RAIRO-Oper. Res. 58 (2024) 373–395
https://doi.org/10.1051/ro/2024001

RAIRO Operations Research
www.rairo-ro.org

MINIMIZING THE MAXIMUM TARDINESS FOR A PERMUTATION FLOW SHOP PROBLEM UNDER THE CONSTRAINT OF SEQUENCE INDEPENDENT SETUP TIME

Oualid Elissaouy* and Karam Allali

Abstract. In this work, we will study a permutation flow shop scheduling problem under the constraint of sequence independent setup time. In our case, each machine requires a certain setup time to process all the different jobs assigned to it. Hence, this setup time will be independent of sequence of jobs, but will depend only on the nature of machine. The optimization objective is to minimize the maximum tardiness criterion. To solve this optimization problem, an exact method, heuristics and metaheuristics are the three main resolution methods that we have used. The exact method is represented by the mixed integer linear programming (MILP) model. In terms of the second category of resolution methods, we have been focused on two methods, the first is a modified heuristic based on Johnson rule (HBJR) while the second is based on the Nawaz–Enscore–Ham (NEH) algorithm. Finally, three metaheuristics have been used, namely the iterated local search (ILS) method, the iterated greedy (IG) algorithm and the genetic algorithm (GA). Our numerical results indicate that for the problems with small size instances, the NEH heuristic outperforms HBJR approach, while for relatively large size instances, the developed IG algorithm gives best results than both other metaheuristics ILS and GA.

Mathematics Subject Classification. 68W50, 90B35, 90C59.

Received April 24, 2022. Accepted December 30, 2023.

1. Introduction

In the current era of industrial development, characterized by both technical and organizational advancements, scheduling in a manufacturing environment can be defined as allocating, over time, specific resources (machines, raw materials, ...) to a sequence of tasks (jobs, operations, ...) in a certain order to have a timely optimal product [1]. The objective of scheduling is then to determine a sequence of jobs that optimizes one criterion or even more under various constraints that reflect the reality of the production environment, such as, processing times, job arrival dates, deadlines, machine availability, resource preparation times, whether or not production resources are shared, the limited capacity of machines, etc. Two primary characteristics are considered when classifying scheduling problems. The first characteristic is related to the types of the used machines during the production process, while the second is determined by the machines' performance and their installation

Keywords. Permutation flow shop scheduling, sequence independent setup time, maximum tardiness, MILP, heuristics, metaheuristics.

Laboratory Mathematics, Computer Science and Applications, University Hassan II of Casablanca, FST, PO Box 146, Mohammedia, Morocco.
*Corresponding author: oualid.elissaouy@gmail.com

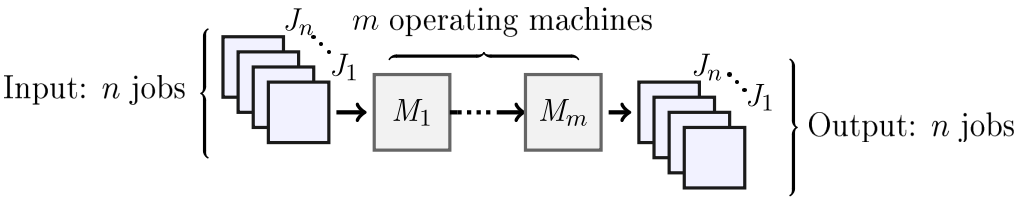

FIGURE 1. Flow shop problem with n jobs and m machines.

configuration (in parallel, in series or mixed) within the workshop. This classification system allows for the identification of the three typical categories that are flow-shop, job-shop and open-shop.

The permutation flow shop scheduling problem (PFSP) is widely regarded as one of the most challenging and complex optimization problems in the field of scheduling, particularly in the context of manufacturing. This problem involves scheduling a set of tasks or jobs across multiple machines or workstations in a predetermined order to optimize a given objective function while satisfying a set of constraints that reflect the practical limitations of the production environment. In Figure 1, we have depicted a workshop that consists of n jobs that must be executed on m machines following the same path, meaning that each job must go through all machines from the first to the last one. In this paper, we will address the PFSP scheduling problem under the sequence independent setup time (SIST) constraint. Incorporating this constraint in scheduling problems leads to more accurate and realistic manufacturing environment models by considering real-world limitations such as machine maintenance, changeover times, and production interruptions. By accurately modeling these constraints, scheduling strategies can be developed to maximize production efficiency and avoid waste times. The PFSP problem has been a subject of interest for numerous researchers in the manufacturing sector for over six decades. Indeed, the first work in this field was developed by Johnson [2], who dealt with the two and three machines flow shop scheduling problem to minimize the maximum of job completion time, also known as makespan. The author considered setup time as part of the job processing time. According to the literature, Yoshida and Hitomi [3] were the first to handle the flow shop problem with machine setup time separated from job processing time. In fact, setup time is often described as the time spent for preparing resources (such as machines and persons) to complete a task (*e.g.*, job, operation). The two well-known types are SIST and sequence-dependent setup time (SDST). The first is when the setup time depends solely on the job to be processed, regardless of its previous job. SDST, on the other hand, which occurs when preparation time is determined by both the job and the job before it [4].

Scheduling issues are commonly classified as NP-hard problems, it is very difficult to find the optimal scheduled solution even for moderate size instances [5]. As a result, the combinatorial optimization community has developed various methods to solve these problems. These methods can be broadly categorized into two main groups. The first group consists of exact methods, which aim to generate solutions that converge towards the optimal solution. However, the optimality of these solutions cannot be guaranteed for the problems dealing with relatively large size instances. On the other hand, the second group covers approximate methods that have gained significant interest due to their ability to generate high-quality solutions in a shorter time. This group can be further divided into two main approaches: heuristics and metaheuristics, both of which aim to efficiently explore the research space to find a good solution.

The PFSP scheduling problem has been extensively studied by researchers, who have utilized various constraints and resolution methods to address it. These constraints include, for instance, SDST, no-idle, no-waiting and no-permutation. Nonetheless, we will concentrate on the PFSP scheduling problem with SIST constraint, which has attracted significant attention for scheduling researchers. The complexity of PFSP-SIST problem poses a significant challenge, necessitating the development of efficient algorithms and heuristics to address it. Therefore, this paper aims to contribute to the existing body of knowledge by focusing on this particular problem. In order to contextualize our study, we present a comprehensive overview of the existing research and

literature on the PFSP scheduling problem. We will examine the various utilized constraints, the involved optimization criteria and the different approaches employed to solve this problem. Indeed, in [6], a hybrid approach combining genetic algorithms with variable neighborhood search was proposed to solve the PFSP problem with setup time. Similarly, Belabid *et al.* [7] employed mixed integer linear programming (MILP) and various heuristics and metaheuristics to investigate the PFSP-SIST problem, aiming to minimize the makespan criterion. On the other hand, to optimize the total flow time criteria, the work [8] compared Nawaz–Enscore–Ham (NEH) algorithm and an heuristic based on Johnson rule (HBJR) to an exact method in the context of PFSP-SIST. Besides, some works have considered bi-criteria objective functions, such as in [9], who developed a heuristic to minimize the makespan and maximum tardiness of all jobs in a flow shop scheduling problem. In [10], the authors investigated a multi-objective optimization distributed no-wait permutation flow shop (DNWPFSP) scheduling problem under the constraint of SDST, using a MILP and a set of efficient meta-heuristics.

In this paper, we are interested in minimizing the maximum tardiness by solving the scheduling problem PFSP under the SIST constraint. There are only a few papers in the literature that discuss the minimization of this criterion in the context of a PFSP issue. In contrast, because achieving client deadlines is a major concern for many production systems, optimizing this criterion seems very important. In this regard, we mobilize the approaches mentioned above to solve our problem. First, we propose an exact method based on a mixed-integer linear programming model. Next, we use two heuristics, the first is HBJR, while the second is based on the NEH algorithm. Furthermore, we select three of the most effective metaheuristics in the literature, namely, the iterated local search (ILS) algorithm, the iterated greedy (IG) algorithm and the genetic algorithm (GA).

The remaining sections of our paper are structured as follows: In Section 2, we will provide a review of the literature on flow shop scheduling problems. In Section 3, the problem will be described. We present all of the suggested resolution approaches in Section 4. A comparison study between the different methods of resolution is discussed in Section 5. Finally, we conclude our study and summarize the results findings in Section 6.

2. Literature review

Scheduling plays a crucial role in the manufacturing sector as it ensures that the right resources are allocated to specific tasks at the right time, resulting in optimal productivity and efficient use of resources. By scheduling the tasks, manufacturers can better plan their production processes, ensure on-time delivery of goods to customers, reduce downtime and minimize overproduction or underproduction associated costs. It also helps manufacturers to manage their inventory levels, adjust production volumes based on customer demand and adapt to changing market conditions. Therefore, scheduling is an essential decision-making process that enables manufacturers to remain competitive in the constantly changing manufacturing industry [11]. In the past, scheduling was often a manual process that required a significant amount of time and effort. However, with the introduction of technological tools and software, scheduling has become more efficient and streamlined. The use of automated scheduling systems has made it possible for organizations to schedule their operations more accurately and quickly, leading to an increase of productivity and efficiency.

The flow shop scheduling problem is an optimization problem that has received significant attention in the field of shop scheduling. This problem involves a set of machines that are arranged sequentially, with each job follows the same production route, starting at the first machine and ending at the last one. When the production lines are replicated, this leads to a more complex version of the problem, known as the distributed flow shop scheduling problem (DFSSP), or the parallel flow shop [12–14]. There are also several extensions to the basic flow shop scheduling problem. For example, when parallel machines are available in the same stage, this creates a hybrid or flexible flow shop problem [15–18]. When this type of problem is duplicated, it results in a new generation of scheduling problems known as the distributed hybrid flow shop [19–21]. Each of these different types of problems may require unique and adapted resolution methods to find optimal or near-optimal solutions. These methods may include heuristic algorithms, metaheuristics, mathematical programming and simulation techniques, among others. Finding the best approach to solve each specific problem is crucial to achieving an optimal outcome.

In this regard, there is encouragement for both researchers and manufacturers to develop models that can solve flow shop scheduling problem (FSP) and optimize the different criteria that are set by the industrial process. It is important to mention that there have been several models created in the past decades of the previous century to solve PFSP scheduling problems. The FSP is a complex problem that has been studied extensively in the literature. In order to develop effective scheduling algorithms, researchers have considered a variety of constraints, including SDST, SIST, no wait, no idle and blocking. SDST constraint, refers to the situation where the time required to change the machine setup from processing one job to another depends on the sequence of jobs being processed. Researchers often use this constraint to address various flow shop scheduling problems, for example, the authors in [22] suggests using adaptive hybrid genetic algorithms to address the PFSP scheduling problem with SDST constraint, with the ultimate goal of minimizing both the makespan and total weighted tardiness. In [23], a novel method is proposed to tackle the multi-objective PFSP scheduling problem with the constraint of SDST. This method, named MODABC/D, employs a multi-objective discrete artificial bee colony algorithm with a decomposition strategy. Its main objectives is to optimize the makespan and total flowtime of the scheduling problem. In the paper [24], two constructive heuristic algorithms are introduced to tackle the permutation scheduling problem in a flow shop with SDST constraint, with the aim of minimizing makespan. The first algorithm, named setup ranking algorithm, determines the sequence based solely on the job setup times. The second algorithm, FJSRA, utilizes the concept of fictitious jobs to obtain the sequence. Other researchers have worked on solving problems related to the FSP while considering the constraint of no-wait. This constraint indicates that jobs cannot wait between two consecutive machines. Consequently, the initial start time of a job at the first machine must be postponed to guarantee that it can complete the flow shop process without any delays caused by waiting for machines. For instance, in [25], a new approach to solving no-wait flow shop scheduling problems is introduced. The method utilized is a discrete differential evolution (DDE) algorithm which is specifically designed to optimize a bi-objective function comprising of two criteria: makespan and maximum tardiness. In [26], the primary goal is to present a solution strategy for the no-wait FSP scheduling problem that utilizes hybrid differential evolution (HDE) and gives priority to minimizing makespan. In [27], a novel hybrid ant colony algorithm is presented, which incorporates both crossover and mutation mechanisms to tackle the no-wait FSP scheduling problem. The objective was to minimize the maximum completion time. Another constraint, often used in manufacturing environments where the demand for efficient resource utilization is high, is the no-idle constraint, which refers to the requirement that once a machine begins processing a job it cannot remain idle until the job is completed. Here are some works that focus on addressing the FSP issues while taking into consideration this constraint: The authors in [28] introduce a hybrid discrete differential evolution (HDDE) algorithm designed to tackle the no-idle PFSP scheduling problem with the objective of minimizing makespan. The work in [29] describes an optimization algorithm, known as invasive weed optimization (IWO), which is designed to solve the no-idle flow shop scheduling problem (NFSP). The ultimate goal is to minimize the maximum completion time. The article [30] discusses the minimization of makespan problem in flow shops with no-idle constraint. The authors propose several polynomial time heuristics based on a geometrical approach, which provide asymptotically optimal solutions for an increasing number of jobs and provide a review of relevant results. The last constraint to be discussed in this section is the blocking, this occurs when a job cannot move forward to the next production stage because the necessary machine is not available. This situation happens when a machine is being used by a job that is not yet ready to move to the next stage, causing delays for subsequent jobs until the machine becomes available. Several studies have addressed the problem of flow shop scheduling under this constraint in the literature. In their work [31], the authors conduct an in-depth review of research on flow shop scheduling with blocking conditions from 1969 to 2019, revealing that makespan was the most frequently adopted objective function and that the majority of studies proposed heuristic or exact solution methods. The paper [32] proposes a bi-criteria optimization model for a PFSP scheduling problem that considers blocking and SDST constraints, aiming to minimize both makespan and total tardiness using a weighting coefficient. The authors suggest a MILP approach and multiple metaheuristic algorithms, such as genetic, iterated greedy, and iterative local search, to solve the problem. The work in [33] evaluates two variable neighborhood search (VNS) methods for optimizing the makespan of FSP scheduling problem with a blocking

constraint. The methods differ in their approach to changing neighborhoods during the improvement phase, with the first randomly switching between swap and insertion neighborhoods and the second starting in one neighborhood and then moving to the other.

We would like to draw attention to the fact that the scheduling literature contains a diverse optimization problems that are commonly denoted using the notation $\alpha \mid \beta \mid \gamma$ with three distinct fields. This notation was originally introduced in [34] and subsequently reviewed in [35]. Its purpose is to facilitate the identification of the problem's fundamental nature and defining characteristics. Specifically, the first field of the notation serves to indicate the problem's inherent nature, the second field represents the constraints that must be taken into account and the final field specifies the optimization criteria that will be used to evaluate potential solutions. In our case, the scheduling problem will be denoted by $Fm \mid$ permu, SIST $\mid T_{\max}$.

Minimizing the maximum tardiness ($T_{\max}$) becomes crucial in manufacturing industry since it allows manufacturers to reduce the lateness in the product delivery. This aims to satisfy customers in such way to have their commands completed in a desired time. The work in [36] seeks to address the optimization of a two-machine flow-shop problem in a batch delivery system, with the primary objective of minimizing both the maximum tardiness and the sum of delivery costs. To tackle this problem, the authors propose the utilization of a MILP approach and a branch-and-bound (B&B) algorithm. Moreover, they introduce an upper bound (UB) heuristic with a quick processing time. The authors in [37] have minimized the maximum lateness in a flow shop scheduling problem with release and due dates. They used the B&B algorithm as method of resolution. Likewise, in [38] we find the three optimization methods based on simulated annealing (SA), namely classical weighted SA (CWSA), normalized weighted SA (NWSA) and FSA (fuzzy simulated annealing) are used to minimize the maximum tardiness along with the makespan but in a no-wait two-stage flexible flow shop scheduling. The consideration of the SIST constraint has received significant attention in the field of operations research and scheduling literature. Numerous studies have been conducted to explore and address various scheduling problems by incorporating this constraint. For instance, Allahverdi adopted this constraint in his work [39], which addresses a two-machine flow shop problem with the objective of minimizing the mean flow time. Other researchers [7] have incorporated the SIST constraint into a PFSP scheduling problem with the objective of minimizing the maximum job completion time. Pranzo [40] tackled the problem of batch scheduling in a two-machine flow shop with limited buffer by incorporating the SIST constraint, with the goal of minimizing the makespan. The current work focuses on addressing the $Fm \mid$ permu, SIST $\mid T_{\max}$ scheduling problem, by using the MILP as exact method, HBJR and NEH as heuristics and ILS, IG and GA as metaheuristics in order to find the right sequence minimizing the maximum tardiness.

3. Problem formulation

In this section, we will outline the PFSP-SIST scheduling problem that we aim to solve in this paper. we recall that our optimization objective is to minimize the maximum tardiness $T_{\max}$ in $Fm \mid$ permu, SIST $\mid T_{\max}$ scheduling case. The PSFP-SIST problem is a well-known scheduling problem in which setup times are separated from jobs processing duration and are depending on type of machine. In this problem, we consider a set of jobs $N = \{J_1, J_2, \ldots, J_n\}$ which must be processed on a set of machines $M = \{M_1, M_2, \ldots, M_m\}$. All the jobs pass on all the machines in the same order $\{M_1, M_2, \ldots, M_m\}$, this is what we called, a flow shop with permutation, in Figure 1, we describe the process of he PSFP-SIST scheduling problem. A job can start on the last machine as soon as it finishes processing on the previous machine, all machines can only perform one operation at a time. The processing time of the job J_j on the machine M_i is given by $p_{j,i}$, setup time of the machine M_i is given by st_i, the due date is given by d_j and $C_{j,i}$ represents the end date of job processing J_j on machine M_i, When the machine is not specified, the end date is noted C_j. We denote by $\pi = \{\pi_1, \ldots, \pi_n\}$ a sequence of jobs, constituting a possible permutation when it is executed by all the machines.

One potential solution to the issue at hand entails the identification of an optimal job sequence that minimize the maximum tardiness $T_{\max}$. This objective function can be defined using the equation (1):

$$T_{\max} = \max\{T_{\pi_j}\}, \qquad j = 1, \ldots, n \tag{1}$$

TABLE 1. The processing times of the four jobs on the three machines.

Job	M_1	M_2	M_3	d_i
J_1	10	7	5	20
J_2	5	7	3	32
J_3	9	7	8	49
J_4	6	7	4	51
st_i	4	3	2	

where T_{π_j} represents the tardiness of the job π_j, which is defined as the time difference between completion and due date and can be calculated using the following formula:

$$T_{\pi_j} = \max\{C_{\pi_j} - d_{\pi_j}, 0\}. \tag{2}$$

It should be noted that when the completion time of a job is shorter than the due date, the job is reported completed early.

Below, we present a set of equations that enable the calculation of the completion time $C_{\pi_j,i}$ for each job π_j on machine i. These equations play a crucial role in determining the value of the objective function considered in this work, incorporating all the aforementioned assumptions.

$$C_{\pi_1,1} = p_{\pi_1,1} + st_1, \tag{3}$$

$$C_{\pi_j,1} = C_{\pi_{(j-1)},1} + p_{\pi_j,1} + st_1, \qquad j = 2,\dots,n \tag{4}$$

$$C_{\pi_1,i} = \max\{st_i, C_{\pi_1,(i-1)}\} + p_{\pi_1,i}, \qquad i = 2,\dots,m \tag{5}$$

$$C_{\pi_j,i} = \max\{C_{\pi_{(j-1)},i} + st_i, C_{\pi_j,(i-1)}\} + p_{\pi_j,i}, \qquad i = 2,\dots,m;\quad j = 2,\dots,n \tag{6}$$

$$T_{\max} = \max_{j=1,\dots,n}\{\max\{C_{\pi_j} - d_{\pi_j}, 0\}\}. \tag{7}$$

Equation (3) is used to determine the completion time of the first job on machine M_1. Equation (4) is used to determine the end date of the other jobs on machine M_1. Equation (5) is used to calculate the end date of the first job in the other machines. Equation (8) is used to calculate the completion time of the other jobs on the other machines. Equation (9) makes it possible to calculate the maximum tardiness $T_{\max}$.

To illustrate the $Fm \mid \text{permu, SIST} \mid T_{\max}$ scheduling problem, we provide an example of a workshop consisting of four jobs $\{J_1, J_2, J_3, J_4\}$ that need to be executed using three machines $\{M_1, M_2, M_3\}$. Table 1 shows the job processing time, due dates and machine setup times.

The Gantt chart for the sequence $\pi = \{1, 2, 3, 4\}$ is shown below in Figure 2, in this case, we can observe that each job's tardiness is recorded as follows: $T_1 = 26-20 = 6$, $T_2 = 34-32 = 2$, $T_3 = 51-49 = 2$, $T_4 = 57-51 = 6$ which implies that the value of the maximum tardiness is equal to 6.

From this numerical example and according to the Gantt chart, we notice that the four jobs were executed on the first machine in a continuous way. On the other hand, the other two machines remain on standby during the execution of the jobs, which leads to a loss of time and consequently a failure to respect the promised deadlines.

4. Resolution of problem

In this section, we propose three resolution approaches aimed at addressing the PFSP-SIST problem with the objective of minimizing the maximum tardiness $T_{\max}$. The first approach involves formulating a mathematical model referred to the MILP. Additionally, we present a set of heuristics as the second approach, including a HBJR and an NEH heuristics. Lastly, the third approach encompasses a set of metaheuristics, including the ILS, IG and GA metaheuristic.

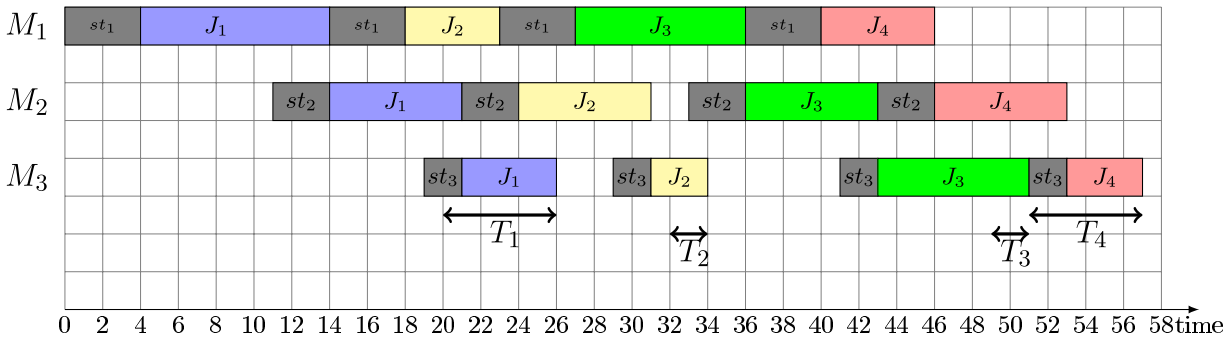

FIGURE 2. The Gantt chart for four jobs and three machines.

4.1. Mixed-integer linear programming model

Several researchers have utilized linear programming models to solve various problems in the field of scheduling [41–43]. In the same context, many works have proposed MILP models as exact methods for solving PFSP scheduling issues [44, 45]. This kind of model consists in minimizing an objective function under the various constraints relating to the industrial problem considered.

We adopt and customize this model to tackle the PFSP-SIST problem investigated in this research paper. We consider a set of jobs $\{J_1, J_2, \ldots, J_n\}$ that must be processed on a set of machines $\{M_1, M_2, \ldots, M_m\}$ and we define the set of parameters and variables that are required to formulate our model.

$$y_{k,j} = \begin{cases} 1, & \text{if the job } j \text{ is assigned in the position } k \text{ in the sequence} \\ 0, & \text{otherwise,} \end{cases} \qquad j,k = 1,\ldots,n,$$

$$C_{k,i} = \text{the end date of job processing in position } k \text{ on machine } i, \qquad k = 1,\ldots,n, \quad i = 1,\ldots,m. \tag{8}$$

The objective is to:

$$\text{Minimize } T_{\max}. \tag{9}$$

Under constraints

$$\sum_{j=1}^{n} y_{k,j} = 1; \qquad k = 1,\ldots,n, \tag{10}$$

$$\sum_{k=1}^{n} y_{k,j} = 1; \qquad j = 1,\ldots,n, \tag{11}$$

$$C_{1,1} \geq st_1 + \sum_{j=1}^{n} y_{j,1} \times p_{1,1}, \tag{12}$$

$$C_{k,1} \geq st_1 + C_{(k-1),1} + \sum_{j=1}^{n} y_{k,j} \times p_{j,1}; \qquad k = 2,\ldots,n \tag{13}$$

$$C_{k,i} \geq C_{k,(i-1)} + \sum_{j=1}^{n} y_{k,j} \times p_{j,i}; \qquad i = 2,\ldots,m; \quad k = 1,\ldots,n \tag{14}$$

$$C_{k,i} \geq st_i + C_{(k-1),i} + \sum_{j=1}^{n} y_{k,j} \times p_{j,i}; \qquad i = 1,\ldots,m; \quad k = 1,\ldots,n \tag{15}$$

$$T_k \geq C_{k,m} - d_k; \qquad k = 1,\ldots,n \tag{16}$$

$$T_{\max} \geq T_k; \qquad k = 1, \ldots, n \tag{17}$$
$$C_{k,i} \geq 0; \qquad i = 1, \ldots, m; \quad k = 1, \ldots, n \tag{18}$$
$$T_k \geq 0; \qquad k = 1, \ldots, n \tag{19}$$
$$y_{k,j} \in \{0, 1\}; \qquad j, k = 1, \ldots, n. \tag{20}$$

The equation (9) represents the objective function of our problem, which consists in minimizing the maximum tardiness, denoted $T_{\max}$. The constraints (10) and (11) make it possible to guarantee that each position of the sequence will be occupied by a single job and that each job can have only one position. The constraint (12) is used to determine the end date of the job occupying the first position in the first machine. The constraint (13) calculates the end date of the job at position $k, k \geq 2$ in the first machine. The (14) constraint guarantees that for a job at position k, the end date in machine i is greater than or equal to the end date in the previous $i-1$ increased its processing time on the machine i. The constraint (15) ensures that for two jobs of successive positions, the end date in the machine of the one in position k is greater than or equal to that in position $k-1$ plus the processing time and the preparation time. The constraint (16) identifies the delay of each job k, $k = 1, \ldots, n$ on the last machine m. The (17) constraint is used to determine the maximum tardiness criterion. The constraints (18)–(20) guarantee that the end date of execution and the tardiness of each job on the last machine m are positive and that the assignment variables $y_{k,i}$ are binary variables.

To illustrate our problem, we consider the same example treated above, where we considered a permutation flow shop problem with four jobs and three machines. The processing time and the due date of each job as well as the preparation time of each machine are presented in Table 1. We note that the MILP has been implemented using the LINGO 18 software on a PC with an AMD Ryzen 5 3500U processor, 2.10 GHz and 8 GB of RAM. Running our model, we find that the exact job sequence is $\pi = \{3, 4, 2, 1\}$ and the optimal value of the maximum tardiness we want to minimize is equal to 6. We note that this example is only to illustrate the MILP model, it concerns a small size instance. However, for medium and relatively large instances, we have reserved it for numerical simulation purposes.

4.2. Heuristics

While there are several exact methods available to address PFSP problems, they may not always be feasible in industrial reality due to the nature of flows and the complex arrangement of productive resources. In this context, the use of heuristic methods represents a promising alternative for solving different optimization problems, also representing a simple less expensive path that ensures good production and that allows to respect the due dates promised by customers. In this section, we present two heuristic methods, the first is a heuristic based on Johnson's rule. The second is a heuristic based on the NEH algorithm.

4.2.1. Heuristic based on Johnson's rule

Johnson [2] considers a two-machine flow-shop problem, the goal is to find an optimal sequence to optimize the makespan. He proved a very important theorem, stating that a job j must precede a job j' in an optimal sequence if $\min\{p_{j,1}, p_{j',2}\} \leq \min\{p_{J',1}, p_{j,2}\}$, with $p_{j,i}$ and $p_{j',i}$ are respectively the processing times of jobs j and j' in machine $i \in \{1, 2\}$. Based on this theorem, Johnson's famous algorithm was deduced. Several works that have been proposed since the first study refer to Johnson's algorithm. The general idea of this algorithm can be expressed as follows, the jobs having the smallest processing times on the first machine will be executed first; those with the smallest processing times on the second machine will be executed at the end.

In this article, we are interested in solving a permutation flow shop problem under the constraint of the setup time independent of the sequence. The objective is to minimize the maximum tardiness criterion, to do this, we propose a heuristic based on the modified Johnson algorithm. The Algorithm 1 presents the adopted heuristic while taking into account the characteristics of our problem.

In this heuristic, we propose the modified Johnson algorithm. The idea is to build two virtual machines, by combining the first i machines where $1 < i < (m-1)$ into a single virtual machine and the remaining $(m-i)$

machines into the second virtual machine. The processing time of a job π_j on the first virtual machine will be $p'_{\pi_j,1}$ which is equal to the sum of the processing times of the job π_j on the machines forming the virtual machine and their preparation times st_i.

Algorithm 1. Pseudo code of Johnson's rule-based heuristics.

Inputs: $\pi^0 = \{\pi_1, \pi_2, \ldots, \pi_n\}$,% The initial sequence %
1: $\pi^* \leftarrow \pi^0$
2: Each machine of $M = \{M_1, M_2, \ldots, M_m\}$ has a setup time st_i.
3: **for** $i = 1$ to $m - 1$ **do**
4: $p'_{\pi_j,1} \leftarrow \sum_{k=1}^{i}(p_{\pi_j,k} + st_k)$, $j = 1, \ldots, n$; $p'_{\pi_j,2} \leftarrow \sum_{k=i+1}^{m}(p_{\pi_j,k} + st_k)$, $j = 1, \ldots, n$
5: $\pi^* \leftarrow$ Johnson's Rule(π^*)
6: $T_{\max} \leftarrow T_{\max}(p'_{\pi_j}, st')$
7: At each iteration, we retain the sequence π^* and the value of $T_{\max}$.
8: **end for**
9: Retain the best sequence π^* which gives a minimum $T^*_{\max}$.
Outputs: π^*, $T^*_{\max}$

4.2.2. Heuristic based on the NEH algorithm

After the rule proposed by Johnson for solving a flow shop problem of two and three machines, several algorithms have been proposed for scheduling a sequence of n jobs on m machines in order to optimize a given criterion in the industrial context. In comparison with existing heuristics, Nawaz *et al.* [46] propose an algorithm generates best sequences which minimize the makespan criterion. This algorithm, known as the NEH algorithm, suggests that jobs with high processing time should be prioritized over jobs with lower processing time. Many research works have adopted heuristics based on the NEH algorithm [47, 48] to obtain the best sequences that minimize the various criteria to be optimized adopted. On the other hand, several works have been the subject of comparisons of the NEH algorithm with modern complex heuristics, Turner and Booth [49] shows that NEH has proven to be more efficient than other heuristics, Ruiz and Maroto [50] concluded that this heuristic algorithm is the best in terms of PFSP problem solving performance.

The Algorithm 2 describes the steps necessary to obtain the sequence of NEH, to adapt the algorithm of NEH to our problem, the setup time of the machines has been added to the sum of the processing times of each sequence job, on the first step of the algorithm, we calculate $T_j = \sum_{i=1}^{m}(p_{j,i} + st_i)$. In the second step, we rank the jobs in decreasing order of T_j. We choose the best sequence of the first two jobs $(\pi_1\pi_2, \pi_2\pi_1)$ which presents an $T_{\max}$ minimum. Subsequently, at each iteration, the job π_j, $(j = 3, \ldots, n)$ is inserted in the different positions of the constructed sequence. We will retain the best sequence π_{NEH}, at each iteration, the process is repeated until all the jobs are completed.

Algorithm 2. Pseudo code of the NEH algorithm.

Inputs: $\pi^0 = \{\pi_1, \pi_2, \ldots, \pi_n\}$,% The initial sequence %
1: Each machine of $M = \{M_1, M_2, \ldots, M_m\}$ has a setup time st_i.
2: For each job, we calculate the sum of the processing time and the preparation time: $T_j = \sum_{i=1}^{m}(p_{j,i} + st_i)$
3: Build the sequence $\pi = \{\pi_1, \pi_2, \ldots, \pi_n\}$ by sorting the jobs in descending order of T_j
4: Retain the best sequence of π_1 and π_2 which minimizes $T_{\max}$
5: **for** $j = 3$ to n **do**
6: Insert each job π_j in the different positions of the current sequence and retain the best sequence π^* which gives a minimum $T^*_{\max}$.
7: **end for**
Outputs: π^*, $T^*_{\max}$

4.3. Metaheuristics

Metaheuristics are high performance methods used to solve complex optimization problems. Generally, they aim to improve an initial solution that was generated by a heuristic. In our case, we concentrate on metaheuristics based on the local search concept and the given current solution is constructed by the NEH algorithm discussed in the previous section. We've chosen three metaheuristics which are some of the best utilized methods in the literature. We'll start with two metaheuristics based on local search concept, the ILS algorithm and the IG algorithm then we move on to implement an inspired nature metaheuristic represented by GA.

4.3.1. Iterated local search

The ILS algorithm is considered among the best-known metaheuristics, theoretically simple and efficient.The central idea of ILS algorithm is that the search for the solution should focus on a neighborhood set of the current solution and retain the best value of the objective function by comparing with the current best value of this function. This process must be repeated until the stopping condition is satisfied. Numerous researchers have applied this metaheuristic approach to address various optimization problems, with a particular focus on the PFSP issues. As far as we are aware, Stützle [51] is the first research that investigated how ILS can be used to solve the PFSP scheduling problem. The authors in [52] examined some difficulties in implementing ILS algorithms for the permutation flow-shop problem. In the local search phase of the ILS algorithm, M'hallah [53] used a variable neighborhood descent. Along the same lines, another study by Dong *et al.* [54] proposed the use of an ILS algorithm with multiple restarts for the resolution of a flow shop problem with permutation, the results obtained demonstrate a better performance when compared to other metaheuristics, despite the approach's simplicity. Finally, an ILS algorithm is presented in [55], the choices of initial approach and perturbation strength are discussed in depth, the results showed that the proposed algorithm is more efficient in comparison with other adopted approaches. Algorithm 3 will be used to illustrate the required steps to develop this metaheuristic, while considering the PFSP-SIST problem. The three major components of the proposed ILS metaheuristic are as follows:

- Generate initial solution: the use of metaheuristics to solve optimization issues, as is well known, necessitates the determination of an initial solution. Due to its key importance in the performance of the ILS metaheuristic algorithm, the choice of the method that generates the starting sequence is an important issue. Because random answers are often of poor quality, it is always desirable to mobilize high-quality solutions. To achieve this, we used the NEH heuristic to obtain an initial solution for our ILS algorithm.
- Local search procedure: local search is a method that involves a neighborhood structure to construct a new enhanced sequence from an initial permutation. There are three basic neighborhood exploration strategies described in the literature: swap moves which consists of swaps of two neighboring jobs at positions i and j, interchange-moves is based on the exchange of jobs at the i-th and j-th positions and insertion-moves the idea behind this method is to remove a job from the i-th position and replacing it in the j-th position.
  For our metaheuristic ILS, we have chosen the neighborhood insertion method as a neighborhood search procedure. The principle of this method is simple, consists of eliminating a job from position i and inserting it into position j while avoiding sequence redundancy. it allows to generate $(n-1)^2$ neighbor permutation with n represent the size of the initial sequence.
  In order to understand the reasoning behind this procedure, an illustrative example is presented as follows: let π^0 be an initial sequence made up of four jobs $\pi^0 = [1, 2, 3, 4]$ with $T_{\max} = 14$ is the value relative to the maximum tardiness corresponding to the sequence π^0. The objective is to identify the sequence within the neighborhood of the initial sequence π^0 that effectively minimizes the objective function. Given that π^0 represents a sequence of four jobs, it generates nine neighboring permutations. In Figure 3, the neighbor permutations resulting from the insertion neighborhood approach are presented, along with their corresponding $T_{\max}$ values. According to the schema below, it is evident that the local search (LS) procedure will return sequence $\pi^{\text{best}} = [2, 3, 4, 1]$ as the output since it displays the lowest $T_{\max}$ value which is equal to 8.

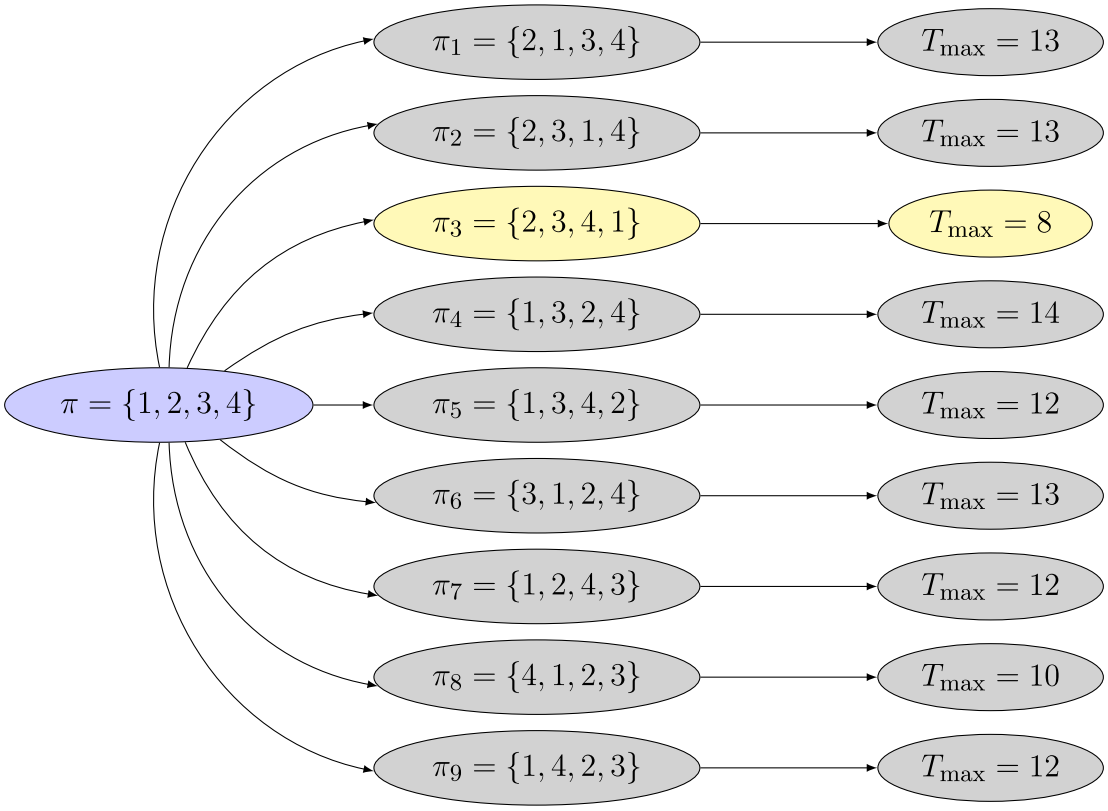

FIGURE 3. Illustration of insertion neighborhood procedure.

– Acceptance criterion: this procedure decides whether the solution obtained as the new current solution is approved or not. In the literature, various strategies have been explored for the benefit of the acceptance procedure. We opted for the choice of a simulated annealing type acceptance criterion, based on the Metropolis condition. This criterion consists in always accepting the new solution obtained if it is better than the initial solution, otherwise, this solution will be accepted if it satisfies the following Metropolis condition:

$$\text{random} \leq \exp\left\{-\left(\frac{T_{\max}(\pi'') - T_{\max}(\pi_0)}{T}\right)\right\} \tag{21}$$

T called temperature depended on the processing times, setup times, the number of jobs and machines, and another adjustable parameter T_0:

$$T = T_0 \times \frac{\sum_{j=1}^{n}\sum_{i=1}^{m} p_{j,i} + st_i}{10 \times n \times m} \tag{22}$$

and random is a random number between $[0, 1]$.

Algorithm 3. Pseudo code of the iterated local search.

Inputs: $\pi^0 = \{\pi_1, \pi_2, \ldots, \pi_n\}$, % The initiale sequence %
$\pi^0 =$ NEH_Heuristic
$\pi^{\text{best}} = \pi^0$;
while {the termination criteria not achieved} **do**
  $\pi' =$ Local_Search$\{\pi^0\}$;
  % **Acceptance Criterion**%
  if $T_{\max}(\pi') < T_{\max}(\pi^0)$ **then**
    $\pi^0 = \pi'$;
    if $T_{\max}(\pi^0) < T_{\max}(\pi^{\text{best}})$ **then**
      $\pi^{\text{best}} = \pi^0$;
    end if
  else if random $\leq \exp\{-(T_{\max}(\pi') - T_{\max}(\pi^{\text{best}}))/T\}$ **then**
    $\pi^{\text{best}} = \pi'$;
  end if
end while
Outputs: π^{best}

4.3.2. Iterated greedy

The second approach mobilized to solve our problem efficiently is the IG metaheuristic. The standard version of this used algorithm as described in [56] is based on the following principle: The IG algorithm begins with an initial solution, which is typically generated using a heuristic approach. The algorithm then proceeds through two main stages known as the destruction and construction phases. During the destruction phase, a predetermined number of jobs is removed from the initial sequence through a uniform random selection process. These jobs are subsequently re-inserted in the same order as they were removed during the construction phase. It is important to highlight that the insertion of jobs was carried out according to the principle of the NEH algorithm. The IG metaheuristic, along with its various extensions, has been widely used in the literature to optimize an objective functions or even more in the context of scheduling problems. Numerous researchers have adopted this approach and achieved promising results. For instance, Ruiz and Stützle [56] were the first to implement this metaheuristic for a flow shop problem with permutation and the results demonstrate that the IG algorithm performs well. In [57] the authors developed a metaheuristic IG algorithm addressing a two-machine flow shop scheduling problem with urgent jobs and limited waiting times, the efficiency of IG was verified against MIP. The work in [58] propose two new IG algorithms for a SDST flow shop problem, despite its simplicity, the IG algorithms demonstrated a good performance for the objective functions to be minimized.

In order to apply the IG algorithm to the PFSP-SIST problem, we will outline the complete process and describe the main steps required to generate an optimal sequence, the pseudo code 4 provides a detailed description of all these steps. Similar to the ILS metaheuristic algorithm mentioned earlier, the initial job sequence for this metaheuristic is obtained using the NEH heuristic. Then the algorithm proceed to the destruction step where two sequences π_R and π_D will be constructed by eliminating d jobs chosen at random and without repetition from the initial sequence. We note that the sequence π_R of size d represents the sequence formed by the jobs that have been deleted in the same order as they were removed and π_D is the sequence of size $n - d$ composed by the jobs of the initial sequence excluding the removed jobs. Then, during the construction step, each element of π_R is reinserted into the π_D sequence using the same NEH heuristic principle. Which means that the first job of π_R is inserted in the $n - d + 1$ possible positions of the sequence π_D, in order to generate $n - d + 1$ sub sequence which includes the job $\pi_R(1)$. The sequence with the smallest objective function value will be selected for the next iteration. The operation will stop, when there are no jobs in the sequence π_R and the sequence π_D return to the size n again. It should be emphasized that in our specific case, we have incorporated a local search procedure, called the insertion neighborhood, after the destruction and construction phases of the adopted IG

algorithm. This additional local search procedure is aimed at improving the performance of the algorithm and achieving more optimal solutions for our particular problem.

At every iteration of this algorithm, it is necessary to evaluate the obtained sequence π'' by comparing it to the current best solution π^0. In other words, if $T_{\max}(\pi'') < T_{\max}(\pi^0)$, π'' replaces π^0. Otherwise, we propose using the concept of simulated annealing acceptance, which has been employed in a number of studies. The replacement is accepted if the following condition is met.

$$\text{random} \leq \exp\left\{-\left(\frac{T_{\max}(\pi'') - T_{\max}(\pi^0)}{T}\right)\right\}, \tag{23}$$

where **T** named temperature depended on the processing times, setup times, the number of jobs and machines, and another adjustable parameter T_0:

$$T = T_0 \times \frac{\sum_{j=1}^{n}\sum_{i=1}^{m} p_{j,i} + st_i}{10 \times n \times m} \tag{24}$$

and random is a random number between $[0, 1]$.

Algorithm 4. Pseudo code of the iterated greedy algorithm with local search algorithm.

Inputs: $\pi^0 = \{\pi_1, \pi_2, \ldots, \pi_n\}$, % The starting sequence obtained by the NEH heuristic %
$\pi^{\text{best}} = \pi^0$;
while the termination criteria not achieved **do**
  $\pi' = \pi^0$; % ***Destruction phase*** %
  for $u = 1$ to d **do**
    eliminate one job at random from π' and insert in π_R;
  end for
  % ***Construction phase***%
  for $u = 1$ to d **do**
    retain the best permutation obtained by inserting job $\pi_R(i)$ in all possible positions of π';
  end for
  $\pi'' = LocalSearch\_Inserion(\pi')$; % ***Local Search process***%
  % ***Acceptance Criterion***%
  if $T_{\max}(\pi'') < T_{\max}(\pi^0)$ **then**
    $\pi^0 = \pi''$;
    if $T_{\max}(\pi^0) < T_{\max}(\pi^{\text{best}})$ **then**
      $\pi^{\text{best}} = \pi^0$;
    end if
  else if random $\leq \exp\{-(T_{\max}(\pi'') - T_{\max}(\pi^{\text{best}}))/T\}$ **then**
    $\pi^{\text{best}} = \pi''$;
  end if
end while
Outputs: π^{best}

4.3.3. Genetic algorithm

In this paper, the genetic meta-heuristic (GA) algorithm was utilized as the last approach to enhance the performance of an ineffective initial solution. The GA meta-heuristic algorithm belongs to a category of algorithms that are inspired by the principles of evolution theory. It is a powerful and widely used method for solving complex optimization problems, including the PFSP. In the process of the GA, the individuals possessing the most desirable fitness function are selected for breeding to produce offspring for the succeeding generation. The principle of this algorithm is relatively simple, Algorithm 5 provides a detailed description of the main steps

required to implement it. These steps are executed until the designated stopping criteria are met. The main phases of this meta-heuristic are selection, crossover and mutation. These phases and the other parameters of the developed GA meta-heuristic can be defined as follows:

- Initial population: we employ the NEH heuristic to generate the first member of this population, while the remaining $N-1$ members are generated randomly. Regarding the size of this population, a larger population allows exploring a broader solution space but may require more computational resources. It is advisable to start with a moderate population size and adjust it based on the complexity of the studied scheduling problem. In our case, we established the initial population size at $N = 50$.
- Fitness function: the adopted fitness function (denoted as $f(k)$), can be defined by the following equation: $f(k) = 2 \times \frac{k}{N}, k \in \{1, \ldots, N\}$ [59]. Here, N represents the population size, while k represents the position of each individual in the population, sorted in decreasing order in terms of the maximum tardiness value $T_{\max}$. In other words, the position of the individual with the highest $T_{\max}$ value is 1, while the position of the individual with the lowest $T_{\max}$ value is N.
- Selection phase: this phase aims to maintain high-quality individuals while eliminating the less performing ones in each generation. The selection of high-quality individuals is determined by a selection probability defined as: $p(k) = \frac{f(k)}{\sum_{k=1}^{N} f(k)}, k \in \{1, \ldots, N\}$. This means that the individual with a better fitness function (a higher ranking) has a higher probability of being selected than the individual with the lowest fitness function value.
- Crossover phase: in this phase, we have chosen two-points crossover method. As it is demonstrated in Figure 4, the principle of this method is very simple. First, two crossover points are chosen randomly noted pt_1 and pt_2. Then, the first children, referred as C_1, is generated by taking a section of the code from the first parent P_1 (from the beginning to the pt_1 and from the pt_2 to the end of the sequence P_1). Any missing jobs within the C_1 sequence are then completed using jobs from the parent P_2. A similar procedure is followed for generating the second child C_2, this ensures us the solution feasibility.
- Mutation operator: the adopted method in this phase consists on the following principle, a job is randomly chosen from position k and placed into a randomly selected position k' within the current sequence. If k is less than k', all jobs between positions k and k' are shifted to the left; otherwise, they are shifted to the right. Figure 4 illustrates an example of the application of this approach.
- Crossover and mutation probabilities: concerning the probabilities of crossover and mutation, we have explored a range of values. Specifically, we have tested the following crossover probabilities: $p_c = 1.0, 0.8, 0.7$. Additionally, we have investigated different mutation probabilities, including $p_m = 0.01, 0.05, 0.1, 0.15$, as reported in the existing literature [59–63]. It should be noted that we have chosen the values $p_c = 0.8$ for the crossover probability and $p_m = 0.05$ for the mutation probability because they yielded the best results.

The GA metaheuristic is considered among the most often used methods for solving scheduling optimization problems. As far as we know, the application of GA metaheuristic for minimizing makespan in PFSP was first proposed in [64]. In this work, the author compared the performance of this algorithm to that of a simple neighborhood search method. The GA metaheuristic and its extensions has been used in a number of works to solve different optimization problems. A comparative analysis was carried out in [65] between the genetic algorithm against other search methods such as local search, taboo search and simulated annealing revealed that the genetic algorithm performed slightly worse. As a result, the authors investigated two hybrid algorithms: genetic local search and genetic simulated annealing, in order to enhance the performance of the genetic algorithm. In [66], an effective hybrid genetic algorithm (HGA) is suggested for a flow shop scheduling problem with a limited buffer. Wang *et al.* [67] proposed an GA-TVNS, a metaheuristic that improves the exploration and exploitation of the search space by combining GA and variable neighborhood search (VNS). The authors in [68], used the GA metaheuristic along with the local search algorithm to solve the problem of scheduling a set of preventive maintenance jobs to be executed on a set of machines. Their goal was to optimize the total preventive maintenance cost. Based on the computational findings, a significant proportion of solutions generated by the GA metaheuristic exhibited superior performance compared to those produced by the local search algorithm.

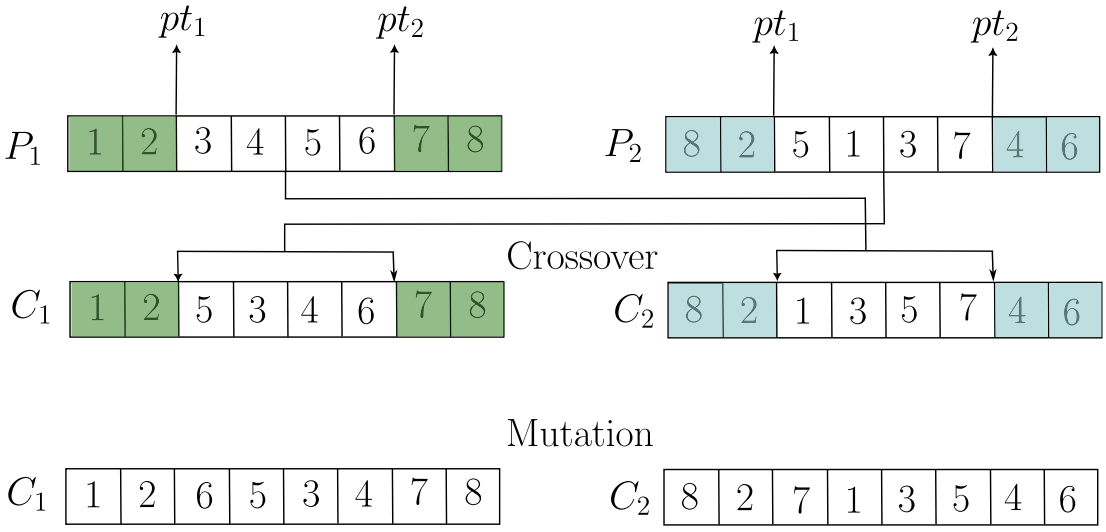

FIGURE 4. Example of the crossover and mutation approaches.

Algorithm 5. Pseudo code of genetic algorithm.

1: Generate an initial population P_{pop} of solutions to start with.
2: Retain the best sequence π_{best} from the initial population that minimizes $T_{\max}$.
3: **while** {the termination criteria not achieved} **do**
4: Apply the selection method to get the two parent sequences P_1 and P_2.
5: Apply the crossover procedure and retain the two children C_1 and C_2.
6: Apply the mutation method on the two children C_1 and C_2.
7: Evaluate the generated individuals and determine the new π_{best}
8: Update the initial population P and return to the selection step.
9: **end while**
Outputs: π^{best}

5. Numerical simulations

In this section, the performance of the proposed algorithms is evaluated and examined through a comparative study on randomly generated problem instances. This study is divided into two parts. On the one hand, a study dedicated to the comparison between the proposed heuristics and the exact method, which in our case design by the MILP method. On the other hand, a comparison between the three implemented metaheuristics.

We note that to generate the problem instances, processing times, setup times and due dates were generated, as they are widely used in the literature. In fact, we distinguish two categories of instances for our numerical simulation, the category of small and medium instances and the category of relatively large instances. Indeed, the set of small and medium instances is composed of the combinations of jobs and machines ($n \times m$) where the number of jobs $n \in \{10, \ldots, 90\}$ and the number of machines $m \in \{5, \ldots, 10\}$ with $p_{j,i}$ the processing time of each job in each machine is uniformly distributed in $[1; 49]$ and st_i is the setup time of each machine is also generated uniformly distributed in $[1, 10]$. For the set of relatively large instances, the number of jobs n varies in $\{100, \ldots, 400\}$ and the number of machine varies in $\{10, \ldots, 30\}$ with $p_{j,i} \in [50, 99]$ and $st_i \in [10, 20]$. For the due dates we adopt the formula most used in the literature, the expression of this formula is as

TABLE 2. Comparison between the MILP, NEH and HBJR algorithms in terms of the values of the maximum tardiness, the CPU time and the RPD indicator.

Instance	MILP		NEH		HJBR		RPD_{NEH}	RPD_{HBJR}
$n \times m$	T_{max}	CPU time	T_{max}	CPU time	T_{max}	CPU time		
4×2	4	1.76	4	0.0115	13	0.0054	0.000	2.250
4×3	11	0.50	11	0.0022	29	0.0076	0.000	1.636
4×4	15	1.76	15	0.0074	51	0.0029	0.000	2.400
4×5	23	1.15	23	0.0038	65	0.0026	0.000	1.826
4×6	30	0.96	30	0.0021	93	0.0030	0.000	2.100
4×7	42	1.20	43	0.0022	125	0.0036	0.024	1.976
4×8	45	1.33	45	0.0024	145	0.0026	0.000	2.222
4×9	50	1.49	50	0.0022	181	0.0026	0.000	2.620
4×10	53	1.78	55	0.0022	215	0.0031	0.038	3.057
8×2	20	34.68	20	0.0030	22	0.0026	0.000	0.100
8×3	27	56.88	27	0.0024	57	0.0029	0.000	1.111
8×4	33	23.11	33	0.0022	86	0.0026	0.000	1.606
8×5	47	25.01	47	0.0022	115	0.0031	0.000	1.447
8×6	57	28.39	58	0.0038	173	0.0026	0.018	2.035
8×7	64	44.41	70	0.0023	217	0.0026	0.094	2.391
8×8	72	73.99	75	0.0022	252	0.0029	0.042	2.500
8×9	78	28.17	82	0.0006	331	0.0026	0.051	3.244
8×10	92	37.85	96	0.0022	370	0.003	0.043	3.022
12×2	33	1895.40	34	0.0005	54	0.0026	0.030	0.636
12×3	43	1185.39	44	0.0023	113	0.0026	0.023	1.628
12×4	60	3283.77	60	0.0022	158	0.0027	0.000	1.633
12×5	74	1190.07	74	0.0022	194	0.0026	0.000	1.622
12×6	95	1066.30	95	0.0022	284	0.0028	0.000	1.989
12×7	98	1172.52	102	0.0035	347	0.0026	0.041	2.541
12×8	107	1356.50	111	0.0033	398	0.0027	0.037	2.720
12×9	134	1252.20	134	0.0022	522	0.0026	0.000	2.896
12×10	143	3486.88	143	0.0025	590	0.0025	0.000	3.126

follows:

$$d_j = \rho \times \left[\sum_{i=1}^{m} (p_{j,i} + st_i)(1 + \text{rand}()) \right]. \tag{25}$$

5.1. Comparison between the heuristics

The first part of this comparative study looked at the evaluation of the performance of the adopted algorithms through a comparative study between NEH, Johnson's rule modified and the exact method designated by the MILP method. This comparaison is performed using a usual performance measure which is known as relative percentage of deviation (RPD). The following formula is used to calculate RPD:

$$\text{RPD}_{H_i} = \left[\frac{T_{\max}^{H_i} - T_{\max}^{\text{best}}}{T_{\max}^{\text{best}}} \right] \times 100; \quad i \in \{1, 2\} \tag{26}$$

where H_1 represents the heuristic based on the modified Johnson algorithm, while H_2 indicates the NEH heuristic. It's worthy to note that $T_{\max}^{H_i}$ represents the solution obtained by the heuristic H_i and $T_{\max}^{\text{best}}$ represents the solution found from the optimal solution generated by the MILP.

Table 2 shows the RPD_{H_i} obtained for different $(m \times n)$ combinations, where m represent the number of machines and n the number of jobs. A first reading of the this table shows that the results obtained by the

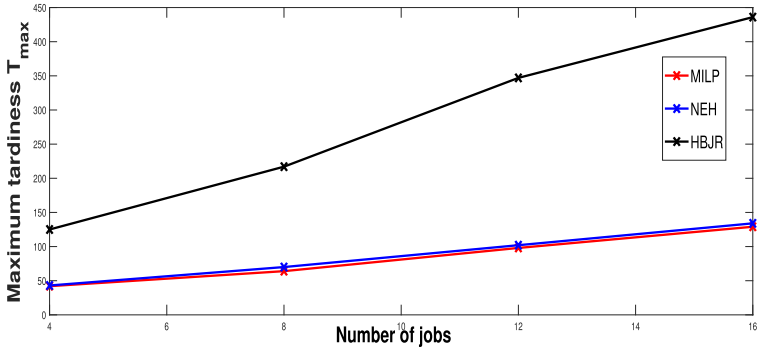

FIGURE 5. The maximum tardiness's values in the case of $m = 7$.

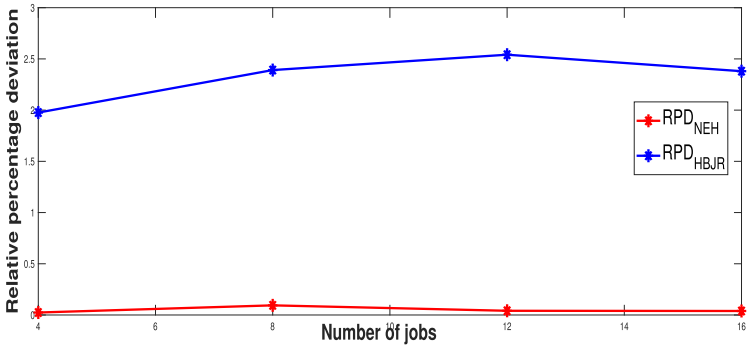

FIGURE 6. The Relative percentage deviation's values in the case of $m = 7$.

NEH heuristic are extremely close to those obtained by the MILP approach. On the other hand, the difference between the values of $T_{\max}$ calculated by the modified Johnson rule and the results obtained by the MILP is significant.

In addition, Table 2 presents the amount of time that each algorithm spends processing a specific instance (CPU time) in units of seconds. The results indicate that the CPU time of MILP increases as the instances become larger, with a significant increase occurring from instance 12×2 onwards. In contrast, NEH and HBJR, both heuristic algorithms, do not require a substantial amount of CPU time. One can conclude that MILP is effecient only for small instance sizes in terms of seeking the optimal solution.

For a combinations $(n \times m)$ with $n \in \{4, 8, 12, 16\}$ and the number of machine m is equal to 7, we calculated the value of the maximum tardiness using the three approaches MILP, NEH, and HBJR. The results are shown in Figure 5. The value of $T_{\max}$ clearly grows with the job values, and the graph also demonstrates the NEH's heuristic algorithm's good performance as compared to modified Johnson's rule outcomes.

In Figure 6 we can see clearly the large difference between the the relative percentage deviation given by NEH and RPD given by the HBJR. This result has exactly confirmed the performance of the NEH algorithm illustrated above. As a result, NEH can be used as an efficient approach to solve our scheduling problem.

Now, we consider the case where the number of machines varies, $m = \{2, \ldots, 10\}$ and the number of jobs remain constant, $n = 8$. The maximum tardiness $T_{\max}$ values are depicted in Figure 7. As we have already noticed, $T_{\max}$ is an increasing function of the number of machines. This means that the value of $T_{\max}$ increases when the number of machines increases. Furthermore, as compared to the adopted HBJR heuristic, the NEH algorithm has a considerable performance advantage.

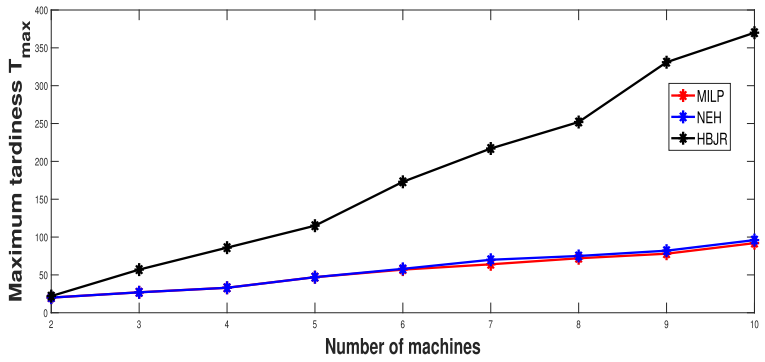

FIGURE 7. The maximum tardiness's values in the case of $n = 8$.

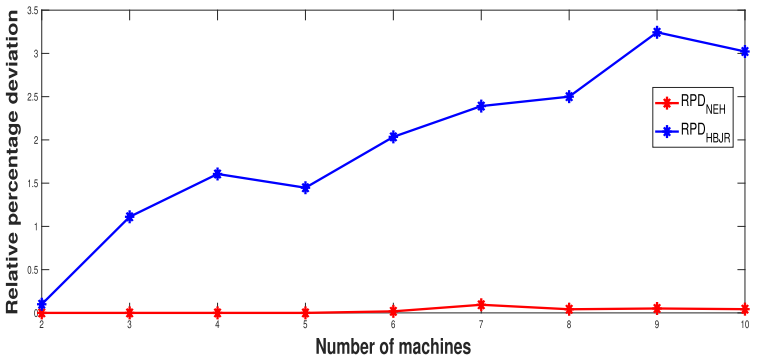

FIGURE 8. The maximum tardiness's values in the case of $n = 8$.

The graphic in Figure 8 validates the performance of NEH heuristic. Its clear that the $\mathrm{RPD}_{\mathrm{HBJR}}$ given by the HBJR is extremely large than the $\mathrm{RPD}_{\mathrm{NEH}}$ obtained by NEH algorithm. Which implies the efficiency of this last approach for solving the permutation flow shop problems.

5.2. Comparison between the metaheuristics

To evaluate the performance of the proposed metaheuristics, we suggest in this part to calculate the RPI (relative percentage increase) indicator to measure the deviation from the best solution given by the three metaheuristics IG, ILS and GA.

The RPI is calculated using the following formula:

$$\mathrm{RPI} = \left[\frac{T_{\max} - T_{\max}^{\mathrm{best}}}{T_{\max}^{\mathrm{best}}}\right] \times 100. \tag{27}$$

We highlight that computing is given for an average of five instances for each algorithm.

In Table 3, we present the values of the RPI obtained for the different combinations $(n \times m)$ with m equal to 20 and $n \in \{10, 20, 30, \ldots, 90\}$. As previously stated, the RPI is an indicator that measures the convergence of $T_{\max}$ value towards the best solution among the solutions given by the three metaheuristics used. For all the evaluated instances, the IG algorithm displays small values of the RPI compared to the two other metaheuristics ILS and GA, this implies the good performance of the IG metaheuristic compared to the two other metaheuristics used.

In percentage points, Figure 9 illustrates the difference between the three metaheuristics, this difference varies between 4.216 percentage points and 32.485 percentage points. These results have just confirmed the

TABLE 3. A comparison between the RPI indicator values for the three meta-heuristics IG, ILS and GA for medium instances.

$n \times m$	RPI_{IG}	RPI_{ILS}	RPI_{GA}
10×10	2.331	2.819	2.365
20×10	2.941	3.160	3.168
30×10	0.368	0.411	0.412
40×10	1.174	1.206	1.274
50×10	1.256	1.297	1.323
60×10	0.468	0.483	0.509
70×10	0.282	0.294	0.302
80×10	0.294	0.312	0.321
90×10	0.240	0.247	0.259
20×20	1.900	2.105	2.190
30×20	0.425	0.478	0.532
40×20	2.176	2.316	2.501
50×20	0.265	0.307	0.338
60×20	1.293	1.345	1.400
70×20	0.731	0.763	0.801
80×20	0.299	0.324	0.339
90×20	0.910	0.967	0.998

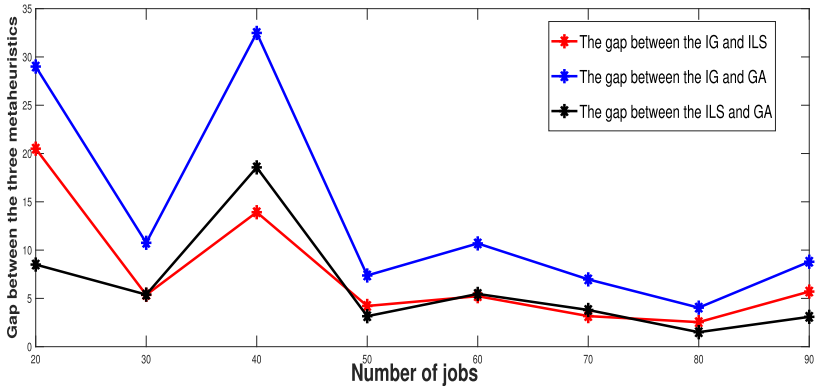

FIGURE 9. The gap between the three proposed metaheuristics in the case of $m = 20$.

good performance of the IG metaheuristic on the one hand, on the other hand we notice that the difference between the two algorithms IG and GA is significant and in comparison to the GA metaheuristic, the ILS metaheuristic gives good results. The final point to take away from this graph is that the difference between the three metaheuristics lessens as the size of the instance grows.

The results given in Table 4 represent the values of $T_{\max}$ and the average percentage of deviation obtained by each metaheuristic proposed for the case of large instances. In a set of 13 instances, the IG algorithm generates 10 best solutions, while the ILS algorithm yields just 2. As can be seen in the last three columns of Table 4, the average deviation percentage obtained by the IG algorithm is lower than the ILS algorithm and the GA algorithm displays low results compared to the two metaheuristics IG and ILS. As a result, the suggested IG algorithm outperforms the ILS and GA algorithms in terms of performance.

TABLE 4. A comparison of the maximum tardiness values and RPI indicator for the three meta-heuristics IG, ILS and GA for relatively large instances.

Instance	Best	Metaheuristic			Deviation		
		IG	ILS	GA	IG	ILS	GA
100 × 10	4703.200	5636.650	5630.150	5642.150	0.1985	**0.1971**	0.1996
150 × 10	1120.650	1120.950	1123.800	1102.9000	**0.0161**	0.0164	0.0190
200 × 10	13 231.500	13 239.500	13 259.000	12 496.000	**0.0589**	0.0595	0.0611
300 × 10	23 752.000	23 755.000	23 755.000	22 744.000	**0.0443**	0.0445	0.0445
400 × 10	31 233.000	31 231.000	31 254.500	30 892.000	0.0110	0.0110	0.0117
100 × 20	1123.500	1484.450	1488.550	1577.650	**0.3213**	0.3249	0.4042
150 × 20	5087.900	5601.800	5628.800	5680.300	**0.1010**	0.1063	0.1164
200 × 20	11 830.000	11 988.500	11 948.500	12 042.500	0.0134	**0.010**	0.0180
300 × 20	21 621.000	22 495.000	22 530.500	22 561.000	**0.0404**	0.0421	0.0435
400 × 20	24 670.000	26 507.000	26 531.500	26 581.000	**0.0745**	0.0755	0.0775
450 × 20	32 094.000	33 311.500	33 317.000	33 384.500	**0.0379**	0.0381	0.0402
500 × 20	35 706.000	36 830.500	36 843.500	36 867.000	**0.0315**	0.0319	0.0325
600 × 20	47 509.000	49 014.500	49 034.000	49 049.500	**0.0317**	0.0321	0.0324

Notes. Bold number signifies best value.

6. Conclusion

The current study aimed to tackle the PFSP under the constraint of SIST, with the objective of minimizing the maximum tardiness criteria. To achieve this goal, we employed a set of resolution approaches, including an exact method based on MILP, two heuristic algorithms, the first based on Johnson rule (HBJR) and the second based on NEH algorithm. In other hand, three meta-heuristic algorithms were implemented such as IG, ILS and GA. We have conducted an experimental evaluation on small, medium and relatively large instances, in order to compare the effectiveness of these approaches. First of all, the two heuristics, namely, NEH and HBJR heuristic were compared to the results of the MILP for small instances. Based on the results obtained, it can be concluded that the NEH heuristic is a more effective approach than the HBJR algorithm for small instances. Additionally, the NEH heuristic requires significantly less CPU time compared to the MILP approach. Therefore, it is recommended to use this heuristic as a resolution approach for flow shop scheduling problems. In other hand, for medium and relatively large instances, the IG algorithm showed better performance compared to the other two meta-heuristics (ILS and GA). Overall, this study contributes to the existing literature on PFSP by providing a thorough investigation of the different adopted approaches to solve the $Fm \mid$ permu, SIST $\mid T_{\max}$ problem with a specific focus on minimizing maximum tardiness. The proposed models and algorithms demonstrated promising results and the findings can serve as a basis for future researches in PFSP problems with SIST constraint.

References

[1] H.V.D. Parunak, Characterizing the manufacturing scheduling problem. *J. Manuf. Syst.* **10** (1991) 241–259.
[2] S.M. Johnson, Optimal two-and three-stage production schedules with setup times included. *Nav. Res. Logistics Q.* **1** (1954) 61–68.
[3] T. Yoshida and K. Hitomi, Optimal two-stage production scheduling with setup times separated. *AIIE Trans.* **11** (1979) 261–263.
[4] A. Allahverdi and H.M. Soroush, The significance of reducing setup times/setup costs. *Eur. J. Oper. Res.* **187** (2008) 978–984.
[5] M.R. Garey, D.S. Johnson and R. Sethi, The complexity of flowshop and jobshop scheduling. *Math. Oper. Res.* **1** (1976) 117–129.
[6] K. Peng, L. Wen, R. Li, L. Gao and X. Li, An effective hybrid algorithm for permutation flow shop scheduling problem with setup time. *Proc. CIRP* **72** (2018) 1288–1292.

[7] J. Belabid, S. Aqil and K. Allali, Solving permutation flow shop scheduling problem with sequence-independent setup time. *J. Appl. Math.* (2020). DOI: 10.1155/2020/7132469.

[8] H. Sadki, J. Belabid, S. Aqil and K. Allali, On permutation flow shop scheduling problem with sequence-independent setup time and total flow time, in International Conference on Advanced Technologies for Humanity. Springer (2021) 507–518.

[9] K. Chakravarthy and C. Rajendran, A heuristic for scheduling in a flowshop with the bicriteria of makespan and maximum tardiness minimization. *Prod. Planning Control* **10** (1999) 707–714.

[10] K. Allali, S. Aqil and J. Belabid, Distributed no-wait flow shop problem with sequence dependent setup time: optimization of makespan and maximum tardiness. *Simul. Modell. Pract. Theory* **116** (2021) 102455.

[11] M.L. Pinedo, Scheduling. Vol. 29. Springer (2012).

[12] R. Ruiz, Q.-K. Pan and B. Naderi, Iterated greedy methods for the distributed permutation flowshop scheduling problem. *Omega* **83** (2019) 213–222.

[13] B. Naderi and R. Ruiz, The distributed permutation flowshop scheduling problem. *Comput. Oper. Res.* **37** (2010) 754–768.

[14] Y. Hou, Y. Fu, K. Gao, H. Zhang and A. Sadollah, Modelling and optimization of integrated distributed flow shop scheduling and distribution problems with time windows. *Expert Syst. App.* **187** (2022) 115827.

[15] A. Sbihi and M. Chemangui, A genetic algorithm for the steel continuous casting with inter-sequence dependent setups and dedicated machines. *RAIRO: Oper. Res.* **52** (2018) 1351–1376.

[16] H.-X. Qin, Y.-Y. Han, B. Zhang, L.-L. Meng, Y.-P. Liu, Q.-K. Pan and D.-W. Gong, An improved iterated greedy algorithm for the energy-efficient blocking hybrid flow shop scheduling problem. *Swarm Evol. Comput.* **69** (2022) 100992.

[17] R. Ruiz and C. Maroto, A genetic algorithm for hybrid flowshops with sequence dependent setup times and machine eligibility. *Eur. J. Oper. Res.* **169** (2006) 781–800.

[18] S. Aqil and K. Allali, Two efficient nature inspired meta-heuristics solving blocking hybrid flow shop manufacturing problem. *Eng. App. Artif. Intell.* **100** (2021) 104196.

[19] J. Cai, R. Zhou and D. Lei, Dynamic shuffled frog-leaping algorithm for distributed hybrid flow shop scheduling with multiprocessor tasks. *Eng. App. Artif. Intell.* **90** (2020) 103540.

[20] B. Xi and D. Lei, Q-learning-based teaching-learning optimization for distributed two-stage hybrid flow shop scheduling with fuzzy processing time. *Complex Syst. Model. Simul.* **2** (2022) 113–129.

[21] Z. Shao, W. Shao and D. Pi, LS-HH: a learning-based selection hyper-heuristic for distributed heterogeneous hybrid blocking flow-shop scheduling. *IEEE Trans. Emerg. Topics Comput. Intell.* **7** (2022) 111–127.

[22] X. Li and Y. Zhang, Adaptive hybrid algorithms for the sequence-dependent setup time permutation flow shop scheduling problem. *IEEE Trans. Autom. Sci. Eng.* **9** (2012) 578–595.

[23] X. Li and S. Ma, Multiobjective discrete artificial bee colony algorithm for multiobjective permutation flow shop scheduling problem with sequence dependent setup times. *IEEE Trans. Eng. Manage.* **64** (2017) 149–165.

[24] R. Vanchipura and R. Sridharan, Development and analysis of constructive heuristic algorithms for flow shop scheduling problems with sequence-dependent setup times. *Int. J. Adv. Manuf. Technol.* **67** (2013) 1337–1353.

[25] Q.-K. Pan, L. Wang and B. Qian, A novel differential evolution algorithm for bi-criteria no-wait flow shop scheduling problems. *Comput. Oper. Res.* **36** (2009) 2498–2511.

[26] B. Qian, L. Wang, R. Hu, D.X. Huang and X. Wang, A de-based approach to no-wait flow-shop scheduling. *Comput. Ind. Eng.* **57** (2009) 787–80.

[27] O. Engin and A. Güçlü, A new hybrid ant colony optimization algorithm for solving the no-wait flow shop scheduling problems. *Appl. Soft Comput.* **72** (2018) 166–176.

[28] G. Deng and X. Gu, A hybrid discrete differential evolution algorithm for the no-idle permutation flow shop scheduling problem with makespan criterion. *Comput. Oper. Res.* **39** (2012) 2152–2160.

[29] Y. Zhou, H. Chen and G. Zhou, Invasive weed optimization algorithm for optimization no-idle flow shop scheduling problem. *Neurocomputing* **137** (2014) 285–292.

[30] Y. Goncharov and S. Sevastyanov, The flow shop problem with no-idle constraints: a review and approximation. *Eur. J. Oper. Res.* **196** (2009) 450–456.

[31] H.H. Miyata and M.S. Nagano, The blocking flow shop scheduling problem: a comprehensive and conceptual review. *Expert Syst. App.* **137** (2019) 130–156.

[32] S. Aqil and K. Allali, On a bi-criteria flow shop scheduling problem under constraints of blocking and sequence dependent setup time. *Ann. Oper. Res.* **296** (2021) 615–637.

[33] I. Ribas, R. Companys and X. Tort-Martorell, A competitive variable neighbourhood search algorithm for the blocking flow shop problem. *Eur. J. Ind. Eng.* **7** (2013) 729–754.

[34] R.L. Graham, E.L. Lawler, J.K. Lenstra and A.H.G. Rinnooy Kan, Optimization and approximation in deterministic sequencing and scheduling: a survey, in Annals of Discrete Mathematics. Vol. 5. Elsevier (1979) 287–326.

[35] A. Vignier, J.-C. Billaut and C. Proust, Les problèmes d'ordonnancement de type ≪flow-shop≫ hybride: état de l'art. *RAIRO: Oper. Res.-Recherche Operationnelle* **33** (1999) 117–183.

[36] M.M. Mazdeh and M. Rostami, A branch-and-bound algorithm for two-machine flow-shop scheduling problems with batch delivery costs. *Int. J. Syst. Sci. Oper. Logistics* **1** (2014) 94–104.

[37] J. Grabowski, E. Skubalska and C. Smutnicki, On flow shop scheduling with release and due dates to minimize maximum lateness. *J. Oper. Res. Soc.* **34** (1983) 615–620.
[38] F. Jolai, H. Asefi, M. Rabiee and P. Ramezani, Bi-objective simulated annealing approaches for no-wait two-stage flexible flow shop scheduling problem. *Sci. Iran.* **20** (2013) 861–872.
[39] A. Allahverdi, Minimizing mean flowtime in a two-machine flowshop with sequence-independent setup times. *Comput. Oper. Res.* **27** (2000) 111–127.
[40] M. Pranzo, Batch scheduling in a two-machine flow shop with limited buffer and sequence independent setup times and removal times. *Eur. J. Oper. Res.* **153** (2004) 581–592.
[41] L. Meng, C. Zhang, X. Shao, Y. Ren and C. Ren, Mathematical modelling and optimisation of energy-conscious hybrid flow shop scheduling problem with unrelated parallel machines. *Int. J. Prod. Res.* **57** (2019) 1119–1145.
[42] L. Meng, C. Zhang, X. Shao, B. Zhang, Y. Ren and W. Lin, More milp models for hybrid flow shop scheduling problem and its extended problems. *Int. J. Prod. Res.* **58** (2020) 3905–3930.
[43] B.N. Srikar and S. Ghosh, A milp model for the n-job, m-stage flowshop with sequence dependent set-up times. *Int. J. Prod. Res.* **24** (1986) 1459–1474.
[44] F.T. Tseng and E.F. Stafford Jr., New milp models for the permutation flowshop problem. *J. Oper. Res. Soc.* **59** (2008) 1373–1386.
[45] M. Takano and M. Nagano, Solving the permutation flow shop problem with blocking and setup time constraints. *Int. J. Ind. Eng. Comput.* **11** (2020) 469–480.
[46] M. Nawaz, E.E. Enscore Jr. and I. Ham, A heuristic algorithm for the m-machine, n-job flow-shop sequencing problem. *Omega* **11** (1983) 91–95.
[47] V. Fernandez-Viagas and J.M. Framinan, NEH-based heuristics for the permutation flowshop scheduling problem to minimise total tardiness. *Comput. Oper. Res.* **60** (2015) 27–36.
[48] H.D. Pour, A new heuristic for the n-job, m-machine flow-shop problem. *Prod. Planning Control* **12** (2001) 648–653.
[49] S. Turner and D. Booth, Comparison of heuristics for flow shop sequencing. *Omega* **15** (1987) 75–78.
[50] R. Ruiz and C. Maroto, A comprehensive review and evaluation of permutation flowshop heuristics. *Eur. J. Oper. Res.* **165** (2005) 479–494.
[51] T. Stützle, Applying iterated local search to the permutation flow shop problem. Technical report. Citeseer (1998).
[52] A.A. Juan, H.R. Lourenço, M. Mateo, R. Luo and Q. Castella, Using iterated local search for solving the flow-shop problem: parallelization, parametrization, and randomization issues. *Int. Trans. Oper. Res.* **21** (2014) 103–126.
[53] R. M'hallah, An iterated local search variable neighborhood descent hybrid heuristic for the total earliness tardiness permutation flow shop. *Int. J. Prod. Res.* **52** (2014) 3802–3819.
[54] X. Dong, P. Chen, H. Huang and M. Nowak, A multi-restart iterated local search algorithm for the permutation flow shop problem minimizing total flow time. *Comput. Oper. Res.* **40** (2013) 627–632.
[55] X. Dong, H. Huang and P. Chen, An iterated local search algorithm for the permutation flowshop problem with total flowtime criterion. *Comput. Oper. Res.* **36** (2009) 1664–1669.
[56] R. Ruiz and T. Stützle, A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem. *Eur. J. Oper. Res.* **177** (2007) 2033–2049.
[57] B.J. Jeong, J.-H. Han and J.-Y. Lee, Metaheuristics for a flow shop scheduling problem with urgent jobs and limited waiting times. *Algorithms* **14** (2021) 323.
[58] R. Ruiz and T. Stützle, An iterated greedy heuristic for the sequence dependent setup times flowshop problem with makespan and weighted tardiness objectives. *Eur. J. Oper. Res.* **187** (2008) 1143–1159.
[59] S. Wang and M. Liu, A genetic algorithm for two-stage no-wait hybrid flow shop scheduling problem. *Comput. Oper. Res.* **40** (2013) 1064–1075.
[60] C. Oğuz and M.F. Ercan, A genetic algorithm for hybrid flow-shop scheduling with multiprocessor tasks. *J. Scheduling* **8** (2005) 323–351.
[61] A. Branda, D. Castellano, G. Guizzi and V. Popolo, Metaheuristics for the flow shop scheduling problem with maintenance activities integrated. *Comput. Ind. Eng.* **151** (2021) 106989.
[62] A. Noorul Haq, T.R. Ramanan, K.S. Shashikant and R. Sridharan, A hybrid neural network–genetic algorithm approach for permutation flow shop scheduling. *Int. J. Prod. Res.* **48** (2010) 4217–4231.
[63] O. Etiler, B. Toklu, M. Atak and J. Wilson, A genetic algorithm for flow shop scheduling problems. *J. Oper. Res. Soc.* **55** (2004) 830–835.
[64] C.R. Reeves, A genetic algorithm for flowshop sequencing. *Comput. Oper. Res.* **22** (1995) 5–13.
[65] T. Murata, H. Ishibuchi and Hideo Tanaka, Genetic algorithms for flowshop scheduling problems. *Comput. Ind. Eng.* **30** (1996) 1061–1071.
[66] L. Wang, L. Zhang and D.-Z. Zheng, An effective hybrid genetic algorithm for flow shop scheduling with limited buffers. *Comput. Oper. Res.* **33** (2006) 2960–2971.
[67] K. Wang, H. Luo, F. Liu and X. Yue, Permutation flow shop scheduling with batch delivery to multiple customers in supply chains. *IEEE Trans. Syst. Man Cybern. Syst.* **48** (2017) 1826–1837.

[68] M. Rebai, I. Kacem and K.H. Adjallah, Earliness–tardiness minimization on a single machine to schedule preventive maintenance tasks: metaheuristic and exact methods. *J. Intell. Manuf.* **23** (2012) 1207–1224.