Contents lists available at GrowingScience

Journal of Project Management

homepage: www.GrowingScience.com

A hybrid genetic algorithm for the hybrid flow shop scheduling problem with machine blocking and sequence-dependent setup times

Ingrid Simões Ferreira Maciel[a], Bruno de Athayde Prata[b*], Marcelo Seido Nagano[a] and Levi Ribeiro de Abreu[a]

[a]*University of São Paulo, Department of Production Engineering, São Carlos, Brazil*
[b]*Federal University of Ceará, Department of Industrial Engineering, Fortaleza, Brazil*

C H R O N I C L E

Article history:
Received: January 25, 2022
Received in revised format: May 2, 2022
Accepted: May 23, 2022
Available online:
May 23, 2022

Keywords:
Production Sequencing
Makespan
Evolutionary Algorithms
Mixed-Integer Linear Programming

A B S T R A C T

This study contributes to the hybrid flow shop due to a lack of consideration of characteristics existing in real-world problems. Prior studies are neglecting identical machines, explicit and sequence-dependent setup times, and machine blocking. We propose a hybrid genetic algorithm to solve the problem. Furthermore, we also propose a mixed-integer linear programming formulation. We note a predominance of the mathematical model for small instances, with five jobs and three machines because of how fast there is convergence. The objective function adopted is to minimize the makespan, and relative deviation is used as a performance criterion. Our proposal incorporates two metaheuristics in this process: a genetic algorithm to generate sequences (the flow shop subproblem) and a GRASP to allocate the jobs in the machines (the parallel machines subproblem). The extensive computational experience carried out shows that the proposed hybrid genetic algorithm is a promising procedure to solve large-sized instances.

1. Introduction

The flow shop scheduling problem is a hard combinatorial optimization problem widely studied in the last decades. The problem consists of sequencing jobs through multiple production stages with the same machine order. Usually, the performance measures taken are the minimization of makespan, the total completion time or total tardiness. With the growing demand for customized products, some companies have adopted strategies to increase the production capacity as well as to balance the capacity in the stages. As production systems have developed in the last decades, the flow shop environment has been changed to present several parallel machines for each stage, to achieve the above mentioned objectives. The hybrid flow shop scheduling problem (HFSP) was firstly proposed by Salvador (1973) with a case study in the synthetic fibers industry. Since then, several researchers have been studying this class of problems (Ribas, Leisten, and Framiñan, 2010). The hybrid flow shop environment is found in several real-world applications, such as electronics, production of paper, textile, and concrete, manufacturing of photographic films, civil construction projects, Internet service architectures, and load transportation systems (Ruiz et al., 2010). The HFSP is a generalization of the flow shop and parallel machines production environment, two well-known NP-hard problems. The hybrid flow shop for the makespan minimization is an NP-complete problem (Garey, Johnson, and Sethi, 1976). Thus, the proposition of approximation algorithms, such as heuristics and metaheuristics, are needed for solving the above mentioned problem within admissible computation times.

* Corresponding author.
E-mail address: baprata@ufc.br (B. de A. Prata)

doi: 10.5267/j.jpm.2022.5.002

Although setup times usually are included in the processing times, this premise can be unrealistic and led to problems in the production schedule, such as an increase in the project finish time. Allahverdi et al. (2008) emphasize the importance of the explicit consideration of setup times. Moreover, in many manufacturing environments, the technological characteristics may constrain the existence of storage capacity between production stages. Therefore, the machine blocking (buffer zero constraints) typically arises in several real-world problems (Hall and Sriskandarajah, 1996).

Even though a lot of attention has been paid to the HFSP, research that approaches this environment taking into consideration simultaneously explicit setup times and machine blocking are quite limited. Up to now, the hybrid flow shop scheduling problem considering such features was presented by Moccellin et al. (2018). However, this paper only addressed the proposition of dispatch rules, not presenting a mixed integer programming formulation and a robust approximation algorithm, such as a hybrid metaheuristic.

Several researchers have reported the application of hybrid evolutionary algorithms to solve complex production and operation management problems, such as genetic algorithms (Nejati et al. 2016; Chamnanlor et al. 2017; Abreu et al., 2020), evolutionary clustering search (Nagano, Komesu, and Miyata, 2019), imperialist competitive algorithm (Garavito-Hernández et al., 2019), and differential evolution (Prata, Rodrigues, and Framinan, 2022). In the hybrid flow shop environment, there are two subproblems. The first one is the identical parallel machine scheduling problem to allocate jobs in the machines available in each production stage. The second one is the flow shop scheduling problem. Our proposal incorporates two metaheuristics in this process: a genetic algorithm to generate sequences (the flow shop subproblem) and a GRASP to allocate the jobs in the machines (the parallel machines subproblem).

The main contributions of this paper are presented as follows. Firstly, we present a mixed-integer linear programming formulation for the HFSP considering sequence-dependent setup times, machine blocking (buffer zero) and identical parallel machines per production stage. Secondly, we present a hybrid genetic algorithm for finding near-optimal solutions with an admissible computational effort. Thereafter, an intense computational evaluation is conducted to evaluate the robustness of the proposed approaches.

The remainder of this paper is organized as follows: in Section 2, the literature review is presented, in Section 3, the proposed solution procedures are addressed; in Section 4, we discuss some results from computational experiments; finally, in Section 5 we draw some conclusions and suggestions for future works.

2. Literature review

In this section, we present related approaches to the variant under study aiming to highlight the contributions provided by the current research.

Liu and Chang (2000) presented a hybrid flow shop scheduling problem with sequence-dependent setup times. In each production stage, there are considered identical parallel machines. An integer linear programming model is presented in which a single objective function aggregates three objectives: the fulfillment of due dates, the reduction of work in process (WIP) and the reduction of machine setups. As a solution procedure, a Lagrangian relaxation (LR) approach is used.

Rubén Ruiz and Maroto (2006) studied a hybrid flow shop environment considering explicitly sequence-dependent setup times, machine eligibility constraints and unrelated parallel machines per production stage. The makespan minimization is the adopted performance measure. Although an integer linear programming model is not presented, a hybrid genetic algorithm (GA) is proposed as a solution procedure.

Take into consideration the hybrid flow shop scheduling problems with sequence-dependent setup times to minimize the makespan, several approaches have been reported for solving the problem: Zandieh, Ghomi, and Husseini (2006) proposed an immune algorithm (IA); Behnamian, Fatemi Ghomi, and Zandieh (2010) proposed a hybrid max-min ant system (MMAS), simulated annealing (SA) and variable neighborhood search (VNS); and Pan et al. (2017) developed nine metaheuristics (six neighborhood search algorithms and three populational algorithms).

M. Gholami, Zandieh, and Alem-Tabriz (2009a) addressed a hybrid flow shop environment with sequence-dependent setup times and machines with random breakdowns. The problem is considered as a dynamic scheduling and a simulation model is hybridized with a GA for solving the variant under study. The makespan minimization is adopted as the performance measure, considered as a random variable.

Hidri and Haouari (2011) approached a variant of HFSP that considers multiple machining centers and the minimization of makespan. In each center is considered a set of identical parallel machines. Lower and upper bounding strategies, as well as a two-phase heuristic, are proposed.

Elmi and Topaloglu (2013) presented a variant of hybrid flow shop in robotic cells considering machine blocking, multiple part types, unrelated parallel machines per stage, multiple robots and eligibility constraints. A mixed integer programming model and an SA algorithm are proposed. The objective function is the makespan minimization.

Ebrahimi, Ghomi, and Karimi (2014) addressed a hybrid flow shop environment with sequence dependent family setup times and uncertain due dates. There are considered two objective functions: makespan minimization and total tardiness minimization. Two multi-objective evolutionary metaheuristics are proposed.

Ramezani, Rabiee, and Jolai (2015) investigated a flexible flow shop environment with uniform parallel machines per stage, sequence-dependent setup times and no-wait constraints. The objective function is the minimization of makespan. As solution approach, a hybrid metaheuristic based on invasive weed optimization (IWO), VNS and SA is proposed.

Li and Pan (2015) addressed a hybrid flow shop environment in which there are buffer capacities between consecutive production stages. The considered performance measure is the makespan minimization. A hybrid metaheurístic based on artificial bee colony (ABC) and tabu search (TS) algorithms is proposed for solving the problem.

Bozorgirad and Logendran (2016) addressed a hybrid flow shop with sequence-dependent setup times for changing the process between groups of jobs. As objective functions are considered the minimization of weighted completion time and the minimization of the total weighted tardiness. A mixed integer programming model is presented for solving the small-sized test problems. Several algorithms based on the well-known TS, SA, and GA are proposed for solving the variant under study.

Shahvari and Logendran (2018) presented a hybrid flow shop batch scheduling problem with both sequence-dependent and machine-dependent family setup times. Two mathematical programming models are proposed for the variant under study and the objective function is the minimization of the linear combination of the weighted sum of the total weighted completion time and the total weighted tardiness. As solution procedures, a hybrid TS with path relinking (PR) and a particle swarm optimization (PSO) with local search are proposed.

Dios, Fernandez-Viagas, and Framinan (2018) addressed a variant of the HFSP taking into consideration the occurrence of missing operations (i.e., some stages are skipped) with makespan minimization criterion. This premise was observed in a real-world problem in the plastic industry. Although a mixed integer programming model is not presented, several priority rules and improvement algorithms are presented for solving the problem.

Moccellin et al. (2018) firstly proposed a hybrid flow shop taking into consideration concomitantly setup times (both sequence-independent and sequence-dependent) and machine blocking. No mathematical programming model was proposed. As solution procedures, a set of priority rules is presented.

Kurdi (2018) presented an ant colony algorithm with non-daemon actions procedures, for a hybrid flow shop environment with multiprocessor task. In this variant, the jobs can be processed by one or more identical parallel machines at each production stage. The objective function is the makespan minimization. In comparison with other 12 well-known algorithms, the proposed ACO outperformed 7 methods in terms of solution quality.

Table 1 summarizes several related approaches presented in the literature and their characteristics. On the basis of literature review, one can observe that all the features considered in this article are not presented yet.

Table 1
Related approaches to the variant under study

Publication	Objective	setup	blocking	MILP	Solution method
Liu and Chang (2000)	combined objective	√		√	LR
Ruiz and Maroto (2006)	C_{max}	√			GA
Zandieh et al. (2006)	C_{max}	√			IA
Behnamian et al. (2010)	C_{max}	√		√	hybrid meta-heuristics
Pan et al. (2017)	C_{max}	√			meta-heuristics
Gholami et al. (2009)	C_{max}	√			simulation + GA
Hidri and Haouari (2011)	C_{max}				heuristics
Elmi and Topaloglu (2013)	C_{max}				SA
Ebrahimi et al. (2015)	$C_{max}, \Sigma T_j$	√			multiobjective GAs
Ramezani et al. (2015)	C_{max}	√			hybrid meta-heuristics
Qing Li and Ke Pan (2015)	C_{max}		√		hybrid meta-heuristics
Bozorgirad and Logendran (2016)	combined objective			√	meta-heuristics
Shahvari and Logendran (2018)	combined objective	√		√	hybrid meta-heuristics
Dios et al. (20180	C_{max}				heuristics
Moccellin et al. (2018)	C_{max}	√	√		priority rules
This article	C_{max}	√	√	√	GA

3. Method

3.1 Proposed mixed-integer linear programming formulation

Let n jobs that must be processed in g production stages in which each stage $k \in \{1,\ldots,g\}$, $g \geq 2$ is composed of m_k identical parallel machines. The processing of the jobs in the machines occurs in distinct moments, i.e., a given job cannot be processed concomitantly for more than one machine. Each machine presents an associated processing time and for each job we have a sequence-dependent setup time. There is no storage capacity between operations of a given job (machine blocking constraints). The problem under study consists in scheduling a set of jobs $j = \{1,\ldots,n\}$ in which each job has only one operation in each stage. The objective function is the makespan minimization. The operations must be performed sequentially, passing through all production stages.

The encoding of the problem under study is composed of two data structures: a vector with the sequence of the jobs on the stage, $order_j \in \{1,\ldots,n\}$ and a matrix stage per jobs, $allocation_{kj} \in \{1,\ldots,m_k\}$.

Take into consideration an illustrative example with five jobs, three production stages and two machines per stage. The processing times and sequence-dependent setup times are presented in Tables 2 and 3, respectively. The $order = \{4,2,1,3,5\}$ and the allocation equals Table 4, it returns a solution with a makespan of 397-time units, as shown in Fig. 1.

Table 2
Processing times for the presented example

P_{ik}	k = 1	k = 2	k = 3
i = 0	0	0	0
i = 1	79	98	49
i = 2	47	77	83
i = 3	88	101	106
i = 4	16	84	118
i = 5	80	35	103

Table 3
Sequence-dependent setup times for the presented example

S_{ijk}	$j = 1$			$j = 2$			$j = 3$			$j = 4$			$j = 5$		
	$k = 1$	$k = 2$	$k = 3$	$k = 1$	$k = 2$	$k = 3$	$k = 1$	$k = 2$	$k = 3$	$k = 1$	$k = 2$	$k = 3$	$k = 1$	$k = 2$	$k = 3$
i = 0	24	16	20	18	2	15	22	1	2	22	18	19	21	11	6
i = 1	21	21	20	4	10	22	9	16	2	23	15	3	3	17	1
i = 2	9	11	17	2	10	10	20	23	1	7	8	18	13	15	21
i = 3	14	18	5	7	10	20	3	17	12	8	24	15	3	3	1
i = 4	6	19	5	7	1	3	22	16	4	20	23	19	22	21	20
i = 5	9	7	12	9	7	16	7	8	8	20	22	7	12	22	19

Table 4
Job allocation per production stage for the presented example

A_{kj}	j = 1	j = 2	j = 3	j = 4	j = 5
k = 1	1	2	2	1	1
k = 2	1	2	2	1	1
k = 3	2	2	1	1	2

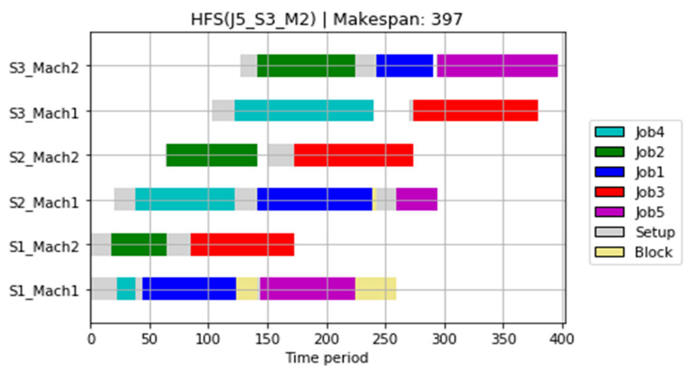

Fig. 1. Gantt chart for the presented instance.

Although mixed integer programming models are not efficient methods for solving production scheduling problems, mainly considering medium and large size instances, here we present a mixed-integer linear programming model for the hybrid flow shop scheduling with machine blocking (buffer zero), sequence-dependent setup times and identical parallel machines per stage. We consider a dummy job 0 preceding the first job on each machine. Hereafter, the notation used for the problem modeling is presented.

Indices and sets

$j \in \{1, \dots, n\}$: set of jobs;

$i \in \{0, \dots, n\}$: set of jobs including dummy job 0;

$k \in \{1, \dots, g\}$: set of stages;

$p \in \{1, \dots, m_k\}$: set of machines.

Parameters

P_{ik}: processing time for job i on stage k;

S_{ijk}: setup time for job j at stage k when it succeeds job i;

M: a sufficiently large integer number (big M).

Decision variables

x_{ijkp}: 1, if job i precedes job j at machine p of stage k, 0, otherwise.

t_{jkp}: release time of job j at machine p of stage k;

C_{max}: maximum makespan on the last stage.

The resulting MILP is as follows.

Objective function

$$min\ C_{max} \tag{3.1}$$

subject to

$$\sum_{i=\{0,\dots,n|i\neq j\}} \sum_{p=1}^{m_k} x_{ijkp} = 1\ \forall j = \{1, \dots, n\}, k = \{1, \dots, g\} \tag{3.2}$$

$$\sum_{j=1}^{n} \sum_{p=1}^{m_k} x_{ijkp} \leq 1\ \ \forall\ i = \{1, \dots, n\}, k = \{1, \dots, g\} \tag{3.3}$$

$$\sum_{j=1}^{n} x_{0jkp} \leq 1\ \forall\ k = \{1, \dots, g\}, p = \{1, \dots, m_k\} \tag{3.4}$$

$$\sum_{h=\{0,\dots,n|h\neq i,h\neq j\}} x_{hikp} \geq x_{ijkp}\ \forall\ i = \{1, \dots, n\}, j = \{1, \dots, n \mid i \neq j\}, k = \{1, \dots, g\}, p = \{1, \dots, m_k\} \tag{3.5}$$

$$t_{0kp} = 0\ \forall\ k = \{1, \dots, g\}, p = \{1, \dots, m_k\} \tag{3.6}$$

$$t_{jkp} + M\left(1 - x_{ijkp}\right) \geq t_{ikp} + S_{ijk} + P_{jk}\ \forall\ i = \{0, \dots, n\}, j = \{1, \dots, n \mid i \neq j\}, k = \{1, \dots, g\}, p = \{1, \dots, m_k\} \tag{3.7}$$

$$t_{jkp} - P_{jk} \geq t_{j(k-1)p0} - M\left(1 - \sum_{i=0}^{n} x_{ijkp}\right) \forall\, j = \{1, \dots, n\}, k = \{2, \dots, g\}, p = \{1, \dots, m_k\}, p0 = \{1, \dots, m_{k-1}\} \tag{3.8}$$

$$t_{j(k-1)p0} - S_{ij(k-1)} - P_{j(k-1)} \geq t_{i(k)p} - P_{ik} - M\big(1 - x_{ij(k-1)p}\big)\, \forall i = \{0, \dots, n\}, j = \{1, \dots, n\}, k = \{2, \dots, g\}, p = \{1, \dots, m_k\}, p0 = \{1, \dots, m_{k-1}\} \tag{3.9}$$

$$C_{max} \geq t_{jgp}\ \forall j = \{1, \dots, n\}, p = \{1, \dots, m_k\} \tag{3.10}$$

$$x_{ijkp} \in \{0,1\}\ \forall i = \{0, \dots, n\}, j = \{1, \dots, n\}, k = \{1, \dots, g\}, p = \{1, \dots, m_k\} \tag{3.11}$$

$$t_{jkp} \geq 0\ \forall j = \{1, \dots, n\}, k = \{1, \dots, g\}, p = \{1, \dots, m_k\} \tag{3.12}$$

$$C_{max} \geq 0 \tag{3.13}$$

The objective (3.1) is to minimize the maximum completion time at the last stage, i.e., the makespan. Constraint set (3.2) ensures that each job is processed on exactly one machine at each stage and has only one predecessor. Constraint set (3.3) ensures that each job has at most one successor for each stage. Constraint set (3.4) ensures that only one job can succeed the dummy job for each machine at each stage. Constraint set (3.5) states that, if job j is assigned to machine p at stage k, its predecessor job i must also be assigned to the same machine on that stage. Constraint set (3.6) sets the release time for the dummy job 0 equals to 0 for all machines and stages. Constraint set (3.7) sets the release date of job j at machine p of stage k to be greater than or equal to the release date of job i (predecessor) plus the setup time between job i and j and the processing time of job j. This constraint gets redundant if the precedence does not occur, due to the big M. Constraint set (3.8) ensures that job j can only start the process at stage k after the same job finishes the process at stage $k-1$. Constraint set (3.9) is the blocking constraints, ensures that if job j is processed after i in stage $k-1$, its processing can only start after job i is allowed to be processed at the subsequent stage, k. Constraint set (3.10) ensures that the makespan is the highest release time of the jobs on the machines of the last stage. Constraint sets (3.11), (3.12) and (3.13) define the domain of the decision variables.

3.2 Proposed genetic algorithm

A two-stage flow shop problem with parallel machines in at least one stage is NP-hard (Gupta, 1988). Hence, the algorithm that runs the proposed mixed-integer linear programming formulation cannot solve large-sized instances to optimality or present high-quality solutions for them. Thereby, we develop a metaheuristic to provide high-quality solutions within an admissible computational effort.

The most common data structure of a solution for a genetic algorithm is a binary array. Although many studies apply this metaheuristic in a problem with a permutational array solution, especially for scheduling problems, such as HFS (Ruben Ruiz and Maroto, 2006).

The allocation argument is more complex, in terms of data structure, and there is a dependency with the job ordering argument. Therefore, this study will apply a different metaheuristic to find the allocation argument, a GRASP algorithm that we detail in subsection 4.2. We will use Taguchi Design to analyze the performance of different parameters, explained later in Section 4.

A Genetic Algorithm (GA) is an evolutionary method. There are successive iterations, also known as generations, where an initial population composed of solutions suffers alterations. The size of the population remains fixed during the generations. Each individual is called a chromosome and can be evaluated by a fitness function. A few individuals are selected and recombined using genetic operators generating offspring. The new individuals improve the population through the generations (Kahraman et al., 2008).

In Fig. 2 we present an overview of the Hybrid Genetic Algorithm: first, we create an initial population generating random permutational arrays for the ordination problem; then, we evaluate the population calculating the makespan using a GRASP algorithm for the allocation subproblem; after we select two elements using one of the selection methods to choose two parents; applying a crossover method, we create a new permutational array; finally, we update the population by replacing the worst solution in the current population with the offspring solution.

```
function HYBRID_GENETIC_ALGORITHM($P$, $S$, ga_iterations, grasp_iterations, viz_iterations, mutation_%)
    iteration = 1
    $population \leftarrow$ CREATE POPULATION ($pop\_size, number\_of\_jobs$)
    $solutions\_pop \leftarrow$ GRASP ($population, grasp\_interations, viz\_iterations, \alpha, P, S$)
    while iteration < ga_iterations do
        $element1, element2 \leftarrow$ SELECTION ($population, solutions\_pop$)
        $offspring \leftarrow$ CROSSOVER($element1, element2$)
        $offspring \leftarrow$ MUTATION($mutation\_\%, offspring$)
        $population, solutions\_pop \leftarrow$ UPDATE POPULATION($population, solutions\_pop, offspring, grasp\_interations, viz\_iterations, \alpha, P, S$)
        iteration = interation + 1
    end while
    return $min(solutions\_pop)$
end function
```

Fig. 2. Hybrid Genetic Algorithm

3.2.1 Encoding and fitness evaluation

One factor that determines the quality of the solution is the population's size, as more solutions are analyzed and, therefore, the greater the potential for obtaining an improved solution (Candan and Yazgan, 2015). However, it is necessary to consider that increasing the population also implies more fitness assessments.

Many authors separate the ordination of jobs and their allocation in the machines, commonly choosing to apply a heuristic to one and evaluate the other. Following (Ruben Ruiz and Maroto, 2006), we calculate the allocation problem in the fitness of the order Genetic Algorithm, but we use a metaheuristic to calculate the solution.

3.2.2 Selection

In each iteration of a GA, we choose parent solutions as the basis for the new generation. The derivation method selects these individuals. We use two methods for analysis, the roulette wheel (RW), and tournament.

In the RW method, two parents are selected at random from a vector of probabilities. Those with better fitness values are more likely to be selected. In the tournament method, four individuals are selected at random. The first two dispute among themselves which is the best to become one of the parents, and the last two do the same. The best of the two pairs is selected.

3.2.3 Crossover method

The crossing is the operator responsible for the recombination of characteristics from the current population. With each new iteration, the selected parents intersect and generate a new possible solution. This offspring solution has qualities of both parents and randomness that depends on the crossing methods chosen.

The first crossover method we use is Block Crossover (BX), whose basic premises were given by Misevičius and Kilda (2005). A random position (rp) is select, where rp is between 25 and 75% of the solution's length. We cross both parents uniting the first part of parent 1 and the second from parent 2. Since we need a feasible solution, element repetition is not allowed. We add the remaining elements following the same sequence as the one in parent 2. Figure 3 presents an example of this method.

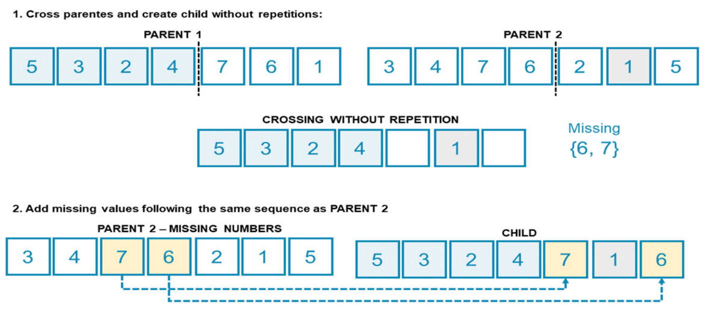
1. Cross parentes and create child without repetitions:
PARENT 1
5 3 2 4 7 6 1
PARENT 2
3 4 7 6 2 1 5
CROSSING WITHOUT REPETITION
5 3 2 4 1
Missing
{6, 7}
2. Add missing values following the same sequence as PARENT 2
PARENT 2 – MISSING NUMBERS
3 4 7 6 2 1 5
CHILD
5 3 2 4 7 1 6

Fig. 3. An example of BX

The second method we use is the Uniform Like Crossover (ULX), a type of adaptation of the uniform crossover for permutational solutions proposed by Tate and Smith (1995). The elements are randomly chosen from one of the parents. Therefore, if both parents have the same element in the same position, so does the offspring. The missing elements, not added so the solution could be feasible, are randomly inserted (Fig. 4).

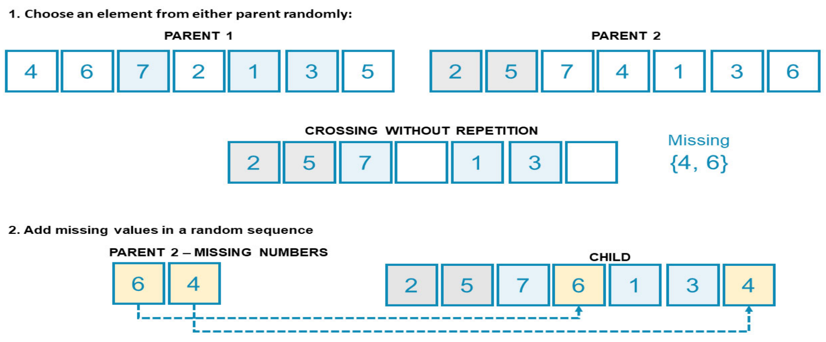
1. Choose an element from either parent randomly:
PARENT 1
4 6 7 2 1 3 5
PARENT 2
2 5 7 4 1 3 6
CROSSING WITHOUT REPETITION
2 5 7 1 3
Missing
{4, 6}
2. Add missing values in a random sequence
PARENT 2 – MISSING NUMBERS
6 4
CHILD
2 5 7 6 1 3 4

Fig. 4. An example of ULX

3.2.4 Mutation

In a genetic algorithm, the mutation serves to give unpredictable randomness to the results and changes the population. As in human genetics, this can be positive or negative for the structure generated. This mutation occurs in a probabilistic manner, with a low probability of occurrence.

When exploring new solutions, the mutation prevents the algorithm from stopping at a local optimum and maintains the diversity of the population but slows down the learning process of the algorithm (Tang and Tseng, 2013).

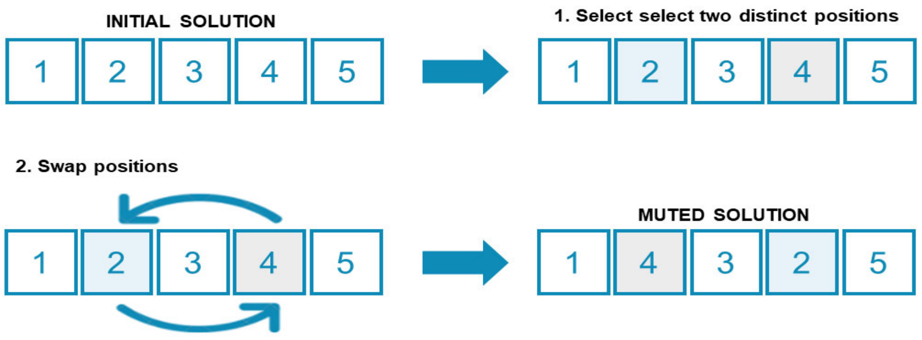

Fig. 5. An example of the swap mutation

The mutation method in a permutational array is simple. If the mutation percentage is reached, the function receives the offspring structure. Then, we select two random and distinct positions of the vector and swap those positions. Therefore, changing the order of processing of two jobs. We show an example of the swap algorithm in Figure 5.

3.3 GRASP

A GRASP (Greedy Randomized Adaptive Search Procedure) is a metaheuristic that randomizes a heuristic solution and improves it through a local search, introduced by Feo and Resende (1995). It is a basic logic for the development of simple and efficient algorithms. It consists of two phases: one constructs an initial solution, the other improves that solution by a local search.

The construction phase requires a heuristic, greedy solution, and a α argument to randomize the solution. The heuristic we apply is the Best-Fit Machine, a derivation of the Best-Fit rule for bin packing problems. Introduced by (Pessoa et al. 2021), we show the heuristic in Equation (3.14).

$$position_{jk} = \min_{p}(P_{jk} + S_{ijk} + machine\_time_{kp}) \tag{3.14}$$

By using only the heuristic solution, we would have only one possible result. However, the algorithm inserts a probabilistic character to the heuristic by creating a restricted candidates list (RCL). A probabilistic argument is required to create this list, the $alpha$ parameter. The α parameter is a number ($0 \leq \alpha \leq 1.$) that indicates how random the solution will be.

$$\beta_i \leq min(\beta) + \alpha \times [max(\beta) - min(\beta)], \forall i \in \{1, \dots, \| \beta \|\} \tag{3.15}$$

The solution will be one of the candidates chosen at random and may or may not be the best solution. We generate the list using Equation (3.15) (Pessoa et al. 2021; González-Neira and Montoya-Torres, 2017). If $\alpha = 0$ we have a heuristic solution and if $\alpha = 1$, we have a random solution.

```
function NEIGHBOR_ALLOCATION_PROBLEM($P$, $S$, $order$, $A$, $solution\_A$, viz_iterations)
    count = 1
    while count < viz_iterations do
        $\widetilde{A} \leftarrow$ SEARCH_NEIGHBOOR ($A$)
        $solution\_\widetilde{A} \leftarrow$ MAKESPAN($P$, $S$, $order$, $\widetilde{A}$)
        if $solution\_\widetilde{A} < solution\_A$ then
            $A \leftarrow \widetilde{A}$
            $solution\_A \leftarrow solution\_\widetilde{A}$
        end if
        count = count + 1
    end while
    return $A$, $solution\_A$
end function
```

Fig. 6. Basis of Local Search - Allocation subproblem

With each iteration of the construction phase, the heuristic needs to adapt because the changes brought about by the previous selection are reflected in the solution of the current element (Feo and Resende, 1995). That is, in this case, in addition to having a percentage affecting the heuristic solution for the current machine, the previously modified choices bring a distinction to the existing alternative.

A local search algorithm is integrated with GRASP so that the solution does not fall into an optimal location. We use a simple local search, where a predetermined number of neighbors is found, and the best solution among the initial one and its neighbors remains.

The basis of the algorithms is shown in Figure 6. To analyze how local search affects the algorithm, we test two forms of a neighboring solution. The first considers that a neighboring solution is one in which the allocation of machines is different in one of the stages for all jobs. The other method is to change the allocation of only one job at all stages.

4. Results and discussion

4.1 Testbed design

The testbed was designed for the assessment of the robustness of the proposed mixed integer programming formulation, as well as the evaluation of the developed hybrid genetic algorithm. We considered a fixed interval for the processing times, with random values uniformly distributed between 1 and 125. For the sequence-dependent setup times, we considered three levels of setups generated with random values also uniformly distributed: $U[1,25]$, $U[26,75]$ and $U[76,125]$.

The total number of instance classes is found by multiplication the number of levels of each parameter. Take into consideration the levels of n = 5, of g = 2, of m_k =2 and of $sd = 3$, we have $5 \times 2 \times 2 \times 3 = 60$ instance classes. For each class, five random instances were generated, which gives a total of 300 test instances. The parameters used in the generated instances are summarized in Table 5.

Table 5
Parameters used in the testbed design

Parameter	Number of levels	Values
Number of jobs (n)	5	5, 10, 25, 50, 100
Number of stages (g)	2	3, 7
Number of parallel machines per stage (m_k)	2	2, 5
Setup times distribution (sd)	3	$U[1.25]. U[26.75]. U[76.125]$

4.2 Taguchi Design

In the 1960s, Dr. Taguchi developed a method to analyze which factors bring less variance for quality characteristics. Using this method, controllable factors will be placed in the internal orthogonal matrix, noise factors in the external orthogonal matrix. Thus, the values of quality characteristics are transformed into noise signals that indicate the best level of combination of parameters (M. Gholami, Zandieh, and Alem-Tabriz, 2009b).

$$\eta = -10 \times log_{10}\left[\frac{1}{n} \times \sum_{i=1}^{n} y_i^2\right] \tag{4.1}$$

To calculate the noise, we use a "small value is good" equation, Equation (4.1) (Candan and Yazgan 2015; M. Gholami, Zandieh, and Alem-Tabriz, 2009b), where n is the number of times the instance is run and y is the objective value.

Table 6
Taguchi Design - L8

	Parameters Design						
	1	**2**	**3**	**4**	**5**	**6**	**7**
1	1	1	1	1	1	1	1
2	1	1	1	2	2	2	2
3	1	2	2	1	1	2	2
4	1	2	2	2	2	1	1
5	2	1	2	1	2	1	2
6	2	1	2	2	1	2	1
7	2	2	1	1	2	2	1
8	2	2	1	2	1	1	2

The orthogonal arrays of a Taguchi Design are conventionally described as $La(b^c)$, where a is the number of experimental runs, b is the number of levels of each factor, and c is the number of variables. We choose the L8 that can have up to seven factors with two levels (2^7). In Table 6 we show the factor's levels in each. Table 7 summarizes the levels of the selected factors.

Table 7
Factors and levels

Factors	**Level 1**	**Level 2**
Population size	50	100
Derivation method	Roulette wheel (RW)	Tournament (T)
Crossover method	Block crossover (BX)	Uniform like crossover (ULX)
Mutation percentage	0.1	0.15
Neighbor population size	150	200
Alpha parameter	0.9	0.75
Neighbor search	Difference in a stage	Difference in a job

4.3 Computational experience

In this section, we evaluate the performance of the mixed-integer linear programming (MILP), implemented in the commercial solver CPLEX 12.8, and the proposed hybrid genetic algorithm implemented in Python 3.8 (Spyder IDE). For each parameter combination level, five random instances were generated. The computer used to run the tests has the following characteristics: AMD Ryzen 5 Processor 3400G and 3.7GHz; 16GB RAM; and operating system Windows 10.

Two types of criteria are essential when comparing the performance of algorithms: Effectiveness or optimality and Efficiency. The first one refers to how good is the solution, concerning the desired criterion. The second indicates the resources necessary to obtain a solution, such as an algorithm complexity and running time (Maccarthy and Liu, 1993).

Since we're talking about implementation in an industrial environment, time is an essential criterion. Therefore, the algorithm has a stop criterion of 600 seconds or ten minutes. In the MILP implementation, the same time limit was established and a tree limit of 12000 megabytes.

We run all instances of five and ten jobs three times for each level of Taguchi's Design. Then, we calculate the signal noise (S/N) using makespan as the objective function. In Table 8 we summarize the mean results and in Table 9 the mean time.

Table 8
Taguchi Design - Summary of Mean Results

INST.	n	g	f_k	SD	L1	L2	L3	L4	L5	L6	L7	L8
01 - 05	5	3	3	(24, 48)	701.1	700.4	701.7	700.4	700.4	700.4	700.4	700.4
06 - 10	5	3	3	(48, 72)	733.3	733.3	733.2	734.3	733.3	733.2	733.2	733.6
11 - 15	5	3	3	(72, 96)	803.4	803.4	856.7	804.3	803.4	803.4	803.4	803.4
16 - 20	5	3	5	(24, 48)	559.2	559.2	559.2	599.1	599.9	559.2	559.2	595.9
21 - 25	5	3	5	(48, 72)	631.8	592.8	592.8	669.6	592.8	592.8	592.8	592.8
26 - 30	5	3	5	(72, 96)	621.8	621.8	664.5	704.3	621.8	621.8	621.8	621.8
31 - 35	5	7	3	(24, 48)	1,365.8	1,365.9	1,369.1	1,366.1	1,366.3	1,365.6	1,365.6	1,365.7
36 - 40	5	7	3	(48, 72)	1,414.2	1,416.7	1,416.4	1,415.5	1,414.8	1,415.1	1,414.3	1,415.9
41 - 45	5	7	3	(72, 96)	1,463.6	1,464.1	1,467.8	1,464.3	1,463.6	1,464.0	1,463.6	1,463.6
46 - 50	5	7	5	(24, 48)	1,245.1	1,478.3	1,317.4	1,237.2	1,277.9	1,312.9	1,312.9	1,268.2
51 - 55	5	7	5	(48, 72)	1,287.2	1,448.2	1,353.4	1,370.5	1,316.0	1,272.4	1,616.3	1,320.7
56 -60	5	7	5	(72, 96)	1,279.8	1,525.1	1,444.0	1,280.4	1,328.0	1,361.6	1,615.5	1,343.1
61 - 65	10	3	3	(24, 48)	1,045.9	1,031.3	1,031.7	1,038.2	1,044.3	1,025.9	1,023.9	1,040.7
66 - 70	10	3	3	(48, 72)	1,140.3	1,129.1	1,129.9	1,143.1	1,142.3	1,131.7	1,129.3	1,144.5
71 - 75	10	3	3	(72, 96)	1,251.9	1,236.7	1,241.3	1,250.2	1,252.6	1,244.5	1,240.3	1,250.9
76 - 80	10	3	5	(24, 48)	761.9	731.6	731.4	752.6	759.7	734.2	731.5	762.5
81 - 85	10	3	5	(48, 72)	805.5	779.1	776.4	795.7	807.4	776.5	776.1	806.9
86 - 90	10	3	5	(72, 96)	888.7	832.5	828.6	872.5	867.3	830.9	829.9	863.5
91 - 95	10	7	3	(24, 48)	1,791.6	1,756.5	1,745.7	1,787.5	1,794.2	1,757.3	1,746.5	1,792.6
96 - 100	10	7	3	(48, 72)	1,860.2	1,838.7	1,827.2	1,848.0	1,860.2	1,834.5	1,827.9	1,859.2
101 - 105	10	7	3	(72, 96)	1,956.7	1,925.3	1,916.7	1,936.8	1,952.0	1,924.1	1,914.7	1,947.2
106 - 110	10	7	5	(24, 48)	1,536.4	1,460.9	1,448.3	1,529.6	1,543.5	1,451.6	1,460.5	1,537.5
111 - 115	10	7	5	(48, 72)	1,626.4	1,527.3	1,504.2	1,619.0	1,631.8	1,543.9	1,544.1	1,633.8
116 - 120	10	7	5	(72, 96)	1,720.1	1,582.3	1,577.9	1,714.5	1,725.6	1,578.7	1,578.2	1,720.9

Table 9
Taguchi Design - Summarize of Times

INST.	n	g	f_k	SD	L1	L2	L3	L4	L5	L6	L7	L8
01 - 05	5	3	3	(24, 48)	24,40	29,13	23,33	36,41	41,27	33,99	44,90	30,09
06 - 10	5	3	3	(48, 72)	25,05	29,92	24,67	37,31	41,62	34,62	42,95	30,99
11 - 15	5	3	3	(72, 96)	23,35	28,43	23,62	36,36	41,65	34,66	42,64	29,23
16 - 20	5	3	5	(24, 48)	95,76	124,29	99,32	150,57	132,90	128,88	158,41	84,41
21 - 25	5	3	5	(48, 72)	95,60	125,60	100,39	146,35	133,08	127,33	158,78	90,94
26 - 30	5	3	5	(72, 96)	94,82	124,73	99,95	172,73	151,52	153,83	158,97	88,03
31 - 35	5	7	3	(24, 48)	157,97	222,79	226,32	254,51	298,76	212,39	252,47	201,39
36 - 40	5	7	3	(48, 72)	159,31	222,60	194,91	242,78	295,40	209,61	252,26	201,15
41 - 45	5	7	3	(72, 96)	168,51	228,57	213,98	275,38	294,63	208,89	267,66	201,12
46 - 50	5	7	5	(24, 48)	378,17	603,23	552,93	577,52	600,94	577,98	608,36	472,18
51 - 55	5	7	5	(48, 72)	389,74	595,52	532,91	585,85	600,72	580,09	614,35	476,47
56 -60	5	7	5	(72, 96)	365,49	593,49	533,56	567,63	600,74	556,50	626,79	481,64
61 - 65	10	3	3	(24, 48)	41,88	47,97	41,31	64,76	69,34	65,96	80,88	49,68
66 - 70	10	3	3	(48, 72)	41,63	48,31	39,60	64,93	69,43	66,09	81,02	49,64
71 - 75	10	3	3	(72, 96)	41,79	48,33	39,14	65,24	69,26	66,10	81,24	49,86
76 - 80	10	3	5	(24, 48)	94,12	106,88	85,46	140,07	145,67	136,05	166,05	100,43
81 - 85	10	3	5	(48, 72)	96,03	106,89	85,44	140,22	145,46	132,12	169,60	100,28
86 - 90	10	3	5	(72, 96)	90,96	106,89	87,39	140,77	140,36	132,75	173,22	100,53
91 - 95	10	7	3	(24, 48)	305,37	390,57	316,50	429,37	483,85	395,49	501,07	345,19
96 - 100	10	7	3	(48, 72)	300,93	397,50	307,25	438,97	485,99	383,70	531,22	346,28
101 - 105	10	7	3	(72, 96)	296,37	409,75	315,87	446,70	503,73	383,36	533,48	350,46
106 - 110	10	7	5	(24, 48)	600,70	601,22	600,90	601,54	601,00	601,08	601,49	600,75
111 - 115	10	7	5	(48, 72)	600,94	601,16	600,86	601,59	601,29	600,93	601,85	601,02
116 - 120	10	7	5	(72, 96)	601,07	601,02	601,06	601,15	601,64	601,25	601,40	600,81

The summary of the mean results of Taguchi's orthogonal design are in Table 8 and the mean time of processing in Table 9. The mean results of the first levels are very similar, but the mean times have a variation that gets more prominent in the larger instances. Figure 7 presents the signal to noise of each Taguchi's Designs by instance level.

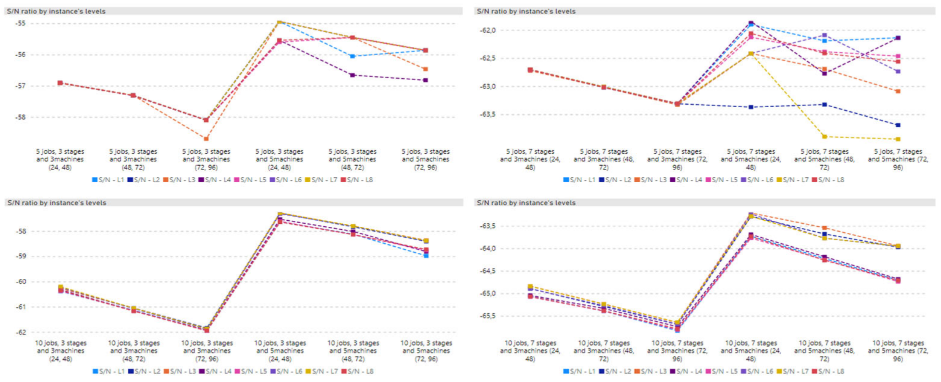

Fig. 7. Taguchi's Designs by instance level

We can analyze the performance of the selected parameters by averaging the levels of Taguchi Designs. For example, level 1 of the order size appears in the first four designs, so its value is an average of these four. Table 10 shows the signal do noise of each parameter. The parameters with higher S/N values, presents the better results. Therefore, we can select the best combination of this parameters, presented in Table 11.

Table 10
Signal to Noise of Parameters

Parameter	**Level 1**	**Level 2**
Order size	-60,9037	-60,8736
Selection method	-60,8712	-60,9062
Crossover method	-60,8993	-60,8780
Mutation Rate	-60,8864	-60,8909
GRASP size	-60,8559	-60,9214
Alpha	-60,9329	-60,8444
Search neighbor method	-60,8920	-60,8958

Table 11
Best Parameters

Parameter	Level / Value
Order size	Level 2 - 100
Selection method	Level 1 - Roulletter wheel (RW)
Crossover method	Level 2 - Uniform like crossover (ULX)
Mutation Rate	Level 1 - 0.15 (15%)
GRASP size	Level 1 - 150
Alpha	Level 2 - 0.75
Search neighbor method	Level 1 - Difference in stage

After the best parameters are selected, we run the experiment with the best level and compare it with the linear-programming. The Relative Percent Deviation (RPD), Equation [rpd], is used for comparison.

$$RPD = \frac{(Cplex\_result - mean\_result)}{Cplex\_result} \tag{4.2}$$

Table 12 summarizes the MILP's and hybrid genetic algorithm results by level, presenting the mean of results and processing times. As can be observed, the algorithm only found the optimal results for the five jobs and three stages instances in the proposed time limit.

The RPD value for mean and minimum values of HGA are very similar e most instances, which indicates the small variation of results. We can see the increase in processing times in instances with seven stages and five machines, because of the iteration variables (number of times the GA and GRASP algorithm's runs).

Table 12
Comparison of results

INST.	n	g	f_k	SD	CPLEX's result	Mean gap	Mean time (CPLEX)	Mean of Minimum results	RPD of Minimum	Mean of mean results	RPD of Mean	Mean time (HGA)
1-5	5	3	3	(24, 48)	700.4	-	2.2	700.4	0.00%	700.4	0.00%	30.4
6-10	5	3	3	(48, 72)	733.2	-	1.7	733.2	0.00%	733.2	0.00%	29.8
11-15	5	3	3	(72, 96)	803.4	-	1.7	803.4	0.00%	803.4	0.00%	29.4
16-20	5	3	5	(24, 48)	559.2	-	1.6	559.2	0.00%	559.2	0.00%	110.6
21-25	5	3	5	(48, 72)	592.8	-	1.1	592.8	0.00%	592.8	0.00%	111.9
26-30	5	3	5	(72, 96)	621.6	-	1.0	621.8	0.04%	621.8	0.04%	110.5
31-35	5	7	3	(24, 48)	1,399.8	0.764	606.0	1,365.6	-2.37%	1,365.6	-2.37%	172.1
36-40	5	7	3	(48, 72)	1,429.6	0.774	604.7	1,414.6	-1.03%	1,414.6	-1.03%	172.1
41-45	5	7	3	(72, 96)	1,474.2	0.771	604.7	1,463.6	-0.71%	1,463.6	-0.71%	172.2
46-50	5	7	5	(24, 48)	1,232.4	0.735	606.8	1,232.4	0.00%	1,232.4	0.00%	499.8
51-55	5	7	5	(48, 72)	1,272.4	0.709	603.3	1,272.4	0.00%	1,272.4	0.00%	529.1
56-60	5	7	5	(72, 96)	1,279.2	0.592	604.7	1,279.2	0.00%	1,279.2	0.00%	494.3
61-65	10	3	3	(24, 48)	1,036	0.705	601.0	1,013.8	-2.11%	1,026.1	-0.92%	59.3
66-70	10	3	3	(48, 72)	1,149	0.768	601.6	1,116.8	-2.74%	1,126.3	-1.98%	58.7
71-75	10	3	3	(72, 96)	1,267.6	0.768	602.4	1,228.8	-3.06%	1,239.3	-2.28%	57.9
76-80	10	3	5	(24, 48)	738.6	0.675	605.0	729.2	-1.25%	732.9	-0.77%	123.9
81-85	10	3	5	(48, 72)	785	0.706	601.3	773.2	-1.48%	778.0	-0.86%	123.7
86-90	10	3	5	(72, 96)	848	0.677	604.0	829.2	-2.18%	831.3	-1.92%	124.3
91-95	10	7	3	(24, 48)	1,798.4	0.889	603.8	1,751.2	-2.61%	1,760.8	-2.07%	361.2
96-100	10	7	3	(48, 72)	1,870.4	0.893	604.2	1,829.4	-2.16%	1,834.9	-1.87%	361.5
101-105	10	7	3	(72, 96)	1,951.4	0.883	603.7	1,914.2	-1.85%	1,925.2	-1.28%	360.6
106-110	10	7	5	(24, 48)	1,412	0.881	602.3	1,439.4	1.93%	1,457.1	3.20%	601.1
111-115	10	7	5	(48, 72)	1,479.4	0.871	604.2	1,485.4	0.42%	1,528.9	3.36%	601.1
116-120	10	7	5	(72, 96)	1,570.4	0.868	603.1	1,581	0.61%	1,594.4	1.45%	600.8
121-125	25	3	3	(24, 48)	2,180	0.903	600.3	2,087	-4.26%	2,100.1	-3.66%	136.4
126-130	25	3	3	(48, 72)	2,432.6	0.917	600.3	2,294.2	-5.65%	2,309.1	-5.03%	136.1
131-135	25	3	3	(72, 96)	2,657	0.923	600.2	2,565.8	-3.43%	2,579.0	-2.93%	136.5
136-140	25	3	5	(24, 48)	1,423.4	0.918	600.5	1,363.6	-4.19%	1,374.0	-3.46%	286.0
141-145	25	3	5	(48, 72)	1,557.4	0.925	600.4	1,491.6	-4.21%	1,498.7	-3.76%	282.5
146-150	25	3	5	(72, 96)	1,636	0.945	600.4	1,602.4	-2.05%	1,606.5	-1.80%	281.9
151-155	25	7	3	(24, 48)	2,940.2	0.970	600.7	2,855.4	-2.86%	2,876.1	-2.16%	601.0
156-160	25	7	3	(48, 72)	3,120.6	0.977	600.8	3,073	-1.52%	3,086.1	-1.10%	601.1
161-165	25	7	3	(72, 96)	3,363.4	0.981	600.5	3,267.2	-2.84%	3,286.4	-2.27%	601.4
166-170	25	7	5	(24, 48)	2,156	0.987	600.2	2,308.8	7.08%	2,348.6	8.93%	601.6
171-175	25	7	5	(48, 72)	2,287.2	0.984	600.2	2,410.8	5.40%	2,449.8	7.11%	602.0
176-180	25	7	5	(72, 96)	2,358.2	0.984	600.4	2,551.4	8.19%	2,593.7	9.99%	602.2

[tab: comparison]

5. Concluding remarks

We addressed a hybrid flow shop problem considering machine blocking and both sequence independent and sequence-dependent setup times. In each production stage, identical parallel machines per stage are considered. The objective function is the minimization of makespan. We propose a hybrid genetic algorithm with two stages. In the first stage, a genetic algorithm produces the sequence of the jobs. In the second one, a GRASP algorithm allocates the jobs in the available machines. Some of the parameters were analyzed by Taguchi's Design and the best combination compared with the proposed mixed-integer linear programming (MILP) model. The MILP presented an excellent performance in small-sized instances, mainly in those with only five jobs. In such instances, the model reached the optimal solution in a few seconds. On the other hand, the hybrid genetic algorithm can return a feasible solution for all the evaluated test instances. We can emphasize that its effectiveness is higher in large-sized problems. The extensive computational experimentation carried out indicates that the proposed hybrid genetic algorithm clearly outperforms the MILP model. As extensions to this work, we suggest further analysis of the crossover parameter using different methods, such as the many cited by Misevičius and Kilda (2005). Besides, we can consider other premises, such as the no-wait constraint. Other performance measures can also be considered, such as total tardiness minimization. We further recommend adding heuristics elements, such as those developed by Moccellin et al. (2018), in the initial population of the genetic algorithm and the implementation of a restart operator, to prevent the algorithm from getting stuck in a local optimum.

References

Abreu, L. R., Cunha, J. O., Prata, B. A., & Framinan, J. M. (2020). A genetic algorithm for scheduling open shops with sequence-dependent setup times. *Computers & Operations Research*, *113*, 104793.

Allahverdi, A., Ng, C. T., Cheng, T. E., & Kovalyov, M. Y. (2008). A survey of scheduling problems with setup times or costs. *European journal of operational research*, *187*(3), 985-1032. https,//doi.org/10.1016/j.ejor.2006.06.060.

Behnamian, J., Fatemi Ghomi, S. M. T., & Zandieh, M. (2012). Hybrid flowshop scheduling with sequence-dependent setup times by hybridizing max–min ant system, simulated annealing and variable neighbourhood search. *Expert systems*, *29*(2), 156-169. https,//doi.org/10.1111/j.1468-0394.2010.00569.x.

Bozorgirad, M. A., & Logendran, R. (2016). A comparison of local search algorithms with population-based algorithms in hybrid flow shop scheduling problems with realistic characteristics. *The international journal of advanced manufacturing technology*, *83*(5), 1135-1151. https,//doi.org/10.1007/s00170-015-7650-9.

Candan, G., & Yazgan, H. R. (2015). Genetic algorithm parameter optimisation using Taguchi method for a flexible manufacturing system scheduling problem. *International Journal of Production Research*, *53*(3), 897-915.

Chamnanlor, C., Sethanan, K., Gen, M., & Chien, C. F. (2017). Embedding ant system in genetic algorithm for re-entrant hybrid flow shop scheduling problems with time window constraints. *Journal of Intelligent Manufacturing*, *28*(8), 1915-1931.

Dios, M., Fernandez-Viagas, V., & Framinan, J. M. (2018). Efficient heuristics for the hybrid flow shop scheduling problem with missing operations. *Computers & Industrial Engineering*, *115*, 88-99. https,//doi.org/10.1016/j.cie.2017.10.034.

Ebrahimi, M., Fatemi Ghomi, S. M. T., & Karimi, B. (2014). Hybrid Flow Shop Scheduling with Sequence Dependent Family Setup Time and Uncertain Due Dates. *Applied Mathematical Modelling, 38*(9), 2490–2504. https,//doi.org/10.1016/j.apm.2013.10.061.

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. https,//doi.org/10.1016/j.cor.2013.01.024.

Feo, T. A., & Resende, M.G.C. (1995). Greedy Randomized Adaptive Search Procedures. *Journal of Global Optimization, 6*(2), 109–33.

Garavito-Hernández, E. A., Peña-Tibaduiza, E., Perez-Figueredo, L. E., & Moratto-Chimenty, E. (2019). A meta-heuristic based on the Imperialist Competitive Algorithm (ICA) for solving Hybrid Flow Shop (HFS) scheduling problem with unrelated parallel machines. *Journal of Industrial and Production Engineering*, *36*(6), 362-370.

Garey, M. R., Johnson, D. S., & Sethi, R. (1976). The complexity of flowshop and jobshop scheduling. *Mathematics of operations research*, *1*(2), 117-129. https,//doi.org/10.1287/moor.1.2.117.

Gholami, M., Zandieh, M., & Alem-Tabriz, A. (2009). Scheduling hybrid flow shop with sequence-dependent setup times and machines with random breakdowns. *The International Journal of Advanced Manufacturing Technology*, *42*(1), 189-201. https,//doi.org/10.1007/s00170-008-1577-3.

Gholami, M., Zandieh, M., & Alem-Tabriz, A. (2009b). Scheduling Hybrid Flow Shop with Sequence-Dependent Setup Times and Machines with Random Breakdowns. *The International Journal of Advanced Manufacturing Technology, 42* (1-2), 189–201.

González-Neira, E. M., & Montoya-Torres, J. R. (2017). A GRASP meta-heuristic for the hybrid flowshop scheduling problem. *Journal of Decision systems*, *26*(3), 294-306.

Gupta, J. ND. (1988). Two-Stage, Hybrid Flowshop Scheduling Problem. *Journal of the Operational Research Society, 39* (4), 359–64.

Hall, N. G., & Sriskandarajah, C. (1996). A Survey of Machine Scheduling Problems with Blocking and No-Wait in Process. *Operations Research, 44*(3), 510–25. https,//doi.org/10.1287/opre.44.3.510.

Hidri, L., & Haouari, M. (2011). Bounding strategies for the hybrid flow shop scheduling problem. *Applied Mathematics and Computation*, *217*(21), 8248-8263. https,//doi.org/10.1016/j.amc.2011.02.108.

Kahraman, C., Engin, O., Kaya, I., & Kerim Yilmaz, M. (2008). An application of effective genetic algorithms for solving hybrid flow shop scheduling problems. *International Journal of Computational Intelligence Systems*, *1*(2), 134-147.

Kurdi, M. (2019). Ant colony system with a novel Non-DaemonActions procedure for multiprocessor task scheduling in multistage hybrid flow shop. *Swarm and evolutionary computation*, *44*, 987-1002.

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. https,//doi.org/10.1016/j.ins.2014.10.009.

Liu, C. Y. (1996, December). Scheduling flexible flow shops with sequence-dependent setup effect. In *Proceedings of 35th IEEE Conference on Decision and Control* (Vol. 2, pp. 1757-1762). IEEE. https,//doi.org/10.1109/70.864235.

Maccarthy, B. L., & Liu, J. (1993). Addressing the gap in scheduling research: a review of optimization and heuristic methods in production scheduling. *The International Journal of Production Research*, *31*(1), 59-79.

Misevičius, A., & Kilda, B. (2005). Comparison of crossover operators for the quadratic assignment problem. *Information Technology and Control*, *34*(2).

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*(2), 1-11. https,//doi.org/10.1007/s40430-018-0980-4.

Nagano, M. S., Komesu, A. S., & Miyata, H. H. (2019). An evolutionary clustering search for the total tardiness blocking flow shop problem. *Journal of Intelligent Manufacturing*, *30*(4), 1843-1857.

Nejati, Mohsen, Iraj Mahdavi, Reza Hassanzadeh, and Nezam Mahdavi-Amiri. (2016). Lot Streaming in a Two-Stage Assembly Hybrid Flow Shop Scheduling Problem with a Work Shift Constraint. *Journal of Industrial and Production Engineering* 33 (7), 459–71. https,//doi.org/10.1080/21681015.2015.1126653.

Pan, Q. K., Gao, L., Li, X. Y., & Gao, K. Z. (2017). Effective metaheuristics for scheduling a hybrid flowshop with sequence-dependent setup times. *Applied Mathematics and Computation*, *303*, 89-112. https,//doi.org/10.1016/j.amc.2017.01.004.

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*.

de Athayde Prata, B., Rodrigues, C. D., & Framinan, J. M. (2022). A differential evolution algorithm for the customer order scheduling problem with sequence-dependent setup times. *Expert Systems with Applications*, *189*, 116097.

Ramezani, P., Rabiee, M., & Jolai, F. (2015). No-wait flexible flowshop with uniform parallel machines and sequence-dependent setup time: a hybrid meta-heuristic approach. *Journal of Intelligent Manufacturing*, *26*(4), 731-744. https,//doi.org/10.1007/s10845-013-0830-2.

Ribas, I., Leisten, R., & Framiñan, J. M. (2010). Review and classification of hybrid flow shop scheduling problems from a production system and a solutions procedure perspective. *Computers & Operations Research*, *37*(8), 1439-1454. https,//doi.org/10.1016/j.cor.2009.11.001.

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.

Ruiz, R., & Vázquez-Rodríguez, J. A. (2010). The hybrid flow shop scheduling problem. *European journal of operational research*, *205*(1), 1-18.

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

Shahvari, O., & Logendran, R. (2018). A comparison of two stage-based hybrid algorithms for a batch scheduling problem in hybrid flow shop with learning effect. *International Journal of Production Economics*, *195*, 227-248. https,//doi.org/10.1016/j.ijpe.2017.10.015.

Tang, P. H., & Tseng, M. H. (2013). Adaptive directed mutation for real-coded genetic algorithms. *Applied Soft Computing*, *13*(1), 600-614.

Tate, D. M., & Smith, A. E. (1995). A genetic approach to the quadratic assignment problem. *Computers & Operations Research*, *22*(1), 73-83.

Zandieh, M., Ghomi, S. F., & Husseini, S. M. (2006). An immune algorithm approach to hybrid flow shops scheduling with sequence-dependent setup times. *Applied Mathematics and Computation*, *180*(1), 111-127.
https,//doi.org/10.1016/j.amc.2005.11.136.