Contents lists available at GrowingScience

International Journal of Industrial Engineering Computations

homepage: www.GrowingScience.com/ijiec

Integrating sequence-dependent setup times and blocking in hybrid flow shop scheduling to minimize total tardiness

Atıl Kurt[a*]

[a]*Industrial Engineering Department, Alanya Alaaddin Keykubat University, 07425, Antalya, Türkiye*

C H R O N I C L E

Article history:
Received September 2 2024
Received in Revised Format September 18 2024
Accepted October 14 2024
Available online October 15 2024

Keywords:
Hybrid flow shop scheduling
Iterative local search
Hybrid genetic algorithm
Total tardiness
Blocking
Sequence-dependent setup time

A B S T R A C T

This study addresses the minimization of total tardiness in a hybrid flow shop scheduling problem with sequence-dependent setup times and blocking constraints. Each production stage includes multiple machines, and there are no buffers between the stages. The setup time required to process a job depends on the previously processed job. Two mixed-integer linear programming models are developed to formulate the problem. Moreover, an iterative local search algorithm and hybrid genetic algorithms are proposed to have quality solutions with minimal computational efforts. Several computational tests are conducted to tune the heuristic parameters for better performance. Computational experiments are carried out to evaluate the performance of solution methodologies in terms of quality and time. The results indicate that while mixed-integer programming models can solve small-size problem instances, they are not capable of solving large-sized instances. However, the proposed heuristic algorithms find quality solutions for all instances in a very short time.

1. Introduction

The classical flow shop scheduling problem (*FSSP*) is crucial in manufacturing industries such as semiconductors, automobiles, and textiles. In *FSSP*, each job must be processed at each stage, with a single machine used at each stage to achieve performance measures such as minimization of Makespan (C_{max}), total completion time $\left(\sum C_j\right)$, and total tardiness $\left(\sum T_j\right)$. To improve the production efficiency based on these performance measures, identical parallel machines are introduced into the classical *FSSP*. This production pattern is known as hybrid (or flexible) flow shop scheduling (*HFSP*). This problem was first introduced by Salvador (1973) in the synthetic fibers industry. In *HFSP*, more than one machine is available at each stage, and jobs need to be processed on one of them. The *HFSP* environment can be observed in several real-world industries, such as electronics, steel, paper production, textiles, and chemicals.

The HFSP can be categorized as no-wait, limited buffer, infinite buffer, and blocking according to the storage strategy between the stages (Zheng et al., 2024). The infinite buffer strategy is the most commonly used in the HFSP literature. However, in many manufacturing industries, storage capacity between the stages is zero, which leads to the consideration of blocking constraints. Two types of blocking have been studied; "release when starting blocking" and "release when completing blocking" (Yuan et al., 2009). In the problem environment of this study, a machine is blocked until the processing of the job on the machine begins at the next stage (release when starting blocking).

* Corresponding author
E-mail atil.kurt@alanya.edu.tr (A. Kurt)
ISSN 1923-2934 (Online) - ISSN 1923-2926 (Print)
2025 Growing Science Ltd.
doi: 10.5267/j.ijiec.2024.10.005

Some tasks, such as tool replacement and cleaning, must be done between processing two jobs, which leads to the consideration of setup time. In the literature, two types of setup time were considered: sequence-dependent setup time (*SDST*) and sequence-independent setup time (*SIST*). Dealing with *SIST* may be trivial, as it can be attributed to the processing time of jobs. On the other hand, *SDST* can be a challenging problem because the sequence of tasks affects the required setup time and, consequently, the performance measure.

Customer satisfaction is a critical consideration in scheduling theory, making the completion of jobs by their given due dates essential. Moreover, real-life scheduling problems are often related to due dates or deadlines. This study aims to minimize the total tardiness of jobs to enhance customer satisfaction. Note that tardiness is the amount of time by which a job's completion time exceeds its given due date.

This study addresses the problem of scheduling jobs to minimize total tardiness in a hybrid flow shop scheduling problem with blocking and sequence-dependent setup time considerations (HFSP-B-SDST). It is desired to find the optimal allocation of jobs to machines at each stage and the sequence of jobs on each machine. To our knowledge, this problem has not been studied in the related literature, making its application to real-life scenarios and the lack of existing research a strong motivation for this study. The contribution of the study to scheduling literature is threefold. First, it is the first attempt to minimize total tardiness in *HFSP-B-SDST*. Second, two mixed-integer linear programming (*MILP*) models are developed to solve the problem. Finally, two heuristic algorithms, namely iterative local search (*ILS*) and hybrid genetic algorithm (*HGA*), are proposed to solve large-scale problem instances in reasonable times.

The rest of this paper is organized as follows. Section 2 reviews the related literature relevant to *HFSP-B-SDST*. The problem definition and the *MILP* models are briefly presented in Section 3. The proposed heuristic algorithms, *ILS* and *HGA*, are described in detail in Section 4. Section 5 presents the computational tests used to evaluate the performance of the solution methodologies, and Section 6 discusses the conclusions and outlines several future directions.

2. Literature Review

The literature review extensively discusses research related to *HFSP-B-SDST*. The HFSP has been widely studied, and readers interested in more detail can refer to the literature reviews by Tosun et al. (2020) and Çolak and Keskin (2021). For the tardiness minimization objective in *FSSP*, Vallada et al. (2008) provide a review and evaluation of heuristic algorithms. Over the past two decades, numerous studies have focused on the blocking *FSSP* problem, and the works by Hall and Sriskandarajah (1996) and Miyata and Nagano (2019) are recommended for those interested in more detail. Table 1 summarizes the papers most closely related to *HFSP-B-SDST,* discussing problem characteristics and constraints, objectives, the presence of *MILP* models, and the algorithms used in each study.

Table 1
A summary of studies

Study	Characteristics and Constraints	Objective	MILP odel	Algorithm
Ruiz and Maroto (2006)	*SDST*, Eligibility, *Rm*	C_{max}	No	*GA*
Rashidi et al. (2010)	*Rm,* Blocking, *SDST*	C_{max} and T_{max}	No	*GA*
Hakimzadeh Abyaneh and Zandieh (2012)	*SDST*, Limited Buffer	C_{max} and $\sum T_j$	No	*SPGA II, NSGA II*
Elmi and Topaloğlu (2013)	Blocking, *Qm,* Eligibility	C_{max}	Yes	*SA*
Tao et al. (2014)	*STST, Rm*	Production Period	No	*SA, GA*
Ebrahimi et al. (2014)	Group Scheduling, *SDGS*, Uncertain Due Date	C_{max} and $\sum T_j$	No	*NSGA II*, Multi-Objective *GA*
Li and Pan (2015)	Limited Buffer, *Rm*	C_{max}	No	Hybrid Artificial Bee Colony
Moccellin et al. (2018)	Blocking, *SIST, SDST*	C_{max}	No	Heuristic Algorithms Along with Priority Rules
Pessoa et al. (2021)	*SDST,* Blocking, *Rm*	C_{max}	Yes	*GRASP*
Aqil and Allali (2021)	*SDST,* Blocking, *Qm*	$\sum(T_j + E_j)$	No	Migratory Bird Optimization, Water Wave Optimization
Qin et al. (2022)	Blocking	Total Energy Consumption	Yes	*IGA*
Maciel et al. (2022)	*SDST,* Blocking	C_{max}	Yes	*HGA*
Wang et al. (2023-a)	Blocking	C_{max}	Yes	Variant *IGA*
Wang et al. (2023-b)	Blocking, Distributed	C_{max}	Yes	Advanced *IGA*
Liu et al. (2024)	Blocking, Distributed	C_{max}	Yes	Tri-individual *IGA*
Shao et al. (2024)	Blocking, Distributed, Assembly, *SDST*	C_{max}	Yes	Feedback Learning-Based Selection Hyper-Heuristic
Zheng et al. (2024)	*SDST,* Limited Buffer, *Rm*, Re-entrant	$\sum w_j C_j$	Yes	Cooperative Adaptive *GA*
This study	*SDST,* Blocking	$\sum T_j$	Yes	*ILS, HGA*

SDGS: Sequence-dependent group schedule, *GRASP*: Greedy randomized adaptive search procedures, *SPGA II*: Sub-population genetic algorithm II approach, *NSGA II*: Non-dominated sort of genetic algorithm.

As shown in Table 1, several problem characteristics have been studied in the context of *HFSP*, including blocking, distributed scheduling, group scheduling, and *SDST*. Additionally, some papers have utilized different parallel machine structures beyond identical parallel machines, such as unrelated (*Rm*) and uniform (*Qm*) parallel machines. Most studies have used makespan as the primary objective function. However, some studies have considered makespan along with other performance measures simultaneously (Rashidi et al., 2010; Naderi et al., 2011; Abyaneh & Zandieh, 2012; Ebrahimi et al., 2014). The studies by Tao et al. (2014) and Qin et al. (2022) used objectives different from those traditionally found in classical scheduling literature. Furthermore, Table 1 indicates that the Genetic Algorithm (*GA*) and its variations are the most popular approach among the algorithms, although other algorithms such as Simulated Annealing (*SA*), various iterated greedy algorithms (*IGA*), and swarm optimization techniques are also used. Preventive maintenance of machines has also been studied in the HFSP literature. Nederi et al. (2010) examined preventive maintenance within HFSP to minimize makespan and propose *GA* and artificial immune systems. Additionally, Khamseh et al. (2015) considered preventive maintenance in the context of group scheduling and *SDGS* to minimize makespan, using *GA* and *SA*. Feng et al. (2018) investigated the same problem to minimize total cost. On the other hand, Işık et al. (2023) studied eligibility on unrelated parallel machines in *HFSP* to minimize makespan and developed a constraint programming approach to solve the problem.

A closely related study by Maciel et al. (2022) addressed *HFSP-B-SDST* to minimize makespan. They proposed an HGA that incorporates Greedy Randomized Adaptive Search Procedures. In contrast, this study focuses on minimizing a different objective function, specifically total tardiness, and proposes two different heuristic algorithms: *ILS* and *HGA*.

3. Problem Definition and Mathematical Models

This section briefly defines the problem under study and its assumptions, followed by its complexity and a numerical example, and then presents two MILP models. The assumptions and characteristics of the problem are as follows:

- The hybrid flow shop environment includes S stages, each denoted by s.
- The system has J jobs, which are ready for processing at time zero. Each job j ($j = 1,2,\dots,J$) has a due date D_j.
- Each job requires S operations, which must be performed at different stages. The processing time of jobs at each stage is independent and known in advance, denoted by P_{js}.
- There are M_s identical parallel machines in each stage. The number of machines may vary across stages, and all machines are available at time zero.
- Jobs and machines are independent, with no eligibility restrictions.
- Machines require a setup when switching between jobs. The manufacturing system should consider deterministic sequence-dependent setup times, denoted by $SS_{jj's}$, which are known in advance.
- Each machine can process only one job at a time.
- Preemption of jobs is not allowed.
- Maintenance of machines is not considered, so the machines are continuously available.
- Due to the lack of storage area between stages, blocking constraints are considered. Therefore, a machine that has processed a job must be blocked until an available machine in the next stage is ready. Consequently, machines cannot process another job until the current job stays.
- The system includes only one factory (no distributed system), jobs are considered individually (no group scheduling), and there is no no-wait constraint.
- The problem considered in this study can be described as follows, based on the assumptions given above: There is a set of sequence-dependent jobs, each requiring multiple operations to be processed sequentially on identical parallel machines at each stage. It is expected to allocate the jobs to machines and determine their sequence on each machine to minimize total tardiness.

Proposition 1. *The problem under study is NP-complete in the strong sense.*

Proof: The Problem can be simplified if *SDST* is not considered and makespan is used as the objective function instead of total tardiness. In this case, the problem reduces to an FSHP problem with blocking to minimize the makespan, which is shown to be strongly *NP-complete* by Hall and Sriskandarajah (1996). Therefore, our problem is also *NP-complete* in the strong sense. ■

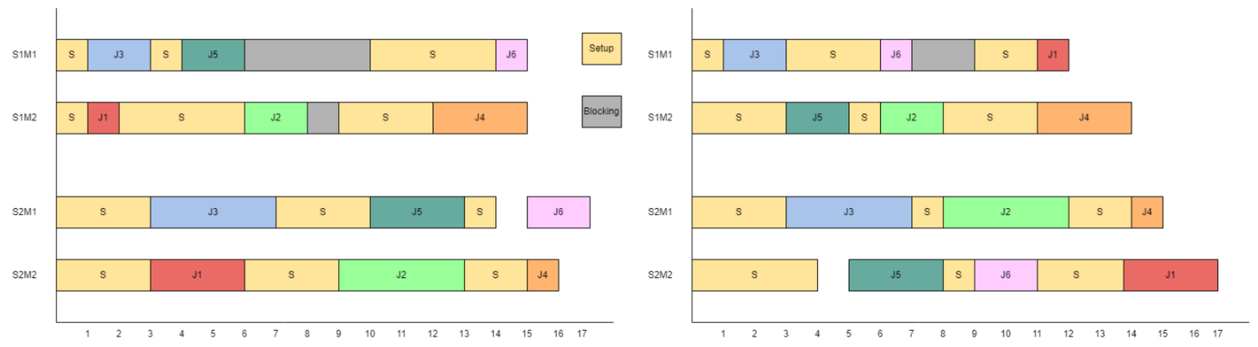
An illustrative example is provided to describe the problem. Six jobs need to be processed across two stages, each with two machines. The corresponding dataset for the problem is presented in Table 2. Note that the diagonal of the setup time matrix indicates the initial setup time of the jobs at the corresponding stage.

Fig. 1 (a) and (b) represent a feasible and optimal solution for the illustrative example, respectively. Blocking of machines can be observed in both schedules: in Stage 1 after Jobs 5 and 2 in the feasible solution, and in Stage 1 after Job 6 in the optimal solution. The total tardiness of the feasible schedule is 16 (0+6+0+2+3+5).

Table 2
Sequence-dependent setup times, processing times, and due dates for the illustrative example

	Stage 1/Stage 2							
Jobs	1	2	3	4	5	6	Processing time	Due Date
1	1/3	4/3	3/2	2/4	3/5	2/2	1/3	9
2	5/2	5/1	3/2	3/2	3/5	4/2	2/4	7
3	3/5	5/1	1/3	3/2	1/3	3/1	2/4	8
4	2/3	5/4	4/2	4/4	2/5	4/5	3/1	14
5	1/5	1/3	3/2	3/2	3/4	4/1	2/3	10
6	2/3	1/4	3/5	5/4	4/3	5/3	1/3	12

This feasible solution can be improved through better assignment and sequencing of jobs, which may lead to eliminating blocking, optimizing sequence dependencies, and achieving a better schedule with reduced total tardiness. By applying one of the MILP models, the optimal solution for the illustrative example, as shown in Fig. 1 (b), yields a total tardiness of 14 (8+5+0+1+0+0).



(a) (b)

Fig. 1. A feasible (a) and optimum (b) schedule of the illustrative example

Based on the assumptions and considerations outlined above, two mixed-integer linear programming (MILP) models have been developed to solve the problem optimally. The following subsections discuss these MILP models in detail.

3.1 Mixed Integer Linear Programming Model 1

The first model uses two primary binary decision variables which are the assignment of jobs to machines at each stage and the order of jobs within the stages. The following decision variables are utilized in MILP model 1.

C_{js}	Completion time of job *j* at stage *s*.
D_{js}	Departure time of job *j* at stage *s*.
T_j	Tardiness of job *j*.

$$Y_{jsm} = \begin{cases} 1 & \text{If job } j \text{ is assigned to machine } m \text{ at stage } s \\ 0 & \text{Otherwise} \end{cases}$$

$$Z_{jj's} = \begin{cases} 1 & \text{If job } j \text{ processed on the same machine before job } j' \ (j' > j) \text{ at stage } s. \\ 0 & \text{Otherwise} \end{cases}$$

Based on the parameters given in Section 3 and decision variables outlined above, the following MILP model is proposed.

$$\min z = \sum_{j=1}^{J} T_j \tag{1}$$

subject to

$$\sum_{m=1}^{M_s} Y_{jsm} = 1 \quad \text{for } j = 1,2,\dots,J; s = 1,2,\dots,S \tag{2}$$

$$C_{j's} \geq D_{js} + SS_{jj's} + P_{j's} + L_s(Y_{jsm} + Y_{j'sm} + Z_{jj's} - 3) \quad \text{for } j,j' = 1,2,\dots,J; j < j'; s = 1,2,\dots,S; m = 1,2,\dots,M_s \tag{3}$$

$$C_{js} \geq D_{j's} + SS_{j'js} + P_{js} + L_s(Y_{jsm} + Y_{j'sm} - Z_{jj's} - 2) \quad \text{for } j,j' = 1,2,\dots,J; j < j'; s = 1,2,\dots,S; m = 1,2,\dots,M_s \tag{4}$$

$$C_{js} \geq P_{js} + SS_{0js} \quad \text{for } j = 1,2,\dots,J; s = 1,2,\dots,S \tag{5}$$

$$C_{js+1} = D_{js} + P_{js+1} \quad \text{for } j = 1,2,\dots,J; s = 1,2,\dots,S-1 \tag{6}$$

$$D_{js} \geq C_{js} \quad \text{for } j = 1,2,\dots,J; s = 1,2,\dots,S \tag{7}$$

$$T_j \geq D_{jS} - D_j \quad \text{for } j = 1,2,\dots,J \tag{8}$$

$$T_j, C_{js}, D_{js} \geq 0 \quad \text{for } j = 1,2,\dots,J; s = 1,2,\dots,S \tag{9}$$

$$Y_{jsm}, Z_{jj's} \in \{0,1\} \quad \text{for } j,j' = 1,2,\dots,J; s = 1,2,\dots,S; m = 1,2,\dots,M_s \tag{10}$$

The total tardiness of jobs is minimized in (1). Jobs at each stage are assigned to one machine in Eq. (2). Eq. (3) and Eq. (4) are used to calculate the completion time of jobs at each stage while considering processing time, setup time, and departure times. Eq. (5) ensures that the completion time of a first assigned job is greater than the sum of its processing and initial setup time. Eq. (6) and Eq. (7) guarantee that the blocking of machines due to jobs is considered. Eq. (8) ensures the tardiness of the job is greater than the differences between its departure time at the final stage and its due date. Eq. (9) and Eq. (10) represent the non-negative continuous variables and binary restrictions, respectively.

3.2 Mixed Integer Linear Programming Model 2

The MILP 2 model uses a single type of binary variable which is used to determine the immediate sequence of jobs. The jobs immediately follow each other starting with a dummy job (Job 0). The defined new decision variable and the formulations are given below.

$$X_{jj's} = \begin{cases} 1 & \text{If job } j \text{ is processed immediately before job } j' \text{ at stage } s \\ 0 & \text{Otherwise} \end{cases}$$

min (1)

subject to

Constraint sets (5), (6), (7), (8) and (9)

$$\textstyle\sum_{j'=1}^{J} X_{0j's} \leq M_s \quad \text{for } s = 1,2,\dots,S \tag{11}$$

$$\textstyle\sum_{j'=1,j'\neq j}^{J} X_{jj's} \leq 1 \quad \text{for } j = 1,2,\dots,J;\ s = 1,2,\dots,S \tag{12}$$

$$\textstyle\sum_{j'=0,j'\neq j}^{J} X_{j'js} = 1 \quad \text{for } j = 1,2,\dots,J;\ s = 1,2,\dots,S \tag{13}$$

$$C_{j's} \geq D_{js} + SS_{jj's} + P_{j's} + L_s(X_{j'js} - 1) \quad \text{for } j,j' = 1,2,\dots,J; j \neq j'; s = 1,2,\dots,S \tag{14}$$

$$X_{j'js} \in \{0,1\} \quad \text{for } j,j' = 0,1,\dots,J; s = 1,2,\dots,S \tag{15}$$

The restriction on the number of machines used is satisfied by counting the number of jobs that follow the dummy job 0 in each stage in Eq. (11). Each job can be followed by at most one job in Eq. (12), while each job must follow one job at each stage in Eq. (13). Eq. (14) is used to calculate the completion time of a job at a stage by considering the departure time of the preceding job, as well as the processing and setup times. Eq. (15) is used to define the new binary variable.

The Eq. (16) is used to determine an efficient $L_s$value, which can lead to finding the MILPs solutions in a shorter time while satisfying the corresponding constraints.

$$L_s = \sum_{s'=1}^{s} \sum_{j=1}^{J} P_{js'} + J \times \max_{j,j'=1,2,\dots,J} \{SS_{jj's}\} \quad \text{for } s = 1,2,\dots,S \tag{16}$$

4. Proposed Heuristic Algorithms

The size of the MILP model increases dramatically with the number of jobs and stages, making it difficult to reach an optimal within a reasonable timeframe. Furthermore, preliminary experiments reveal that MILP 1 failed to obtain a feasible solution for some problem instances. Additionally, as discussed in Section 3, there is no polynomial-time algorithm available to solve the problem. These challenges motivated us to develop heuristic algorithms that can provide promising solutions with small computational efforts. Two heuristic algorithms have been developed for the solution: an Iterative Local Search (ILS) algorithm and a hybrid genetic algorithm (HGA). Both heuristic algorithms aim to find the best sequence of jobs to solve the problem. The sequence of jobs is decoded according to the framework given below.

```
The framework of decoding
1.   for all jobs (according to sequence) do
2.       for all stages do
3.           Calculate temporary machine load ($TML$) which is equal to the sum of machine load and setup time.
4.           If all $TML > D_{[j],s-1}$ Then
5.               Choose the machine having the smallest $TML$.
6.           Otherwise
7.               Choose the machine among the machines having $TML \leq D_{[j],s-1}$ which has the smallest $\left|D_{[j],s-1} - TML\right|$
8.           Assign a job to the machine and calculate the completion time.
9.           Update the departure time of the machine on Stage s-1.
10.      end for
11.  end for
```

4.1 Iterative Local Search

Many problems in the scheduling literature have been solved using different versions of the ILS algorithm. The ILS algorithm in this study begins by considering eight different dispatching rules in combination with the well-known NEH algorithm. Each dispatching rule generates a sequence of jobs, and then the NEH algorithm along with the decoding procedure described above is applied to find a feasible solution. The total tardiness is calculated for the eight feasible solutions obtained by the dispatching rules, and the improvement phase of the algorithm starts with the sequence that has the minimum total tardiness among them. The dispatching rules applied are listed below.

(1) Shortest processing time: Jobs are sequenced according to ascending order based on their total processing time, which is the sum of processing times at each stage.
(2) Longest processing time: Descending order of total processing times of jobs is used for sequencing.
(3) Earliest due date: Ascending order of due date of jobs are used for sequencing.
(4) Slack time: Jobs are sequenced in ascending order based on the difference between their due date and total processing time.
(5) Slack time with critical stage. First, the load of each stage is calculated by summing the processing time of the jobs at that stage. Then, the critical stage having the maximum load is identified. Jobs are sequenced in increasing order of the difference between their due date and processing time in the critical stage.
(6) Slack with final stage: Jobs are sequenced based on the difference between their due date and processing time on the final stage.
(7) Slack with the first stage: Ascending order of the difference between the due date and processing time on the first stage is used for sequencing.
(8) Critical ratio: The jobs are ordered according to the ratio of their due date to total processing time.

The ILS algorithm generally consists of three phases: Initialization, improvement, and perturbation. The initialization phase has been discussed above. In the Improvement phase, the algorithm uses insertion and pairwise interchange search techniques. In the insertion search, jobs are sequentially selected and inserted into different positions to create new candidate solutions. This process is repeated for all jobs and positions to generate a set of candidate solutions. The best candidate solution is then chosen, and the procedure is repeated if an improvement is observed. In the pairwise interchange search, two jobs are selected, and their positions are swapped to obtain a new feasible solution. This procedure is similar to the insertion search and is applied to create improved solutions. However, relying solely on these two search techniques may cause the algorithm to get stuck in a local optimum. To address this, the perturbation phase is introduced. In this phase, the job with the maximum lateness (i.e., the difference between its due date and completion time) is selected and moved to the final position if the final iteration solution differs from the previous one. If not, the sequence of jobs is randomly divided into two parts, and the order of these parts is interchanged. After the perturbation phase, the algorithm returns to apply the insertion and pairwise interchange procedures until the pre-determined number of iterations (*PNI*) is completed.

The total tardiness value obtained by the *ILS* algorithm improves with a higher *PNI* value, but it will not be significantly affected after a certain PNI value. On the other hand, the solution time of the algorithm increases with higher *PNI* values. To determine an efficient *PNI* value, seven different *PNI* values were tested:10, 20, 30, 50, 70, and 100. Out of the 300 problem instances described in Section 5, 60 were used for parameter tuning. The average total tardiness (*ATT*) and the solution time (*ST*) for each *PNI* value are reported and illustrated in Fig. 2. According to Fig. 2, the *PNI* value 20 was chosen because, at higher *PNI* values, the *ATT* does not reduce significantly, while the *ST* increases substantially.

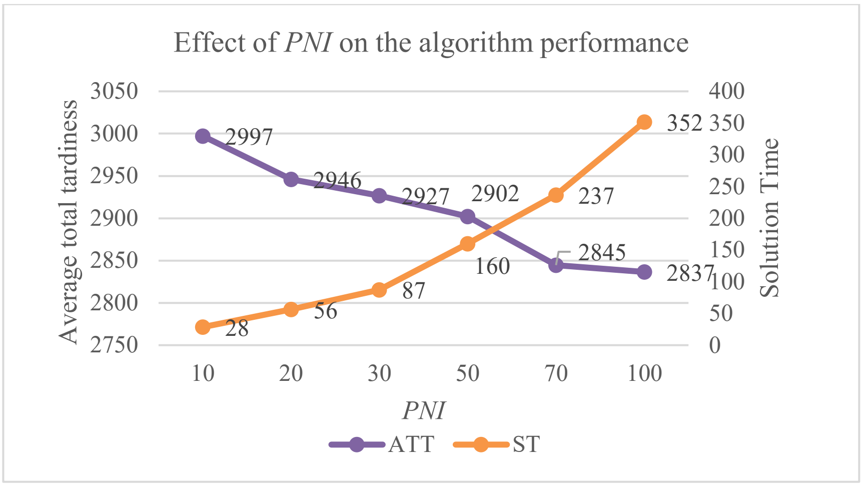


Fig. 2. Effect of *PNI* at *ILS* algorithm on the *ATT* and *ST* values

4.2 Hybrid Genetic Algorithm

The proposed Hybrid Genetic Algorithm (HGA) combines a genetic algorithm with the Iterative Local Search (ILS) algorithm described earlier. The general procedure of the genetic algorithm is used to enhance the solution. However, whenever a sequence generated after crossover and mutation operators significantly diverge from the current population, the improvement and perturbation procedures of the ILS algorithm are applied.

To determine divergence, Kendall's Tau coefficient, as proposed by Kendall (1938), is used to calculate the correlation between the new sequence and the population members. First, Kendall's Tau coefficient is calculated between the new sequence and all population members. Then, the maximum of these values is compared to a predetermined Kendall's Tau value (PKTau). If the obtained maximum value is less than PKTau, the ILS algorithm is applied. Otherwise, the genetic algorithm procedures are repeated to generate different population members.

In the proposed algorithm, each gene of a chromosome is represented by a unique positive integer, corresponding to a specific job number. As a result, the chromosome represents a sequence of jobs, with the length of the chromosome equal to the total number of jobs. After generating a new solution, the decoding procedure described in Section 4 is applied to obtain a feasible solution for the original problem. The total tardiness score derived from this solution is then used as the fitness value of the chromosome.

The predetermined number of members is used as the population size (PS) in the GA. Eight members of the population are generated using the dispatching rules described in Section 4.1 in combination with the NEH algorithm, while the remaining members are created randomly. After the population is generated or updated, the members are ranked in ascending order based on their fitness values. The next generations are formed through parent selection, crossover, and mutation procedures. The Ranking Selection method is employed for parent selection, with linearly assigned selection probabilities (SP_i) as given in Eq. (17).

$$SP_i = \frac{PS - \text{rank of } i + 1}{PS(PS+1)/2} \tag{17}$$

In the crossover procedure, four types of crossover methods were analyzed for efficiency during parameter setting, and one was selected for use in the GA:

(1) *Uniform Crossover:* A random number between 0 and 1 is generated for each gene. If the number is less than the predetermined uniform crossover probability (UCP), the gene is taken from the first parent for the first offspring and from the second parent for the second offspring. Otherwise, the reverse is done. If a gene is duplicated, the original sequence of the parent is used to replace the duplicate.
(2) *Single-Point Crossover:* A random integer between 1 and J (the chromosome length) is generated, dividing the parents into two parts. The first offspring inherits the genes from the first parent up to the crossover point, and the remaining genes are filled in from the second parent, ensuring no gene is duplicated.

(3) *Two-Point Crossover:* Two random points are generated between 1 and *I*, splitting the parents into three parts. The middle segment of the first parent is directly passed to the first offspring, and the remaining genes are filled from the second parent, starting after the second crossover point, ensuring no gene duplication.
(4) *Order Crossover:* Similar to the two-point crossover, but the filling of the remaining genes begins with the first part of the offspring rather than the third, ensuring the integrity of the sequence without duplicates.

The mutation operator is selectively applied to offspring in the genetic algorithm, with a small probability of introducing variation. A random number between 0 and 1 is generated, and if it is less than the predetermined mutation probability (*MP*), the mutation is performed. During mutation, two different jobs are randomly selected, and their positions are swapped in the offspring's sequence.

When a new member is created, the population is updated to ensure continuous improvement. In this HGA, a full elitism strategy is applied, meaning only the best chromosomes survive. New members are added to the population if they are better than at least one existing member. Once a new member joins the population, the correlation between the new and existing members is calculated using Kendall's Tau coefficient. If the maximum Kendall's Tau coefficient is less than the predetermined threshold (*PKTau*), the ILS algorithm is executed for a specified number of iterations (*PNI_GA*). If the new member is distinct from the existing population after applying *ILS*, the population is updated.

The proposed *HGA* is terminated after a pre-determined number of iterations (*GPNI*). In other words, the *HGA* algorithm stops and returns the best solution obtained once the number of iterations reaches *GPNI*.

To execute the *HGA*, pre-determined parameters must be selected as they directly affect the solution efficiency obtained by the *HGA*. In this *HGA*, the parameters *GPNI, PNI_GA, MP, PS,* and *PKTau* need to be determined. Additionally, one of the four crossover types should be chosen, and the *UCP* in the first crossover type should be selected whenever the uniform crossover type is used. For each parameter, four different levels are tested while keeping the rest of the parameters at fixed values, similar to Salvietti et al. (2014). While selecting the parameters, the *ATT* and *ST* performance measures were used to evaluate the parameters. The parameters with their levels, the obtained *ATT* and *ST* values, and the best, fixed, and selected levels are given in Table 2. Subsequently, the selected parameters provided in Table 2 are used during the experiments in the *HGA*.

Table 3
Parameter tuning results for *HGA*

Parameter	Levels	1	2	3	4	Best	Selected	Fixed
GPNI	Values	200	400	600	800		400	400
	ATT	2803	2802	2802	2796	800		
	ST	118	118	118	118	400		
PNI_GA	Values	1	2	3	5		2	2
	ATT	2803	2802	2723	2728	3		
	ST	92	118	188	314	1		
MP	Values	0.01	0.04	0.07	0.1		0.04	0.07
	ATT	2798	2740	2802	2785	0.04		
	ST	110	109	118	127	0.04		
PS	Values	10	20	30	50		20	20
	ATT	2978	2802	2720	2760	30		
	ST	39	118	167	276	10		
PKTau	Values	0.3	0.5	0.7	0.9		0.9	0.7
	ATT	2884	2851	2802	2700	0.9		
	ST	60	76	118	137	0.3		
Crossover Type	Values	1	2	3	4		3	3
	ATT	2887	4113	2802	2803	3		
	ST	119	1	118	139	2		
UCP	Values	0.3	0.5	0.7	0.9		0.3	0.7
	ATT	2835	2856	2887	3596	0.3		
	ST	117	133	119	17	0.9		

5. Computational Experiments

Parameter setting, performance measures, and performance of the two *MILP* models and two heuristic algorithms are discussed in this section. In the solution of *MILP* models, Microsoft Visual C++ implementation of IBM ILOG CPLEX Optimization Studio V22.1 was used under the time limit of one hour. Moreover, the *ILS* and *HGA* algorithms are coded in Microsoft Visual C++. Servers equipped with a 2.93 GHz CPU and 96 GB of RAM are used for all computations.

5.1 Parameter Setting and Problem Instances Generation

The experiments in this study used the following parameter settings. Most of the parameters are designed according to the testbed given by Maciel et al. (2022).

(1) *Processing time*: Generated from discrete uniform distribution *DU*[1, 125].
(2) *Sequence-dependent setup times*: Generated by discrete uniform distribution and three types of setup types are used: Small *DU*[1, 25], medium *DU*[26, 75], and large *DU*[76, 125].
(3) *Due date*: First, a modified NEH algorithm is utilized with the longest processing time dispatching rule, the decoding method given above, and the objective of minimizing makespan (C_{max}). Then, the due dates of the jobs are generated from a discrete uniform distribution $[0.1C_{max}, C_{max}]$.
(4) *Number of jobs, machines, and stages*: The numbers of jobs are taken as 5, 10, 25, 50, and 100, whereas the numbers of stages are taken as 3 and 7. For each stage, 2 and 5 machines (*M*) are used.

Five problem instances are generated for each combination of the above parameters. Hence, the computational study includes a total of 300 (5×2×2×3×5) problem instances.

5.2 Performance Measures

This section introduces the performance measures used for the *MILP* models and the heuristic algorithms. Their performances are evaluated according to the following five metrics:

(1) *NFS*: The number of times the *MILP* models have been able to find a feasible integer solution.
(2) *NOS*: The number of problem instances for which the optimality is proven by the *MILP* models.
(3) *NBS*: The number of problem instances where the solution methodology finds the best solution among all methods. The best solution is the minimum total tardiness obtained across all solution methodologies.
(4) *ATT*: Average total tardiness returned by solution methodology.
(5) *ST*: The amount of time (in seconds) required to obtain a solution from the methodology.

Note that the lower bound returned by the MILP models is zero for some problem instances. Therefore, calculating the percent deviation from this lower bound may lead to misleading results. Moreover, percent deviations from the best solutions are not considered a performance measure, as the best solution is close to zero in some problem instances. In these cases, even small deviations in tardiness from the best solution can appear disproportionately large, which could lead to misleading interpretations.

5.3 Discussion of the results

The performance of the two *MILP* models, *ILS,* and *HGA* algorithms based on the performance measures given in Section 5.2 is discussed in this section. First, *MILP* models are compared based on *NFS* and *NOS,* as provided in Table 4. The table presents the results categorized according to a number of jobs, stages, machines, and setup types to observe their effect. The last column of the table represents the total problem instances that fall into each corresponding measure.

It can be seen from Table 4 that *MILP* 1 obtained feasible solutions for 258 out of 300 problem instances, whereas *MILP* 2 found feasible solutions for all problem instances. Additionally, *MILP* 1 achieved the optimum solution for 63 problem instances, while *MILP* 2 achieved this for 92 instances. These results indicate that *MILP* 2 outperforms *MILP* 1 based on these two performance measures. According to the *NOS* and *NFS* values, it can be concluded that the performance of the *MILP* models deteriorates with the increasing number of jobs due to the direct effect of the number of binary variables. However, the same conclusion cannot be reached regarding the number of stages and setup types, as the performance measures are similar across these factors. On the other hand, the performance of the *MILP* models improves with the number of machines. This improvement may be attributed to the increased flexibility and the greater number of alternative solutions available with more machines.

Table 4
Comparison of MILP models based on the number of feasible and optimum solutions obtained

		J					*S*		*M*		Setup Type			
	Level	5	10	25	50	100	3	7	2	5	1	2	3	Sum
MILP 1	*NFS*	60	60	60	45	33	138	120	108	150	85	86	87	258
	NOS	60	3	0	0	0	32	31	31	32	22	21	20	63
MILP 2	*NFS*	60	60	60	60	60	150	150	150	150	100	100	100	300
	NOS	60	32	0	0	0	48	44	33	59	32	29	31	92

Secondly, the performance measures *NBS*, *ATT*, and *ST* are used to evaluate the performance of the MILP models and heuristic algorithms. The results are presented in Table 5, which is organized according to the number of jobs, stages, machines, and setup type. The last row of the table represents the total problem instances. Similar to the previous discussion, the *MILP* 2 consistently performs better than the *MILP* 1 based on these performance measures. Note that the infeasible solutions returned by *MILP* 1 are not included in the average computations.

Table 5
Comparison of solution methods based on *NBS*, *ATT*, and *ST*.

		J					*S*		*M*		Setup Type			
	Level	5	10	25	50	100	3	7	2	5	1	2	3	Average (Sum)
MILP 1	*NBS*	60	41	0	0	0	52	49	47	54	37	31	33	(101)
	ATT	685	1186	5033	32599	218084	24523	47448	11463	52266	22829	30067	52319	35072
	ST	70.7	3469.6	3600.0	3600.0	3600.0	2787.4	2707.2	2633.5	2834.1	2704.8	2735.7	2808.6	2749.7
MILP 2	*NBS*	60	42	0	0	0	56	49	47	54	37	31	33	(101)
	ATT	685	1227	7133	25822	98379	23468	29831	36819	16479	20961	25217	33770	26649
	ST	0.4	1786.0	3600.0	3600.0	3600.0	2472.7	2566.2	2825.0	2213.8	2469.4	2569.1	2519.8	2519.4
ILS	*NBS*	52	32	14	9	6	57	56	46	67	34	34	45	(113)
	ATT	687	1191	2729	4424	7250	1677	4835	3693	2819	3022	2894	3854	3256
	ST	0.03	0.09	1.9	24.0	259.5	34.9	79.3	33.6	80.6	59.5	57.3	54.5	57.1
HGS	*NBS*	52	32	46	51	54	118	117	116	119	76	82	77	(235)
	ATT	686	1192	2650	4073	6265	1433	4514	3299	2647	2785	2619	3515	2973
	ST	0.05	0.14	3.2	41.0	584.1	81.9	169.5	87.7	163.8	130.5	122.6	124.1	125.7

When comparing the heuristic algorithms with the *MILP* models based on these three measures, the heuristic algorithms outperform the *MILP* models. The *ILS* algorithm found the best solution for 113 problem instances with an average solution time of 51.1 seconds. In contrast, the *HGA* algorithm achieved the best solution for 235 problem instances with an average solution time of 125.7 seconds. The average *ATT* values values for the *ILS* and *HGA* algorithms are 3256 and 2973, respectively. These results indicate that the *HGA* algorithm exhibits the best performance among the solution methodologies. On average, the *ILS* algorithm produces results that are 9.5% worse than those of the *HGA* algorithm. Additionally, the *MILP* 1 and *MILP* 2 models yield average total tardiness values that are more than eleven and eight times higher than those obtained by the *HGA*, respectively. Therefore, it is recommended for decision-makers to use the *HGA* algorithm to achieve efficient solutions.

The increase in the number of jobs results in higher *ATT* values and longer solution times for all solution approaches. However, the increase in solution time for heuristic algorithms is slower compared to the *MILP* models. This suggests that heuristic algorithms will remain efficient as problem sizes grow larger. Conversely, the performance measures from the solution methodologies are not significantly affected by the number of stages, machines, or setup types.

6. Conclusion and Future Research Directions

This paper considers a total tardiness minimization problem of scheduling within an *HFSP* problem with sequence-dependent setup times and blocking constraints. Two *MILP* models are proposed to tackle this problem. Due to the complexity of the problem and inefficiencies of the *MILP* models, two heuristic algorithms are developed.

MILP models are practical for finding optimal solutions for small-sized problem instances, with *MILP* 2 being preferred over *MILP* 1 due to its superior performance. However, as the problem size increases, *MILP* models become inefficient due to their deteriorating performance. In contrast, heuristic algorithms provide effective solutions in a shorter time for both small and large-sized problem instances. Among the proposed methodologies, the Hybrid Genetic Algorithm (*HGA*) is recommended for its strong performance in terms of solution quality and computational efficiency.

The *MILP* models and heuristic algorithms proposed in this study can be readily adapted to solve other types of scheduling problems, such as job shop scheduling or mixed problem environments. Additionally, these methodologies can be extended to accommodate other considerations commonly addressed in scheduling literature, including group scheduling, distributed systems, and maintenance. Moreover, the integration of Iterative Local Search (*ILS*) and Genetic Algorithm (*GA*) in the proposed Hybrid Genetic Algorithm (*HGA*) through Kendall's Tau coefficient can also be applied to incorporate other metaheuristics, such as Tabu Search, Simulated Annealing, or *GA*s.

The current study operates under certain limiting assumptions, such as using identical parallel machines and focusing solely on total tardiness. These assumptions may restrict the applicability of the model. For instance, using unrelated parallel machines or accounting for deteriorating jobs and learning effects are potential extensions. Moreover, optimizing only total tardiness may lead to increased makespan, which might be less effective for decision-makers. Thus, considering both total tardiness and makespan could be more realistic and beneficial.

Future research could explore relaxing these limiting assumptions and investigating various extensions. This might include studying additional objective functions such as the minimization of maximum lateness, number of tardy jobs, or total weighted tardiness, either individually or simultaneously. Incorporating constraints like eligibility, limited buffer capacities, and tool changes into the model could also be valuable for further research.

References

Aqil, S., & Allali, K. (2021). Two efficient nature inspired meta-heuristics solving blocking hybrid flow shop manufacturing problem. *Engineering Applications of Artificial Intelligence*, *100*, 104196.

Ebrahimi, M., Ghomi, S. F., & Karimi, B. (2014). Hybrid flow shop scheduling with sequence dependent family setup time and uncertain due dates. *Applied mathematical modelling*, *38*(9-10), 2490-2504.

Elmi, A., & Topaloglu, S. (2013). A scheduling problem in blocking hybrid flow shop robotic cells with multiple robots. *Computers & operations research*, *40*(10), 2543-2555.

Hakimzadeh Abyaneh, S., & Zandieh, M. (2012). Bi-objective hybrid flow shop scheduling with sequence-dependent setup times and limited buffers. *The International Journal of Advanced Manufacturing Technology*, *58*, 309-325.

Hall, N. G., & Sriskandarajah, C. (1996). A survey of machine scheduling problems with blocking and no-wait in process. *Operations research*, *44*(3), 510-525.

Işık, E. E., Topaloglu Yildiz, S., & Şatır Akpunar, Ö. (2023). Constraint programming models for the hybrid flow shop scheduling problem and its extensions. *Soft Computing*, *27*(24), 18623-18650.

Kendall, M. (1938). A new measure of rank correlation, *Biometrika*, 30, (1–2), 81–89. https://doi.org/10.1093/biomet/30.1-2.81.

Khamseh, A., Jolai, F., & Babaei, M. (2015). Integrating sequence-dependent group scheduling problem and preventive maintenance in flexible flow shops. *The International Journal of Advanced Manufacturing Technology*, *77*, 173-185.

Li, J. Q., & Pan, Q. K. (2015). Solving the large-scale hybrid flow shop scheduling problem with limited buffers by a hybrid artificial bee colony algorithm. *Information Sciences*, *316*, 487-502.

Liu, F., Li, G., Lu, C., Yin, L., & Zhou, J. (2024). A tri-individual iterated greedy algorithm for the distributed hybrid flow shop with blocking. *Expert Systems with Applications*, *237*, 121667.

Maciel, I., Prata, B., Nagano, M., & Abreu, L. (2022). A hybrid genetic algorithm for the hybrid flow shop scheduling problem with machine blocking and sequence-dependent setup times. *Journal of Project Management*, *7*(4), 201-216.

Miyata, H. H., & Nagano, M. S. (2019). The blocking flow shop scheduling problem: A comprehensive and conceptual review. *Expert Systems with Applications*, *137*, 130-156.

Moccellin, J. V., Nagano, M. S., Pitombeira Neto, A. R., & de Athayde Prata, B. (2018). Heuristic algorithms for scheduling hybrid flow shops with machine blocking and setup times. *Journal of the Brazilian Society of Mechanical Sciences and Engineering*, *40*, 1-11.

Naderi, B., Zandieh, M., & Aminnayeri, M. (2011). Incorporating periodic preventive maintenance into flexible flowshop scheduling problems. *Applied Soft Computing*, *11*(2), 2094-2101.

Pessoa, R., Maciel, I., Moccellin, J., Pitombeira-Neto, A., & Prata, B. (2021). Hybrid flow shop scheduling problem with machine blocking, setup times and unrelated parallel machines per stage. *Investigacion Operacional*.

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

Rashidi, E., Jahandar, M., & Zandieh, M. (2010). An improved hybrid multi-objective parallel genetic algorithm for hybrid flow shop scheduling with unrelated parallel machines. *The International Journal of Advanced Manufacturing Technology,* 49, 1129-1139.

Ruiz, R., & Maroto, C. (2006). A genetic algorithm for hybrid flowshops with sequence dependent setup times and machine eligibility. *European journal of operational research,* 169(3), 781-800.

Salvador, M. S. (1973, January). A solution to a special case of flow shop scheduling problems. *In Symposium on the theory of scheduling and its applications* (pp. 83-91). Berlin: Springer.

Salvietti, L., Smith, N R. and Cárdenas-Barrón, L.E. (2014). A stochastic profit maximising economic lot scheduling problem with price optimisation. *European Journal of Industrial Engineering*, 8 (2), 193–221.

Shao, Z., Shao, W., Chen, J., & Pi, D. (2024). A feedback learning-based selection hyper-heuristic for distributed heterogeneous hybrid blocking flow-shop scheduling problem with flexible assembly and setup time. *Engineering Applications of Artificial Intelligence*, *131*, 107818.

Tao, Z., Liu, X., & Zeng, P. (2014). Study on hybrid flow shop scheduling problem with blocking based on GASA. *The Open Automation and Control Systems Journal*, 6(1).

Tosun, Ö., Marichelvam, M. K., & Tosun, N. (2020). A literature review on hybrid flow shop scheduling. *International Journal of Advanced Operations Management*, 12(2), 156-194.

Vallada, E., Ruiz, R., & Minella, G. (2008). Minimising total tardiness in the m-machine flowshop problem: A review and evaluation of heuristics and metaheuristics. *Computers & Operations Research*, 35(4), 1350-1373.

Wang, Y., Wang, Y., & Han, Y. (2023). A variant iterated greedy algorithm integrating multiple decoding rules for hybrid blocking flow shop scheduling problem. *Mathematics*, 11(11), 2453.

Yuan, K., Sauer, N., & Sauvey, C. (2009, September). Application of EM algorithm to hybrid flow shop scheduling problems with a special blocking. *In 2009 IEEE Conference on Emerging Technologies & Factory Automation* (pp. 1-7). IEEE.

Zheng, Q., Zhang, Y., Tian, H., & He, L. (2024). A cooperative adaptive genetic algorithm for reentrant hybrid flow shop scheduling with sequence-dependent setup time and limited buffers. *Complex & Intelligent Systems*, 10(1), 781-809.