RAIRO-Oper. Res. 55 (2021) 1603–1616
https://doi.org/10.1051/ro/2021076

RAIRO Operations Research
www.rairo-ro.org

AN EFFECTIVE ITERATED GREEDY ALGORITHM FOR BLOCKING HYBRID FLOW SHOP PROBLEM WITH DUE DATE WINDOW

Ahmed Missaoui* and Younès Boujelbene

Abstract. Nowadays many industry consider an interval time as a due date instead of precise points in time. In this study, the hybrid flow shop scheduling problem with basic blocking constraint is tackled. Where jobs, if done within a due window, are deemed on time. Therefore, the criterion is to minimize the sum of weighted earliness and tardiness. This variant of the hybrid flowshop problem is not investigated to the best of our knowledge. we introduced a new metaheuristic centered on the iterated greedy approach. to evaluate the proposed method we start by the re-implementation and the comparison of seven well-selected procedures that treat the hybrid flowshop problem. In order to prove the robustness of our method, we evaluated it using a new benchmark of more than 1000 instances. The experimental results demonstrated that the proposed algorithm is effective and produces a very high solution.

Mathematics Subject Classification. 90B35, 90C27.

Received October 4, 2020. Accepted May 3, 2021.

1. Introduction

Since 1960s, the study of the production systems has attracted the attention of researchers. In fact, it showed that the efficient job scheduling is an essential element to increase productivity and achieve industry goals. Moreover, the variety of the production systems creates diversity in scheduling problem, based on many characteristics and details. Among the well studied scheduling problem we have the hybrid flow shop scheduling HFS, this variant can represent an extension of flowshop problem and the only difference is that we have at least on stage when there is more than one machines. This problem arises in a lot of application such as healthcare, military, and in several industry. With some industries, the type of product and the conditions of production impose certain constraints. This could be found in the production of glasses for example where the primary material goes through a number of stages but with the constraints that the product cannot wait between stages, otherwise the job can leave one machine only when there is another free in the next stage. Therefore we can talk about a lack of storage between stages. This condition is known in the literature by the blocking constraint. Consequently, our work treat HFS with basic constraint known by BHFS. This problem is investigated in the literature when the objective is the minimization of total completion time known by makespan such as in Elmi and Topaloglu [4]. According to Framinan *et al.* [7], makespan increase the resources utilization, that was meaningful in the early days of production but at the same time this ignore the most wanted objective today

Keywords. Hybrid flow shop, scheduling, blocking, iterated greedy, due date window.

Faculté des Sciences Économiques et de Gestion de Sfax, Université de Sfax, Route de l'Aéroport Km 4, Sfax 3018, Tunisie.
*Corresponding author: amd.missaoui@gmail.com

which is the high level service. Nowadays, industry needs to minimize inventory levels and avoid overstocking. On the other hand, it need to increase the customer's satisfaction by meeting their job due dates, which can increase the market share and avert the customer's loss. This objective can be summarised by the minimization of the sum of weighted earliness and tardiness TWET. Therefore, this paper deals with the BHFS with TWET minimization. According to Hall and Sriskandarajah [10], manufacturing technology itself could be a reason for the incidence of a blocking production system. The blocking restriction means that a waiting job does not quit the system until one free machine is accessible in the next stage. In this paper, we are interested to the release when starting blocking (RSb) between all stages *i.e.* a machine stays blocked by a job until this job begins in the next stage. Therefore, this job remains blocked on the machine as long as there is no machine free in the next stage. This blocking mode is used in several scheduling problems by some researchers, such as Yuan *et al.* [35], Zhang *et al.* [37]. However, a few study in the literature introduced the TWET objective function. Nearly 1% of the scheduling problems reviewed by Ruiz and Vázquez-Rodríguez [30]. In fact, HFS has a high level of practicality, that's why this is why it has increased an importance among scheduling problem. Many definitions are introduced for the HFS problem. Simply, it consists of a multistage flow shop, where at least there is a stage contain more than one machine. We have to recall that every machine can process one job at a time. Following the Graham *et al.* [9] norm three-field notation, the $m-$machines BHFS with due date window release dates of jobs where the objective function is the minimization of TWET, which can be denoted as: $((\mathrm{PM}(i))mi = 1)||\mathrm{RSb}||\mathrm{TWET}^{dw}$. If the research studies carried out until now have proven that BHFS with more than two machines is strongly NP-hard when the makespan is considered as measure of performance, it can be obviously inferred that the considered problem cannot optimally be solved in $O(n \log n)$. The HFS problem is a mixture of parallel machines and flow-shop problems.The goal is to get the best order for a set of jobs $j = 1, \ldots, n$ in order to optimize the objective function. This article is structured as follows; in Section 2, we present a literature review then, in Section 3, we introduce the studied scheduling problem, and in Section 4, we describe the proposed algorithm in detail, then, in Section 5, analyze and discuss the obtained results. Finally, conclusions and suggestions for future work are given in Section 6.

2. Literature review

The HFS is a broadly studied optimization problem, which represents several industrial applications. In this context, Ruiz and Vázquez-Rodríguez [30] carried out a review on the HFS problem in which they classify the HFS based on the resolution methods and the objective function. In that paper several several variants of HFS are discussed . For their part, Moccellin *et al.* [17] tackled a different variant of HFS while considering simultaneous blocking constraint and setup time aiming to minimise the makespan. To solve this problem a constructive heuristic algorithm centered on the rules of priority was developed. Zhang *et al.* [37] also considered a particular version of the HFS-derived vehicle scheduling issue with a specific form of blocking constraints. In order to minimize the makespan, the authors used a new heuristic approach to solve the problem. In Ribas *et al.* [27], authors introduced an efficient iterated greedy algorithm to tackle the flowshop scheduling problem with blocking constraint aiming to minimize makespan. Therefore, this paper provides evidence of the robustness of the iterated method with scheduling problem. Newton *et al.* [19] addressed the flow shop scheduling problem simultaneously with setup times and blocking constraint, in that paper, the calculation of the makespan is performed using an improved acceleration method and a constraint-guided local search. For their part, Rashidi *et al.* [26] investigated the HFS with unrelated machines and processor blocking while considering the $C_{\max}$ and $T_{\max}$ as an objective function, to develop a hybrid genetic algorithm to tackle this multi-objective problem. Pan *et al.* [22], proposed four efficient methods based on local search and iterated greedy to tackle the HFS with due date window, comparing with the results of 13 well selected method in the literature of scheduling problem, the proposed method shows a great results. Lebbar *et al.* [15] studied the flow shop scheduling problem with simultaneously blocking constraint and release dates. To optimally solve this problem, two improved evolutionary algorithms are proposed to minimize the weighted sum of the makespan and the maximum tardiness. Grabowski and Pempera [8] studied the HFS simultaneously with RSb and no-wait constraint. To minimize the makespan,

authors applied the tabu-search procedure. In this context, Sawik [32] focused on the classical blocking constraint called release when started blocking (RSb) and proposed a heuristic for multi-stage flow shop problem without storage capacity for the resolution of a mixed integer programming is introduced. Khare and Agrawal [14] studied the HFS with sequence depend setup times, in this work, authors proposed three methods based on metaphors: whale optimization algorithm (OBWOA), hybrid squirrel search algorithm (HSSA) and discrete grey wolf optimization (DGWO) to minimize the TWET. Alfaro-Fernández *et al.* [1] considered the HFS problem with due window where the objective is the minimization of TWET. An auto-designed algorithm (ADA) based on iterated greedy is used for the resolution. In the work of Jing *et al.* [12], an effective IG is developed to tackle the distributed flowshop and TWET minimization. Recently, Aqil and Allali [2] proposed two method inspired from nature to investigate the HFS with blocking constraint and setup times to minimize the sum of earliness and tardiness

Trabelsi *et al.* [34] investigated the HFS scheduling problem with mixed blocking constraint, in this problem, each time only one kind of the studied constraint (RSb and RCb) is considered between stages. Genetic algorithm is used to solve the problem.

For their part, Zeng and Yang [36] proposed a new mixed integer linear program (MILP) and two meta-heuristics to study the HFS with a limited storage. To solve this problem, two assignment rules are used, which are the first available machine (FAM) and the last free machine (LFM). The result showed that the second rule is more efficient. To the best of our knowledge, no research has studied the HFS with (RSb) constraint and TWET penalties simultaneously.

3. Description of problem and notation

3.1. Problem description

Let g be the number of stages and m the number of parallel identical machines per stage. We have to mention that there are n jobs to be scheduled without interruption. At time zero, all these jobs are available. Every one of them will be processed by one machine in each stage. More precisely, each job visits the first stage, then the next stage and so on ... until stage g. We have to mention that we have a limited storage between stages. This is denotes that a machine cannot release the current job and remains blocked until the availability of a machine in the next stage. When Jobs are allocated to machines according to the LFM rule, every job is assigned to the last machine that becomes free *i.e.* the machine with the smallest idle time among all available machines. given that our scheduling problem has no intermediate storage, the jobs the handling operations of which finish earlier will be immediately moved to the next stage.

3.2. Assumptions

Briefly we present the following assumptions for the problem:

- RSb is considered between all stages.
- The value of n, g and m is known.
- The processing time of jobs, the due date window, and the corresponding penalty of both tardiness and earliness are known.
- The last free machine is considered as an assignment rule in order to minimize the idle time.
- When a job starts, before it progresses to the next level, it must be thoroughly processed.
- Each job must be processed on one machine at each stage.
- Release dates at the first stage are zero for all jobs.
- Machine breakdown is not considered.

3.3. Illustrative example

Considering an illustrative example of five jobs, two stages and two machines per stage, the processing time, the due date and earliness tardiness weight are presented in Table 1.

TABLE 1. Processing times, due dates and earliness-tardiness penalties for the example.

	Stage		Due dates		Weights	
Job	1	2	d_j^-	d_j^+	w_j	w_j'
1	3	4	6	8	1	2
2	5	3	5	7	2	1
3	6	2	8	10	4	2
4	2	2	11	13	3	2
5	3	2	11	14	2	1

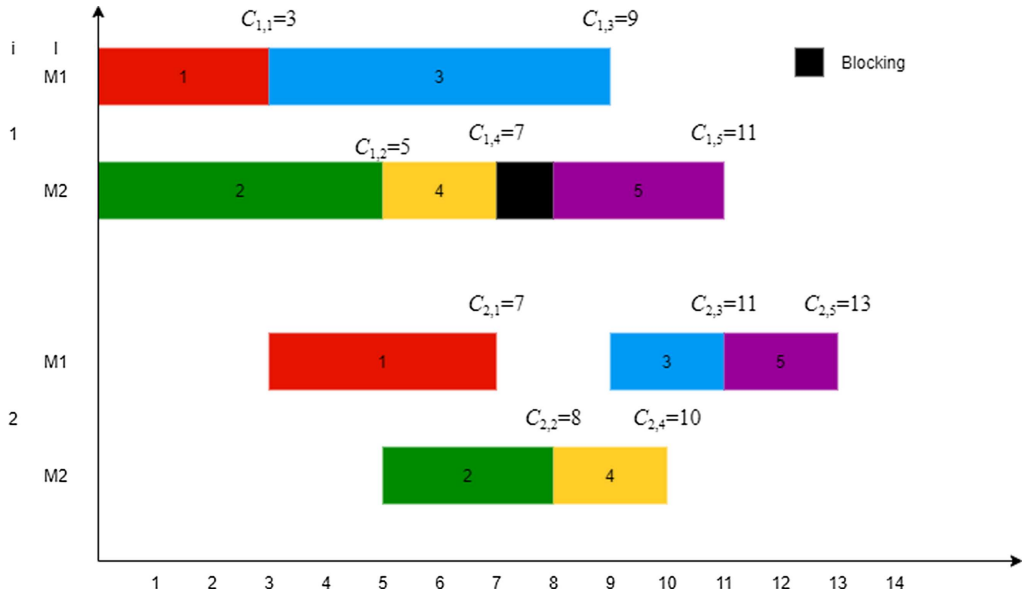

FIGURE 1. Gantt chart for permutation $Seq1$ (1, 2, 3, 4, 5) in the example.

Figure 1 shows the Gant chart of the permutation Seq (1, 2, 3, 4, 5). In Figure 1 the blocking time appears in black as it is a non productive time represents the limit buffer constraint. The completion times of jobs are respectively (7, 8, 11, 10, 13) so $j4$ is early while $j2$ and $j3$ are tardy jobs. As a result: $\text{TWET} = (1 \times 1 + 1 \times 2 + 1 \times 3) = 6$.

4. PROPOSED ITERATED GREEDY PROCEDURES

Iterated greedy (IG) method was first used by Ruiz and Stützle [29] to tackle the flow shop scheduling problem when the makespan was considered as an objective function. In fact, IG has proven great effectiveness with flow shop and scheduling problems in general. Unrelated parallel system scheduling problems were discussed in Fanjul-Peyro and Ruiz [6]. Moreover, a regular flow shop with blocking constraints was approached by Ribas *et al.* [27] using the same method. Also, the blocking job shop scheduling problem (BJSS) was tackled by Pranzo and Pacciarelli [25] using the IG method. On the other hand, the distributed flow shop scheduling problem has been recently solved by Ruiz *et al.* [31] by the IG method. Other objective apart from the total completion time like the total tardiness were considered by Ribas *et al.* [28] for the blocking flowshop problem. In this work also the IG was used for the resolution. In view of all these previous achievements, applying IG to the BHFS-DDW

seems promising. Despite its simplicity, IG has given competitive results with many tested scheduling settings. As most metaheuristics, IG has some parameters that must be calibrated to achieve a high performance. Besides, it does not need specific knowledge of the studied problem. We will refer to the proposed IG as HIGT.

As for most iterated metaheuristics, the IG begins with an initial solution. After that, we have to apply four steps. This initial solution is followed by the local search. The obtained solution is considered to start the method process. In fact, the first step is the deconstruction phase from which a prefixed number of jobs (random jobs) are removed. In the second step, which is called the construction step, the removed jobs are reinserted into the solution following the greedy heuristic. The third step is the local search in which the solution can be improved. Finally, in the last step, the method will decide to consider or not the incumbent solution. This 4th operator is the acceptance criterion. For the initial solution, we have selected a quick and logical heuristics while taking into account some characteristics of our problem where the objective function which deals with earliness-tardiness uses two heuristics. In the first one, jobs are sorted in an increasing order according to their due dates. This dispatching rule is well known as the earliest due date (EDD) gives a permutation of jobs according: $\pi = \{\pi_1, \pi_2, \ldots, \pi_n\}$ where $R^+_{\pi_{(j)}} < R^+_{\pi_{(j+1)}}$, $j = 1, \ldots, n-1$ and $\pi_{(j)}$ indicate the job occupying the jth location in the sequence.

Then, the second heuristic tries to consider the difference between the the sum of the processing time for each job j and R^+. After calculating this amount, jobs are sorted according to the following rule $R^+_{\pi_{(j)}} - \sum_{i=1}^{m} Pi \cdot \pi_{(j)} \leq R^+_{\pi_{(j)}} - \sum_{i=1}^{m} Pi \cdot \pi_{(j+1)}$, $j = 1, \ldots, n-1$. Where $\pi_{(j)}$ specifies the work which in the permutation occupies the jth position. This rule, which has been recently considered by Pan *et al.* [22] is defined as the overall slack time (OSL). These two heuristics are very quick, needs just $O(n \log(n))$ steps and able to produce a high solution. After calculating the two given permutations, we keep the best one as an initial solution for our method.

4.1. Local search

In this method, the local search is applied in two ways. The first one is just after initialization and the second one is inside the main loop of the IG. The objective here is to improve the solution by the use of two neighbourhoods, which are the insertion and the interchange both of which are explored inside a variable neighbourhood descent (VND), which is a basic variant of the variable neighbourhood search (VNS). As depicted in the VND Algorithm 1, it starts from an initial solution to explore one among the neighbourhood structures. Starting with the insertion, we randomly select a job and reinsert it in another random position. If the solution is improved, the procedures will be repeated, otherwise, the VND procedure moves to the next neighbourhood structure, which is the interchange. The VND with the insertion and interchange as neighborhood structure was used to solve several scheduling problems such as: Pan *et al.* [22], Zhang *et al.* [37]. This operator has as parameter the maximum number of iteration. This parameter needs to be calibrated.

Algorithm 1: VND.

```
loop := 1;
while loop < loop_max do
    N_counter := 1
    while N_counter < 3 do
        if N_counter = 1 then π' = Insertion movement(π)
        if N_counter = 2 then π' = Interchange movement(π)
        if TWET_dw(π') < TWET_dw(π) then
        π = π'
        N_counter := 1
        else N_counter := N_counter + 1
    end
    loop := loop + 1;
end
```

4.2. Destruction reconstruction

In the IG algorithm, there are two fundamental procedures, which are the destruction and the construction. The first phase is based on the original work of Ruiz and Stützle [29], which starts by randomly extracting d jobs from the partial solution of $Seq1$ and putting them in a new sub-sequence $Seq2$ of d jobs, knowing that parameter d has to be calibrated later. Once the d jobs are removed, the next step consists in the reconstruction phase in the original way of this operator used in Ruiz and Stützle [29]. All removed jobs are reinserted one by one in all possible positions of the partial solution $Seq1$ to form a new complete sequence. At each insertion, there is an evaluation of the sequence in order to select the best position for every job. Let's us consider that we have a sequence of 50 jobs and we have to remove 4 jobs as a parameter d. Here, we have $n - d + 1$ possible insertion and therefore an evaluation for each job, means that the number is equal to 47 * 4 = 188. However, this looks a very complicated and expensive step especially when we deal with large instances. In general, this can be the most disadvantage of the iterated greedy method. Therefore, we have proposed in this work a new reconstruction process that works as follows: once the d jobs are removed from the partial sequence, the reconstruction phase starts according to a new insertion rule that's deals 8 will all removed jobs. Then, we insert a job into the first possible insertion position P_i where $i = 0$ and test the temporary solution (TempFit). If there is an improvement, we move to the next possible insertion position $i + K$ (here, the increment factor $K = 1$), however, if there is a deterioration of the temporary fitness, we try to change the research space and move to insert our job into another locality of the sequence. Indeed, this skipping is simply the increase of the increment factor $K{+}{+}$. Therefore, once we have an improvement of the Temp Fit, we return to the normal loop of insertion position by putting $K = 1$ again. This operation is repeated until the assignment of all $Seq2$ job list. In this way, there is no need to insert the selected jobs in all possible positions in order to avoid a very high amount of calculations, especially with large instances. Algorithm 2 presents the pseudo-code of this second phase. In this operator, we have two parameters that have to be calibrated, such as the destruction size d and the threshold of no-improvement S.

4.3. Example of the destruction reconstruction

The initial job sequence:

0	1	2	3	4	5	6	7	8	9

$Seq1$

Destruction
Lets $d = 3$ and the threshold of no-improvement $S = 2$, we have the two job sequence:

0	3	8

$Seq2$

1	2	4	5	6	7	9

$Seq1$

Construction
The position in which we will insert the job pos = 0.
The increment factor $K = 1$.
The counter of no improvement no-imp = 0 and bestFit = Max−int.

(1) We start by the job 0.

We insert the job 0 in the first position *pos* then we have $Seq1$:

0	1	2	4	5	6	7	9

$Seq1$

Let's consider Fit ($Seq1$) < Int−Max and then we have an improvement. We update bestFit. No-imp = 0.

Algorithm 2: Construction.

```
1  Input Seq1, ..., Seq2, ..., S, ..., d
2  Output BestSeq, ..., BestFit
3  TempFit := INTMAX;
4  N := Seq2.size
5  for i := 1 to N do
6  |  Q := Seq1.size
7  |  j := 0
8  |  K := 1
9  |  No-imp := 0
10 |  Job := Seq2[i]
11 |  while j < Q+1 do
12 |  |  Insert Job in Seq1 on the position j
13 |  |  Fit := f(Seq1)
14 |  |  if Fit < TempFit then
15 |  |  |  TempFit := Fit
16 |  |  |  BestSeq := Seq1
17 |  |  |  K := 1
18 |  |  |  No-imp := 0
19 |  |  end
20 |  |  else
21 |  |  |  K := K + 1
22 |  |  |  No-imp := 0
23 |  |  end
24 |  end
25 |  j := j + K
26 end
```

$K = 1$. And we move to insert job 0 in the next possible position $\text{pos} + K$ $(0 + 1)$ to obtain:

1	0	2	4	5	6	7	9	$Seq1$

We consider that we have not an improvement then: no-imp = 1.
We insert the job in the next position $\text{pos} + K$:

1	2	0	4	5	6	7	9	$Seq1$

We consider that we haven't an improvement so no-imp = 2; Here we have no-imp equal to the threshold 2 then, we increase the increment factor K $(K{+}{+})$ We move to insert our job (job 0) in the position $\text{pos} + K$:

1	2	4	5	0	6	7	9	$Seq1$

Now, we consider that we have an improvement then We update bestFit.
$K = 1$ no-imp = 0 and we move to insert the job in the next possible position $\text{pos} + K$:

1	2	4	5	6	0	7	9	$Seq1$

We will always respect that we have $n - d + 1$ possible position, then continue in the same way for all jobs of $Seq2$.

4.4. Acceptance criterion

At each loop, three phases, destruction, reconstruction and VND, are carried out, which could give a new solution different from the incumbent one. In fact, the new schedule might be worse or better than the initial solution. Accepting only better solution in any algorithm leads to local optimum, which might be an obstacle for convergence. In the initial work of Ruiz and Stützle [29], a simple simulated annealing acceptance criterion with invariable temperature T is applied; where T is a parameter that needs to be calibrated. Most IG in scheduling problems have considered this acceptance criterion such as Ruiz and Stützle [29], Ruiz *et al.* [31] and Tasgetiren *et al.* [33]. The common point in all these works is that T is not statically significant in calibration tests. In this work, the tournament selection is proposed as an acceptance criterion, which is a simple acceptance criterion used with genetic algorithm methods to solve different scheduling problems, such as Lei [16] and Pezzella *et al.* [23]. This approach is a based list method, therefore at each iteration, the resolution method gave a new solution, in this case, two possibilities are probable, the first one is where the new solution improves the results, here we talk about a new best sequence. We take it as a new incumbent solution and we move to the next iteration. Therefore, this is not always the case and, in some iteration, the worst solution is obtained. In this case, the acceptance criterion has to decide which sequence should be considered as the incumbent solution. The basic principle of tournament selection is simple. In a dynamic list R, we insert all sequence that doesn't improve the results known that the maximum size of this list the number of iterations. At each time, a Ts random individuals are selected from the list to participate in a tournament and the winner will be considered as a new incumbent solution. In the literature, the tournament selection needs a calibration step in order to choose the best tournament size Ts. The pseudo-code of tournament selection operator is given in Algorithm 3.

Algorithm 3: Acceptance criterion.

```
Input Seq3, BestFit, BestSeq, Ts
Output Seq
if Fit(Seq3) < BestFit then
    Clear R and FitList
    Insert Seq3 into R
    Insert Fit(Seq3) into FitList
    BestFit := Fit(Seq3)
end
else
    Affix Seq3 to R
    Insert Fit(Seq3) into FitList
    Avg_fit := Average(FitList)
    if TS < List_size(R) then
        Seq := Random sequence from R
    end
    else
        Aux_L:= Select TS random sequences from R
        Seq := Best sequence in Aux_L
    end
end
```

5. Computational evaluation and statistical experimentation

This section demonstrates in detail the used benchmark and the competing methods that have been adapted and re-implemented as well as all their calibrated parameters. Moreover, the proposed iterated greedy tournament (IGT) is calibrated in order to choose the most suitable parameter.

5.1. Benchmark, competing methods and experimental setting

The studied problem (BHFS-DDW) has never been considered in the literature. Therefore, we tried to generate a new Benchmark of small and large instances. In this way, many parameters need to be controlled. Firstly, as in all HFS problems, we need a number of jobs n a number of stages g, and a number of machines per stage mi. As in all scheduling problems with due date or due window, we need to generate a due date for each job but to achieve this, two factors are required; a tardiness factor (F) and a due date range (R). Finally, the last factor is the width of the due window (W). As in scheduling literature, the processing times are generated in the range of U [1, 99] and earliness and tardiness weights in the range U [1, 9]. Following Pan *et al.* [22] and Potts and Van Wassenhove [24], the due dates are uniformly distributed as:

$$d_j = \max(0, U[[P(1 - T - R/2)], [P(1 - T + R/2)]]$$

P is the makespan lower bound for the classical permutation hybrid flowshop scheduling problem. To define the lower bound we have used the existing state-of-the-art bounds for the HFS, which were suggested by Hidri and Haouari [11]. In their paper, the authors presented 9 lower bounds, the maximum of which is selected as a P value. In fact, using the due date, we could generate the due window as: $d_j^- = \max(0, d_j - d_j \cdot H/100)$ and $d_j^+ = \max(0, d_j + d_j \cdot H/100)$, where $H = U[1, W]$.

On the other hand, for small instances, the used combinations of factors are: $n = (10, 15, 20)$, $g = (2, 3, 4)$, $m_i = (2, 3)$, $T = (0.2, 0.4)$, $R = (0.6, 1.2)$ and $W = (10, 20)$, where for every seed of parameters, 5 replicates are generated, which makes the total be $3 \cdot 3 \cdot 2 \cdot 2 \cdot 2 \cdot 2 \cdot 5 = 720$ small instances. However, for large instances, we kept the same combinations of T, R and W. Only the value of n, g and mi are changed to be: $n = (50, 100)$, $g = (5, 10)$, $mi = (5, 10)$ then, with 5 replicates, we have $2 \cdot 2 \cdot 2 \cdot 2 \cdot 2 \cdot 2 \cdot 5 = 320$ large instances. In the same way, we generated 72 small instances and 32 large instances for the calibration step. In the literature of HFS, several methods are used with many variants among which we selected 4 from the best methods: the genetic algorithm (GA) of Kahraman *et al.* [13], the artificial bee colony (ABC) of Pan *et al.* [20], the improved simulated annealing (HSA) of Naderi *et al.* [18], the artificial immune system (AIS) of Engin and Döyen [5]. the two methods based on iterated local search ILS_{VNS} and iterated greedy IG_{VNS} of Pan *et al.* [21] and finally the auto-designed algorithm (ADA) of Alfaro-Fernández *et al.* [1]. All these seven methods are carefully adapted and re-implemented for the studied problem. Moreover, all the compared methods have been coded in C++ language using the software visual studio (2019). On the other hand, the calibration of all methods uses windows 7 virtual machines with 8 GB of RAM. As for the final experiments, they are performed on a windows 7 virtual machine with 16 GB of RAM.

5.2. Calibration of methods

In fact, all competing methods as well as the proposed IGT are carefully calibrated. Starting with the proposed method, we notice that the first factor is the maximum number of iterations in the VND local search called loop size, this parameter is tested at five levels, $(50, 100, 200, 300, 400)$. And the second parameter is the destruction size d, which is tested at three levels $(2, 3, 4)$. Then, the third one is the threshold S tested at 4 levels, $(1, 2, 3, 4)$ and finally, the last parameter is the tournament size Ts, which is also tested at four levels, $2, 3, 4, 5$. Therefore, the improved IGT has 240 combinations each of which is run 5 times with every instance. On the other hand, the number of experiments is $240 \cdot 5 \cdot 72 = 86\,400$ for the small instances and $240 \cdot 5 \cdot 32 = 38\,400$ for large instances. One from the best methodology used for calibration, is the design of experiments (DOE). This method is employed in our work where the relative deviation index (RDI) is the considered response variable. while the RDI = (Meth.sol − Best.sol)/(Best.sol − Worst.sol) * 100. Best.sol and Worst.sol are respectively the best and the worst solutions given by all replications of one setting. Moreover, since the value of RDI is between 0 and 100, it is therefore possible that with some instances, the best solution might be equal to the worst one. In this case, the RDI is replaced by 0. Besides, the analysis of the variance (ANOVA) is employed to examine all the obtained results, however this powerful technique has been widely used in the last few years to study and calibrate methods in the scheduling literature. Many other techniques of method calibration are explained in

TABLE 2. Possible value of each parameter.

Parameter	Setting
VNS loop size (Len_M)	50, 100, 200, 300, 400
Destruction size (d)	2, 3, 4
Threshold (S)	1, 2, 3, 4
Tournament size (Ts)	2, 3, 4, 5

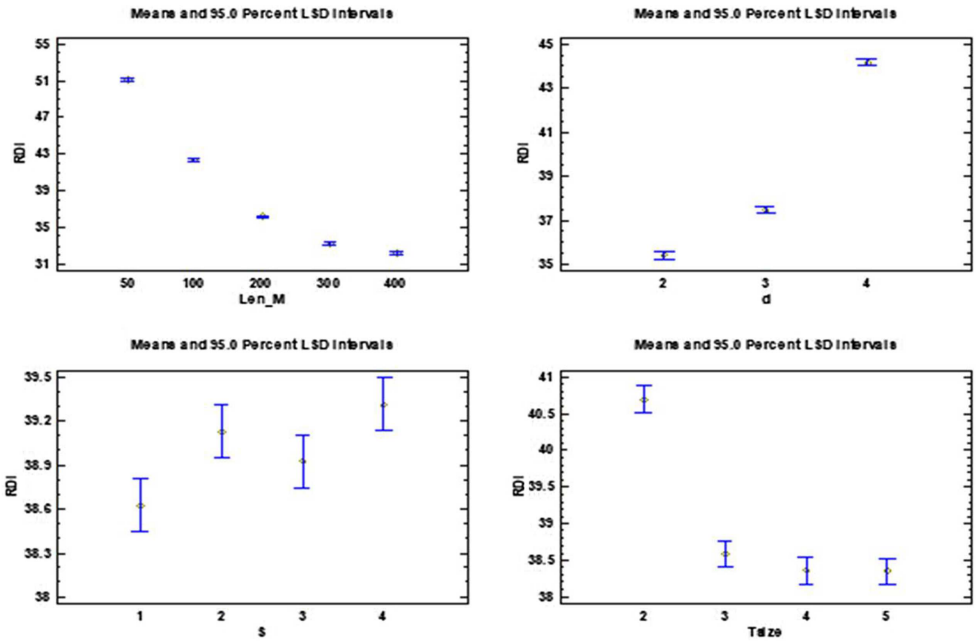

FIGURE 2. Means plots for all the factors in the ANOVA HIGT calibration with large instances. All means have Tukey's honest significant difference (HSD) 95% confidence intervals.

Bartz-Beielstein *et al.* [3]. In fact, all the methods and the combination of the parameters considered in this paper have a stopping criterion represented by fixed CPU time. Regarding calibration, this CPU time is equal to 30 ms with an adjustment by the size of instances while large instances need more time than small instances. For instance, Figure 2 shows the calibration of all the parameters for the proposed IG method with a large Benchmark. In fact, all methods are calibrated using the same tool. Taking the constraint of insufficient space, only the detailed calibration of the proposed method with the large instances is illustrated here. After finished all experiments, we started the process by calculate the RDI value of each set of parameters and then the test ANOVA is applied. Table 2 presents all possibles value for each parameter of the proposed IG.

5.3. Results evaluation and statistical analyses

In this final evaluation, the benchmark testing, which includes 720 small and 320 large instances, is used to prove the effectiveness of the proposed HIGT method. In fact, all the calibrated methods are run 5 times with every instance. To have a more extensive image of execution, we utilize three distinctive stopping times as: $p \cdot n \cdot g$ seconds knowing that $p = 30, 60, 90$, this is will give 0.12 s for a small instance of 10 jobs and 2 stages if we consider $p = 60$ while 60 s for an instance of 100 jobs and 10 stages. All methods are coded with C++ using

TABLE 3. Average relative deviation index (RDI) for small instances. The best values of RDI are in bold.

CPU	Jobs	Stages	GA	AIS	ABC	HSA	IG_{VNS}	ILS_{VNS}	ADA	HIGT
30	10	2	20.68	13.89	0.25	3.35	**0.00**	**0.00**	0.25	**0.00**
		3	20.90	15.25	0.35	4.18	**0.00**	**0.00**	**0.00**	**0.00**
		4	20.38	21.38	0.21	2.10	0.02	0.06	**0.00**	0.04
	15	2	31.40	29.02	3.85	3.99	1.29	**1.15**	1.38	1.16
		3	37.63	34.23	6.19	5.09	0.56	0.74	1.25	**0.30**
		4	39.29	32.48	5.39	6.15	1.29	1.21	1.35	**0.86**
	20	2	39.24	31.76	12.89	5.47	3.65	4.06	3.85	**2.04**
		3	40.61	35.61	10.80	6.29	5.41	5.68	4.08	**2.70**
		4	47.38	41.26	14.76	7.79	5.45	7.19	5.21	**2.82**
Average			33.06	28.32	6.08	4.94	1.96	2.23	1.93	**1.10**
60	10	2	22.69	14.09	0.25	1.56	**0.00**	**0.00**	0.25	**0.00**
		3	17.29	15.20	0.35	2.55	**0.00**	**0.00**	**0.00**	**0.00**
		4	17.35	18.10	0.21	1.19	**0.00**	0.04	**0.00**	0.02
	15	2	31.97	25.72	3.13	1.71	0.78	**0.49**	0.87	0.51
		3	33.97	32.94	5.23	3.01	0.41	0.57	0.69	**0.03**
		4	36.82	32.48	4.69	4.05	0.59	0.90	0.63	**0.44**
	20	2	36.59	29.04	10.63	3.46	2.77	2.80	2.40	**1.07**
		3	38.98	29.92	9.04	4.53	4.34	4.19	2.68	**1.58**
		4	45.32	38.41	12.40	5.12	3.93	5.62	3.24	**1.92**
Average			31.22	26.21	5.10	3.02	1.42	1.62	1.20	**0.62**
90	10	2	19.66	13.83	0.25	1.12	**0.00**	**0.00**	**0.00**	**0.00**
		3	16.08	13.83	0.07	1.96	**0.00**	**0.00**	**0.00**	**0.00**
		4	19.36	17.76	0.21	0.93	**0.00**	0.02	**0.00**	0.02
	15	2	28.03	24.81	2.61	1.46	0.51	0.34	0.76	**0.16**
		3	34.25	31.96	4.28	2.04	0.22	0.45	0.10	**0.02**
		4	36.58	31.76	4.25	2.71	0.32	0.76	**0.19**	0.34
	20	2	38.02	28.60	9.59	2.23	2.22	2.36	1.84	**0.78**
		3	36.55	30.50	8.66	3.23	3.50	3.44	2.09	**1.17**
		4	44.08	36.46	11.31	3.58	3.55	4.96	2.48	**1.60**
Average			30.29	25.50	4.58	2.14	1.15	1.37	0.83	**0.45**
Tot. average			31.52	26.68	5.25	3.36	1.51	1.74	1.32	**0.73**

Microsoft visual studio 2019 which shares the same objective function evaluation. Therefore, the correlation is reasonable and the outcome is totally comparable. Moreover, each method is run 5 times with 720 small and 320 large instances while considering 3 different CPU times. We have $5 \cdot (720 + 320) \cdot 5 \cdot 3 = 78\,000$ experimental results, which will help to extract a very strong conclusion. The, the average RDI grouped by p, n and g is given in Table 3 for small instances and in Table 4 for large instances.

We noticed that GA and AIS have not provided good results with both large and small instances despite the calibration. Similarly, the relatively recent ABC has not given a good result especially with large instances whereas IG_{VNS}, ILS_{VNS} and ADA showed a relatively good performance. According to Table 3, ADA has an RDI close to the best results for the small benchmark with all CPU times. We also observed that, on average, the proposed HIGT gave better results for small instances with the three stopping times where the average RDI value is 0.73 for the proposed method while the nearest value is given by ADA of Alfaro-Fernández *et al.* [1] with an average RDI = 1.32. However, our approach shows a reduction of 44.69% against the closest RDI value. Knowing that the ADA method has recently proven its effectiveness with HFS scheduling problems. As with the small instances, our proposed approach has the best average RDI = 5.70 for the large benchmark. Always ADA is the closest competitor with an average RDI = 7.09 *i.e.* HIGT shows a reduction of 19.60%. Also, the recent

TABLE 4. Average relative deviation index (RDI) for large instances. The best values of RDI are in bold.

CPU	Jobs	Stages	GA	AIS	ABC	HSA	IG_{VNS}	ILS_{VNS}	ADA	HIGT
30	50	5	59.02	40.15	38.81	11.51	10.06	14.25	10.35	**7.29**
		10	58.82	42.20	48.07	10.33	10.02	14.35	10.28	**7.40**
	100	5	67.47	49.67	28.57	10.14	8.59	12.90	7.19	**6.40**
		10	67.53	48.46	53.60	9.89	9.88	14.51	**7.19**	7.48
Average			63.21	45.12	42.26	10.47	9.64	14.01	8.76	**7.14**
60	50	5	56.64	37.53	33.06	8.86	8.55	12.77	7.94	**5.56**
		10	54.79	39.36	40.68	7.55	8.93	12.59	8.00	**5.45**
	100	5	62.56	43.96	24.83	7.85	7.58	11.82	5.55	**4.89**
		10	64.25	43.96	47.09	7.87	8.71	13.22	**5.61**	5.70
Average			59.56	41.20	36.42	8.03	8.44	12.60	6.78	**5.40**
90	50	5	52.49	36.01	30.07	7.19	8.06	11.65	6.45	**4.51**
		10	51.24	37.94	37.22	5.94	8.16	11.56	6.77	**4.64**
	100	5	62.70	41.04	22.94	6.77	7.11	11.28	4.80	**4.15**
		10	60.96	41.95	44.05	6.59	8.18	12.40	**4.88**	4.91
Average			56.85	39.24	33.57	6.62	7.88	11.72	5.73	**4.55**
Tot. average			59.87	41.85	37.42	8.37	8.65	12.78	7.09	**5.70**

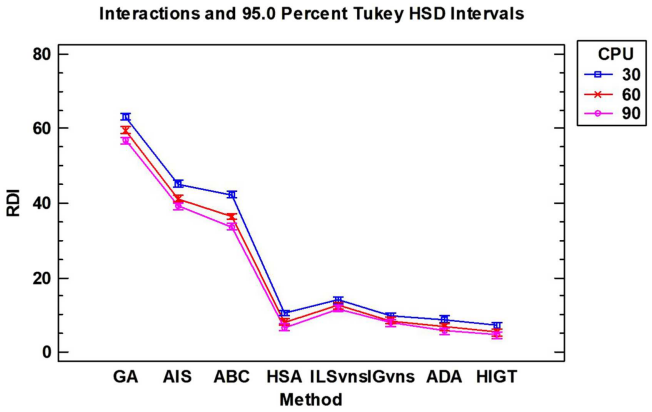

FIGURE 3. Interaction between the proposed methods and stopping criterion. All means with Tukey's honest significant difference (HSD) 95% confidence intervals. Large instances.

method IG_{VNS} showed a relative performance with an RDI = 8.65. While ILS_{VNS} hasn't proved its effectiveness with the large benchmark. Despite the calibration GA, ABC and AIS haven't shown good results. The results of all algorithms improve from CPU = 30 to 90, for example, HIGT improves 36.27% its average RDI when going from CPU = 30 to CPU = 90. In addition, the best competitor ADA improves 34.58% its average RDI moving from CPU = 30 to CPU = 90. If we summarize, our approach is able to beats a set of very recent methods developed for the same kind of problems. We have to note that the best competing method with the two studied benchmarks is the ADA. This latter is based on iterated greedy approach and here we mention the effect of the proposed destruction reconstruction operator which is the key of HIGT method. Figure 3 shows the interaction between our proposed HIGT and the seven competing methods with three CPU. As a matter of fact, what attracts attention in this plot is that the average RDI of HIGT with CPU = 60 is statically equivalent to the average RDI of the best competitor ADA with CPU = 90 which proves the effectiveness of our proposal.

6. Conclusions

This paper is devoted to studying the hybrid flow shop scheduling problem with basic blocking constraint. The considered objective function is the minimization of the total weighted earliness tardiness. This NP-hard problem is not present in literature to our best knowledge. Therefore, to resolve it, we have proposed an iterated greedy procedure. On the other hand, in order to check the effectiveness of our procedure, we have implemented seven competing methods well selected from the literature on the hybrid flow shop scheduling. After the calibration of all methods, a generated benchmark testing instances was used to analyze the results of the proposed method and compare them to those of competing methods. The results obtained show that the proposed HIGT is effective, since it produced the best results against competing methods despite the novelty of most of them. Moreover, many future research studies can be introduced to make the problem more realistic, by setting up the time or machine breakdowns, setup times and adaptation of assignment rule, which can be changed to be more adapted to the objective function. Furthermore, other modes of blocking could be considered to study new production systems.

Acknowledgements. We are grateful to the reviewers whose comments have helped to improve our paper. We would like to thank Pr. Ruben Ruiz García for his help and explanations.

References

[1] P. Alfaro-Fernández, R. Ruiz, F. Pagnozzi and T. Stützle, Automatic algorithm design for hybrid flowshop scheduling problems. *Eur. J. Oper. Res.* **282** (2020) 835–845.

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

[3] T. Bartz-Beielstein, M. Chiarandini, L. Paquete and M. Preuss, Experimental Methods for the Analysis of Optimization Algorithms. Springer (2010).

[4] A. Elmi and S. Topaloglu, A scheduling problem in blocking hybrid flow shop robotic cells with multiple robots. *Comput. Oper. Res.* **40** (2013) 2543–2555.

[5] O. Engin and A. Döyen, A new approach to solve hybrid flow shop scheduling problems by artificial immune system. *Future Gener. Comput. Syst.* **20** (2004) 1083–1095.

[6] L. Fanjul-Peyro and R. Ruiz, Iterated greedy local search methods for unrelated parallel machine scheduling. *Eur. J. Oper. Res.* **207** (2010) 55–69.

[7] J.M. Framinan, R. Leisten and R.R. García, Manufacturing scheduling systems. In: An Integrated View on Models, Methods and Tools (2014) 51–63.

[8] J. Grabowski and J. Pempera, Nowy algorytm tabu search dla zagadnienia kolejnościowego przepływowego. *Automatyka/Akademia Górniczo-Hutnicza im. Stanisława Staszica w Krakowie* **3** (1999) 125–133.

[9] R.L. Graham, E.L. Lawler, J.K. Lenstra and A.R. Kan, Optimization and approximation in deterministic sequencing and scheduling: a survey. *Ann. Discrete Math.* **5** (1979) 287–326.

[10] N.G. Hall and C. Sriskandarajah, A survey of machine scheduling problems with blocking and no-wait in process. *Oper. Res.* **44** (1996) 510–525.

[11] L. Hidri and M. Haouari, Bounding strategies for the hybrid flow shop scheduling problem. *Appl. Math. Comput.* **217** (2011) 8248–8263.

[12] X.-L. Jing, Q.-K. Pan, L. Gao and Y.-L. Wang, An effective iterated greedy algorithm for the distributed permutation flowshop scheduling with due windows. *Appl. Soft Comput.* **96** (2020) 106629.

[13] C. Kahraman, O. Engin, I. Kaya and M. Kerim Yilmaz, An application of effective genetic algorithms for solving hybrid flow shop scheduling problems. *Int. J. Comput. Intell. Syst.* **1** (2008) 134–147.

[14] A. Khare and S. Agrawal, Scheduling hybrid flowshop with sequence-dependent setup times and due windows to minimize total weighted earliness and tardiness. *Comput. Ind. Eng.* **135** (2019) 780–792.

[15] G. Lebbar, I. El Abbassi, A. El Barkany, A. Jabri and M. Darcherif, Solving the multi objective flow shop scheduling problems using an improved nsga-ii. *Int. J. Oper. Quant. Manage.* **24** (2018) 211–230.

[16] D. Lei, Co-evolutionary genetic algorithm for fuzzy flexible job shop scheduling. *Appl. Soft Comput.* **12** (2012) 2237–2245.

[17] J.V. Moccellin, M.S. Nagano, A.R.P. Neto and B. de Athayde Prata, Heuristic algorithms for scheduling hybrid flow shops with machine blocking and setup times. *J. Braz. Soc. Mech. Sci. Eng.* **40** (2018) 40.

[18] B. Naderi, M. Zandieh, A.K.G. Balagh and V. Roshanaei, An improved simulated annealing for hybrid flowshops with sequence-dependent setup and transportation times to minimize total completion time and total tardiness. *Expert Syst. Appl.* **36** (2009) 9625–9633.

[19] M.H. Newton, V. Riahi, K. Su and A. Sattar, Scheduling blocking flowshops with setup times via constraint guided and accelerated local search. *Comput. Oper. Res.* **109** (2019) 64–76.

[20] Q.-K. Pan, L. Wang, K. Mao, J.-H. Zhao and M. Zhang, An effective artificial bee colony algorithm for a real-world hybrid flowshop problem in steelmaking process. *IEEE Trans. Autom. Sci. Eng.* **10** (2012) 307–322.

[21] Q.-K. Pan, L. Gao, X.-Y. Li and K.-Z. Gao, Effective metaheuristics for scheduling a hybrid flowshop with sequence-dependent setup times. *Appl. Math. Comput.* **303** (2017) 89–112.

[22] Q.-K. Pan, R. Ruiz and P. Alfaro-Fernández, Iterated search methods for earliness and tardiness minimization in hybrid flowshops with due windows. *Comput. Oper. Res.* **80** (2017) 50–60.

[23] F. Pezzella, G. Morganti and G. Ciaschetti, A genetic algorithm for the flexible job-shop scheduling problem. *Comput. Oper. Res.* **35** (2008) 3202–3212.

[24] C.N. Potts and L.N. Van Wassenhove, A decomposition algorithm for the single machine total tardiness problem. *Oper. Res. Lett.* **1** (1982) 177–181.

[25] M. Pranzo and D. Pacciarelli, An iterated greedy metaheuristic for the blocking job shop scheduling problem. *J. Heurist.* **22** (2016) 587–611.

[26] E. Rashidi, M. Jahandar and M. Zandieh, An improved hybrid multi-objective parallel genetic algorithm for hybrid flow shop scheduling with unrelated parallel machines. *Int. J. Adv. Manuf. Technol.* **49** (2010) 1129–1139.

[27] I. Ribas, R. Companys and X. Tort-Martorell, An iterated greedy algorithm for the flowshop scheduling problem with blocking. *Omega* **39** (2011) 293–301.

[28] I. Ribas, R. Companys and X. Tort-Martorell, An iterated greedy algorithm for solving the total tardiness parallel blocking flow shop scheduling problem. *Expert Syst. Appl.* **121** (2019) 347–361.

[29] R. Ruiz and T. Stützle, A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem. *Eur. J. Oper. Res.* **177** (2007) 2033–2049.

[30] R. Ruiz and J.A. Vázquez-Rodríguez, The hybrid flow shop scheduling problem. *Eur. J. Oper. Res.* **205** (2010) 1–18.

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

[32] T.J. Sawik, A scheduling algorithm for flexible flow lines with limited intermediate buffers. *Appl. Stoch. Models Data Anal.* **9** (1993) 127–138.

[33] M.F. Tasgetiren, D. Kizilay, Q.-K. Pan and P.N. Suganthan, Iterated greedy algorithms for the blocking flowshop scheduling problem with makespan criterion. *Comput. Oper. Res.* **77** (2017) 111–126.

[34] W. Trabelsi, C. Sauvey and N. Sauer, A genetic algorithm for hybrid flowshop problem with mixed blocking constraints. In: *IFAC Conference on Manufacturing, Modelling, Management and Control* (2013).

[35] K. Yuan, N. Sauer and C. Sauvey, Application of em algorithm to hybrid flow shop scheduling problems with a special blocking. In: 2009 IEEE Conference on Emerging Technologies & Factory Automation. IEEE (2009) 1–7.

[36] Q. Zeng and Z. Yang, A hbrid flow shop scheduling model for loading outbound containers in container terminals. In: *Proceedings of the Eastern Asia Society for Transportation Studies* Vol. 6 (*The 7th International Conference of Eastern Asia Society for Transportation Studies, 2007*). Eastern Asia Society for Transportation Studies (2007) 381.

[37] Y. Zhang, X. Liang, W. Li and Y. Zhang, Hybrid flow shop problem with blocking and multi-product families in a maritime terminal. In: 2013 10th IEEE International Conference on Networking, Sensing and Control (ICNSC). IEEE (2013) 59–64.