by this procedure, namely the $NeighborSwap$ operator, which does not allow an important exploration of the search space and is restricted to a neighborhood limited to $\sum_{p=1}^{P}(n_{1,p} - 1)$ (See Table 2).

The evaluation of the procedures $TS3$ and $TS5$ revealed that their performance was comparable, with a maximum deviation from the lower bound of no more than 0.2%. However, procedure $TS1$ did not perform as well, with a deviation from the lower bound of 9.3% and no optimal solutions obtained.

The reason behind the suboptimal performance of the $TS1$ procedure can be attributed to the use of the $NeighborSwap$ operator in its neighborhood search. This operator only permits local moves in the solution space and is limited to a small neighborhood size ($\sum_{p=1}^{P}(n_{1,p} - 1)$), as indicated by Table 2. Consequently, the procedure's ability to explore the search space is limited and it may become trapped in a suboptimal region of the solution space. This limitation may prove insufficient for solving more complex optimization problems where larger, non-local moves are necessary to escape local optima. As a result, it may be necessary to employ more advanced change operators that can facilitate larger and more diverse moves in the solution space to improve the optimization performance.

Workshop 1 : $F2(D^2, D^2)||C_{max}$

Table 8. Meta heuristics results for Cl_1, $F2(D^2, D^2)||C_{max}$.

	Class Cl_1							
	$n = 20$		$n = 50$		$n = 150$		$n = 300$	
	Deviation	*Opt*	*Deviation*	*Opt*	*Deviation*	*Opt*	*Deviation*	*Opt*
TS1	6.95	4	8.96	0	5.27	0	4.56	0
TS2	0.00	20	0.00	19	0.00	18	0.00	20
TS3	0.05	19	0.00	19	0.00	18	0.00	19
TS4	0.61	15	0.92	6	0.82	5	0.15	5
TS5	0.00	19	0.00	19	0.00	18	0.00	16

Table 9. Meta heuristics results for Cl_2, $F2(D^2, D^2)||C_{max}$.

	Class Cl_2							
	$n = 20$		$n = 50$		$n = 150$		$n = 300$	
	Deviation	*Opt*	*Deviation*	*Opt*	*Deviation*	*Opt*	*Deviation*	*Opt*
TS1	6.71	2	9.52	1	5.17	0	4.56	1
TS2	0.00	20	0.00	19	0.00	18	0.00	17
TS3	0.21	15	0.00	17	0.00	16	0.00	16
TS4	0.87	12	0.71	5	0.41	4	0.15	4
TS5	0.05	18	0.00	18	0.00	16	0.00	12

Table 10. Meta heuristics results for Cl_3, $F2(D^2, D^2)||C_{max}$.

	Class Cl_3							
	$n = 20$		$n = 50$		$n = 150$		$n = 300$	
	Deviation	*Opt*	*Deviation*	*Opt*	*Deviation*	*Opt*	*Deviation*	*Opt*
TS1	1.59	1	1.18	1	0.31	1	0.05	0
TS2	0.00	20	0.00	19	0.00	20	0.15	17
TS3	0.00	20	0.00	20	0.00	20	0.15	17
TS4	0.00	17	0.00	17	0.00	11	0.05	19
TS5	0.00	20	0.00	19	0.00	19	0.00	20

Table 11. Meta heuristics results for Cl_4, $F2(D^2, D^2)||C_{max}$.

	Class Cl_4							
	$n = 20$		$n = 50$		$n = 150$		$n = 300$	
	Deviation	*Opt*	*Deviation*	*Opt*	*Deviation*	*Opt*	*Deviation*	*Opt*
TS1	3.18	5	3.07	2	2.25	1	1.38	1
TS2	0.00	20	0.00	19	0.00	20	0.00	20
TS3	0.00	18	0.00	18	0.00	18	0.00	20
TS4	0.05	18	0.00	17	0.00	19	0.00	19
TS5	0.00	20	0.00	19	0.00	20	0.00	20

Table 12. Meta heuristic results for Cl_5, $F2(D^2, D^2)||C_{max}$.

	Class Cl_5							
	$n = 20$		$n = 50$		$n = 150$		$n = 300$	
	Deviation	*Opt*	*Deviation*	*Opt*	*Deviation*	*Opt*	*Deviation*	*Opt*
TS1	6.81	2	6.49	0	2.77	0	2.82	0
TS2	0.00	20	0.00	17	0.00	17	0.20	15
TS3	0.10	15	0.00	19	0.00	14	0.10	15
TS4	0.25	15	0.71	9	0.00	6	0.00	5
TS5	0.00	20	0.00	17	0.00	19	0.10	15

Workshop 2 : $F2(D^3, D^4)||C_{max}$

For the second considered configuration, where $P = 3$ and $M = 4$, Tables 13–17 show that the procedure $TS1$ failed to perform with a deviation that reaches 18.7% for the instance class Cl_5 and $n = 20$ jobs. Other procedures, such as $TS2$, $TS3$, and $TS5$, performed well with a maximum deviation recorded of 0.65% for $TS2$ with instance class Cl_1 and $n = 20$ jobs.

Table 13. Meta heuristics results for Cl_1, $F2(D^3, D^4)||C_{max}$.

	Class Cl_1							
	$n = 20$		$n = 50$		$n = 150$		$n = 300$	
	Deviation	*Opt*	*Deviation*	*Opt*	*Deviation*	*Opt*	*Deviation*	*Opt*
TS1	15.60	1	10.38	0	4.50	0	3.58	0
TS2	0.67	16	0.51	13	0.10	15	0.00	18
TS3	0.41	16	0.10	17	0.00	19	0.00	20
TS4	1.74	10	0.20	11	0.10	11	0.00	10
TS5	0.10	18	0.00	20	0.00	20	0.00	19

Table 14. Meta heuristics results for Cl_2, $F2(D^3, D^4)||C_{max}$.

	Class Cl_2							
	$n = 20$		$n = 50$		$n = 150$		$n = 300$	
	Deviation	*Opt*	*Deviation*	*Opt*	*Deviation*	*Opt*	*Deviation*	*Opt*
TS1	13.30	1	10.49	1	6.10	0	3.38	0
TS2	0.46	16	0.51	12	0.15	13	0.00	12
TS3	0.15	19	0.05	15	0.00	14	0.00	14
TS4	0.97	12	0.10	13	0.15	5	0.00	5
TS5	0.15	19	0.00	20	0.00	17	0.00	20

Table 15. Meta heuristics results for Cl_3,$F2(D^3, D^4)||C_{max}$.

	Class Cl_3							
	$n = 20$		$n = 50$		$n = 150$		$n = 300$	
	Deviation	*Opt*	*Deviation*	*Opt*	*Deviation*	*Opt*	*Deviation*	*Opt*
TS1	2.05	6	0.82	3	0.46	0	0.05	0
TS2	0.00	20	0.00	20	0.00	20	0.15	17
TS3	0.00	20	0.00	20	0.00	20	0.15	17
TS4	0.05	18	0.00	17	0.00	15	0.15	15
TS5	0.00	20	0.00	20	0.00	20	0.15	16

Table 16. Meta heuristics results for Cl_4, $F2(D^3, D^4)||C_{max}$.

	Class Cl_4							
	$n = 20$		$n = 50$		$n = 150$		$n = 300$	
	Deviation	*Opt*	*Deviation*	*Opt*	*Deviation*	*Opt*	*Deviation*	*Opt*
TS1	7.48	2	2.61	2	1.23	1	0.36	2
TS2	0.00	20	0.00	17	0.00	17	0.00	19
TS3	0.15	16	0.00	19	0.00	19	0.00	20
TS4	0.00	20	0.00	18	0.00	19	0.00	19
TS5	0.00	20	0.00	20	0.00	20	0.00	20

Table 17. Meta heuristics results for Cl_5, $F2(D^3, D^4)||C_{max}$.

	Class Cl_5							
	$n = 20$		$n = 50$		$n = 150$		$n = 300$	
	Deviation	*Opt*	*Deviation*	*Opt*	*Deviation*	*Opt*	*Deviation*	*Opt*
TS1	19.09	2	12.16	0	9.02	0	6.71	0
TS2	0.00	20	0.05	17	0.05	19	0.05	16
TS3	0.15	18	0.00	20	0.00	19	0.00	15
TS4	3.69	11	0.46	14	0.00	4	0.05	7
TS5	0.05	19	0.00	20	0.00	20	0.00	16

We also note that, for all problem sizes and for all instance classes, we have an average of 19.3 optimal solutions among the 20 instances tested (with 16 optimal solutions at worst case for Cl_3, $TS5$, $n = 300$) which represents an overall performance of 96.5% for a workshop of $P = 3$ and $M = 4$.

It is worth noting that the choice of the neighborhood operator can have a significant impact on the performance of the tabu search procedure. In particular, our experiments show that the $NonNeighborSwap$ operator is well-suited for the movement-based management mode employed by the $TS2$ procedure, while the $InsrtShift$ operator is better suited for the makespan-based management mode used by the $TS5$ procedure (as illustrated in Table 3). This suggests that the choice of neighborhood operator should be carefully considered in the design of a tabu search algorithm, and should be tailored to the specific search strategy employed by the algorithm. Furthermore, it highlights the importance of investigating the compatibility between different components of a metaheuristic algorithm, as this can significantly affect its performance.

The influence of instance diversity on the performance of different tabu search versions was investigated. Our results show that changes in process times and job distribution did not have a significant impact on the performance of the tabu search procedures. However, altering the workshop configuration had a noticeable effect on the performance of these procedures. For instance, in the case of class Cl_1 and $n = 20$ jobs, procedure $TS4$ saw an increase in deviation from the lower bound from 0.60% to 1.70%, while procedure $TS2$ increased from 0.00% to 0.65%. These findings suggest that diversification techniques may be particularly useful for optimizing scheduling problems in which the workshop configuration is subject to change.

10. Conclusions

In this study, we conducted a comprehensive analysis of the two-stage flow shop problem with dedicated machines, specifically the $F2(D^P, D^M)||C_{max}$ problem. Our research explored fundamental properties, including elimination rules, and investigated various resolution techniques.

For exact resolution, we employed a constraint programming approach, which proved to be a powerful tool for solving the problem efficiently. However, we also identified limitations in certain instances, indicating the need for further research and improvement in this area.

Furthermore, we explored approximate solution methods, including heuristics and variations of the tabu search metaheuristic. Our simulation results demonstrated the effectiveness of these approaches, particularly in the context of constraint programming and the tabu search procedure.

While our findings highlight the potential of constraint programming and the tabu search metaheuristic, addressing the identified limitations and enhancing the metaheuristic approach should be a priority. This can involve improving constraint programming techniques, exploring advanced metaheuristic algorithms, and integrating machine learning and artificial intelligence approaches. These research directions aim to enhance the resolution of the two-stage flow shop problem with dedicated machines, ultimately advancing scheduling techniques in real-life industrial scenarios.

In conclusion, this study contributes to the understanding of the two-stage flow shop problem by showcasing the strengths of constraint programming and the tabu search metaheuristic. Our findings provide a foundation for future investigations to build upon these methodologies and advance the field further.

Author Contributions: Conceptualization and methodology, M.K.H., H.H. and N.D.; software, M.K.H.; validation, H.H. and N.D.; formal analysis, M.K.H.; investigation, M.K.H. and H.H.; data curation, M.K.H. and H.H.; writing—original draft preparation, M.K.H.; writing—review and editing, M.K.H.; visualization, M.K.H.; supervision, H.H. and N.D. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: The data is available upon request.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Esswein, C. Un Apport de Flexibilité Séquentielle pour L'ordonnancement Robuste. Ph.D. Thesis, Tours, France, 2003.
2. Grabowski, J.; Pempera, J. Sequencing of jobs in some production system. *Eur. J. Oper. Res.* **2000**, *125*, 535–550. [CrossRef]
3. Jin, Z.; Ohno, K.; Ito, T.; Elmaghraby, S. Scheduling hybrid flowshops in printed circuit board assembly lines. *Prod. Oper. Manag.* **2002**, *11*, 216–230. [CrossRef]
4. Lin, H.-T.; Liao, C.-J. A case study in a two-stage hybrid flow shop with setup time and dedicated machines. *Int. J. Prod. Econ.* **2003**, *86*, 133–143. [CrossRef]
5. Hadda, H. Contribution à L'étude et à la Résolution des Problèmes D'ordonnancement de Flow Shops D'assemblage et de Flow Shops Hybrides à Machines Dédiées. Ph.D. Thesis, École Nationale d'ingénieurs de Tunis, Tunis, Tunisia, 2009.
6. Tomazella, C.P.; Nagano, M.S. A comprehensive review of branch-and-bound algorithms: Guidelines and directions for further research on the flowshop scheduling problem. *Expert Syst. Appl.* **2020**, *158*, 113556. [CrossRef]
7. Ignall, E.; Schrage, L. Application of the branch and bound technique to some flow-shop scheduling problems. *Oper. Res.* **1965**, *13*, 400–412. [CrossRef]
8. Besbes, W.; Loukil, T.; Teghem, J. A two-stage flow shop with parallel dedicated Machines. In Proceedings of the 8th International Conference of Modeling and Simulation—MOSIM, Hammamet, Tunisia, 10–12 May 2010.
9. Lei, D.; Guo, X. A parallel neighborhood search for order acceptance and scheduling in flow shop environment. *Int. J. Prod. Econ.* **2015**, *165*, 12–18. [CrossRef]
10. Ramezanian, R. Milp formulation and genetic algorithm for non-permutation flow shop scheduling problem with availability constraints. *Int. J. Appl. Oper. Res.* **2014**, *4*, 11–26.
11. Zeballos, L.J.; Castro, P.M.; Mendez, C.A. Integrated constraint programming scheduling approach for automated wet-etch stations in semiconductor manufacturing. *Ind. Eng. Chem. Res.* **2011**, *50*, 1705–1715. [CrossRef]
12. Samarghandi, H.; Behroozi, M. On the exact solution of the no-wait flow shop problem with due date constraints. *Comput. Oper. Res.* **2017**, *81*, 141–159. [CrossRef]
13. Samarghandi, H. Minimizing the makespan in a flow shop environment under minimum and maximum time-lag constraints. *Comput. Ind. Eng.* **2019**, *136*, 614–634. [CrossRef]
14. Said, A.B.; Mouhoub, M. A constraint satisfaction problem (csp) approach for the nurse scheduling problem. In Proceedings of the 2022 IEEE Symposium Series on Computational Intelligence (SSCI), Singapore, 4–7 December 2022; pp. 790–795.
15. Gehring, M.; Volk, R.; Braun, N.; Schultmann, F. Scheduling projects with converging and diverging material flows using ibm ilog cp optimizer—An experimental performance analysis. In *International Conference on Operations Research*; Springer: Berlin/Heidelberg, Germany, 2021; pp. 341–346.
16. Nishikawa, H.; Shimada, K.; Taniguchi, I.; Tomiyama, H. A constraint programming approach to scheduling of malleable tasks. *Int. J. Netw. Comput.* **2019**, *9*, 131–146. [CrossRef] [PubMed]
17. Müller, D.; Müller, M.G.; Kress, D.; Pesch, E. An algorithm selection approach for the flexible job shop scheduling problem: Choosing constraint programming solvers through machine learning. *Eur. J. Oper. Res.* **2022**, *302*, 874–891. [CrossRef]
18. Yunusoglu, P.; Topaloglu Yildiz, S. Constraint programming approach for multi-resource-constrained unrelated parallel machine scheduling problem with sequence-dependent setup times. *Int. J. Prod. Res.* **2022**, *60*, 2212–2229. [CrossRef]
19. Herrmann, J.W.; Lee, C.-Y. *Three-Machine Look-Ahead Scheduling Problems*; Department of Industrial and Systems Engineering, University of Florida: Gainesville, FL, USA, 1992.
20. Johnson, S.M. Optimal two-and three-stage production schedules with setup times included. *Nav. Res. Logist. Q.* **1954**, *1*, 61–68. [CrossRef]
21. Conway, R.W.; Maxwell, W.L.; Miller, L.W. *Theory of Scheduling*; Addison-Wesley Publishing Company: Boston, MA, USA, 1967.
22. Xiao, W.; Hao, P.; Zhang, S.; Xu, X. Hybrid flow shop scheduling using genetic Algorithms. In Proceedings of the 3rd World Congress on Intelligent Control and Automation (Cat. No. 00EX393), Hefei, China, 26 June–2 July 2000; Volume 1, pp. 537–541.
23. Haouari, M.; M'Hallah, R. Heuristic algorithms for the two-stage hybrid flowshop problem. *Oper. Res. Lett.* **1997**, *21*, 43–53. [CrossRef]
24. Hajji, M.K.; Hadda, H. Contribution à la Résolution des Problèmes de Flow Shops avec Mechines Dédiées, Dates de Disponiblité et Délais de Livraison. Master's Thesis, Institut Supèrieur de Transport et de la Logistique, Sousse, Tunisia, 2012.
25. Hajji, M.K.; Hadda, H.; Dridi, N. The two-stage hybrid flow shop problem with dedicated machines under release dates and delivery times. *Int. J. Adv. Oper. Manag.* **2015**, *7*, 300–316. [CrossRef]
26. Nowicki, E.; Smutnicki, C. The flow shop with parallel machines: A tabu search approach. *Eur. J. Oper. Res.* **1998**, *106*, 226–253. [CrossRef]
27. Wardono, B.; Fathi, Y. A tabu search algorithm for the multi-stage parallel machine problem with limited buffer capacities. *Eur. J. Oper. Res.* **2004**, *155*, 380–401. [CrossRef]
28. Jin, Z.; Yang, Z.; Ito, T. Metaheuristic algorithms for the multistage hybrid flowshop scheduling problem. *Int. J. Prod. Econ.* **2006**, *100*, 322–334. [CrossRef]

29. Grabowski, J.; Wodecki, M. A very fast tabu search algorithm for the permutation flow shop problem with makespan criterion. *Comput. Oper. Res.* **2004**, *31*, 1891–1909. [CrossRef]
30. Chen, L.; Bostel, N.; Dejax, P.; Cai, J.; Xi, L. A tabu search algorithm for the integrated scheduling problem of container handling systems in a maritime terminal. *Eur. J. Oper. Res.* **2007**, *181*, 40–58. [CrossRef]
31. Umam, M.S.; Mustafid, M.; Suryono, S. A hybrid genetic algorithm and tabu search for minimizing makespan in flow shop scheduling problem. *J. King Saud Univ. Comput. Inf. Sci.* **2022**, *34*, 7459–7467. [CrossRef]
32. Xu, W.; Hu, Y.; Luo, W.; Wang, L.; Wu, R. A multi-objective scheduling method for distributed and flexible job shop based on hybrid genetic algorithm and tabu search considering operation outsourcing and carbon emission. *Comput. Ind. Eng.* **2021**, *157*, 107318. [CrossRef]
33. Ma, Y.; He, Z.; Wang, N.; Demeulemeester, E. Tabu search for proactive project scheduling problem with flexible resources. *Comput. Oper. Res.* **2023**, *153*, 106185. [CrossRef]
34. Zhang, X.; Guo, A.; Ai, Y.; Tian, B.; Chen, L. Real-time scheduling of autonomous mining trucks via flow allocation-accelerated tabu search. *IEEE Trans. Intell. Veh.* **2022**, *7*, 466–479. [CrossRef]
35. Graham, R.L.; Lawler, E.L.; Lenstra, J.K.; Kan, A.R. Optimization and approximation in deterministic sequencing and scheduling: A survey. *Ann. Discret. Math.* **1979**, *5*, 287–326.
36. Lawler, E.L. Optimal sequencing of a single machine subject to precedence Constraints. *Manag. Sci.* **1973**, *19*, 544–546. [CrossRef]
37. Brucker, P.; Knust, S. Complexity Results for Scheduling Problems. 2009. Available online: http://www.mathematik.uni-osnabrueck.de/research/OR/class (accessed on 14 June 2023).
38. Baptiste, P.; Le Pape, C.; Nuijten, W. *Constraint-Based Scheduling: Applying Constraint Programming to Scheduling Problems*; Springer Science & Business Media: Berlin/Heidelberg, Germany, 2012; Volume 39.
39. Huang, T.-C.; Lin, B.M. Batch scheduling in differentiation flow shops for makespan minimization. *Int. J. Prod. Res.* **2013**, *51*, 5073–5082. [CrossRef]
40. Hajji, M.K.; Hadda, H.; Dridi, N. Une heuristique pour le flow shop hybride à deux étages avec machines dédiées. In Proceedings of the ROADEF 2016: 17eme Congré Annuel de le Société Française de Recherche Opérationnelle et d'Aide à la Décision, Compiègne, France, 10–12 February 2016.
41. Dridi, N.; Hadda, H.; Hajri-Gabouj, S. Méthode heuristique pour le problème de flow shop hybride avec machines dédiées. *RAIRO-Oper. Res.* **2009**, *43*, 421–436. [CrossRef]
42. Hadda, H.; Dridi, N.; Hajri-Gabouj, S. Exact resolution of the two-stage hybrid flow shop with dedicated machines. *Optim. Lett.* **2014**, *8*, 2329–2339. [CrossRef]