Combining Hash-based Tabu Search and Frequent Pattern Mining for Job-Shop Scheduling

Mingjie Li [a], Jin-Kao Hao [b] and Qinghua Wu [c] (Corresponding Author)
[a] Zhongnan University of Economics and Law, 430073 Wuhan, China
mingjie.li@zuel.edu.cn
[b] LERIA, Université d'Angers, 2 Boulevard Lavoisier, 49045 Angers, France
jin-kao.hao@univ-angers.fr
[c] School of Management, Huazhong University of Science and Technology, 430074 Wuhan, China
qinghuawu1005@gmail.com

IISE Transactions, 1–24, 2025. https://doi.org/10.1080/24725854.2025.2532071

Abstract

The Job Shop Scheduling Problem (JSP) is a classical combinatorial optimization problem that involves scheduling a set of jobs on a set of machines to minimize the makespan. Because of its importance, the JSP has been extensively studied in the fields of operations research and manufacturing. This study presents a highly effective hybrid approach that integrates tabu search with frequent pattern mining within a population-based search framework for addressing the JSP. Of particular interest of the tabu search is its novel tabu strategy, which employs hash techniques to accurately mark visited solutions and rapidly determine the tabu status of neighboring solutions in the local search. Such a tabu strategy enables a more efficient exploration of the solution space. Furthermore, the hybrid approach fully embraces the potential of frequent pattern mining by incorporating a frequent job order recognition method to identify promising solution structures. These structures are not only used to generate new promising solutions but also to guide the local search process. Extensive experiments on the benchmark instances widely used in the literature show that the hybrid algorithm performs remarkably well, by improving 13 best-known results (new upper bounds). Additional experiments are presented to gain insight into the role of the key elements of the algorithm.

Keywords: Job-shop scheduling; Frequent pattern based search; Hash-based prohibition; Heuristics.

1 Introduction

The job shop scheduling problem (JSP) is not only one of the most notorious and intractable NP-hard combinatorial optimization problems (Garey et al., 1976), but also one of the most critical and challenging scheduling problems in the fields of operations management and production planning. Over the decades, the JSP has attracted significant research attention,

due to its broad applications in various domains, not limited to industrial manufacturing and job-oriented production environments (Lamorgese and Mannino, 2019; Etesami, 2022).

Because of the theoretical and practical importance of the JSP, numerous solution methods have been proposed in the literature to effectively tackle the problem, including exact algorithms, heuristics, and metaheuristic. Exact techniques based on branch and bound (Barker and McMahon, 1985; Carlier and Pinson, 1989), Lagrangian relaxation (Kaskavelis and Caramanis, 1998; Chen and Luh, 2003), mixed integer programming (Lamorgese and Mannino, 2019; Naderi et al., 2023), and constraint programming (Ku and Beck, 2016; Naderi et al., 2023) can find optimal solutions for problem instances with no more than 400 operations (except trivial cases). However, their computational requirements grow exponentially with problem size, making them impractical for larger instances. Therefore, studies on heuristic and metaheuristic approaches have received increasing attention over the last 30 years. These methods provide a good balance between computational efficiency and solution quality, thus gaining wide acceptance for tackling the JSP.

Within the class of metaheuristic methods for the JSP, the most efficient methods are usually dedicated local search algorithms (Vaessens et al., 1996; Grimes and Hebrard, 2015) and their combinations with population-based search frameworks. In particular, tabu search has maintained its dominance among state-of-the-art local search methods for the past two decades (Cheng et al., 2016; Xie et al., 2022, 2023). Two typical tabu strategies are widely applied in the JSP literature: one preventing reversing moves (Beck et al., 2011) and the other employing a partial solution-based tabu strategy (Zhang et al., 2007). Nevertheless, studies by Zhang et al. (2007); Cheng et al. (2016); Peng et al. (2015) have revealed notable limitations in these tabu strategies: (i) incorrectly marking unvisited solutions, (ii) repetitive visits to the same solutions, and (iii) the high time complexity associated with recording partial solutions and checking tabu status. Therefore, we conjecture that employing a tabu strategy capable of accurately tracking visited solutions and quickly determining tabu status could further improve the performance of tabu search.

The combination of frequent pattern mining and optimization methods has proven to be a highly effective search framework for solving challenging optimization problems (Reddy et al., 2012; Umetani, 2015; Zhou et al., 2020, 2023, 2024). Especially in the context of the JSP, frequent pattern mining can help discover and identify useful scheduling-specific

rules and information from high-quality solutions encountered, such as the precise order of job operations on different machines and the synchronization of job dependencies. These scheduling-specific rules and information, when fully leveraged and exploited, can be used to guide the local search algorithms to make more informed search decisions and improve search effectiveness.

The above observations and considerations motivate us to develop a more robust heuristic algorithm by combining hash-based tabu search and frequent pattern mining within a population-based search framework to solve the JSP. The aim of this paper is twofold. The first goal is to enrich the JSP literature with a new effective solution approach, capable of finding higher-quality upper bounds with reasonable computing effort. The other goal is to investigate methodological contributions on how to mark visited solutions effectively to overcome the main limitations experienced by the traditional tabu strategy, and on how to use scheduling-specific rules and knowledge to guide the local search. The detailed contributions of this work are summarized as follows:

- **Methodological Contributions:** First, we devise a novel tabu strategy for the JSP employing hashing techniques to precisely record each visited solution and determine the tabu status of neighboring solutions. Our tabu search method employs hash functions to accurately track visited solutions, thereby avoiding missing out any high-quality non-visited solution and effectively preventing short-term and long-term cycling. We design fast computation and update techniques, which allow us to determine the tabu status of neighboring solutions in constant time. Given the generality of the proposed tabu strategy, it is applicable to various scheduling problems like flow shop scheduling (Minella et al., 2008), flexible job shop scheduling (Vandevelde et al., 2005), and machine scheduling (Kowalczyk and Leus, 2018).

  Second, we explore the usefulness of frequent pattern mining in solving the JSP by integrating it and tabu search within a population-based search framework. The resulting hybrid algorithm (denoted by HTSFP) uses frequent pattern mining to discover scheduling-specific rules and knowledge about the order and precedence of job operations from high-quality solutions in the population, which are then used to generate promising new initial solutions for the tabu search procedure and reduce the search space explored by the algorithm. Until now, frequent pattern mining has

rarely been explored in the context of solving the JSP, so this work fills this gap and demonstrates its usefulness for better solving the problem.

- **Computational Contributions:** We show extensive experiments to validate the effectiveness and competitiveness of the proposed hybrid method by comparing it with state-of-the-art JSP algorithms from the literature. This study demonstrates that, for the benchmark instances widely used in the literature, the hybrid algorithm statistically outperforms the state-of-the-art algorithms. In particular, it is able to discover 13 improved best results (new upper bounds) on these commonly used benchmarks. Additionally, we show an in-depth analysis on two critical algorithmic components: the hash-based tabu strategy and the frequent pattern mining procedure, to provide insights into their respective roles in the hybrid algorithm.

The remaining sections are organized as follows. In Section 2, a comprehensive literature review for the JSP is provided. Section 3 presents the problem definition and formulation of the JSP. The main algorithmic components of the proposed HTSFP algorithm are detailed in Section 4. Section 5 presents the computational results of our algorithm and compares it with several state-of-the-art methods. Section 6 provides an analysis on the algorithmic usefulness of two critical components. Finally, Section 7 summarizes the conclusion.

2 Literature Review

This section provides a review of the literature on solving the JSP using tabu search and hybrid algorithms.

2.1 Tabu Search for the JSP

Starting with an early algorithm by Taillard (1994), tabu search approaches have consistently represented the state of the art in achieving high-quality solutions for the JSP due to their ability to balance exploration and exploitation in the search space effectively. By maintaining a memory structure that guides the search process, tabu search can efficiently navigate complex solution spaces, avoid getting stuck in local optima, and converge towards high-quality solutions. Several comprehensive reviews of the JSP methods (Błażewicz,

2019; Michael, 2012) have identified tabu search as one of the most effective standalone metaheuristic for the JSP. Some early foundational studies also demonstrated that tabu search outperforms standalone genetic algorithm (GA) and simulated annealing (SA) for the JSP. For instance, Nowicki and Smutnicki (1996) demonstrated through computational experiments that tabu search outperforms SA. Jain and Meeran (1999) reviewed heuristic methods for the JSP and concluded that tabu search generally outperforms simulated SA due to SA's parameter sensitivity and slower convergence, which results from its random-walk behavior caused by probabilistically accepting worse solutions. Gonçalves et al. (2005) demonstrated by computational experiment that pure GA underperforms tabu search for the JSP but achieves better results when hybridized with tabu search. Yamada and Nakano (1992) also highlighted limitations of pure GA for the larger JSP instances, noting that GA is prone to premature convergence. This occurs because GA relies on crossover and mutation for exploration but ofen lack localized intensification unless hybridized with local search.

Tabu search approaches for the JSP are typically built on a foundation of one or more problem-specific move operators to identify promising feasible and high-quality solutions in the neighborhood of a given solution. The most famous neighborhood structures are N5, N6, N7, and N8. As shown in Figure 1, the core idea of these neighborhood structures is to change the critical path of an original scheduling scheme. Precisely, the N5 neighborhood (Nowicki and Smutnicki, 1996) is to swap the first two operations or the last two operations. The N6 neighborhood (Balas and Vazacopoulos, 1998) consists of moving an operation to the beginning of the block or moving an operation to the end of the block. Based on the N6 neighborhood, the N7 neighborhood (Zhang et al., 2007) further considers moving the first and the last operation inside the critical block. More recently, Xie et al. (2023) present the N8 neighborhood by further extending N7 neighborhood.

In addition to neighborhood structure, the tabu strategy is another critical component in tabu search. It typically employs a "tabu list" to keep track of recently visited solutions and prevent the search from short-term cycling. As shown in Table 1, currently, there are two typical tabu strategies widely applied in the literature. An early tabu strategy for the JSP is based on the attributes of moves (Nowicki and Smutnicki, 1996, 2005; Pardalos and Shylo, 2006; Beck et al., 2011), where the reverse move of a performed move is forbidden for

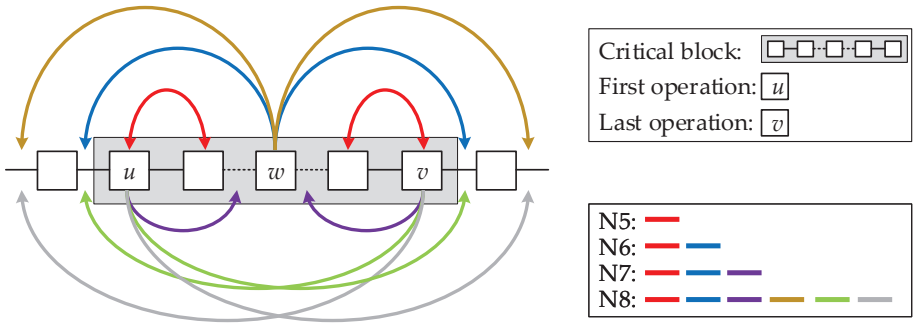


Figure 1: Illustration of the Famous N5-N8 Neighborhoods.

Table 1: Different Tabu Strategies for the JSP in the Literature

Tabu strategy	Description	Literature
Move based tabu strategy	When operations u and v are swapped, the reverse move is forbidden for the next few iterations.	Nowicki and Smutnicki (1996, 2005); Pardalos and Shylo (2006); Beck et al. (2011)
Partial solution based tabu strategy	When operation u is relocated to a position behind operation v (or v is positioned ahead of u), moves resulting in the same sequence of operations between u and v $(u, \ldots, i, \ldots, v)$ and their corresponding machine positions $(p_u, \ldots, p_i, \ldots, p_v)$ are forbidden.	Zhang et al. (2007, 2008); Peng et al. (2015); Xie et al. (2023, 2022)

the next few iterations called tabu tenure. This restriction aims to prevent the search from rapidly returning to the recently visited solution or avoid short-term cycles between two or more consecutively visited solutions. Subsequently, with the application of more complex move operators in TS for the JSP, a partial solution based tabu strategy was proposed by Zhang et al. (2007), where a sequence of operations and their positions in the machine between two swapped operations is recorded in the tabu list, and any move resulting in the same sequence and positions of operations are deemed forbidden. Specifically, if a move involves swapping two operations u and v, any move that results in the same sequence of operations and positions (i.e., operations from u to v $(u, \ldots, w, \ldots, v)$ and their corresponding machine positions from u to v $(p_u, \ldots, p_w, \ldots, p_v)$ is prohibited. In recent years, this

tabu strategy has been extensively employed (Zhang et al., 2008; Peng et al., 2015; Xie et al., 2023, 2022) and is regarded as more effective than move based strategy for the JSP.

However, there are some potential side effects of implementing the above traditional tabu strategies. First, they may incorrectly mark many unvisited solutions as tabu, potentially missing high-quality solutions. Second, early tabu decisions may become outdated, leading to repeated visits to the same solutions and the risk of long-term cycling. Furthermore, the implementation of the partial solution-based tabu strategy, which has been widely adopted recently, requires a high time complexity for recording partial solutions and checking tabu status, which significantly affects the overall efficiency of the tabu search method. A key contribution of this study is the introduction of a novel hashing technique that uses hash functions to accurately mark visited solutions and quickly determine the tabu status of candidate solutions. Our proposed tabu strategy can effectively address the challenges faced by traditional tabu strategies.

2.2 Hybrid Algorithms for the JSP

To achieve a more balanced search between intensification and diversification, tabu search has been integrated with various population-based search frameworks in the literature, resulting in the development of several hybrid evolutionary algorithms. For instance, Peng et al. (2015) devised a TS/PR algorithm that blends tabu search with a path-relinking procedure. Their TS/PR algorithm successfully updated the upper bounds for 49 benchmark instances. Cheng et al. (2016) introduced a hybrid evolutionary algorithm that combines tabu search with the longest common sequence based recombination operator. Their approach successfully identified better upper bounds for two benchmark instances. Gonçalves and Resende (2014) introduced a tabu search algorithm based on an extension of the graphical method proposed by Akers Jr (1956), and applied it within a biased random-key genetic algorithm. González et al. (2015) proposed a scatter search algorithm that uses tabu search and path relinking in its core. Xie et al. (2022) proposed a hybrid algorithm that incorporates innovative crossover and mutation operators in the genetic algorithm part and employs an expanded neighborhood structure in the tabu search part. The computation results show that their proposed hybrid algorithm outperforms previous algorithms in terms of both computational efficiency and solution quality. Constraint programming has proven

to be highly effective for real-world scheduling problems due to its scheduling-specific inference capabilities. Combining constraint programming with tabu search forms another important hybrid approach for solving the JSP. An example of this hybrid approach can be found in Beck et al. (2011), where the authors presented a hybrid algorithm for the JSP that leverages both the fast, broad search capabilities of tabu search with domain-specific inference inference capabilities of constraint programming.

Finally, in recent years, the emergence of new manufacturing modes, such as distributed manufacturing, intelligent manufacturing, and green manufacturing, along with advances in data analysis and artificial intelligence (including machine learning and deep learning), has led to the development of several hybrid algorithms that combine learning-based techniques with optimization methods (Lu et al., 2025). These hybrid algorithms aim to address various new JSP models that incorporate a wide range of realistic requirements and practical production environments. For example, Liu et al. (2023) combine machine learning with mathematical optimization, improving decision-making in complex job shops. Liu et al. (2024) develop an exact algorithm and a machine learning heuristic for stochastic lot streaming and scheduling, providing robust solutions for uncertain environments. Zhang et al. (2023) offer a comprehensive survey on genetic programming and machine learning techniques for heuristic design in job shop scheduling. For a recent overview of the latest advances in applying hybrid algorithms to new JSP models, interested readers can refer to Xiong et al. (2022).

3 Problem Definition and Notations

The JSP can be formally defined as follows: Let $\mathcal{J}$ be a finite set of jobs and $\mathcal{M}$ be a finite set of machines. Each job $j \in J$ must be processed exactly once on each of the $|\mathcal{M}|$ machines. Thus, each job j is composed of an ordered operation list $(O_j^1, O_j^2, \ldots, O_j^{|\mathcal{M}|})$, where each operation O_j^h $(1 \leq h \leq |\mathcal{M}|)$ must be processed on a specific machine, and the ordered list of operations for job j is processed on a corresponding machine list $(\sigma_j^1, \sigma_j^2, \ldots, \sigma_j^{|\mathcal{M}|})$. Additionally, for each job $j \in \mathcal{J}$ and each machine $m \in \mathcal{M}$, we are given a non-negative integer p_{jm}, which represents the processing time of j on m. Each machine can handle only one operation at a time, and once an operation is initiated on a given machine, it

must be completed on that machine without interruption. The objective is to minimize the makespan, denoted as $C_{\max}$, which corresponds to the maximum completion time of the last operation among all jobs. A mathematical formulation of the problem is shown in Appendix A.1.

4 Hybrid Algorithm Combining Hash-based Tabu Search and Frequent Pattern Mining

To achieve a more balanced search between intensification and diversification, this work proposes a hybrid algorithm (denoted by HTSFP) for the JSP, which integrates hash-based tabu search and frequent pattern mining within a general population-based search framework.

4.1 Main Scheme

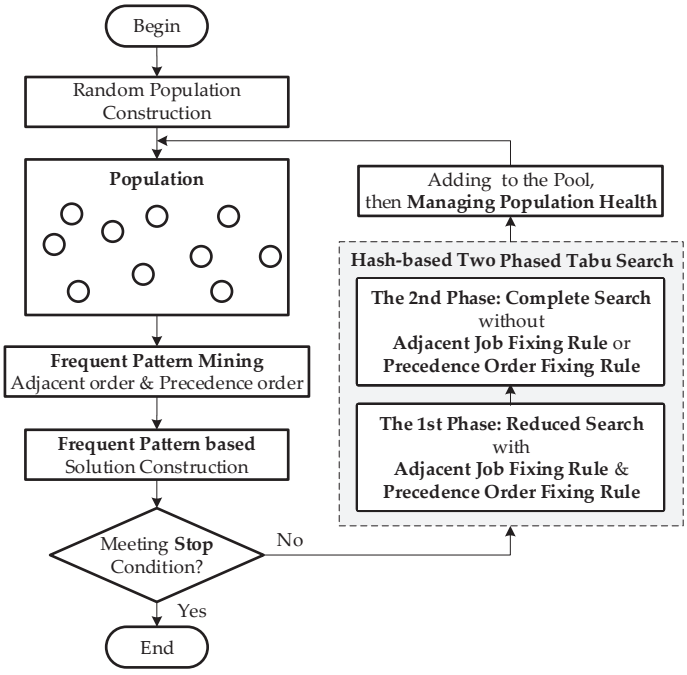


Figure 2: The General Scheme of the HTSFP algorithm.

Figure 2 summarizes the general scheme of the HTSFP algorithm. It begins with an initial population of $|Pop|$ distinct random solutions. Subsequently, the algorithm enters

its main loop to evolve the population iteratively. At each generation, a frequent pattern mining procedure is first applied to identify two types of scheduling order information from the population (Section 4.3). The first one involves the adjacent order relationship between job operations in the job sequence of the same machine, while the second one involves the precedence order between job operations in the job sequence of the same machine. In our context, two jobs j and k are adjacent jobs in the job sequence of machine m, if j and k are immediate successor and predecessor in the job sequence of machine m. Additionally, if job j precedes (i.e., not necessarily immediately) job k in the job sequence of machine m, we say that there is a precedence order between j and k in the job sequence of machine m. The first type of scheduling information is first applied to generate a new starting solution by preserving the adjacent job order frequently occurring in the job sequence of the same machine across all solutions in the population. Afterward, this new starting solution is further improved by the hash-based two phased tabu search procedure. The first phase of the tabu search involves a reduced search procedure (Section 4.5.3), where the two types of scheduling order information are employed to fix certain job operation orders frequently appearing in the population. This can considerably reduce the search space and allow the algorithm to explore the search space in a more focused manner. However, it is possible that some orders can be wrongly fixed. To mitigate this risk, the improved solution is further refined through the second phase of the tabu search procedure (Section 4.5.4), which considers all possible candidate neighboring solutions by ignoring the prefixed order. Both search phases rely on the tabu search framework, where a hash-based tabu strategy is applied to precisely record visited solutions and quickly determine the tabu status of candidate neighboring solutions (Section 4.5.2). Finally, the refined solution is used to update the population through a population management procedure (Section 4.6). The process repeats until a given time limit $t_{\max}$ is reached. The best solution S^* found during the search is finally returned as the result of the algorithm.

4.2 Solution Encoding and Decoding

We adopt the solution representation proposed by Falkenauer and Bouffouix (1991) and widely used in the literature (Zhang et al., 2007; Xie et al., 2023; Peng et al., 2015). This representation is based on the concept that once the sequence of job operations on machines

is established, a unique scheduling scheme can be derived. Specifically, a solution to the JSP is encoded as a chromosome comprising a set of $|\mathcal{M}|$ substrings, with each substring representing the job sequence processed on each machine, denoted by a permutation of $\{1, \ldots, |\mathcal{J}|\}$. Table 2 illustrates an example of the JSP featuring three machines and three jobs. Here, substrings $(3, 2, 1)$, $(2, 1, 3)$, and $(1, 3, 2)$ represent the job sequences processed on machines 1, 2, and 3, respectively.

Table 2: An Instance of the JSP with Format "{Machine, Processing Time}"

Job	1st operation	2nd operation	3rd operation
1	{3, 2}	{2, 2}	{1, 3}
2	{2, 1}	{1, 3}	{3, 2}
3	{1, 2}	{2, 2}	{3, 1}

Table 3: Sequences of the operations processed on each machine deduced from the encoded chromosome $\{(3, 2, 1), (2, 1, 3), (1, 3, 2)\}$

Machine	1st operation	2nd operation	3rd operation
1	O_3^1	O_2^2	O_1^3
2	O_2^1	O_1^2	O_3^2
3	O_1^1	O_3^3	O_2^3

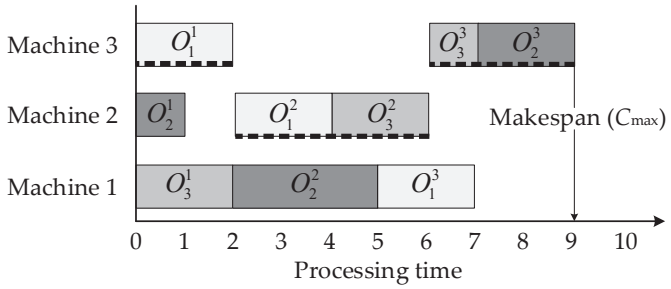


Figure 3: Example of the JSP with the Solution $\{(3, 2, 1), (2, 1, 3), (1, 3, 2)\}$.

Conversely, from the encoded chromosome $\{(3, 2, 1), (2, 1, 3), (1, 3, 2)\}$, we can also deduce the only corresponding schedule as shown in Figure 3 using the decoding method described in Falkenauer and Bouffouix (1991) and Ren and Wang (2012). Recall that O_j^h is the h-th operation of job j (see Section 3). According to Table 2, operations O_3^1, O_2^2,

O_1^3 must be scheduled on machine 1, operations O_2^1, O_1^2, O_3^2 must be scheduled on machine 2 and operations O_1^1, O_2^3, O_3^3 must be scheduled on machine 3. And the schedule must meet the job process constraint $O_1^1 - O_1^2 - O_1^3$, $O_2^1 - O_2^2 - O_2^3$ and $O_3^1 - O_3^2 - O_3^3$, where $O_{j1}^{h1} - O_{j2}^{h2}$ means that operation O_{j1}^{h1} must be scheduled before O_{j2}^{h2}. The substrings $\{(3,2,1),(2,1,3),(1,3,2)\}$ represent the sequences of the operations processed on machine 1, 2 and 3, respectively. Therefore, the sequences of the operations processed on machine 1 is $O_3^1 - O_2^2 - O_1^3$, machine 2 is $O_2^1 - O_1^2 - O_3^2$ and machine 3 is $O_1^1 - O_3^3 - O_2^3$ (see Table 3). In the decoding method, an operation O_j^h can be scheduled immediately if and only if both its preceding operation in its job's operation sequence and its preceding operation in its machine's operation sequence have been completed. Based on the above schedule rule, we schedule operation O_3^1 on machine 1, operation O_2^1 on machine 2, and operation O_1^1 on machine 3 at time 0. Then at time 1, operation O_2^1 is completed, however, we cannot schedule operation O_2^2 immediately because its preceding operation in the operation sequence of its machine (i.e., O_3^1 on machine 1) has not been completed. At time 2, operations O_1^1 and O_3^1 are completed, all three machines are idle at this moment. Therefore, we schedule operation O_2^2 on machine 1, operation O_1^2 on machine 2 at time 2. Note that at this moment, we cannot schedule operation O_3^2 on machine 2 because the operation sequence on machine 2 is $O_2^1 - O_1^2 - O_3^2$, therefore, O_3^2 can be scheduled only after the completion of O_1^2. In a similar manner, we schedule operation O_3^2 on machine 2 at time 4, operation O_1^3 on machine 1 at time 5, operation O_3^3 on machine 3 at time 6, and operation O_2^3 on machine 3 at time 7.

4.3 Frequent Pattern Mining

Our hybrid algorithm maintains a population of elite solutions. Two types of useful frequent patterns related to both the adjacent order and precedence order of jobs on each machine are mined and extracted from the solutions in the population. In the context of job shop scheduling, frequent patterns refer to recurring sequences or orders of job operations on machines that occur frequently in historical or current scheduling solutions. The identification and analysis of these frequent patterns contribute significantly to the efficiency and effectiveness of scheduling algorithms. On the other hand, the adjacent order and precedence order of jobs on machines is crucial for optimizing the completion time of the jobs. Therefore, frequent adjacent order patterns and frequent precedence order patterns

represent two important types of frequent patterns in job shop scheduling.

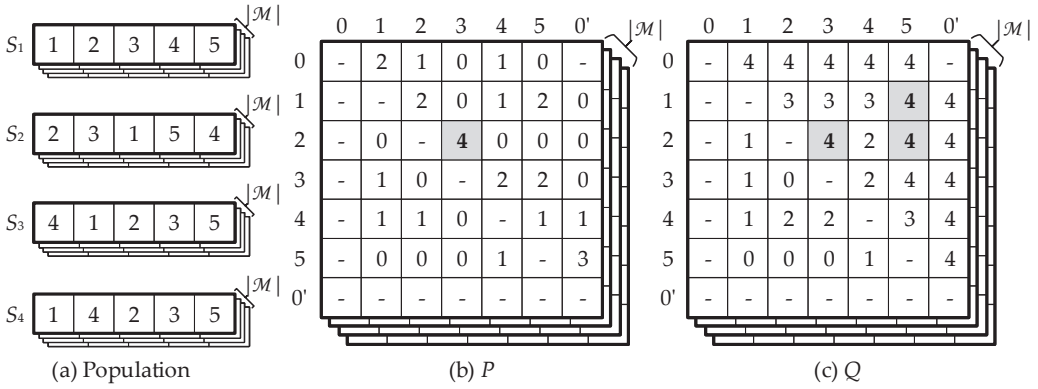


Figure 4: The Example of Mining Frequent Patterns from the Population.

Given a population of solutions that are initially randomly constructed and gradually converge towards better solutions as generations progress, the hybrid algorithm first identifies common adjacent orders and precedence orders of job operations on machines based on the structures of these solutions. It then records these identified order patterns in corresponding matrices. Specifically, for each machine m, an adjacent order matrix P^m is introduced, where the element P^m_{jk} represents the frequency of jobs j and k being immediate successors and predecessors in the job sequence of machine m across all solutions in the population. To facilitate tracking the initial and last jobs in the job sequence of machine m, two dummy jobs, labeled as 0 and $0'$, are introduced. Using a population with four JSP solutions as an example, and assuming that the job sequences on machine 1 of these four solutions are respectively $(1, 2, 3, 4, 5)$, $(2, 3, 1, 5, 4)$, $(4, 1, 2, 3, 5)$, $(1, 4, 2, 3, 5)$, as illustrated in Figure 4(a). The corresponding precedence matrix P^1 is presented in Figure 4(b). For instance, since jobs 2 and 3 are immediate successor and predecessor in the job sequence of machine 1 for all four solutions, we have $P^1_{23} = 4$.

Similarly, for each machine m, we introduce a precedence order matrix Q^m, where the element Q^m_{jk} denotes the frequency with which job j precedes (not necessarily immediately precedes) job k on machine m. Figure 4(c) illustrates the corresponding precedence order matrix Q^1 for the example presented in Figure 4(a). For instance, as job 1 precedes job 5 in the job sequence of machine 1 for all four solutions, we have $Q^1_{15} = 4$.

4.4 The Frequent Pattern based Method for Solution Construction

The JSP is a typical sequence problem that involves finding a job sequence on each machine that satisfies a given set of constraints while minimizing the makespan. The design of this frequent pattern based method relies on the observation that some common adjacent job orders frequently appear in the job sequence of many high-quality solutions. This suggests that if these adjacent job orders frequently appear in the job sequence of the same machine across the high-quality solutions, there is a strong chance that these adjacent job orders also exist in the job sequence of the global optimum as shown in Section 6.2. Therefore, algorithms can take advantage of these frequently recurring job orders, which lead to successful schedules, to construct more promising initial solutions. This, in turn, facilitates a more efficient exploration of the solution space.

Based on the mined frequent adjacent order patterns stored in the matrix P^m ($m \in \mathcal{M}$), our frequent pattern-based method constructs a new JSP solution by considering the machines one by one. For each considered machine m, the frequent pattern-based method is to find a job sequence that maximizes the total frequency of the adjacent order between immediate successor and predecessor jobs. Specifically, this problem can be formulated as the following model:

$$\max \quad \sum_{j\in\{0\}\cup\mathcal{J}} \sum_{k\in\mathcal{J}\cup\{0'\}} P^m_{jk}\eta_{jk} \tag{1}$$

$$\text{s.t.} \quad \sum_{k\in\mathcal{J}\cup\{0'\}} \eta_{jk} = 1, \qquad \forall j \in \{0\}\cup\mathcal{J} \tag{2}$$

$$\sum_{j\in\{0\}\cup\mathcal{J}} \eta_{jk} = 1, \qquad \forall k \in \{0'\}\cup\mathcal{J} \tag{3}$$

$$\sum_{j\in\mathcal{U}} \sum_{k\in\mathcal{U}} \eta_{jk} \le |\mathcal{U}| - 1, \qquad \forall \mathcal{U} \subset \mathcal{J}, 2 \le |\mathcal{U}| \le |\mathcal{J}| \tag{4}$$

$$\eta_{jk} \in \{0,1\}, \qquad \forall j \in \{0\}\cup\mathcal{J}\ \forall k \in \{O'\}\cup\mathcal{J} \tag{5}$$

where η_{jk} is a binary decision variable equal to 1 if job j immediately precedes k in the constructed job sequence for the currently considered machine, and 0 otherwise. Constraints (2) and (3) ensure that each job is assigned to exactly one position on machine m, guaranteeing that every job is processed once on machine m. Constraints (4) state that for any job subset $\mathcal{U}$ with at least two jobs and at most $|\mathcal{J}|-1$ jobs, at most one job in $\mathcal{U}$ can precede

another job in $\mathcal{U}$ on machine m, ensuring an acyclic order of jobs on the machine. To solve the model described above, we can develop dedicated heuristic approaches aimed at achieving high-quality solutions specifically for the problem. However, as discussed below, this model is equivalent to the well-studied Asymmetric Traveling Salesman Problem (ATSP), for which many effective solution methods have already been developed. By utilizing these established ATSP methods, we can potentially tackle the new problem more efficiently.

Let $d_{jk} = H - P^m_{jk}$, where $H = |Pop|$ is an upper bound for P^m_{jk} ($P^m_{jk} \leq H$, $\forall j \in \mathcal{J}$, $k \in \mathcal{J}$, $m \in \mathcal{M}$). Then we can rewrite the objective (1) as $\max \frac{|\mathcal{J}|(|\mathcal{J}|+1)H}{2} - \sum_{j\in\{0\}\cup\mathcal{J}} \sum_{k\in\mathcal{J}\cup\{0'\}} d_{jk}\eta_{jk}$, which is equivalent to minimizing $\sum_{j\in\{0\}\cup\mathcal{J}} \sum_{k\in\mathcal{J}\cup\{0'\}} d_{jk}\eta_{jk}$. Therefore, the model defined by (1)–(5) is equivalent to the following model:

$$\min \quad \sum_{j\in\{0\}\cup\mathcal{J}} \sum_{k\in\mathcal{J}\cup\{0'\}} d_{jk}\eta_{jk} \tag{6}$$

$$\text{s.t.} \quad (2) - (5)$$

The above model defined by (6) and (2) - (5) is precisely the DFJ model (Öncan et al., 2009) of the ATSP problem, which is defined on a directed graph $G = (\mathcal{V} \cup \{0\} \cup \{0'\}, \mathcal{E})$, where each vertex $i \in \mathcal{V}$ corresponds to a job in $\mathcal{J}$, vertex 0 is the depot and corresponds to the start job 0, while $0'$ is a copy of 0 and corresponds to the finish job. Each arc $\{j, k\} \in \mathcal{E}$ ($j \in \{0\} \cup \mathcal{V}$, $k \in \mathcal{V} \cup \{0'\}$) is assigned a distance $d_{jk} = H - P^m_{jk}$. Then searching a job sequence with the maximum total frequency for machine m corresponds to identifying the shortest route visiting each city (vertex) exactly once in $G = (\mathcal{V} \cup \{0\} \cup \{0'\}, \mathcal{E})$, implying any solution method for the ATSP can be used to solve our job sequence problem defined by constraints (1) - (5). Since the ATSP is a well-studied problem in the literature, for which many highly effective solution methods have been developed (Helsgaun, 2009; Nagata and Kobayashi, 2013; Pop et al., 2024), we can directly use these ATSP methods to solve our job sequence problem and applying existing sophisticated methods can lead to more efficient and reliable solutions. To this end, we adopt the LKH-2 algorithm (Helsgaun, 2009), one of the most powerful ATSP heuristics, whose latest version 2.0.10 is available at http://akira.ruc.dk/~keld/research/LKH/. Note that the schedule generated by the LKH-2 approach can ensure the precedence order between operations on the same machine. However, when considering all job sequences on all $|\mathcal{M}|$ machines, it may violate

the precedence constraints, which require that all operations of a job are executed in the given order. In such cases, we employ the procedure proposed by Ren and Wang (2012) to repair an infeasible schedule to a feasible one.

4.5 Hash-based Two Phased Tabu Search for the JSP

The hash-based two phased tabu search is another critical component of our HTSFP algorithm and plays the key role of search intensification. In what follows, we introduce its neighborhood structure, its tabu strategy using hashing vectors to accurately mark visited solutions and rapidly determine the tabu status of neighboring solutions, as well as its reduced search phase and complete search phase.

4.5.1 Neighborhood Structure

Our tabu search procedure relies on the N8 neighborhood structure recently proposed in Xie et al. (2022). The foundation of the N8 neighborhood structure is based on the concept of the critical path, which represents the longest sequence of dependent operations that determine makespan. Then a critical operation is defined as the operation on the critical path, while a critical block refers to a maximal sequence of adjacent operations on the same machine in a critical path. Based on these defined concepts, the N8 neighborhood structure is defined by the relocation move operator which consists of moving a critical operation outside its critical block and inserting it at a different position within the sequence on the same machine. As shown in Figure 1, compared to the well-known N5, N6, and N7 neighborhood structures, which focus on moving critical operations within a critical block, the N8 neighborhood explores a more extensive search space by considering moving critical operations outside their critical block, thereby increasing the chances of the algorithm discovering high-quality solutions.

4.5.2 Hashing Vectors for Solution Marking

Our hashing strategy is designed to address the challenge of managing a large number of visited solutions during the search phase. Instead of maintaining an impractically massive pool of solutions and comparing each candidate neighboring solution against it, we employ

the hashing technique. By mapping each visited solution to an integer and marking this integer with an index in the associated hashing vector, we efficiently track visited solutions. This approach proves effective in both recording visited solutions and rapidly determining whether a solution has been encountered previously (Wang et al., 2017).

More precisely, with a hash function h and hashing vector H of size L, a candidate neighboring solution $S \in \Omega$ can be mapped to an integer $h(S)$ using the hash function h ($h : S \in \Omega \rightarrow \{0, 1, 2, ..., L-1\}$). This integer serves as an index in the hash vector H, enabling efficient storage and tracking of visited solutions. Specifically, the binary value in the $h(S)$-th position of H (i.e., $H[h(S)]$) indicates the status of solution S: $H[h(S)] = 1$ implies that the candidate solution S has been previously visited; otherwise, $H[h(S)] = 0$. The hash vector H is initialized to 0, indicating no solutions have been visited.

However, an issue arises regarding the computation and update of these hash functions. The challenge is to make their calculation and update as easy and straightforward as possible. Specifically, the structure of the hash functions should align with the structure of the adopted neighborhoods, streamlining the computation of hashing values for candidate neighboring solutions. In our work, we propose hash functions with unified forms that are applicable to all types of JSP problems and are easy to compute, enabling the calculation of hash functions for any neighboring solution in $O(1)$ time complexity. Precisely, for each triple (j, k, m) where $j \in \mathcal{J} \cup \{0\}$, $k \in \mathcal{J} \cup \{0'\}$ and $m \in \mathcal{M}$, we associate it with a predefined dummy value w_{jkm}. For a JSP solution denoted by its job sequences on all machines, let y_{jkm} represent a binary variable where $y_{jkm} = 1$ if jobs j and k are two immediate successor and predecessor jobs in the job sequence of machine m in S, and $y_{jkm} = 0$ otherwise. The hash value $h(S)$ is computed as follows:

$$h(S) = \Big(\sum_{j \in \mathcal{J} \cup \{0\}} \sum_{k \in \mathcal{J} \cup \{0'\}} \sum_{m \in \mathcal{M}} w_{jkm} y_{jkm} \Big) \mod L \tag{7}$$

where L is the length of the hashing vectors which is empirically set to 10^8 in this work, and the weight w_{jkm} for (j, k, m) takes a prefixed value randomly generated in $[0, ..., T]$, where T is a parameter. In the above hash function, due to the size limitation of the hash vector, we modulate the raw hash value $\sum_{j \in \mathcal{J} \cup \{0\}} \sum_{k \in \mathcal{J} \cup \{0'\}} \sum_{m \in \mathcal{M}} w_{jkm} y_{jkm}$ by the hash vector size L to ensure it remains within the vector's boundaries. Figure 5 illustrates the application of this hash function to mark a solution on the presented instance with three machines and

three jobs.

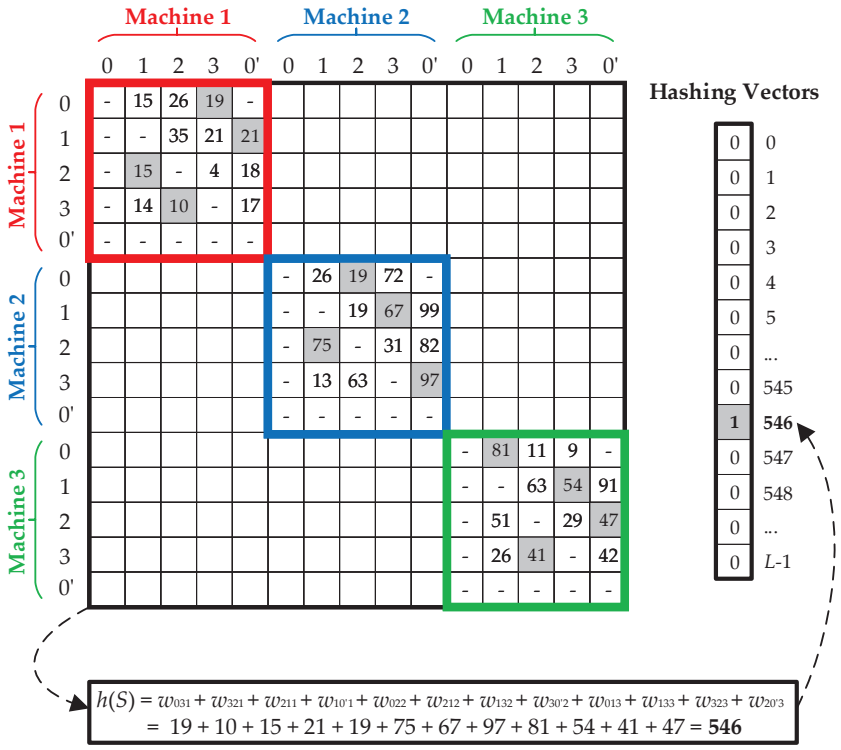


Figure 5: The Use of a Hash function to Mark Solution $\{(3,2,1),(2,1,3),(1,3,2)\}$ on an Instance with Three Machines and Three Jobs.

In such an implementation, a fast incremental evaluation technique can be used to compute the hash function value of any neighboring solution S' based on the hash function value of the current solution S. Specifically, our tabu search procedure relies solely on the N8 neighborhood, employing a relocation move that shifts a critical operation of job i on machine m outside its critical block and re-inserts it elsewhere in the job sequence on machine m. Assume that j and k are the immediate predecessor and successor jobs of i respectively before the move, and i is re-inserted between another two immediate predecessor and successor jobs j' and k' in the same sequence, the hash value of the resulting solution S' can then be easily calculated as:

$$h(S') = (h(S) + w_{jkm} + w_{j'im} + w_{ik'm} - w_{jim} - w_{ikm} - w_{j'k'm}) \mod L \tag{8}$$

However, due to the "birthday paradox" where the probability of a shared birthday exceeds 50% in a group of only 23 people (Carlton and Barnes, 1996), collisions can frequently

occur when two different solutions S_1 and S_2 are mapped into the same integer by h, i.e., $h(S_1) = h(S_2)$. Such collisions may lead to an incorrect determination of a neighboring solution's tabu status, causing a non-visited candidate solution to be mistakenly identified as a previously visited one.

To effectively reduce hashing collisions, we adopt the technique from Wang et al. (2017) by simultaneously using multiple hash functions and their associated vectors to track each visited solution. Specifically, three hash functions and their associated vectors are employed to record each visited solution in this study. By setting different values for w_{jkm}, we obtain three distinct hash functions, namely $h_1(S)$, $h_2(S)$, and $h_3(S)$. The rationale behind using multiple hash functions (vectors) to reduce collisions is twofold. First, each hash function generates a different hash value for the same input, so even if one function produces a collision, the others may not. Therefore, as long as collisions do not simultaneously occur in all three hashing vectors (i.e., $H_1[h_1(S)] = 0$ or $H_2[h_2(S)] = 0$ or $H_3[h_3(S)] = 0$), we clearly confirm such a candidate solution S has never been visited previously, and there is no risk of misclassifying the tabu status for this candidate solution. Second, the probability of collisions occurring simultaneously in all three hash vectors (i.e., $H_1[h_1(S)] = 1$ and $H_2[h_2(S)] = 1$ and $H_3[h_3(S)] = 1$) is significantly lower compared to using only a single hashing vector. As shown in Section A.3, our three hash functions can achieve an extremely low collision rate, approaching zero.

The method for recording visited solutions and determining the tabu status of a candidate solution operates as follows. At the beginning of the search, all hashing vectors are initialized with 0, indicating that no solution has been visited. Subsequently, each time a solution is visited during the search process, the corresponding positions in the hashing vectors are marked with 1 by setting $H_k[h_k(S)] = 1$ for $k = 1, 2$, and 3. To determine the tabu status of a candidate neighboring solution S', we first compute the hashing values $h_k(S')$ and then check the values of $H_k[h_k(S')]$. If any of them is 0, the candidate solution S' is classified as an admissible solution that has not been visited previously. Otherwise i.e.,(i.e., $H_k[h_k(S')] = 1$ for $k = 1, 2$ and 3), the candidate solution is identified as a previously visited solution, and it is prohibited from being considered again.

4.5.3 The Reduced Search Phase

Compared to well-known neighborhood structures like N5 (Nowicki and Smutnicki, 1996), N6 (Balas and Vazacopoulos, 1998), and N7 (Zhang et al., 2007), N8 (Xie et al., 2023) explores a broader solution space. While a large neighborhood increases the chances of discovering better solutions, it also demands evaluating a larger number of neighbors. Especially for the JSP, the calculation for the makespan of a neighboring solution requires a time complexity of $O(|\mathcal{M}||\mathcal{J}|)$, further intensifying the computational burden on neighborhood evaluation. However, not all neighbors are equally likely to improve solutions. Therefore, we propose a reduced neighborhood search, allowing the algorithm to focus on promising regions of the search space. To define our search space, we use the frequent adjacent order patterns and frequent precedence order patterns identified in Section 4.3 to temporarily constrain a subset of job orders through the following two rules.

- **Adjacent Job Order Fixing Rule:** If jobs j and k are immediate successor and predecessor in the job sequence of machine m across all elite solutions in the population and the constructed solution produced by the frequent pattern based method (i.e., $P_{jk}^{m} = |Pop|$), then during the subsequent local search procedure, the relative positions of jobs j and k on machine m will be restricted from changing to strictly maintain this adjacent order between the two jobs. For example, in Figure 4, jobs 2 and 3 are immediate successor and predecessor in the job sequence of machine 1 across all solutions. Therefore, during the subsequent local search, the relative positions of jobs 2 and 3 in the job sequence of machine 1 will be kept unchanged, but the absolute positions of jobs 2 and 3 in machine 1 can change. The time complexity of checking whether a move respects this rule is bounded by $O(1)$.

- **Precedence Order Fixing Rule:** If job j always precedes (not necessarily immediately precedes) job k in the job sequence of machine m across all solutions of the population (i.e., $Q_{jk}^{m} = |Pop|$), then during the subsequent local search procedure, we do not consider moves that place job i after job j in the job sequence of machine m. For example, in Figure 4, job 2 is always processed before jobs 3 and 5 on machine 1. Therefore, during the subsequent local search, we do not consider moves that place job 2 after job 3 or 5 in the job sequence of machine 1. The time complexity of

checking whether a move respects this rule is bounded by $O(|\mathcal{J}|)$.

At each iteration of the reduced search, only moves respecting both fixing rules are used to generate candidate neighboring solutions. From these solutions, the reduced search then selects the best non-visited neighboring solution (i.e., the one not marked by the hash-based tabu strategy and with the minimum makespan) to replace the current solution. The reduced search stops when the best-found solution fails to be improved for a number of ω_1 consecutive iterations, where ω_1 is a parameter called search depth. By concentrating on the reduced search space, the neighborhood becomes smaller and more focused, potentially leading to more efficient searches. However, there is a risk that some job orders may be incorrectly fixed. To mitigate this risk, the tabu search further resorts to a complete search phase, where an exhaustive search is performed in the N8 neighborhood.

4.5.4 The Complete Search Phase

When the reduced search phase stops, the tabu search switches to the complete search phase to allow the search to explores the full neighborhood of N8 to further improve the solution. Specifically, at each iteration of the complete search, it considers all candidate neighboring solutions in N8, and selects the overall best non-visited neighboring solution to replace the current solution. The complete search stops when the best-found solution can not be improved for a number of ω_2 consecutive iterations. The general scheme of our hash-based two phased tabu search is presented in Algorithm 1 in Appendix A.2.

4.6 Population Health Management

This work employs a population management strategy taking both solution quality and distance information into account to maintain a healthy population (Lü and Hao, 2010; Cheng et al., 2016; Zhou et al., 2024).

Our evaluation of the distance between two solutions is based on the similarity of solutions. Currently, two main approaches exist in the literature for evaluating the similarity of JSP solutions. The first approach employs the concept of the "Hamming distance", where similarity increases by one if the job operations at the corresponding position on the same machine are identical. The second approach uses the method of finding the longest

common subsequence to determine the similarity between two solutions, which is generally considered more accurate than the first method. In this work, we propose a job order-based method for assessing the similarity between two solutions S_a and S_b. Precisely, for a pair of jobs (j, k) and a machine $m \in \mathcal{M}$, if j immediately precedes k in the job sequence of machine m in both solutions S_a and S_b, the similarity between the two solutions increases by 1. Then, the job order-based similarity $Sim(S_a, S_b)$ is defined as follows:

$$Sim(S_a, S_b) = \sum_{j \in \mathcal{J}} \sum_{k \in \mathcal{J}} \sum_{m \in \mathcal{M}} \psi(j, k, m) \tag{9}$$

where $\psi(j, k, m)$ equals 1 if job j is the immediate predecessor of job j in the job sequence of m in both solutions, and 0 otherwise.

Job Order-based Distance between Two Schedulings: Given two individuals S_a and S_b, let $Sim(S_a, S_b)$ be their job order-based similarity, then the job order-based distance between these two solutions is defined by:

$$D_{a,b} = |\mathcal{M}||\mathcal{J}| - Sim(S_a, S_b) \tag{10}$$

Distance between One Scheduling and a Population: Given a population $Pop = \{S_1, \ldots, S_p\}$ and the distance D_{ab} between any two schedulings S_a and S_b $(a, b = 1, \ldots, p, a \neq b)$, the distance between a scheduling S_a $(a = 1, \ldots, p)$ and the population Pop is defined as the minimum distance between S_a and any other scheduling in Pop, denoted by $D_{a,Pop}$:

$$D_{a,Pop} = \min\{D_{ab} \mid S_b \in Pop, b \neq a\} \tag{11}$$

Health Score of a Scheduling for a Population: Given a population $Pop = \{S_1, \ldots, S_p\}$ and the distance $D_{a,Pop}$ for any solution S_a $(a = 1, \ldots, p)$, the health score of solution S_a for population Pop is defined as:

$$Q(a, Pop) = \beta \cdot \tilde{A}(f(S_a)) + (1 - \beta) \cdot \tilde{A}(D_{a,Pop}) \tag{12}$$

where $f(S_a)$ is the makespan of solution S_a and $\tilde{A}(\cdot)$ represents the normalized function:

$$\tilde{A}(t) = \frac{t - t_{\min}}{t_{\max} - t_{\min} + 1} \tag{13}$$

where $t_{\max}$ and $t_{\min}$ are respectively the maximum and minimum values of t in the population Pop, the number "1" is used to avoid the possibility of a 0 denominator, and β is a constant parameter.

Our population management strategy to determine whether to insert a solution S into Pop or discard it operates as follows. We temporarily add the solution S to the population Pop, resulting in a modified population Pop', i.e., $Pop' \leftarrow Pop \cup S$. We evaluate each individual in the population Pop' using the scoring function (12) and identify the worst solution S_c, i.e., $S_c \leftarrow \arg\max_{S_a \in Pop'} Q(a, Pop')$. We compare S_c with S. If S_c is different from S, we replace S_c with S. Otherwise, we discard S.

5 Computational Results

In this section, we present computational results of our proposed HTSFP algorithm and provide a comparison with the state-of-the-art JSP heuristics.

5.1 Benchmark Instances and Experimental Settings

To evaluate the performance of HTSFP, we only focus on these notoriously difficult instances, which are considered to be challenging for most JSP algorithms, while excluding from consideration those instances for which the optimal solution can be readily reached in a very short time by most algorithms in the literature. Consequently, we test our HTSFP algorithm on the following four sets of a total of 185 JSP benchmark instances.

- The first set of benchmark instances features the 50 most challenging cases TA01–TA50 by Taillard (1994).
- The second set of benchmark instances comprises the 40 classic instances LA01–LA40 introduced by Lawrence (1984).
- The third set of benchmark instances includes the 15 instances SWV01–SWV15 presented by Storer et al. (1992).
- The fourth set of benchmark instances encompasses the 80 most demanding scenarios DMU01-DMU80 compiled by Demirkol et al. (1998).

These instances have been extensively tested by various algorithms for JSP. All these benchmark instances can be accessed for testing from the OR-Library and Shylo's webpage (see https://optimizizer.com/jobshop.php).

We implemented HTSFP in C++ and executed it on a personal computer equipped with an i7-12700 processor clocked at 2.1GHz and 16GB of RAM, operating on the Windows 11 platform. HTSFP requires four parameters as summarized in Table 4. To tune these parameters, we resort to IRACE (Birattari et al., 2002), an automatic parameter configuration tool. IRACE was executed using 20 randomly selected instances, with a tuning budget of 2000 HTSFP executions and a time limit of 300s per execution. Table 4 presents the parameter settings recommended by IRACE. These settings are considered as the default setting of the algorithm and were used for all experiments in this study.

Table 4: Settings of the Required Parameters of the HTSFP Algorithm.

Parameter	Section	Description	Considered values	Final
ω_1	4.5.4	Search depth of the reduced search	{1e3, 2e3, 3e3, 4e3, 5e3}	2e3
ω_2	4.5.4	Search depth of the complete search	{1e3, 2e3, 3e3, 4e3, 5e3}	2e3
β	4.6	Population quality parameter	{0.1, 0.3, 0.5, 0.7, 0.9}	0.5
p	4.5.2	Number of individuals in population	$\{20, 30, 40, 50, 60\}$	30
T	4.6	Range of weights used in the hash functions	$\{300, 600, 1000, 10000, 100000\}$	1000

Considering the stochastic nature of HTSFP, we independently execute the HTSFP algorithm ten times for each problem instance. Additionally, due to the variation in the sizes of these 185 instances, different stopping conditions have been applied in the literature on these instances. Following Xie et al. (2022), we employ different time limits of Table 5.

Table 5: The Time Limit for Each Instance Set.

Instance name	SWV12 (15)	DMU56-65	DMU66-70	DMU71-80	Other
Time limit (h)	2	2	4	5	1

To measure the performance of HTSFP, we calculate the relative error (RE) using the following relative deviation formula on each instance, following Xie et al. (2022).

$$RE = 100 \times \frac{UB_{\text{solve}} - LB_{\text{best}}}{LB_{\text{best}}}$$

where LB_{best} and UB_{solve} are the best known lower bound and upper bound reported in the literature, respectively. Subsequently, we calculate the mean relative error (MRE) for a given algorithm as the mean RE over all the tested instances.

To demonstrate the effectiveness of HTSFP, we conduct a comparison with several state-of-the-art algorithms from the literature. According to the computational results presented in recent studies on JSP, the following three algorithms can reach the best-known upper bounds reported in the literature for almost all of the instances and can be considered as the current state-of-the-art algorithms for JSP.

- BRKGA by Gonçalves and Resende (2014), was implemented in C++ and tested on a computer equipped with an AMD 2.2 GHz Opteron 2427 CPU running the Linux (Fedora release 12) operating system.
- TS/PR by Peng et al. (2015), was implemented in C++ and executed on a PC with a Quad-Core AMD Athlon 3.0 GHz CPU and 2 GB RAM, running the Windows 7 operating system.
- HA by Xie et al. (2022), was implemented in Java and executed on an Intel Core PC with a 3.6 GHz processor and 16 GB of memory.

Note that the reference algorithms were tested on different computing platforms. According to the product information provided by Intel (https://www.intel.com/) and Advanced Micro Devices (https://www.amd.com/), our computing platform has a slower single-thread CPU speed compared to that of Peng et al. (2015) and Xie et al. (2022), and is roughly comparable to that of Gonçalves and Resende (2014). Providing a fully fair comparison between HTSFP and the reference algorithms is challenging due to differences in coding languages, implementations, computing platforms, etc. Therefore, we primarily focus on the quality criterion of the obtained solutions and provide computation times just for indicative purposes. Finally, we have made the validated best-known solutions and all instances from this study publicly accessible on Shylo's webpage at https://optimizizer.com/jobshop.php.

5.2 Computational Comparisons

We present a comparative analysis of the HTSFP algorithm against reference algorithms across the four sets of benchmark instances in Table 6. Detailed computational results for each instance can be found in Appendix (Tables 11 – 15).

Each row in Table 6 reports the results on a subset of instances of the same type from the same data set. The first three columns in the table indicate which data set each subset

of instances belongs to, the compared algorithm pair, and the number of compared instances in this subset when comparing each pair of algorithms. Columns "#Win", "#Tie" and "#Lose" summarize respectively the number of instances for which HTSFP obtained a better (#Win), equal (#Tie) or worse (#Lose) result compared to each reference algorithm. Column "p-value" provides the p-values from the Wilcoxon signed-rank test with a significance level of 0.05 for the results of each compared algorithm pair.

Table 6: Computational comparison results on the JSP instances.

			Best				Avg.			
Instance	Algorithm pair	#Case	#Win	#Tie	#Lose	p-value	#Win	#Tie	#Lose	p-value
TA	HTSFP vs. HA	50	6	44	0	2.6e-02	29	19	2	2.3e-03
	HTSFP vs. TS/PR	50	13	37	0	1.4e-03	32	17	1	5.9e-07
	HTSFP vs. BRKGA	50	9	41	0	7.1e-03	38	11	1	7.7e-08
LA	HTSFP vs. HA	40	0	40	0	-	0	40	0	-
	HTSFP vs. TS/PR	40	0	40	0	-	0	40	0	-
	HTSFP vs. BRKGA	40	0	40	0	-	2	38	0	1.8e-01
SWV	HTSFP vs. HA	15	1	14	0	9.3e-04	7	5	3	1.7e-05
	HTSFP vs. TS/PR	15	4	11	0	1.1e-08	12	2	1	5.2e-10
	HTSFP vs. BRKGA	15	7	8	0	7.6e-09	14	1	0	1.6e-08
DMU	HTSFP vs. HA	80	14	66	0	3.2e-01	37	31	12	2.8e-02
	HTSFP vs. TS/PR	80	43	37	0	6.6e-02	57	20	3	2.4e-03
	HTSFP vs. BRKGA	80	44	36	0	1.8e-02	54	20	6	9.8e-04

Based on the summarized results in Table 6 and the detailed results in Tables 11-15, the following observations can be made.

- TA instance set. As shown in the row "TA" of Table 6, the results achieved by our HTSFP algorithm are highly competitive when compared to the three reference algorithms. Specifically, in terms of the best objective values, HTSFP can achieve better results than HA, TS/PR, BRKGA respectively for 6, 13, 9 out of the 50 instances, and matches the remaining instances. In terms of the average objective values, HTSFP can achieve better results than HA, TS/PR, BRKGA respectively for 29, 32, 38 instances, and worse results only on 2, 1, 1 cases. By examining Table 11 of the online supplemental appendix, one can observe that HTSFP improves the previously best-known result (new upper bound) for one instance (TA49).

- LA instance set. From the row "LA" of Table 6, one can observe that the results obtained by HTSFP are comparable to those of HA, TS/PR and BRKGA. Indeed,

across all 40 TA instances, HTSFP consistently achieves equal or better results in terms of both the best and average objective values. From Table 12 of the online supplemental appendix, we observe that our HTSFP algorithm can consistently attain the known optimal solution for all except instance LA29 in each run. From Table 6, one can also observe that all compared methods perform similarly on the set of LA instances as all of them can reach the same best result on the 40 LA instances. This can be attributed to the relatively small size of the LA instances and the fact that the processing times of the operations in these instances vary within a narrow range, making the LA instances easier for these algorithms to converge to the best solutions.

- SWV instance set. As shown in the row "SWV" of Table 6, compared to the three reference algorithms HA, TS/PR and BRKGA, HTSFP has an excellent performance on the 15 SWV instances. Furthermore, as shown by the row "MRE" in Table 13 of the online supplemental appendix, our HTSFP algorithm has a much smaller MRE value compared to the other three algorithms, indicating that the makespan of the solutions produced by HTSFP is closer to the lower bound for these 15 SWV instances. This confirms the competitiveness of our HTSFP algorithm on the SWV instance set.

- DMU instance set. As shown in the row "DMU" of Table 6, compared to the HA, TS/PR, BRKGA heuristic algorithms, HTSFP shows a significant performance advantage in terms of the best objective values, by achieving better results for 14, 43, 44 out of 80 instances and matches the remaining instances. Furthermore, by examining Table 14 of the online supplemental appendix, we observe that HTSFP improves the previously best-known result (new upper bound) for 13 instances. HTSFP is able to reach its best objective value in a relatively short time compared to the time required by HA and TS/PR.

In summary, across the four sets comprising a total of 185 extensively tested JSP benchmark instances, the HTSFP algorithm performed remarkably well in terms of both the best and average results. Notably, it improves the previous best-known results (establishing new upper bounds) for 13 instances. A summary of the updated best results achieved by HTSFP is provided in Table 9.

5.3 Comparisons between HTSFP and Exact Methods based on Mathematical Models

To further confirm the effectiveness of the proposed HTSFP, we compare it with exact methods based on two well-known Mixed Integer Programming (MIP) models for the JSP, namely, the disjunctive model and the time-indexed model (Ku and Beck, 2016). Both models were solved using CPLEX 22.1.0 on the same computer that was used to test our HTSFP approach, ensuring a fair comparison. CPLEX 22.1.0 stops when the given instance is solved to optimality or when the time limit of 7200 s is reached. Table 7 in Appendix summarizes the comparative results of the three methods. For the two compared models, we report the achieved upper bound (UB), lower bound (LB), the percentage gap between the LB and UB (GAP), and the running time in seconds required to reach the optimal solution for each instance. If an instance cannot be solved to optimality within the time limit, the reported value for Time(s) is 7200 seconds. It can be observed from Table 7 that, for the cases where the disjunctive model or time-indexed model achieves optimality, HTSFP easily and consistently achieves the optimal solutions. Comparing HTSFP with the time-indexed model, it is observed that the time-indexed model can only report UB and LB for the set of LA instances within the given time limit. On the other hand, HTSFP outperforms the time-indexed model on almost all LA instances by producing smaller upper bounds. In comparison to the disjunctive model, it is observed that the disjunctive model can only optimally solve cases up to a size of 15×15; larger instances cannot be optimally solved within the given time, and the solution quality is significantly inferior to that of HTSFP. For the LA instances, the disjunctive model can optimally solve cases up to a maximum size of 10×10, while neither 15×5 nor larger instances can be optimally solved, yielding results that are far inferior to HTSFP. For the SWV and DMU instances, the disjunctive model cannot optimally solve even the smallest cases.

5.4 Computational results on larger instances

As mentioned in Section 5.1, our experiments conducted in Sections 5.2 and 5.3 focus on these notoriously difficult and large instances from the literature. To further evaluate the scalability of the proposed HTSFP algorithm, we generate 10 new larger instances: five with

80 jobs and 30 machines, and five with 90 jobs and 35 machines. These 10 instances, named DMU81–DMU90, are generated using the method for generating DMU instances (Demirkol et al., 1998), which are considered the most challenging JSP instances in the literature. The processing sequence of each job is a random permutation of all the machines in the shop. The operation processing times of the operations are generated from a discrete uniform distribution between 1 and 200. These new large instances are available for download at https://github.com/MINGJIE666/JSP.

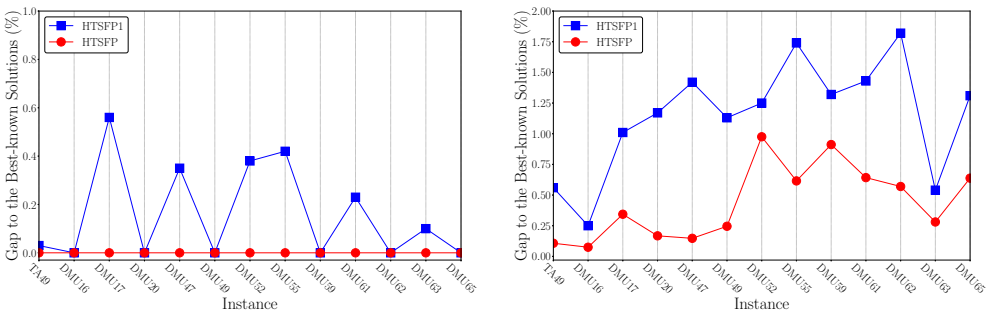
We tested HTSFP on the 10 newly generated instances and compared it with the disjunctive model and the time-indexed model. Table 8 in the Appendix summarizes the comparative results of the three methods. As shown in Table 8, the disjunctive model is only able to obtain upper bounds (UBs) for two instances within the 7200 s time limit, while the time-indexed model fails to reach any UBs for all instances within the same time frame. Furthermore, HTSFP provides significantly better UBs (derived from its feasible solutions) within the given time compared to the two MIP models for the tested instances. Overall, our HTSFP employs hash technology to quickly mark solutions and effectively reduce invalid visits, and also applies data mining to identify meaningful orders that guide the algorithm toward more promising solutions. These techniques enable HTSFP to significantly outperform state-of-the-art models proposed in the literature, both in terms of runtime and solution quality when solving large-scale problems.

6 Analysis

In this section, we analyze the key components of the proposed HTSFP algorithm to verify their impacts on the performance of the algorithm: the usefulness of the hashing vectors for solution marking and the rationale behind the frequent pattern mining. The effectiveness of the multiple hash function strategy on reducing hashing collisions, the importance of the frequent pattern based method for solution construction, and the sensitivity of the parameters required by HTSFP are analyzed in Appendix A.3-A.5 respectively.

6.1 The Usefulness of the Hashing Vectors for Solution Marking

To assess the benefit of the hashing vectors for solution marking, we compared HTSFP with an HTSFP variant (named HTSFP1), where we replaced the hashing vector solution marking with a traditional move based tabu strategy. For HTSFP1, each time a move is performed, the reversed move is no longer eligible for selection for the next tt iterations, where tt is the tabu tenure. We tested three different values of tt (tt = 5, 10, and 15 respectively) to fully evaluate the performance of HTSFP1. Preliminary experimental results indicate that HTSFP1 with $tt = 10$ has better overall performance, therefore, we set $tt = 10$ for our HTSFP1. In addition to the traditional move based tabu strategy, the classic aspiration criterion is also applied to further reinforce HTSFP1, which permits a move to be performed despite being tabu if that move produces a solution better than the best solution ever found. For this comparison, we used the 13 difficult instances whose upper bounds are updated in this paper as shown in Table 9. HTSFP and HTSFP1 are run 10 times independently to solve each instance with a time limit described in Section 5. The comparative results are shown in Figure 6.



(a) Best results

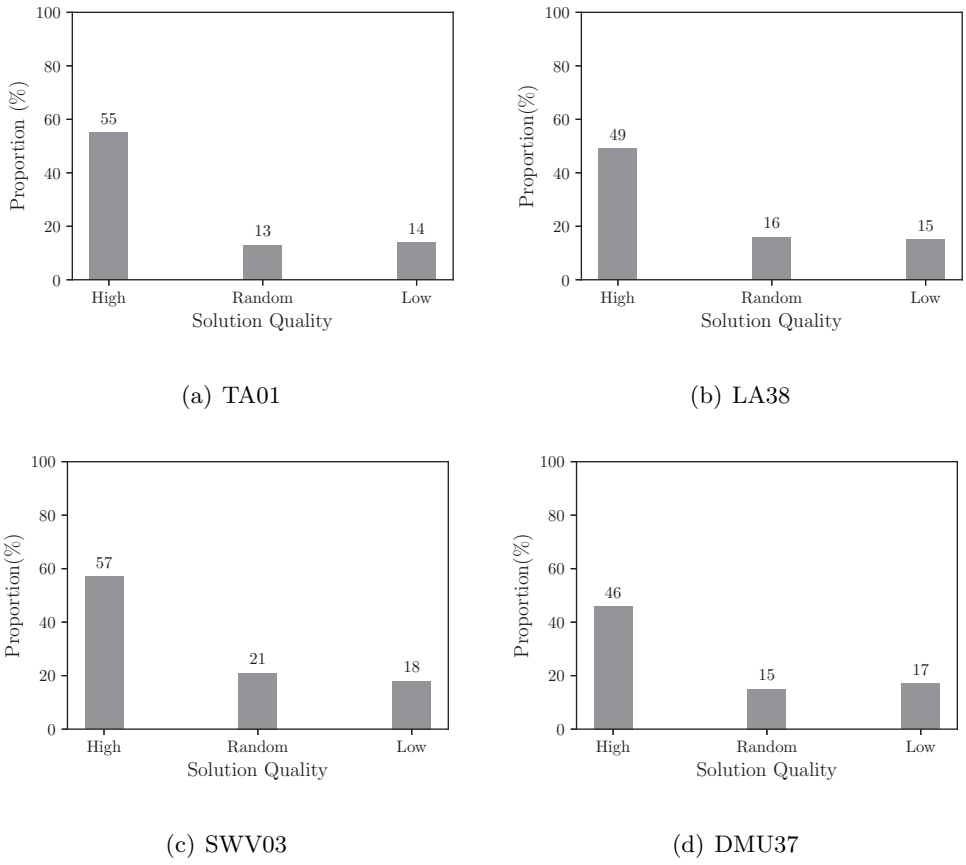
(b) Average results

Figure 6: Comparisons of the HTSFP with Its Variant HTSFP1.

From Figure 6, it can be observed that HTSFP and its variant HTSFP1 respectively achieve the best results in 13 and 5 instances in terms of the best objective value, and in 13 and 0 instances in terms of the average objective value. Furthermore, the small p-values (1.80E-02 and 2.44E-04) from the Wilcoxon signed-rank test confirm the statistically significant differences between HTSFP and its variant HTSFP1 in terms of the best and average objective.

6.2 Rationale behind the Frequent Pattern Mining for Solution Construction

As mentioned in Section 4.4, the design of our frequent pattern based method relies on the observation that some common adjacent job orders frequently appear in the job sequence of many high-quality solutions, which form the backbone of optimal solutions. In this section, we provide empirical motivation for this solution construction method. For this purpose, we show an analysis on the proportion of commonly shared adjacent job orders between solutions of various quality.



(a) TA01

(b) LA38

(c) SWV03

(d) DMU37

Figure 7: Proportion of Commonly Shared Adjacent Job Orders between Solutions of Various Quality.

For the experiment, we used four representative instances with the known optimal value: TA01, LA38, SWV03, and DMU37. We applied our HTSFP with different stop conditions to solve these instances and recorded a number of solutions. To collect more solutions of

different quality, we also applied a HTSFP variant that replaces the frequent pattern mining part with a multi-restart component. Therefore, for each instance, we collect 600 solutions of different quality using HTSFP and its multi-restart variant. Among these 600 solutions, we select the top 5% (30 cases) with the smallest objective values and call them 'high-quality solutions'. Similarly, we take the bottom 5% (30 cases) with the largest objective values and call them 'low-quality solutions'. We also randomly select 5% from the 600 solutions and call them 'random solutions'. For each instance, we identify its optimal solution and put it directly into the set of high-quality solutions.

Then, for each solution set, we calculated the proportion of common adjacent job orders among all pairwise adjacent job orders, computed as $\frac{n_s}{|\mathcal{M}|(|\mathcal{J}|+1)} \times 100\%$, where n_s is the number of common adjacent job orders shared by all solutions in the set. The results are presented in Figure 7. From Figure 7, we can clearly see that, the proportion of common adjacent job orders between solutions in the high-quality set is significantly higher than that between solutions in the low-quality set and the random set.

L_1:	0	12	10	8	11	9	13	4	3	2	15	7	6	5	1	15	0'
L_2:	0	12	8	10	11	9	13	4	3	2	15	7	6	5	1	15	0'
L_3:	0	12	8	10	11	9	13	4	3	2	15	7	6	5	1	15	0'
L_4:	0	12	10	8	11	9	13	4	3	2	15	7	6	5	1	15	0'
L_5:	0	12	10	8	11	9	13	4	3	2	15	7	6	5	1	15	0'
L_6:	0	12	10	8	11	9	13	4	3	2	15	7	6	5	1	15	0'
L_7:	0	12	10	8	11	9	13	4	3	2	15	7	6	5	1	15	0'
L_8:	0	12	10	8	11	9	13	4	3	2	15	7	6	5	1	15	0'
L_9:	0	12	8	10	11	9	13	4	3	2	15	7	6	5	1	15	0'
L_{10}:	0	12	8	10	11	9	13	4	3	2	15	7	5	6	1	15	0'

Figure 8: Schedules among Ten High-Quality Solutions on Machine 1 of Instance TA01.

To complement this experiment, we also use a small instance, TA01, as a case study. From the 600 collected solutions, we record the ten highest quality solutions including the optimal solution, denoted as $L1 - L10$, with $L1$ being the optimal solution. Figure 8 illustrates the job processing order on machine 1 for these ten high-quality solutions (Note that two solutions in $L1 - L10$ may have the same job sequence on machine 1, but their job sequences may be different on other machines, resulting in two different solutions). It is evident that in these high-quality solutions, the majority of the adjacent job orders are identical and very close to those of the optimal solution. This observation further validates the rationale for using frequent pattern mining for solution construction.

7 Conclusion

This study investigated the Job Shop Scheduling Problem (JSP), a classic and important problem in the field of operations research and manufacturing. To effectively solve JSP, this study proposed an original hybrid algorithm (denoted by HTSFP) that combines hashing vectors and frequent pattern mining for JSP. The proposed HTSFP algorithm incorporates several innovative components. Firstly, since their introduction, tabu search algorithms have consistently represented the state-of-the-art in obtaining high-quality solutions to JSP. However, the traditional tabu strategies often encounter difficulties such as giving a tabu status to unvisited solutions or falling into long-term cycles. To overcome these challenges, our algorithm introduces a novel hash technique that employs hash functions to precisely mark visited solutions and quickly determine the tabu status of neighboring solutions. Additionally, the hash functions are implemented with a fast streamline evaluation, ensuring that their computation and update can be easily achieved with a complexity of $O(1)$. Secondly, while frequent pattern mining approaches have proven to be effective in solving various combinatorial optimization problems, they have rarely been investigated for the JSP. Our hybrid algorithm tages advantage of the potential of frequent pattern mining by incorporating a frequent pattern recognition method to identify promising solution structures, which are used not only to generate new promising solutions, but also to reduce the search space by focusing on areas of the solution space that are more likely to contain promising solutions.

The performances of the proposed algorithm was tested on four sets of benchmark instances in the literature. Compared with the state-of-the-art heuristic algorithms, the proposed algorithm proved to be highly competitive. In particular, it obtained improved best-known results (i.e., new UBs) for 13 benchmark instances. Additional experiments were performed to study the impact of the essential components of the proposed HTSFP, confirming the usefulness of the hashing vector solution marking strategy, the joint use of multiple hash functions, and the frequent pattern mining strategy. Since these strategies have a general nature and significantly contribute to the performance of the proposed algorithm, it would be interesting for future research to explore the application of the HTSFP algorithm to address other JSPs with different objective functions and practical operational rules.

Acknowledgments

We are grateful to the editors and reviewers for their valuable comments, which helped us to significantly improve this paper. This work was partially supported by the National Natural Science Foundation Program of China (Grant No. 72471100, 72131008), the Interdisciplinary Research Program of HUST (Grant No. 5003300129), the Postdoctoral Fellowship Program of China Postdoctoral Science Foundation (Grant No. GZC20242029), and the Hubei Provincial Natural Science Foundation of China (Grand No. 2025AFC038).

References

Akers Jr, S. B. (1956). A graphical approach to production scheduling problems. *Operations Research 4*(2), 244–245.

Balas, E. and A. Vazacopoulos (1998). Guided local search with shifting bottleneck for job shop scheduling. *Management Science 44*(2), 262–275.

Barker, J. R. and G. B. McMahon (1985). Scheduling the general job-shop. *Management Science 31*(5), 594–598.

Beck, J. C., T. Feng, and J.-P. Watson (2011). Combining constraint programming and local search for job-shop scheduling. *INFORMS Journal on Computing 23*(1), 1–14.

Birattari, M., T. Stützle, L. Paquete, and K. Varrentrapp (2002). A racing algorithm for configuring metaheuristics. In *GECCO 2002: Proceedings of the Genetic and Evolutionary Computation Conference, New York, USA, 9-13 July 2002*, pp. 11–18.

Błażewicz, J. (2019). *Handbook on scheduling: From theory to practice*. Springer.

Carlier, J. and É. Pinson (1989). An algorithm for solving the job-shop problem. *Management Science 35*(2), 164–176.

Carlton, W. B. and J. W. Barnes (1996). A note on hashing functions and tabu search algorithms. *European Journal of Operational Research 95*(1), 237–239.

Chen, H. and P. B. Luh (2003). An alternative framework to lagrangian relaxation approach for job shop scheduling. *European Journal of Operational Research 149*(3), 499–512.

Cheng, T. C. E., B. Peng, and Z. Lü (2016). A hybrid evolutionary algorithm to solve the job shop scheduling problem. *Annals of Operations Research 242*(2), 223–237.

Demirkol, E., S. Mehta, and R. Uzsoy (1998). Benchmarks for shop scheduling problems. *European Journal of Operational Research 109*(1), 137–141.

Etesami, S. R. (2022). An optimal control framework for online job scheduling with general cost functions. *Operations Research 70*(5), 2674–2701.

Falkenauer, E. and S. Bouffouix (1991). A genetic algorithm for job shop. In *Proceedings of the 1991 IEEE International Conference on Robotics and Automation, Sacramento, CA, USA, 9-11 April 1991*, pp. 824–829.

Garey, M. R., D. S. Johnson, and R. Sethi (1976). The complexity of flowshop and jobshop scheduling. *Mathematics of Operations Research 1*(2), 117–129.

Gonçalves, J. F., J. J. de Magalhães Mendes, and M. G. Resende (2005). A hybrid genetic algorithm for the job shop scheduling problem. *European Journal of Operational Research 167*(1), 77–95.

Gonçalves, J. F. and M. G. Resende (2014). An extended akers graphical method with a biased random-key genetic algorithm for job-shop scheduling. *International Transactions in Operational Research 21*(2), 215–246.

González, M. A., C. R. Vela, and R. Varela (2015). Scatter search with path relinking for the flexible job shop scheduling problem. *European Journal of Operational Research 245*(1), 35–45.

Grimes, D. and E. Hebrard (2015). Solving variants of the job shop scheduling problem through conflict-directed search. *INFORMS Journal on Computing 27*(2), 268–284.

Helsgaun, K. (2009). General k-opt submoves for the lin–kernighan tsp heuristic. *Mathematical Programming Computation 1*, 119–163.

Jain, A. S. and S. Meeran (1999). Deterministic job-shop scheduling: Past, present and future. *European Journal of Operational Research 113*(2), 390–434.

Kaskavelis, C. A. and M. C. Caramanis (1998). Efficient lagrangian relaxation algorithms for industry size job-shop scheduling problems. *IIE Transactions 30*(11), 1085–1097.

Kowalczyk, D. and R. Leus (2018). A branch-and-price algorithm for parallel machine scheduling using zdds and generic branching. *INFORMS Journal on Computing 30*(4), 768–782.

Ku, W.-Y. and J. C. Beck (2016). Mixed integer programming models for job shop scheduling: A computational analysis. *Computers & Operations Research 73*, 165–173.

Lamorgese, L. and C. Mannino (2019). A noncompact formulation for job-shop scheduling problems in traffic management. *Operations Research 67*(6), 1586–1609.

Lawrence, S. (1984). Resouce constrained project scheduling: An experimental investigation of heuristic scheduling techniques (supplement). *Graduate School of Industrial Administration, Carnegie-Mellon University*.

Liu, A., P. B. Luh, K. Sun, M. A. Bragin, and B. Yan (2023). Integrating machine learning and mathematical optimization for job shop scheduling. *IEEE Transactions on Automation Science and Engineering 21*(3), 4829–4850.

Liu, R., C. Wang, H. Ouyang, and Z. Wu (2024). Exact algorithm and machine learning-based heuristic for the stochastic lot streaming and scheduling problem. *IISE Transactions*, 1–15.

Lu, Y., K. You, Y. Wang, Y. Liu, C. Zhou, Y. Jiang, and Z. Wu (2025). Deep reinforcement learning for automated scheduling of mining earthwork equipment with spatio-temporal safety constraints. *Frontiers of Engineering Management 12*(1), 39–58.

Lü, Z. and J.-K. Hao (2010). A memetic algorithm for graph coloring. *European Journal of Operational Research 203*(1), 241–250.

Michael, P. (2012). *Scheduling. Theory, Algorithms and systems.* Springer.

Minella, G., R. Ruiz, and M. Ciavotta (2008). A review and evaluation of multiobjective algorithms for the flowshop scheduling problem. *INFORMS Journal on Computing 20*(3), 451–471.

Naderi, B., R. Ruiz, and V. Roshanaei (2023). Mixed-integer programming vs. constraint programming for shop scheduling problems: New results and outlook. *INFORMS Journal on Computing 35*(4), 817–843.

Nagata, Y. and S. Kobayashi (2013). A powerful genetic algorithm using edge assembly crossover for the traveling salesman problem. *INFORMS Journal on Computing 25*(2), 346–363.

Nowicki, E. and C. Smutnicki (1996). A fast taboo search algorithm for the job shop problem. *Management Science 42*(6), 797–813.

Nowicki, E. and C. Smutnicki (2005). An advanced tabu search algorithm for the job shop problem. *Journal of Scheduling 8*, 145–159.

Öncan, T., I. K. Altınel, and G. Laporte (2009). Acomparative analysis of several asymmetric traveling salesman problem formulations. *Computers & Operations Research 36*, 637—654.

Pardalos, P. M. and O. V. Shylo (2006). An algorithm for the job shop scheduling problem based on global equilibrium search techniques. *Computational Management Science 3*, 331–348.

Peng, B., Z. Lü, and T. C. E. Cheng (2015). A tabu search/path relinking algorithm to solve the job shop scheduling problem. *Computers & Operations Research 53*, 154–164.

Pop, P. C., O. Cosma, C. Sabo, and C. P. Sitar (2024). A comprehensive survey on the generalized traveling salesman problem. *European Journal of Operational Research 314*(3), 819–835.

Reddy, D. S., A. Govardhan, and S. Sarma (2012). Hybridization of neighbourhood search metaheuristic with data mining technique to solve p-median problem. *International Journal of Computational Engineering Research 2*(7), 159–163.

Ren, Q. and Y. Wang (2012). A new hybrid genetic algorithm for job shop scheduling problem. *Computers & Operations Research 39*(10), 2291–2299.

Storer, R. H., S. D. Wu, and R. Vaccari (1992). New search spaces for sequencing problems with application to job shop scheduling. *Management Science 38*(10), 1495–1509.

Taillard, E. D. (1994). Parallel taboo search techniques for the job shop scheduling problem. *ORSA Journal on Computing 6*(2), 108–117.

Umetani, S. (2015). Exploiting variable associations to configure efficient local search in large-scale set partitioning problems. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence, January 25–30, Austin, TX, USA*, pp. 1226–1232.

Vaessens, R. J. M., E. H. L. Aarts, and J. K. Lenstra (1996). Job shop scheduling by local search. *INFORMS Journal on Computing 8*(3), 302–317.

Vandevelde, A., H. Hoogeveen, C. Hurkens, and J. K. Lenstra (2005). Lower bounds for the head-body-tail problem on parallel machines: a computational study of the multiprocessor flow shop. *INFORMS Journal on Computing 17*(3), 305–320.

Wang, Y., Q. Wu, and F. Glover (2017). Effective metaheuristic algorithms for the minimum differential dispersion problem. *European Journal of Operational Research 258*(3), 829–843.

Xie, J., X. Li, L. Gao, and L. Gui (2022). A hybrid algorithm with a new neighborhood structure for job shop scheduling problems. *Computers & Industrial Engineering 169*, 108205.

Xie, J., X. Li, L. Gao, and L. Gui (2023). A new neighbourhood structure for job shop scheduling problems. *International Journal of Production Research 61*(7), 2147–2161.

Xiong, H., S. Shi, D. Ren, and J. Hu (2022). A survey of job shop scheduling problem: The types and models. *Computers & Operations Research 142*, 105731.

Yamada, T. and R. Nakano (1992). A genetic algorithm applicable to large-scale job-shop problems. In *PPSN*, Volume 2, pp. 281–290.

Zhang, C., P. Li, Z. Guan, and Y. Rao (2007). A tabu search algorithm with a new neighborhood structure for the job shop scheduling problem. *Computers & Operations Research 34*(11), 3229–3242.

Zhang, C. Y., P. Li, Y. Rao, and Z. Guan (2008). A very fast ts/sa algorithm for the job shop scheduling problem. *Computers & Operations Research 35*(1), 282–294.

Zhang, F., Y. Mei, S. Nguyen, and M. Zhang (2023). Survey on genetic programming and machine learning techniques for heuristic design in job shop scheduling. *IEEE Transactions on Evolutionary Computation 28*(1), 147–167.

Zhou, Y., J.-K. Hao, and B. Duval (2020). Frequent pattern-based search: A case study on the quadratic assignment problem. *IEEE Transactions on Systems, Man, and Cybernetics: Systems 52*(3), 1503–1515.

Zhou, Y., J. Li, J.-K. Hao, and F. Glover (2024). Detecting critical nodes in sparse graphs via "reduce-solve-combine" memetic search. *INFORMS Journal on Computing 36*(1), 39–60.

Zhou, Y., X. Zhang, N. Geng, Z. Jiang, S. Wang, and M. Zhou (2023). Frequent itemset-driven search for finding minimal node separators and its application to air transportation network analysis. *IEEE Transactions on Intelligent Transportation Systems 24*(8), 8348–8360.

A Appendix

A.1 Problem Definition and Notations

Table 10: The Sets, Parameters and Variables Used to Formulate the JSP

Notation	Description
Sets:	
$\mathcal{M}$	Set of machines
$\mathcal{J}$	Set of jobs
Parameters:	
O_j^h	The h-th operation of job $j \in \mathcal{J}$
σ_j^h	The machine for processing O_j^h
p_{jm}	Processing time of job $j \in \mathcal{J}$ on machine $m \in \mathcal{M}$
V	Total processing time computed as: $V = \sum_{j\in\mathcal{J}} \sum_{m\in\mathcal{M}} p_{jm}$
Variables:	
x_{jm}	The integer start time of job $j \in \mathcal{J}$ on machine $m \in \mathcal{M}$
z_{jkm}	Equal to 1 if job $j \in \mathcal{J}$ precedes job $k \in \mathcal{J}$ on machine $m \in \mathcal{M}$

With the notations shown in Table 10, the JSP can be formulated as the following disjunctive model (Ku and Beck, 2016):

$$\min \quad C_{\max} \tag{14}$$

$$\text{s.t.} \quad x_{j\sigma_j^h} \geq x_{j\sigma_j^{h-1}} + p_{j\sigma_j^{h-1}}, \qquad \forall j \in \mathcal{J}, h \in \mathcal{M} \setminus \{1\} \tag{15}$$

$$x_{jm} \geq x_{km} + p_{km} - V \cdot z_{jkm}, \qquad \forall j,k \in \mathcal{J}, j < k, m \in \mathcal{M} \tag{16}$$

$$x_{km} \geq x_{jm} + p_{jm} - V \cdot (1 - z_{jkm}), \qquad \forall j,k \in \mathcal{J}, j < k, m \in \mathcal{M} \tag{17}$$

$$C_{\max} \geq x_{j\sigma_j^{|\mathcal{M}|}} + p_{j\sigma_j^{|\mathcal{M}|}}, \qquad \forall j \in \mathcal{J} \tag{18}$$

$$x_{jm} \geq 0, \qquad \forall j \in \mathcal{J}, m \in \mathcal{M} \tag{19}$$

$$z_{jkm} \in \{0,1\}, \qquad \forall j,k \in \mathcal{J}, m \in \mathcal{M} \tag{20}$$

The objective (14) is to minimize the makespan $C_{\max}$. Constraints (15) are the precedence constraints ensuring that all operations of a job are executed in the given order. The disjunctive constraints (16) and (17) ensure that two jobs can not be scheduled on the same machine at the same time. V is assigned to a large enough value $V = \sum_{j\in\mathcal{J}} \sum_{m\in\mathcal{M}} p_{jm}$

to ensure the correctness of constraints (16) and (17). Constraints (18) ensure that the makespan is at least the largest completion time of the last operation of all jobs. Constraints (19) ensure that the start time of each job is greater or equal to 0. Constraints (20) guarantee that z are binary variables.

A.2 Main Scheme of the Hash-based Two-phased Tabu Search

Algorithm 1 Pseudo-code of the Hash-based Two-phased Tabu Search

Input: The initial solution S, the matrix P, the matrix Q, the reduced search depth ω_1, the complete search depth ω_2.

Output: The best solution S^* found so far.

for $0 \leq i \leq L-1$ **do**
  $H_1[h_1(i)] = H_2[h_2(i)] = H_3[h_3(i)] = 0$ /* Initialize the hashing vectors */
end for
$N \leftarrow 0$ /* Record the number of consecutive unimproved iterations */
/* Begin the **Reduced Search** */
while $\$N < \omega_1$ **do**
  Identify the set of neighboring solutions, Neighbor_Reduce (S), for S, which satisfy the **Adjacent Job Order Fixing Rule** and the **Precedence Order Fixing Rule**
  Identify the best non-visited neighboring solution S' from the reduced neighborhood of S, i.e., $S' = \arg\min_{S_1 \in \text{Neighbor\_Reduce}(S)} C_{\max}(S_1)$, and $H_1[h_1(S')] = 0$ or $H_2[h_2(S')] = 0$ or $H_3[h_3(S')] = 0$
  $S \leftarrow S'$
  $H_1[h_1(S)] = H_2[h_2(S)] = H_3[h_3(S)] = 1$ /* Record S as the visited solution by the hash functions */
  if $C_{\max}(S) < C_{\max}(S^*)$ **then**
    $S^* \leftarrow S$, $N \leftarrow 0$
  else
    $N \leftarrow N + 1$
  end if
end while
/* Begin the **Complete Search** */
$N \leftarrow 0$, $S \leftarrow S^*$
while $N < \omega_2$ **do**
  Identify the best non-visited neighboring solution S' of S determined by the hash functions, i.e., $S' = \arg\min_{S_1 \in \text{Neighbor}(S)} C_{\max}(S_1)$, and $H_1[h_1(S')] = 0$ or $H_2[h_2(S')] = 0$ or $H_3[h_3(S')] = 0$
  $S \leftarrow S'$
  $H_1[h_1(S)] = H_2[h_2(S)] = H_3[h_3(S)] = 1$ /* Record S as the visited solution by the hash functions */
  if $C_{\max}(S) < C_{\max}(S^*)$ **then**
    $S^* \leftarrow S$, $N \leftarrow 0$
  else
    $N \leftarrow N + 1$
  end if
end while
return S^*

The general scheme of our hash-based two phased tabu search is presented in Algorithm 1. At the beginning, the algorithm first initializes the hashing vectors (lines 1-3), and then performs the reduced search phase (lines 5-16) and complete search phase (lines 17-26) to improve the current solution. During the search, each time a solution is visited, we mark the corresponding solution as a visited one by setting $H_k[h_k(S)] = 1$ ($k = 1$, 2 and 3), and the best solution encountered S^* is updated if a better solution is found. Finally, the algorithm terminates when the stopping condition is reached, and then returns the best solution S^* found during the search process.

A.3 The Effect of the Multiple Hash functions

In order to show the effectiveness of the multiple hash function strategy on reducing hashing collisions, we compare it with three other strategies in terms of the number of errors encountered during the search process. The first strategy (HTSFP2) uses one hash function and a vector of length L, the second (HTSFP3) uses two hash functions and two vectors of length L, and the third (HTSFP4) uses four hash functions and four vectors of length L. Our proposed strategy (denoted by HTSFP) uses three hash functions along with three hashing vectors of length L. An error is counted when an unvisited solution is incorrectly determined as tabu. To be specific, we perform a single run of the HTSFP for 10,000 iterations on the DUM01 instance and preserve each visited solution in a population. Each iteration checks all the neighboring solutions of the current solution to find those determined as tabu by the corresponding hashing strategy, i.e., filled with 1s in the corresponding positions of all hashing vectors. For all such neighbor solutions, the number of errors denoted by ec is incremented by one if the solution population does not contain a duplicate of this solution; otherwise the non-error counter nec is incremented by one. Finally, the error rate is calculated as $ec/(ec + nec)$.

Figure 9 depicts how the error rate changes as a function of search iterations during the search process. This figure discloses that during the first 1000 iterations, the error rate caused by any hashing strategy is very low and close to 0. Later, the error rate for the HTSFP with a single hash function increases drastically with search iterations. After iteration 4×10^6, the error rates even become 90%, making it useless to mark visited solutions for the HTSFP. On the contrary, the error rate is significantly reduced by jointly

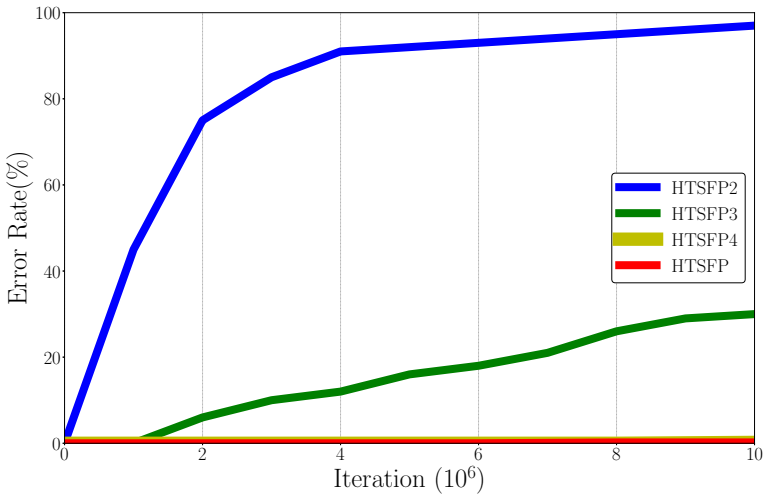
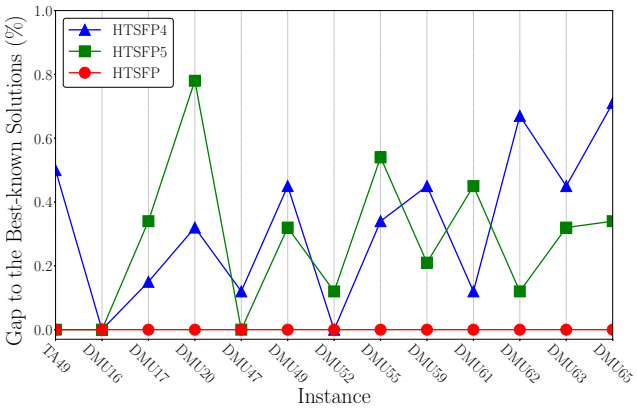


Figure 9: Error Rates of the HTSFP Iterations.

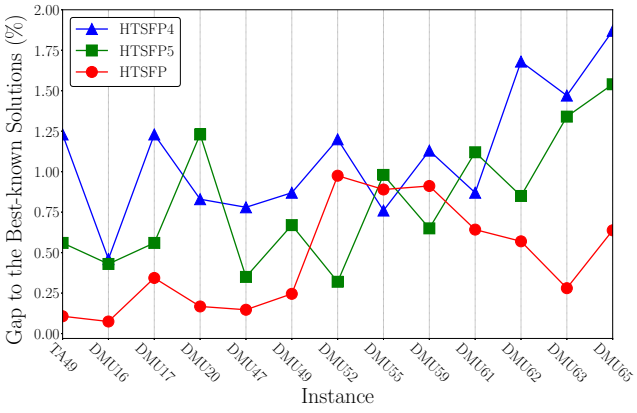
using two hashing vectors, with a maximum error rate of only 30%. Furthermore, when three or four hash functions are applied, the error rate drops very close to 0 across all iterations, with a maximum value of just 0.01%. This experiment confirms that three hash functions are sufficient to reduce the error rate to negligible levels. Adding a fourth hash function (or more) provides only very marginal improvement (with error rate differences becoming negligible as they approach 0.01%) while introducing unnecessary computational overhead for checking a solution's tabu status. Therefore, in this study, we adopt the three-hash-function strategy, which not only achieves near-zero error rate, but also avoids the redundant computational cost of a fourth function.

Using multiple hash functions can significantly reduce the likelihood of collisions. Each hash function generates a different hash value for the same input, so even if one function produces a collision, the others may not. This is analogous to a group of people, where the probability of having two individuals with the same height is quite high, but the probability of having two individuals with the same height, weight, and age is close to zero. On the other hand, using a single function where each triple (j, k, m) is assigned a different weight in the format of 2^n (n is an integer) can potentially avoid collisions. However, this approach can easily risk an integer overflow when computing the hash value, making it inapplicable.

A.4 Importance of Frequent Pattern based Method for Solution Construction



(a) Best results



(b) Average results

Figure 10: Comparisons of HTSFP with Its Variant HTSFP4&5.

As introduced in Section 4.3, the HTSFP algorithm employs a frequent pattern based method to construct new initial solutions. To verify the effectiveness of this frequent pattern based method, we compared HTSFP with two variants HTSFP4 and HTSFP5. HTSFP4 replaces the frequent pattern based method with the crossover method proposed by Xie et al. (2022), while HTSFP5 replaces the frequent pattern based method with the path relinking procedure proposed by Peng et al. (2015). To make a fair comparison, We run all approaches using selected instances and the same experimental condition like Section 6.1.

Figure 10 exhibits the comparison results. From Figure 10, one can observe that HTSFP,

HTSFP4 and HTSFP5, respectively, produce the best result in 13/10, 2/1 and 2/2 cases in terms of the best/average objective value. This experiment demonstrates that the informed frequent pattern mining mechanism positively contributes to the overall performance of the HTSFP algorithm.

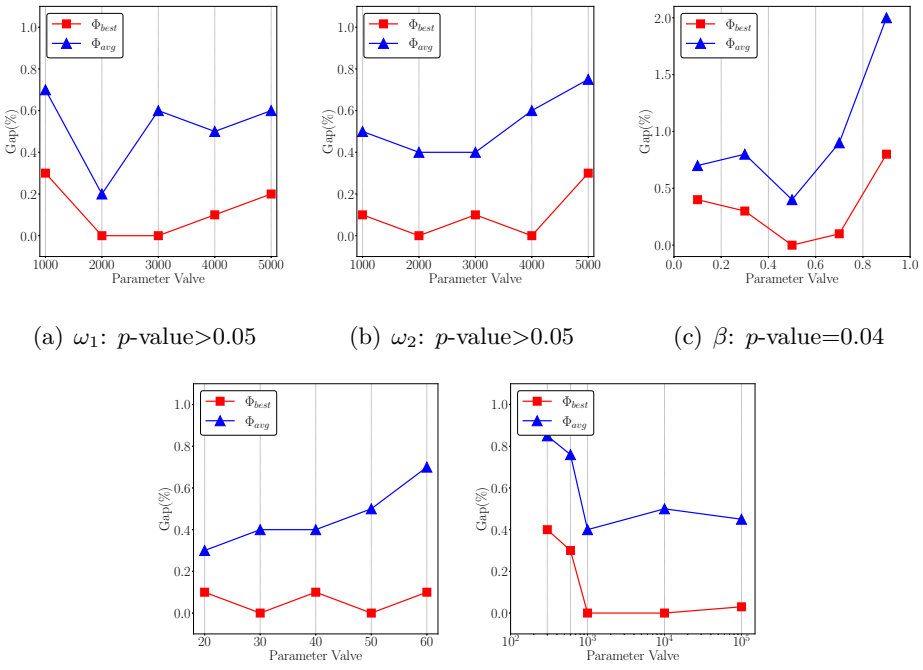
A.5 Sensitivity analysis of the parameters

The HTSFP algorithm requires five parameters as shown in Table 4, including ω_1 (Search depth of the reduced search), ω_2 (Search depth of the complete search), β (Population quality parameter), p (Number of individuals in population), and T (Range of weights used in the hash functions).

We first perform a 2-level full factorial experiment to test the interaction effects among these four parameters. The low and high levels of each parameter are respectively set as the smallest and largest values in Table 4. Since each parameter has two levels, this results in a total of 32 ($2^5 = 32$) combinations for the five parameters. The experiment is conducted on twenty randomly selected instances used in Section 5.1. Each instance is solved independently 10 times with a time limit of 300s per run for each parameter combination. Then the average results of the best objective values obtained on the twenty instances are considered for each parameter combination. The Friedman test indicates no statistically significant difference (p-values > 0.05) in terms of the considered average results, which means that the interaction effects between these five parameters are not statistically significant.

Then, for each single parameter, we perform a one-at-a-time sensitivity analysis to analyze the influence of the parameter on the performance of HTSFP and to determine its most appropriate value. To achieve this, we test its value within the range of possible values as listed in the column 4 of Table 4 while fixing the other parameters to their default values in Table 4. The HTSFP algorithm is independently run 10 times under a time limit of 300s per run for each parameter value. We report the obtained best objective value (denoted by Φ_{best}) and the average objective value (denoted by Φ_{avg}) over the 10 runs on the twenty instances in Figure 11, where the X-axis indicates the values of each parameter, and the Y-axis presents the best/average gaps to the best-known results over the four parameters. From Figure 11, we observe that the recommended parameter values from this

calibration experiment are the same as those recommended by IRACE. Especially, the choice of parameter T is crucial in determining the range of hash values $h(S)$. A small T restricts the hash value to a narrower range, potentially increasing the likelihood of collisions as fewer indexes are available to store unique solutions. For example, setting T to 300 or 600 would map many solutions to the same integer, resulting in a high collision rate, thus affecting the quality of the solutions as shown in Figure 11(e). As T increases and reaches an appropriate value such as 1,000 and 10,000, the algorithm tends to stabilize. However, an excessively large T value (e.g., 100,000) can cause integer overflow in hash calculations. In this study, T is set to 1000 to balance collision risk and computational efficiency.



(a) ω_1: p-value>0.05 (b) ω_2: p-value>0.05 (c) β: p-value=0.04

(d) p: p-value>0.05 (e) T: p-value>0.05

Figure 11: Sensitivity analysis.

Furthermore, we employ the Friedman test to determine whether there exists a statistically significant difference in solution quality for different values of a given parameter. The Friedman test indicates that the HTSFP algorithm is sensitive to the setting of β (with p-value = 0.04), while this is not the case for the other parameters. The HTSFP algorithm is sensitive to β, possibly due to the fact that a too small/large β may lead to a too much diversification/greediness into the population health management procedure. Thus, suit-

able values for the β is critical to the performance of the HTSFP algorithm. To sum up, there are no significant interaction effects among the four parameters required by HTSFP. Therefore, if the user needs to tune the parameters, more effort should be devoted to β.

A.6 Detailed Computational Results on Instances

In Tables 11 – 15, the first three columns indicate the instance name (Instance), the number of jobs and machines (Size), and the best-known lower bound and upper bound reported in the literature (UB(LB)). Columns 4–13 list the results obtained by the four compared heuristic approaches on the 50 TA instances, including the best (Best), worst (Worst) and the average (Avg.) objective value (i.e., makespan), as well as the average run time (Time(s)) in seconds to reach the best result. In the "Best" and "Worst" columns, bold values indicate that the best result of HTFSP outperforms the best result of all other compared methods, while bold values in the "Worst" column signify that the worst result of HTFSP is worse than the average result of all other compared methods. The row labeled "#Best" shows the number of instances in which the respective algorithm achieves the best result among all compared algorithms in terms of the best and average objective value. The row labeled "Avg." provides a summary of the results averaged across all tested instances. The row labeled "MRE" presents the mean relative error (MRE) of a given algorithm across tested instances.

Table 7: Comparative Results between the HTSFP and Disjunctive Model and Time-Indexed Model. '-' Indicates That the Method Fails to Report the Corresponding Results within the Given Time Limit.

		HTSFP				Disjunctive Model				Time-indexed Model			
Instance	Size	Best (UB)	Worst	Avg.	Time(s)	UB	LB	Time(s)	GAP(%)	UB	LB	Time(s)	GAP(%)
TA01	15×15	1231	1231	1231	5.2	1231	1231	179.66	0	-	-	-	-
TA02	15×15	1244	1244	1244	12.7	1244	1244	1316.25	0	-	-	-	-
TA03	15×15	1218	1218	1218	65.3	1218	1218	513.75	0	-	-	-	-
TA04	15×15	1175	1175	1175	54.1	1175	1175	376.61	0	-	-	-	-
TA05	15×15	1224	1224	1224	15.3	1224	1224	7087.49	0	-	-	-	-
TA06	15×15	1238	1238	1238	145.6	1238	1028	7200	18.67	-	-	-	-
TA07	15×15	1228	1228	1228	0.2	1227	1227	1176.51	0	-	-	-	-
TA08	15×15	1217	1217	1217	5.1	1217	1217	156.67	0	-	-	-	-
TA09	15×15	1274	1274	1274	12.3	1274	1274	693.72	0	-	-	-	-
TA10	15×15	1241	1241	1241	14.6	1241	1241	566.56	0	-	-	-	-
TA11	20×15	1357	1357	1357	98.3	1420	1141	7200	19.65	-	-	-	-
TA12	20×15	1367	1367	1367	116.3	1399	1118	7200	20.09	-	-	-	-
TA13	20×15	1342	1352	1345	85.4	1371	1115	7200	18.67	-	-	-	-
TA14	20×15	1345	1345	1345	6.1	1357	1073	7200	20.93	-	-	-	-
TA15	20×15	1339	1339	1339	74.3	1387	1089	7200	21.49	-	-	-	-
TA16	20×15	1360	1360	1360	29.2	1446	1078	7200	25.43	-	-	-	-
TA17	20×15	1462	1480	1470	85.6	1522	1185	7200	22.14	-	-	-	-
TA18	20×15	1396	1400	1397	111.7	1484	1097	7200	26.08	-	-	-	-
TA19	20×15	1332	1334	1333	72.6	1421	1027	7200	27.73	-	-	-	-
TA20	20×15	1348	1351	1350	153.9	1390	1121	7200	19.35	-	-	-	-
LA01	10×5	666	666	666	0.1	666	666	1.39	0	666	666	711.2	0
LA02	10×5	655	655	655	0.1	655	655	0.69	0	656	642	7200	2.13
LA03	10×5	597	597	597	0.1	597	597	1.38	0	605	579	7200	4.3
LA04	10×5	590	590	590	0.1	590	590	0.41	0	624	567	7200	9.13
LA05	10×5	593	593	593	0.1	593	593	5.59	0	593	571	7200	3.69
LA06	15×5	926	926	926	0.1	926	618	7200	33.26	1035	741	7200	28.34
LA07	15×5	890	890	890	0.1	890	701	7200	21.24	1041	685	7200	34.23
LA08	15×5	863	863	863	0.1	863	654	7200	24.22	1043	666	7200	36.15
LA09	15×5	951	951	951	0.1	951	636	7200	33.12	1084	731	7200	32.51
LA10	15×5	958	958	958	0.1	958	656	7200	31.52	1078	754	7200	30.06
LA11	20×5	666	666	666	0.1	1222	616	7200	49.59	-	-	-	-
LA12	20×5	655	655	655	0.1	1039	643	7200	38.11	1375	726	7200	47.24
LA13	20×5	597	597	597	0.1	1150	544	7200	52.7	1377	777	7200	41.65
LA14	20×5	590	590	590	0.1	1292	591	7200	54.26	-	-	-	-
LA15	20×5	593	593	593	0.1	1207	683	7200	43.41	-	-	-	-
LA16	10×10	945	945	945	0.1	945	945	1.73	0	-	-	-	-
LA17	10×10	784	784	784	0.1	784	784	1.94	0	830	689	7200	17.09
LA18	10×10	848	848	848	0.1	848	848	0.97	0	-	-	-	-
LA19	10×10	842	842	842	0.1	842	842	0.95	0	-	-	-	-
LA20	10×10	902	902	902	0.1	902	902	0.86	0	-	-	-	-
SWV01	20×10	1407	1407	1407	201.6	1519	913	7200	39.89	-	-	-	-
SWV02	20×10	1475	1475	1475	175	1553	857	7200	44.82	-	-	-	-
SWV03	20×10	1398	1398	1398	196.5	1516	869	7200	42.68	-	-	-	-
SWV04	20×10	1464	1472	1468.1	101.7	1610	910	7200	43.48	-	-	-	-
SWV05	20×10	1424	1426	1425.2	359.1	1563	933	7200	40.31	-	-	-	-
DMU01	20×15	2563	2563	2563	207.1	2706	2090	7200	22.76	-	-	-	-
DMU02	20×15	2706	2715	2710	13.7	2784	2297	7200	17.49	-	-	-	-
DMU03	20×15	2731	2731	2731	28.5	2893	2177	7200	24.75	-	-	-	-
DMU04	20×15	2669	2669	2669	15.2	2841	2171	7200	23.58	-	-	-	-
DMU05	20×15	2749	2751	2750.4	108.9	2848	2130	7200	25.21	-	-	-	-

Table 8: Comparative Results between the HTSFP and Disjunctive Model and Time-Indexed Model on New Larger Instances. '-' Indicates That the Method Fails to Report the Corresponding Results within the Given Time Limit.

		HTSFP				Disjunctive Model				Time-indexed Model			
Instance	Size	Best (UB)	Worst	Avg.	Time(s)	UB	LB	Time(s)	GAP(%)	UB	LB	Time(s)	GAP(%)
DMU81	80×30	9233	9386	9285.5	4351.6	14098	3764	7200	73.3	-	-	-	-
DMU82	80×30	9223	9223	9223	5631.2	-	-	-	-	-	-	-	-
DMU83	80×30	8928	9231	9123.2	4185.9	-	-	-	-	-	-	-	-
DMU84	80×30	9148	9245	9185.7	6237.8	14772	3756	7200	74.49	-	-	-	-
DMU85	80×30	9130	9268	9231.6	5412.6	-	-	-	-	-	-	-	-
DMU86	90×35	10067	10456	10213.1	6413.5	-	-	-	-	-	-	-	-
DMU87	90×35	10415	10792	10543.8	7435.6	-	-	-	-	-	-	-	-
DMU88	90×35	9969	10510	10359.4	6137.1	-	-	-	-	-	-	-	-
DMU89	90×35	10775	10819	10801.6	6531.8	-	-	-	-	-	-	-	-
DMU90	90×35	10055	10411	10256.3	5179.8	-	-	-	-	-	-	-	-

Table 9: Summary of the 13 Improved Best-Known Solutions.

			HTSFP			HA			TS/PR			BRKGA	
Instance	Size	UB(LB)	Best	Avg.	Time(s)	Best	Avg.	Time(s)	Best	Avg.	Time(s)	Best	Avg.
TA49	30×20	1963(1931)	**1960***	1962.1	450.4	1963	1967.0	592.7	1963	1971.5	1035.8	1964	1972.6
DMU16	30×20	3751(3734)	**3750***	3752.8	1366.8	3751	3754.0	917.0	3753	3765.4	1303.4	3751	3758.9
DMU17	30×20	3814(3709)	**3812***	3825.1	72.3	3814	3841.1	516.4	3819	3843.3	734.0	3830	3850.6
DMU20	30×20	3703(3604)	**3699***	3705.2	286.9	3703	3716.2	493.4	3710	3726.5	701.3	3712	3715.3
DMU47	20×20	3942(3522)	**3939***	3944.8	336.3	3943	3950.7	692.9	3942	3963.6	829.3	3939	3968.0
DMU49	20×20	3710(3403)	**3706***	3715.1	107.1	3710	3720.0	660.3	3710	3736.1	633.8	3723	3729.6
DMU52	30×15	4303(4065)	**4297***	4338.9	2344.6	4304	4341.2	1684.4	4311	4353.2	2232.6	4341	4353.2
DMU55	30×15	4263(4140)	**4258***	4284.2	1230.5	4271	4288.1	1343.4	4271	4295.2	1914.4	4290	4299.4
DMU59	30×20	4616(4217)	**4607***	4649.0	3020.4	4620	4652.0	2560.6	4624	4670.6	3614.5	4630	4633.3
DMU61	40×15	5171(4917)	**5169***	5202.2	2716.0	5171	5201.0	2635.3	5195	5217.1	4739.1	5224	5233.3
DMU62	40×15	5248(5033)	**5247***	5276.9	4231.6	5248	5277.0	3334.1	5268	5301.0	4853.8	5301	5304.4
DMU63	40×15	5313(5111)	**5312***	5326.9	4331.2	5313	5336.2	3414.7	5326	5347.5	4122.7	5357	5386.6
DMU65	40×15	5184(5105)	**5173***	5206.0	607.8	5184	5202.0	3156.9	5196	5203.2	4963.8	5197	5211.5

Table 11: Computational Results on the TA Instance Set.

			HTSFP				HA			TS/PR			BRKGA	
Instance	Size	UB(LB)	Best	Worst	Avg.	Time(s)	Best	Avg.	Time(s)	Best	Avg.	Time(s)	Best	Avg.
TA01	15×15	1231(1231)	1231	1231	1231	5.2	1231	1231	1	1231	1231	2.9	1231	1231
TA02	15×15	1244(1244)	1244	1244	1244	12.7	1244	1244	15.7	1244	1244	38.1	1244	1244
TA03	15×15	1218(1218)	1218	1218	1218	65.3	1218	1218	24.3	1218	1218	43.7	1218	1218
TA04	15×15	1175(1175)	1175	1175	1175	54.1	1175	1175	21.7	1175	1175	38.7	1175	1175
TA05	15×15	1224(1224)	1224	1224	1224	15.3	1224	1224	6.3	1224	1224	11.2	1224	1224.9
TA06	15×15	1238(1238)	1238	1238	1238	145.6	1238	1238	99.7	1238	1238.4	178.1	1238	1238.9
TA07	15×15	1227(1227)	1228	1228	1228	0.2	1228	1228	0.4	1228	1228	0.6	1228	1228
TA08	15×15	1217(1217)	1217	1217	1217	5.1	1217	1217	1.4	1217	1217	2.4	1217	1217
TA09	15×15	1274(1274)	1274	1274	1274	12.3	1274	1274	10.4	1274	1274	18.7	1274	1277
TA10	15×15	1241(1241)	1241	1241	1241	14.6	1241	1241	23.7	1241	1241	42.3	1241	1241
TA11	20×15	1357(1357)	1357	1357	1357	98.3	1357	1357	104.3	1357	1359.9	186.2	1357	1360
TA12	20×15	1367(1367)	1367	1367	1367	116.3	1367	1367	114.4	1367	1369.9	206.1	1367	1372.6
TA13	20×15	1342(1342)	1342	**1352**	1345	85.4	1342	1345	90.4	1342	1346	161.4	1344	1347.3
TA14	20×15	1345(1345)	1345	1345	1345	6.1	1345	1345	4.6	1345	1345	8.3	1345	1345
TA15	20×15	1339(1339)	1339	1339	1339	74.3	1339	1339	97.2	1339	1339	173.5	1339	1348.9
TA16	20×15	1360(1360)	1360	1360	1360	29.2	1360	1360	35.5	1360	1360	63.4	1360	1362.1
TA17	20×15	1462(1462)	1462	**1480**	1470	85.6	1462	1470	114	1463	1473	203.5	1462	1470.5
TA18	20×15	1396(1377)	1396	1400	1397	111.7	1396	1397	51.1	1396	1401	91.1	1396	1400.9
TA19	20×15	1332(1332)	1332	**1334**	1333	72.6	1332	1333	81.5	1332	1336.6	145.4	1332	1333.2
TA20	20×15	1348(1348)	1348	1351	1350	153.9	1348	1350	123.4	1348	1351.3	216.7	1348	1350.4
TA21	20×20	1642(1642)	1642	**1650**	1645	251.8	1642	1645	281.8	1644	1645.2	503	1642	1647
TA22	20×20	1600(1561)	1600	1600	1600	102	1600	1600.3	123.2	1600	1603.8	228.9	1600	1600
TA23	20×20	1557(1518)	1557	1561	1559	154.3	1557	1558.1	201.6	1557	1559.6	359.8	1557	1562.6
TA24	20×20	1644(1644)	1644	1647	1645.8	510.9	1644	1646	436.6	1645	1647.7	779.3	1646	1650.6
TA25	20×20	1595(1558)	1595	1597	1595.4	150.3	1595	1595.7	233.1	1595	1597	416.1	1595	1602
TA26	20×20	1643(1591)	1643	1651	1646.5	230.5	1645	1647	149.9	1647	1651.4	267.5	1643	1652.3
TA27	20×20	1680(1652)	1680	**1689**	1683.1	105.3	1680	1682.3	143.8	1680	1686.7	254.7	1680	1685.6
TA28	20×20	1603(1603)	1603	1607	1604.5	121.4	1603	1605.2	183.8	1613	1616.2	326.2	1603	1611.7
TA29	20×20	1625(1583)	1625	1625	1625	26.9	1625	1625.4	52.5	1625	1627.4	93.5	1625	1627.4
TA30	20×20	1584(1528)	1584	1587	1585	153.8	1584	1588.5	217.7	1584	1588.3	388.7	1584	1588.5
TA31	30×15	1764(1764)	1764	1764	1764	41.5	1764	1764	19.9	1764	1764	35.6	1764	1764.4
TA32	30×15	1784(1774)	1785	1789	1787	516.6	1785	1787	393.9	1787	1803.5	703.1	1785	1794.1
TA33	30×15	1791(1788)	1791	1793	1791.5	127.7	1791	1792.3	256.3	1791	1794.6	457.6	1791	1793.7
TA34	30×15	1828(1828)	1829	1829	1829	98.3	1829	1832	176.9	1829	1831.2	315.7	1829	1832.1
TA35	30×15	2007(2007)	2007	2007	2007	0.1	2007	2007	0.3	2007	2007	0.6	2007	2007
TA36	30×15	1819(1819)	1819	1819	1819	54.1	1819	1819	68.7	1819	1819	122.7	1819	1822.9
TA37	30×15	1771(1771)	1771	1775	1772.6	285.3	1771	1773	365.4	1771	1776.8	652.2	1771	1777.8
TA38	30×15	1673(1673)	1673	1673	1673	94.4	1673	1673	172.2	1673	1673	307.3	1673	1676.7
TA39	30×15	1795(1795)	1795	1795	1795	85.6	1795	1795	64.8	1795	1795	115.6	1795	1801.6
TA40	30×15	1669(1651)	1669	1678	1673	435.4	1670	1675	252.1	1671	1676	450	1669	1678.1
TA41	30×20	2006(1906)	2006	2013	2011.2	521.6	2006	2010.1	710.2	2010	2018.6	1267.8	2008	2018.7
TA42	30×20	1937(1884)	1937	**1951**	1943	863.8	1939	1946	871.9	1949	1950.3	1556.4	1937	1949.3
TA43	30×20	1846(1809)	1846	1855	1850	1030.8	1846	1850	967.3	1846	1865.1	1726.8	1852	1863.1
TA44	30×20	1979(1948)	1979	1982	1980.1	639.7	1979	1981.2	731.9	1982	1989.1	1304.7	1983	1992.4
TA45	30×20	2000(1997)	2000	2001	2000.6	513.1	2000	2004	592.8	2000	2000.5	1057.8	2000	2000
TA46	30×20	2004(1957)	2004	2015	2009.1	899.7	2006	2010	692.2	2008	2022.3	1236	2004	2015.5
TA47	30×20	1889(1807)	1889	1902	1900.4	854.2	1896	1902	577.5	1889	1906.2	1030.9	1894	1902.1
TA48	30×20	1937(1912)	1937	1952	1946	566.7	1937	1952	587.7	1947	1955.5	1047.4	1943	1959.2
TA49	30×20	1963(1931)	**1960***	1965	1962.1	450.4	1963	1967	592.7	1963	1971.5	1035.8	1964	1972.6
TA50	30×20	1923(1833)	1923	**1934**	1927	779.3	1923	1928	738.8	1923	1931.4	1318.1	1925	1927
#best			50		46		44	31		37	17		41	12
Avg.			1559.7	1593.6	1561.4	236.6	1559.9	1561.7	237.6	1560.6	1564.0	423.8	1560.2	1564.2
MRE			0.97	1.19	1.08		0.99	1.11		1.04	1.26		1.01	1.29

Table 12: Computational Results on the LA Instance Set.

			HTSFP				HA			TS/PR			BRKGA	
Instance	Size	OPT	Best	Worst	Avg.	Time(s)	Best	Avg.	Time(s)	Best	Avg.	Time(s)	Best	Avg.
LA01	10×5	666	666	666	666	0.1	666	666	0.0	666	666	0.1	666	666.0
LA02	10×5	655	655	655	655	0.1	655	655	0.0	655	655	0.1	655	655.0
LA03	10×5	597	597	597	597	0.1	597	597	0.0	597	597	0.1	597	597.0
LA04	10×5	590	590	590	590	0.1	590	590	0.0	590	590	0.1	590	590.0
LA05	10×5	593	593	593	593	0.1	593	593	0.1	593	593	0.1	593	593.0
LA06	15×5	926	926	926	926	0.1	926	926	0.1	926	926	0.1	926	926.0
LA07	15×5	890	890	890	890	0.1	890	890	0.1	890	890	0.1	890	890.0
LA08	15×5	863	863	863	863	0.1	863	863	0.0	863	863	0.1	863	863.0
LA09	15×5	951	951	951	951	0.1	951	951	0.0	951	951	0.1	951	951.0
LA10	15×5	958	958	958	958	0.1	958	958	0.1	958	958	0.1	958	958.0
LA11	20×5	1222	1222	1222	1222	0.1	1222	1222	0.1	1222	1222	0.1	1222	1222.0
LA12	20×5	1039	1039	1039	1039	0.1	1039	1039	0.1	1039	1039	0.1	1039	1039.0
LA13	20×5	1150	1150	1150	1150	0.1	1150	1150	0.1	1150	1150	0.1	1150	1150.0
LA14	20×5	1292	1292	1292	1292	0.1	1292	1292	0.1	1292	1292	0.1	1292	1292.0
LA15	20×5	1207	1207	1207	1207	0.1	1207	1207	0.1	1207	1207	0.1	1207	1207.0
LA16	10×10	945	945	945	945	0.1	945	945	0.1	945	945	0.2	945	945.0
LA17	10×10	784	784	784	784	0.1	784	784	0.1	784	784	0.1	784	784.0
LA18	10×10	848	848	848	848	0.1	848	848	0.1	848	848	0.1	848	848.0
LA19	10×10	842	842	842	842	0.1	842	842	0.0	842	842	0.2	842	842.0
LA20	10×10	902	902	902	902	0.1	902	902	0.1	902	902	0.1	902	902.0
LA21	15×10	1046	1046	1046	1046	2.5	1046	1046	4.7	1046	1046	7.3	1046	1046.0
LA22	15×10	927	927	927	927	2.1	927	927	3.4	927	927	3.9	927	927.0
LA23	15×10	1032	1032	1032	1032	0.2	1032	1032	0.1	1032	1032	0.1	1032	1032.0
LA24	15×10	935	935	935	935	2.1	935	935	3.1	935	935	3.1	935	935.0
LA25	15×10	977	977	977	977	0.9	977	977	1.1	977	977	1.4	977	977.0
LA26	20×10	1218	1218	1218	1218	0.1	1218	1218	0.2	1218	1218	0.3	1218	1218.0
LA27	20×10	1235	1235	1235	1235	1.2	1235	1235	1.9	1235	1235	2.2	1235	1235.0
LA28	20×10	1216	1216	1216	1216	0.1	1216	1216	0.3	1216	1216	0.4	1216	1216.0
LA29	20×10	1153	1153	1153	1153	11.6	1153	1153	35.5	1153	1153	73.8	1153	1154.7
LA30	20×10	1355	1355	1355	1355	0.3	1355	1355	0.2	1355	1355	0.3	1355	1355.0
LA31	30×10	1784	1784	1784	1784	0.2	1784	1784	0.2	1784	1784	0.3	1784	1784.0
LA32	30×10	1850	1850	1850	1850	0.4	1850	1850	0.2	1850	1850	0.3	1850	1850.0
LA33	30×10	1719	1719	1719	1719	0.7	1719	1719	0.2	1719	1719	0.3	1719	1719.0
LA34	30×10	1721	1721	1721	1721	0.1	1721	1721	0.2	1721	1721	0.3	1721	1721.0
LA35	30×10	1888	1888	1888	1888	0.2	1888	1888	0.1	1888	1888	0.3	1888	1888.0
LA36	15×15	1268	1268	1268	1268	1.5	1268	1268	2.6	1268	1268	4.5	1268	1268.0
LA37	15×15	1397	1397	1397	1397	10.3	1397	1397	15.8	1397	1397	26.2	1397	1397.0
LA38	15×15	1196	1196	1196	1196	11.7	1196	1196	21.3	1196	1196	32.6	1196	1196.0
LA39	15×15	1233	1233	1233	1233	6.2	1233	1233	5.7	1233	1233	11.6	1233	1233.0
LA40	15×15	1222	1222	1222	1222	85.4	1222	1222	110.3	1222	1222	384.8	1222	1223.2
#best			40		40		40	40		40	40		40	38
Avg.			1081.3	1081.3	1081.3	3.5	1081.3	1081.3	5.2	1081.3	1081.3	13.9	1081.3	1081.3
MRE			0.002	0.002	0.002		0.002	0.002		0.002	0.002		0.002	0.008

Table 13: Computational Results on the SWV Instance Set.

			HTSFP				HA			TS/PR			BRKGA	
Instance	Size	UB(LB)	Best	Worst	Avg.	Time(s)	Best	Avg.	Time(s)	Best	Avg.	Time(s)	Best	Avg.
SWV01	20×10	1407(1407)	1407	1407	1407.0	201.6	1407	1408.9	272.3	1407	1411.4	575.8	1407	1408.9
SWV02	20×10	1475(1475)	1475	1475	1475.0	175.0	1475	1475.0	189.7	1475	1475.1	294.1	1475	1478.2
SWV03	20×10	1398(1398)	1398	1398	1398.0	196.5	1398	1398.0	315.3	1398	1398.9	613.0	1398	1400.0
SWV04	20×10	1464(1464)	1464	1472	1468.1	101.7	1464	1470.1	189.6	1464	1473.5	257.6	1470	1472.8
SWV05	20×10	1424(1424)	1424	1426	1425.2	359.1	1424	1425.0	313.5	1425	1426.0	612.8	1425	1431.4
SWV06	20×15	1667(1630)	1667	1675	1671.1	201.7	1667	1670.0	182.5	1671	1675.9	385.7	1675	1682.1
SWV07	20×15	1594(1513)	1594	**1610**	1598.1	286.9	1595	1600.0	316.5	1595	1605.0	626.5	1594	1601.2
SWV08	20×15	1751(1671)	1752	1762	1756.1	195.6	1752	1758.2	253.0	1752	1760.4	503.0	1755	1764.3
SWV09	20×15	1655(1633)	1655	1659	1657.2	165.9	1655	1658.9	128.5	1655	1661.8	521.9	1656	1667.9
SWV10	20×15	1743(1663)	1743	1748	1745.6	128.6	1743	1748.0	138.9	1743	1756.6	441.4	1743	1754.6
SWV11	50×10	2983(2983)	2983	2983	2983.0	359.0	2983	2983.0	420.7	2983	2984.5	940.7	2983	2985.9
SWV12	50×10	2972(2972)	2972	2981	2976.3	1563.7	2972	2977.8	2763.8	2977	2985.3	6097.4	2979	2989.7
SWV13	50×10	3104(3104)	3104	3104	3104.0	714.8	3104	3104.0	562.1	3104	3104.0	1111.2	3104	3111.6
SWV14	50×10	2968(2968)	2968	2968	2968.0	243.6	2968	2968.0	321.4	2968	2968.0	422.8	2968	2968.0
SWV15	50×10	2885(2885)	2885	2898	2890.1	1956.7	2885	2889.0	2455.8	2885	2889.4	6000.6	2901	2902.9
#best			15		12		14	8		11	2		8	1
Avg.			1906.6	2037.7	1908.4	456.7	1906.6	1908.9	588.2	1907.1	1911.1	1293.6	1908.8	1913.8
MRE			1.24	1.52	1.36		1.25	1.41		1.28	1.56		1.38	1.70

Table 14: Computational Results on the DMU Instance Set.

			HTSFP				HA			TS/PR			BRKGA	
Instance	Size	UB(LB)	Best	Worst	Avg.	Time(s)	Best	Avg.	Time(s)	Best	Avg.	Time(s)	Best	Avg.
DMU01	20×15	2563(2501)	2563	2563	2563.0	207.1	2563	2563.0	234.2	2563	2563.0	332.9	2563	2563.0
DMU02	20×15	2706(2651)	2706	**2715**	2710.0	13.7	2706	2710.0	126.1	2706	2713.2	179.2	2706	2714.5
DMU03	20×15	2731(2731)	2731	2731	2731.0	28.5	2731	2732.3	273.4	2731	2733.1	388.6	2731	2736.5
DMU04	20×15	2669(2601)	2669	2669	2669.0	15.2	2669	2669.0	67.9	2669	2670.2	96.5	2669	2672.4
DMU05	20×15	2749(2749)	2749	2751	2750.4	108.9	2749	2755.4	213.2	2749	2758.6	303.0	2749	2755.4
DMU06	20×20	3244(2998)	3244	**3250**	3247.1	699.7	3244	3246.0	579.1	3245	3249.2	823.2	3244	3246.6
DMU07	20×20	3046(2815)	3046	3060	3052.3	341.8	3046	3050.1	253.7	3046	3062.3	360.6	3046	3058.6
DMU08	20×20	3188(3051)	3188	3188	3188.0	30.3	3188	3188.0	208.1	3188	3194.3	295.8	3188	3188.3
DMU09	20×20	3092(2956)	3092	3094	3093.0	100.3	3092	3093.0	104.1	3094	3097.4	148.0	3092	3094.4
DMU10	20×20	2984(2858)	2984	2984	2984.0	236.2	2984	2984.0	177.6	2985	2991.0	252.5	2984	2984.8
DMU11	30×15	3430(3395)	3430	3441	3435.0	1411.0	3430	3434.0	1053.1	3430	3435.2	1496.9	3445	3445.8
DMU12	30×15	3492(3418)	3492	3501	3497.3	265.6	3492	3496.1	633.2	3495	3509.7	900.0	3513	3518.9
DMU13	30×15	3681(3681)	3681	3681	3681.0	54.5	3681	3681.0	437.7	3681	3682.8	622.1	3681	3690.6
DMU14	30×15	3394(3394)	3394	3394	3394.0	0.9	3394	3394.0	2.1	3394	3394.0	3.0	3394	3394.0
DMU15	30×15	3343(3343)	3343	3343	3343.0	0.4	3343	3343.0	1.3	3343	3343.0	1.8	3343	3343.0
DMU16	30×20	3751(3734)	**3750***	3755	3752.8	1366.8	3751	3754.0	917.0	3753	3765.4	1303.4	3751	3758.9
DMU17	30×20	3814(3709)	**3812***	3841	3825.1	72.3	3814	3841.1	516.4	3819	3843.3	734.0	3830	3850.6
DMU18	30×20	3844(3844)	3844	3844	3844.0	3509.3	3844	3844.0	2664.5	3844	3849.5	3787.4	3844	3845.4
DMU19	30×20	3764(3669)	3764	**3792**	3781.3	726.9	3764	3781.0	505.6	3768	3787.4	718.7	3770	3791.8
DMU20	30×20	3703(3604)	**3699***	3712	3705.2	286.9	3703	3716.2	493.4	3710	3726.5	701.3	3712	3715.3
DMU21	40×15	4380(4380)	4380	4380	4380.0	0.5	4380	4380.0	0.5	4380	4380.0	0.7	4380	4380.0
DMU22	40×15	4275(4275)	4275	4275	4275.0	0.8	4275	4275.0	1.0	4725	4725.0	1.5	4725	4725.0
DMU23	40×15	4668(4668)	4668	4668	4668.0	0.9	4668	4668.0	0.9	4668	4668.0	1.3	4668	4668.0
DMU24	40×15	4648(4648)	4648	4648	4648.0	0.3	4648	4648.0	0.5	4648	4648.0	0.8	4648	4648.0
DMU25	40×15	4164(4164)	4164	4164	4164.0	0.3	4164	4164.0	0.4	4164	4164.0	0.6	4164	4164.0
DMU26	40×20	4647(4647)	4647	4647	4647.0	1504.7	4647	4647.0	1147.7	4647	4647.3	1631.4	4647	4658.4
DMU27	40×20	4848(4848)	4848	4848	4848.0	0.1	4848	4848.0	8.6	4848	4848.0	12.2	4848	4848.0
DMU28	40×20	4692(4692)	4692	4692	4692.0	8.4	4692	4692.0	12.4	4692	4692.0	17.7	4692	4692.0
DMU29	40×20	4691(4691)	4691	4691	4691.0	50.1	4691	4691.0	44.7	4691	4691.0	63.5	4691	4691.0
DMU30	40×20	4732(4732)	4732	4732	4732.0	31.7	4732	4732.0	86.5	4732	4732.0	123.0	4732	4732.0
DMU31	50×15	5640(5640)	5640	5640	5640.0	0.5	5640	5640.0	0.6	5640	5640.0	0.8	5640	5640.0
DMU32	50×15	5927(5927)	5927	5927	5927.0	0.3	5927	5927.0	0.4	5927	5927.0	0.6	5927	5927.0
DMU33	50×15	5728(5728)	5728	5728	5728.0	0.0	5728	5728.0	0.3	5728	5728.0	0.4	5728	5728.0
DMU34	50×15	5385(5385)	5385	5385	5385.0	0.1	5385	5385.0	1.6	5385	5385.0	2.2	5385	5385.0
DMU35	50×15	5635(5635)	5635	5638	5636.0	0.4	5635	5636.0	0.5	5635	5635.0	0.7	5635	5635.0
DMU36	50×20	5621(5621)	5621	5621	5621.0	3.5	5621	5621.0	5.5	5621	5621.0	7.8	5621	5621.0
DMU37	50×20	5851(5851)	5851	5851	5851.0	11.4	5851	5851.0	8.0	5851	5851.0	11.4	5851	5851.0
DMU38	50×20	5713(5713)	5713	5713	5713.0	4.3	5713	5713.0	7.5	5713	5713.0	10.7	5713	5713.0
DMU39	50×20	5747(5747)	5747	5747	5747.0	0.6	5747	5747.0	1.4	5747	5747.0	2.0	5747	5747.0
DMU40	50×20	5577(5577)	5577	5577	5577.0	4.1	5577	5577.0	3.5	5577	5577.0	4.9	5577	5577.0

Table 15: (Continue.) Computational Results on the DMU Instance Set.

			HTSFP				HA			TS/PR			BRKGA	
Instance	Size	UB(LB)	Best	Worst	Avg.	Time(s)	Best	Avg.	Time(s)	Best	Avg.	Time(s)	Best	Avg.
DMU41	20×15	3248(3007)	3248	3280	3268.5	421.0	3248	3274.0	294.0	3248	3281.9	417.8	3261	3281.9
DMU42	20×15	3390(3172)	3390	3401	3394.5	406.6	3390	3395.5	315.8	3390	3409.8	449.0	3395	3403.9
DMU43	20×15	3441(3292)	3441	3452	3446.7	2.1	3441	3447.0	3.5	3441	3450.5	399.3	3441	3452.7
DMU44	20×15	3475(3283)	3475	3495	3488.3	212.8	3475	3490.0	280.9	3489	3509.7	371.3	3488	3510.7
DMU45	20×15	3266(3001)	3266	**3293**	3283.7	193.3	3267	3285.0	261.2	3273	3287.9	709.2	3272	3287.3
DMU46	20×20	4035(3575)	4035	**4056**	4043.2	365.2	4035	4042.2	498.9	4035	4051.8	984.9	4035	4043.2
DMU47	20×20	3942(3522)	**3939***	3948	3944.8	336.3	3943	3950.7	692.9	3942	3963.6	829.3	3939	3968.0
DMU48	20×20	3763(3447)	3763	3805	3798.7	508.1	3763	3798.2	583.4	3778	3814.1	938.6	3781	3800.9
DMU49	20×20	3710(3403)	**3706***	3721	3715.1	107.1	3710	3720.0	660.3	3710	3736.1	633.8	3723	3729.6
DMU50	20×20	3729(3496)	3729	3746	3739.9	645.8	3729	3740.0	445.9	3729	3741.2	609.6	3732	3746.5
DMU51	30×15	4156(3917)	4156	4203	4192.5	40.7	4156	4200.0	428.9	4167	4205.9	2394.3	4201	4222.9
DMU52	30×15	4303(4065)	**4297***	4350	4338.9	2344.6	4304	4341.2	1684.4	4311	4353.2	2232.6	4341	4353.2
DMU53	30×15	4378(4141)	4381	**4453**	4421.7	1130.2	4388	4421.4	1570.7	4394	4425.7	2161.8	4415	4420.2
DMU54	30×15	4361(4202)	4361	**4399**	4379.5	157.0	4361	4381.2	1520.9	4371	4390.5	1909.5	4396	4402.7
DMU55	30×15	4263(4140)	**4258***	**4310**	4284.2	1230.5	4271	4288.1	1343.4	4271	4295.2	1914.4	4290	4299.4
DMU56	30×20	4939(4554)	4939	**4998**	4974.6	1700.8	4939	4976.0	1346.8	4941	4990.6	3825.4	4961	4768.4
DMU57	30×20	4647(4302)	4647	**4723**	4698.1	1116.8	4647	4701.0	2691.3	4663	4714.0	3649.4	4698	4704.9
DMU58	30×20	4701(4319)	4708	4769	4752.1	922.3	4708	4760.0	2567.4	4708	4779.4	3639.7	4751	4752.8
DMU59	30×20	4616(4217)	**4607***	4657	4649.0	3020.4	4620	4652.0	2560.6	4624	4670.6	3614.5	4630	4633.3
DMU60	30×20	4721(4319)	4745	4781	4763.9	3135.2	4745	4770.0	2542.9	4755	4804.3	3745.9	4774	4777.0
DMU61	40×15	5171(4917)	**5169***	5231	5202.2	2716.0	5171	5201.0	2635.3	5195	5217.1	4739.1	5224	5233.3
DMU62	40×15	5248(5033)	**5247***	5287	5276.9	4231.6	5248	5277.0	3334.1	5268	5301.0	4853.8	5301	5304.4
DMU63	40×15	5313(5111)	**5312***	5341	5326.9	4331.2	5313	5336.2	3414.7	5326	5347.5	4122.7	5357	5386.6
DMU64	40×15	5226(5130)	5226	**5289**	5260.8	3473.9	5226	5269.1	2900.4	5252	5279.8	4487.3	5312	5321.8
DMU65	40×15	5184(5105)	**5173***	**5218**	5206.0	607.8	5184	5202.0	3156.9	5196	5203.2	4963.8	5197	5211.5
DMU66	40×20	5701(5391)	5714	5768	5749.0	1532.9	5714	5756.0	3492.1	5717	5788.7	9543.9	5796	5806.6
DMU67	40×20	5779(5589)	5779	5825	5802.7	9127.9	5779	5810.0	6714.3	5816	5852.5	8431.5	5863	5881.3
DMU68	40×20	5763(5426)	5763	**5912**	5894.3	4366.7	5763	5902.0	5931.7	5773	5801.8	8739.5	5826	5843.7
DMU69	40×20	5688(5423)	5688	5701	5694.6	8358.8	5688	5700.0	6148.4	5709	5754.4	8107.6	5775	5804.0
DMU70	40×20	5868(5501)	5868	5934	5916.4	4891.3	5868	5921.0	5703.9	5903	5924.2	7285.3	5951	5968.2
DMU71	50×15	6207(6080)	6207	6251	6231.4	5601.8	6207	6232.0	5125.3	6223	6264.8	9835.1	6293	6603.8
DMU72	50×15	6463(6395)	6463	6517	6497.9	6476.3	6463	6507.1	6919.2	6483	6510.9	10881.8	6503	6560.7
DMU73	50×15	6136(6001)	6136	6199	6178.4	8676.4	6136	6187.2	7655.5	6163	6199.8	11475.2	6219	6250.5
DMU74	50×15	6196(6123)	6196	6213	6205.8	5434.6	6196	6211.1	8073.0	6227	6266.4	11164.4	6277	6312.6
DMU75	50×15	6189(6029)	6189	6221	6203.0	4668.8	6189	6200.1	7854.4	6197	6239.4	11330.9	6248	6282.4
DMU76	50×20	6718(6342)	6718	**6897**	6814.7	7948.6	6718	6810.0	7971.5	6813	6854.8	9998.2	6876	6885.4
DMU77	50×20	6747(6499)	6747	6812	6790.7	4147.4	6747	6800.0	7033.9	6822	6879.9	12062.9	6857	6892.7
DMU78	50×20	6755(6586)	6755	**6901**	6801.0	3878.3	6755	6801.0	8486.5	6770	6813.2	10346.6	6831	6855.7
DMU79	50×20	6910(6650)	6910	6998	6971.4	8253.7	6910	6970.0	7279.0	6970	7003.0	9818.9	7049	7060.9
DMU80	50×20	6634(6459)	6634	6701	6669.5	5035.7	6634	6670.0	6907.8	6686	6700.1	10332.0	6736	6757.9
#best			80		64		66	41		37	22		36	24
Avg.			4553.3	4636.7	4569.7	1660.7	4554.0	4571.4	1872.8	4568.1	4586.3	2791.2	4584.3	4595.1
MRE			3.41	3.98	3.76		3.43	3.81		3.73	4.15		4.05	4.27