MILP-based local search procedures for minimizing total tardiness in the No-idle Permutation Flowshop Problem

Computers & Operations Research 146 (2022) 105862

Contents lists available at ScienceDirect

Computers and Operations Research

journal homepage: www.elsevier.com/locate/cor

MILP-based local search procedures for minimizing total tardiness in the No-idle Permutation Flowshop Problem

Andrea Balogh [a], Michele Garraffa [a,b,*], Barry O'Sullivan [a,b], Fabio Salassa [c]

[a] Confirm Centre for Smart Manufacturing, School of Computer Science and IT, University College Cork, Ireland
[b] Insight SFI Research Centre for Data Analytics, School of Computer Science and IT, University College Cork, Ireland
[c] Dipartimento di Ingegneria Gestionale e della Produzione (DIGEP), Politecnico di Torino, Torino, Italy

ARTICLE INFO

Keywords:
Scheduling
Flowshop
No-idle
MILP
Hybrid heuristics

ABSTRACT

We consider the No-idle Permutation Flowshop Scheduling Problem (NPFSP) with a total tardiness criterion. We present two Mixed Integer Linear Programming (MILP) formulations based on positional and precedence variables, respectively. We study six local search procedures that explore two different neighborhoods by exploiting the MILP formulations. Our computational experiments show that two of the proposed procedures strongly outperform the state-of-the-art metaheuristic. We update 63% of the best known solutions of the instances in Taillards' benchmark, and 77% if we exclude those instances for which we proved that the previous best known solutions are optimal.

1. Introduction

Flowshop problems are a well-studied class of scheduling problems, where each job in a certain set must be processed by an ordered sequence of machines. A member of the above-mentioned class is the so called No-idle Permutation Flowshop Scheduling Problem (NPFSP), where two additional constraints are included. First, the jobs are processed by all the machines in the same order. This is usually motivated by the structure of the production floor, where the different pieces, or materials, share the same production route. Second, each machine must process the full set of jobs without interruption (the no-idle constraint) once it starts processing the first job. This constraint can arise in practice for many reasons. For example, it can be motivated by physical constraints on the production process that preclude machines from stopping and resuming their work, in which cases schedules that contain idle times are simply unfeasible. On the other hand, machines may be expensive resources that generate significant costs from the time they begin to the time they end a set of jobs. In this latter case, schedules with idle-time can be prohibitively costly and the decision-maker might wish to exclude them entirely.

The earliest research on the NPFSP dates back to 1982 when (Adiri and Pohoryles, 1982) focused on the case involving two machines and the minimization of the sum of completion times. Numerous other studies have been conducted since then, in particular the NPFSP with makespan minimization has received the greatest attention. The problems can be denoted as $Fm|prmu, noidle|C_{max}$ using Graham's three-field notation (Graham et al., 1979). Detailed surveys of $Fm|prmu, noidle|C_{max}$ appear in the literature (Goncharov and Sevastyanov, 2009; Ruiz et al., 1970). The problem is solvable in polynomial time in the two-machine case (Adiri and Pohoryles, 1982), while it is NP-hard for three or more machines (Baptiste and Hguny, 1997). Other polynomial-time solvable cases arise whenever certain dominance relationships among machines hold (Wang and Xia, 2005). A variety of solution approaches have been developed to tackle this problem, including both exact approaches and heuristics. The range of exact approaches is limited. Some initial studies (Vachajitpan, 1982) and Baptiste and Hguny (1997) described branch-and-bound algorithms, which were able to solve small instances to optimality. Only recently, a Benders decomposition (Bektaş et al., 2020) improved those results. It is designed to tackle a generalization of the NPFSP, called the MNPFSP, where each machine may or may not respect the no-idle constraint. This approach was able to solve instances with up to 500 jobs and 5 machines to optimality.

Metaheuristic approaches for $Fm|prmu, noidle|C_{max}$ are more numerous. The first heuristic study was performed in 1986 (Woollam, 1986). Subsequent approaches were developed such as Saadani et al. (1999), Kalczynski and Kamburowski (2005) and Baraz and Mosheiov (2008). A hybrid discrete particle-swarm optimization (HDPSO) and a differential evolution (DE) metaheuristic were presented in Pan and Wang (2008b) and Pan and Wang (2008a), respectively. A comprehensive computational assessment of fourteen different heuristic approaches, developed

* Corresponding author.
E-mail addresses: a.balogh@cs.ucc.ie (A. Balogh), michele.garraffa@cs.ucc.ie (M. Garraffa), b.osullivan@cs.ucc.ie (B. O'Sullivan), fabio.salassa@polito.it (F. Salassa).

https://doi.org/10.1016/j.cor.2022.105862
Received 20 July 2021; Received in revised form 28 February 2022; Accepted 28 April 2022
Available online 17 May 2022

prior to 2009, was performed in Ruiz et al. (1970). The results revealed that the best performing metaheuristics at that time were (Ruiz et al., 1970) and Rad et al. (2009), which are iterated-greedy approaches (IGAs). In subsequent work a hybrid discrete differential evolution algorithm (Deng and Gu, 2012), a variable iterated greedy algorithm with differential evolution (Fatih Tasgetiren et al., 2013), and an invasive weed algorithm (Zhou et al., 2014) achieved better performance than the previous approaches. In 2017, Shao et al. (2017) proposed a memetic algorithm with a hybrid node and edge histogram that improved the results obtained on 89 instances of the dataset used in Ruiz et al. (1970). Subsequently, two general variable neighborhood search algorithms were proposed (Shen et al., 2019) and Öztop et al. (2020). Both approaches used simple operators like one-job insertion and swap for shaking purposes, while more advanced metaheuristics, such as IGA, were used to intensify the search. The approach in Öztop et al. (2020) also includes a Q-learning mechanism to determine the parameters of the algorithm and was able to improve 104 best known solutions out of 250 instances. Very recently, another work (Öztop et al., 2022) by the authors of Öztop et al. (2020) tackled the same problem. The goal of that work was twofold. First, the authors propose two MILP formulations and a CP formulation of the problem. The computational experiments show that the CP formulation, based on the CP optimizer global constraints and structures, outperforms the MILP formulations on the small instances of Ruiz's dataset. Second, the authors propose an iterated greedy and an iterated local search algorithm with restart and learning mechanisms, which are currently best performing heuristics for $Fm|prmu, noidle|C_{max}$.

Some recent studies focus on the NPFSP in which total tardiness is minimized, denoted as $Fm|prmu, noidle|\sum_j T_j$, where T_j is the tardiness of job j. To the best of our knowledge, the studies about this problem have focused on the development of efficient metaheuristics, while there are no ad-hoc exact approaches (Tasgetiren et al., 2011, 2013). A bi-population-based approach combining a population-based strategy and a local search procedure based on a one-insertion neighborhood has been introduced (Shen et al., 2014). The approach dominated previously available heuristics reported in Tasgetiren et al. (2011, 2013). Later, a hybrid discrete teaching-learning-based meta-heuristic was proposed (Shao et al., 2018) that outperformed previous approaches on the main two benchmark instance sets (Taillard's set (Taillard, 1993) and Ruiz's set (Ruiz et al., 1970)). Recently, an iterated greedy approach has been proposed (Riahi et al., 2020), which enabled better solutions to be found for approximately 50% of the best known results related to Taillard's set, as well as outperforming all previous approaches on the same data set. The main two ingredients of such an approach are a new local search procedure based on an insertion move and a new destruction–construction phase.

Other versions of the NPFSP have been studied recently. Total flow time as the problem objective has been considered (Fatih Tasgetiren et al., 2013; Rossi and Nagano, 2019). The impact of sequence dependent setup times has been considered (Rossi and Nagano, 2019, 2020). Tasgetiren et al. (2019) deals with an energy efficient version of the problem, where two objectives are considered: the makespan and the total energy consumption. A distributed version of the problem has been considered (Ying et al., 2017; fang Chen et al., 2019).

The objective of this paper is to propose local search procedures based on Mixed Integer Linear Programming (MILP) for $Fm|prmu, noidle|\sum_j T_j$ and demonstrate their effectiveness. MILP-based heuristics, or *matheuristics*, have been successfully applied to efficiently solve a variety of combinatorial problems, including scheduling problems such as project scheduling (Toffolo et al., 2016), energy scheduling (Della Croce et al., 2017), flowshop scheduling (Della Croce et al., 2019; Ta et al., 2018). We consider a matheuristic framework where half of the runtime is spent on executing the state-of-the-art metaheuristic approach (Riahi et al., 2020) and the other half is devoted to run one of the MILP-based local search procedures on the best solution found at the previous step. These heuristics are compared with the state-of-the-art (Riahi et al., 2020) over Taillard's dataset.

The remainder of the paper is organized as follows. Section 2 describes two MILP formulations for $Fm|prmu, noidle|\sum_j T_j$: one is a slight modification of the one proposed in Pan and Ruiz (2014), originally designed for $Fm|prmu, noidle|C_{max}$, while the other is a contribution of this work. Section 3 describes a fast procedure to compute the total tardiness in the exploration of the one-opt neighborhood, which is based on the speed up proposed in Ding et al. (2015) for $Fm|nowait|\sum_j T_j$. We encoded such a procedure in our reimplementation of the approach in Riahi et al. (2020). Section 4 presents six different MILP-based local search procedures and the overall structure of the proposed approach. Section 5 is divided in three subsections. First, we perform a computational assessment of the MILP models when they are used to solve small instances to optimality. Second, we compare our hybrid heuristic procedures with the reimplementation of the state-of-the-art metaheuristic. Finally, we use the MILP solver to compute lower bounds, we compare the solutions obtained in our experiments with the best known solutions reported in Riahi et al. (2020), and we provide the corresponding optimality gaps. The paper is concluded by Section 6, which provides some conclusions and future research directions.

2. Problem formulations

We first provide a detailed definition of $Fm|prmu, noidle|\sum_j T_j$. Let J be a set of jobs that are all available at the initial time instant, and let M be an ordered set of machines. The cardinality of these sets are indicated with $n = |J|$ and $m = |M|$, respectively. Each job $j \in J$ requires a time $p_{q,j}$ to be processed by a machine $q \in M$ and has a due date d_j with respect to its completion time on the last machine. Since it is a flowshop problem, each job can start being processed by a machine $q \in M$ when completed by the previous one $q-1 \in M$. All the machines process the jobs one at a time, in the same order, and work continuously without idle intervals. We assume that there is a one-to-one correspondence between a feasible solution and a jobs permutation, indicating the order in which the jobs are processed by each machine. This means that, given a jobs permutation, the starting time of the first job on each machine is adjusted such that no-idle times are required and the jobs are scheduled as early as possible. The set of positions where a job can be scheduled is indicated with P and $n = |J| = |P|$, since the number of positions in a permutation is equal to the number of jobs. The problem involves scheduling the jobs such that the total tardiness $\sum_{j \in J} T_j$ is minimized. The tardiness of each job is defined as $T_j = \max\{0, C_j - d_j\}$, where C_j is the completion time of a job $j \in J$ on the last machine.

Sections 2.1 and 2.2 present two MILP formulations for the problem, based on two different types of decision variables.

2.1. A MILP formulation based on positional variables

The first formulation is denoted as `MILP_Form_Pos`, since it is based on positional variables. This formulation is a slight adaptation of the one in Pan and Ruiz (2014), which is related to $Fm|prmu, noidle|C_{max}$. The minor changes introduced relate to the different objective, which is different in this case.

The decision variables in `MILP_Form_Pos` are as follows:

- $x_{j,k} \in \{0,1\}$ is equal to 1 if job $j \in J$ is scheduled at position $k \in P$, 0 otherwise;
- $c_{q,k} \in \mathbb{R}_+$ indicates the completion time of the job at position $k \in P$ in machine $q \in M$;
- $T_k \in \mathbb{R}_+$ indicates the tardiness of the job at position $k \in P$.

The formulation `MILP_Form_Pos` of the $Fm|prmu, noidle|\sum_j T_j$ is presented below.

$$\min \sum_{k \in P} T_k \tag{1}$$

subject to:

$$\sum_{j\in J} x_{j,k} = 1 \qquad \forall k \in P \qquad (2)$$

$$\sum_{k\in P} x_{j,k} = 1 \qquad \forall j \in J \qquad (3)$$

$$c_{1,k} \geq \sum_{j\in J} p_{1,j} x_{j,1} \qquad \forall k \in P \qquad (4)$$

$$c_{q,k} \geq c_{q-1,k} + \sum_{j\in J} p_{q,j} x_{j,k} \qquad \forall q \in M - \{1\}, \forall k \in P \qquad (5)$$

$$c_{q,k} = c_{q,k-1} + \sum_{j\in J} p_{q,j} x_{j,k} \qquad \forall q \in M, \forall k \in P - \{1\} \qquad (6)$$

$$T_k \geq c_{m,k} - \sum_{j\in J} d_j x_{j,k} \qquad \forall k \in P \qquad (7)$$

$$x_{j,k} \in \{0,1\} \qquad \forall j \in J, \forall k \in P \qquad (8)$$

$$c_{q,k} \in \mathbb{R}_+ \qquad \forall q \in M, \forall k \in P \qquad (9)$$

$$T_k \in \mathbb{R}_+ \qquad \forall k \in P \qquad (10)$$

The objective (1) is the minimization of the total tardiness. Constraint (2) ensures that each position is assigned to a unique job, while Constraint (3) ensures that each job is assigned to a unique position. Constraint (4) sets a lower bound on the completion time of each job in the first machine. Constraint (5) ensures that a job is completed on the qth machine, after being completed on the $(q-1)$th machine and processed by machine q. Constraint (6) ensures that the no-idle requirement is enforced. Constraint (7) computes the total tardiness of the job in the kth position, expressed as the difference between the job completion time in the last machine and the job due date. Finally, (8)–(10) specify the domains of each of the decision variables in the formulation.

The number of variables in the model is $\mathcal{O}(n^2 + nm)$, due to the variables $x_{j,k}$ and $c_{q,k}$ defined in (8) and (9). The number of constraints is $\mathcal{O}(nm)$.

2.2. A MILP formulation based on precedence variables

An alternative formulation of the problem is based on precedence variables. We denote this formulation as `MILP_Form_Ins`. To the best of our knowledge, this formulation is novel for $Fm|prmu, noidle|\sum_j T_j$ and it can be easily modified to model $Fm|prmu, noidle|C_{max}$ as well.

The decision variables considered in `MILP_Form_Ins` are:

- $x_{i,j} \in \{0,1\}$ is equal to 1 if job $i \in J$ is scheduled before another job $j \in J$, 0 otherwise;
- $y_{j,q} \in \mathbb{R}_+$ is the sum of the duration of job $j \in J$ and the ones preceding it, on machine $q \in M$;
- $\Delta_q \in \mathbb{R}_+$ with $q \in M$ is the difference between the starting time of the first job in the qth machine and the $(q-1)$th machine if $q \geq 2$, while Δ_1 is imposed to be equal to 0;
- $T_j \in \mathbb{R}_+$ indicates the tardiness of the job j.

The formulation of `MILP_Form_Ins` for $Fm|prmu, noidle|\sum_j T_j$ is as follows:

$$\min \sum_{j\in J} T_j \qquad (11)$$

subject to:

$$x_{i,j} + x_{j,i} = 1 \qquad \forall i \neq j \in J \qquad (12)$$

$$x_{i,j} + x_{j,k} + x_{k,i} \leq 2 \qquad \forall i \neq j \neq k \in J \qquad (13)$$

$$y_{j,q} = p_{q,j} + \sum_{i\in J} p_{q,i} x_{i,j} \qquad \forall j \in J, \forall q \in M \qquad (14)$$

$$T_j \geq y_{j,m} + \sum_{q\in M} \Delta_q - d_j \qquad \forall j \in J \qquad (15)$$

$$\Delta_1 = 0 \qquad (16)$$

$$\Delta_q \geq y_{j,q-1} - \sum_{i\in J} p_{q,i} x_{i,j} \qquad \forall j \in J, \forall q \in M - \{1\} \qquad (17)$$

$$x_{i,j} \in \{0,1\} \qquad \forall i \neq j \in J \qquad (18)$$

$$y_{j,q} \in \mathbb{R}_+ \qquad \forall j \in J, \forall q \in M \qquad (19)$$

$$\Delta_q \in \mathbb{R}_+ \qquad \forall q \in M \qquad (20)$$

$$T_j \in \mathbb{R}_+ \qquad \forall j \in J \qquad (21)$$

The objective (11) is again the minimization of the sum of the tardiness of the jobs. Constraint (12) ensures that $i \in J$ precedes $j \in J$ or vice-versa. Constraint (13) enforces that there are no precedence loops. Constraints (14) and (15) impose that the variables $y_{j,q}$ and T_j are computed as per their definitions. Constraints (16) and (17) ensure that Δ_q is adjusted such that each job starts by being processed by a certain machine after completing the work on the previous one. Finally, (18)–(21) state the domains of the decision variables.

Similar to `MILP_Form_Pos`, the number of variables in the model is $\mathcal{O}(n^2 + nm)$, due to the variables $x_{i,j}$ defined in (18) and the variables $y_{j,q}$ defined in (19). The number of constraints is $\mathcal{O}(n^3 + nm)$.

3. Fast computation of total tardiness

This section proposes a fast procedure to compute the total tardiness for the NPFP, based on Ding et al. (2015), where a similar procedure was proposed for another flowshop problem, i.e. $Fm|nowait|\sum_j T_j$. State-of-the-art metaheuristic approaches for $Fm|prmu, noidle|\sum_j T_j$ are based on the fast exploration of the one-opt insertion neighborhood, tuned for the total tardiness objective.

Such a speed up is presented in Tasgetiren et al. (2013), but there seems to be a mistake in the computational complexity of the local search procedure described at Section 2.3 of that paper. Specifically, the main loop requires n iterations of Step 2 and Step 3, and Step 2 can be performed in $\mathcal{O}(mn)$. However, Step 3 has a time complexity of $\mathcal{O}((m+n)n) = \mathcal{O}(n^2 + mn)$, instead of $\mathcal{O}(mn)$ as wrongly reported. Step 3 consists of Steps 3a, 3b, 3c, 3d and 3e, which are iterated over all the $O(n)$ insertion positions. Step 3a requires the computation of $F(\Delta\pi_h^E, k, k+1)$ for $k = 1, \ldots, m-1$, which can be done in $O(m)$, since each $F(\Delta\pi_h^E, k, k+1)$ can be computed in $O(1)$. Similarly, Step 3b can be performed in $O(m)$ operations, which are required for the computation of $F(c\pi, k, k+1)$ for $k = 1, \ldots, m-1$. Step 3c requires summing up $m-1$ values and its complexity is $O(m)$. In fact, the second sum over n is the sum of the processing times over the first machine and can be computed a-priori. Both Step 3d and Step 3e perform an elementary operation by scanning the jobs, and their computational cost is $O(n)$. In conclusion, the complexity of Step 3 is $O(n) \times (O(m) + O(m) + O(m) + O(n) + O(n)) = O(n(n+m))$. It seems like the complexity of Step 3e and Step 3d, which are necessary for the total tardiness computation and require $\mathcal{O}(n)$ additions, is ignored in Tasgetiren et al. (2013). Since Step 3 is repeated n times, the complexity of the procedure in Section 2.3 of Tasgetiren et al. (2013) is $\mathcal{O}((m+n)n^2)$.

However, the procedure defined in Ding et al. (2015) can be adapted to speed up the total tardiness computation, by requiring less than $\mathcal{O}(n)$ additions in practice. The main idea is to separate jobs into three categories: early, late and sensitive. These categories are defined after removing the selected job from the initial solution, when we seek for the best insertion position. Early jobs are completed before or at their due dates no matter where the removed job is inserted. On the contrary, late jobs are always completed after their due dates. Sensitive jobs may, or may not, be completed before their due dates, depending on the position chosen for the removed job. The rationale behind the fast tardiness computation is that the tardiness of the jobs in the early category can be excluded from the total tardiness computation, while the tardiness of the jobs in the late category can be computed in constant time. Hence, only the jobs in the sensitive category need to be scanned for the computation of the total tardiness.

Algorithm 1 describes the improved version of the procedure described in Section 2.3 of Tasgetiren et al. (2013), where the local search is performed by exploiting the fast total tardiness computation. The input arguments of this procedure are the initial solution π and a random or greedy jobs permutation π^{ref}. The output is the best solution found in the local search, π^{best}. The first loop at line 3 iterates over all the positions of the reference permutation π^{ref}. Inside such loop, we indicate with j the selected job in the reference permutation, as indicated in line 4. Moreover, we indicate with π' the partial permutation obtained by eliminating the job j from π (line 5). Lines 6–8 compute all the C_{max} values associated with the different insertion positions, by exploiting the forward backward pass procedure for C_{max} calculation. We indicate with C^k_{max} the makespan value when insertion is performed at the kth position. The function `ForwardBackwardPassInitialize(`π`, j)` computes the values of $E(\cdot,\cdot,\cdot)$ and $F(\cdot,\cdot,\cdot)$ as indicated in Step 2a and Step 2b at page 6767 of Tasgetiren et al. (2013). These data are indicated with D in our pseudocode. `ForwardBackwardPassInsertion(k, j,` π'`, D)` corresponds to Steps 3a–3d of Tasgetiren et al. (2013), with the only difference that all the values $\{C^k_{max}\}_{k\in P}$ are stored. The maximum and minimum values of the makespans are indicated with C^{LB}_{max} and C^{UB}_{max} (lines 9–10). Lines 11–15 derive the bounds C^{LB}_k and C^{UB}_k on the completion times of the job at the kth position in π'. They are computed by considering the bounds on the makespan C^{LB}_{max} and C^{UB}_{max} and the best/worst insertion positions for the job j. Lines 16–25 classify the jobs in categories. We indicate with $\mathcal{L}$, $\mathcal{E}$ and S the set of positions with late, early and sensitive jobs, respectively. First, bounds on the tardiness of the job at the kth position are derived at lines 18–19. Hence, we evaluate these bounds and put the job in the related category (lines 20–25). Finally, the loop at line 26 is devoted to determine the best total tardiness value and the corresponding insertion position. For each insertion position, the total tardiness is decomposed in the contributions of late jobs ($T_{\mathcal{L}}$), of sensitive jobs (T_S) and of the job to be inserted (T_{ins}). The value of $T_{\mathcal{L}}$ is computed w.r.t. the lowest total tardiness value of the late jobs, indicated with $\sum_{l\in\mathcal{L}} T^{\text{LB}}_l$. The increase w.r.t. that value, is proportional to the gap δ between the makespan C^k_{max} obtained when the job j is inserted at position k and the lowest makespan C^{LB}_{max}. Such delay needs to be added for all the late jobs. Hence, we need to add the processing time of j for all the jobs with positions in $\mathcal{L}_k$, which are the late jobs from position k onwards. The formula used for the computation of $T_{\mathcal{L}}$ is reported at line 28. The tardiness T_{ins} of the inserted job is computed at line 29. The computation of T_S is performed by following a similar rationale followed for $T_{\mathcal{L}}$, but here we need to iterate over all the jobs whose position belongs to S and sum up their tardiness. For each sensitive job at position s, we compute its completion time C_s by summing the lower bound of the completion time C^{LB}_s, the gap δ and the processing time of job j in case that the insertion position k is before or at position s. The tardiness of the job at position s is obtained by computing the maximum between $C_s - d_{\pi'_s}$ and 0. The previous steps allow to compute the total tardiness at line 34. The best insertion position is found after scanning all the positions and the solution with the lowest total tardiness π^{best} is updated. Finally, the best solution π^{best} found during the execution of the procedure is provided as an output when the reference permutation π^{ref} is fully scanned.

In the following, an iteration of the fast tardiness computation is described through an example with 4 jobs and 4 machines. Note that this is only for presentation purposes, since the proposed procedure is designed to achieve a significant speed up for instances with a large number of jobs. The instance considered is denoted as I_0. The set of machines and jobs of I_0 is $M = \{q_1, q_2, q_3, q_4\}$ and $J = \{j_1, j_2, j_3, j_4\}$, respectively. The processing times are reported in Table 1, while the due dates are $d_{j_2} = d_{j_4} = 13$, $d_{j_1} = 14$, $d_{j_3} = 15$. An iteration of Algorithm 1 is described in the case where $\pi = [j_1, j_2, j_3, j_4]$ and the first job of π^{ref} is j_4. This means that j_4 has to be re-inserted at position

Algorithm 1: Exploration of the one-opt neighborhood with fast computation of $\sum_j T_j$

```
1  Function Fast_one_opt is
       Input: Solution π , reference permutation π^ref
       Output: Solution π^best
2      π^best ← π ;
3      for i = 1 ··· n do
4          j ← π^ref_i ;
5          π' ← delete j from π ;
6          D ← ForwardBackwardPassInitialize(π, j) ;
7          for k = 1 ··· n do
8              C^k_max ← ForwardBackwardPassInsertion(k,
                 j, π', D) ;
9          C^LB_max ← min{C^k_max}_{k∈P} ;
10         C^UB_max ← max{C^k_max}_{k∈P} ;
11         C^LB_{n−1} ← C^LB_max − p_{m,j} ;
12         C^UB_{n−1} ← C^UB_max + p_{m,j};
13         for k = n − 1 ··· 1 do
14             C^LB_{k−1} ← C^LB_k − p_{m,π'_k} ;
15             C^UB_{k−1} ← C^UB_k − p_{m,π'_k} ;
16         L ← E ← S ← ∅ ;
17         for k = 1 ··· n − 1 do
18             T^LB_k ← C^LB_k − d_{π'_k} ;
19             T^UB_k ← C^UB_k − d_{π'_k} ;
20             if T^UB_k > 0 and T^LB_k ≥ 0 then
21                 L ← L ∪ {k} ;
22             else if T^UB_k ≤ 0 and T^LB_k ≤ 0 then
23                 E ← E ∪ {k} ;
24             else
25                 S ← S ∪ {k} ;
26         for k = 1 ··· n do
27             δ ← C^k_max − C^LB_max ;
28             T_L ← Σ_{l∈L} T^LB_l + |L| × δ + |L_k| × p_{m,j} ;
29             T_ins ← max{C^k_max − Σ^n_{v=k} p_{m,π'_v} − d_j, 0} ;
30             for s ∈ S do
31                 C_s ← C^LB_s + δ + { p_{m,j}  if s ≥ k
                                      { 0        otherwise ;
32                 T_s ← max{C_s − d_{π'_s}, 0} ;
33             T_S ← Σ_{s∈S} T_s
34             TT ← T_L + T_ins + T_S ;
35             if TT < TT_best then
36                 TT_best ← TT ;
37                 π^best ← insert job j to position k in π' ;
38     return π^best
```

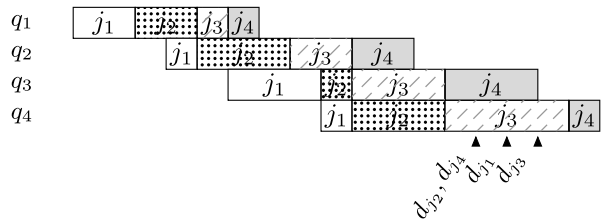
Table 1
Processing times of instance I_0.

	j_1	j_2	j_3	j_4
q_1	2	2	1	1
q_2	1	3	2	2
q_3	3	1	3	3
q_4	1	3	4	1

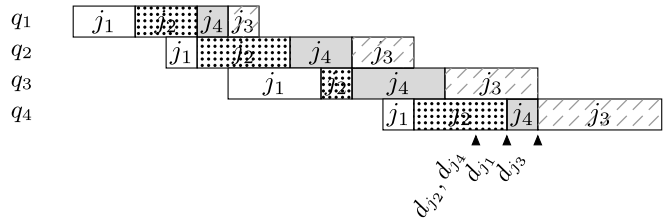
1,2 and 3 of $\pi' = [j_1, j_2, j_3]$. First, the makespan is computed for each possible insertion position by following the approach in Tasgetiren et al. (2013). The values obtained are $C^1_{max} = 18$, $C^2_{max} = 17$ and $C^3_{max} = 19$, hence C^{UB}_{max} and C^{LB}_{max} are set to 19 and 17, respectively (see Fig. 1).

The bounds on the completion time and tardiness of each job are derived from C^{UB}_{max} and C^{LB}_{max}. The first job considered is j_3, which

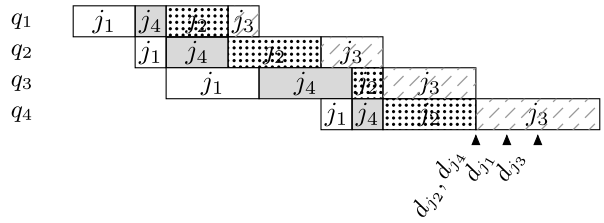
Initial solution $\pi = [j_1, j_2, j_3, j_4]$, $C_{max} = 17$, $\sum_{j\in J} T_j = 5$



Insertion at the 3-rd position $\pi_3 = [j_1, j_2, j_4, j_3]$, $C_{max} = 19$, $\sum_{j\in J} T_j = 7$



Insertion at the 2-nd position $\pi_2 = [j_1, j_4, j_2, j_3]$, $C_{max} = 17$, $\sum_{j\in J} T_j = 2$



Insertion at the 1-st position $\pi_1 = [j_4, j_1, j_2, j_3]$, $C_{max} = 18$, $\sum_{j\in J} T_j = 4$

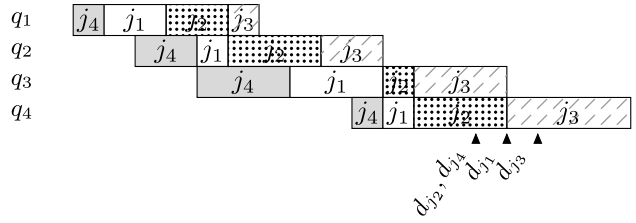


Fig. 1. Initial solution $\pi = [j_1, j_2, j_3, j_4]$ and solutions π_1, π_2, π_3, obtained by re-inserting job j_4 at position 1,2,3.

occupies the third position of π':

$$C_3^{LB} = C_{max}^{LB} - p_{m,j_4} = 17 - 1 = 16$$

$$C_3^{UB} = C_{max}^{UB} + p_{m,j_4} = 19 + 1 = 20$$

$$T_3^{LB} = C_3^{LB} - d_{j_3} = 16 - 15 = 1$$

$$T_3^{UB} = C_3^{UB} - d_{j_3} = 20 - 15 = 5$$

Since both bounds of the tardiness are greater than or equal to 0, j_3 is a late job and $\mathcal{L} = \{3\}$. The following computations are related to the job j_2, which is at the second position of π':

$$C_2^{LB} = C_3^{LB} - p_{m,j_3} = 16 - 4 = 12$$

$$C_2^{UB} = C_3^{UB} - p_{m,j_3} = 20 - 4 = 16$$

$$T_2^{LB} = C_2^{LB} - d_{j_2} = 12 - 13 = -1$$

$$T_2^{UB} = C_2^{UB} - d_{j_2} = 16 - 13 = 3$$

In this case, T_2^{LB} is negative, while T_2^{UB} is positive. This means that j_2 is a sensitive job, then $S = \{2\}$. Finally, we consider job j_1 at the first position of π':

$$C_1^{LB} = C_2^{LB} - p_{m,j_2} = 12 - 3 = 9$$

$$C_1^{UB} = C_2^{LB} - p_{m,j_2} = 16 - 3 = 13$$

$$T_1^{LB} = C_1^{LB} - d_{j_1} = 9 - 14 = -5$$

$$T_1^{UB} = C_1^{UB} - d_{j_1} = 13 - 14 = -1$$

T_1^{LB} and T_1^{UB} are both negative, hence j_1 is an early job and $\mathcal{E} = \{1\}$. At this point, the total tardiness is calculated for each possible insertion position. If we insert j_4 at the first position, we obtain:

$$T_{\mathcal{L}} = T_3^{\text{LB}} + |\mathcal{L}| \times (C_{max}^1 - C_{max}^{\text{LB}}) + |\mathcal{L}_1| \times p_{m,j_4} = 3$$

$$T_{ins} = \max\{C_{max}^1 - \sum_{v=1}^{n} p_{m,\pi'_v} - d_{j_4}, 0\} = 0$$

$$T_S = T_2 = \max\{C_2^{LB} + (C_{max}^1 - C_{max}^{\text{LB}}) + p_{m,j_4} - d_{\pi'_2}, 0\} = 1$$

Hence, the total tardiness TT is equal to 4 in this case. In the case where j_4 is inserted at the second position:

$$T_{\mathcal{L}} = T_3^{\text{LB}} + |\mathcal{L}| \times (C^2_{max} - C^{\text{LB}}_{max}) + |\mathcal{L}_2| \times p_{m,j_4} = 2$$

$$T_{ins} = \max\{C^2_{max} - \sum_{v=1}^{n} p_{m,\pi'_v} - d_{j_4}, 0\} = 0$$

$$T_S = T_2 = \max\{C_2^{LB} + (C^2_{max} - C^{\text{LB}}_{max}) + p_{m,j_4} - d_{\pi'_2}, 0\} = 0$$

This implies that the total tardiness is equal to 2 in this case. Finally, when we insert j_4 at the last position:

$$T_{\mathcal{L}} = T_3^{\text{LB}} + |\mathcal{L}| \times (C^3_{max} - C^{\text{LB}}_{max}) + |\mathcal{L}_3| \times p_{m,j_4} = 4$$

$$T_{ins} = \max\{C^3_{max} - \sum_{v=1}^{n} p_{m,\pi'_v} - d_{j_4}, 0\} = 2$$

$$T_S = T_2 = \max\{C_2^{LB} + (C^3_{max} - C^{\text{LB}}_{max}) + p_{m,j_4} - d_{\pi'_2}, 0\} = 1$$

This implies that the total tardiness is equal to 7. In conclusion, the best insertion position is the second position, since it leads to the best total tardiness value.

The complexity of the proposed procedure is equal to $\mathcal{O}(mn^2+n^2|S|)$, hence it depends on the number of sensitive jobs $|S|$. The term mn^2 is due to the procedure `ForwardBackwardPassInsertion(k, j, π', D)` since it has a complexity of $\mathcal{O}(m)$ and it runs for n^2 times. The term $n^2|S|$ is due to the loop at line 30 over the sensitive jobs, which is run for all the insertion positions and for all the positions in the reference permutation. The value of $|S|$ is equal to n in the worst case, leading again to the same complexity of the procedure in Tasgetiren et al. (2013). However, Ding et al. (2015) show that the number of sensitive jobs is very low with respect to the overall number of jobs. We performed some preliminary test running the reimplementation of the state-of-the-art algorithm with the improved one-opt insertion over the 120 instances of Taillards' dataset with tight($\tau = 1$), medium($\tau = 2$) and loose($\tau = 3$) due dates. Summing up all the observations of jobs belonging to a category, for $\tau = 1$ only 5 instances out of 120 have some sensitive jobs, the rest are late jobs. For $\tau = 2$, a looser set of due dates, around 0.08% of the jobs inspected are early, 1.1% are sensitive and 98.82% late. As for $\tau = 3$ around 5.37% of the jobs are early, 9.73% are sensitive and 84.90% are late. Hence we obtain significant speed up since the number of the sensitive jobs is very small.

4. MILP-Based local search procedures

We first describe two neighborhoods and the corresponding properties in Section 4.1. We then propose six MILP-based local search procedures in Section 4.2. They are based on exploring the neighborhoods by means of a MILP solver, which uses the formulations `MILP_Form_Pos` and `MILP_Form_Ins`, presented in Section 2. Finally, Section 4.3 describes the general framework in which the local search procedures are embedded.

4.1. Definition of the neighborhoods

We define two neighborhoods of a given initial permutation π over the jobset J. Please note that we indicate with π_k the job at the kth position of π.

Definition 1. Let $\hat{P} \subseteq P$ be a set of positions. $\mathcal{N}^1_{\pi,\hat{P}}$ is the set of all the feasible solutions such that the kth position of the permutation is occupied by π_k for each $k \in P - \hat{P}$.

Definition 2. Let $\hat{J} \subseteq J$ be a set of jobs. $\mathcal{N}^2_{\pi,\hat{J}}$ is the set of all the feasible solutions such that the precedence relationships in π, between couples of jobs (i, j) such that $i, j \in J - \hat{J}$, are verified.

$\mathcal{N}^1_{\pi,\hat{P}}$ refers to all the solutions obtained by fixing all the positions that are not in $\hat{P}$ to their corresponding job in π. Finding the best solution in $\mathcal{N}^1_{\pi,\hat{P}}$ involves rearranging the jobs at the positions in $\hat{P}$ to optimality. On the contrary, $\mathcal{N}^2_{\pi,\hat{J}}$ refers to all the solutions obtained by fixing all the jobs' precedence relationships in π that are not involving jobs in $\hat{J}$. Hence, finding the best solution in $\mathcal{N}^2_{\pi,\hat{J}}$ is equivalent to reinserting the jobs of $\hat{J}$ in the best possible positions.

Given an initial solution π, the definitions of $\mathcal{N}^1_{\pi,\hat{P}}$ and $\mathcal{N}^2_{\pi,\hat{J}}$ depend on the set of positions $\hat{P}$ and the set of jobs $\hat{J}$, respectively. The following proposition characterizes the size of both neighborhoods when the cardinality of $\hat{P}$ and $\hat{J}$ are the same ($w = |\hat{P}| = |\hat{J}|$). Recall that $n = |J| = |P|$.

Proposition 1. *$|\mathcal{N}^2_{\pi,\hat{J}}| = (n-w+1)\cdots n$ and $|\mathcal{N}^1_{\pi,\hat{P}}| = w!$ if $w = |\hat{P}| = |\hat{J}|$.*

Proof. Each solution included in $\mathcal{N}^1_{\pi,\hat{P}}$ is a rearrangement of the jobs in the positions $\hat{P}$, hence each element of the set is identified by a permutation of these w jobs and $|\mathcal{N}^1_{\pi,\hat{P}}| = w!$. Each solution in $\mathcal{N}^2_{\pi,\hat{J}}$ corresponds to inserting the jobs in $\hat{J}$ in the partial sequence π', obtained by removing from π the jobs in $\hat{J}$. We need to enumerate all the possible permutations that can be created by inserting all the jobs one by one, given a certain order of the jobs. In fact, when the first job is inserted, there are $n - w + 1$ possible positions where it can be inserted, including the first and the last positions. When the second job is inserted, the possible choices are $n - w + 2$, and this number increases by one unit each time we insert one more job. Hence, $|\mathcal{N}^2_{\pi,\hat{J}}| = (n - w + 1)\cdots n$. □

Therefore, we can state the following relationship between the structure of the two neighborhoods:

Lemma 1. *$|\mathcal{N}^2_{\pi,\hat{J}}| > |\mathcal{N}^1_{\pi,\hat{P}}|$ if $w = |\hat{P}| = |\hat{J}|$ and $w < n$.*

Proof. In the extreme case where $n = w$, both neighborhoods $\mathcal{N}^1_{\pi,\hat{P}}$ and $\mathcal{N}^2_{\pi,\hat{J}}$ refer to the whole solution space, formed by $n!$ permutations. This case is excluded by the condition $w < n$. In the case where $w < n$, we have that $(n - w + 1)\cdots n > w!$. In fact, this inequality is equivalent to $\frac{(n-w+1)\cdots n}{1\cdots w} > 1$. The fraction in the left is always greater than 1 since each term at the numerator is greater than the corresponding term in the denominator, as a consequence of $w < n$. □

In the case where the set $\hat{J}$ is formed by the jobs of π assigned to the positions in the set $\hat{P}$, we can derive a stronger relationship between the two neighborhoods.

Proposition 2. *$\mathcal{N}^1_{\pi,\hat{P}} \subset \mathcal{N}^2_{\pi,\hat{J}}$ if $\pi_k \in \hat{J}$ $\forall k \in \hat{P}$ and $P - \hat{P} \neq \emptyset$.*

Proof. We show that each solution $s \in \mathcal{N}^1_{\pi,\hat{P}}$ is also included in $\mathcal{N}^2_{\pi,\hat{J}}$. The only difference between the permutation s and the initial solution π is how the jobs at the positions in $\hat{P}$ are assigned. In order to prove that $s \in \mathcal{N}^2_{\pi,\hat{J}}$, we show that it can be obtained by considering the partial sequence π', obtained by removing from π the jobs belonging to $\hat{J}$, and inserting the jobs in $\hat{J}$ one by one. In fact, any insertion operation does not change the jobs precedence in π', which implies that the solution constructed belongs to $\mathcal{N}^2_{\pi,\hat{J}}$ (see Definition 2). First, we consider the job s_k at the smallest position $k \in \hat{P}$, and we insert it in the same position of π'. Afterward, we proceed with inserting the job of s at the next smallest position of $\hat{P}$ in π'. In this way, we are able to recreate s, which means that it belongs to $\mathcal{N}^2_{\pi,\hat{J}}$. Lemma 1 shows that $\mathcal{N}^2_{\pi,\hat{J}}$ has a greater size than $\mathcal{N}^1_{\pi,\hat{P}}$, which concludes the proof. □

Example 1. Let us consider an instance with $J = \{j_1, j_2, j_3, j_4\}$ and an initial solution $\pi = [j_1, j_2, j_3, j_4]$. Let the set $\hat{P}$ be equal to $\{1, 2\}$ and let $\hat{J}$ be equal to $\{j_1, j_2\}$, hence $\pi_k \in \hat{J}$ $\forall k \in \hat{P}$ and $P - \hat{P} = \{3, 4\}$.

Table 2
Sets $\mathcal{N}^1_{\pi,\hat{P}}$ and $\mathcal{N}^2_{\pi,\hat{J}}$ of Example 1.

$\mathcal{N}^1_{\pi,\hat{P}}$		$\mathcal{N}^2_{\pi,\hat{J}}$	
$[j_1, j_2, j_3, j_4]$	$[j_1, j_2, j_3, j_4]$	$[j_2, j_3, j_1, j_4]$	$[j_3, j_2, j_1, j_4]$
$[j_2, j_1, j_3, j_4]$	$[j_2, j_1, j_3, j_4]$	$[j_3, j_2, j_4, j_1]$	$[j_2, j_3, j_4, j_1]$
	$[j_3, j_1, j_2, j_4]$	$[j_3, j_4, j_2, j_1]$	$[j_3, j_4, j_1, j_2]$
	$[j_1, j_3, j_2, j_4]$	$[j_1, j_3, j_4, j_2]$	$[j_3, j_1, j_4, j_2]$

The sets $\mathcal{N}^1_{\pi,\hat{P}}$ and $\mathcal{N}^2_{\pi,\hat{J}}$ are enumerated in Table 2. $\mathcal{N}^1_{\pi,\hat{P}}$ contains two permutations obtained by swapping j_1 and j_2, while $\mathcal{N}^2_{\pi,\hat{J}}$ contains all the permutations of the jobs j_1, j_2, j_3, j_4 such that j_3 precedes j_4. As stated by the previous propositions, we can observe that $|\mathcal{N}^2_{\pi,\hat{J}}| = 12$ and $|\mathcal{N}^1_{\pi,\hat{P}}| = 2$, hence $|\mathcal{N}^1_{\pi,\hat{P}}| < |\mathcal{N}^2_{\pi,\hat{J}}|$. Finally, the two solutions of $\mathcal{N}^1_{\pi,\hat{P}}$ are included in $\mathcal{N}^2_{\pi,\hat{J}}$, thus $\mathcal{N}^1_{\pi,\hat{P}} \subset \mathcal{N}^2_{\pi,\hat{J}}$.

4.2. Structure of the local search procedures

The exploration of both neighborhoods can be done by enumeration, but this is very time consuming even for small values of w. However, $\mathcal{N}^1_{\pi,\hat{P}}$ can be explored efficiently by solving a model `MILP_Form_Pos` where the variables $x_{\pi_k,k}$ are fixed to 1 for each $k \in P - \hat{P}$. Analogously, $\mathcal{N}^2_{\pi,\hat{J}}$ can be explored efficiently by solving a model `MILP_Form_Ins` where the variables $x_{i,j}$ are fixed to 1 if $i, j \notin \hat{J}$ and i precedes j in π. The effectiveness of such neighborhood explorations depends on the quality of the linear relaxations yielded by such models, which determine different performances in solving small instances of the NPFSP, as will be shown in Section 5.1.

Different strategies to select sets of positions/jobs can be used to define $\mathcal{N}^1_{\pi,\hat{P}}$ and $\mathcal{N}^2_{\pi,\hat{J}}$. Let us consider the following two strategies:

1. Select $w \in \{1, \dots, n-1\}$ positions randomly (or the corresponding jobs);
2. Select all the positions from a certain value ψ to a subsequent value $\psi + w$ (or the corresponding jobs).

We define six local search procedures based on the exploration of such neighborhoods. The first three procedures are described in Algorithm 2 and are based on the exploration of $\mathcal{N}^1_{\pi,\hat{P}}$ by means of `MILP_Form_Pos`. The first one is denoted as Random Positional Local Search (`RPLS`). It keeps selecting random positions to define $\hat{P}$, finds the best solution in $\mathcal{N}^1_{\pi,\hat{P}}$ and updates π. The second one and the third one are indicated with Sliding Window Positional Local Search (`SWPLS`) and Random Window Positional Local Search (`RWPLS`), respectively. Both procedures share the same steps to be done at each iteration. First, the positions from a ψ to $\psi + w_p$ are selected to construct the set $\hat{P}$ in both procedures, where $\psi \in [1, |P| - w_p]$ is an integer value and $w_p \in [1, |P| - 1]$ is a given integer parameter. Second, the best solution in $\mathcal{N}^1_{\pi,\hat{P}}$ is computed by using the formulation `MILP_Form_Pos` and π is updated. The difference between `SWPLS` and `RWPLS` is how ψ changes in the various iterations. In the first case, ψ starts from 1 and is increased by γ at each iteration. The value of ψ is set again to 1 whenever it exceeds $|P| - w_p$. In the second case, ψ takes a values from β_1 to $\beta_{|P|-w_p}$, where β is a random permutation of $1, \dots, |P| - w_p$. These first three procedures are iterated until a certain time limit θ is reached and the best solution found is provided as an output.

The other three local search procedures are presented in Algorithm 3. They are indicated with Random Insertion Local Search (`RILS`), Sliding Window Insertion Local Search (`SWILS`) and Random Window Insertion Local Search (`RWILS`), respectively. They are very similar to the first three procedures (`RPLS`, `SWPLS` and `RWPLS`). Each of them, after the computation of $\hat{P}$, does not explore the neighborhood $\mathcal{N}^1_{\pi,\hat{P}}$. Instead, it computes a set of jobs $\hat{J}$ as the set of the jobs π_p such that $p \in \hat{P}$, and finds the best solution in $\mathcal{N}^2_{\pi,\hat{J}}$ by means of `MILP_Form_Ins`. The other operations in each of these three procedures are the same of `RPLS`, `SWPLS` and `RWPLS`, respectively. Note that we use another parameter w_i instead of w_p, in order to allow the possibility to regulate the size of both neighborhoods differently.

Algorithm 2: Local search procedures relying on `MILP_Form_Pos`

```
Random Positional Local Search (RPLS) ;
Function RPLS is
    Input: Solution π, time limit θ
    Output: Solution π^best
    while θ is not reached do
        P̂ ← w_p random positions ;
        π ← best sol. in N^1_{π,P̂} computed by means of
          MILP_Form_Pos ;
    π^best ← π ;
    return π^best
Sliding Window Positional Local Search (SWPLS) ;
Function SWPLS is
    Input: Solution π, time limit θ
    Output: Solution π^best
    ψ ← 0 ;
    while θ is not reached do
        P̂ ← all the positions from ψ to ψ + w_p ;
        π ← best sol. in N^1_{π,P̂} computed by means of
          MILP_Form_Pos ;
        ψ ← ψ + γ ;
        if (ψ > |P| − w_p) then ψ ← 1 ;
    π^best ← π ;
    return π^best
Random Window Positional Local Search (RWPLS) ;
Function RWPLS is
    Input: Solution π, time limit θ
    Output: Solution π^best
    k ← 1 ;
    β ← random permutation of 1, ··· , |P| ;
    while θ is not reached do
        ψ ← β_k ;
        P̂ ← all the positions from ψ to ψ + w_p ;
        π ← best sol. in N^1_{π,P̂} computed by means of
          MILP_Form_Pos ;
        k ← k + 1 ;
        if (k = |P| + 1) then k ← 1 ;
    π^best ← π ;
    return π^best
```

The approaches proposed in this section require finding good values for the parameters w_p, w_i and γ. The parameters w_p and w_i determine the size of the neighborhoods. These parameters need to be set to a value for which the corresponding neighborhood can be explored in a small amount of time. The parameter γ is used only in `SWPLS` and `SWILS` and provides the increment of the start of the positions to be considered unfixed. This parameter needs to be set to a value that enables the full scan of the positions' range within the given time limit.

4.3. Matheuristic framework

The MILP-based local search procedures introduced in the previous section can be embedded in a matheuristic framework. In the last decades, combinatorial optimization problems have been tackled by two main types of approaches. On the one hand, mathematical solvers have been relying on a formulation of the problem and provide an out-of-the-shelf approach taking advantage of many years of algorithmic developments. Despite the fact that mathematical solvers allow to

Algorithm 3: Local search procedures relying on `MILP_Form_Ins`

```
Random Insertion Local Search (RILS) ;
Function RILS is
    Input: Solution π, time limit θ
    Output: Solution π^best
    while θ is not reached do
        P̂ ← w_i random positions ;
        Ĵ ← {π_p : p ∈ P̂} ;
        π ← best sol. in N^2_{π,Ĵ} computed by means of
          MILP_Form_Ins ;
    π^best ← π ;
    return π^best
Sliding Window Insertion Local Search (SWILS);
Function SWILS is
    Input: Solution π, time limit θ
    Output: Solution π^best
    ψ ← 1 ;
    while θ is not reached do
        P̂ ← all the positions from ψ to ψ + w_i ;
        Ĵ ← {π_p : p ∈ P̂} ;
        π ← best sol. in N^2_{π,Ĵ} computed by means of
          MILP_Form_Ins ;
        ψ ← ψ + γ ;
        if (ψ > |P| − w_i) then ψ ← 1 ;
    π^best ← π ;
    return π^best
Random Window Insertion Local Search (RWILS);
Function RWILS is
    Input: Solution π, time limit θ
    Output: Solution π^best
    k ← 1 ;
    β ← random permutation of 1, ··· , |P| ;
    while θ is not reached do
        ψ ← β_k ;
        P̂ ← all the positions from ψ to ψ + w_i ;
        Ĵ ← {π_p : p ∈ P̂} ;
        π ← best sol. in N^2_{π,Ĵ} computed by means of
          MILP_Form_Ins ;
        k ← k + 1 ;
        if (k = |P| + 1) then k ← 1 ;
    π^best ← π ;
    return π^best
```

easily obtain a solution method for a combinatorial problem, they usually do not scale well. On the other hand, metaheuristic approaches provide a much more scalable approach, but usually require a high development time due to the design and implementation of specific problem-dependent optimization routines. Matheuristics combine the advantages of both approaches by using mathematical solvers to design efficient and scalable heuristics. As an example, the local search procedures described in Section 4.2 can take advantage of the use of well-performing MILP solvers, while controlling the size of the subproblem by tuning the parameters w_p and w_i.

The performances of the MILP-based local search procedures depend on the quality of the initial solution. Hence, we take advantage of a matheuristic framework, depicted in Fig. 2, whose general structure has been used by a few previous works (such as Della Croce and Salassa (2014)). First, a metaheuristic algorithm is used to compute a good initial solution. Second, one of the MILP-based local search procedures described in Section 4.2 is applied to improve the initial solution. Please note that only one of the MILP-based local search procedures is applied. In fact, our preliminary experiments showed that

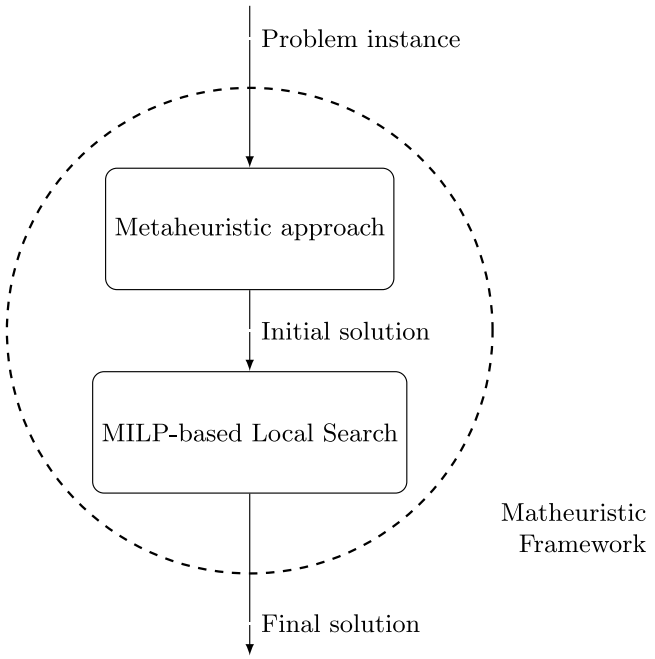


Fig. 2. Matheuristic framework based on using a metaheuristic and a MILP-based local search in a sequence.

the use of multiple MILP-based local search heuristics in a sequence, did not improve significantly the performances of the framework. This framework is used in Section 5 to evaluate the performances of the proposed procedures.

5. Computational experiments

We evaluate the performance of the MILP models and the proposed local search heuristics, presented in Sections 2 and 4.2, respectively. The experiments are conducted on Taillard's dataset (Taillard, 1993), comprising 120 instances with $n \in \{20, 50, 100, 200, 500\}$ and $m \in \{5, 10, 20\}$. The due date of each job $j \in J$ is chosen as a function of a tightness factor τ, by using the formula $d_j = \tau \times \sum_{q \in M} p_{q,j}$. Each instance in the dataset is taken into account three times, corresponding to the cases where τ is equal to 1, 2 and 3. The idea behind this choice is to evaluate the cases with tight, medium and loose due dates, similarly to Riahi et al. (2020).

In these experiments, the proposed approaches are compared with the best performing heuristic in the state-of-the-art, denoted by `EIGA` (Riahi et al., 2020). Such an approach was compared to four iterated greedy algorithms developed for other scheduling problems in the literature, and adapted to address $Fm|prmu, noidle|\sum_j T_j$. These algorithms are denoted as IGA_{RS1}, IGA_{RS2}, IGA_{PTL} and IGA_{PR} in Riahi et al. (2020). IGA_{RS1} (Ruiz and Stützle, 2007) is a basic iterated greedy approach with a loop including a destruction–reconstruction procedure and a local search. IGA_{RS2} (Ruiz and Stützle, 2008) was developed for flowshop problem with sequence-dependent setup times, both for the makespan and weighted tardiness objectives. It uses a different constructive algorithm to generate the initial solution. IGA_{PTL} (Pan et al., 2008) was developed for the flowshop problem with total flowtime objective. It relies on the constructive NEH procedure to generate the initial solution, with the addition of a greedy local search. IGA_{PR} (Pan and Ruiz, 2014) was introduced for the mixed no-idle flowshop scheduling problem, where the no-idle constraint is imposed on a subset of the machines. Moreover, `EIGA` was compared to the following state-of-the-art heuristics specifically designed for $Fm|prmu, noidle|\sum_j T_j$: the genetic algorithm proposed

in Tasgetiren et al. (2013), which consists of multiple crossover strategies, the discrete artificial bee colony algorithm in Tasgetiren et al. (2013), which uses six insertion and swap operations to diversify explored neighborhoods, and the effective bi-population estimation of distribution algorithm (Shen et al., 2014), which combines a global and a local probability model.

Recently, the authors in Riahi et al. (2020) reimplemented all the approaches mentioned above and tested them on the same machine. The results of their experiments reveal that EIGA outperforms all the other approaches. For this reason, we solely compare our results to EIGA. Thus, this heuristic was reimplemented with the fast tardiness computation discussed in Section 3. The time limit used for EIGA is $\theta = \rho \times \frac{nm}{1000}$ seconds. We indicate with RPH, SWPH, RWPH, RIH, SWIH and RWIH the heuristics that are obtained by using EIGA as a metaheuristic, and RPLS, SWPLS, RWPLS, RILS, SWILS, RWILS, respectively, as a MILP-based local search procedure in the general scheme described in Section 4.3. A time limit of θ seconds is assigned to each of the two phases of the hybrid heuristic framework.

All the approaches discussed are implemented in Java 8, while CPLEX 12.10 is the MILP solver used for testing both the MILP models and the MILP-based local search heuristics. Two different machines have been used for the experiments. An Intel Core i5-3550 3.30 GHz with 4 GB of RAM was used to test the MILP models (see Section 5.1 for the corresponding results), to compute the lower bounds considered reported in Section 5.3, and to evaluate the impact of using different time limits for the two phases of the hybrid heuristic scheme. An Intel Core i7-4712HQ 2.30 GHz×8 CPU with 16 GB of RAM was used to run all the heuristics (see Sections 5.2 and 5.3).

Section 5.1 is devoted to assessing the performances of the MILP models, in order to evaluate computationally the effectiveness of the two formulations proposed. Section 5.2 provides a comparison of the results obtained with the hybrid heuristics RPH, SWPH, RWPH, RIH, SWIH and RWIH, which are compared to each other and with the reimplementation of the approach in Riahi et al. (2020). Finally, Section 5.3 discusses the performances of the proposed approaches w.r.t. the results published in the state-of-the-art, by analyzing the amount of best known solutions updated and comparing them to the calculated lower bounds.

5.1. Computational assessment of the MILP models

This section assesses the performance of the MILP models described in Section 2 by testing them on Taillard's instances with $n = 20$ and $\tau = 1$. As discussed in the introduction, there are no ad-hoc exact algorithms to solve the NPFSP with total tardiness as an objective. A time limit of 1 h was set for each of the runs both for MILP_Form_Pos and MILP_Form_Ins. The results obtained are shown in Table 3, where we report the lower bounds computed by the solver, the percentage relative gap between the upper bound and the lower bound, and the computational time, for each model. The values reported in bold are referring to runs where optimality was proven by the MILP solver within the time limit.

We first consider the results related to the instances with $m = 5$. The model MILP_Form_Ins was not able to compute the optimal solutions in 2 cases out of 10 and required a much longer time in average with respect to MILP_Form_Pos, which was able to compute all the optima within 5 min. Analogously, MILP_Form_Pos was able to solve all the instances with $m = 10$ to optimality within the time limit, while MILP_Form_Ins was not able to compute the optimal solutions in 8 cases out of 10. Finally, both models struggled to solve the instances with 20 machines: MILP_Form_Pos proved optimality 4 times out of 10, while MILP_Form_Ins did not in any of the runs. We also note that MILP_Form_Pos computed better lower bounds with respect to MILP_Form_Ins in the great majority of our runs. The values reported in the last row of Table 3 represent the average of the gap and computational time.

Table 3
Results obtained by the MILP models over Taillard's instances with $\tau = 1$ and $n = 20$. The values in bold refer to the runs where optimality was reached.

m	Index	MILP_Form_Pos			MILP_Form_Ins		
		LB	Gap (%)	Time (s)	LB	Gap (%)	Time (s)
5	1	**11 967**	**0.00%**	**13**	**11 967**	**0.00%**	**205**
	2	**10 752**	**0.00%**	**99**	**10 752**	**0.00%**	**756**
	3	**12 041**	**0.00%**	**274**	11 662	3.41%	3600
	4	**11 163**	**0.00%**	**173**	**11 163**	**0.00%**	**3107**
	5	**12 812**	**0.00%**	**34**	**12 812**	**0.00%**	**604**
	6	**13 264**	**0.00%**	**66**	**13 264**	**0.00%**	**139**
	7	**9017**	**0.00%**	**41**	**9017**	**0.00%**	**1542**
	8	**10 564**	**0.00%**	**109**	10 316	2.35%	3600
	9	**11 887**	**0.00%**	**178**	**11 887**	**0.00%**	**725**
	10	**10 057**	**0.00%**	**97**	**10 057**	**0.00%**	**1961**
10	1	**22 317**	**0.00%**	**73**	**22 317**	**0.00%**	**3039**
	2	**17 657**	**0.00%**	**412**	16 271	10.71%	3600
	3	**18 568**	**0.00%**	**274**	**18 568**	**0.00%**	**2072**
	4	**19 259**	**0.00%**	**60**	18 872	2.11%	3600
	5	**14 414**	**0.00%**	**422**	13 456	7.97%	3600
	6	**19 029**	**0.00%**	**453**	18 488	2.93%	3600
	7	**16 091**	**0.00%**	**1294**	14 523	11.74%	3600
	8	**18 196**	**0.00%**	**400**	17 221	5.79%	3600
	9	**17 898**	**0.00%**	**420**	17 307	3.30%	3600
	10	**16 051**	**0.00%**	**1832**	14 567	9.25%	3600
20	1	33 889	5.01%	3600	29 875	19.48%	3600
	2	30 516	7.31%	3600	25 773	22.99%	3600
	3	32 946	6.65%	3600	30 305	15.45%	3600
	4	29 601	8.13%	3600	25 739	20.46%	3600
	5	34 736	2.52%	3600	30 928	17.61%	3600
	6	**37 514**	**0.00%**	**513**	34 998	6.71%	3600
	7	**33 469**	**0.00%**	**1494**	30 095	10.53%	3600
	8	**36 358**	**0.00%**	**1656**	31 488	15.74%	3600
	9	32 893	5.29%	3600	27 427	21.30%	3600
	10	**42 405**	**0.00%**	**1369**	39 201	9.11%	3600
	Average		1.16%	1111.86		7.30%	2871.66

In conclusion, MILP_Form_Pos strongly outperformed the model MILP_Form_Ins in our computational test. This is not just due to the smaller number of constraints, which is $\mathcal{O}(nm)$ for the first model and $\mathcal{O}(n^3 + nm)$ for the second one. In fact, the values of n and m are low in our experiments and it is likely that MILP_Form_Pos yields a much tighter relaxation. We leave to future works the development of new mathematical models and/or effective exact algorithms to solve $Fm|prmu, noidle|\sum_j T_j$.

5.2. Computational assessment of the local search procedures

Although MILP_Form_Pos outperformed MILP_Form_Ins, we compare their behavior when used in the local search procedures. This section compares the performances of the six hybrid heuristic approaches discussed before (RPH, SWPH, RWPH and RIH, SWIH, RWIH) with the reimplementation of EIGA.

Parameters used. The time limits used for each run were computed by considering the formula $2\theta = \rho \times \frac{nm}{2000}$, which provides the overall time limit in seconds, as a function of ρ, that is a given parameter, the number of jobs n and the number of machines m. Two sets of experiments were performed, one with a shorter time limit ($\rho = 60$) and another with a longer time limit ($\rho = 600$), whose results are shown in Tables 4 and 5, respectively. In the tables EIGAθ and EIGA2θ, respectively, present our reimplementation of the approach in Riahi et al. (2020) with the improved one-opt insertion, when it is run for a time limit of θ and 2θ. The parameters of the MILP-based approaches for the number of positions/job to unfix, w_p, w_i and the number of positions shifted, γ were chosen after some preliminary computational experiments.

The two neighborhoods described in Section 4.1 have different cardinalities, if the amount of free positions/jobs is fixed to the same

Table 4
Relative percentage deviation (RPD) gaps w.r.t the best objective for $\rho = 60$. The values in bold highlight the best performing approach per instance.

τ	n	m	EIGAθ	EIGA2θ	SWPH	RWPH	RPH	SWIH	RWIH	RIH
$\tau = 1$	20	5	0.57	0.36	0.32	**0.30**	0.49	0.56	0.52	0.56
		10	0.52	0.48	0.36	**0.33**	0.49	0.52	0.51	0.52
		20	0.89	0.86	0.64	**0.61**	0.73	0.89	0.87	0.89
	50	5	1.65	1.38	0.93	**0.38**	1.39	1.63	1.55	1.64
		10	1.92	1.55	1.21	**0.78**	1.73	1.92	1.92	1.92
		20	1.58	1.15	1.13	**0.99**	1.50	1.58	1.58	1.58
	100	5	1.71	1.49	1.11	**0.51**	1.58	1.70	1.71	1.70
		10	1.62	1.30	1.09	**0.54**	1.54	1.62	1.62	1.62
		20	1.53	0.91	1.01	**0.59**	1.47	1.53	1.53	1.53
	200	10	1.43	1.03	1.01	**0.52**	1.40	1.43	1.43	1.43
		20	1.57	0.91	1.17	**0.77**	1.55	1.57	1.57	1.57
	500	20	1.02	**0.49**	0.88	0.88	1.02	1.02	1.02	1.02
$\tau = 2$	20	5	0.74	0.42	0.40	**0.30**	0.63	0.74	0.68	0.71
		10	0.92	0.64	0.81	**0.58**	0.91	0.92	0.92	0.92
		20	1.19	1.17	1.02	**1.00**	1.08	1.19	1.19	1.19
	50	5	2.03	1.74	1.12	**0.53**	1.76	2.02	1.93	2.02
		10	2.59	2.16	1.81	**1.16**	2.46	2.59	2.58	2.59
		20	1.88	1.51	1.30	**0.94**	1.79	1.88	1.88	1.88
	100	5	1.98	1.59	1.32	**0.57**	1.80	1.98	1.98	1.98
		10	1.91	1.48	1.31	**0.68**	1.81	1.91	1.91	1.91
		20	1.98	1.36	1.35	**0.89**	1.91	1.98	1.98	1.98
	200	10	1.85	1.34	1.42	**0.89**	1.82	1.85	1.85	1.85
		20	1.63	0.93	1.17	**0.69**	1.61	1.63	1.63	1.63
	500	20	0.82	**0.45**	0.66	0.68	0.82	0.82	0.82	0.82
$\tau = 3$	20	5	0.96	**0.46**	0.86	0.89	0.91	0.96	0.91	0.96
		10	0.39	**0.15**	0.33	0.31	0.39	0.39	0.39	0.39
		20	0.41	**0.36**	0.41	0.41	0.41	0.41	0.41	0.41
	50	5	2.41	2.06	1.48	**0.57**	2.08	2.40	2.31	2.41
		10	4.02	3.36	2.89	**1.78**	3.68	4.01	3.98	4.02
		20	3.52	2.33	2.69	**1.89**	3.35	3.52	3.52	3.52
	100	5	2.20	1.80	1.47	**0.72**	2.04	2.20	2.20	2.20
		10	2.31	1.87	1.50	**0.86**	2.18	2.31	2.31	2.31
		20	2.42	1.65	1.65	**0.96**	2.33	2.42	2.42	2.42
	200	10	1.81	1.23	1.33	**0.76**	1.77	1.81	1.81	1.81
		20	2.04	1.22	1.52	**1.00**	2.01	2.04	2.04	2.04
	500	20	0.97	**0.44**	0.79	0.81	0.96	0.97	0.97	0.97
Average			1.63	1.21	1.15	**0.75**	1.54	1.64	1.62	1.64

value. We decided to set $w_i = \frac{w_p}{2}$, since such configuration yielded the best results in our preliminary tests. This takes into account the properties in Section 4.1 and the different performances of the two MILP models (Section 5.1). The value of w_p was then chosen in order to allow the neighborhood explorations to be completed in a short amount of time. We set $w_p = 10$ for all the instances, but the largest ones with $n = 500$, where $w_p = 20$. This is due to the fact that the creation of the model is much more expensive in the case where $n = 500$ and we prefer to focus on fewer, but larger neighborhood explorations. The value of γ used by SWPH and RWPH was chosen equal to 1 for all the instances, but the ones with $n = 500$ where γ is set to 10, where a much larger shift was beneficial. Finally, we set a time limit of 5 s for all the MILP-based neighborhood explorations, with the only exception of the instances with $n = 500$ where the time limit was set to 20 s.

We performed five experiments per instance, each one associated with a different seed. In order to compare the performance of the different approaches, we consider the relative percentage deviation (RPD). Given a certain instance i, an approach a and a seed s, we compute this measure as $RPD = \frac{obj_{i,a,s} - min_i}{min_i} \times 100$, where $obj_{i,a,s}$ is the objective computed by approach a running on instance i with seed s, and min_i is the best objective found among all the approaches and all the seeds for instance i. Each entry of Tables 4 and 5 reports the average RPD of each approach over instances with the same value of n and m, by considering all seeds.

In Table 4, the values in bold denote the minimum gaps per row, in order to highlight the best performing approach on average. The approaches RWPH and SWPH based on MILP_Form_Pos achieved the best results for almost all the instances, but the ones with $n = 500$, for which the time limit used was too short to perform the MILP-based search effectively. Moreover, RWPH achieved usually better results than SWPH. This is due to the fact that we increase by one unit the beginning of the window of unfixed positions in SWLS, which leads to re-including $w - 1$ positions in two subsequent neighborhood explorations. This strategy is poor in the case where the time limit is too short for reaching the last position. The procedure RPH yielded worse performances than RWPH and SWPH, because the strategy to select non-adjacent random positions seems to be not adequate.

The local search procedures based on MILP_Form_Ins performed poorly. This is mainly due to two reasons. On the one hand, the formulation MILP_Form_Ins is weaker than MILP_Form_Pos, as shown computationally in Section 5.1. This leads to performing the search inefficiently. On the other hand, the initial solutions used by all the procedures are local minima w.r.t. the one-opt neighborhood, which is contained in the neighborhoods used in such procedures. This is clearly evidenced by the results. In fact, RIH, RWIH and SWIH improved the initial solution only in 22% 42% and 19% of the cases. On the contrary, RWPH, SWPH and RPH improved the initial solution in 97%, 97%, 94% of the runs, respectively.

Table 5 shows results for the two best approaches from Table 4 (RWPH, SWPH) and the reimplementation of EIGA run for a larger time limit. In here we can see that EIGA2θ is outperformed in almost all

Table 5
Relative percentage deviation (RPD) w.r.t best objective per row, min_i for $\rho = 600$. In bold values denote the best approach for the specific instances.

τ	n	m	EIGAθ	EIGA2θ	SWPH	RWPH
$\tau = 1$	20	5	0.08	0.06	**0.05**	**0.05**
		10	0.44	0.44	**0.31**	0.32
		20	0.81	0.81	**0.42**	**0.42**
	50	5	1.67	1.55	**0.31**	0.36
		10	2.18	1.98	**0.63**	0.74
		20	1.14	1.01	0.58	**0.53**
	100	5	2.89	2.70	0.49	**0.45**
		10	2.03	1.91	0.42	**0.41**
		20	1.63	1.43	**0.43**	0.49
	200	10	2.61	2.43	0.66	**0.41**
		20	2.15	1.89	0.66	**0.45**
	500	20	1.22	0.81	**0.40**	0.45
$\tau = 2$	20	5	0.06	0.03	**0.01**	**0.01**
		10	0.57	0.57	**0.31**	**0.31**
		20	1.15	1.15	0.97	**0.84**
	50	5	2.10	1.93	**0.50**	0.54
		10	3.02	2.73	**1.11**	1.15
		20	1.93	1.75	**0.94**	0.97
	100	5	3.19	2.97	0.66	**0.49**
		10	2.48	2.32	0.57	**0.56**
		20	1.99	1.85	0.56	**0.55**
	200	10	2.84	2.64	0.63	**0.48**
		20	2.38	2.09	0.65	**0.52**
	500	20	1.30	0.95	**0.38**	0.44
$\tau = 3$	20	5	0.05	**0.00**	0.03	0.03
		10	**0.00**	**0.00**	**0.00**	**0.00**
		20	**0.18**	**0.18**	**0.18**	**0.18**
	50	5	2.46	2.28	0.56	**0.49**
		10	4.52	4.13	1.66	**1.56**
		20	3.30	2.90	**1.34**	1.52
	100	5	3.34	3.13	0.47	**0.42**
		10	2.84	2.70	0.63	**0.58**
		20	2.65	2.29	**0.77**	0.80
	200	10	3.05	2.79	0.68	**0.48**
		20	2.95	2.61	0.87	**0.64**
	500	20	1.43	0.99	**0.43**	0.51
Average			1.90	1.72	0.56	**0.53**

cases by either RWPH or SWPH. The larger time limit allows the MILP-based local search procedures to find better objectives than EIGA. In Table 4 RWPH clearly outperformed SWPH, whereas here results are mixed, with only slight deviations. Looking at the overall average performance RWPH performs slightly better then SWPH. Both approaches improved the initial solution given by EIGAθ in all the cases and 91% of the time they outperformed EIGA2θ. On average RWPH achieved 0.53% RPD w.r.t the best objective given by all approaches, whereas SWPH achieved 0.56%.

In all the experiments described above, the time available was split fairly between the two phases of the hybrid approaches. An interesting aspect is related to the possibility of evaluating the performances when splitting the computational budget 2θ in a different fashion. With this aim in mind, we run a set of experiments where the heuristic RWPH was considered, with a change in the time assigned to the two different phases. More specifically, a time $\alpha 2\theta$ is assigned to EIGA, while the remaining time $2(1-\alpha)\theta$ is assigned to RWPLS, with $\alpha \in [0, 1]$. We run experiments by considering all the instances with $n = 50$ and $n = 100$, by using 3 seeds, setting $\rho = 600$ and varying α from 0 to 1 with a step increase of 0.1. Fig. 3 depicts two plots of the average RPD obtained in the experiments versus α. When α is equal to 0 only RWPH runs, while only EIGA is considered when $\alpha = 1$. These two choices are discouraged by the results, because the corresponding points have a high value of average RPD. Moreover, there is a sharp improvement when we consider the values of α in the range $[0.2, 0.8]$. Indeed, there does not seem to be a relevant difference among the performances obtained in that wide interval. In conclusion, the choice of splitting the time fairly between the two phases of the approaches seems to be reasonable.

5.3. Best known solutions and optimality gaps

We compare our results with the ones presented in Table 14 of Riahi et al. (2020) by providing an updated list of best known solutions and the optimality gaps. Furthermore, Riahi et al. (2020) provides a table with the best known solutions found during their computational campaign, where a variety of state-of-the-art approaches were considered. The time limits used in Section 5.2, $\rho = 600$ is larger than the ones used in Riahi et al. (2020), $\rho = 20, 40$ and 60. However, there is a remarkable difference between the two machines used to perform the experiments. Moreover, the time limits used in this paper are much lower than the ones used in Tasgetiren et al. (2013).

Table 6 provides the objective value of the best known solutions. In addition to these, we computed a lower bound for each instance, by running MILP_Form_Pos for a time limit of 1 hour on all the instances but the ones with $n = 200$ and $m = 20$, for which we needed to run the solver for 2 hours to obtain meaningful bounds. We did not compute the bounds for the instances with $n = 500$, since it was computationally too expensive. Each instance is identified by its id, as in Table 14 of Riahi et al. (2020).

For each instance, we report the best known objective value (Bks), the source (Src) and the percentage optimality gap (Gap) between the best known objective value and the lower bound. Dash values for the gap denote instances where optimality was reached with respect to the calculated lower bound. The absolute values of the lower bounds are accessible in the supplementary material. For two instances, $\tau = 3$, id = 24 and id = 29 the lower bound obtained was 0, thus we denote this with a dash. For instances with $\tau = 3$ and id $\in \{21, 22, 23, 25, 26, 28, 30\}$ the lower bounds obtained were much lower then objectives from any of the approaches. In Table 14 of Riahi et al. (2020) there are four values reported that were incorrect: for $\tau = 1$, $Id = 17$ the value reported was 16 089 instead of 16 091; for $\tau = 2$, $Id = 15$ 5289 instead of 5301; for $\tau = 3$, $Id = 15$ 44 instead of 48 and for $\tau = 3$, $Id = 17$ 825 was reported instead of 836. These values are smaller than the lower bounds computed by the MILP model and also we did not find any other reference to these values. In fact, Tasgetiren et al. (2013) report the same values that we computed. The entries in the column reporting the source follow the following notation:

0: Reimplementation of EIGA with the improved one-opt method defined in Section 3;

1: The results reported in Riahi et al. (2020);

2: The SWPH procedure;

3: The RWPH procedure.

In Table 6 we notice that most of the instances with $n = 20$ (around 72%) were solved to optimality by the solver within the time limit, hence we can report the optimal objective function value. Two optimal objectives were calculated only by Riahi et al. (2020) and two calculated only by the reimplementation of EIGA with the one-opt speed up, SWPH and RWPH. For the remainder of the instances usually a single source achieves the best result, except in four cases where both SWPH and RWPH obtain the best values. The exact counts of how many times only SWPH, only RWPH and both of them obtain the best objectives can be found in Table 7. Here we notice that we improve solutions for all values of n and m, especially when $n = 100$ and 200.

Considering instances with $n = 50$ in Table 7 and corresponding instances with ids 30 – 60 in Table 6 we observe that for 30% of

Table 6
Best known values from all approaches.

Id	$\tau = 1$			$\tau = 2$			$\tau = 3$		
	Bks	Src	Gap	Bks	Src	Gap	Bks	Src	Gap
1	11 967	0,1,**2,3**	–	6941	0,1,**2,3**	–	2543	0,1,**2,3**	–
2	10 752	0,1,**2,3**	–	6042	0,1,**2,3**	–	2423	0,1,**2,3**	–
3	12 041	1,**2,3**	–	7463	0,1,**2,3**	–	3637	0,1,**2,3**	–
4	11 163	0,1,**2,3**	–	5784	0,1,**2,3**	–	1901	0,1,**2,3**	–
5	12 812	0,1,**2,3**	–	7844	0,1,**2,3**	–	3174	0,1,**2,3**	–
6	13 264	0,1,**2,3**	–	8209	0,1,**2,3**	–	3592	0,1,**2,3**	–
7	9017	0,1,**2,3**	–	4395	0,1,**2,3**	–	1036	0,1,**2,3**	–
8	10 564	0,1,**2,3**	–	5542	0,1,**2,3**	–	1657	0,1,**2,3**	–
9	11 887	0,1,**2,3**	–	6645	0,1,**2,3**	–	2618	0,1,**2,3**	–
10	10 057	0,1,**2,3**	–	5341	0,1,**2,3**	–	1670	0,1,**2,3**	–
11	22 317	0,1,**2,3**	–	11 988	0,1,**2,3**	–	2993	0,1,**2,3**	–
12	17 657	0,1,**2,3**	–	7321	1	–	1026	0,1,**2,3**	–
13	18 568	0,1,**2,3**	–	8863	0,1,**2,3**	–	1430	0,1,**2,3**	34.91
14	19 259	0,1,**2,3**	–	10 329	0,1,**2,3**	–	2754	0,1,**2,3**	–
15	14 414	0,1,**2,3**	–	5301	0,**2,3**	–	48	0,**2,3**	–
16	19 029	1	–	9889	0,1,**2,3**	–	2305	0,1,**2,3**	5.11
17	16 091	0,**2,3**	–	6879	0,1,**2,3**	–	836	0,**2,3**	–
18	18 196	0,1,**2,3**	–	8294	0,1,**2,3**	–	993	0,1,**2,3**	–
19	17 898	0,1,**2,3**	–	8053	0,1,**2,3**	–	800	0,1,**2,3**	–
20	16 051	0,1,**2,3**	–	5881	0,1,**2,3**	2.28	522	0,1,**2,3**	–
21	34 821	0,1,**2,3**	2.74	14 548	0,1,**2,3**	4.28	989	0,1,**2,3**	2435.90
22	32 635	0,1,**2,3**	6.94	13 863	0,1,**2,3**	18.13	630	0,1,**2,3**	201.44
23	34 341	1	4.22	14 114	1	9.89	336	1	510.91
24	31 682	0,1,**2,3**	7.02	12 020	0,**2,3**	18.18	233	0,1,**2,3**	inf
25	35 635	0,1,**2,3**	2.56	15 168	0,1,**2,3**	4.30	888	0,1,**2,3**	69.47
26	37 514	1,**2,3**	–	17 813	1,**2,3**	–	1978	0,1,**2,3**	70.37
27	33 469	1	–	13 378	0,1,**2,3**	27.37	119	0,1,**2,3**	–
28	36 358	0,1,**2,3**	–	16 601	0,1,**2,3**	–	920	0,1,**2,3**	144.68
29	34 532	1	4.97	14 331	0,1,**2,3**	11.67	642	0,1,**2,3**	inf
30	42 405	0,**2,3**	–	23 067	0,**2,3**	–	4659	0,1,**2,3**	18.67
31	74 541	**2**	1.06	62 494	1	1.40	50 387	**2,3**	1.67
32	63 760	**3**	1.83	50 731	**2**	2.40	37 421	1	1.94
33	60 721	**3**	2.92	48 628	**3**	3.53	36 982	**3**	4.69
34	63 731	**2**	3.01	50 814	**2**	3.68	38 429	**2**	5.00
35	75 703	**2**	2.15	62 738	**2**	2.24	50 191	**3**	3.20
36	63 127	**3**	1.41	50 138	**2**	1.81	37 445	**3**	2.36
37	68 228	**3**	1.79	55 870	**3**	2.00	43 987	**3**	3.28
38	64 953	**2**	1.79	52 524	**2**	2.17	40 358	**3**	3.16
39	62 487	**2**	1.77	50 882	**3**	2.20	39 133	**2**	2.46
40	61 063	**3**	4.34	48 120	**2**	5.25	36 017	**3**	6.11
41	81 900	1	5.75	57 108	**2,3**	9.38	32 774	**3**	16.07
42	79 129	1	9.80	55 386	1	15.39	30 994	1	21.41
43	79 910	**2**	2.66	55 769	**3**	3.56	32 680	1	8.29
44	86 073	**3**	3.17	60 735	**2**	4.54	35 801	**3**	8.18
45	72 261	1	14.50	47 384	**3**	29.27	25 118	1	35.92
46	101 016	**2**	2.41	75 915	**2**	3.08	51 049	**3**	5.03
47	87 530	1	9.75	61 896	1	14.36	37 392	1	27.57
48	100 118	1	1.13	75 706	1	1.64	51 141	**2**	4.20
49	87 320	1	4.31	63 424	1	6.83	39 623	**3**	12.60
50	87 030	1	12.50	61 396	1	18.09	35 829	1	29.98
51	148 793	1	4.85	97 060	1	7.80	45 628	**3**	19.63
52	176 401	**3**	7.30	126 955	**3**	10.04	77 923	1	11.77
53	131 053	1	13.94	82 986	1	21.50	35 849	1	78.35
54	162 768	1	4.94	113 747	1	8.03	65 049	1	15.37
55	132 781	1	9.00	84 159	**2**	16.38	38 149	**2,3**	49.61
56	172 416	**3**	4.38	123 341	**3**	6.14	74 605	**3**	11.23
57	165 312	1	4.45	115 439	**2,3**	5.70	65 989	**2**	10.48
58	144 215	1	7.67	95 333	**2**	17.24	46 638	**2**	30.07
59	145 411	**3**	6.96	94 775	1	10.25	45 378	1	26.70
60	159 676	**3**	4.21	109 197	**2,3**	5.97	58 575	1	12.97

(*continued on next page*)

the instances SWPH achieves the best objectives and for 32% of the instances RWPH does. By taking into account instances where both of these achieve the best objective, 66.78% of the instances get updated by either SWPH or RWPH. Furthermore, the average gap between best known solutions and corresponding lower bounds has been reduced from 9.94% to 9.63% in case of $n = 50$.

Similarly, looking at instances with $n = 100$ with ids 60 – 90 in Table 6, we observe that SWPH achieves the best objectives over 35.56% of the instances and RWPH outperforms all other approaches for 60% of the instances. In total the two approaches update 95.56% of the best known solutions and the gap is reduced from 5.79% to 4.57%.

Considering instances with $n = 200$ with ids 90 – 110 in Table 6, we observe that SWPH outperforms all other approaches 13.3% of the time whereas RWPH does so 85% of the time. In total the two approaches update 98.3% of the best known solutions and the gap is reduced from 11.36% to 9.23%.

At last, for instances with $n = 500$ with ids 110 – 120 in Table 6, SWPH outperforms all other approaches 56.67% of the time but in this

Table 6 (*continued*).

Id	$\tau = 1$			$\tau = 2$			$\tau = 3$		
	Bks	Src	Gap	Bks	Src	Gap	Bks	Src	Gap
61	267 568	3	3.06	242 639	**3**	3.76	216 195	**2**	3.91
62	226 506	**3**	3.56	200 768	**3**	3.74	175 924	**2**	4.06
63	223 895	**3**	2.82	199 215	**3**	3.19	174 984	**3**	3.64
64	228 977	**3**	2.82	205 388	**3**	3.39	181 400	**3**	3.87
65	253 082	**2**	1.60	227 607	**3**	1.60	202 526	**3**	1.82
66	224 899	**3**	3.99	200 147	**3**	4.35	176 698	**2**	5.54
67	275 192	**2**	1.28	250 982	**3**	1.64	224 922	**3**	1.22
68	215 408	**2**	3.80	191 576	**3**	4.51	166 888	**3**	4.61
69	232 823	**3**	3.59	207 032	**3**	4.05	182 035	**3**	4.82
70	234 786	**3**	3.31	208 858	**2**	3.55	184 144	**3**	4.65
71	332 645	**2**	4.63	278 605	**3**	4.31	226 799	**3**	5.76
72	286 264	**2**	2.98	237 911	**2**	3.66	189 503	**3**	4.86
73	353 928	**2**	0.78	303 661	**2**	1.02	253 643	**3**	1.36
74	307 918	**2**	5.37	253 189	**3**	5.76	202 971	**3**	8.41
75	314 698	1	4.02	267 006	**3**	5.83	216 478	**3**	7.31
76	321 480	**3**	2.77	274 811	**3**	4.12	226 965	**2**	4.54
77	367 726	**2**	1.32	317 752	**3**	1.22	269 513	**2**	1.76
78	356 692	**2**	1.21	306 780	**3**	1.46	256 401	**3**	1.48
79	364 683	**2**	2.44	312 439	**3**	2.57	260 076	1	2.73
80	349 742	**3**	2.80	299 824	**3**	3.77	248 616	**3**	4.24
81	531 624	**2**	3.20	431 624	**3**	3.52	333 113	**3**	4.69
82	464 597	**3**	5.87	363 177	**3**	7.34	264 811	**3**	10.65
83	497 296	**3**	3.63	397 892	**2**	4.47	298 183	**2**	5.95
84	562 720	**2**	3.15	465 244	**3**	4.04	366 476	**3**	5.20
85	577 634	**2**	5.10	477 736	**2**	6.08	379 757	**3**	8.16
86	450 889	**2**	5.07	352 424	**2**	6.89	253 122	**2**	10.54
87	485 126	**3**	5.30	384 345	**2**	6.59	283 455	**3**	8.81
88	458 026	1	7.35	356 825	**3**	9.85	255 342	1	14.40
89	515 888	**3**	2.29	416 425	**2**	3.02	316 150	**2**	4.14
90	465 192	**3**	9.87	361 663	**2**	12.07	262 570	**2**	18.41
91	1 165 345	**3**	6.83	1 063 466	1	7.38	962 038	**3**	8.09
92	1 321 868	**3**	4.96	1 224 035	**2**	5.51	1 123 236	**3**	5.88
93	1 285 145	**3**	3.17	1 188 449	**3**	3.84	1 087 884	**3**	4.27
94	1 049 740	**3**	8.58	949 618	**3**	9.57	852 810	**2**	11.17
95	1 165 360	**3**	5.58	1 066 539	**3**	6.24	970 042	**3**	7.31
96	1 223 276	**3**	4.28	1 126 315	**3**	4.70	1 023 445	**3**	4.61
97	1 038 159	**3**	8.12	939 295	**3**	9.45	837 171	**3**	10.59
98	1 258 674	**3**	4.87	1 156 433	**2**	5.22	1 052 982	**2**	5.52
99	1 029 757	**3**	7.99	928 935	**3**	8.70	830 552	**3**	9.90
100	1 235 415	**3**	5.48	1 133 591	**3**	5.84	1 032 048	**3**	6.31
101	1 698 502	**3**	19.92	1 499 299	**3**	5.86	1 300 936	**3**	6.59
102	1 608 342	**3**	19.09	1 406 950	**3**	4.17	1 206 945	**3**	4.92
103	1 573 104	**3**	22.39	1 375 888	**3**	7.04	1 168 077	**3**	7.70
104	1 496 860	**2**	26.00	1 300 233	**2**	9.45	1 094 058	**3**	10.76
105	1 693 136	**3**	20.95	1 491 360	**3**	6.54	1 291 342	**3**	7.45
106	1 651 972	**3**	24.19	1 451 214	**2**	9.10	1 256 967	**3**	11.24
107	1 741 548	**3**	21.57	1 533 745	**3**	7.07	1 334 230	**2**	8.38
108	1 698 926	**3**	17.79	1 492 582	**3**	3.49	1 292 142	**3**	3.99
109	1 523 665	**3**	23.59	1 333 876	**3**	8.20	1 129 245	**3**	9.32
110	1 544 178	**3**	19.20	1 343 156	**3**	3.68	1 142 323	**3**	4.24
111	8 051 419	**2**	inf	7 564 765	**3**	inf	7 068 870	**3**	inf
112	8 207 466	**2**	inf	7 694 540	1	inf	7 192 935	**2**	inf
113	9 056 723	**2**	inf	8 463 618	**3**	inf	8 000 797	**2**	inf
114	7 886 351	**2**	inf	7 390 342	**2**	inf	6 902 134	**2**	inf
115	8 893 900	**2**	inf	8 395 506	**3**	inf	7 893 921	**2**	inf
116	8 205 656	**3**	inf	7 691 993	**2**	inf	7 157 162	**2**	inf
117	8 862 049	1	inf	8 344 824	1	inf	7 832 861	**2**	inf
118	9 181 333	1	inf	8 634 395	1	inf	8 171 015	**2**	inf
119	7 795 109	1	inf	7 336 217	**2**	inf	6 816 062	1	inf
120	8 938 644	1	inf	8 419 274	**2**	inf	7 904 686	**2**	inf

case RWPH outperforms a smaller amount of instances, only 16% of it. This leads to 73.3% of an update of all instances with $n = 500$ by either SWPH or RWPH.

In summary, 63% of all instances have been updated by either SWPH or RWPH, which corresponds to 77% if we disregard instances where optimality has been achieved already by previous approaches. The average gap between best known solutions and corresponding lower bounds has been reduced from 8.74% to 7.63%, if we consider instances with $n = 50$, $n = 100$ and $n = 200$. The two proposed methods SWPH and RWPH perform particularly well for instances where $n = 100$ and 200 both in terms of number of best known solutions updated and gap w.r.t the lower bounds closed.

Next we look at these two methods separately and compare their results independently of each other to the state-of-the-art results published in Riahi et al. (2020). By comparing SWPH and the state-of-the-art, SWPH managed to update 212 out of 360 best known solutions, which corresponds to 58%. Similarly, comparing RWPH to only the state-of-the-art, RWPH updated 213 best known solutions, which corresponds to 59%. If we disregard instances for which optimality was proven, both SWPH and RWPH managed to update 72% of the best known solutions separately.

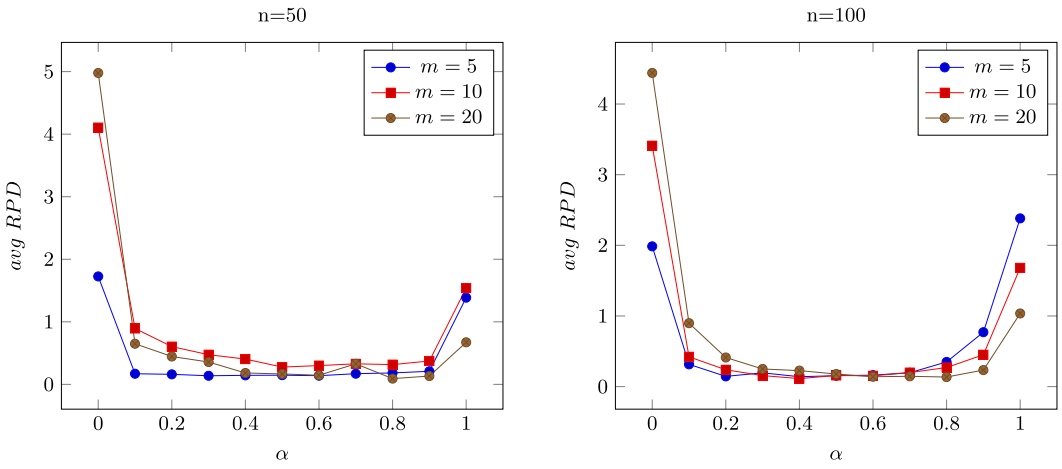


Fig. 3. Plot of the average RPD w.r.t. the parameter α (all the instances with n = 50 and n = 100, 3 seeds).

Table 7
Number of best known values updated.

τ	n	m	SWPH	RWPH	SWPH and RWPH
$\tau = 1$	50	5	5	5	–
		10	2	1	–
		20	–	4	–
	100	5	3	7	–
		10	7	2	–
		20	4	5	–
	200	10	–	10	–
		20	1	9	–
	500	20	5	1	–
$\tau = 2$	50	5	6	3	–
		10	3	2	1
		20	4	2	2
	100	5	1	9	–
		10	2	8	–
		20	6	4	–
	200	10	2	7	–
		20	2	8	–
	500	20	4	3	–
$\tau = 3$	50	5	3	6	1
		10	1	4	–
		20	3	2	1
	100	5	3	7	–
		10	2	7	–
		20	4	5	–
	200	10	2	8	–
		20	1	9	–
	500	20	8	1	–
Sum			84	139	5

6. Conclusions

We have presented six local search procedures for $Fm|prmu, noidle|\sum_j T_j$ that are based on two different neighborhoods and explore the search space by means of a MILP solver. We have shown that two hybrid approaches based on these procedures, denoted SWPH and RWPH, are able to significantly improve the results provided by the state-of-the-art metaheuristic, based on the fast exploration of the one-opt neighborhood. These results are achieved because of the effectiveness of the MILP-based search and on the different neighborhoods used, denoted as $\mathcal{N}^1_{\pi,\hat{P}}$ in Section 4.1. Such neighborhoods consider a different part of the search space with respect to the one-opt neighborhood. In our computational experiments we updated around 63% of all the best known solutions, and around 77% if we exclude the instances for which we proved that the previous best known solution is optimal.

We believe that future studies should be devoted to the design of hybrid heuristics for other permutation flowshop problems, with a similar structure to SWPH and RWPH, where high quality one-opt local minima are improved by means of a MILP-based search, performed on a different neighborhood.

Another interesting future research direction is the study of lower bounds and dominances for $Fm|prmu, noidle|\sum_j T_j$, which may lead to the definition of efficient exact approaches, improving the performances of the MILP models described in this work.

CRediT authorship contribution statement

Andrea Balogh: Visualization, Data curation, Software, Writing – original draft. **Michele Garraffa:** Conceptualization, Investigation, Methodology, Software, Writing – original draft. **Barry O'Sullivan:** Conceptualization, Methodology, Funding acquisition, Writing – review & editing. **Fabio Salassa:** Conceptualization, Investigation, Methodology, Writing – review & editing.

Acknowledgments

This work was supported by Science Foundation Ireland (SFI) under grant number 16/RC/3918 (Confirm) and 12/RC/2289-P2 (Insight), and co-funded under the European Regional Development Fund.

Appendix A. Supplementary data

The lower bounds obtained in all the experiments conducted are available for future studies, as well as in bold values that our approach obtained as well.

Supplementary material related to this article can be found online at https://doi.org/10.1016/j.cor.2022.105862.

References

Adiri, I., Pohoryles, D., 1982. Flowshop/no-idle or no-wait scheduling to minimize the sum of completion times. Nav. Res. Logist. Q. 29 (3), 495–504.

Baptiste, P., Hguny, L.K., 1997. A branch and bound algorithm for the $F|no-idle|C_{max}$. In Proceedings of the International Conference on Industrial Engineering and Production Management, IEPM, Vol. 97. pp. 429–438.

Baraz, Daniel, Mosheiov, Gur, 2008. A note on a greedy heuristic for flow-shop makespan minimization with no machine idle-time. European J. Oper. Res. 184 (2), 810–813.

Bektaş, Tolga, Hamzadayı, Alper, Ruiz, Rubén, 2020. Benders decomposition for the mixed no-idle permutation flowshop scheduling problem. J. Sched. 23 (4), 513–523.

fang Chen, Jing, Wang, Ling, ping Peng, Zhi, 2019. A collaborative optimization algorithm for energy-efficient multi-objective distributed no-idle flow-shop scheduling. Swarm Evol. Comput. 50, 100557.
Della Croce, Federico, Garraffa, Michele, Salassa, Fabio, Borean, Claudio, Di Bella, Giuseppe, Grasso, Ennio, 2017. Heuristic approaches for a domestic energy management system. Comput. Ind. Eng. 109, 169–178.
Della Croce, Federico, Grosso, Andrea, Salassa, Fabio, 2019. Minimizing total completion time in the two-machine no-idle no-wait flow shop problem. J. Heuristics.
Della Croce, Federico, Salassa, Fabio, 2014. A variable neighborhood search based matheuristic for nurse rostering problems. Ann. Oper. Res. 218 (1), 185–199.
Deng, Guanlong, Gu, Xingsheng, 2012. A hybrid discrete differential evolution algorithm for the no-idle permutation flow shop scheduling problem with makespan criterion. Comput. Oper. Res. 39 (9), 2152–2160.
Ding, Jianya, Song, Shiji, Zhang, Rui, Gupta, Jatinder N.D., Wu, Cheng, 2015. Accelerated methods for total tardiness minimisation in no-wait flowshops. Int. J. Prod. Res. 53 (4), 1002–1018.
Fatih Tasgetiren, M., Pan, Quan-Ke, Suganthan, P.N., Buyukdagli, Ozge, 2013. A variable iterated greedy algorithm with differential evolution for the no-idle permutation flowshop scheduling problem. Comput. Oper. Res. 40 (7), 1729–1743.
Goncharov, Yaroslav, Sevastyanov, Sergey, 2009. The flow shop problem with no-idle constraints: A review and approximation. European J. Oper. Res. 196 (2), 450–456.
Graham, R.L., Lawler, E.L., Lenstra, J.K., Kan, A.H.G. Rinnooy, 1979. Optimization and approximation in deterministic sequencing and scheduling: a survey. In: Discrete Optimization II, Vol. 5. Elsevier, pp. 287–326.
Kalczynski, Pawel Jan, Kamburowski, Jerzy, 2005. A heuristic for minimizing the makespan in no-idle permutation flow shops. Comput. Ind. Eng. 49 (1), 146–154.
Öztop, Hande, Tasgetiren, Mehmet Fatih, Kandiller, Levent, Pan, Quan-Ke, 2020. A novel general variable neighborhood search through Q-learning for no-idle flowshop scheduling. In: 2020 IEEE Congress on Evolutionary Computation (CEC). pp. 1–8.
Öztop, Hande, Tasgetiren, M. Fatih, Kandiller, Levent, Pan, Quan-Ke, 2022. Metaheuristics with restart and learning mechanisms for the no-idle flowshop scheduling problem with makespan criterion. Comput. Oper. Res. 138, 105616.
Pan, Quan-Ke, Ruiz, Rubén, 2014. An effective iterated greedy algorithm for the mixed no-idle permutation flowshop scheduling problem. Omega 44, 41–50.
Pan, Quan-Ke, Tasgetiren, Mehmet Fatih, Liang, Yun-Chia, 2008. A discrete differential evolution algorithm for the permutation flowshop scheduling problem. Comput. Ind. Eng. 55 (4), 795–816.
Pan, Quan-Ke, Wang, Ling, 2008a. A novel differential evolution algorithm for no-idle permutation flow-shop scheduling problems. Eur. J. Ind. Eng. 2, 279–297.
Pan, Quan-Ke, Wang, Ling, 2008b. No-idle permutation flow shop scheduling based on a hybrid discrete particle swarm optimization algorithm. Int. J. Adv. Manuf. Technol. 39 (7), 796–807.
Rad, Shahriar Farahmand, Ruiz, Rubén, Boroojerdian, Naser, 2009. New high performing heuristics for minimizing makespan in permutation flowshops. Omega 37 (2), 331–345.
Riahi, Vahid, Chiong, Raymond, Zhang, Yuli, 2020. A new iterated greedy algorithm for no-idle permutation flowshop scheduling with the total tardiness criterion. Comput. Oper. Res. 117, 104839.
Rossi, Fernando Luis, Nagano, Marcelo Seido, 2019. Heuristics for the mixed no-idle flowshop with sequence-dependent setup times and total flowtime criterion. Expert Syst. Appl. 125, 40–54.
Rossi, Fernando Luis, Nagano, Marcelo Seido, 2020. Heuristics and metaheuristics for the mixed no-idle flowshop with sequence-dependent setup times and total tardiness minimisation. Swarm Evol. Comput. 55, 100689.
Ruiz, Rubén, Stützle, Thomas, 2007. A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem. European J. Oper. Res. 177 (3), 2033–2049.
Ruiz, Rubén, Stützle, Thomas, 2008. An iterated greedy heuristic for the sequence dependent setup times flowshop problem with makespan and weighted tardiness objectives. European J. Oper. Res. 187 (3), 1143–1159.
Ruiz, Rubén, Vallada, Eva, Fernández-Martínez, Carlos, 1970. Scheduling in flowshops with no-idle machines. 230, pp. 21–51,
Saadani, Nour El Houda, Guinet, Alain, Moalla, Mohamed, 1999. A travelling salesman approach to solve the f/no-idle/cmax problem. European J. Oper. Res. 161 (1), 11–20.
Shao, Weishi, Pi, Dechang, Shao, Zhongshi, 2017. Memetic algorithm with node and edge histogram for no-idle flow shop scheduling problem to minimize the makespan criterion. Appl. Soft Comput. 54, 164–182.
Shao, Weishi, Pi, Dechang, Shao, Zhongshi, 2018. A hybrid discrete teaching-learning based meta-heuristic for solving no-idle flow shop scheduling problem with total tardiness criterion. Comput. Oper. Res. 94, 89–105.
Shen, Liangshan, Tasgetiren, Mehmet Fatih, Öztop, Hande, Kandiller, Levent, Gao, Liang, 2019. A general variable neighborhood search for the NoIdle flowshop scheduling problem with makespan criterion. In: 2019 IEEE Symposium Series on Computational Intelligence (SSCI). pp. 1684–1691.
Shen, Jing-nan, Wang, Ling, Wang, Sheng-yao, 2014. A bi-population EDA for solving the no-idle permutation flow-shop scheduling problem with the total tardiness criterion. Knowl.-Based Syst. 74.
Ta, Quang Chieu, Billaut, Jean-Charles, Bouquard, Jean-Louis, 2018. Matheuristic algorithms for minimizing total tardiness in the m-machine flow-shop scheduling problem. J. Intell. Manuf. 29 (3), 617–628.
Taillard, E., 1993. Benchmarks for basic scheduling problems. European J. Oper. Res. 64 (2), 278–285, Project Management and Scheduling.
Tasgetiren, M. Fatih, Öztop, Hande, Gao, Liang, Pan, Quan-Ke, Li, Xinyu, 2019. A variable iterated local search algorithm for energy-efficient no-idle flowshop scheduling problem. Procedia Manuf. 39, 1185–1193, 25th International Conference on Production Research Manufacturing Innovation: Cyber Physical Manufacturing August 9-14, 2019 | Chicago, Illinois (USA).
Tasgetiren, M. Fatih, Pan, Quan-Ke, Suganthan, P.N., Chua, Tay Jin, 2011. A differential evolution algorithm for the no-idle flowshop scheduling problem with total tardiness criterion. Int. J. Prod. Res. 49 (16), 5033–5050.
Tasgetiren, M. Fatih, Pan, Quan-Ke, Suganthan, P.N., Oner, Adalet, 2013. A discrete artificial bee colony algorithm for the no-idle permutation flowshop scheduling problem with the total tardiness criterion. Appl. Math. Model. 37 (10), 6758–6779.
Toffolo, Túlio A.M., Santos, Haroldo G., Carvalho, Marco A.M., Soares, Janniele A., 2016. An integer programming approach to the multimode resource-constrained multiproject scheduling problem. J. Sched. 19 (3), 295–307.
Vachajitpan, Porpan, 1982. Job sequencing with continuous machine operation. Comput. Ind. Eng. 6 (3), 255–259.
Wang, Ji-Bo, Xia, Zun-Quan, 2005. No-wait or no-idle permutation flowshop scheduling with dominating machines. J. Appl. Math. Comput. 17 (1), 419.
Woollam, C.R., 1986. Flowshop with no idle machine time allowed. Comput. Ind. Eng. 10 (1), 69–76.
Ying, Kuo-Ching, Lin, Shih-Wei, Cheng, Chen-Yang, He, Cheng-Ding, 2017. Iterated reference greedy algorithm for solving distributed no-idle permutation flowshop scheduling problems. Comput. Ind. Eng. 110, 413–423.
Zhou, Yongquan, Chen, Huan, Zhou, Guo, 2014. Invasive weed optimization algorithm for optimization no-idle flow shop scheduling problem. Neurocomputing 137, 285–292, Advanced Intelligent Computing Theories and Methodologies.