symmetry

MDPI

Article

Effective Two-Phase Heuristic and Lower Bounds for Multi-Stage Flexible Flow Shop Scheduling Problem with Unloading Times

Lotfi Hidri

Industrial Engineering Department, College of Engineering, King Saud University, Riyadh 11421, Saudi Arabia; lhidri@ksu.edu.sa

Abstract: This paper addresses the flexible flow shop scheduling problem with unloading operations, which commonly occurs in modern manufacturing processes like sand casting. Although only a few related works have been proposed in the literature, the significance of this problem motivates the need for efficient algorithms and the exploration of new properties. One interesting property established is the symmetry of the problem, where scheduling from the first stage to the last or vice versa yields the same optimal solution. This property enhances solution quality. Considering the problem's theoretical complexity as strongly NP-Hard, approximate solutions are preferable, especially for medium and large-scale instances. To address this, a new two-phase heuristic is proposed, consisting of a constructive phase and an improvement phase. This heuristic builds upon an existing efficient heuristic for the parallel machine-scheduling problem and extends it to incorporate unloading times efficiently. The selection of the two-phase heuristic is justified by its ability to generate high-quality schedules at each stage. Moreover, new efficient lower bounds based on estimating minimum idle time in each stage are presented, utilizing the polynomial parallel machine-scheduling problem with flow time minimization in the previous stage. These lower bounds contribute to assessing the performance of the two-phase heuristic over the relative gap performance measure. Extensive experiments are conducted on benchmark test problems, demonstrating the effectiveness of the proposed algorithms. The results indicate an average computation time of 9.92 s and a mean relative gap of only 2.80% for several jobs up to 200 and several stages up to 10.

Keywords: flexible flow shop; unloading times; lower bounds; heuristic

Citation: Hidri, L. Effective Two-Phase Heuristic and Lower Bounds for Multi-Stage Flexible Flow Shop Scheduling Problem with Unloading Times. *Symmetry* **2023**, *15*, 2005. https://doi.org/10.3390/sym15112005

Academic Editor: Guangdong Tian

Received: 26 August 2023
Revised: 13 October 2023
Accepted: 24 October 2023
Published: 31 October 2023

1. Introduction

Scheduling plays a critical role in the production planning of manufacturing systems and serves as a key determinant of success in today's competitive marketplace [1]. The flexible flow shop scheduling problem (FFSP) is prevalent in diverse industries, including the chemical industry [2], metallurgy [3], textile manufacturing [4], semiconductor production [5], logistics, paper manufacturing, and construction. An FFSP configuration comprises multiple production stages, where each stage contains several identical machines operating in parallel. Although some stages may have only a single machine, an FFSP must include at least one stage with multiple machines to qualify as an FFSP. Within the FFSP setup, the objective is to process a set of jobs while optimizing a specific objective function. It is important to note that each job must traverse through all the stages and be processed by precisely one machine at each stage. Unlike the flow shop scheduling problem (FSP), the FFSP presents a larger solution space and a more complex solving process due to the presence of parallel machines at different processing stages [6–9]. This characteristic of the FFSP introduces additional challenges in scheduling jobs efficiently. The FFSP can be further divided into two sub-problems. The first sub-problem involves machine selection for each operation, where decisions must be made regarding which

machine should perform a particular job at each stage. The second sub-problem entails determining the optimal sequence of operations for each machine in the system.

The FFSP can be categorized into three types based on the characteristics of parallel machines: FFSP with identical parallel machines (FFSP-IPM), FFSP with uniform parallel machines (FFSP-UM), and FFSP with unrelated parallel machines (FFSP-UPM) ([10]). These classifications help differentiate the variations of FFSP based on the nature of the parallel machines involved.

1.1. FFSP Recent Publications

The FFSP has been gaining significant attention in both academic research and the manufacturing industry, primarily because of its relevance and applicability in diverse production systems. Flexible production systems are particularly favored, especially in make-to-order environments where unexpected disruptions often arise during the execution of scheduling plans. Researchers have observed in recent years that flexible systems, such as the flexible flow shop, offer efficient approaches to address the inherent uncertainties in these environments [11,12]. The literature on FFSP is abundant. Comprehensive literature reviews on FFSP can be found in [13–16]. These sources offer detailed insights and analysis of the FFSP problems. The following presents a brief literature review of recent publications focusing on various variants of the FFSP.

In [17], the authors focused on the distributed FFSP with blocking constraints. They proposed the utilization of greedy algorithms to tackle this problem. In the latter algorithm, machine idle time is minimized through active decoding. A neighborhood search framework is then implemented to increase the diversity of solutions. For generating favorable initial solutions, a blocking constraint heuristic rule is introduced. In [18], a FFSP problem with energy considerations is studied. A multi-objective MILP is proposed for the latter problem. In addition, a Q-learning and an improved genetic algorithm are presented to solve the addressed problem. The authors in [19] discuss the different solution representations for the FFSP, their advantages and their disadvantages, and how to strike a balance between reducing the solution space and ensuring an efficient search within the reduced solution space. The authors in [20] address the dynamic FFSP with re-entrant jobs and take into account worker fatigue and skill levels. A multi-agent technique reinforced with deep learning is used to provide a near-optimal solution. Extensive experimental study shows the efficiency of the proposed algorithm. The work in [21] studies an FFSP problem with fuzzy maintenance time and with robots. A bi-objective mathematical model is presented, and two multi-objective decision-making approaches, namely LP-metric and goal attainment (GA), are utilized to solve the problem. The efficiency of the proposed solution approaches is evaluated and ranked using the "TOPSIS" (Technique for Order of Preference by Similarity to Ideal Solution) method. In the study [22], several uncertain parameters related to the production process in FFSP are considered. A two-stage stochastic programming is proposed to solve the addressed problem. To effectively solve the problem at hand, a novel variant of the pointer-based discrete differential evolution (PDDE) algorithm, called H-PDDE, is proposed. This variant aims to enhance the performance and efficiency of the PDDE algorithm in tackling the given problem. A distributed FFSP with multiprocessor jobs is addressed in [23]. The studied problem is formulated as a Markov Decision Process (MDP) and subsequently solved using a hybrid Q-learning-local search algorithm. This approach combines the advantages of Q-learning, a reinforcement learning technique, with local search methods to effectively find near-optimal solutions for the problem. Revised and improved mathematical integer formulations are proposed for the FFSP with chaining time-lag and time-varying resources in [24]. To strengthen the formulations, valid inequalities are developed. These valid inequalities are tested and assessed and the obtained results show their performance.

In [25], a comprehensive review of constructive heuristics for the FFSP is performed. The reviewed heuristics are assessed experimentally and compared to four new proposed heuristics. Two memory-based constructive heuristics are proposed, wherein jobs are

sequentially inserted into a partial sequence. Promising insertions are retained in a list to form the constructed sequence. Two Johnson's algorithm-based constructive heuristics are proposed as the second approach. The authors in [26] addressed an FFSP problem with batch arising in the sand casting industry and proposed an enhanced cuckoo algorithm was developed. In this improved version, crossover and mutation operations were introduced to enhance the search capability of the cuckoo algorithm by replacing long and short flight strategies. An FFSP problem taking into account human factors effect is studied in [27]. The two considered human factors are the fatigue factor and the multi-skilled workers. An agent-based simulation system was created to handle uncertainties in the worker fatigue model. In addition, this work introduces a novel simulation-based optimization (SBO) framework that combines reinforcement learning (RL) and genetic algorithm (GA) to tackle the hybrid flow shop scheduling problem. In [28], a distributed two-stage FFSP problem with maintenance requirements is considered. A mixed-integer programming model and a genetic algorithm are proposed. The FFSP problem with chaining time-lag constraints and time-varying resources is investigated over the research work [24]. The latter work presents new enhanced integer mathematical formulations and valid inequalities. The authors in [29] study the FFSP with energy considerations. An optimization algorithm based on the Hybrid Aquila Optimizer (HAO) is proposed to solve the latter problem. In [30], the distributed heterogeneous FFSP with lot-streaming is investigated. A mixed-integer linear programming model is presented, and a family of constructive heuristics and an iterated local search algorithm are proposed. The presented constructive heuristics are time-based rules. For more details on the FFSP, the reader is referred to the recent review paper [13–16].

1.2. FFSP Resolution Approaches

The scheduling problems associated with FFSP are known to be NP-Hard combinatorial optimization problems, as established in prior research [31–33]. This classification indicates the inherent computational complexity of solving FFSP scheduling problems. NP-hardness implies that finding an optimal solution for these problems is challenging and time-consuming, especially as the size of the problem instance increases. Consequently, developing efficient algorithms and heuristics becomes crucial in addressing FFSP scheduling problems and obtaining near-optimal solutions within reasonable time frames. The NP-Hard nature of FFSP scheduling problems underscores the significance of ongoing research efforts to devise effective methodologies and optimize scheduling strategies for practical applications in various industries.

According to existing literature [9,34,35], the FFSP can be primarily tackled using two types of methods: exact methods and approximate methods. Exact methods encompass approaches such as mixed-integer linear programming (MILP) [36], branch-and-bound algorithm (B&B) [37,38], and branch-and-cut method [39,40]. On the other hand, approximate methods mainly include heuristic methods and metaheuristic algorithms. Heuristic algorithms leverage the specific characteristics of FFSP [31,41], such as dispatching rules and NEH [42], to obtain feasible solutions within a relatively short time. However, the quality of these solutions cannot be guaranteed.

In contrast, metaheuristic algorithms, such as simulated annealing (SA) [43], artificial bee colony algorithm (ABC) [44], migrating birds optimization algorithm (MBO) [45], genetic algorithms (GA) [10], and others [9,46], are efficient and popular methods for solving FFSP. It is worth noting that for the effective implementation of metaheuristic algorithms, specific encoding and decoding rules must be designed for the FFSP problem [47–49]. If not properly designed, the solution quality of metaheuristic algorithms can suffer ([50–53]). In comparison, exact algorithms can solve small-sized problems optimally, albeit with slower computation speed. However, for large-sized problems, finding a feasible solution becomes challenging, and it may even lead to memory limitations [40]. On the other hand, approximate methods provide approximate solutions with unpredictable performance. Consequently, studying the efficiency of exact algorithms in addressing scheduling

problems, especially for small-sized instances where optimal solutions are desired, holds significant importance.

A hybrid metaheuristic combines multiple metaheuristic algorithms or incorporates elements from different techniques to solve optimization problems. This approach has gained attention from scholars across various fields. In [54], the authors proposed a hybrid tournament differential evolution algorithm to solve a two-warehouse inventory control problem with deteriorating items. The objective was to determine the lot size, maximum shortage level, and cycle length for the system. Similarly, in [55], the authors utilized binary tournament-based quantum-behaved particle swarm optimization algorithms to tackle an imperfect production inventory problem with shortages. In the context of the FFSP, several studies proposed such a technique (hybrid metaheuristics) to tackle complex FFSP problems. In this regard, the authors in [56] proposed a Hybrid Brain Storm optimization algorithm (HBOS) to solve a distributed FFSP problem. In addition, a mathematical model is formulated. A hybrid EDA-DE algorithm is proposed in [57] to address the two-stage FFSP. This algorithm combines the estimation of distribution algorithm (EDA) and differential evolution (DE) algorithm. It takes into account job-dependent deteriorating effects and non-identical job sizes simultaneously, especially in the context of parallel batching machines. In [58], Evolutionary algorithms such as particle swarm optimization and genetic algorithms are combined with a clustering-based approach to handle the hybrid flow shop scheduling problem. This problem involves unrelated and eligible machines, as well as fuzzy processing times and fuzzy due dates.

To the best of the author's knowledge, there is limited literature on the MILP modeling of FFSP. For example, authors in [36] have proposed the only MILP model for FFSP-UP. However, for the extended problems of FFSP, such as FFSP with sequence-dependent setup times, no-wait FFSP, and FFSP with blocking, there are currently no feasible MILP models available to solve them. Although the popularity of MILP modeling for scheduling problems has grown in recent years, thanks to advancements in computing performance and the availability of efficient software tools like CPLEX and GUROBI, there is still a need for significant efforts to enhance their performance in the field of scheduling. Indeed, when it comes to medium-sized and large-sized problems, these mathematical models still encounter several significant challenges. The complexity and computational requirements of the models increase exponentially as the problem size grows, leading to difficulties in finding feasible solutions within reasonable time limits ([10,13–16,39,40]).

1.3. Manufacturing Practical Example for the FFSP with Unloading Operations

From a practical point of view, unloading operations can indeed be as important as setup times in certain real-life manufacturing processes. The sand casting process serves as a real-life manufacturing example where the unloading operation plays a critical role [26]. Indeed, in the context of sand casting, the unloading (or removal) operation refers to the process of removing the sand mold from the solidified metal casting. Sand casting is a widely used manufacturing method for producing metal components with complex shapes and sizes. After the molten metal has been poured into the sand mold and allowed to cool and solidify, the unloading operation begins. The primary goal is to separate the solidified metal casting from the surrounding sand mold without causing any damage to the casting itself. The removal process typically involves various techniques and equipment, including:

1. Shakeout: The sand mold is vibrated or mechanically shaken to loosen and remove the sand from the casting. This can be done manually or using specialized equipment such as shakeout machines or tumblers.
2. Knockout: In this method, the sand mold is physically knocked or struck to dislodge the sand from the casting. This can be done by hand or with the help of hammers, mallets, or pneumatic tools.
3. Sandblasting: High-pressure air or abrasive materials are used to remove the remaining sand particles from the casting surface. Sandblasting helps to achieve a clean and smooth finish on the casting.

4. Cleaning and finishing: After most of the sand has been removed, the casting may undergo further cleaning and finishing processes to remove any remaining traces of sand, smooth rough edges, or remove any surface imperfections.

The unloading operation in sand casting is a critical step in the manufacturing process. It requires careful handling to ensure the integrity of the casting while effectively and efficiently separating it from the sand mold, which requires skilled operators.

1.4. FFSP with Unloading Operations and Research Gaps Identification

It is important to mention that machining and non-machining times are the two main inputs into scheduling problems. The machining time is the processing time, while the non-machining time contains the setup time (preparation time), the loading, and the unloading time. The non-machining activities time is about 95% of the total time utilized in production [59]. Therefore, it is very crucial to take into account the non-machining time during scheduling to narrow the gap between theory and practical. In this context, several research works in the area of FFSP considering setup time (loading and/or preparation of machines) are presented [60–66].

In the majority of the presented works [13–16] for the FFSP, the unloading time, or the time required to remove a job from a machine, is neglected. A few research papers taking into account the unloading time (independently from the processing time) were presented. A review of these few papers is presented in the sequel. In this context, authors in [67] addressed the two-stage FFSP with independent setup and unloading times where the first stage contains only one machine. The authors proposed a family of efficient heuristics, which are based on Sule's rule [68], initially developed for scheduling in permutation flow shop environments. In [69], the authors addressed the multi-stage Flexible Flow Shop Problem (FFSP) with the objective of minimizing maximum lateness. The problem involved jobs with precedence constraints, due dates, and lag times, which could include setup, unloading, or transportation times. The study presented a family of heuristics, four of which were extensions of existing heuristics initially developed for the flow shop scheduling problem. Additionally, other heuristics were specifically designed to effectively consider the precedence constraints. In [43], the study focused on the Flexible Flow Shop Problem with an unrelated parallel machine (FFSP-UP) with total flow time minimization. The jobs are subject to setup and unloading time constraints. The authors proposed a metaheuristic based on simulated annealing to address this problem effectively. The study in [70] focuses on the two-stage no-wait FFSP problem, where one stage contains one machine, and the setup and unloading times for each job are independent of the processing time. Recognizing the problem's NP-completeness, the research proposes a heuristic algorithm that independently addresses the sequencing and assignment of jobs to each stage. This heuristic employs Sule's rule to schedule jobs in the first stage, which consists of a single machine. In the second stage, each completed job from the first stage is assigned to the most available machine. The work in [71] focuses on a multiple-lot lot-streaming problem in a two-stage FFSP with unloading times comprising one machine in the first stage and two parallel machines in the second stage. The study utilizes fundamental findings from the single-lot problem to develop mathematical programming-based heuristic methods for solving the problem.

The literature review of FFSP with unloading operation reveals a relatively limited body of research on this specific aspect. Despite the considerable attention given to FFSP, studies that specifically take into account unloading operations are scarce. Recognizing the importance of the unloading phase in several manufacturing processes, it becomes evident that this aspect merits a more comprehensive investigation. Efficient unloading operations are integral to the seamless functioning of FFSP, influencing the overall productivity and effectiveness of manufacturing processes. Given the current gaps in the literature, there exists a need for further exploration and analysis of the FFSP unloading operation. Understanding and addressing the unique challenges and opportunities presented by this component can contribute significantly to the advancement of manufacturing systems

and optimization strategies. Based also on the existing literature for FFSP with unloading times ([43,67–71]), it turns out that these studies focus on limited configurations involving two stages, with one machine in one of them. However, this motivates the consideration of the general case of multi-stage FFSP with unloading operations. Moreover, none of the previously presented papers solely address the unloading operation, which highlights the importance of gaining a comprehensive understanding of its impact.

Therefore, this paper integrates the unloading operation in a Flexible Flow Shop Problem with identical parallel machines at each stage. The obtained problem is an extension of the classical FFSP. This is the flexible flow shop scheduling problem with unloading operations (FFSPU). The unloading operation for each processed job on a machine requires a certain time to be performed. Until and during the unloading phase, a machine is not available for treating other jobs. Generally, the unloading operation is performed by an operator, which can be a human, a robot, or other equipment. The operator has the flexibility to initiate the unloading immediately after the completion of the processing stage. Alternatively, the unloading operation can be postponed or delayed to a later date without extending the overall makespan of the process. This flexibility provides the operator with the opportunity to make decisions regarding the optimal timing for starting the unloading operation. During the waiting period for unloading, the operator can utilize this time to perform other jobs, therefore maximizing efficiency and productivity. This flexibility in scheduling allows for improved resource utilization and adaptability within the overall operational process. The goal of this research work is to study the FFSPU with the purpose of finding the optimal processing sequences on the machines that minimize the makespan.

1.5. Comparison of the Proposed Algorithms with the Existing Ones and Main Contributions

It is important to note that the FFSP is a challenging problem from a theoretical perspective since it is strongly NP-Hard, even if only two stages are involved, and one of them contains more than one machine [31]. Even if preemption is allowed, the FFSP remains NP-Hard [32]. Consequently, the majority of the FFSP variants are strongly NP-Hard [13–16]. In our case, the current studied problem FFSPU is NP-Hard in the strong sense. To tackle the latter problem, a two-phase efficient heuristic is proposed. This heuristic is based on solving iteratively a parallel identical machine-scheduling problem with release dates, unloading times, and delivery times. The proposed heuristic is a two-phase procedure. In the first phase, an initial feasible solution is derived. The second phase is an improvement. In addition, a family of efficient lower bounds is presented. These lower bounds allow the evaluation of the performance of the proposed heuristic over the relative gap. An extensive computational study is carried out on benchmark test problems. This computational study provides evidence that the newly proposed lower bounds and the heuristic are efficient.

To the best of the author's knowledge, there is currently no existing algorithm proposed to solve the FFSPU (Flexible Flow Shop Scheduling Problem with Unloading). Consequently, in this study, the proposed algorithms are qualitatively compared to existing procedures, including metaheuristics, hybrid metaheuristics, and exact solutions such as the MILP (Mixed-Integer Linear Programming). This comparison serves to highlight the novelty and effectiveness of the proposed algorithms in addressing the FFSPU, filling the gap in the existing literature. Indeed, when compared to the existing approximate solutions for the FFSP (especially the metaheuristics and hybrid metaheuristics), the two-phase heuristic proposed in this study is considered preferable. Indeed, the two-phase heuristic generates schedules that exhibit high-quality solutions at each stage. This implies that the resulting schedules are likely to be closer to the optimal solution compared to the existing approximate solution. In addition, the two-phase heuristic is designed to provide near-optimal solutions within an acceptable computation time. This is especially important for medium and large-scale instances of the FFSPU, where exact methods tend to be inefficient, and metaheuristics are time-consuming procedures. The preference for the two-phase heuristic is justified by its ability to strike a balance between solution quality

and computational efficiency. Furthermore, the two-phase heuristic uses an extended version of an existing efficient heuristic for the parallel machine-scheduling problem with release and delivery times that incorporate unloading times efficiently. By leveraging this extension, the heuristic becomes more adaptable to the specific requirements of the FFSPU. Finally, an extensive experimental study conducted on benchmark test problems supports the effectiveness of the two-phase heuristic. The results demonstrate its effectiveness, as indicated by the average computation time and mean relative gap. Considering these factors, the two-phase heuristic emerges as a preferable choice for addressing the FFSPU. Its ability to produce high-quality solutions within a reasonable computation time makes it a valuable contribution to the field.

The main contributions of the current study are as follows:

- The study focuses on the FFSPU, which has received limited attention in existing literature. By addressing this underexplored problem, the research contributes to gaining deeper insights and expanding the existing knowledge base in this field.
- The research presents intriguing new properties of the studied problem, such as its symmetry, which has a positive impact on enhancing the solution quality. These novel properties contribute to advancing the potential improvements in solving the problem.
- The research introduces a novel and efficient two-phase heuristic that offers a short computational time while producing near-optimal solutions. This proposed heuristic provides a promising approach to address the problem effectively and efficiently.
- The study presents new lower bounds that serve as a metric to measure the maximum deviation of the proposed two-phase heuristic from the optimal solution. These lower bounds provide valuable insight into evaluating the performance of the heuristic and its proximity to the optimal solution.
- Consider the multi-stage FFSPU with up to ten stages, capturing the essence of the FFSPU.

1.6. Outline of the Paper

The remainder of this paper follows the following structure. Section 2 presents the definition of the problem as well as its important features. Section 3 presents the set of lower bounds. Section 4 describes the details of the proposed heuristic. Section 5 presents the experimental study as well as the obtained results. Lastly, the conclusion and future work are presented in Section 6.

2. Problem Description

In this section, we will formally define the study problem. In addition, some of its key characteristics are discussed.

2.1. Problem Definition

The flexible flow shop scheduling problem with unloading operations (FFSPU) consists of processing a set $J = \{1, 2, \ldots, n\}$ of n jobs over K stages $S_1, S_2, \ldots, S_K$ in series. A set $M_i = \{M_{i,1}, M_{i,2}, \ldots, M_{i,m_i}\}$ of m_i identical parallel machines is composing the stage S_i $(i = 1, \ldots, K)$. The processing time of job j at stage S_i is denoted by $pr_{i,j}$. After finishing processing on a machine $M_{i,u}$ of stage S_i the job $j \in J$ requires $un_{i,j}$ units of time to be unloaded. It is worth mentioning that the starting of the unloading operation for a job $j \in J$ could not be its completion of processing. Indeed, the unloading is independent of the processing. During the period between the ending processing and the ending of the unloading period, the machine $M_{i,u}$ is not available for processing other jobs. This is because the unloading is independent of processing. In addition, the unloading operation's earliest starting time for a job $j \in J$ is its ending processing date. Each job $j \in J$ is processed from stage S_1 to stage S_K in that order. The scheduling is performed under the following assumptions:

- Job preemption during the processing is not allowed.
- Each job is processed only in one machine.
- A machine processes at most one job at the same time.

- The buffer between two consecutive stages has unlimited capacity.
- All the processing and unloading times $pr_{i,j}$, $un_{i,j}$ $(i = 1, \ldots, K; j = 1, \ldots, n)$ are deterministic and positive integers.
- All machines and all jobs are available from time zero onwards.

The completion time of a job within a feasible schedule is the date of completing the unloading operation. If σ denotes a feasible schedule and $c_{i,j}(\sigma)$ the completion unloading time of job j in stage S_i relative to σ, then the maximum completion time (or makespan) is given by:

$$C_{\max}(\sigma) = \max_{j \in J}(c_{K,j}(\sigma)).$$

The objective is to find a feasible schedule σ^* that minimizes the makespan or the makespan. Following the three-field notation [72], the current studied problem is denoted $FH_K, \left(\left(PM^{(l)}\right)_{l=1}^{K}\right)|un_{i,j}|C_{\max}$.

In the latter notation, the first field, FH_K indicates the flexible flow shop with K stages. The symbol $\left(\left(PM^{(l)}\right)_{l=1}^{K}\right)$ is precise that in each stage, we have identical parallel machines. In the second field, the unloading time is presented. The last field is reserved for the objective function, which is the makespan.

An illustrative example is presented as follows.

Example 1. *Consider the two-stage FFSPU where* $n = 5$, *and* $m_1 = m_2 = 2$. *The processing times* $pr_{i,j}$ *and the unloading times* $un_{i,j}$ $(i = 1, 2$ *and* $j \in J)$ *are presented in Table 1.*

Table 1. Data of Example 1.

	j	**1**	**2**	**3**	**4**	**5**
S_1	$pr_{1,j}$	1	2	1	2	2
	$un_{1,j}$	1	1	2	1	2
S_2	$pr_{2,j}$	2	1	1	2	1
	$un_{2,j}$	1	1	1	1	1

Figure 1 presents a feasible schedule for the addressed scheduling problem.

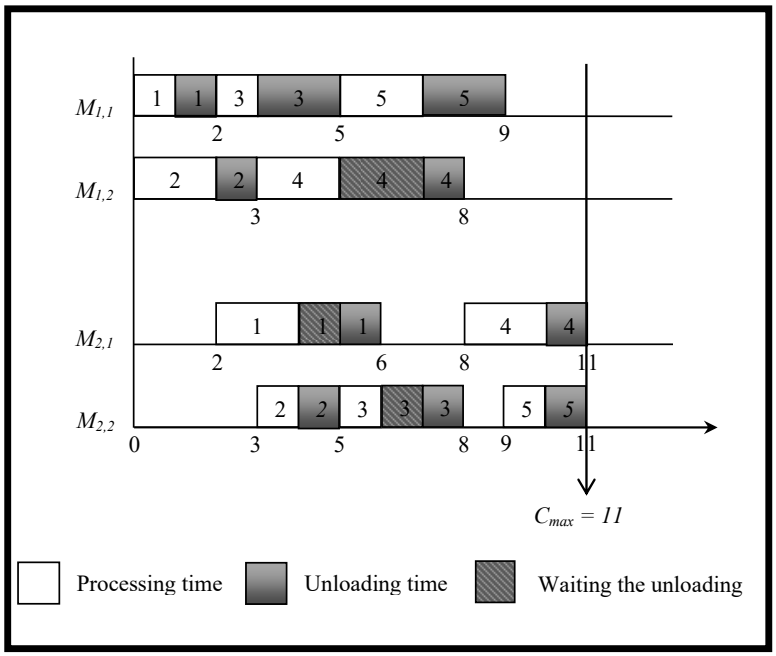

Figure 1. Gantt chart of a feasible schedule for Example 1.

In Figure 1, job 4 remains on machine $M_{1,2}$ after completing processing at Time 5. The unloading operation starts later at Time 7 and finishes at Time 8. This is the completion time $c_{1,4}$ of job 4. This shows that the unloading operation is independent of processing time.

In this regard, the waiting unloading time refers to the duration during which a job must wait before it can undergo the unloading operation. This typically occurs when the unloading resource (operator) or equipment is occupied with another job or when some specific constraints or dependencies delay the start of the unloading process. In our case, the waiting unloading is managed to ensure the schedule stays within the makespan of the critical path.

2.2. Problem's Proprieties

2.2.1. Complexity

The complexity of the studied problem is the subject of Lemma 1.

Lemma 1. *The problem* $FH_K, \left(\left(PM^{(l)}\right)_{l=1}^{K}\right)|un_{i,j}|C_{max}$ *is NP-Hard in the strong sense.*

Proof. The problem $FH_K, \left(\left(PM^{(l)}\right)_{l=1}^{K}\right)|un_{i,j}|C_{\max}$ is an extension of the two-stage FFSP $FH_2, \left(\left(PM^{(l)}\right)_{l=1}^{2}\right)|\,|C_{\max}$ ($K = 2$ and $un_{i,j} = 0$), which is strongly NP-Hard [73]. □

2.2.2. Symmetry Propriety

Definition 1. *The reverse problem of* $FH_K, \left(\left(PM^{(l)}\right)_{l=1}^{K}\right)|un_{i,j}|C_{max}$ *is defined as scheduling from the last stage* S_K *until reaching the first stage* S_1*, in that order. In other terms, the reverse problem (the symmetric) is obtained by inverting the roles of the stages.*

Based on the latter definition, the following notations and definitions are presented.

- The forward problem of $FH_K, \left(\left(PM^{(l)}\right)_{l=1}^{K}\right)|un_{i,j}|C_{\max}$ is the scheduling from S_1 to S_K.
- The backward (or symmetric) problem of $FH_K, \left(\left(PM^{(l)}\right)_{l=1}^{K}\right)|un_{i,j}|C_{\max}$ is the scheduling from S_K to S_1.

Definition 2

- *The stages and the number of machines for the backward problem are, respectively, denoted* $S_k^B = S_{K-k+1}$ *and* $m_k^B = m_{K-k+1}$ $(k = 1, \ldots, K)$.
- *The machines are denoted* $M_{k,l}^B = M_{K-k+1,l}$ $\left(k = 1, \ldots, K; l = 1, 2, \ldots, m_k^B\right)$.
- *The processing and unloading times for the backward problem are denoted* $pr_{k,j}^B$ *and* $un_{k,j}^B$*, respectively.*
- *The processing and unloading times for the backward problem are defined* $pr_{k,j}^B = un_{K-k+1,j}$ *and* $un_{k,j}^B = pr_{K-k+1,j}$*, respectively.*
- *The backward (or symmetric) problem is denoted* $FH_K, \left(\left(PM^{(l)}\right)_{l=K}^{1}\right)\left|un_{k,j}^B\right|C_{max}$.
- $FH_K, \left(\left(PM^{(l)}\right)_{l=K}^{1}\right)\left|un_{k,j}^B\right|C_{max}$ *is the notation of the symmetric of* $FH_K, \left(\left(PM^{(l)}\right)_{l=1}^{K}\right)|un_{i,j}|C_{max}$.

It is worth mentioning that for the symmetric problem, the unloading time for a job is equivalent to its processing time. Conversely, the processing time serves as the unloading

time for the symmetric problem. This occurs because scheduling for the symmetric problem proceeds from the last stage to the first in that particular direction, resulting in an inversion of the roles between processing time and unloading time.

The following result illustrates the importance of studying the symmetric problem.

Proposition 1. *Any feasible schedule FS for the problem* $FH_K, \left(\left(PM^{(l)}\right)_{l=1}^{K}\right)|un_{i,j}|C_{max}$ *is naturally transformed into a feasible schedule* FS^B *for the symmetric problem and vice versa. Moreover, the schedules FS and* FS^B *have the same makespan.*

Proof. Let FS be a feasible schedule for $FH_K, \left(\left(PM^{(l)}\right)_{l=1}^{K}\right)|un_{i,j}|C_{\max}$. Naturally, a feasible schedule FS^B for the symmetric problem is obtained by keeping the same assignments and sequences in the machines as for FS. A new reversed time scale is adopted and given by the following expression: $t^B = C_{\max} - t$, where t is the forward problem time scale. The two feasible schedules FS and FS^B have the same critical path and, therefore, have the same makespan. Similarly, by adopting the same above strategy, a feasible schedule FS^B for the symmetric problem is transformed into a feasible schedule FS for the original problem with $t = C_{\max} - t^B$ and t^B the reverse time scale. □

To illustrate the symmetric problem concept, example 1 is reconsidered. For this example, the data for the reverse (symmetric) problem is given in the following table (Table 2).

Table 2. Data of Example 1 for the symmetric problem.

	j	**1**	**2**	**3**	**4**	**5**
$S_1^B = S_2$	$pr_{1,j}^B$	1	1	1	1	1
	$un_{1,j}^B$	2	1	1	2	1
$S_2^B = S_1$	$prB_{2,j}$	1	1	2	1	2
	$un_{2,j}$	1	2	1	2	2

Figure 2 presents a feasible schedule for the symmetric problem.

It is worth mentioning that the symmetric problem is investigated automatically for the proposed procedures (lower bounds and heuristic) to improve the quality of the final obtained solution.

To better illustrate the symmetric aspect of the studied problem, Figure 3 has been included. This figure visually represents the symmetrical characteristics inherent in the problem under investigation. By examining this figure, readers can gain a better understanding of the symmetries present in the problem structure and how they may impact the formulation and solution approach. The inclusion of Figure 3 enhances the clarity and visual representation of the symmetric nature of the problem at hand.

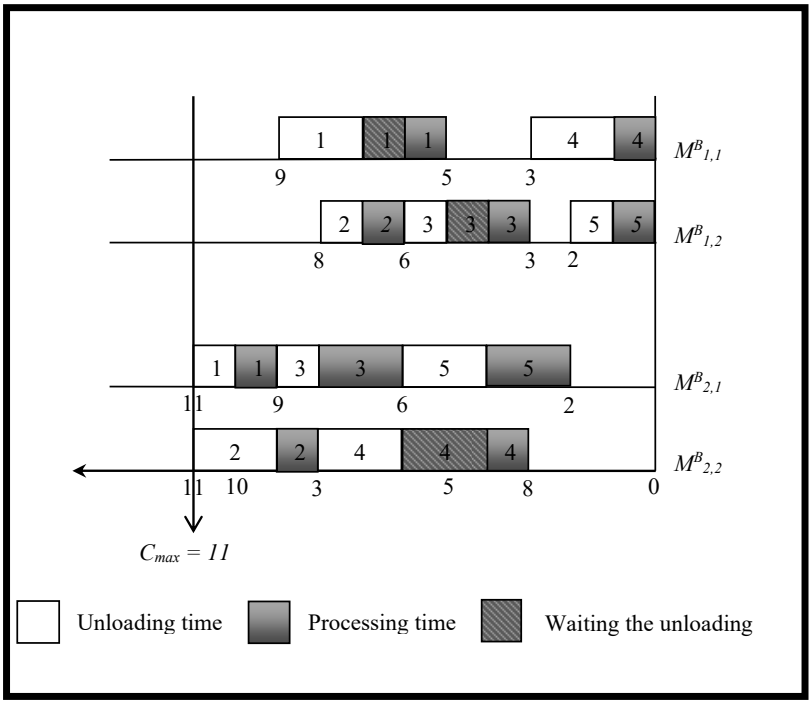

Figure 2. Gantt chart of a feasible schedule for the symmetric problem relative to Example 1.

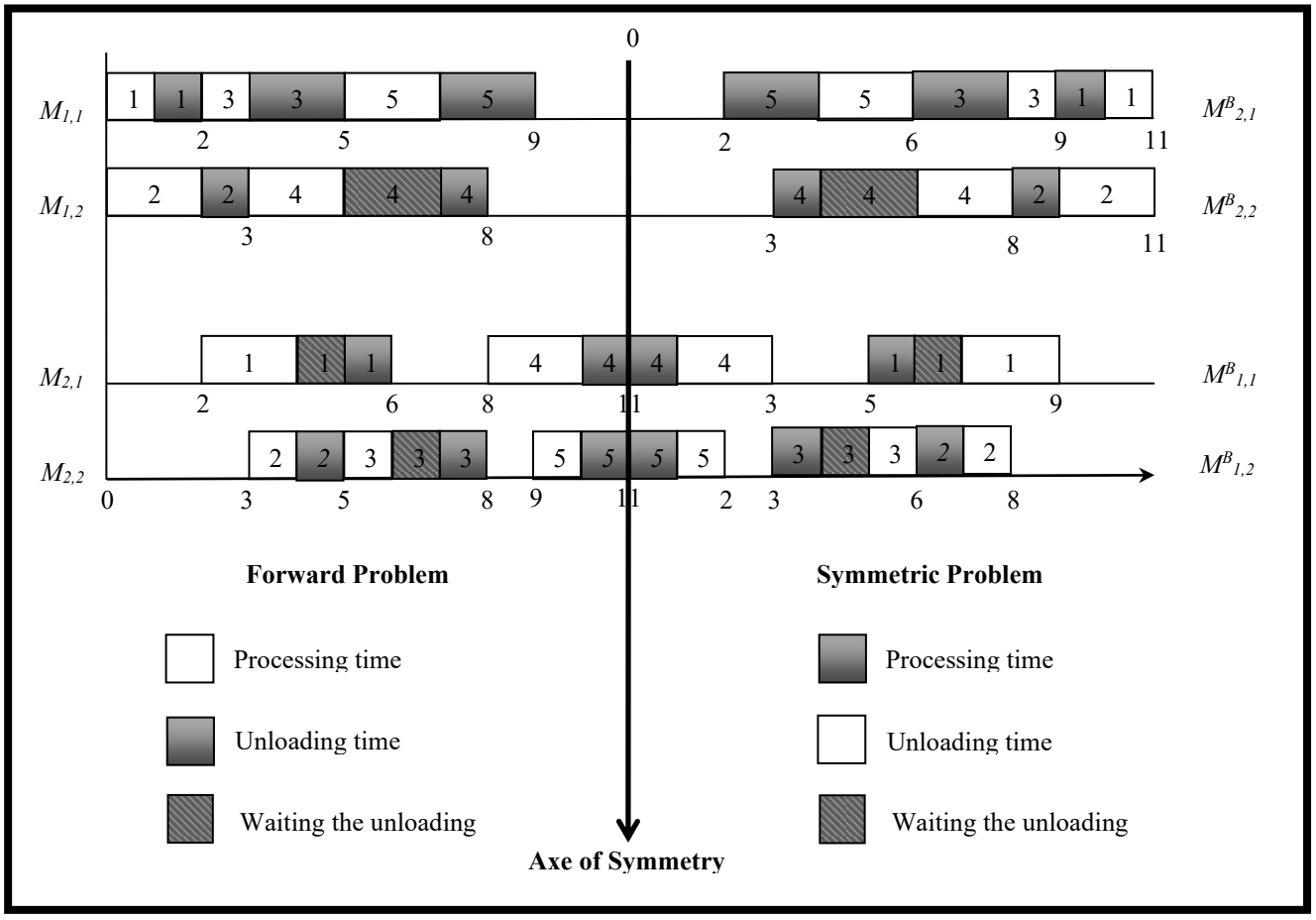

Figure 3. The symmetric aspect of the FFSPU relative to Example 1.

The latter proposition (Proposition 1) allows obtaining the following important result.

Corollary 1. *The two problems* $FH_K, \left(\left(PM^{(l)}\right)_{l=1}^{K}\right)\left|un_{i,j}\right|C_{max}$ *(Forward) and* $FH_K, \left(\left(PM^{(l)}\right)_{l=K}^{1}\right)\left|un^{B}_{k,j}\right|C_{max}$ *(symmetric) present the same optimal makespan.*

Proof. An immediate consequence of Proposition 1. □

In the sequel, some useful notations are presented. For each stage $S_k (k = 1, 2, \ldots, K)$ and for each job $j \in J$, a release date $re_{k,j}$ and a delivery time $qe_{k,j}$ are expressed as follows.

$$\begin{cases} re_{k,j} = \sum_{i=1}^{k-1}\left(pr_{i,j} + un_{i,j}\right) \text{ if } k > 1 \\ re_{kj} = 0 \text{ if } k = 1 \end{cases}, \text{ and } \begin{cases} qe_{k,j} = \sum_{i=k+1}^{K}\left(pr_{i,j} + un_{i,j}\right) \text{ if } k < K \\ qe_{k,j} = 0 \text{ if } k = K \end{cases}$$

respectively. In addition, for each stage $S_k (k = 1, \ldots, K)$:

- $\overline{re_{k,j}}_{(i)}$ denotes the i^{th} smallest value of the $re_{k,j}$'s ($j = 1, \ldots, n$).
- $\overline{qe_{k,j}}_{(i)}$ denotes the i^{th} smallest value of the $qe_{k,j}$'s ($j = 1, \ldots, n$).
- $\overline{\left(pr_{k,j} + un_{k,j}\right)}_{(i)}$ denotes the i^{th} smallest value of the $\left(pr_{i,j} + un_{i,j}\right)$'s ($j = 1, \ldots, n$).

3. Lower Bounds

A set of lower bounds for the studied problem is presented in this section. These lower bounds are of two kinds. The first type relies on the capacity relaxation of all stages except one. A second lower bound is based on estimating the minimum idle time in each stage. The lower bounds are also useful for assessing the effectiveness of the heuristic throughout measuring the relative gap.

3.1. One-Stage-Based Lower Bound

By relaxing the capacities (Considering an infinite number of machines) for all stages but stage S_k, we obtained a parallel machine-scheduling problem $P_m|\ re_{k,j}, un_{k,j}, qe_{k,j}\left|C_{\max}\right.$ with release dates $re_{k,j}$, unloading times $un_{k,j}$, delivery times $qe_{k,j}$ and several machines $m = m_k$. The latter problem is relaxed by omitting the idle time between the complete processing of a job and the starting of the unloading operation. In so doing, the obtained problem is a parallel machine one $P_m|\ re_{k,j}, qe_{k,j}\left|C_{\max}\right.$ with only the release dates $re_{k,j}$ and delivery times $qe_{k,j}$. The processing time is $pr_{k,j} + un_{k,j}$ for each job j.

In [74], authors proposed a lower bound for the parallel machine-scheduling problem with release dates and delivery times $P_m|\ re_{k,j}, qe_{k,j}\left|C_{\max}\right.$. The latter lower bound is used to provide a lower bound for the relaxed problem $P_m|\ re_{k,j}, qe_{k,j}\left|C_{\max}\right.$. This lower bound is given by the following expression:

$$LB_S^k = \left\lceil \frac{1}{m_k}\left(\sum_{i=1}^{m_k}\overline{re_{k,j}}_{(i)} + \sum_{j=1}^{n}\left(pr_{k,j} + un_{k,j}\right) + \sum_{i=1}^{m_k}\overline{qe_{k,j}}_{(i)}\right)\right\rceil \quad (1)$$

Consequently, the following result holds.

Proposition 2.

$$LB_S = \max_{1 \le k \le K}\{LB_S^k\} \quad (2)$$

is a valid lower bound for the problem $FH_K, \left(\left(PM^{(l)}\right)_{l=1}^{K}\right)|un_{i,j}|C_{max}$ *with time complexity* $O(Kn)$.

Proof. LB_S^k is a lower for a relaxed problem $P_{m_k}|\ re_{k,j}, qe_{k,j}|C_{\max}$ then LB_S^k is also a lower bound for the problem $FH_K, \left(\left(PM^{(l)}\right)_{l=1}^{K}\right)|un_{i,j}|C_{\max}$. Therefore, $LB_S = \max\limits_{1\leq k\leq K}\{LB_S^k\}$ is a lower bound for the studied problem. In addition, the time complexity of LB_S^k is $O(n)$, thus repeating it K times results in a time complexity $O(Kn)$. □

3.2. Two-Stage Based Lower Bound

In this subsection, the second lower bound is derived. This lower bound is based on estimating the minimum idle time in stage S_k based on a particular schedule in the predecessor stage S_{k-1}, this is the two-stage-based lower bound. Before the presentation of the derived lower bounds, we recall useful parallel machine-scheduling problems. This problem is denoted $P_m|\ |\sum C_j$ and described as follows. A set $J = \{1, 2, \ldots, n\}$ of n jobs must be processed on m identical parallel machines M_i $(i = 1, 2, \ldots, m)$ without preemption. Each job $j \in J$ needs to be processed during p_j units of time. The purpose is to find a feasible schedule σ that minimizes $\sum C_j$, where C_j is the completion time of job j with respect to σ. It is worth mentioning that the $Pm|\ |\sum c_j$ problem is a polynomial one, and the optimal solution is obtained using the Shortest Processing Time (SPT) dispatching rule [75].

The following useful notations and definitions are also required.

- $SPT_k(l)$: is the sum of the completion times of the l jobs with the smallest $\overline{\left(pr_{k,j} + un_{k,j}\right)}_{(i)}$ $(j \in J)$ and scheduled on the m_k machines of stage S_k according to the SPT rule $(i \in \{1, 2, \ldots, K-1\})$.

The following result presents the second lower bound.

Proposition 3. LB_{2S}^k *given by the following expression:*

$$LB_{2S}^k = \overline{re_{k-1,j}}_{(1)} + \left\lceil \frac{1}{m_k}\left(SPT_{k-1}(m_k) + \sum_{j=1}^{n}\left(pr_{k,j} + un_{k,j}\right) + \sum_{i=1}^{m_k}\overline{qe_{k,j}}_{(i)}\right)\right\rceil \tag{3}$$

is a valid lower bound in stage S_k $(2 \leq k \leq K)$ *with time complexity* $O(nlogn)$.

Proof. For an optimal schedule with makespan $C^*_{\max}$ and a stage S_k $(2 \leq k \leq K)$, consider the followings:

- h_i: The first processed job in machine $M_{k,i}$
- n_i: The last unloaded job from machine $M_{k,i}$
- s_{k,h_i}: The starting time of job h_i on machine $M_{k,i}$
- $P_{k,i}$: The total processing time in machine $M_{k,i}$
- $Un_{k,i}$: The total unloading time in machine $M_{k,i}$
- $Id_{k,i}$: The total idle time of machine $M_{k,i}$ from time 0 until the completion of unloading of job n_i. □

Thus,

$$s_{k,h_i} + P_{k,i} + Un_{k,i} + Id_{k,i} + qe_{k,n_i} \leq C^*_{\max}. \tag{4}$$

We have,

$$\sum_{i=1}^{m_k} s_{k,h_i} + \sum_{i=1}^{m_k} P_{k,i} + \sum_{i=1}^{m_k} Un_{k,i} + \sum_{i=1}^{m_k} Id_{k,i} + \sum_{i=1}^{m_k} qe_{k,n_i} \leq m_k C^*_{\max}, \tag{5}$$

Furthermore, we observe that:

$$\sum_{i=1}^{m_k} P_{k,i} = \sum_{j=1}^{n} p_{k,j} \text{ and } \sum_{i=1}^{m_k} Un_{k,i} = \sum_{j=1}^{n} un_{k,j} \quad (6)$$

In addition, we have:

$$\sum_{i=1}^{m_k} \overline{qe_{k,j}}_{(i)} \leq \sum_{i=1}^{m_k} qe_{k,n_i} \quad (7)$$

By relaxing the release dates in stage S_{k-1} and considering $re_{k,j} = \overline{re_{k,j}}_{(1)}$, a parallel machine problem $P_{m_{k-1}} \left| \right| \sum C_j$ is obtained and

$$m_k \times \overline{re_{k,j}}_{(1)} + SPT_{k-1}(m_k) \leq \sum_{i=1}^{m_k} s_{k,h_i} \quad (8)$$

Based on (5)–(8) it follows:

$$\overline{re_{k-1,j}}_{(1)} + \left\lceil \frac{1}{m_k} \left(SPT_{k-1}(m_k) + \sum_{j=1}^{n} \left(pr_{k,j} + un_{k,j} \right) + \sum_{i=1}^{m_k} \overline{qe_{k,j}}_{(i)} \right) \right\rceil \leq C^*_{\max}$$

The main effort of computing LB^k_{2S} is sorting $pr_{k,j} + un_{k,j}$ $(j \in J)$ which requires $O(n\log n)$ time.

An immediate result is presented over the following corollary.

Corollary 2.

$$LB^F_{2S} = \max_{2 \leq k \leq K} \left\{ LB^k_{2S} \right\} \quad (9)$$

is a valid lower bound with time complexity $O(Kn\log n)$.

Proof. Based on the above proposition (Proposition 3), LB^k_{2S} is a valid lower bound. Thus, $LB^F_{2S} = \max_{2 \leq k \leq K} \left\{ LB^k_{2S} \right\}$ is also a lower bound requiring $O(Kn\log n)$ time. □

Taking advantage of the symmetry propriety of the studied problem, the lower bound given in Corollary 2 might be enhanced as follows.

Corollary 3. *Considering the symmetric problem as defined in Definition 1, a similar lower bound as* LB^B_{2S} *is obtained and*

$$LB_{2S} = max \left\{ LB^F_{2S}, LB^B_{2S} \right\} \quad (10)$$

is a valid lower bound with time complexity $O(Kn\log n)$.

It is interesting to note that LB_S and LB^F_{2S} do not show dominance relationships. The following are two examples that illustrate this.

Example 2. *Consider the following instance:* $K = 3$, $m_1 = m_2 = m_3 = 2$ *and* $n = 4$. *The processing and unloading times in each stage are presented in Table 3.*

Table 3. Data of Example 2 ($LB_S > LB^F_{2S}$).

	j	**1**	**2**	**3**	**4**
S_1	$pr_{1,j}$	13	10	5	6
	$un_{1,j}$	10	11	9	7
S_2	$pr_{2,j}$	12	4	13	7
	$un_{2,j}$	9	8	5	12
S_3	$pr_{3,j}$	6	6	11	12
	$un_{3,j}$	8	12	13	15

In this case, we have:

$$LB_S^3 = \left\lceil \frac{\{(13)+(19)\}+\{(14)+(18)\}+[(14)+(20)+(24)+(27)]+(0)}{2} \right\rceil = 75$$

Moreover, we have:

$$LB_{2S}^1 = 0 + \left\lceil \frac{(0)+(23)+(21)+(14)+(13)+\{(32)+(35)\}}{2} \right\rceil = 69$$

$$LB_{2S}^2 = 0 + \left\lceil \frac{(14+13)+(21+12+18+19)+(14+20)}{2} \right\rceil = 66$$

$$LB_{2S}^3 = 13 + \left\lceil \frac{(12+18)+(14+20+24+27)+(0+0)}{2} \right\rceil = 71.$$

Thus, in this case, we have $LB_S < LB_{2S}^F$.

Example 3. *Consider the following two-stage instance:* $m_1 = 1$, $m_2 = 2$ *and* $n = 5$. *The processing and unloading times are displayed in Table 4.*

Table 4. Data of Example 3 ($LB_S < LB_{2S}$).

	j	**1**	**2**	**3**	**4**	**5**
S_1	$pr_{1,j}$	1	1	1	1	1
	$un_{1,j}$	1	1	1	1	1
S_2	$pr_{2,j}$	2	4	3	1	2
	$un_{2,j}$	2	1	2	3	5

In this case, we have:

$$LB_S^1 = \left\lceil \frac{(0)+(2+2+2+2+2)+(4)}{1} \right\rceil = 14$$

$$LB_S^2 = \left\lceil \frac{(2+2)+(4+5+5+4+7)+(0+0)}{2} \right\rceil = 15$$

Therefore, $LB_S = \max\left(LB_S^1, LB_S^2\right) = 15$.
On the other hand, we have:

$$LB_{2S}^2 = 0 + \left\lceil \frac{(2+4)+(4+5+5+4+7)+(0+0)}{2} \right\rceil = 16.$$

Hence, we obtain $LB_S < LB_{2S}^F$.

3.3. A General Lower Bound

A more general and stronger lower bound can be derived by simultaneously considering LB_S and LB_{2S}. Therefore,

Corollary 4.

$$LB = max(LB_S, LB_{2S}) \tag{11}$$

is a valid lower bound for the studied problem.

Proof. Immediate. □

4. A Two-Phase Optimization-Based Heuristic (*H*)

In this section, we develop a heuristic that is based on a multi-start stage and which is composed of two phases. This heuristic is designed to deliver approximate solutions that are as close as possible to the optimal solution for $FH_K, \left(\left(PM^{(l)}\right)_{l=1}^{K}\right)|un_{i,j}|C_{\max}$ problem. Phase 1 (PH1) involves developing an initial feasible solution, which is then improved in Phase 2 (PH2). During both phases, identical parallel machine-scheduling problems with release dates r_j, unload times u_j and delivery times q_j ($P_m|r_j, u_j, q_j|C_{\max}$) must be solved, or an equivalent variant that minimizes the maximum lateness ($P_m|r_j, u_j|L_{\max}$). In this approach, a heuristic so-called Approximate Decomposition Algorithm for Unloading (ADAU) is developed and used to derive an approximate solution for the $P_m|r_j, u_j, q_j|C_{\max}$ problem since it is known to be intractable. It is worth mentioning that ADAU is generalizing the Approximate Decomposition Algorithm (ADA), which is developed initially to provide an approximate solution for $P_m|r_j, q_j|C_{\max}$ [76].

ADAU is specifically designed to address the identical parallel machine-scheduling problem with release dates, delivery times, and unloading times. It takes into account the complexities associated with unloading times, which are the durations required to remove completed jobs from machines before they can start processing the next job. By considering these factors, ADAU aims to optimize scheduling outcomes comprehensively. On the other hand, ADA is tailored to provide feasible solutions for the same parallel machine problem but without considering unloading times. It focuses on finding schedules that meet the necessary constraints and requirements without incorporating unloading times.

In each iteration of ADAU (Adaptive Dispatching Algorithm with Unloading Times), the algorithm selects the two machines with the lowest and highest completion times, along with the corresponding scheduled jobs on those machines. This selection results in a two-machine-scheduling problem that incorporates release dates, delivery times, and unloading times ($P_2|r_j, u_j, q_j|C_{\max}$).

Solving this two-machine problem is comparatively easier than solving the general problem $P_m|r_j, u_j, q_j|C_{\max}$ because it involves only two machines and a restricted number of jobs. Thus, the focus of the algorithm shifts to finding an optimal solution for this simplified problem.

To address this two-machine problem, in this work, a truncated branch-and-bound algorithm is developed. If the newly obtained makespan is lower than the existing one, an improvement is detected, and the existing solution is updated accordingly. However, if no further improvement is detected or a predefined stopping condition is reached, the algorithm proceeds by selecting the machine with the highest completion time and another machine repeating the same procedure.

By iteratively applying these steps, ADAU aims to refine the scheduling solution and improve the overall makespan. The use of the truncated branch-and-bound algorithm allows for an efficient exploration of the solution space, ultimately leading to better scheduling outcomes. The ADAU algorithm is fast and yields optimal scheduling in most cases, according to our experiments.

Detailed descriptions of both the constructive (PH1) and improvement (PH2) phases are provided in the sequel.

4.1. The Initial Solution Construction Phase

For each i ($1 \leq i \leq K$), a feasible schedule γ_i is built using a constructive procedure. The pseudo-code for stage S_i is presented over the following algorithm (Algorithm 1).

As a first step, a $P_{m_i}|r_j, u_j, q_j|C_{\max}$ problem that is defined on stage S_i is solved (Steps 1–2). The previous resolution is performed using the already mentioned ADAU heuristic. The obtained completion unloading times $UC_{i,j}$ ($j \in J$) in stage S_i are considered to be release dates for the stage S_{i+1} and the resulting problem $P_{m_{i+1}}|r_j, u_j, q_j|C_{\max}$ is solved (Steps 3.1–3.2). This operation is repeated for each downstream stage (from S_{i+2} until S_K in that order) (Steps 3.1–3.2).

Solving the parallel machine $P_{m_K}|r_j, u_j, q_j|C_{\max}$ in the last stage S_K gives a preliminary estimate of the makespan (Step 4). As a last step in obtaining a complete solution, the same iterative strategy is used to schedule the upstream stages $S_{i-1}, \ldots, S_1$, as well. In fact, a $P|r_j, u_j|L_{\max}$ problem defined in stage S_{i-1} by setting a due date d_j for each job j equal to its start time in a stage in S_i (Step 5.1). Using ADAU, a schedule of S_{i-1} that has a maximum lateness of $L_{\max}^{i-1}$, is generated (Step 5.2). There are two possible scenarios:

- Scenario 1: $L_{\max}^{i-1} > 0$, it is necessary in this case that all the starting times are right-shifted in all subsequent stages $(S_i, \ldots, S_K)$ by $L_{\max}^{i-1}$ units of time;
- Scenario: $L_{\max}^{i-1} \leq 0$, it is necessary in this case that all the starting times are left-shifted in all subsequent stages $(S_i, \ldots, S_K)$ by $L_{\max}^{i-1}$ units of time;

The value of UB_i is obviously updated accordingly in both cases (Step 5.3).

Consequently, a feasible solution γ_i for $FH_K, \left(\left(PM^{(l)}\right)_{l=1}^{K}\right)|un_{i,j}|C_{\max}$ problem is obtained by repeating this process for stages $S_{i-1}, \ldots, S_1$, in that order.

As a result, we can obtain up to K different schedules $\gamma_1, \ldots, \gamma_K$ for $i = 1, \ldots, K$. The best-derived schedule is denoted by γ with makespan UB.

Algorithm 1: Construction of an initial feasible schedule γ_i

Step 1: *For the problem* $P_{m_i}\left|r_j, u_j, q_{,j}\right|C_{\max}$ *assign a release date* $r_j = re_{i,j}$, *a processing time* $p_j = pr_{i,j}$, an unloading time $u_j = un_{i,j}$, and *a delivery time* $q_j = qe_{i,j}$ *for each* $j \in J$.

Step 2: *Solve* $P_{m_i}\left|r_j, u_j, q_{,j}\right|C_{\max}$ *given in Step 1 using ADAU. Let* $CU_{i,j}$ *be the completion unloading time for each* $j \in J$. *If* $i = K$ *then go to* ***Step 4***.

Step 3: *For* $s = i + 1$ to K ***/downstream* phase**
Begin

Step 3.1: *For the problem* $P_{m_s}\left|r_j, u_j, q_j\right|C_{\max}$ *assign for* $j \in J$ *a release date* $r_j = re_{s,j}$, *a processing time* $p_j = pr_{s,j}$, *an unloading time* $u_j = un_{i,j}$, and *a delivery time* $q_j = qe_{s,j}$.

Step 3.2: *Solve* $P_{m_s}\left|r_j, u_j, q_j\right|C_{\max}$ *given in Step 3.1 using ADAU. Let* $CU_{h,j}$ *be the completion unloading time for each* $j \in J$.
End (For)

Step 4: *Set* $UB_i = \max_{j \in J}\left(CU_{K,j}\right)$. If $i = 1$, *then go to* **Step 6**.

Step 5: *For* $s = i - 1$ *down to* 1 ***/upstream* phase**
Begin

Step 5.1: *For the problem* $P_{m_s}\left|r_j, u_j\right|L_{\max}$ *assign for* $j \in J$ *a release date* $r_j = re_{s,j}$, *a processing time* $p_j = pr_{s,j}$, *an unloading time* $u_j = un_{i,j}$, *and a due date* $d_j = T_{s+1,j}$ where $T_{s+1,j}$ is the staring time of $j \in J$ *in stage* S_{s+1}

Step 5.2: *Solve* $P_{m_s}\left|r_j, u_j\right|L_{\max}$ *given in Step 5.1 using ADAU. Let* $T_{s,j}$ and $L_{\max}^{s}$ *denote the starting time of* j and *the maximum lateness*, respectively.

Step 5.3: *For all stages* S_l $(s + 1 \leq l \leq K)$ *and all* $j \in J$, *Set* $T_{l,j} := T_{l,j} + L_{\max}^{s}$. *Set* $UB_i := UB_i + L_{\max}^{s}$.
End (For)

Step 6: Save the obtained schedule γ_i and its corresponding makespan UB_i.

It is important to highlight the potential benefits of leveraging the symmetry property of the problem to improve solution quality. To support this assertion, an illustrative example is provided below. This example serves to demonstrate how exploiting the symmetry property can lead to enhancements in the overall solution approach and results. By showcasing this specific case, the significance of considering and utilizing symmetry becomes evident, reinforcing the claim that it can positively impact the quality of the

solution obtained. To illustrate the above statement, the following example is provided as supporting evidence.

Example 4. *Considering a five-job and two-operation instance with* $m_1 = 3$ *and* $m_2 = 2$. *The processing and unloading times are include in* Table 5.

Table 5. Processing and unloading times of Example 4.

	j	**1**	**2**	**3**	**4**	**5**
S_1	$pr_{1,j}$	10	3	2	16	12
	$un_{1,j}$	5	5	8	3	5
S_2	$pr_{2,j}$	15	19	6	2	8
	$un_{2,j}$	1	19	6	12	17

By utilizing the initial solution procedure, we have obtained the following feasible schedule.

- In stage 1 (S_1),
  - In machine $M_{1,1}$ the scheduled jobs are 2 and 1.
  - In machine $M_{1,2}$ the scheduled jobs are 5 and 3.
  - In machine $M_{1,3}$ the scheduled job is 4.
- In stage 2 (S_2),
  - In machine $M_{2,1}$ the scheduled jobs are 2 and 1.
  - In machine $M_{2,2}$ the scheduled jobs are 5, 4, and 3.

The makespan of this schedule is 68.

Applying the same procedure (initial solution) for the symmetric problem, we have obtained the following feasible solution:

- In stage 1 $\left(S_1^B\right)$,
  - In machine $M_{1,1}^B$ the scheduled jobs are 2 and 1.
  - In machine $M_{1,2}^B$ the scheduled jobs are 5 and 3.
- In stage 2 $\left(S_2^B\right)$,
  - In machine $M_{2,1}^B$ the scheduled jobs are 2 and 1.
  - In machine $M_{2,2}^B$ the scheduled jobs are 5, 4, and 3.
  - In machine $M_{2,3}^B$ the scheduled jobs are 5, 4, and 3.

The makespan of this schedule is 62.

Hence, exploring the symmetric problem in this example leads to the attainment of a better solution.

4.2. The Improvement Phase

The obtained feasible schedule γ in Phase 1 serves as the input for a sequence of iterative improvements. To stop the process, a convergence criterion needs to be met. In more precise terms, the improvement phase involves iteratively fixing the existing schedules in all stages except one (S_h). Rescheduling in this latter stage S_h requires solving a $P_{m_h}|r_j, u_j|L_{\max}$ problem. In the latter problem, the release dates r_j are set as the completion unloading times $r_j = UN_{h-1,j}$ in the previous stage S_{h-1}. Moreover, the jobs' due dates are defined as the latest starting times in the subsequent stage S_{h+1}. To obtain the latest starting times, all jobs from S_K down to S_{h+1} are right-shifted iteratively. After that and using the ADAU heuristic, the problem $P_{m_h}|r_j, u_j|L_{\max}$ is solved. Subsequently, two cases must be taken into account:

- Case 1: $L^h_{\max} < 0$: in this case, a new improved solution is found in stage S_h with makespan $:= UB + L^h_{\max}$. The new schedule is obtained by left-shifting all the jobs to downstream stages (from S_h down to S_K) by $L^h_{\max}$ unites of time.
- Case 2: $L^h_{\max} \geq 0$: in this case, no improvement is detected.

Remark 1. *There is no maximum lateness associated with the input schedule* γ due to its zero-maximum lateness. Therefore, the optimal solution of the $P_{m_h}|r_j, u_j|L_{max}$ problem satisfies $L^h_{max} \leq 0$.

Algorithm 2 presents a pseudo-code for the improvement phase based on stage S_h as a starting one.

More precisely, an initial stage S_h is selected, and a problem $P_{m_h}|r_j, u_j|L_{\max}$ is constructed following Step 1. The latter problem is solved using the ADAU heuristic (Step 2). The solution may result in a rescheduling of stage S_h if an improvement is detected (Step 3). Similarly, the upstream stages S_{h-1}, S_{h-2},..., S_1 are successively rescheduled according to the same procedure (Step 4). Once the first stage is reached (Step 3.2), the improvement process is iteratively performed for the downstream stages S_2, S_3,..., S_K (Steps 5–8). Reaching this step, the upstream stages are revisited and rescheduled following the same procedure (Step 7.2). If $2K-2$ consecutive problems are solved without any improvement, the process is halted (Step 9).

The obtained schedule γ in Phase 1 is considered to be input for the improvement Phase. The proposed improvement phase could start from any starting stage S_h ($h = 1, \ldots, K$). In the presented improvement phase, each time a different stage is considered to be a starting stage, thus K feasible schedules are obtained, and the best one is selected and denoted γ^*.

Algorithm 2: Improvement phase

Step 0: *Initialization, Set* $= h$, $q = 0$.
Step 1: *Set* $q := q + 1$, *for the problem* $P_{m_k}|r_j, u_j|L_{\max}$
assign a release date $r_j = CU_{k-1,j}$,
a processing time $p_j = p_{k,j}$, *an unloading time* $u_j = un_{k,j}$,
and a due date $d_j = T_{k+1,j}$, $j \in J$.
Step 2: *Solve the problem* $P_{m_k}|r_j, u_j|L_{\max}$ *given in Step 1 using ADAU.*
Let $T_{k,j}$ and $CU_{k,j}$ *denote the start* and completion
unloading times of j ($j \in J$), *respectively.*
Step 3: Step 3.1: *If* $L^k_{\max} < 0$, *Set* $UB := UB + L^k_{\max}$
(an improvement is detected), Set $q = 0$.
Step 3.2: *If* $k = 1$ *go to Step 5.*
Step 4: *Set* $k := k - 1$, If $q \leq 2K - 1$ *go to Step 1, Else go to Step 9.*
Step 5: *Set* $q := q + 1$, *for the problem* $P_{m_k}|r_j, u_j|L_{\max}$
assign a release date $r_j = CU_{k-1,j}$,
a processing time $p_j = p_{k,j}$, *an unloading time* $u_j = un_{k,j}$
and a due date $d_j = T_{k+1,j} (j \in J) \left(d_j = UB \text{ if } k = K \right)$.
Step 6: *Solve the defined problem* $P_{m_k}|r_j, u_j|L_{\max}$ *in Step 5 using ADAU.*
Step 7: Step 7.1: *If* $L^k_{\max} < 0$, *Set* $UB := UB + L^k_{\max}$
(an improvement is detected), Set $q = 0$.
Step 7.2: *If* $k = K$ *go to Step 4.*
Step 8: *Set* $k := k + 1$, If $q \leq 2K - 1$ *go to Step 5.*
Step 9: *Store the obtained schedule* γ^h *and the corresponding makespan* UB.

Remark 2. *To take advantage of the FFSPU's symmetry propriety, the previous two-phase heuristic is systematically applied to the symmetric problem (as defined in Definition 1). By doing so, it is possible to have better solutions.*

5. Computational Results

5.1. Test Problems

Test problems are generated similarly to those in [77]. Essentially, we consider $K \in \{2, 4, 6, 8, 10\}$ and $n \in \{10, 20, 40, 80\}$. Table 6 displays the configurations of stage-machines.

Table 6. The configurations of stages and machines ([77]).

Configuration	2 Stages	4 Stages	6 Stages	8 Stages	10 Stages
1	2-2	2-2-2-2	2-2-2-2-2-2	2-2-2-2-2-2-2-2	2-2-2-2-2-2-2-2-2-2
2	1-2	2-4-4-6	1-2-3-4-5-6	1-1-2-2-3-3-4-4	1-1-2-2-3-3-4-4-5-5
3	1-4	2-4-2-4	1-2-3-1-2-3	1-3-1-3-1-3-1-3-	1-2-3-4-5-1-2-3-4-5
4	3-5	2-3-4-2	1-2-4-4-2-1	1-2-3-4-1-2-3-4	2-2-3-3-4-4-3-3-2-2
5		3-1-2-3	5-5-1-1-5-5	1-2-3-4-4-3-2-1	5-4-3-2-1-1-2-3-4-5
6			4-2-1-1-2-4	5-4-3-2-2-3-4-5	1-2-4-2-1-3-4-4-2-2
7				1-3-2-3-1-4-2-3	5-4-3-2-3-4-5-2-3-5
8					1-3-2-4-1-3-2-4-1-4

The patterns are categorized into the following groups:

- All-equal patterns: like 2-2-2-2
- Increasing patterns: like 1-2-3-4-5-6
- Top patterns: like 1-2-3-4-4-3-2-1
- Valley patterns: like 4-2-1-1-2-4
- Random patterns: like 1-3-2-3-1-4-2-3.

It is important to note that the numbers in the pattern represent the number of machines in the corresponding stage. For instance, Pattern 3–5 indicates that there are three identical parallel machines in the first stage and five identical parallel machines in the second stage.

The processing times are generated uniformly from $[1, 20]$.

The following are the patterns used to generate unloading times:

- Type 1: The unloading time is generated uniformly from $[1, 10]$.
- Type 2: The unloading time is generated uniformly from $[1, 20]$.
- Type 3: The unloading time is generated uniformly from $[1, 40]$.

The three types indicate the importance of unloading time relative to processing time. For example, unloading time is less significant than processing time in the first type (Type 1). In addition, based on the number of jobs, the processing time, the type of unloading time, and the configuration of the stage machine, five instances were generated for each. This totaled 1800 instances. This testbed is highly diversified due to its combination of a variety of different problem sizes, varying machine distribution patterns, and varying processing and unloading times. Hence, we provide a way to evaluate the performance of the proposed procedures based on an unbiased experiment.

5.2. Performance of the Two-Phase Heuristic

5.2.1. Empirical Analysis of the Two-Phase Heuristic

We have coded the two-phase heuristic and used it to solve 1800 instances in the testbed to assess its performance. The global results are included in Table 7. The detailed results according to the type, number of jobs n, and the number of stages K are presented in Table 8. In addition, the relative gap for each instance is calculated as follows:

$$rg = 100 \times \frac{UB - LB}{LB} \tag{12}$$

with UB representing the makespan obtained with the heuristic and LB being the corresponding general lower bound. For each class of instances, the average relative gap MG is also computed. Therefore, the three following performance measures are reported.

Table 7. Types Heuristic global results.

	MT	MG	MaxG
Type 1	8.00	2.71	25.89
Type 2	9.36	2.72	20.36
Type 3	12.40	2.96	21.23
All types	9.92	2.80	25.89

Table 8. Detailed results for Phase 1 (PH1) and Phase 2 (PH2).

			K = 2			K = 4			K = 6			K = 8			K = 10		
	n		MT	MG	MaxG	MT	MG	MaxG	MT	MG	MaxG	MT	MG	MaxG	MT	MG	MaxG
Type 1	10	PH1	0.31	0.80	6.41	0.02	0.00	0.00	0.03	0.26	3.28	2.69	2.60	15.00	1.84	3.49	25.89
		PH2	0.50	0.80	6.41	0.02	0.00	0.00	0.20	0.21	3.28	4.75	2.14	12.90	2.70	2.82	25.89
	20	PH1	1.96	0.89	3.64	3.37	5.57	21.19	6.93	1.35	8.73	0.91	1.85	13.48	0.30	0.75	5.83
		PH2	2.65	0.60	3.64	5.51	4.56	19.21	12.25	1.20	8.73	0.94	1.66	13.48	0.33	0.72	5.83
	40	PH1	22.28	11.98	22.51	3.46	1.90	15.23	0.30	2.84	9.41	1.40	2.79	17.74	4.22	5.51	29.50
		PH2	37.47	10.84	21.26	3.50	1.68	11.72	0.32	2.58	9.41	1.49	2.04	14.52	4.69	4.78	24.50
	80	PH1	37.68	7.10	19.86	8.97	2.37	12.17	12.38	7.68	21.32	3.12	1.37	8.42	4.54	4.82	17.65
		PH2	66.62	5.73	18.95	9.09	2.03	12.17	21.14	6.34	16.67	3.26	1.22	8.42	12.77	4.28	15.44
Type 2	10	PH1	0.65	0.89	5.08	0.04	0.00	0.00	0.06	0.24	2.60	1.90	3.01	13.38	1.96	3.30	14.67
		PH2	0.79	0.70	4.24	0.04	0.00	0.00	0.19	0.21	2.60	3.36	2.78	13.38	2.90	2.72	13.33
	20	PH1	1.22	0.73	3.64	3.69	6.75	21.54	8.81	1.39	7.46	0.75	1.69	9.69	0.51	0.71	4.14
		PH2	1.77	0.45	2.43	6.68	5.90	19.49	12.49	1.25	7.46	0.78	1.45	9.38	0.55	0.70	4.14
	40	PH1	28.87	11.74	20.80	5.89	1.94	13.29	0.54	2.47	12.30	1.89	2.20	7.67	4.60	5.07	22.55
		PH2	55.11	10.12	17.74	5.95	1.90	13.29	0.57	2.08	11.72	1.97	1.86	7.67	5.46	4.34	20.36
	80	PH1	43.78	7.07	18.73	9.46	1.91	10.29	13.94	7.62	21.64	3.34	1.46	8.29	5.61	5.30	15.63
		PH2	75.90	5.92	15.32	9.62	1.64	7.94	23.81	6.35	16.71	3.45	1.36	8.01	14.56	4.60	13.82
Type 3	10	PH1	0.76	0.11	1.02	0.19	0.16	4.00	0.12	0.24	2.22	3.52	3.34	20.76	2.94	3.40	17.94
		PH2	1.31	0.09	0.88	0.20	0.16	4.00	0.22	0.24	2.22	7.64	3.07	20.76	4.65	2.82	16.14
	20	PH1	1.63	0.87	6.67	5.37	6.57	15.47	9.76	1.63	8.02	1.32	2.01	12.68	0.89	1.01	6.59
		PH2	2.59	0.66	5.87	7.97	5.77	13.85	19.15	1.41	7.08	1.35	1.63	12.68	0.92	1.01	6.59
	40	PH1	33.72	12.54	19.17	5.78	1.71	11.16	0.98	3.42	10.29	2.85	2.75	18.15	6.24	6.44	24.58
		PH2	62.34	10.45	17.98	5.85	1.57	11.16	1.01	3.01	9.86	2.98	2.41	14.23	9.74	5.70	21.23
	80	PH1	51.10	6.73	19.98	13.25	1.80	7.76	18.43	8.68	19.73	4.63	1.78	9.26	7.87	5.23	14.33
		PH2	91.27	5.77	17.95	13.39	1.65	6.96	33.97	7.02	15.52	4.83	1.48	7.00	19.35	4.39	12.80

- MT: the average CPU time in seconds.
- MG : the average relative gap.
- $MaxG$: the maximal value of the relative gap.

In the subsequent sections, the term "All types" used in the relevant tables refers to the average value obtained by combining the data from the three types: Type 1, Type 2, and Type 3.

Based on Table 7, we observe that the proposed two-phase heuristic consistently produces high-quality solutions, which makes it highly effective. Indeed, the average required CPU time does not exceed 10 s, and the average relative gap is only 2.80%. We

found, in particular, that for Types 1 and 2, the average relative gaps are 2.71% and 2.72%, respectively. However, the average relative gap and the average CPU time are more important in Type 3, and they are 2.96% and 12.40 s, respectively. Thus, Type 3 instances are harder to solve than the ones of Type 1 and Type 2. Recall that Type 3 is characterized by an important unloading time compared to the processing times.

The detailed performance of Phase 1 (PH1) and Phase 2 (PH2) are presented in Table 8. Based on these more detailed results (presented in Table 8), we observe that the maximum average relative gap (MG) is reached at $n = 40$ and $K = 2$, with a value of 10.84%. In addition, for each number of stage K, the maximum MG is reached for the same number of jobs n in each type. As an example, for $K = 8$, the maximum relative gaps for Types 1, 2, and 3 are reached for $n = 10$ with respective values 2.14%, 2.78%, and 3.07%. Interestingly, the same remark holds for the maximum gap $MaxG$. These instances are identified to be the hardest ones to solve. The average CPU time MT almost reaches its maximum value for $n = 80$ regardless of the configuration.

Furthermore, the required CPU time running the proposed methods is barely longer than 1.5 min, even though it involves solving a set of hard parallel machine problems (approximately).

Notably, when considering $n = 40$ and $n = 80$ with $K = 2$, the mean time MT is observed to be the highest compared to other numbers of stages, namely $K = 4$, $K = 6$, $K = 8$, and $K = 10$. This observation can be attributed to the specific characteristics of the two-stage configuration. In the case where $K = 2$, implying only two stages, the proposed procedures (ADAU), which are designed to handle release dates and delivery times when they are all different from zero, face limitations. This limitation arises due to the specific nature of the configuration. In the first stage, all release dates are equal to zero, while in the second stage, all delivery times are equal to zero. As a result, the ADAU procedures are unable to fully utilize their features and take advantage of the available information due to the absence of variation in release dates and delivery times across the stages. This limitation contributes to the higher mean time MT observed for $n = 40$ and $n = 80$ with $K = 2$ compared to configurations with a higher number of stages. Overall, this observation highlights the impact of the number of stages on the effectiveness of the ADAU procedures, particularly in the cases where the release dates and delivery times lack variation across the stages.

The impact of the configurations on the MT, MG, and MaxG is presented in Table 9.

Table 9. Configuration impact.

	Type 1			Type 2			Type 3			All Types		
Configuration	**MT**	**MG**	**MaxG**	**MT**	**MG**	**MaxG**	**MT**	**MG**	**MaxG**	**MT**	**MG**	**MaxG**
1	27.34	6.98	24.50	32.52	7.15	20.36	40.75	7.56	21.23	33.54	7.23	24.50
2	4.01	0.29	3.17	4.84	0.35	8.42	6.28	0.28	4.00	5.05	0.31	8.42
3	1.13	2.03	12.17	1.27	1.80	13.29	2.00	1.98	11.16	1.47	1.94	13.29
4	7.18	3.40	25.89	7.84	3.23	16.71	11.14	3.67	16.14	8.72	3.43	25.89
5	1.36	0.88	14.52	1.61	0.79	7.67	2.30	1.12	13.77	1.76	0.93	14.52
6	3.38	1.44	8.42	3.80	1.47	8.01	7.18	1.64	14.23	4.78	1.52	14.23
7	12.76	2.71	11.07	14.51	3.07	12.68	19.34	3.88	16.91	15.54	3.22	16.91
8	0.62	4.66	15.44	1.50	5.16	13.82	1.71	4.14	11.48	1.28	4.65	15.44

Based on Table 9, the most challenging configuration to solve is the first one, referred to as configuration 1. This is evident from various metrics, such as the largest values of MT and MG observed among all configurations. Specifically, the values of MT and MG for configuration 1 are MT = 33.54 s and MG = 7.23%, respectively, indicating their prominence compared to the other configurations. Furthermore, when considering the maximum gap (MaxG), configuration 1 ranks among the highest, specifically in second place, with a value of MaxG = 24.5%. This further emphasizes the complexity and difficulty

associated with solving this particular configuration. An interesting observation is that as the unloading times increase, both MT and MG also exhibit an increasing trend. This correlation implies that the higher the unloading times, the more challenging the resolution becomes. Configuration 1 is characterized by the presence of equal parallel machines (two machines) across all stages. The difficulty encountered in solving the corresponding test problems arises from the fact that configuration 1 features a uniform distribution of machines across all stages, leading to a balanced load among these stages. This characteristic, while promoting fairness and equal utilization of resources, also presents a challenge in finding an optimal solution.

Configuration 2 stands out as the configuration with the highest number of optimal solutions, as indicated by the smallest mean relative gap (MG = 0.31). This particular configuration demonstrates an increasing number of parallel machines as we move from the first stage to the last stage. The flexibility provided by this progressive increase in the number of parallel machines enhances the optimization process, ultimately leading to a higher number of optimal solutions. On the other hand, configuration 8, which represents a random pattern, exhibits the smallest mean computation time (MT) with a value of MT = 1.28 s. This suggests that this configuration allows for faster computation and processing of tasks compared to the other configurations. It is worth noting that as the unloading time increases, MT shows an upward trend. For instance, configuration 4 exhibits the following MT values: MT = 7.18 s (Type 1), MT = 7.84 s (Type 2), and MT = 11.14 s (Type 3). This increase in MT with higher unloading times is consistent across all the remaining configurations.

5.2.2. The Second Phase PH2 Implementation Effect

Interestingly, from Table 10, we observe that Phase 2 very often achieves a significant improvement over the solution delivered by Phase 1 in terms of quality and efficiency. A pairwise comparison is conducted for each type of instance to gain a better understanding of what Phase 2 accomplishes compared to Phase 1. These results are displayed in Table 10. These results demonstrate the effectiveness of the improvement phase (Phase 2). As an example, for Type 1, Phase 2 reduces MG and $MaxG$ from 3.18% and 29.5% to 2.71% and 25.89%, respectively, within just 3 s.

Table 10. Comparison between PH1 and PH2 according to types.

		MT	**MG**	**MaxG**
Type 1	PH1	4.92	3.18	29.50
	PH2	8.00	2.71	25.89
Type 2	PH1	5.75	3.16	22.55
	PH2	9.36	2.72	20.36
Type 3	PH1	7.31	3.45	24.58
	PH2	12.40	2.96	21.23
All types	PH1	5.99	3.26	29.5
All types	PH2	9.92	2.80	25.89

A second pairwise comparison is conducted between PH1 and PH2 to compare:

- $(PH2 < PH1)$: The percent of Time PH2 is strictly dominant over PH1.
- $(PH2 = PH1)$: The percentage of Times PH2 and PH1 are the same.

The corresponding results are presented in Table 11. Based on this table, PH2 is strictly dominating PH1 in 34.67% of instances. A slight advantage is observed for Type 3, with a percentage of 36.17%.

Table 11. Types pairwise comparison between PH1 and PH2.

	PH2 < PH1	**PH2 = PH1**
Type 1	33.00	67.00
Type 2	34.83	65.17
Type 3	36.17	63.83
all types	34.67	65.33

It is important to note that PH2 is an iterative procedure that involves traversing all stages multiple times in both backward and forward directions. This thorough exploration significantly increases the execution time required by PH2. Furthermore, similar to PH1, PH2 also experiences longer processing times when the number of stages, K, is set to 2. This is attributed to the same underlying reason explained earlier for the longer time taken by PH1 for $K = 2$.

5.2.3. Symmetric Problem Exploration Impact

A comparison study is performed between the forward problem of Phase 2 (PH2D) and the symmetric problem of Phase 2 (PH2S) to assess the exploration of the symmetric problem (as defined in Definition 1). The global results are reported in Tables 11 and 12.

Table 12. Global comparison between PH2S and PH2D according to types.

		MT	**MG**	**MaxG**
Type 1	PH2D	4.29	3.48	33.93
	PH2S	3.71	2.71	25.89
Type 2	PH2D	4.72	3.44	23.08
	PH2S	4.64	2.72	20.36
Type 3	PH2D	6.21	3.79	24.11
	PH2S	6.19	2.96	21.23

Table 12 shows that exploring the symmetric problem improves the quality of the solution and reduces the average relative gap MG as well as the maximum relative gap $MaxG$ significantly.

Table 13 presents:

- $(PH2D < PH2S)$: The percent of Time PH2D is strictly dominant over PH2S.
- $(PH2D = PH2S)$: the percentage of Times PH2D and PH2S are the same.
- $(PH2D > PH2S)$: the percentage of Times PH2S is strictly dominant over PH2D.

Table 13. Pairwise comparison between PH2D and PH2S.

	PH2D > PH2S	**PH2D = PH2S**	**PH2D < PH2S**
Type 1	39.00	38.00	23.00
Type 2	39.00	35.50	25.50
Type 3	40.83	34.17	25.00
All types	39.61	35.89	24.50

In almost 40% of instances, the symmetric problem strictly improves the quality of the solution, as shown in Table 13. In Type 3, the percentage of instances where the symmetric problem improves the solution is 40.83%. This illustrates the benefits, including the exploration of symmetric problems.

6. Conclusions

In this paper, we have addressed the flexible flow shop scheduling problem with unloading operations and proposed several lower bounds and a heuristic to tackle this complex problem. Our approach involved deriving two types of lower bounds: one based on the relaxation of capacities in all stages except one and the other utilizing the estimation of minimum idle time in each stage determined by solving a polynomial parallel machine problem in the previous stage. These lower bounds were then combined to form a strong general lower bound. Additionally, we developed a two-phase heuristic that effectively solves identical parallel machine-scheduling problems considering release dates, unloading times, and delivery times. The problem's symmetry is a significant characteristic that merits attention. It is worth highlighting that the proposed procedures (lower bounds and heuristic) autonomously investigate the symmetric problem, aiming to enhance the quality of the final solution obtained.

To evaluate the performance of our proposed lower bounds and heuristic, we conducted an extensive experimental study, and the utilized performance measures are the mean computational time and the mean relative gap. The results obtained demonstrated the effectiveness and efficiency of our procedures. We also observed that the problem complexity increases when unloading time becomes more significant than processing time (Type 3). Furthermore, the computational results provide compelling evidence of the efficiency of the developed procedures. Within a reasonable computational time of less than 10 s, the procedures consistently generate solutions that exhibit a satisfactory mean relative gap of 2.8%

Looking ahead, future research efforts should center on exploring alternative approaches to address the Flexible Flow Shop Scheduling Problem with Unloading (FFSPU). One promising avenue is the investigation of metaheuristic algorithms, such as genetic algorithms, particle swarm optimization, or differential evolution. These metaheuristics can offer efficient and effective solutions within acceptable CPU time for FFSPU. Furthermore, hybrid metaheuristic approaches that combine different optimization techniques, such as integrating metaheuristics with mathematical programming or machine-learning algorithms, hold great potential for further improving the performance of FFSPU algorithms. These hybrid methods can leverage the strengths of different optimization paradigms to achieve better solutions and overcome the limitations of individual approaches.

Additionally, it is worthwhile to explore variations of the FFSPU problem. One possibility is to consider a restricted number of operators charged with the unloading operations, which can introduce new constraints and complexities to the scheduling problem. Another avenue is to incorporate setup times into the scheduling process, which would reflect the practical reality of changing configurations and preparation requirements for different jobs. By investigating these variations of the FFSPU problem, the scope of research can be expanded, leading to a more comprehensive understanding of the challenges and potential solutions in different contexts. Conducting further investigation is necessary to explore these possibilities and advance the understanding of FFSPU, ultimately enabling the development of more robust and flexible scheduling methodologies.

However, it is important to acknowledge the limitations of our study. The scalability of our approaches to larger problem instances needs to be examined. Moreover, the proposed heuristic could benefit from enhancements to improve its performance and robustness in handling various problem variations, especially in the case of two stages, where the proposed procedures face difficulties in finding near-optimal solutions in reasonable computation time. Additionally, conducting real-life case studies and validating our methodologies in practical settings would provide valuable insight and further support their applicability.

Funding: This research is supported by the Deputyship for Research and Innovation, Ministry of Education in Saudi Arabia, grant number IFKSUOR3-365-1.

Acknowledgments: The author would like to express gratitude to the Deputyship for Research and Innovation, Ministry of Education in Saudi Arabia, for providing funding support for this research under grant number IFKSUOR3-365-1.

Conflicts of Interest: The author declares no conflict of interest.

References

1. Yue, Q.; Wan, G. Order scheduling with controllable processing times, common due date and the processing deadline. *J. Syst. Sci. Syst. Eng.* **2017**, *26*, 199–218. [CrossRef]
2. Zhang, B.; Pan, Q.-K.; Gao, L.; Meng, L.-L.; Li, X.-Y.; Peng, K.-K. A three-stage multiobjective approach based on decomposition for an energy-efficient hybrid flow shop scheduling problem. *IEEE Trans. Syst. Man Cybern. Syst.* **2019**, *50*, 4984–4999. [CrossRef]
3. Pan, Q.-K.; Wang, L.; Mao, K.; Zhao, J.-H.; Zhang, M. An effective artificial bee colony algorithm for a real-world hybrid flowshop problem in steelmaking process. *IEEE Trans. Autom. Sci. Eng.* **2012**, *10*, 307–322. [CrossRef]
4. Meng, L.; Zhang, C.; Shao, X.; Zhang, B.; Ren, Y.; Lin, W. More MILP models for hybrid flow shop scheduling problem and its extended problems. *Int. J. Prod. Res.* **2020**, *58*, 3905–3930. [CrossRef]
5. Qin, W.; Zhuang, Z.; Liu, Y.; Tang, O. A two-stage ant colony algorithm for hybrid flow shop scheduling with lot sizing and calendar constraints in printed circuit board assembly. *Comput. Ind. Eng.* **2019**, *138*, 106115. [CrossRef]
6. Rossit, D.A.; Tohmé, F.; Frutos, M. The Non-Permutation Flow-Shop Scheduling Problem: A Literature Review. *Omega* **2018**, *77*, 143–153. [CrossRef]
7. Yu, C.; Semeraro, Q.; Matta, A. A genetic algorithm for the hybrid flow shop scheduling with unrelated machines and machine eligibility. *Comput. Oper. Res.* **2018**, *100*, 211–229. [CrossRef]
8. De Felice, L. A Simulation Model for Solving the Flow Shop Scheduling Problem under Uncertainty. 2018. Available online: https://hdl.handle.net/10589/140026 (accessed on 1 January 2023).
9. Li, J.Q.; Sang, H.Y.; Han, Y.Y.; Wang, C.G.; Gao, K.Z. Efficient multi-objective optimization algorithm for hybrid flow shop scheduling problems with setup energy consumptions. *J. Clean. Prod.* **2018**, *181*, 584–598. [CrossRef]
10. Meng, L.; Zhang, C.; Shao, X.; Ren, Y.; Ren, C. Mathematical modelling and optimisation of energy-conscious hybrid flow shop scheduling problem with unrelated parallel machines. *Int. J. Prod. Res.* **2019**, *57*, 1119–1145. [CrossRef]
11. Fattahi, P.; Hosseini, S.M.H.; Jolai, F. A mathematical model and extension algorithm for assembly flexible flow shop scheduling problem. *Int. J. Adv. Manuf. Technol.* **2013**, *65*, 787–802. [CrossRef]
12. Gen, M.; Gao, J.; Lin, L. Multistage-Based Genetic Algorithm for Flexible Job-Shop Scheduling Problem. In *Intelligent and Evolutionary Systems. Studies in Computational Intelligence*; Springer: Berlin/Heidelberg, Germany, 2009; pp. 183–196.
13. Tosun, Ö.; Marichelvam, M.K.; Tosun, N. A literature review on hybrid flow shop scheduling. *Int. J. Adv. Oper. Manag.* **2020**, *12*, 156–194. [CrossRef]
14. Ruiz, R.; Vázquez-Rodríguez, J.A. The hybrid flow shop scheduling problem. *Eur. J. Oper. Res.* **2010**, *205*, 1–18. [CrossRef]
15. Ribas, I.; Leisten, R.; Framiñan, J.M. Review and classification of hybrid flow shop scheduling problems from a production system and a solutions procedure perspective. *Comput. Oper. Res.* **2010**, *37*, 1439–1454. [CrossRef]
16. Lee, T.; Loong, Y. A review of scheduling problem and resolution methods in flexible flow shop. *Int. J. Ind. Eng. Comput.* **2019**, *10*, 67–88. [CrossRef]
17. Liu, F.; Li, G.; Lu, C.; Yin, L.; Zhou, J. A tri-individual iterated greedy algorithm for the distributed hybrid flow shop with blocking. *Expert Syst. Appl.* **2024**, *237*, 121667. [CrossRef]
18. Li, P.; Xue, Q.; Zhang, Z.; Chen, J.; Zhou, D. Multi-objective energy-efficient hybrid flow shop scheduling using Q-learning and GVNS driven NSGA-II. *Comput. Oper. Res.* **2023**, *159*, 106360. [CrossRef]
19. Guan, Y.; Chen, Y.; Gan, Z.; Zou, Z.; Ding, W.; Zhang, H.; Liu, Y.; Ouyang, C. Hybrid flow-shop scheduling in collaborative manufacturing with a multi-crossover-operator genetic algorithm. *J. Ind. Inf. Integr.* **2023**, *36*, 100514. [CrossRef]
20. Liu, Y.; Fan, J.; Zhao, L.; Shen, W.; Zhang, C. Integration of deep reinforcement learning and multi-agent system for dynamic scheduling of re-entrant hybrid flow shop considering worker fatigue and skill levels. *Robot. Comput. Integr. Manuf.* **2023**, *84*, 102605. [CrossRef]
21. Ghodratnama, A.; Amiri-Aref, M.; Tavakkoli-Moghaddam, R. Solving a new bi-objective mathematical model for a hybrid flow shop scheduling problem with robots and fuzzy maintenance time. *Comput. Ind. Eng.* **2023**, *182*, 109349. [CrossRef]
22. Huang, Y.; Deng, L.; Wang, J.; Qiu, W.; Liu, J. Modeling and solution for hybrid flow-shop scheduling problem by two-stage stochastic programming. *Expert Syst. Appl.* **2023**, *233*, 120846. [CrossRef]
23. Gholami, H.; Sun, H. Toward automated algorithm configuration for distributed hybrid flow shop scheduling with multiprocessor tasks. *Knowl. Based Syst.* **2023**, *264*, 110309. [CrossRef]
24. Tran, Q.N.H.; Nguyen, N.Q.; Yalaoui, F.; Amodeo, L.; Chehade, H. Improved formulations and new valid inequalities for a Hybrid Flow Shop problem with time-varying resources and chaining time-lag. *Comput. Oper. Res.* **2023**, *149*, 106018. [CrossRef]
25. Fernandez-Viagas, V.; Molina-Pariente, J.M.; Framinan, J.M. New efficient constructive heuristics for the hybrid flowshop to minimise makespan: A computational evaluation of heuristics. *Expert Syst. Appl.* **2018**, *114*, 345–356. [CrossRef]
26. Li, X.; Guo, X.; Tang, H.; Wu, R.; Liu, J. An improved cuckoo search algorithm for the hybrid flow-shop scheduling problem in sand casting enterprises considering batch processing. *Comput. Ind. Eng.* **2023**, *176*, 108921. [CrossRef]

27. Liu, Y.; Shen, W.; Zhang, C.; Sun, X. Agent-based simulation and optimization of hybrid flow shop considering multi-skilled workers and fatigue factors. *Robot. Comput. Integr. Manuf.* **2023**, *80*, 102478. [CrossRef]
28. Wang, Z.; Deng, Q.; Zhang, L.; Li, H.; Li, F. Joint optimization of integrated mixed maintenance and distributed two-stage hybrid flow-shop production for multi-site maintenance requirements. *Expert Syst. Appl.* **2023**, *215*, 119422. [CrossRef]
29. Utama, D.M.; Primayesti, M.D. A novel hybrid Aquila optimizer for energy-efficient hybrid flow shop scheduling. *Results Control Optim.* **2022**, *9*, 100177. [CrossRef]
30. Shao, W.; Shao, Z.; Pi, D. Modelling and optimization of distributed heterogeneous hybrid flow shop lot-streaming scheduling problem. *Expert Syst. Appl.* **2023**, *214*, 119151. [CrossRef]
31. Gupta, J.N. Two-stage, hybrid flowshop scheduling problem. *J. Oper. Res. Soc.* **1988**, *39*, 359–364. [CrossRef]
32. Hoogeveen, J.A.; Lenstra, J.K.; Veltman, B. Preemptive scheduling in a two-stage multiprocessor flow shop is NP-hard. *Eur. J. Oper. Res.* **1996**, *89*, 172–175. [CrossRef]
33. Marichelvam, M.; Tosun, Ö. Performance comparison of cuckoo search algorithm to solve the hybrid flow shop scheduling benchmark problems with makespan criterion. *Int. J. Swarm Intell. Res. (IJSIR)* **2016**, *7*, 1–14. [CrossRef]
34. Wang, L.; Zhou, G.; Xu, Y.; Wang, S. An artificial bee colony algorithm for solving hybrid flow-shop scheduling problem with unrelated parallel machines. *Control Theory Appl.* **2012**, *29*, 1551–1556.
35. Pan, Q.-K.; Wang, L.; Li, J.-Q.; Duan, J.-H. A novel discrete artificial bee colony algorithm for the hybrid flowshop scheduling problem with makespan minimisation. *Omega* **2014**, *45*, 42–56. [CrossRef]
36. Wang, F.; Tang, Q.; Rao, Y.; Zhang, C.; Zhang, L. Efficient Estimation of Distribution for Flexible Hybrid Flow Shop Scheduling. *Acta. Autom. Sin.* **2017**, *43*, 280–292.
37. Dessouky, M.M.; Dessouky, M.I.; Verma, S.K. Flowshop scheduling with identical jobs and uniform parallel machines. *Eur. J. Oper. Res.* **1998**, *109*, 620–631. [CrossRef]
38. Néron, E.; Baptiste, P.; Gupta, J.N. Solving hybrid flow shop problem using energetic reasoning and global operations. *Omega* **2001**, *29*, 501–511. [CrossRef]
39. Naderi, B.; Gohari, S.; Yazdani, M. Hybrid flexible flowshop problems: Models and solution methods. *Appl. Math. Model.* **2014**, *38*, 5767–5780. [CrossRef]
40. Meng, L.; Zhang, C.; Shao, X.; Ren, Y. MILP models for energy-aware flexible job shop scheduling problem. *J. Clean. Prod.* **2019**, *210*, 710–723. [CrossRef]
41. Oğuz, C.; Ercan, M.F.; Cheng, T.E.; Fung, Y.-F. Heuristic algorithms for multiprocessor task scheduling in a two-stage hybrid flow-shop. *Eur. J. Oper. Res.* **2003**, *149*, 390–403. [CrossRef]
42. Ruiz, R.; Şerifoğlu, F.S.; Urlings, T. Modeling realistic hybrid flexible flowshop scheduling problems. *Comput. Oper. Res.* **2008**, *35*, 1151–1175. [CrossRef]
43. Low, C. Simulated annealing heuristic for flow shop scheduling problems with unrelated parallel machines. *Comput. Oper. Res.* **2005**, *32*, 2013–2025. [CrossRef]
44. Li, J.-Q.; Pan, Q.-K.; Wang, F.-T. A hybrid variable neighborhood search for solving the hybrid flow shop scheduling problem. *Appl. Soft Comput.* **2014**, *24*, 63–77. [CrossRef]
45. Pan, Q.-K.; Dong, Y. An improved migrating birds optimisation for a hybrid flowshop scheduling with total flowtime minimisation. *Inf. Sci.* **2014**, *277*, 643–655. [CrossRef]
46. Peng, K.; Pan, Q.-K.; Gao, L.; Li, X.; Das, S.; Zhang, B. A multi-start variable neighbourhood descent algorithm for hybrid flowshop rescheduling. *Swarm Evol. Comput.* **2019**, *45*, 92–112. [CrossRef]
47. Pan, Q.-K.; Gao, L.; Wang, L.; Liang, J.; Li, X.-Y. Effective heuristics and metaheuristics to minimize total flowtime for the distributed permutation flowshop problem. *Expert Syst. Appl.* **2019**, *124*, 309–324. [CrossRef]
48. Ruiz, R.; Pan, Q.-K.; Naderi, B. Iterated Greedy methods for the distributed permutation flowshop scheduling problem. *Omega* **2019**, *83*, 213–222. [CrossRef]
49. Zheng, Z.-X.; Li, J.-Q.; Duan, P.-Y. Optimal chiller loading by improved artificial fish swarm algorithm for energy saving. *Math. Comput. Simul.* **2019**, *155*, 227–243. [CrossRef]
50. Duan, P.y.; Li, J.q.; Wang, Y.; Sang, H.y.; Jia, B.x. Solving chiller loading optimization problems using an improved teaching-learning-based optimization algorithm. *Optim. Control Appl. Methods* **2018**, *39*, 65–77. [CrossRef]
51. Tian, G.; Zhou, M.; Li, P. Disassembly sequence planning considering fuzzy component quality and varying operational cost. *IEEE Trans. Autom. Sci. Eng.* **2017**, *15*, 748–760. [CrossRef]
52. Li, J.; Duan, P.; Sang, H.; Wang, S.; Liu, Z.; Duan, P. An efficient optimization algorithm for resource-constrained steelmaking scheduling problems. *IEEE Access* **2018**, *6*, 33883–33894. [CrossRef]
53. Zheng, Z.-X.; Li, J.-Q. Optimal chiller loading by improved invasive weed optimization algorithm for reducing energy consumption. *Energy Build.* **2018**, *161*, 80–88. [CrossRef]
54. Manna, A.K.; Akhtar, M.; Shaikh, A.A.; Bhunia, A.K. Optimization of a deteriorated two-warehouse inventory problem with all-unit discount and shortages via tournament differential evolution. *Appl. Soft Comput.* **2021**, *107*, 107388. [CrossRef]
55. Kumar, N.; Manna, A.K.; Shaikh, A.A.; Bhunia, A.K. Application of hybrid binary tournament-based quantum-behaved particle swarm optimization on an imperfect production inventory problem. *Soft Comput.* **2021**, *25*, 11245–11267. [CrossRef]
56. Hao, J.-H.; Li, J.-Q.; Du, Y.; Song, M.-X.; Duan, P.; Zhang, Y.-Y. Solving distributed hybrid flowshop scheduling problems by a hybrid brain storm optimization algorithm. *IEEE Access* **2019**, *7*, 66879–66894. [CrossRef]

57. Liu, S.; Pei, J.; Cheng, H.; Liu, X.; Pardalos, P.M. Two-stage hybrid flow shop scheduling on parallel batching machines considering a job-dependent deteriorating effect and non-identical job sizes. *Appl. Soft Comput.* **2019**, *84*, 105701. [CrossRef]
58. Golneshini, F.P.; Fazlollahtabar, H. Meta-heuristic algorithms for a clustering-based fuzzy bi-criteria hybrid flow shop scheduling problem. *Soft Comput.* **2019**, *23*, 12103–12122. [CrossRef]
59. Wang, U. Top-level design of integrated manufacturing system for capital spaceflight machinery company. *Aerosp. Manuf. Technol.* **2005**, *10*, 8–12.
60. Kurz, M.E.; Askin, R.G. Scheduling flexible flow lines with sequence-dependent setup times. *Eur. J. Oper. Res.* **2004**, *159*, 66–82. [CrossRef]
61. Lin, S.-W.; Gupta, J.N.; Ying, K.-C.; Lee, Z.-J. Using simulated annealing to schedule a flowshop manufacturing cell with sequence-dependent family setup times. *Int. J. Prod. Res.* **2009**, *47*, 3205–3217. [CrossRef]
62. Behnamian, J.; Fatemi Ghomi, S.; Zandieh, M. Development of a hybrid metaheuristic to minimise earliness and tardiness in a hybrid flowshop with sequence-dependent setup times. *Int. J. Prod. Res.* **2010**, *48*, 1415–1438. [CrossRef]
63. Naderi, B.; Zandieh, M.; Roshanaei, V. Scheduling hybrid flowshops with sequence dependent setup times to minimize makespan and maximum tardiness. *Int. J. Adv. Manuf. Technol.* **2009**, *41*, 1186–1198. [CrossRef]
64. Qiao, Y.; Wu, N.; He, Y.; Li, Z.; Chen, T. Adaptive genetic algorithm for two-stage hybrid flow-shop scheduling with sequence-independent setup time and no-interruption requirement. *Expert Syst. Appl.* **2022**, *208*, 118068. [CrossRef]
65. Nahhas, A.; Kharitonov, A.; Alwadi, A.; Turowski, K. Hybrid Approach for Solving Multi-Objective Hybrid Flow Shop Scheduling Problems with Family Setup Times. *Procedia Comput. Sci.* **2022**, *200*, 1685–1694. [CrossRef]
66. Oujana, S.; Yalaoui, F.; Amodeo, L. A linear programming approach for hybrid flexible flow shop with sequence-dependent setup times to minimise total tardiness. *IFAC-PapersOnLine* **2021**, *54*, 1162–1167. [CrossRef]
67. Gupta, J.N.; Tunc, E.A. Scheduling a two-stage hybrid flowshop with separable setup and removal times. *Eur. J. Oper. Res.* **1994**, *77*, 415–428. [CrossRef]
68. Szwarc, W.; Gupta, J.N. A flow-shop problem with sequence-dependent additive setup times. *Nav. Res. Logist. (NRL)* **1987**, *34*, 619–627. [CrossRef]
69. Botta-Genoulaz, V. Hybrid flow shop scheduling with precedence constraints and time lags to minimize maximum lateness. *Int. J. Prod. Econ.* **2000**, *64*, 101–111. [CrossRef]
70. Chang, J.; Yan, W.; Shao, H. Scheduling a two-stage no-wait hybrid flowshop with separated setup and removal times. In Proceedings of the 2004 American Control Conference, Boston, MA, USA, 30 June–2 July 2004; pp. 1412–1416.
71. Cheng, M.; Sarin, S.C. Two-stage, multiple-lot, lot streaming problem for a 1+ 2 hybrid flow shop. *IFAC Proc. Vol.* **2013**, *46*, 448–453. [CrossRef]
72. Brucker, P. Some Problems in Combinatorial Optimization. In *Scheduling Algorithms*; Springer: Berlin/Heidelberg, Germany, 1998; pp. 11–35. [CrossRef]
73. Gupta, J.N.; Tunc, E.A. Schedules for a two-stage hybrid flowshop with parallel machines at the second stage. *Int. J. Prod. Res.* **1991**, *29*, 1489–1502. [CrossRef]
74. Carlier, J. Scheduling jobs with release dates and tails on identical machines to minimize the makespan. *Eur. J. Oper. Res.* **1987**, *29*, 298–306. [CrossRef]
75. Pinedo, M.L.; Pinedo, M.L. Deterministic models: Preliminaries. In *Scheduling: Theory, Algorithms, and Systems*; Springer: Cham, Switzerland, 2016; pp. 13–32.
76. Gharbi, A.; Haouari, M. An approximate decomposition algorithm for scheduling on parallel machines with heads and tails. *Comput. Oper. Res.* **2007**, *34*, 868–883. [CrossRef]
77. Vandevelde, A.; Hoogeveen, H.; Hurkens, C.; Lenstra, J.K. Lower bounds for the head-body-tail problem on parallel machines: A computational study of the multiprocessor flow shop. *INFORMS J. Comput.* **2005**, *17*, 305–320. [CrossRef]