TOP (2024) 32:304–322
https://doi.org/10.1007/s11750-024-00669-9

ORIGINAL PAPER

An ALNS to optimize makespan subject to total completion time for no-wait flow shops with sequence-dependent setup times

Fernando Siqueira de Almeida[1] · Marcelo Seido Nagano[1]

Received: 8 August 2022 / Accepted: 20 February 2024 / Published online: 29 March 2024

Abstract
In this article, we address the no-wait flow shop scheduling problem with sequence dependent setup times. The objective is to minimize makespan subject to an upper bound on total completion time. Although these performance measures and constraints have been extensively studied, they have never been considered together in this problem before. To solve the problem, we propose an adaptive large neighborhood search algorithm called $ALNS_A$. Essentially, $ALNS_A$ improves an initial solution by dynamically selecting and executing a pair of destroy and repair methods based on their performance history. In addition to classic greedy and random methods used, we present two new mechanisms in which the greediness-randomness behavior is balanced. To evaluate performance, the proposed approach is compared with three heuristic methods—*GL*, *HH1* and *TOB*—developed for the most similar problems found in the literature. Computational experiments show that the proposed method outperforms state-of-the-art approaches in the literature for the no-wait flow shop scheduling problem with sequence dependent setup times and is therefore recommended to solve the problem.

Keywords Adaptive large neighborhood search · Flow shop · No-wait · Sequence-dependent setup times

Mathematics Subject Classification 68M20

Fernando Siqueira de Almeida and Marcelo Seido Nagano have contributed equally to this work.

✉ Marcelo Seido Nagano
drnagano@usp.br

Fernando Siqueira de Almeida
fernando.siqueira.almeida@gmail.com

[1] Department of Production Engineering, University of Sao Paulo, Av. Trabalhador Saocarlense, Sao Carlos 13566-590, Sao Paulo, Brazil

1 Introduction

In this article, we address the flow shop scheduling problem (FSP) with two processing constraints: no-wait (NWT) and sequence dependent setup times (SDST). The FSP is a system where a set of jobs have to be processed through a series of machines in the same processing routing. In this system, often the same job sequence occurs on all machines, so that a schedule can be defined by a single permutation of jobs. When the NWT constraint is in place, there is no waiting time between successive operations. Therefore, no job is permitted to utilize a buffer or to wait in an upstream machine. NWT may occur due to process requirements or unavailability of waiting space. Having setup constraints means that a machine requires some preparation before processing a particular job. This preparation may include cleaning, retooling, adjustments, inspection and rearrangement of the work station. In SDST, the length of these times depends on the difficulty involved in switching from one processing configuration to another. SDST are common in multipurpose machines or when a single facility produces a variety of products. In those situations, instead of absorbing the setup times in the processing times, it is recommended to make explicit considerations (Emmons and Vairaktarakis 2013; Pinedo 2016).

The problem is considered with makespan and total completion time. Makespan and total completion time are among the most important performance measures in the field of scheduling. Makespan represents the maximal completion time among all jobs in the system. Minimizing makespan is appropriate when a complete batch of jobs needs to be dispatched as quickly as possible. A reduced makespan also allows an efficient use of resources, that is, it decreases equipment idle time. Total completion time is defined by the sum of all completion times. Schedulers try to minimize total completion time to increase processing rate, which decreases work-in-process inventory and increases the response to demands (Baker and Trietsch 2019). Optimization problems involving two (or more) conflicting objectives, such as these, need to be considered in the presence of trade-offs because no single solution optimizes each objective simultaneously. The approach adopted in this work is to optimize makespan subject to total completion time. This objective function is appropriate when makespan needs to be minimized, but there is no need to optimize total completion time as long as it does not exceed an upper bound. A practical example is when a scheduler sets an upper bound on total completion time to prevent inventory from growing beyond the facility's capacity, but no benefit is gained by reducing it further after achieving this.

Many researchers have proposed algorithms to minimize makespan or total completion time in NWT–FSP–SDST. The most relevant studies include greedy algorithms (Bianco et al 1999; Xu et al 2012), simulated annealing (Lee and Jung 2005), hybrid genetic algorithm (Franca et al 2006), constructive heuristics (Araújo and Nagano 2011; Nagano et al 2015), differential evolution (Qian et al 2011, 2012), greedy randomized adaptive search and evolutionary local search based (Zhu et al 2013b), iterative algorithm (Zhu et al 2013a), hybrid evolutionary cluster search (Nagano and Araújo 2014), hybrid greedy algorithm (Zhuang et al 2014), particle swarm optimization (Samarghandi and ElMekkawy 2014), genetic algorithms (Samarghandi 2015a, b) and local search (Miyata et al 2019).

Despite the extensive research on multi-objective optimization of NWT–FSP, few studies have addressed objectives of type *A subject to B*. Moreover, none of them have considered the SDST feature, to the best of our knowledge. Allahverdi (2004) tried to minimize a linear combination of makespan and maximum tardiness under the condition of not allowing maximum tardiness to exceed a given value. Framinan and Leisten (2006) studied the FSP with the objective of minimizing makespan, such that maximum tardiness is not greater than an acceptable limit. Aydilek and Allahverdi (2012) and Nagano et al (2020) addressed the NWT–FSP with the objective of minimizing makespan under the constraint that mean completion time (or the equivalent total completion time) does not exceed a maximum value. Allahverdi and Aydilek (2013) addressed the NWT–FSP and tried to minimize total completion time while keeping makespan less than or equal to an upper bound, and Allahverdi and Aydilek (2014) considered the same problem with separate setup times. Recently, Allahverdi et al (2018) proposed an algorithm to solve the NWT–FSP with the objective of minimizing total tardiness, such that makespan does not exceed a given value, and Allahverdi et al (2020) studied the same problem with separate setup times.

The NWT–FSP–SDST belongs to the class of NP-*hard* problems (Bianco et al 1999). As exact methods are often impractical or ineffective for solving large or complex problems, heuristic methods are preferable. Heuristics use simple rules and shortcuts to find satisfactory solutions for various combinatorial optimization problems in a reasonable time. Unfortunately, this flexibility comes at a cost: there is no unique optimal parameter setting suitable for all problems and instances. This means that it will always take some effort to properly calibrate the algorithms in each new instance configuration if the scheduler really wants to extract maximum performance. The possibility of improvement can even be assumed to heuristics targeting the most convenient instance configuration, since heuristic solutions are not guaranteed to be optimal. Furthermore, the complexity of the algorithm and its computational time are always properties to be improved. In other words, there will always be possibilities for improvement, whether to get easier, better or faster solutions.

Focusing on these opportunities, we present the algorithm called $ALNS_A$ for the NWT–FSP–SDST to minimize makespan subject to total completion time. This method is based on the adaptive large neighborhood search (ALNS) algorithm originally presented by Ropke and Pisinger (2006), which extends the large neighborhood search (LNS) heuristic proposed by Shaw (1998). In LNS, a destroy and a repair method iteratively rebuild an initial solution in an attempt to improve it. In ALNS, multiple destroy and repair methods are used, and the performance of each destroy/repair method determines how often that particular method is executed during the search. This allows the algorithm to adapt to search conditions, reducing the need for calibration for different instance configurations. ALNS has been mainly applied in vehicle routing problems (VRP), e.g., Hemmelmayr et al (2012), Qu and Bard (2012), Demir et al (2012) and Azi et al (2014). However, the number of scheduling

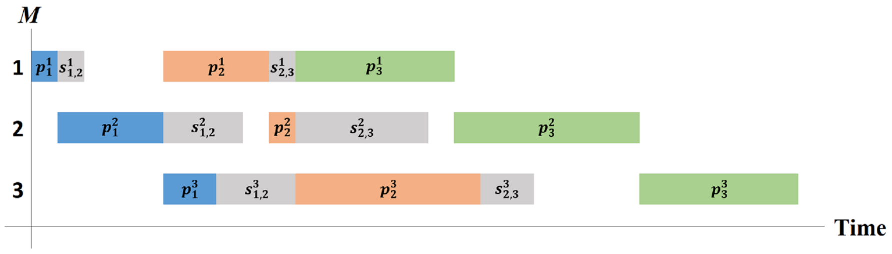


Fig. 1 Gantt chart of an example problem with three jobs and three machines

applications has also grown in recent years, e.g., Lin and Ying (2014), Rifai et al (2016) and Beezão et al (2017). In this work, our proposed $ALNS_A$ accesses a set of distinct mechanisms and dynamically selects a pair of destroy and repair methods based on their performance history. Among the mechanisms used, we present two new ones in which the greediness–randomness behavior is balanced to obtain better results. Extensive experiments are made to compare $ALNS_A$ with three heuristics for similar problems found in the literature. In addition, we present a mathematical model to be used as benchmark for small instances problems. All results are statistically verified.

The remainder of this article is organized as follows. The problem definition is described in Sect. 2. The mathematical model is presented in Sect. 3 and the algorithms are presented in Sect. 4. Section 5 is dedicated to the experimental design and analysis of results. Some concluding remarks are given in Sect. 6.

2 Problem definition

The FSP consists of a set $J = \{i \mid i \in \mathbb{N}, 1 \leq i \leq n\}$ of n jobs which needs to be processed on a set $M = \{k \mid k \in \mathbb{N}, 1 \leq k \leq m\}$ of m machines. Each machine can process only one job at a time. All jobs are processed once on each of the m machines, where the operation O_i^k of job i on machine k is executed during the processing time p_i^k without preemption. Moreover, all jobs follow the same processing order and the job sequence is kept fixed through all machines. It is assumed that all jobs are ready at time 0.

The NWT constraint imposes that operation O_i^{k+1} must begin immediately after operation O_i^k is completed. The SDST constraint implies that, after operation O_i^k is processed, machine k requires a sequence dependent setup s_{ij}^k before being able to start O_j^k. Therefore, setup times are not-attached; that is, a setup process must begin before the next job is ready for a given machine. It is assumed that there is no initial setup required by any machine before starting its first operation, and there is no final setup required to bring any machine back to its initial state after its last operation is executed. Figure 1 shows an example of the NWT–FSP–SDST. In particular, an instance with $n = 3$ jobs and $m = 3$ machines is represented.

Given the set $I = \{q \mid q \in \mathbb{N}, 1 \leq q \leq n\}$ of n positions of a production sequence of J, let t_q^k be the starting time of the qth job on machine k, and c_q^k the completion time of the qth job on machine k. For simplicity, let $t_q = t_q^1$ and $c_q = c_q^m$. Based on the formulation of Bianco et al (1999), we have

$$t_{q+1} = t_q + \max_{1 \leq k \leq m} \left[s_{q,q+1}^k + \sum_{h=1}^{k} \left(p_q^h - p_{q+1}^h \right) + p_{q+1}^k \right], \quad q \in I/\{n\} \tag{1}$$

and

$$c_q = t_q + \sum_{k \in M} p_q^k, \quad q \in I. \tag{2}$$

Equation (1) defines t_{q+1} as t_q plus its distance to t_{q+1}. And since the problem has the no-wait constraint, the completion time c_q of a job is equal to its starting time t_q plus the sum of its processing times, as defined in Eq. (2). Moreover, let C_{max} and TCT represent the performance measures makespan and total completion time, respectively. Hence, we have $C_{max} = c_n$ and $TCT = \sum_{q \in I} c_q$.

The optimization problem modeling the NWT–FSP–SDST minimizing makespan subject to an upper bound on total completion time can be described as follows. The search space is defined by the set of feasible solutions $X = \{\pi \mid TCT(\pi) \leq K\}$, where $TCT(\pi)$ is the total completion time value of the solution π and K is a constant used as upper bound. The objective is to find a solution with the lowest makespan value among all feasible solutions, that is, find a $\pi^* \in X$, so that $C_{max}(\pi^*) \leq C_{max}(\pi)$, $\forall \pi \in X$. Each solution $\pi \in X$ defines a production sequence which we represent by a vector $\pi = (\pi_1, ..., \pi_n)$, being $\pi_i \in J$ and $\pi_i \neq \pi_j$ if $i \neq j$. Clearly this is a permutation problem.

Using the notation proposed by Graham et al (1979) and T'kindt and Billaut (2006) the addressed problem can be written as

$$Fm \ / \ nwt, s_{i,j}^k \ / \ \epsilon(C_{max}/TCT).$$

where $\epsilon(C_{max}/TCT)$ represents the objective function of minimizing C_{max} subject to an upper bound on TCT, while Fm and nwt represent flow shop and no-wait, respectively.

3 Mixed-integer linear programming model

We formulate a mixed-integer linear programming model (MIP) for $Fm \ / \ nwt, s_{i,j}^k \ / \ \epsilon(C_{max}/TCT)$ as follows. Let x_i^q be a binary decision variable, such that $x_i^q = 1$ if job i is the qth job processed, otherwise $x_i^q = 0$. Now, let y_{ij}^q be a binary auxiliary variable, such that $y_{ij}^q = 1$ if job i is the qth job processed immediately before job j, otherwise $y_{ij}^q = 0$. Hence, MIP is defined as follows:

$$z = \min c_n \tag{3}$$

Subject to:

$$\sum_{q\in I} c_q \leq K \tag{4}$$

$$\sum_{i\in J} x_i^q = 1, \quad q \in I \tag{5}$$

$$\sum_{q\in I} x_i^q = 1, \quad i \in J \tag{6}$$

$$\sum_{j\in J} y_{ij}^q = x_i^q, \quad q \in I/\{n\}, \quad i \in J, \quad i \neq j \tag{7}$$

$$\sum_{i\in J} y_{ij}^q = x_i^{q+1}, \quad q \in I/\{n\}, \quad j \in J, \quad i \neq j \tag{8}$$

$$c_q^k = t_q^k + \sum_{i\in J} (p_i^k \cdot x_i^q), \quad q \in I, \quad k \in M \tag{9}$$

$$t_q^{k+1} = c_q^k, \quad q \in I, \quad k \in M/\{m\} \tag{10}$$

$$t_{q+1}^k \geq c_q^k + \sum_{i\in J}\sum_{j\in J} (s_{ij}^k \cdot y_{ij}^q), \quad q \in I/\{n\}, \quad k \in M, \quad i \neq j \tag{11}$$

$$t_1 = 0 \tag{12}$$

$$x_i^q = \{0, 1\}, \quad i \in J, \quad q \in I \tag{13}$$

$$y_{ij}^q = \{0, 1\}, \quad q \in I/\{n\}, \quad i,j \in J, \quad i \neq j \tag{14}$$

$$t_q^k \geq 0, \quad q \in I, \quad k \in M \tag{15}$$

$$c_q^k \geq 0, \quad q \in I, \quad k \in M \tag{16}$$

The objective function (3) minimizes makespan. Constraint (4) imposes the upper bound K on *TCT*. Constraints (5) and (6) are the job assignment constraints, and constraints (7) and (8) are the precedence constraints. Constraints (9) state the right difference between the start time and the end time of a job on a machine according to the job assigned. Constraints (10) guarantee NWT more easily than Eq. (2), because they ensure no gap between successive processing of a job with a simple equality. Constraints (11) ensure the SDST constraints with a simple inequality. This avoids dealing with the "max" term from Eq. (1). Constraint (12) states that the first job on the first machine starts processing at time zero. Constraints (13)–(16) give the domains of the variables.

4 Heuristic algorithms

As mentioned earlier, the problem $Fm/nwt/\epsilon(C_{max}/TCT)$—without SDST constraints—has already been addressed by Aydilek and Allahverdi (2012) and Nagano et al (2020). Ye et al (2020) also proposed an algorithm for a similar environment. In this work, the best algorithm from each study is adapted to be used as a benchmark. The three methods are briefly explained in the next subsection, followed by a complete description of our proposed algorithm $ALNS_A$ to solve the problem (3)–(16).

4.1 Literature algorithms

Aydilek and Allahverdi (2012) proposed the algorithm *HH1*, composed by the algorithms called *mSA* and *HA*. The algorithm *mSA* is a modified simulated annealing, which tries to improve the incumbent solution by iteratively moving a random job to a random position. The algorithm *HA* iteratively applies an insertion local search. Then, it tries to improve the solution generated by using an interchange local search (local search that swaps random pairs of jobs). Only solutions with *TCT* less than or equal to a given upper bound can be candidates to update the incumbent solution. The final solution generated by *mSA* is used as initial solution of *HA*.

Nagano et al (2020) proposed the algorithm *GL* for the same problem. *GL* iterates through a reconstruction procedure followed by an insertion local search. The reconstruction step creates candidate solutions by removing random jobs from the incumbent solution and reinserting them back by using a constructive heuristic based on *NEH* (Nawaz et al 1983). This loop repeats until a defined number of iterations is achieved. Finally, a transposition local search (local search that swap pairs of adjacent jobs) tries to improve the incumbent solution. Throughout the execution, only sequences with *TCT* less than or equal to the upper bound are considered candidate solutions.

Ye et al (2020) proposed the algorithm called *TOB*. First, this method performs a reconstruction process that combines the *NEH* heuristic with an interchange local search. Then, a modified insertion local search is applied (local search that inserts jobs only ahead of its original position). The algorithm repeats these steps until a maximum number of iterations is achieved. The objective function was defined as a mathematical relationship between the two performance measures.

4.2 Algorithm $ALNS_A$

The adaptive large neighborhood search (ALNS), originally proposed by Ropke and Pisinger (2006), has access to a set of destructive and constructive heuristics. At each iteration, a given solution is destroyed by a destructive procedure and rebuilt by a constructive procedure. A weight is assigned to each heuristic according to its performance, giving to those that perform well a higher probability to be selected again. Usually, the weights are updated, so that all heuristics have at least some chance to be executed. This dynamic selection adapts the algorithm to the instance and the state of the search at hand (Pisinger and Ropke 2019).

The proposed ALNS ($ALNS_A$) starts with a permutation π^0 as initial solution. Algorithm is the main structure that uses other heuristics to perform the search, and the performance of these methods determines how often each one is executed. Given the set Ω^- of destroy heuristics and the set Ω^+ of repair heuristic, Algorithm iteratively tries to improve the incumbent solution π^* by using a destroy heuristic $d \in \Omega^-$ and a repair mechanism $r \in \Omega^+$ to create a new candidate solution π', so that $\pi' = r(d(\pi^*))$.[12]

Algorithm 1 $ALNS_A$

```
1: Input: $\lambda$, $\pi^0$, $T_0$, $c_f$;
2: $\rho^- := (1, 1, 1)$; $\rho^+ := (1, 1, 1)$;
3: $\pi := \pi^0$;
4: $\pi^* := \pi^0$;
5: $T := T_0$;
6: repeat
7:     $\pi' := \pi^*$;
8:     select $d \in \Omega^-$ and $r \in \Omega^+$ using $\phi_i^-$ and $\phi_i^+$; [1]
9:     set $\pi' := r(d(\pi'))$ if $r(d(\pi')) \neq empty$;
10:    if $C_{max}(\pi') < C_{max}(\pi^*)$ then
11:        $\pi^* := \pi'$;
12:        $\Psi := 3$;
13:        if $C_{max}(\pi^*) < C_{max}(\pi)$ then
14:            $\pi := \pi^*$;
15:            $\Psi := 4$;
16:        end if
17:    else
18:        if Uniform[0,1] $> exp(-\frac{C_{max}(\pi') - C_{max}(\pi^*)}{C_{max}(\pi^*) \times T})$ then
19:            $\pi^* := \pi'$;
20:            $\Psi := 2$;
21:        else
22:            $\Psi := 1$;
23:        end if
24:    end if
25:    $T := T \times c_f$;
26:    update $\rho^-$ and $\rho^+$ using $\Psi$; [2]
27: until Time Limit
28: return $\pi$;
```

[1] See Eq. (17).

[2] See Eqs. (18)–eqrefeq:rosps.

Algorithm 2 Destroy d

```
1: Input: $\pi$;
2: $\pi^e :=$ empty
3: $i := n/2$;
4: while $i > 0$ do
5:     $\pi^e :=$ insert$_d$ $(\pi, \pi^e)$;
6:     $i := i - 1$;
7: end while
8: return $\pi, \pi^e$;
```

Algorithm 3 Repair r

```
1: Input: $\pi, \pi^e$;
2: while $\pi^e \neq$ empty do
3:     $\pi :=$ insert$_r(\pi^e, \pi)$;
4:     if $TCT(\pi) > K$ then
5:         $\pi :=$ empty
6:         break;
7:     end if
8: end while
9: return $\pi$;
```

The set Ω^- is composed by three destroy algorithms. Destroy algorithm $d = 1$ removes a predefined number of jobs from a sequence at random. Destroy algorithm $d = 2$ tries to remove from a sequence the best set of jobs that optimize the objective. Destroy algorithm $d = 3$ is similar to the previous one, but it differs by skipping some jobs at probability of 50%. These three heuristics are represented by the generic destroy Algorithm 2. The set Ω^+ is composed by three repair algorithms. Repair algorithm $r = 1$ reinserts the removed jobs back to the partial sequence at random positions. Repair algorithm $r = 2$ reinserts each removed job back to the partial sequence in the best possible position. Repair algorithm $r = 3$ is similar to the previous one, but it differs by skipping some insertion options at probability of 50%. These three heuristics are represented by the generic repair Algorithm 3.

Line 5 of Algorithm 2 works as follows.

- $\text{insert}_1(\pi, \pi^e)$ inserts a random job from π into the start of π^e;
- $\text{insert}_2(\pi, \pi^e)$ removes the job that minimizes C_{max} of the remainder of π and insert it at the start of π^e;
- $\text{insert}_3(\pi, \pi^e)$ removes the job that minimizes C_{max} of the remainder of π and insert it at the start of π^e, but skipping some job options at probability of 50%;

Line 3 of Algorithm 3 works as follows.

- $\text{insert}_1(\pi^e, \pi)$ inserts the first job of π^e into a random position of π;
- $\text{insert}_2(\pi^e, \pi)$ inserts the first job of π^e into the best position of π that minimizes its new C_{max};
- $\text{insert}_3(\pi^e, \pi)$ inserts the first job of π^e into the best position of π that minimizes its new C_{max}, but skipping some position options at probability of 50%;

Pure random procedures as $d = 1$ and $r = 1$ have low performance but provide high diversification. On the other hand, pure greedy procedures as Algorithms $d = 2$ and $r = 2$ have high performance but provide low diversification and therefore tend to get stuck in a local optimum. Algorithms $d = 3$ and $r = 3$ have been developed to provide balanced options on these behaviors. The balance is achieved by adding to standard greedy mechanisms the simple but effective procedure of skipping some job manipulations at random, rather than testing every insertion or removal option. If the mechanisms selected are not able to produce a candidate solution π', so that $TCT(\pi') \le K$, the search proceeds to the next iteration without changes.

The parameters $\rho_i^- \in \mathbb{R}^{\|\Omega^-\|}$ and $\rho_i^+ \in \mathbb{R}^{\|\Omega^+\|}$ store the weights of destroy and repair methods, respectively. Equation 17 describes how Algorithm calculates the probabilities ϕ_i^- and ϕ_i^+ to select the ith destroy and repair heuristics, respectively.

$$\phi_i^- = \frac{\rho_i^-}{\sum_{k=1}^{\|\Omega^-\|} \rho_k^-} \quad \phi_i^+ = \frac{\rho_i^+}{\sum_{k=1}^{\|\Omega^+\|} \rho_k^+} \tag{17}$$

After each iteration of Algorithm , a score Ψ is assigned to the methods used, so that

$$\Psi = \begin{cases} 4, & \text{if generated solution is a new global best;} \\ 3, & \text{if generated solution is a new local best;} \\ 2, & \text{if generated solution is accepted;} \\ 1, & \text{if generated solution is rejected.} \end{cases} \tag{18}$$

Initially, we define $\rho^- = (1, 1, 1)$ and $\rho^+ = (1, 1, 1)$. Then, the methods have their weights updated by computing

$$\rho_i^- = \lambda\rho_i^- + (1 - \lambda)\Psi\alpha_i^- \tag{19}$$

and

$$\rho_i^+ = \lambda\rho_i^+ + (1-\lambda)\Psi\alpha_i^+, \tag{20}$$

with $\lambda \in [0,1]$. The parameters α_i^- and α_i^+ represent the values used to normalize the score $\Psi \in \{1,2,3,4\}$ with a measure of the CPU time of the corresponding heuristic. This is important, because some methods can be significantly slower than others, so the normalization ensures a proper trade-off between CPU time and solution quality. Note that the values of Ψ {1, 2, 3, 4} are just a sequence of numbers in ascending order to establish a status ranking. The parameter that really controls the sensibility to change in performance is λ. Furthermore, only the weights ρ_i^- and ρ_i^+ corresponding to the methods used in the current iteration are updated.

5 Computational experiments

We implemented all algorithms in C++ and ran them on a Windows 11 PC with a processor Intel® Core^{TM} i5-11400 H and 16 GB of RAM. The literature methods *HH1*, *GL* and *TOB* were adapted and re-implemented. The source code and scripts can be downloaded from 10.5281/zenodo.10577724. To find optimal solutions for the proposed *MIP*, we employ the Gurobi Optimizer solver version 10.0.2. This solver is widely used in the field of operations research and has proven to be efficient and robust for solving large-scale linear and mixed-integer programming problems.

We defined the stopping criterion for the heuristic methods as $n \times (m/2) \times t$ *milliseconds*, with $t \in \{10, 20, 30, 40\}$ to analyze consistency across different computational times. In real life, the initial solution π^0 and the upper bound K must be given by the scheduler. For our experimental purposes, each π^0 is obtained by optimizing a random permutation through a simple local search that swaps pairs of adjacent jobs and has a neighborhood space of size $(n-1)$. The K value is defined as the total completion time of the initial solution, that is, $K = TCT(\pi^0)$. The same initial solutions and upper bounds were used for all methods for a fair comparison.

The solutions were evaluated by using the Average Relative Percentage Deviation (*ARPD*) defined as

$$ARPD_h = \frac{100}{N}\sum_{i=1}^{N} \frac{C_{max_i}^{\ h} - C_{max_i}^{\ best}}{C_{max_i}^{\ h}}, \tag{21}$$

where h represents the evaluated solution, i is a problem instance, *best* is the best known solution for that instance and N is the number of instances. In summary, this measure represents the arithmetic mean of the deviations from the best solutions found. Therefore, the best method is the one with lowest *ARPD* value.

The experiments are divided in three phases. In the first phase, a parameter tuning for $ALNS_A$ is performed. In the second phase, the heuristics methods are compared with an exact method for small instances. And in the third phase, the heuristics are tested in large instance benchmarks from literature. These three phases are presented in the next subsections.

5.1 $ALNS_A$ parameters tuning

The parameter tuning had been performed for $ALNS_A$ as follows. Parameter λ was calibrated in a set of 120 instances, with 5 different problems for each combination $n \in \{5, 10, 20, 40, 80, 160\} \times m \in \{4, 8, 12, 16\}$ by using the IRACE software package (Lopez-Ibanez et al 2016). We separate the tuning and validation instances to reduce the risk of over-fitting the parameters tuned to the validation instances. This tuning set has processing times with a uniform distribution in the range [1, 99]. The SDST have a uniform distribution in the range [1, 9]. The tuning was performed by using a computation time limit of $n \times (m/2) \times 25$ *milliseconds* as stopping criterion (Ruiz and Stützle 2008). We consider $\lambda \in \{0, 0.1, 0.2, ..., 1\}$ for tuning in IRACE and the result obtained was $\lambda = 0.3$. We obtained the normalization values $\alpha^- = (0.98, 0.3, 0.72)$ and $\alpha^+ = (0.99, 0.44, 0.57)$ by dividing the average time of each method by the maximum time of its set (destroy or repair) after running them 10 times on a random instance. The parameters T_0 (initial temperature), and c_f (cooling factor) control the convergence of the algorithm. Based on the similar approach used in the simulated annealing heuristic presented by Aydilek and Allahverdi (2012), we take directly from these works the values $T_0 = 0.1$ and $c_f = 0.98$.

5.2 Experiment with small instances

In this section, we create a set of instances with 5 different problems for each combination $n \in \{6, 8, 10, 12\} \times m \in \{3, 4, 5, 6\}$. The processing times have a uniform distribution in the range [1, 99]. The set has setup times uniformly distributed in the ranges [1, 9], [1, 49], [1, 99] and [1, 124]. Therefore, 320 different problem instances were generated.

The *ARPD* results with their standard deviations for jobs and machines are presented in Table 1. These values are averages of 80 measures (5 problems $\times$ 4 setups $\times$ 4 times). Obviously, *MIP* has all values equal to zero, as it always produces optimal solutions. Note that $ALNS_A$ performs consistently better when compared to the literature methods and very close to the *MIP* benchmark. Figure 2 illustrates the aggregated *ARPD* and CPU time values for each parameter. As can be seen, the heuristics are not very sensitive to parameter variation, except *GL* which performs better as the number of jobs increases, and worst as the number of machines increases. The methods *HH*1 and *TOB* have the worst performances which are very similar. In general, the exact method can be used until a number of 8 jobs. Above that, its CPU time starts to increase exponentially, so that a heuristic method is recommended. Because *MIP* failed to produce feasible solutions in many cases with stopping criterion, we run it without a time limit and presented its results over time in Figs. 2e and f instead of Fig. 2d. The overall *ARPD* values of $ALNS_A$, *GL*, *HH1*, *TOB* are 0.02%, 0.23%, 1.21% and 1.16%, respectively.

Table 1 ARPD of jobs (n) × machines (m)—[small instances]

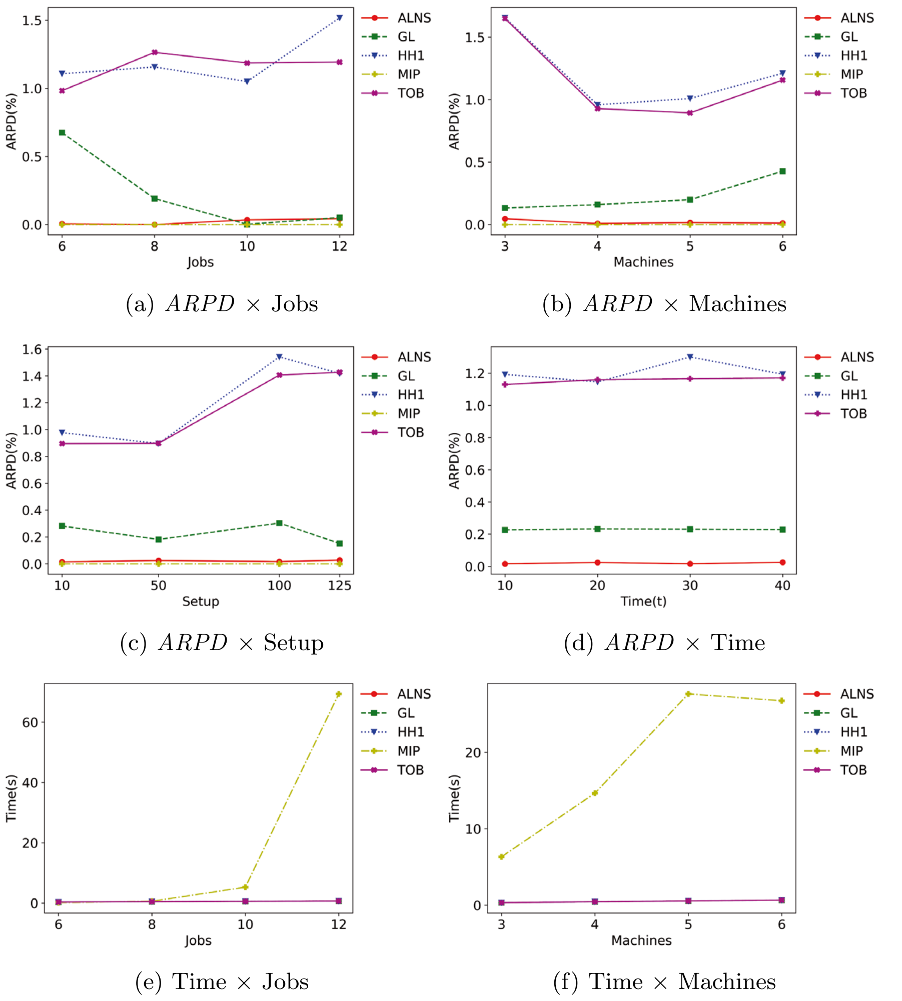
Method*		*ALNS*$_A$		*GL*		*HH1*		*TOB*	
n	*m*	ARPD	STD	ARPD	STD	ARPD	STD	ARPD	STD
6	3	0.00	0.00	0.25	0.83	0.75	1.44	0.99	1.54
	4	0.00	0.00	0.57	1.92	0.70	1.92	0.72	1.92
	5	0.00	0.00	0.37	0.93	0.76	1.50	0.59	1.29
	6	0.02	0.21	1.51	3.66	2.22	3.89	1.63	3.65
8	3	0.00	0.00	0.16	0.75	2.07	3.65	2.01	3.47
	4	0.00	0.00	0.00	0.00	0.83	1.46	0.53	0.98
	5	0.00	0.00	0.41	1.82	1.03	2.14	1.28	2.42
	6	0.00	0.00	0.19	0.83	0.70	1.67	1.25	2.13
10	3	0.10	0.31	0.01	0.05	1.53	1.60	1.87	1.76
	4	0.00	0.00	0.00	0.03	0.87	1.46	1.13	1.06
	5	0.03	0.14	0.00	0.00	0.95	1.14	0.92	1.16
	6	0.01	0.07	0.00	0.00	0.84	0.95	0.82	0.87
12	3	0.09	0.30	0.11	0.31	2.27	2.29	1.72	1.67
	4	0.03	0.12	0.06	0.17	1.43	1.38	1.34	0.85
	5	0.04	0.12	0.02	0.08	1.29	1.41	0.79	0.75
	6	0.01	0.08	0.01	0.05	1.08	1.26	0.92	1.35
Overall		0.02	0.14	0.23	1.26	1.21	2.05	1.16	1.92

*Since *MIP* is an exact method that guarantees the optimal solution, its optimality gaps are zero. Hence, we do not report its results for simplicity

5.3 Experiment with large instances

This phase considers the problem instances of Ruiz and Stützle (2008), which are extensions of the Taillard's benchmark (Taillard 1993). The extensions are divided in four sets, each consisting of 10 problems for each combination $n \times m$ for {20, 50, 100} × {5, 10, 20} and 200 × {10, 20}. The processing times have a uniform distribution in the range [1, 99]. Each set has setup times uniformly distributed in the ranges [1, 9], [1, 49], [1, 99] and [1, 124], respectively. This means that in total, there are 440 different problem instances.

The *ARPD* results for jobs and machines are presented in Table 2. These values are averages of 160 measures (10 problems × 4 setups × 4 times). Table 3 gives the results for setup and time, where the values are averages over 110 measures (the size of each instance set). Note that *ALNS*$_A$ performs consistently better when compared to the literature methods in each table. Figure 3 illustrates the *ARPD* values aggregated for each parameter. As can be seen, the heuristics are not very sensitive to variations on the parameters, except *TOB* which have values disproportionately high especially for 200 jobs (Fig. 3a) and $t = 10$ (Fig. 3c). These are extreme conditions because computational time is the main resource for the search and the amount spend by a heuristic increases exponentially with the number of jobs. It seems that the complex constructive mechanisms of *TOB* have not enough time to finish with these parameters. However, the other methods converge much faster and are



(a) $ARPD \times$ Jobs (b) $ARPD \times$ Machines

(c) $ARPD \times$ Setup (d) $ARPD \times$ Time

(e) Time $\times$ Jobs (f) Time $\times$ Machines

Fig. 2 Aggregated ARPD and CPU times

therefore less influenced by the lack of time. The overall *ARPD* values of $ALNS_A$, *GL*, *HH1*, *TOB* are 0.03%, 1.32%, 2.85% and 4.02%, respectively.

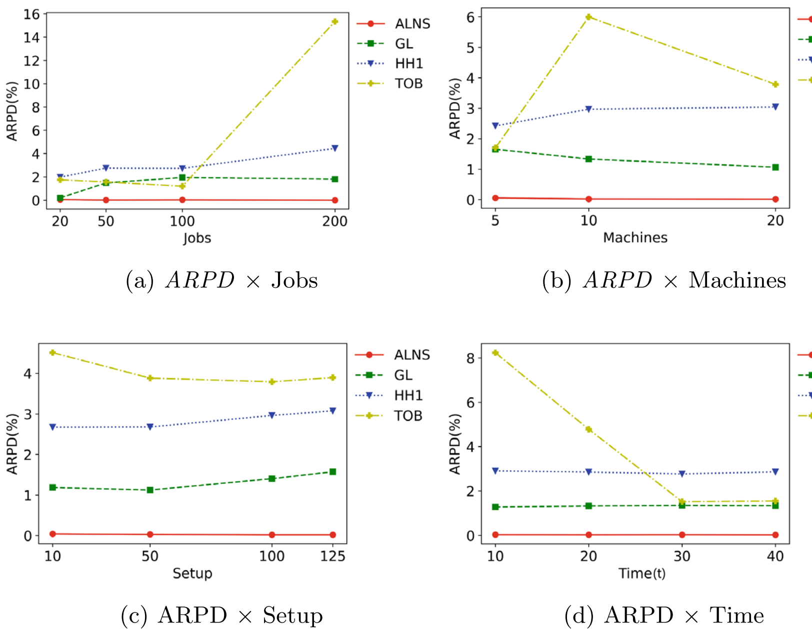
The Tukey' honestly significant difference (HSD) test was conducted to analyze the statistic difference between the methods. The null hypotheses that two algorithms have equal performances was tested at a significance level of 5%. The results in Table 4 show that all algorithms are statistically different from each other. Figure 4 shows the quartile distributions of the results. $ALNS_A$ has a less skewed distribution than the literature methods, indicating its consistent performance. It's worth noting that *TOB* has many outliers with *ARPD* over 50 (omitted to prevent excessive image compression), which mainly reflect its poor performance for $n = 200$ and $t = 10$.

Table 2 ARPD for jobs (n) × machines (m)—[large instances]

Heuristic		$ALNS_A$		GL		$HH1$		TOB	
n	*m*	ARPD	STD	ARPD	STD	ARPD	STD	ARPD	STD
20	5	0.05	0.18	0.46	0.52	2.39	1.40	2.12	1.17
	10	0.07	0.17	0.13	0.23	1.96	1.10	1.82	0.98
	20	0.04	0.14	0.03	0.08	1.59	1.16	1.30	0.87
50	5	0.03	0.15	2.00	0.85	2.44	1.34	1.56	0.99
	10	0.00	0.02	1.40	0.62	2.97	1.16	1.47	0.64
	20	0.01	0.05	1.03	0.49	2.86	0.98	1.68	0.81
100	5	0.08	0.24	2.49	0.86	2.43	1.58	1.42	1.00
	10	0.01	0.05	1.89	0.67	2.81	0.93	1.19	0.67
	20	0.01	0.07	1.47	0.43	2.96	0.90	0.99	0.58
200	10	0.00	0.00	1.91	0.56	4.14	0.80	19.52	18.09
	20	0.00	0.00	1.71	0.37	4.75	1.12	11.16	17.50
Overall		0.03	0.13	1.32	0.96	2.85	1.44	4.02	9.47

Table 3 ARPD for setup (s) × time (t)—[large instances]

Heuristic		$ALNS_A$		GL		$HH1$		TOB	
s	*t*	ARPD	STD	ARPD	STD	ARPD	STD	ARPD	STD
10	10	0.03	0.15	1.16	0.85	2.78	1.88	9.78	18.01
	20	0.04	0.15	1.18	0.88	2.70	1.78	5.33	12.47
	30	0.05	0.20	1.23	0.83	2.61	1.73	1.49	0.86
	40	0.05	0.17	1.17	0.88	2.60	1.55	1.44	0.88
50	10	0.05	0.17	1.08	0.75	2.69	1.60	8.32	15.28
	20	0.03	0.10	1.13	0.83	2.68	1.26	4.64	10.81
	30	0.03	0.11	1.14	0.80	2.68	1.26	1.26	0.64
	40	0.01	0.08	1.14	0.86	2.66	1.38	1.30	0.68
100	10	0.02	0.12	1.31	0.93	3.08	1.35	7.47	12.87
	20	0.01	0.05	1.41	0.95	2.97	1.30	4.56	9.42
	30	0.02	0.10	1.43	1.02	2.87	1.17	1.53	0.83
	40	0.02	0.10	1.46	1.04	2.93	1.20	1.60	0.87
125	10	0.02	0.14	1.56	1.02	3.08	1.38	7.36	11.94
	20	0.02	0.09	1.59	1.09	3.06	1.31	4.60	9.19
	30	0.02	0.13	1.59	1.08	2.92	1.29	1.78	1.03
	40	0.01	0.08	1.56	1.17	3.25	1.25	1.85	1.03
Overall		0.03	0.13	1.32	0.96	2.85	1.44	4.02	9.47



(a) $ARPD \times$ Jobs (b) $ARPD \times$ Machines

(c) ARPD $\times$ Setup (d) ARPD $\times$ Time

Fig. 3 ARPD values aggregated for each parameter

Table 4 Multiple comparison of means—Tukey' HSD test

Methods		Mean difference (*I*–*J*)	Std. error	Sig.	95% CI		Reject
I	*J*				LB	UB	
$ALNS_A$	GL	−1.30	0.166	0.000	−1.73	−0.87	True
	HH1	−2.83	0.166	0.000	−3.25	−2.40	True
	TOB	−4.02	0.166	0.000	−4.44	−3.59	True
GL	$ALNS_A$	1.30	0.166	0.000	0.87	1.73	True
	HH1	−1.53	0.166	0.000	−1.95	−1.10	True
	TOB	−2.72	0.166	0.000	−3.14	−2.29	True
HH1	$ALNS_A$	2.83	0.166	0.000	2.40	3.25	True
	GL	1.53	.166	.000	1.10	1.96	True
	TOB	−1.19	0.166	0.000	−1.61	−0.76	True
TOB	$ALNS_A$	4.01	0.166	0.000	3.59	4.44	True
	GL	2.72	0.166	0.000	2.29	3.14	True
	HH1	1.19	0.166	0.000	0.76	1.61	True

FWER = 0.05

6 Conclusion

In this work, we propose the algorithm called $ALNS_A$ to solve the no-wait flow shop scheduling problem with sequence-dependent setup times to minimize makespan

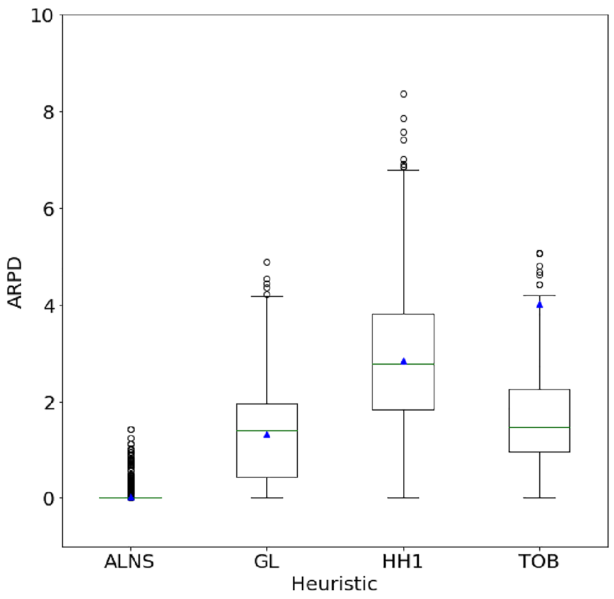


Fig. 4 Boxplot of the ARPD results

subject to an upper bound on total completion time. First, the proposed method was tested with small instance problems and compared with a mixed-integer linear programming model along with three existent algorithms—*GL*, *HH1* and *TOB*—designed to solve the most similar problems found in the literature. This experiment shows that $ALNS_A$,*GL*, *HH1* and *TOB* obtained overall *ARPD* values of 0.02%, 0.23%, 1.21% and 1.16%, respectively. Furthermore, the heuristic methods were tested by using a large instances problems $ALNS_A$,*GL*, *HH1* and *TOB* obtained overall *ARPD* values of 0.03%, 1.32%, 2.85% and 4.02%, respectively. Therefore, $ALNS_A$ outperforms the literature methods.

Despite the superior performance of $ALNS_A$, there are still possibilities for improvement. For example, regarding the heuristic evaluation process, it can be appropriate to assign weights to pairs of methods instead of each method individually because their performances may not be the same across different combinations. Another option is to calibrate more parameters. Here, the randomness levels that determine the number of jobs removed and reinserted by the algorithm have been fixed at 50%. Therefore, it could be possible to increase performance with additional tuning. Algorithms for different applications can be adapted and included in future experiments to further test its efficiency.

The objective function makespan subject to total completion time is an example of how to provide more realistic solutions to scheduling problems that in practice often involve more than one objective. But this approach has a limitation: it does not provide recommendations for the upper bound on total completion time and therefore it must be given by the scheduler. And since the objectives involved are conflicting, this choice must be made carefully, as a tight upper bound makes the search for a feasible solution very difficult.

Acknowledgements This work was supported by the Brazilian National Council for Scientific and Technological Development (Conselho Nacional de Desenvolvimento Científico e Tecnológico - CNPq) [Nos. 312585/2021-7 and 404819/2023-0].

Declarations

Conflict of interest The authors report no declarations of interest.

References

Allahverdi A (2004) A new heuristic for m-machine flowshop scheduling problem with bicriteria of makespan and maximum tardiness. Comput Oper Res 31(2):157–180

Allahverdi A, Aydilek H (2013) Algorithms for no-wait flowshops with total completion time subject to makespan. Int J Adv Manuf Technol 68(9–12):2237–2251

Allahverdi A, Aydilek H (2014) Total completion time with makespan constraint in no-wait flowshops with setup times. Eur J Oper Res 238(3):724–734

Allahverdi A, Aydilek H, Aydilek A (2018) No-wait flowshop scheduling problem with two criteria; total tardiness and makespan. Eur J Oper Res 269(2):590–601

Allahverdi A, Aydilek H, Aydilek A (2020) No-wait flowshop scheduling problem with separate setup times to minimize total tardiness subject to makespan. Appl Math Comput 365(124):688

Araújo DC, Nagano MS (2011) A new effective heuristic method for the no-wait flowshop with sequence-dependent setup times problem. Int J Ind Eng Comput 2(1):155–166

Aydilek H, Allahverdi A (2012) Heuristics for no-wait flowshops with makespan subject to mean completion time. Appl Math Comput 219(1):351–359

Azi N, Gendreau M, Potvin JY (2014) An adaptive large neighborhood search for a vehicle routing problem with multiple routes. Comput Oper Res 41:167–173

Baker KR, Trietsch D (2019) Principles of sequencing and scheduling. Wiley

Beezão AC, Cordeau JF, Laporte G et al (2017) Scheduling identical parallel machines with tooling constraints. Eur J Oper Res 257(3):834–844

Bianco L, Dell'Olmo P, Giordani S (1999) Flow shop no-wait scheduling with sequence dependent setup times and release dates. INFOR: Inform Syst Oper Res 37(1):3–19

Demir E, Bektaş T, Laporte G (2012) An adaptive large neighborhood search heuristic for the pollution-routing problem. Eur J Oper Res 223(2):346–359

Emmons H, Vairaktarakis G (2013) Flow shop scheduling. Theoretical results, algorithms, and applications. Springer Science & Business Media, Cham

Framinan JM, Leisten R (2006) A heuristic for scheduling a permutation flowshop with makespan objective subject to maximum tardiness. Int J Prod Econ 99(1–2):28–40

Franca PM, Tin G Jr, Buriol L (2006) Genetic algorithms for the no-wait flowshop sequencing problem with time restrictions. Int J Prod Res 44(5):939–957

Graham RL, Lawler EL, Lenstra JK et al (1979) Optimization and approximation in deterministic sequencing and scheduling: a survey. Ann Discrete Math, vol 5. Elsevier, pp 287–326

Hemmelmayr VC, Cordeau JF, Crainic TG (2012) An adaptive large neighborhood search heuristic for two-echelon vehicle routing problems arising in city logistics. Comput Oper Res 39(12):3215–3228

Lee YH, Jung JW (2005) New heuristics for no-wait flowshop scheduling with precedence constraints and sequence dependent setup time. In: International Conference on Computational Science and Its Applications, Springer, pp 467–476

Lin SW, Ying KC (2014) Minimizing shifts for personnel task scheduling problems: a three-phase algorithm. Eur J Oper Res 237(1):323–334

Lopez-Ibanez M, Dubois-Lacoste J, Caceres LP et al (2016) The irace package: iterated racing for automatic algorithm configuration. Oper Res Perspect 3:43–58. https://doi.org/10.1016/j.orp.2016.09.002

Miyata HH, Nagano MS, Gupta JN (2019) Integrating preventive maintenance activities to the no-wait flow shop scheduling problem with dependent-sequence setup times and makespan minimization. Comput Ind Eng 135:79–104

Nagano MS, Araújo DC (2014) New heuristics for the no-wait flowshop with sequence-dependent setup times problem. J Braz Soc Mech Sci Eng 36(1):139–151

Nagano MS, Miyata HH, Araújo DC (2015) A constructive heuristic for total flowtime minimization in a no-wait flowshop with sequence-dependent setup times. J Manuf Syst 36:224–230

Nagano MS, Almeida FSd, Miyata HH (2020) An iterated greedy algorithm for the no-wait flowshop scheduling problem to minimize makespan subject to total completion time. Eng Optim, pp 1–19

Nawaz M, Enscore EE Jr, Ham I (1983) A heuristic algorithm for the m-machine, n-job flow-shop sequencing problem. Omega 11(1):91–95

Pinedo M (2016) Scheduling. Theory, algorithms, and systems, 5th edn. Springer Science & Business Media, Cham

Pisinger D, Ropke S (2019) Large neighborhood search. Handbook of metaheuristics. Springer, Cham, pp 99–127

Qian B, Zhou HB, Hu R, et al (2011) Hybrid differential evolution optimization for no-wait flow-shop scheduling with sequence-dependent setup times and release dates. In: International Conference on Intelligent Computing, Springer, pp 600–611

Qian B, Du P, Hu R, et al (2012) A differential evolution algorithm with two speed-up methods for nfssp with sdsts and rds. In: Proceedings of the 10th World Congress on Intelligent Control and Automation, IEEE, pp 490–495

Qu Y, Bard JF (2012) A grasp with adaptive large neighborhood search for pickup and delivery problems with transshipment. Comput Oper Res 39(10):2439–2456

Rifai AP, Nguyen HT, Dawal SZM (2016) Multi-objective adaptive large neighborhood search for distributed reentrant permutation flow shop scheduling. Appl Soft Comput 40:42–57

Ropke S, Pisinger D (2006) An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows. Transp Sci 40(4):455–472

Ruiz R, Stützle T (2008) An iterated greedy heuristic for the sequence dependent setup times flowshop problem with makespan and weighted tardiness objectives. Eur J Oper Res 187(3):1143–1159

Samarghandi H (2015) A no-wait flow shop system with sequence dependent setup times and server constraints. IFAC-PapersOnLine 48(3):1604–1609

Samarghandi H (2015) Studying the effect of server side-constraints on the makespan of the no-wait flowshop problem with sequence-dependent set-up times. Int J Prod Res 53(9):2652–2673

Samarghandi H, ElMekkawy TY (2014) Solving the no-wait flow-shop problem with sequence-dependent set-up times. Int J Comput Integr Manuf 27(3):213–228

Shaw P (1998) Using constraint programming and local search methods to solve vehicle routing problems. In: International Conference on Principles and Practice of Constraint Programming, Springer, pp 417–431

Taillard E (1993) Benchmarks for basic scheduling problems. Eur J Oper Res 64(2):278–285. https://doi.org/10.1016/0377-2217(93)90182-M

T'kindt V, Billaut JC, (2006) An approach to multicriteria scheduling problems. Theory, models and algorithms, multicriteria scheduling, pp 113–134

Xu T, Zhu X, Li X (2012) Efficient iterated greedy algorithm to minimize makespan for the no-wait flowshop with sequence dependent setup times. In: Proceedings of the 2012 IEEE 16th International Conference on Computer Supported Cooperative Work in Design (CSCWD), IEEE, pp 780–785

Ye H, Li W, Nault BR (2020) Trade-off balancing between maximum and total completion times for no-wait flow shop production. Int J Prod Res 58(11):3235–3251

Zhuang WJ, Xu T, Sun MY (2014) A hybrid iterated greedy algorithm for no-wait flowshop with sequence dependent setup times to minimize makespan. In: Advanced materials research, Trans Tech Publ, pp 459–466

Zhu X, Li X, Gupta JN (2013a) Iterative algorithms for no-wait flowshop problems with sequence-dependent setup times. In: 2013 25th Chinese Control and Decision Conference (CCDC), IEEE, pp 1252–1257

Zhu X, Li X, Wang Q (2013b) An adaptive intelligent method for manufacturing process optimization in steelworks. In: Proceedings of the 2013 IEEE 17th International Conference on Computer Supported Cooperative Work in Design (CSCWD), IEEE, pp 363–368