Highlights

- We utilize the pyramidal-shaped property of the ordered flow shop scheduling problem and propose efficient solution methods for the problem with the objective of minimizing the makespan.
- We generate three sets of benchmark consisting of 600 difficult instances, ranging from 10 to 800 jobs and 5 to 60 machines, to be used as the benchmarks for the ordered flow shop scheduling problem.
- We propose two efficient heuristic algorithms for the problem, that are applicable for any scheduling problem with the pyramidal-shaped or V-shaped properties.
- We show that the proposed Pair-Insert heuristic algorithm is very effective and obtains the best solution for 73% of instances, in comparison to the best available heuristic algorithm for the permutation flow shop scheduling problem.
- Our experiments confirm that the proposed Iterated Local Search algorithm obtains the best solution for 75% of instances, in comparison to the best available meta-heuristic algorithm for the permutation flow shop scheduling problem, and the solver CPLEX.

Makespan minimization for the m-machine ordered flow shop scheduling problem

Mostafa Khatami [*] Amir Salehipour [†] F.J. Hwang [‡]

July 3, 2019

Abstract

As a subcategory of the classical flow shop scheduling problem, the ordered flow shop scheduling problem deals with the case where processing times follow a specific structure. This study addresses the m-machine n-job ordered flow shop problem with the objective of minimizing the makespan, which is known to be NP-hard. Two efficient heuristics, and one fast Iterated Local Search algorithm are proposed. The algorithms are developed by utilizing the pyramidal-shaped property of the problem. To the best of our knowledge, there has been no benchmark instances published for the ordered flow shop scheduling problem. This study first generates three sets of benchmarks consisting of 600 instances in total, ranging from 10 to 800 jobs and 5 to 60 machines. The algorithms are tested on those instances. The outcomes demonstrate the efficiency and effectiveness of the proposed algorithms, in particular, the Iterated Local Search algorithm in solving the ordered flow shop scheduling problem. The solutions are also compared with those of the best available algorithm for the permutation flow shop problem, and it is shown that the Iterated Local Search performs better. We also discuss the suitability of the proposed algorithms for scheduling problems with the V-shaped property.

Key words: Ordered flow shop scheduling; Pyramidal-shaped property; V-shaped property; Makespan minimization; Iterated local search; Heuristics

1 Introduction

The ordered flow shop scheduling problem, which was first introduced by Smith (1968), is a subcategory of the classical flow shop scheduling problem, where there are structured properties for processing times. In the classical flow shop problem, the processing times of jobs are usually assumed to be independent of each other, as well as independent of the machines. The jobs' processing times in an industrial environment, however, may be related to the physical characteristics of the jobs and/or machines (Smith et al., 1975). For example, the processing times of jobs may be considered to be correlated because the machine with old design would process the jobs slower than that with the up-to-date technology. As another example, in comparison to the small-size job, the large-size job would usually involve a relatively complicated operation and thus require a relatively long processing time on any machine. In addition, set-up times, transportation times, and learning and aging effects may be dependent, to some extent, on jobs and/or machines. Therefore, scheduling with jobs and machines following those "structured" properties are frequently occurring in real-world situations. In the ordered flow shop scheduling problem, the structured properties of jobs and machines can be described by the following two conditions, which apply to all jobs and machines: (1) if a job's processing time is smaller than that of another job on some machine, then it should be the case on all machines, and (2) if the processing time of a job on a machine

[*]School of Mathematical and Physical Sciences, University of Technology Sydney, Australia. Email: mostafa.khatami@student.uts.edu.au

[†]School of Mathematical and Physical Sciences, University of Technology Sydney, Australia. Email: amir.salehipour@uts.edu.au

[‡]School of Mathematical and Physical Sciences, University of Technology Sydney, Australia. Email: feng-jang.hwang@uts.edu.au

is smaller than that on another machine, then it should be the case for all jobs. The first condition is called the job-ordered condition, and the second one is named the machine-ordered condition. We shall later formally state those conditions.

The contributions of this research can be summarized as follows: (1) two efficient heuristic algorithms are proposed for the problem; (2) a fast and effective Iterated Local Search (ILS) algorithm benefiting from the properties of the ordered flow shop scheduling problem is developed; (3) three sets consisting of 600 challenging instances, which are selected from 520,800 randomly generated instances, are produced to serve as the benchmark instances for the ordered flow shop scheduling problem; (4) it is shown that the heuristic algorithms solve the largest instances within a second; and, (5) the proposed ILS algorithm is shown to outperform the best available algorithm for the permutation flow shop problem, and also the solver CPLEX, in terms of solution quality and computation time.

The remainder of this paper is organized as follows. Section 2 defines the problem and notations used in the paper, as well as discussing a mixed-integer linear programming formulation for the ordered flow shop scheduling problem. Review of the literature and real-world applications of the problem are presented in Section 3. In Section 4, we propose solution methods for solving the ordered flow shop scheduling problem, including two heuristics and an ILS algorithm. We also discuss the applicability of the algorithms to scheduling problems with similar properties. In Section 5, we report the computational performance of the proposed algorithms. We first discuss a procedure for generating three sets of benchmark instances for the ordered flow shop scheduling problem, and then we detail the computational outcomes. The conclusions and future research directions are provided in Section 6.

2 Problem definition and formulation

Given a set of jobs $J = \{1, \dots, n\}$ and a set of different machines $M = \{1, \dots, m\}$, we denote the processing time of job j on machine r by $p_{r,j}$, where $r \in M$ and $j \in J$. The ordered flow shop scheduling problem is characterized by the following two conditions:

1. if for any two jobs $j, k \in J, p_{r,j} < p_{r,k}, r \in M$, then $p_{q,j} \leqslant p_{q,k}, \forall q \in M$; and,
2. if for any two machines $r, q \in M, p_{r,k} < p_{q,k}, k \in J$, then $p_{r,j} \leqslant p_{q,j}, \forall j \in J$.

We assume that the job processing orders on all m machines are identical, i.e. only permutation schedules are allowed. Also, all jobs are assumed to be available at time zero, and preemption of jobs is not allowed, i.e. once the processing of a job is started it cannot be interrupted by other jobs. Finally, each machine can process at most one job at a time. The ordered flow shop scheduling problem aims to obtain the best order of performing jobs on machines (a permutation) with respect to some criteria. This paper considers the objective function of minimizing the makespan denoted by $C_{\max}$, which is the completion time of the last job in a schedule. Table 1 summarizes the notations used in this paper for the ordered flow shop scheduling problem.

Table 1: Mathematical notations used in this paper.

Notation	Description
Sets:	
J	Set of jobs, $J = \{1, \dots, n\}$, indexed by j.
M	Set of machines, $M = \{1, \dots, m\}$, indexed by r.
Parameters:	
n	Number of jobs and positions.
m	Number of machines.
$p_{r,j}$	Processing time of job j on machine r, $p_{r,j} \in \mathbb{Z}^+$.
Decision variables:	
$z_{j,i}$	Takes a value of 1 if job j is assigned to position i, and 0 otherwise, $z_{j,i} \in \{0, 1\}, \forall i, j \in J$.
$s_{r,i}$	Start time of the job in position i on machine r, $s_{r,i} \geqslant 0, \forall r \in M, i \in J$.

An example of the ordered flow shop, which is borrowed from Panwalkar and Woollam (1980), is shown in Table 2. Note that the processing times follow both conditions discussed above.

Table 2: An instance of a 5-job 6-machine ordered flow shop scheduling problem (Panwalkar and Woollam, 1980).

	Machine					
Job	1	2	3	4	5	6
1	2	3	15	3	8	8
2	3	4	21	7	13	15
3	9	12	30	15	19	21
4	13	14	34	19	24	26
5	15	16	37	24	28	35

It is known that a pyramidal-shaped sequence is optimal for the ordered flow shop scheduling problem (Smith et al., 1976). A pyramidal-shaped sequence consists of two sub-sequences, in which the jobs in the first sub-sequence are ordered by the shortest processing time (SPT) dispatching rule, and the jobs in the second sub-sequence are sequenced in the longest processing time (LPT) order. Those two sub-sequences will be hereafter called SPT sub-sequence and LPT sub-sequence, respectively. We note that the V-shaped property implies that the jobs in the first and second sub-sequences are ordered in the LPT and SPT orders, respectively.

To better illustrate the pyramidal-shaped property of the problem, we present some examples in the following. Based on the instance presented in Table 2, $(J_1, J_3, J_5, J_4, J_2)$ and $(J_4, J_5, J_3, J_2, J_1)$ are two pyramidal-shaped sequences. On the other hand, $(J_3, J_5, J_2, J_4, J_1)$ is not a pyramidal-shaped sequence because J_2 is between J_4 and J_5 (note that $p_{q,2} \leqslant p_{q,5}$ and $p_{q,2} \leqslant p_{q,4}, \forall q \in M$ that breaks the top of the sequence with a V shape). Similarly, $(J_5, J_3, J_1, J_2, J_4)$ and $(J_5, J_4, J_3, J_1, J_2)$ are two V-shaped sequences.

The available mixed-integer linear programming (MILP) formulations for the flow shop scheduling problem can be utilized to solve the ordered flow shop scheduling problem. This is because the aforementioned job-ordered and machine-ordered conditions apply to the job processing times, which are the given parameters, and play no role in the MILP formulation. Among different MILP formulations available for the flow shop scheduling problem we implement the "starting time-based" formulation proposed by Wilson (1989), which according to Tseng et al. (2004), is one of the best performing models for solving the permutation flow shop scheduling problem. In this formulation, which is represented by Model I in the following, a binary decision variable $z_{j,i}$ takes a value of 1 if job j is assigned to position $i \in J$ in the sequence, and 0 otherwise. In addition, the non-negative decision variable $s_{r,i}$ indicates the starting time of the job in position i on machine r. Based on those notations, the Wilson's model for the flow shop scheduling problem can be presented by Model I:

Model I

$$z = \min C_{max} = \min(s_{m,n} + \sum_{j=1}^{n} p_{m,j} z_{j,n}) \tag{1}$$

subject to

$$\sum_{j=1}^{n} z_{j,i} = 1, \quad 1 \leqslant i \leqslant n, \tag{2}$$

$$\sum_{i=1}^{n} z_{j,i} = 1, \quad 1 \leqslant j \leqslant n, \tag{3}$$

$$s_{1,1} = 0, \tag{4}$$

$$s_{1,i} + \sum_{j=1}^{n} p_{1,j} z_{j,i} = s_{1,i+1}, \quad 1 \leqslant i \leqslant n-1, \tag{5}$$

$$s_{r,1} + \sum_{j=1}^{n} p_{r,j} z_{j,1} = s_{r+1,1}, \quad 1 \leqslant r \leqslant m-1, \tag{6}$$

$$s_{r,i} + \sum_{j=1}^{n} p_{r,j} z_{j,i} \leqslant s_{r+1,i}, \quad 1 \leqslant r \leqslant m-1, \quad 2 \leqslant i \leqslant n, \tag{7}$$

$$s_{r,i} + \sum_{j=1}^{n} p_{r,j} z_{j,i} \leqslant s_{r,i+1}, \quad 2 \leqslant r \leqslant m, \quad 1 \leqslant i \leqslant n-1, \tag{8}$$

$$z_{j,i} \in \{0,1\}, \quad 1 \leqslant j \leqslant n, \quad 1 \leqslant i \leqslant n, \tag{9}$$

$$s_{r,i} \geqslant 0, \quad 1 \leqslant r \leqslant m, \quad 1 \leqslant i \leqslant n. \tag{10}$$

The objective function (eq. (1)) minimizes the completion time of the last job in the sequence on the last machine, i.e. the makespan. Constraints (2) and (3) are of the assignment problem, and ensure that exactly one job is assigned to each position and exactly one position is assigned to each job. Constraints (4) set the starting time of the schedule to zero. Constraints (5) and (6) ensure that there is no idle time on the first machine, and the first job in the sequence is processed on all m machines without delay. Constraints (7) indicate that the starting time of each job on machine $r+1$ is no earlier than its completion time on machine r. Constraints (8) ensure that the job in position $i+1$ does not start on machine r until the job in position i has completed its processing on that machine. Finally, constraints (9) and (10) state decision variables are binary and non-negative.

3 Literature review and applications

In this section we first review major studies on the ordered flow shop scheduling problem that were published after 2013. For a detailed review of earlier studies, we refer the interested reader to the review paper by Panwalkar et al. (2013). Next, we present the real-world applications of the problem.

3.1 Literature review

Followed by the pioneering work of Smith (1968) many studies have investigated the ordered flow shop scheduling problem with the objective of minimizing the makespan. Smith et al. (1975) showed that the LPT first dispatching rule is optimal when the largest processing times occur on the first machine. Analogically, the SPT first dispatching rule is optimal for the problem when the largest processing times occur on the last machine. However, when the largest processing times do not occur on either the first or the last machine, the problem is NP-hard in the ordinary sense (Hou and Hoogeveen, 2003). In particular, they considered the three-machine problem where the time for which a job occupies a machine is equal to the processing time of the job divided by the speed of the machine. Then they proved that the problem is NP-hard if the second machine is the slowest. Panwalkar et al. (2013) also discussed that the three-machine ordered flow shop scheduling problem, in which the largest processing times occur on the second machine is NP-hard.

Smith et al. (1976) showed that a pyramidal-shaped sequence is optimal for the m-machine n-job ordered flow shop scheduling problem. The optimality is proved by observing that the critical path passes through the jobs with the largest processing times. Those theoretical results further helped them to transform the problem into a partitioning problem, and therefore, the number of sequences needed for an exhaustive enumeration was reduced from $n!$ to 2^{n-1}. Panwalkar and Khan (1977) demonstrated that the makespan is a piecewise convex function of the position of the job with the largest processing times. Note that the number of sequences is at most $\binom{n-1}{n/2}$ if the position of the job with the largest processing times is fixed.

Considering the makespan minimization criterion, Choi and Park (2016) studied the m-machine ordered flow shop scheduling problem with two competing agents. In their setting each agent aims at minimizing the makespan for his own set of jobs. They developed a pseudo-polynomial time algorithm to solve the problem. In addition, they showed that a three-machine case of the problem is polynomially solvable if the time for which a job occupies a machine is equal to the processing time of the job plus a setup time required by the machine. A similar problem with two machines was studied by Kim et al. (2017), where the objective is to minimize the makespan for the set of jobs allocated to one agent, while ensuring the makespan of the jobs allocated to the other agent does not exceed a given bound. They investigated three cases, for which processing times follow certain structures. Lee et al. (2019) studied the on-line variant of the m-machine flow shop scheduling problem with the objective of minimizing the makespan. In the on-line variant jobs arrive at different times. In particular, they studied the problems with machine-ordered assumptions, and also with job-ordered assumptions. They analyzed competitive ratio of a greedy algorithm, and derived lower bounds on the ratio for several cases.

The makespan minimization is also considered for the ordered flow shop scheduling problem in the no-wait environment, in which once a job is started it must go through machines without any delay. The strong NP-hardness of the makespan minimization in the three-machine no-wait flow shop was proved by Röck (1984). However, when ordered processing times are considered, Panwalkar and Woollam (1979) showed that the SPT (LPT) sequence is optimal for the m-machine setting if the largest processing times occur on the last (first) machine. Recently, Koulamas and Panwalkar (2018) derived a new priority rule for the no-wait ordered flow shop scheduling problem with m-machines. In particular, they studied the job-ordered case where the largest processing time of each job occurs on either the first or the last machine. For the objective of minimizing the makespan, they showed that the problem is optimally solvable with the proposed rule in $O(n \log n)$ time.

Other objective functions such as the total completion time and number of tardy jobs have been studied for the ordered flow shop scheduling problem. For example, Panwalkar and Koulamas (2013) studied a three-machine ordered flow shop scheduling problem with flexible ordering, i.e. the order of performing jobs can be changed to some extent. The problem is to determine the optimal layout and schedule with respect to both makespan and total completion time criteria. They showed that the Small-Median-Large layout and sequencing the jobs based on the SPT first dispatching rule is the best design for the problem. A job selection problem in the two-machine shop scheduling was studied by Koulamas and Panwalkar (2015). They utilized an existing $O(n^2)$ algorithm for the makespan criterion, and solved the two-machine flow shop problem with ordered machines and objective function of minimizing the total completion time. They also studied similar problems in the open shop and job shop environments. Panwalkar and Koulamas (2017) studied the permutation schedules for some special cases of the ordered flow shop scheduling problem with the objective functions of minimizing the makespan and the total completion time. They also showed that a non-permutation schedule is dominant for a class of four-machine ordered flow shop, for both makespan and total completion time criteria. A due date related objective function was studied by Panwalkar and Koulamas (2012). They attempted to obtain the shortest common due date such that the number of tardy jobs is minimum. They proposed an $O(n^2)$ algorithm for the problem when machines are ordered. Later, Ilić (2015) improved their algorithm and reduced its complexity to $O(n \log n)$.

An outsourcing and scheduling variant of two-machine ordered flow shop scheduling problem, in which each job is processed either in-house or outsourced to a subcontractor, was studied by Chung and Choi (2013). Because this problem was proved to be NP-hard (Choi and Chung, 2011) they proposed an approximation algorithm for the cases of minimizing the sum of makespan and the total outsourcing cost. In addition, they studied three special cases, aiming to model processing requirements of jobs on machines. They showed that only one case is polynomially solvable and the other two cases are NP-hard.

Park and Choi (2015) considered uncertain processing times in the m-machine flow shop scheduling problem. Since the general problem with the objective function of minimizing the maximum deviation from the optimality for all scenarios is proved to be NP-hard (Kouvelis and Daniels, 2000), they studied the problem with ordered processing times. Particularly, they set the processing time with a function of one job-dependent parameter and one machine-dependent parameter. Given a fixed number of machines and scenarios, they developed a pseudo-polynomial time solution method. Table 3 and Table 4 summarize the relevant studies on the ordered flow shop scheduling problem that we discussed above. We note that

$F2, F3$ and Fm in the shop setting column in Table 4 denote the flow shop problem with 2, 3 and m machines.

Table 3: Fundamental studies for the makespan minimization in the m-machine ordered flow shop.

Study	Main outcomes
Smith (1968)	Foundations of the ordered flow shop scheduling problem.
Smith et al. (1975)	SPT (LPT) is optimal when the last (first) machine performs the largest operation of jobs (i.e. with the largest processing time).
Smith et al. (1976)	A pyramidal-shaped (SPT followed by LPT) sequence is optimal for the problem.
Panwalkar and Khan (1977)	The makespan is a piecewise convex function of the job with largest processing times.
Panwalkar and Woollam (1979)	In the no-wait variant the SPT (LPT) is optimal when the last (first) machine performs the largest operation of jobs (similar to above).

Table 4: Summary of major studies published after Panwalkar et al. (2013).

Study	Shop setting	Special setting	Objective function(s) (minimization)	Outcomes
Chung and Choi (2013)	$F2$	Outsourcing	Makespan and outsourcing cost	Approximation algorithms
Panwalkar and Koulamas (2013)	$F3$	Flexible machine ordering	Makespan and total completion time	Optimal layout and schedule
Ilić (2015)	$F2$	-	Common due date and number of tardy jobs	An $O(n \log n)$ algorithm
Koulamas and Panwalkar (2015)	Fm	Job selection	Total completion time	An $O(n^2)$ algorithm
Park and Choi (2015)	Fm	Uncertain processing times	Maximum deviation of makespan for all scenarios	A pseudo-polynomial time algorithm
Choi and Park (2016)	Fm	Two-agent scheduling	Makespan	A pseudo-polynomial time algorithm
Kim et al. (2017)	$F2$	Two-agent scheduling	Makespan	Complexity of special cases
Panwalkar and Koulamas (2017)	Fm	-	Makespan and total completion time	Dominance of permutation schedules
Koulamas and Panwalkar (2018)	Fm	No-wait flow shop	Makespan	Priority rules for the job-ordered problem
Lee et al. (2019)	Fm	On-line scheduling	Makespan	Competitive ratio for the greedy algorithm

3.2 Applications

The first study introducing the practical importance of the ordered flow shop scheduling problem is due to Smith (1968). They discussed that several factors, including the number of units or parts of a job or the complexity of each job on all machines may lead to structural ordering of the jobs' processing times, and therefore, resulting in an ordered flow shop scheduling problem.

By investigating a large number of plants, Panwalkar et al. (1973) realized that in 77% of them the products of similar types or sizes have similar trends for the processing times. In other words, if a job has a longer processing time on one machine than other machines, then other jobs will also have longer processing times on that particular machine. Two of such industries were recently discussed by Lee et al. (2019). The first example includes a semiconductor and liquid crystal display (LCD) panels manufacturing facility, in which wafers and glasses are processed in batches over a series of stages. The processing time for each batch is proportional to the number of wafers or LCDs in the batch. The second example is the painting process of expensive wooden doors. Several layers of paint and varnish may be required for such doors. A different machine is associated with each different layer of paint or varnish. The processing times are usually dependent on the size of a door. In addition, the machines may have different setup times.

In addition to the real-world applications of the ordered flow shop scheduling problem, the theoretical aspect of the problem is also important, particularly in modeling other production environments. For example, Şen et al. (1998) showed that a single-job lot-streaming problem in a two-machine flow shop turns to be an ordered problem, if the size of the sub-lots is given. In fact, this is similar to the ordered assumptions, particularly, if a sub-lot is larger and has the largest processing time than another sub-lot on a particular machine, then this is the case on all machines. Likewise, if a particular sub-lot is larger on some machine than on another machine, then the machine-ordered condition applies to all other sub-lots.

4 Proposed solution methods

Smith et al. (1976) showed that a pyramidal-shaped sequence is optimal for the m-machine n-job ordered flow shop scheduling problem. That is, an optimal job sequence can be split into two sub-sequences, in which the first sub-sequence includes a non-decreasing order of job processing times, i.e. the SPT rule, and the second one follows a non-increasing order of job processing times, i.e. the LPT order. Thanks to the pyramidal-shaped property, the size of the set of all candidate feasible solutions significantly reduces from $n!$ to 2^{n-1}. Hence, it is sensible to utilize the pyramidal-shaped property in the development of solution methods for the ordered flow shop scheduling problem. In this study, we utilize the pyramidal-shaped property in developing problem-specific heuristics and meta-heuristics. The heuristic algorithms include a pyramidal variant of the Nawaz-Enscore-Ham (NEH) algorithm (Nawaz et al., 1983) and a procedure named "Pair-Insert". We also develop an Iterated Local Search (ILS) meta-heuristic.

Note that the pyramidal-shaped property is related to the well-known V-shaped property, because in both an optimal job sequence consists of two sub-sequences. However, the V-shaped property implies that the first sub-sequence follows the LPT order, and the second one is in the SPT order. Solution approaches to scheduling problems with the V-shaped property have been developed in several studies. For example, Mittenthal et al. (1993) proposed a Simulated Annealing-based procedure and Al-Turki et al. (2001) developed a Tabu Search algorithm for the single machine scheduling problems with the V-shaped property.

4.1 The Pyramidal-NEH algorithm

The well-known NEH algorithm is one of the most effective heuristics for solving the permutation flow shop scheduling problem. Ruiz and Maroto (2005) showed this through a comprehensive evaluation of available heuristics for the permutation flow shop scheduling problem. Kalczynski and Kamburowski (2007) also investigated the superiority of the NEH algorithm. We refer the interested reader to Dong et al. (2008)

and Kalczynski and Kamburowski (2008) for more details on the applications of the NEH algorithm and its variants.

The NEH algorithm consists of three steps. In Step 1, a sequence π_0 is generated by sorting the n jobs in a non-increasing order of the total processing times of m operations. In Step 2, the first two jobs from π_0 are selected, and a sequence π for the two jobs with the minimum makespan is generated. A complete sequence is then constructed in Step 3, where the remaining jobs of π_0 are inserted in the sequence π one by one, such that a job is inserted into a position, among all possible positions, which minimizes the makespan of the yielded partial sequence. Note that there are i possible positions for inserting job $3 \leqslant i \leqslant n$ into the partial sequence. The NEH algorithm has a computational time of $O(n^2m)$, which was improved from $O(n^3m)$ by Taillard (1990), denoted as Taillard's acceleration.

Note that the procedure of Step 3 may generate a large number of ties. Fernandez-Viagas and Framinan (2014) showed that their idle time-based tie-breaking mechanism outperforms all other mechanisms. Their mechanism does not alter the $O(n^2m)$ complexity of the NEH algorithm.

Because a pyramidal-shaped sequence is optimal for an ordered flow shop scheduling problem (Smith et al., 1976), we therefore propose a modification to Step 3 of the original NEH algorithm to improve the efficiency for solving the ordered flow shop scheduling problem. The modification, which is based on the pyramidal-shaped property, aims to restrict the search space of the algorithm to only pyramidal-shaped sequences. Our proposed solution procedure is thus named the Pyramidal-NEH algorithm and is shown in Algorithm 1.

Algorithm 1: The Pyramidal-NEH algorithm.

```
1  Input: The processing times of $n$ jobs on all the $m$ machines.
2  Output: A job sequence $\pi$.
3  Step 1: Generate a sequence $\pi_0$ by sorting the $n$ jobs in the non-increasing order of the total processing
      times of $m$ operations;
4  Step 2: Select from $\pi_0$ the first two jobs, and generate a sequence $\pi$ of the two jobs with the minimum
      makespan;
5  Step 3 (Complete sequence construction):
6  for $3 \leqslant i \leqslant n$ do
7      Generate $\pi'$ and $\pi''$ by letting $\pi$ be prefixed and suffixed respectively with the $i$th job of $\pi_0$;
8      Compare the makespans of $\pi'$ and $\pi''$, and set $\pi$ as the one with smaller makespan;
9  end
10 return the constructed sequence $\pi$
```

We note that in case of ties, Algorithm 1 chooses the sequence π'. The Pyramidal-NEH algorithm restricts the allocation of an unassigned job only to the first or the last position of the partially built sequence, maintaining therefore the pyramidal-shaped property of the final sequence. Note that in Step 3 because an unassigned job has smaller total processing time than all already assigned jobs, hence, the pyramidal-shaped property is maintained if the unassigned job is placed in either the first position or the last position of the sequence. For example, consider the instance presented in Table 2. Assume that the partial sequence is $\pi = (J_5, J_4)$. In order to maintain the pyramidal property of the sequence, the next job, i.e. J_3 can only be inserted into either the first or the last position of the sequence. This leads to either $\pi = (J_3, J_5, J_4)$ or $\pi = (J_5, J_4, J_3)$. We note that the sequence $\pi = (J_5, J_3, J_4)$ is therefore ignored since it is not pyramidal. Due to this, in the Pyramidal-NEH algorithm $2(n-1)$ partial sequences are investigated in order to build a complete solution, whereas $\frac{n\times(n+1)}{2} - 1$ partial sequences need be considered in the original NEH algorithm. The time complexities of Steps 1 and 2 of Algorithm 1 are $O(n \log n)$ and $O(m)$, identical to those of the NEH algorithm. Step 3, however, performs $2mn$ operations when calculating the makespan of the partial sequence at each iteration. Therefore, the time complexity of the Pyramidal-NEH algorithm is $O(n^2m)$.

4.2 The Pair-Insert algorithm

Another heuristic algorithm that we propose for the ordered flow shop scheduling problem is named the “Pair-Insert” algorithm. The Pair-Insert algorithm utilizes the pyramidal-shaped property as well as the general idea of the NEH algorithm. Algorithm 2 summarizes the proposed Pair-Insert algorithm. Performing the same first two steps as Algorithm 1, the Pair-Insert algorithm constructs a partial sequence in Step 3 by adding unassigned jobs in pairs. The pyramidal-shaped property of the problem leaves only four possible choices for concatenating a pair of unassigned jobs with a partial sequence. Let π_0 be the sequence obtained in Step 1 and π be a partial sequence constructed with the first $i-1$ jobs of π_0. Then the four choices of concatenation are: (1) $(i+1, i, \pi)$, i.e. prefix both jobs $i+1$ and i to π; (2) $(\pi, i, i+1)$, i.e. suffix both jobs $i+1$ and i to π; (3) $(i, \pi, i+1)$, i.e. prefix job i and suffix job $i+1$ to π; and, (4) $(i+1, \pi, i)$, i.e. prefix job $i+1$ and suffix job i to π. Among those four permutations, the one with the least makespan is selected as the partial sequence constructed with the first $i+1$ jobs of π_0. Consider the instance of Table 2. If $\pi = (J_5, J_4)$, the two upcoming jobs, i.e. J_3 and J_2 can be added to the partial sequence as the followings in order to maintain the pyramidal property: $\pi = (J_2, J_3, J_5, J_4)$, $\pi = (J_5, J_4, J_3, J_2)$, $\pi = (J_3, J_5, J_4, J_2)$ and $\pi = (J_2, J_5, J_4, J_3)$.

Algorithm 2: The Pair-Insert algorithm.

```
1  Input: The processing times of $n$ jobs on all the $m$ machines.
2  Output: A job sequence $\pi$.
3  Step 1: Generate a sequence $\pi_0$ by sorting the $n$ jobs in the non-increasing order of the total processing
      times of $m$ operations;
4  Step 2: Select from $\pi_0$ the first two jobs, and generate a sequence $\pi$ of the two jobs with the minimum
      makespan;
5  Step 3 (Complete sequence construction):
6  Set $i = 3$;
7  while $i \leqslant n$ do
8      Generate four partial sequences by concatenating $\pi$ with the $i$th and $(i+1)$th jobs of $\pi_0$ in accordance
       with (1): $(i+1, i, \pi)$, (2): $(\pi, i, i+1)$, (3): $(i, \pi, i+1)$, and (4): $(i+1, \pi, i)$;
9      Compare the makespans of the four permutations, and set $\pi$ as the one with the smallest makespan;
10     Set $i = i + 2$;
11 end
12 return the constructed sequence $\pi$
```

Notice that in case of ties, the algorithm chooses the sequence with the smallest index among the four choices. Similar to the Pyramidal-NEH algorithm, the Pair-Insert algorithm also runs in $O(n^2m)$ and the total number of partial sequences to be evaluated is equal to $2(n-1)$. The main advantage of the Pair-Insert algorithm lies in the fact that it could be applied to the scheduling problems with the pyramidal-shaped or V-shaped properties, such as the single machine scheduling problems with linear deteriorating jobs for the total completion time minimization (Mosheiov, 1991) and for the objective function of total absolute differences in completion times (Li et al., 2009), and several classes of common due date assignment problems (Gordon et al., 2002).

4.3 Extending the proposed heuristics for problems with the V-shaped property

As we discussed earlier, our proposed heuristics can be extended to the scheduling problems with the V-shaped property as well. In that case, Steps 1 and 2 of the Pyramidal-NEH and Pair-Insert algorithms remain the same, although the sorting may be adjusted following the objective function of those problem. Step 3, however, must be modified.

When the Pyramidal-NEH algorithm is applied to a problem with the V-shaped property, every unassigned job can be allocated either to the left position of the most recent assigned job with the smallest total processing time, or to its right position. Applying the Pair-Insert algorithm to the problems with the V-shaped property is slightly different. Let l be the job with the smallest total processing time in the partial sequence constructed by the first $i-1$ jobs. Then, the four alternatives for concatenation include $(\dots, i+1, i, l, \dots)$, $(\dots, i, i+1, l, \dots)$, $(\dots, l, i, i+1, \dots)$ and $(\dots, l, i+1, i, \dots)$. The complexity of both algorithms remains the same as the number of evaluated partial sequences is the same.

4.4 An Iterated Local Search algorithm

The Iterated Local Search (ILS) algorithm has been widely applied to solve the flow shop scheduling problems, and has delivered high quality solutions. Examples include the studies of Stützle (1998), Marmion et al. (2011) and Lin et al. (2013). In this section, we propose an ILS algorithm for the ordered flow shop scheduling problem. The general idea behind operating the ILS algorithm is as follows. Upon generating an initial solution, a local search is applied to obtain an improved solution. Then, a perturbation is iteratively applied to the current solution, followed by the local search, which is applied to the perturbed solution. The process continues until the stopping criterion is met (Talbi, 2009).

Our proposed ILS algorithm incorporates a post-processing procedure in order to further improve the best obtained solution. For this reason, the post-processing is initiated when the main algorithm completes. The post-processing includes a local search algorithm, which utilizes a different neighborhood structure from that used in the main algorithm. The motivation behind the post-processing is that exploring the search space with a different neighborhood structure that does not utilize the pyramidal-shaped property may lead to exploring some yet unexplored solutions, and we may therefore obtain improved solutions. Recall that although we know that one of the pyramidal-shaped sequences must be optimal, a non-pyramidal sequence may also be optimal. Algorithm 3 summarizes our ILS algorithm, where Phase 1 represents the main algorithm and Phase 2 represents the post-processing procedure. Next, we discuss the components of the ILS algorithm.

Algorithm 3: The ILS algorithm for the ordered flow shop scheduling problem.

```
1  Input: The processing times of $n$ jobs on $m$ machines, $d_0, d_1, T_0, F, S, time\_limit$.
2  Output: A job sequence.
3  Phase 1:
4  Pair-Insert algorithm()
5  Local search algorithm()                          % Algorithm 4
6  $d = d_0$, $flag[F] = 0$, $T = T_0$;
7  while $elapsed\_time \leqslant time\_limit$ do
8      for $1 \leqslant l \leqslant d$ do
9          Perturbation()
10         Local search algorithm()                  % Algorithm 4
11     end
12     if "no new solution" found then
13         $flag[F] = flag[F] + 1$;
14         if $flag[F] \geqslant F$ then
15             $d = d_1$;
16         end
17     else
18         $flag[F] = 0$, $d = d_0$;
19     end
20     $T = \alpha \times T$;
21 end
22
23 Phase 2:
24 while $elapsed\_time \leqslant time\_limit$ do
25     Swap two randomly selected jobs in the current solution;
26     if superior solution found then
27         Update the best found solution;
28     end
29 end
30 return the best found solution
```

4.4.1 Generating initial solutions

Both random and greedy approaches have been used to generate initial solutions for the flow shop scheduling problem (Ponnambalam et al., 2001; Osman and Potts, 1989). Because the ILS algorithm includes applying random perturbations to the solution, we therefore utilize the Pair-Insert heuristic (Algorithm 2), which is a deterministic algorithm, to build an initial solution for the ILS algorithm. This implies that the initial solution will be a pyramidal-shaped sequence.

4.4.2 Perturbation

The main procedure in the ILS algorithm includes two operations of perturbation and local search. Through those operations, the ILS iteratively obtains improved solutions.

Each iteration of perturbation includes removing some jobs from their current positions in the sequence, and inserting them at some other positions. While jobs are randomly selected to be removed, their insertion positions are determined such that the pyramidal-shaped property of the solution is maintained. More precisely, if job j is removed from SPT sub-sequence, then it is inserted into LPT sub-sequence, and between two adjacent jobs l and k, where $\sum_{r=1}^{m} p_{r,l} \geqslant \sum_{r=1}^{m} p_{r,j} \geqslant \sum_{r=1}^{m} p_{r,k}$. Likewise, if job j is removed from LPT sub-sequence, it is then inserted into SPT sub-sequence, and that between two adjacent jobs l and k, where $\sum_{r=1}^{m} p_{r,l} \leqslant \sum_{r=1}^{m} p_{r,j} \leqslant \sum_{r=1}^{m} p_{r,k}$.

In order to control the diversification and intensification aspects of the ILS algorithm, the number of such removals and insertions (i.e. d) is adaptively changed during the course of algorithm's operation. For this reason, the number of removals and insertions is increased to d_1, from its default value of d_0, whenever certain number of consecutive iterations, denoted as F, are performed with no improvements in the solution. On the other hand, if an improved solution is obtained, the number of removals and insertions is reduced to the default value of d_0. The sequence generated by the perturbation operation is always accepted.

4.4.3 Local search

The local search method applied in the proposed ILS algorithm utilizes the "insert" operator to build new neighboring solutions, and operates as follows. A job is removed from its position in the sequence, and is inserted into another position such that the pyramidal-shaped property is maintained. This operation, which is performed for all jobs in the sequence, starts from the job in the first position in the sequence, and proceeds to the job in the last position. The process is continued until either a "new solution" is obtained, or all neighboring solutions are explored without obtaining such a new solution. A neighbor solution is defined as a "new solution", if either it has a better objective function value, or it has a worse objective function value, however, meeting the "acceptance criterion". Note that if a new solution is obtained, then it is accepted, i.e. the "first improvement strategy" is applied, and the process is restarted from the beginning of the sequence. This process is continued until S iterations are performed. Algorithm 4 summarizes the proposed local search.

We note that in the typical iterated local search algorithm the local search goes for a local optimum (Stützle and Ruiz, 2018). However, in our algorithm because the local search is applied for a certain number of iterations, that is the parameter S, delivering a local optimum is not guaranteed.

4.4.4 Acceptance and termination criteria

We implement an acceptance criterion that accepts both superior and inferior solutions. More precisely, an improved solution is always accepted. A worse solution can be accepted if the local search cannot deliver a new solution (see Section 4.4.3) and also the degradation amount is not greater than a certain threshold.

The threshold parameter is initialized as T_0 and decremented according to the geometric decrement rule of $T = \alpha \times T$, where $0 < \alpha < 1$. The algorithm continues until a certain time limit is elapsed.

Algorithm 4: Local search procedure of the ILS algorithm.

```
1  Input: Current solution.
2  Output: A new solution or the current solution.
3  $flag[P] = 1$, $flag[S] = 0$;
4  while $flag[P] = 1$ and $flag[S] \leqslant S$ do
5      $i = 1$, $flag[C] = 0$;
6      while $flag[C] = 0$ do
7          Remove the job in the $i$th position and insert it into another position in the sequence, maintaining
           the pyramidal property;
8          if a "new solution" (superior solution, or inferior one passing the acceptance criterion) obtained or
           all neighbor solutions are generated with "no new solution" then
9              $flag[C] = 1$;
10         end
11         $i = i + 1$;
12     end
13     if a "new solution" is found then
14         Update the current solution;
15         if the "new solution" is a superior solution then
16             Update the best found solution;
17         end
18     else if "no new solution" found then
19         $flag[P] = 0$;
20     end
21     $flag[S] = flag[S] + 1$;
22 end
23 return the solution
```

4.4.5 Post-processing

The Phase 2 of the ILS algorithm includes a post-processing procedure, which is applied to the best delivered solution of Phase 1. The Phase 2 applies a "swap" local search and aims to deliver new high quality solutions. For this reason, the neighboring solutions are built by using the swap operator, applied to randomly selected jobs, regardless of maintaining the pyramidal-shaped property. Discovering non-pyramidal sequences is a way to let the algorithm search for better solutions in unexplored spaces. Phase 2 is run for the same amount of time as Phase 1, i.e. they have the same time limit.

5 Computational experiments

In this section we report the computational results of the developed heuristic algorithms on three sets of randomly generated benchmark instances. First, we explain the instance generation procedure, and discuss tuning the parameters of the algorithms. Then, we run the algorithms on the generated instances and report the computational results.

The proposed algorithms were coded in the computational package MATLAB version 9.4 (MATLAB, 2018). All computational experiments were performed on a PC with Intel® Core^{TM} i5-7500 CPU clocked at 3.40GHz with 8GB of memory under Windows 10 operating system. To the best of our knowledge, there has been no solution method published specifically for the ordered flow shop scheduling problem. Therefore, for comparison purposes we solve the instances with the available solution method for the permutation flow shop problem. The top performing methods for the permutation flow shop problem include the Iterated Greedy Algorithms (IGA) of Ruiz and Stützle (2007) and of Fernandez-Viagas and Framinan (2014). Recently, Benavides and Ritt (2018) proposed a new IGA (denoted as IGA_{BR} throughout this study) that reports superior results for the permutation flow shop problem. The recently published IGA$_{BR}$ is very effective in solving both the permutation and the non-permutation flow shop scheduling problems. The IGA$_{BR}$ algorithm was coded in the programming language C++, and the code is available to the public. We use the same code. In addition, we use the IBM ILOG CPLEX version

12.8.0 (CPLEX, 2017) to solve Model I to optimality (where possible). Because the proposed ILS and the IGA$_{BR}$ start with an initial solution, we use the same initial solution to warm start the solver CPLEX. This ensures a fair comparison. We also force the solver CPLEX to utilize only one processor (thread), and to use a time limit as a stopping criterion. For the remaining parameters of the CPLEX we use the default values.

5.1 Instance generation and evaluation

To the best of our knowledge there are no published benchmark instances for the ordered flow shop scheduling problem. Therefore, we generate ordered instances, as benchmarks, for the problem, based on the available instances for the permutation flow shop problem, i.e. based on the benchmarks of Taillard (1993) and Vallada et al. (2015).

Taillard's instances were chosen from a large set of randomly generated instances, aiming to generate a set of the most difficult instances. The Taillard's instances range from 20 to 500 jobs, and 5 to 20 machines, and include 12 combinations of number of jobs and number of machines denoted as (n, m). Those 12 combinations include (20, 5), (20, 10), (20, 20), (50, 5), (50, 10), (50, 20), (100, 5), (100, 10), (100, 20), (200, 10), (200, 20), and (500, 20). Each combination consists of 10 instances. Therefore, a total number of 120 instances were produced. For more recent studies of Taillard's instances we refer the interested reader to Khatami and Zegordi (2017); Khorasanian and Moslehi (2017) and Liu et al. (2017).

Vallada et al. (2015) carried out extensive experiments and selected 240 "small" instances and 240 "large" instances, from a pool of 72,000 instances. The number of jobs in the Vallada et al.'s small instances is chosen from the set $\{10, 20, 30, 40, 50, 60\}$, and the number of machines is chosen from the set $\{5, 10, 15, 20\}$. Hence, there are $6 \times 4 = 24$ combinations of n and m. Each combination consists of 10 instances. This results in 240 instances. For the large instances, the number of jobs is selected from the set $\{100, 200, 300, 400, 500, 600, 700, 800\}$ and the number of machines is selected from the set $\{20, 40, 60\}$. Therefore, there exist $8 \times 3 = 24$ combinations of n and m, where each combination consists of 10 instances, and therefore, a total number of 240 large instances are produced. We refer the interested reader to Fernandez-Viagas et al. (2017) and Dubois-Lacoste et al. (2017) for two recent studies of Vallada et al.'s instances.

We note that Taillard's and Vallada et al.'s instances are for the permutation flow shop scheduling problem. In order to generate challenging benchmark instances for the ordered flow shop scheduling problem, the following three steps are performed on all the 120 Taillard's instances and on all the 480 Vallada et al.'s instances.

- Step 1: Sort, for each machine, the operation processing times of all the n jobs in a non-decreasing order;
- Step 2: Sort, for each job, the operation processing times on all the m machines in a non-decreasing order;
- Step 3: Permute the order of machines.

Steps 1 and 2 generate the instances in which the processing times of jobs are in the non-decreasing order, i.e. every job has its smallest processing time on the first machine, the second smallest processing time on the second machine and so on. Those instances can be easily solved by the SPT rule. Watson et al. (2002) also showed that the structured problems are relatively easy to solve. Thus, we perform Step 3 for permuting the order of machines, aiming at producing non-trivial instances. We let `T`, `S` and `L` denote our three sets of instances derived from the Taillard's and the Vallada et al.'s small and large instances.

According to Vallada et al. (2015), the difficulty of an instance can be measured as the percentage of the deviation from its lower bound (LB) solution, i.e. $\delta = 100 \times (UB - LB)/LB$, where the LB

Table 5: Number of instances that was solved to optimality with the Pair-Insert algorithm.

Problem size (*n*-*m*)	Benchmark `T` instances	Arbitrary instances	Problem size (*n*-*m*)	Benchmark `T` instances	Arbitrary instances
20-5	0	4	100-5	0	3
20-10	0	5	100-10	0	2
20-20	0	4	100-20	0	3
50-5	0	4	200-10	0	1
50-10	0	4	200-20	0	1
50-20	0	4	500-20	0	2

is obtained by utilizing the method of Ladhari and Haouari (2005) (eq. (11)), and UB is the objective function's upper bound calculated by using the Pair-Insert algorithm.

$$LB = \max_{1 \leqslant r < q \leqslant m} [\min_{j \in J} R_{r,j} + C_{max}^{r,q}(J) + \min_{j \in J} Q_{q,j}], \tag{11}$$

In eq. (11) $C_{max}^{r,q}(J)$ denotes the optimal makespan of a two-machine permutation flow shop problem with time lags, defined on the job set J and machine pair (r, q). In addition, $R_{r,j}$ and $Q_{q,j}$ are the summation of the processing times of job j on some subset of machines. For more details on the implementation of this lower bound see "LB5" in Ladhari and Haouari (2005). Note that this tight lower bound is calculated based on the bottleneck machine, which is suitable for our problem, particularly, because the number of bottleneck machines in the ordered flow shop scheduling problem is small.

For the 5-machine instances we can easily evaluate all 120 possible permutations of machines, and choose the one with the largest value of δ. For instances with 10 or more machines, however, we evaluate 1000 random permutations of machines instead, because evaluating all possible permutations is very expensive, if not impractical. Indeed, evaluating 1000 permutations ensures, to some extent, that the so generated instances are very difficult to solve. In summary, a total number of 520,800 instances were evaluated (93,600 instances of `T`, 187,200 instances of `S` and 240,000 instances of `L`), out of which 600 instances were produced as very difficult and challenging instances for the ordered flow shop scheduling problem (120 for `T`, 240 for `S`, and 240 for `L`). For the illustration purposes, we show the name of an instance in the format `B-n-m-x`, where `B` is the benchmark name that can be `T`, `S` or `L`, `n` and `m` denote the number of jobs and the number of machines, and `x` denotes the instance number, ranging from 1 to 10. For example, an instance named `T-20-5-1` is the instance in benchmark `T`, that includes 20 jobs and 5 machines and is assigned the index number 1.

Figure 1 highlights the necessity of producing instances with large value of δ. For comparison purposes, the box-plot includes a set of "arbitrary" instances, which were generated by randomly permuting machines based on instances of benchmark `T`. According to the box-plot, the δ metric may reveal how easily such arbitrary instances might be solved. The overall δ for the benchmark `T` is 3.33, while it is 0.64 for the arbitrary instances. In addition, our preliminary tests showed that the Pair-Insert algorithm obtains a proven optimal solution for 37 (out of 120) arbitrary instances, even for the largest size (a proven optimal solution is either reported by the solver CPLEX or is extracted when $UB = LB$). This means that at least about 31% of arbitrary instances were optimally solved with the Pair-Insert algorithm. In addition, we realize that the minimum of δ for every category of arbitrary instances is 0. Note that a value of 0 for δ implies that there was at least one instance solved to optimality by the Pair-Insert algorithm. The number of instances of both benchmark `T` and arbitrary instances (`R`) that were solved to optimality by the Pair-Insert algorithm is shown in Table 5. We also used statistical tests to validate the obtained results. Because the data do not follow a normal distribution, we applied the Wilcoxon signed-rank test for the paired data with a 95% confidence level. We observed a p-value of 0 for the test, confirming therefore the statistical significance of selecting instances with large values of δ. Overall, the results indicate the impact of generating instances with large deviations (associated with large values of δ) to build challenging benchmark instances for the ordered flow shop scheduling problem.

We note that all 600 benchmark instances, which we generate in this study, are available online in the Mendeley data repository at http://dx.doi.org/10.17632/cd2rv7hyyj.1. Future studies for the ordered

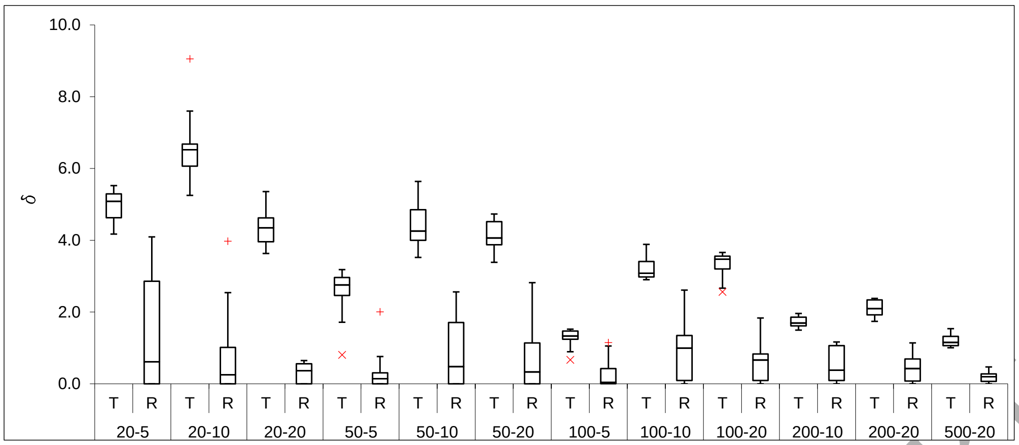

Figure 1: Comparison between the instances of benchmark `T` and the arbitrary instances (denoted as `R` in the figure).

flow shop scheduling problem could benefit from these benchmark sets in evaluating their methods.

5.2 Evaluation of the proposed heuristic algorithms

We first evaluate the proposed heuristic algorithms, i.e. the Pyramidal-NEH algorithm (denoted as NEH$_{Pyr}$) and the Pair-Insert algorithm (denoted as PI), by solving the three sets of benchmark instances and comparing the outcomes with those obtained by the NEH algorithm with Taillard acceleration (denoted as NEH$_{Tai}$), and also with the NEH algorithm with Taillard acceleration and tie-breaking rule of Fernandez-Viagas and Framinan (2014) (denoted as NEH$_{FF}$).

We use three criteria of "number of best obtained solutions (N_{Best})", "average relative percentage deviation ($ARPD$)", and "average relative percentage time ($ARPT$)" to evaluate the performance of those four heuristic algorithms.

The quality of the results is shown by the relative percentage deviation (RPD), which is calculated as $\frac{z-z^*}{z^*} \times 100$, where z is the objective function value, i.e. the makespan delivered by an algorithm, and z^* is the best known objective function value for the instance. The metric $ARPD$ is the average of RPDs over groups of instances. The metric RPT denotes the relative percentage computation time of an algorithm, and is calculated as $\frac{CPU-ACT}{ACT} \times 100$, where CPU is the computation time of the algorithm for an instance, and ACT is the average computation time for all algorithms on the same instance. The $ARPT'$ is the average of RPTs over groups of instances, and $ARPT$ is then calculated as $ARPT = ARPT' + 1$. For more details on the metrics $ARPD$ and $ARPT$ we refer the interested reader to Fernandez-Viagas et al. (2017).

Table 6 presents the values of the three metrics N_{Best}, $ARPD$ and $ARPT$ for the four heuristics of NEH$_{Pyr}$, PI, NEH$_{Tai}$, and NEH$_{FF}$ over the three sets of benchmark instances. The highlighted numbers denote the outperforming values. The results show that the PI is the best algorithm in terms of N_{Best}, and the NEH$_{Pyr}$ is the second best. The PI generates the best solution for almost 73% of the instances, and in the order of over 4 times more than the NEH$_{Tai}$ algorithm. The quality of the PI algorithm is further confirmed based on the metric $ARPD$, where it delivers solutions with an $ARPD$ value of 0.81, where the $ARPD$ of NEH$_{FF}$ (the second best algorithm) is 1.08. The quality of solutions of NEH$_{Pyr}$ is similar to that of NEH$_{Tai}$, and slightly worse than that of NEH$_{FF}$. Finally, the NEH$_{Pyr}$ is the fastest and the PI is the slowest algorithms based on the metric $ARPT$.

Table 6: Summary of metrics N_{Best}, $ARPD$ and $ARPT$ for the four heuristic algorithms on the three benchmark sets.

Metric	Benchmark	NEH$_{Tai}$	NEH$_{FF}$	NEH$_{Pyr}$	PI
N_{Best}	T	20	16	20	**89**
	S	42	39	42	**211**
	L	44	50	55	**137**
	Sum	106	105	117	**437**
$ARPD$	T	1.30	1.22	1.30	**0.91**
	S	1.56	1.51	1.56	**1.10**
	L	0.61	0.52	0.61	**0.42**
	Average	1.16	1.08	1.16	**0.81**
$ARPT$	T	0.54	0.76	**0.43**	2.28
	S	0.49	0.66	**0.38**	2.47
	L	0.86	1.10	**0.65**	1.39
	Average	0.63	0.84	**0.49**	2.05

The superiority of the PI algorithm can be further investigated in Figure 2, where the RPDs of the heuristic algorithms are compared based on the three benchmark sets. In addition, the Wilcoxon signed-rank test for the paired data of the RPD values of the heuristic algorithms with a 95% confidence level, which is detailed in Table 7, confirms the quality of the PI algorithm.

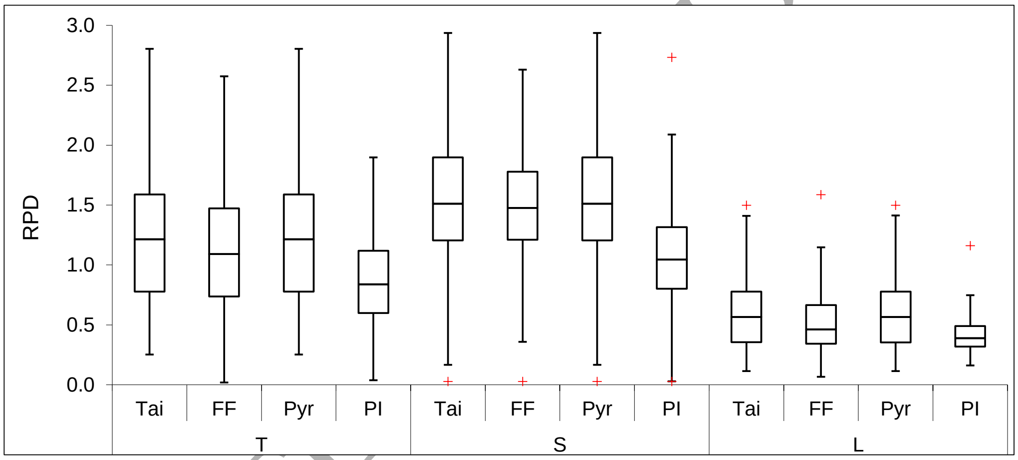

Figure 2: RPDs of the heuristic algorithms on the three benchmark sets T, S and L.

Table 7: Wilcoxon signed-rank test for RPDs of the heuristic algorithms, based on the three benchmark sets.

Benchmark	Comparison	Wilcoxon signed-rank test	
		Test statistic	p-value
T	PI vs. NEH$_{Tai}$	455	0.000
	PI vs. NEH$_{FF}$	466	0.000
	PI vs. NEH$_{Pyr}$	457	0.000
S	PI vs. NEH$_{Tai}$	970	0.000
	PI vs. NEH$_{FF}$	726	0.000
	PI vs. NEH$_{Pyr}$	970	0.000
L	PI vs. NEH$_{Tai}$	2621	0.000
	PI vs. NEH$_{FF}$	4100	0.000
	PI vs. NEH$_{Pyr}$	2667	0.000

We should note that although the PI heuristic is not as fast as the other three heuristics, the computation time is not a concern because the largest instances (i.e. `L-800-60-x`) is solved within a second by any of the four heuristic algorithms. To conclude, the PI can be considered as the best performing heuristic for generating initial solutions for the ordered flow shop scheduling problem, and therefore, we utilize the PI heuristic to generate initial solutions for the ILS algorithm.

5.3 Parameter tuning for the ILS algorithm

Before performing the experiments, the parameters of the proposed ILS algorithm must carefully be tuned. Since there are only five parameters in the proposed ILS algorithm, a full factorial design of experiments (Montgomery, 2017) is applied to tune the values of the parameters. Those parameters include T_0 (the initial threshold value), α (the threshold decrement parameter), the upper and lower levels of d (the number of modifications in the perturbation), F (the number of iterations with no improvement), and S (the maximum number of times that the local search is performed at each iteration). To set the initial threshold value for each instance, a coefficient of its lower bound LB is used. Four candidate values of 0.05, 0.1, 0.15 and 0.2 are considered for the coefficient. For the threshold decrement parameter, we consider $(1/T_0)^{(1/t)}$, where t is selected from $\{400, 600, 800, 1000\}$. Also, recall that the number of modifications in the perturbation is changed adaptively as the algorithm proceeds. It starts with a default value of d_0, and if F consecutive iterations are performed with no improvement, it is increased to d_1. We investigate two scenarios for the values of the pair $(d_0,\, d_1)$, that are: $(\lfloor \log(n/2) \rceil,\, \lfloor \log 10n \rceil)$ and $(\lfloor 2\log n \rceil,\, \lfloor 3\log n \rceil)$. We also consider four candidate values of 5, 10, 15 and 20 for F. For the intensification parameter S, that is the maximum number of times that the local search is performed at each iteration, we consider four values of 6, 8, 10 and 12. Those candidate values lead to a total number of $4 \times 4 \times 2 \times 4 \times 4 = 512$ combinations. To tune the algorithm, 35 instances with different sizes are solved with those 512 parameter combinations. Hence, a total number of 17,920 problems are solved and RPD of each problem is used to represent its quality. Based on the obtained results of the means of the analysis of variance (ANOVA) technique, d and S statistically play a significant role in the performance of the algorithm. The mean plots and least significant difference (LSD) intervals (at the 95% confidence level) of Figure 3 are used to set the value of the parameters. The optimal value of those parameters are presented in Table 8.

Table 8: Value of the parameters used in the ILS algorithm.

Parameter	Definition	Value
T_0	Initial threshold value	$0.1 \times LB$
α	Threshold decrement factor	$(1/T_0)^{(1/600)}$
(d_0, d_1)	Default and increased number of removals and insertions	$(\lfloor \log(n/2) \rceil, \lfloor \log 10n \rceil)$
F	Number of iterations with no improvement	15
S	The maximum number of times the local search is performed at each iteration	10

5.4 Evaluation of the ILS algorithm

We test the proposed ILS algorithm on the three sets of benchmark instances of Section 5.1. For comparison purposes, we solve those instances with the IGA$_{BR}$, and also with the solver CPLEX (of solving Model I) with a warm start. The solution of the PI heuristic is utilized to warm start the CPLEX.

We use the three criteria of N_{Best}, $ARPD$ and $ARPT$ to evaluate the performance of ILS and IGA$_{BR}$ algorithms, and the solver CPLEX. We use the same time limit (milliseconds) for ILS and IGA$_{BR}$ algorithms, which is calculated as $\tau \times n \times m$, where τ is set to 15 (hence, each phase of ILS algorithm is set to run for $\frac{\tau \times n \times m}{2}$ milliseconds). We solve each instance for three times by the ILS and IGA$_{BR}$ algorithms, and report the average of the three runs. A computation time limit of 3600 seconds (60 minutes) is applied to the CPLEX, and the CPLEX is set to utilize one processor. Other than those, we use the default values for the remaining parameters of the CPLEX. We note that an optimal solution to Model I is not guaranteed by setting a time limit for the CPLEX.

Table 9 presents the results of the three methods on the instances of benchmark `T`, where the best values across the methods are highlighted. It is easily realized that the ILS algorithm is performing very well on the larger instances, where it is the superior algorithm for the 80 large instances (`T-50-10-x` to `T-500-20-x`).

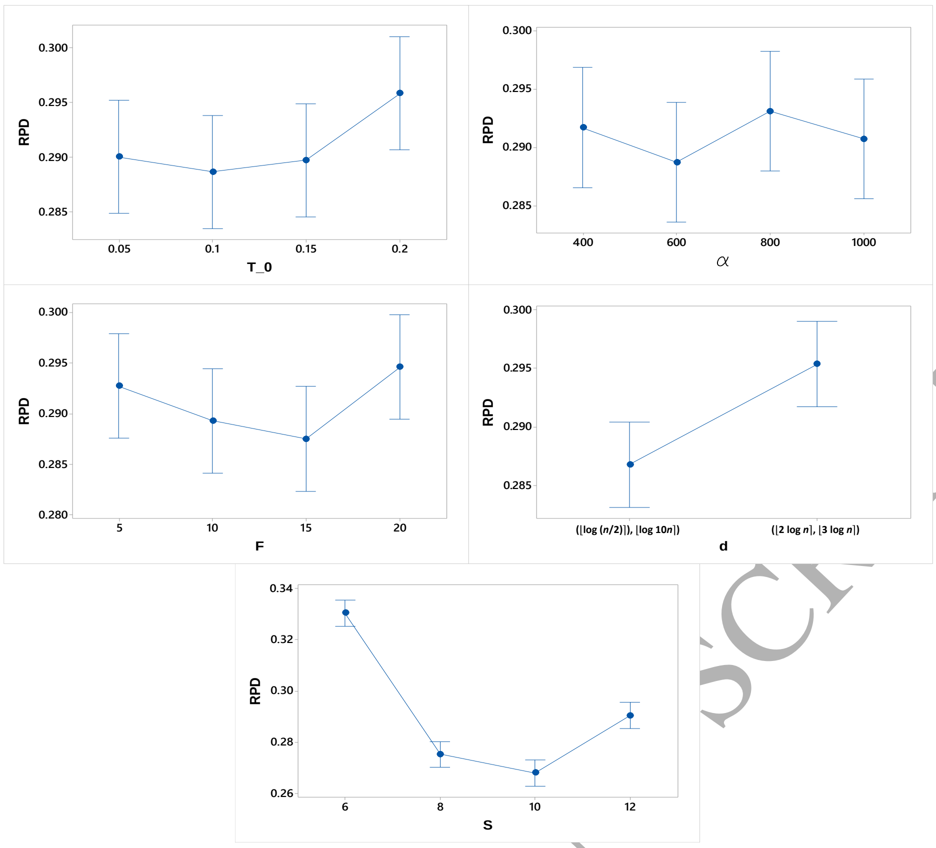

Figure 3: Mean plots and LSD intervals for ILS parameters.

Table 9: Detailed comparison of the methods on the benchmark `T`.

Problem size	ILS			IGA$_{BR}$			CPLEX		
(n-m)	N_{Best}	$ARPD$	$ARPT$	N_{Best}	$ARPD$	$ARPT$	N_{Best}	$ARPD$	$ARPT$
20-5	5	0.06	**0.86**	**10**	**0.00**	**0.86**	**10**	**0.00**	1.28
20-10	3	0.03	**0.64**	**10**	**0.00**	**0.64**	**10**	**0.00**	1.72
20-20	5	0.01	**0.65**	**10**	**0.00**	**0.65**	**10**	**0.00**	1.71
50-5	4	**0.03**	**0.00**	3	0.04	**0.00**	**5**	0.05	2.99
50-10	**8**	**0.02**	**0.01**	0	0.09	**0.01**	3	0.10	2.99
50-20	**9**	**0.02**	**0.01**	0	0.09	**0.01**	1	0.06	2.98
100-5	**9**	**0.01**	**0.01**	1	0.08	**0.01**	0	0.18	2.99
100-10	**10**	**0.01**	**0.01**	0	0.17	**0.01**	0	0.32	2.98
100-20	**10**	**0.01**	**0.02**	0	0.16	**0.02**	0	0.32	2.95
200-10	**10**	**0.01**	**0.02**	0	0.13	**0.02**	0	0.54	2.95
200-20	**10**	**0.01**	**0.05**	0	0.16	**0.05**	0	0.52	2.90
500-20	**10**	**0.00**	**0.12**	0	0.10	**0.12**	0	0.36	2.77
Total	**93**	**0.02**	**0.20**	34	0.08	**0.20**	39	0.20	2.60

Table 10 presents the results of the methods on the instances of benchmark `S`. The best values across the methods are highlighted. It is clear that the ILS algorithm is the superior method for the larger instances, i.e. for the last 80 ones (`S-50-5-x` to `S-60-20-x`). The average computation time for the solver CPLEX is more than 1800 seconds, while this is less than 7 seconds for ILS and IGA$_{BR}$ algorithms.

The results of the three methods on the instances of benchmark `L`, which are reported in Table 11, show the good performance of the ILS algorithm. We note that the ILS algorithm outperforms both IGA$_{BR}$ and CPLEX, in terms of solution time and quality.

Table 10: Detailed comparison of the methods on the benchmark `S`.

Problem size	ILS			IGA$_{BR}$			CPLEX		
(n-m)	N_{Best}	$ARPD$	$ARPT$	N_{Best}	$ARPD$	$ARPT$	N_{Best}	$ARPD$	$ARPT$
10-5	**10**	**0.00**	1.42	**10**	**0.00**	1.42	**10**	**0.00**	**0.15**
10-10	**10**	**0.00**	1.43	**10**	**0.00**	1.43	**10**	**0.00**	**0.14**
10-15	**10**	**0.00**	1.44	**10**	**0.00**	1.44	**10**	**0.00**	**0.13**
10-20	**10**	**0.00**	1.45	**10**	**0.00**	1.45	**10**	**0.00**	**0.11**
20-5	4	0.04	**0.57**	**10**	**0.00**	**0.57**	**10**	**0.00**	1.87
20-10	4	0.04	**0.64**	**10**	**0.00**	**0.64**	**10**	**0.00**	1.72
20-15	5	0.02	**0.60**	**10**	**0.00**	**0.60**	**10**	**0.00**	1.81
20-20	4	0.02	**0.74**	**10**	**0.00**	**0.74**	**10**	**0.00**	1.53
30-5	3	0.06	**0.02**	**8**	0.01	**0.02**	**8**	**0.01**	2.97
30-10	2	0.08	**0.08**	5	0.01	**0.08**	**9**	**0.00**	2.84
30-15	2	0.03	**0.06**	5	0.01	**0.06**	**9**	**0.00**	2.88
30-20	4	0.02	**0.16**	4	0.01	**0.16**	**9**	**0.01**	2.68
40-5	2	0.05	**0.00**	3	0.03	**0.00**	**8**	**0.01**	2.99
40-10	2	0.05	**0.03**	1	0.05	**0.03**	**9**	**0.02**	2.95
40-15	4	0.02	**0.02**	0	0.04	**0.02**	**8**	**0.01**	2.97
40-20	2	0.01	**0.01**	0	0.05	**0.01**	**9**	**0.01**	2.97
50-5	**5**	**0.02**	**0.00**	2	0.05	**0.00**	4	0.03	2.99
50-10	**7**	**0.02**	**0.01**	0	0.10	**0.01**	4	0.08	2.99
50-15	4	**0.01**	**0.01**	0	0.09	**0.01**	**6**	0.02	2.98
50-20	3	**0.02**	**0.01**	1	0.07	**0.01**	**7**	0.03	2.97
60-5	**9**	**0.01**	**0.05**	3	0.04	**0.05**	2	0.10	2.89
60-10	4	**0.02**	**0.01**	0	0.12	**0.01**	**7**	0.03	2.99
60-15	5	**0.02**	**0.01**	0	0.11	**0.01**	**6**	0.06	2.98
60-20	**9**	**0.02**	**0.01**	0	0.10	**0.01**	1	0.08	2.97
Total	124	0.02	**0.37**	112	0.03	**0.37**	**186**	**0.02**	2.27

Table 11: Detailed comparison of the methods on the benchmark `L`.

Problem size	ILS			IGA$_{BR}$			CPLEX		
(n-m)	N_{Best}	$ARPD$	$ARPT$	N_{Best}	$ARPD$	$ARPT$	N_{Best}	$ARPD$	$ARPT$
100-20	**10**	**0.01**	**0.02**	0	0.16	**0.02**	0	0.35	2.95
100-40	**10**	**0.01**	**0.05**	0	0.13	**0.05**	0	0.47	2.90
100-60	**10**	**0.00**	**0.07**	0	0.11	**0.07**	0	0.42	2.86
200-20	**10**	**0.00**	**0.05**	0	0.16	**0.05**	0	0.59	2.90
200-40	**10**	**0.01**	**0.09**	0	0.14	**0.09**	0	0.58	2.81
200-60	**10**	**0.00**	**0.14**	0	0.12	**0.14**	0	0.47	2.73
300-20	**10**	**0.00**	**0.07**	0	0.13	**0.07**	0	0.57	2.86
300-40	**10**	**0.01**	**0.14**	0	0.12	**0.14**	0	0.46	2.73
300-60	**10**	**0.01**	**0.20**	0	0.10	**0.20**	0	0.39	2.61
400-20	**10**	**0.00**	**0.09**	0	0.11	**0.09**	0	0.43	2.81
400-40	**10**	**0.01**	**0.18**	0	0.09	**0.18**	0	0.37	2.65
400-60	**10**	**0.00**	**0.25**	0	0.09	**0.25**	0	0.37	2.50
500-20	**10**	**0.01**	**0.12**	0	0.09	**0.12**	0	0.45	2.77
500-40	**10**	**0.00**	**0.21**	0	0.10	**0.21**	0	0.32	2.57
500-60	**10**	**0.01**	**0.30**	0	0.08	**0.30**	0	0.34	2.40
600-20	**10**	**0.00**	**0.14**	0	0.07	**0.14**	0	0.35	2.73
600-40	**10**	**0.01**	**0.25**	0	0.11	**0.25**	0	0.39	2.50
600-60	**10**	**0.01**	**0.35**	0	0.10	**0.35**	0	0.35	2.31
700-20	**8**	**0.00**	**0.16**	2	0.06	**0.16**	0	0.33	2.69
700-40	**10**	**0.00**	**0.28**	0	0.10	**0.28**	0	0.29	2.43
700-60	**10**	**0.01**	**0.39**	0	0.08	**0.39**	0	0.36	2.22
800-20	**7**	**0.01**	**0.18**	3	0.05	**0.18**	0	0.27	2.65
800-40	**9**	**0.01**	**0.32**	1	0.06	**0.32**	0	0.29	2.37
800-60	**10**	**0.00**	**0.43**	0	0.08	**0.43**	0	0.31	2.14
Total	**234**	**0.01**	**0.19**	6	0.10	**0.19**	0	0.40	2.63

Table 12 provides a summary of the computational results of the three methods. Based on the metric N_{Best}, the ILS is the best performing algorithm among all, because it delivers the best solution for more than 75% of the instances. Also, the solutions' quality of the ILS algorithm is very good according to the metric $ARPD$. More precisely, the ILS algorithm generates superior solutions than the IGA$_{BR}$ algorithm, and that in the order of almost 7 times. The quality of the ILS algorithm can be further observed in Figure 4, where the *RPD*s of the methods are compared across the three benchmark sets. Since ILS and IGA$_{BR}$ algorithms use the same time limit, the *ARPT* measure is similar for both, and far better than that of the solver CPLEX. The Wilcoxon signed-rank test for paired data with 95% confidence level is carried out on the *RPD*s of the three algorithms (see Table 13), and confirms the quality of the ILS algorithm, in particular, for solving larger instances.

Table 12: Metrics N_{Best}, *ARPD* and *ARPT* for ILS, IGA$_{BR}$, and CPLEX on the three benchmark sets.

Metric	Benchmark	ILS	IGA$_{BR}$	CPLEX
N_{Best}	T	**93**	34	39
	S	124	112	**186**
	L	**234**	6	0
	Sum	**451**	152	225
ARPD	T	**0.02**	0.08	0.20
	S	0.02	0.03	**0.02**
	L	**0.01**	0.10	0.40
	Average	**0.02**	0.07	0.21
ARPT	T	**0.20**	**0.20**	2.60
	S	**0.37**	**0.37**	2.27
	L	**0.19**	**0.19**	2.63
	Average	**0.25**	**0.25**	2.50

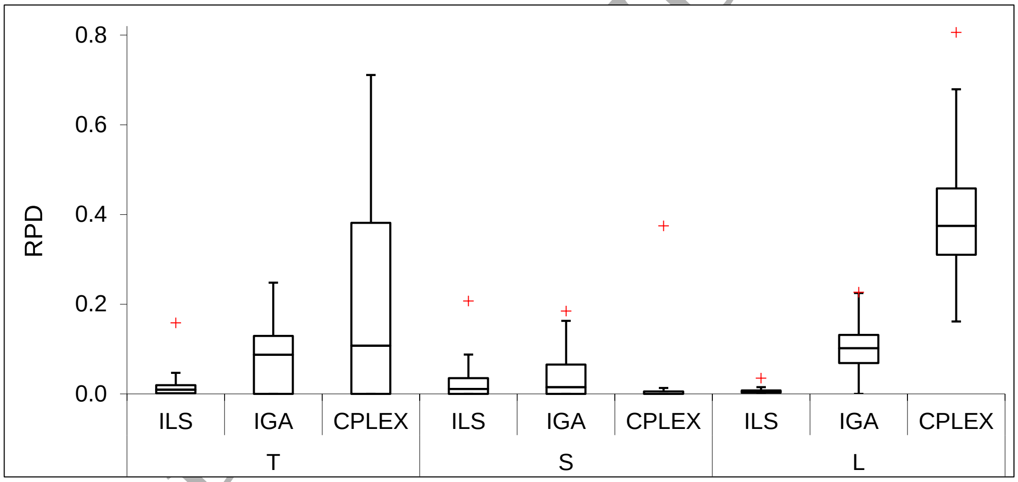

Figure 4: *RPD*s of the ILS, IGA$_{BR}$ and the solver CPLEX on the three benchmark sets T, S and L.

Table 13: Wilcoxon signed-rank test for *RPD*s of ILS, IGA$_{BR}$ and CPLEX, based on the three benchmark sets.

Benchmark	Comparison	Wilcoxon signed-rank test	
		Test statistic	*p*-value
T	ILS vs. IGA$_{BR}$	606	0.000
	ILS vs. CPLEX	518	0.000
S	CPLEX vs. ILS	4993	0.005
	ILS vs. IGA$_{BR}$	4485	0.001
L	ILS vs. IGA$_{BR}$	67	0.000
	ILS vs. CPLEX	0	0.000

5.5 Effectiveness of Phase 2 of the ILS algorithm

The Phase 2 of the ILS algorithm was incorporated to further improve the quality of solutions. To evaluate its effectiveness, the total improvement gained by the ILS algorithm is analyzed after the completion of each phase. On average, 63.73%, i.e. nearly two thirds of the improvements are gained by Phase 1 of the algorithm. However, according to Table 14 we can observe that as the size of the instances increases, the contribution of Phase 2 in delivering high quality solutions has significantly risen. In particular, the improvement due to Phase 2 is greater than 50% for large instances with 300 jobs and more, and also with 40 and 60 machines.

Table 14: Comparison of the performance between Phases 1 and 2 of the ILS algorithm.

Number of jobs	Total ILS improvement (%) gained in:	
	Phase 1	Phase 2
10	100.00	0.00
20	92.52	7.48
30	79.13	20.87
40	73.69	26.31
50	70.69	29.31
60	66.60	33.40
100	57.40	42.60
200	52.02	47.98
300	45.64	54.36
400	45.68	54.32
500	43.59	56.41
600	39.50	60.50
700	41.33	58.67
800	36.18	63.82
Number of machines		
5	77.45	22.55
10	74.87	25.13
15	80.58	19.42
20	60.45	39.55
40	44.82	55.18
60	48.46	51.54
Average:	**63.73**	**36.27**

We further investigate the impact of Phase 1 and Phase 2 on the ILS algorithm. More precisely, we solve instances of benchmarks `T`, `S` and `L` with two settings: (1) allocating the total running time to Phase 1 only, and (2) allocating the total running time to Phase 2 only. In both settings, we utilize the Pair-Insert algorithm to generate the initial solution, as we used in the proposed ILS algorithm. Table 1 presents the results and shows that the solutions obtained by setting 1 are of very poor quality in comparison to the solutions produced by the ILS algorithm. In addition, we observe that setting 2 is outperformed by the ILS algorithm for instances of benchmarks `T` and `S`. Setting 2 has led to slightly better solutions for the instances of benchmark `L`, however, in overall the ILS algorithm generates superior solutions. In total, results of settings 1 and 2 confirm our previous observation regarding the effectiveness of Phase 2 for large instances.

Table 15: Comparing the performance of of ILS, and settings 1 and 2.

Benchmark	*ARPD*		
	ILS	Setting 1	Setting 2
`T`	**0.02**	0.25	0.02
`S`	**0.03**	0.21	0.05
`L`	0.01	0.23	**0.00**
Average	**0.02**	0.23	0.02

6 Conclusion

In this study we have investigated the makespan minimization for solving the ordered flow shop scheduling problem, which is a variant of the flow shop problem. We have utilized the pyramidal-shaped property of the problem and designed two efficient heuristics, and an Iterated Local Search (ILS) algorithm. Because there have not been any benchmark instances published for the ordered flow shop scheduling problem, we have designed a procedure that generates challenging random instances for the ordered flow shop scheduling problem. We produced a total number of 600 instances in three sets, ranging from 10 to 800 jobs and 5 to 60 machines. We solved the instances by the proposed heuristics and the ILS algorithm, and compared the outcomes with those by the state-of-the-art IGA$_{BR}$ and the solver CPLEX. We showed the efficiency and effectiveness of the proposed algorithms, in particular, the ILS in solving the ordered flow shop scheduling problem. In summary, we obtained the best solutions for 75% of instances, and in a short amount of time. In addition, $ARPD$ of the ILS is shown to be 0.01%, which is significantly less than that of the IGA$_{BR}$ and CPLEX that are 0.07% and 0.20%, respectively.

One direction of the future research could be applying the proposed ILS algorithm to scheduling problems with the V-shaped property, such as the single machine problem with linear deteriorating jobs. In addition, the problem can be investigated with due-date related criteria.

Acknowledgments

The authors would like to thank the four anonymous referees for their valuable suggestions and comments. Mostafa Khatami is the recipient of UTS International Research Scholarship (IRS) and UTS President's Scholarship (UTSP). Amir Salehipour is the recipient of an Australian Research Council Discovery Early Career Researcher Award (DECRA) funded by the Australian Government.

References

Al-Turki, U., Fedjki, C., and Andijani, A. (2001). "Tabu search for a class of single-machine scheduling problems". *Computers & Operations Research* 28(12), 1223 –1230.

Benavides, Alexander J. and Ritt, Marcus (2018). "Fast heuristics for minimizing the makespan in non-permutation flow shops". *Computers & Operations Research* 100, 230 –243.

Choi, Byung-Cheon and Chung, Jibok (2011). "Two-machine flow shop scheduling problem with an outsourcing option". *European Journal of Operational Research* 213(1), 66 –72.

Choi, Byung-Cheon and Park, Myoung-Ju (2016). "An ordered flow shop with two agents". *Asia-Pacific Journal of Operational Research* 33(05), 1650037.

Chung, Dae-Young and Choi, Byung-Cheon (2013). "Outsourcing and scheduling for two-machine ordered flow shop scheduling problems". *European Journal of Operational Research* 226(1), 46 –52.

CPLEX, IBM ILOG (2017). *version 12.8.0.* IBM Corp.: Armonk, New York, U.S.

Dong, Xingye, Huang, Houkuan, and Chen, Ping (2008). "An improved NEH-based heuristic for the permutation flowshop problem". *Computers & Operations Research* 35(12). Part Special Issue: Telecommunications Network Engineering, 3962 –3968.

Dubois-Lacoste, Jrmie, Pagnozzi, Federico, and Stützle, Thomas (2017). "An iterated greedy algorithm with optimization of partial solutions for the makespan permutation flowshop problem". *Computers & Operations Research* 81, 160 –166.

Fernandez-Viagas, Victor and Framinan, Jose M. (2014). "On insertion tie-breaking rules in heuristics for the permutation flowshop scheduling problem". *Computers & Operations Research* 45, 60 –67.

Fernandez-Viagas, Victor, Ruiz, R., and Framinan, Jose M. (2017). "A new vision of approximate methods for the permutation flowshop to minimise makespan: State-of-the-art and computational evaluation". *European Journal of Operational Research* 257(3), 707 –721.

Gordon, Valery, Proth, Jean-Marie, and Chu, Chengbin (2002). "A survey of the state-of-the-art of common due date assignment and scheduling research". *European Journal of Operational Research* 139(1), 1 –25.

Hou, Sixiang and Hoogeveen, Han (2003). "The three-machine proportionate flow shop problem with unequal machine speeds". *Operations Research Letters* 31(3), 225 –231.

Ilić, Aleksandar (2015). "On the variable common due date, minimal tardy jobs bicriteria two-machine flow shop problem with ordered machines". *Theoretical Computer Science* 582, 70 –73.

Kalczynski, Pawel Jan and Kamburowski, Jerzy (2007). "On the NEH heuristic for minimizing the makespan in permutation flow shops". *Omega* 35(1), 53 –60.

Kalczynski, Pawel Jan and Kamburowski, Jerzy (2008). “An improved NEH heuristic to minimize makespan in permutation flow shops”. *Computers & Operations Research* 35(9). Part Special Issue: Bio-inspired Methods in Combinatorial Optimization, 3001 –3008.

Khatami, Mostafa and Zegordi, Seyed Hessameddin (2017). “Coordinative production and maintenance scheduling problem with flexible maintenance time intervals”. *Journal of Intelligent Manufacturing* 28(4), 857–867.

Khorasanian, Danial and Moslehi, Ghasem (2017). “Two-machine flow shop scheduling problem with blocking, multi-task flexibility of the first machine, and preemption”. *Computers & Operations Research* 79, 94 –108.

Kim, Byung-Gyoo, Choi, Byung-Cheon, and Park, Myoung-Ju (2017). “Two-machine and two-agent flow shop with special processing times structures”. *Asia-Pacific Journal of Operational Research* 34(04), 1750017.

Koulamas, Christos and Panwalkar, S. S. (2018). “New index priority rules for no-wait flow shops”. *Computers & Industrial Engineering* 115, 647 –652.

Koulamas, Christos and Panwalkar, S.S. (2015). “Job selection in two-stage shops with ordered machines”. *Computers & Industrial Engineering* 88, 350 –353.

Kouvelis, P. and Daniels, R. L. (2000). “Robust scheduling of a two machine flow shop with uncertain processing times”. *IIE Transactions* 32, 421 –432.

Ladhari, Talel and Haouari, Mohamed (2005). “A computational study of the permutation flow shop problem based on a tight lower bound”. *Computers & Operations Research* 32(7), 1831 –1847.

Lee, Kangbok, Zheng, Feifeng, and Pinedo, Michael L. (2019). “Online scheduling of ordered flow shops”. *European Journal of Operational Research* 272(1), 50 –60.

Li, Yongqiang, Li, Gang, Sun, Linyan, and Xu, Zhiyong (2009). “Single machine scheduling of deteriorating jobs to minimize total absolute differences in completion times”. *International Journal of Production Economics* 118(2), 424 –429.

Lin, Bertrand M. T., Lin, Y. Y., and Fang, K. T. (2013). “Two-machine flow shop scheduling of polyurethane foam production”. *International Journal of Production Economics* 141(1), 286 –294.

Liu, Weibo, Jin, Yan, and Price, Mark (2017). “A new improved NEH heuristic for permutation flowshop scheduling problems”. *International Journal of Production Economics* 193, 21 –30.

Marmion, Marie-Eléonore, Dhaenens, Clarisse, Jourdan, Laetitia, Liefooghe, Arnaud, and Verel, Sébastien (2011). “NILS: a neutrality-based iterated local search and its application to flowshop scheduling”. *European Conference on Evolutionary Computation in Combinatorial Optimization.* Ed. by Peter Merz. Springer Berlin Heidelberg: Berlin, Heidelberg, 191–202.

MATLAB (2018). *version 9.4.0 (R2018a).* The MathWorks Inc.: Natick, Massachusetts, U.S.

Mittenthal, John, Raghavachari, Madabhushi, and Rana, Arif I. (1993). “A hybrid simulated annealing approach for single machine scheduling problems with non-regular penalty functions”. *Computers & Operations Research* 20(2), 103 –111.

Montgomery, Douglas C (2017). *Design and analysis of experiments.* John wiley & Sons.

Mosheiov, G. (1991). “V-shaped policies for scheduling jobs”. *Operations Research* 39, 979 –991.

Nawaz, Muhammad, Enscore, E Emory, and Ham, Inyong (1983). “A heuristic algorithm for the m-machine, n-job flow-shop sequencing problem”. *Omega* 11(1), 91 –95.

Osman, I. H. and Potts, C. N. (1989). “Simulated annealing for permutation flow-shop scheduling”. *Omega* 17(6), 551 –557.

Panwalkar, S. S. and Khan, A. W. (1977). “A convex property of an ordered flow shop sequencing problem”. *Naval Research Logistics Quarterly* 24(1), 159–162.

Panwalkar, S. S. and Woollam, C. R. (1979). “Flow shop scheduling problems with no in-process waiting: A special case”. *Journal of the Operational Research Society* 30(7), 661–664.

Panwalkar, S. S. and Woollam, C. R. (1980). “Ordered flow shop problems with no in-process waiting: Further results”. *Journal of the Operational Research Society* 31(11), 1039–1043.

Panwalkar, S. S., Dudek, R. A., and Smith, M. L. (1973). "Sequencing research and the industrial scheduling problem". *Symposium on the theory of scheduling and its applications.* Ed. by Salah E. Elmaghraby. Springer Berlin Heidelberg: Berlin, Heidelberg, 29–38.

Panwalkar, S.S. and Koulamas, Christos (2012). "An $O(n^2)$ algorithm for the variable common due date, minimal tardy jobs bicriteria two-machine flow shop problem with ordered machines". *European Journal of Operational Research* 221(1), 7 –13.

Panwalkar, S.S. and Koulamas, Christos (2013). "The three-stage ordered flow shop problem with flexible stage ordering". *Computers & Industrial Engineering* 64(4), 1093 –1095.

Panwalkar, S.S. and Koulamas, Christos (2017). "On the dominance of permutation schedules for some ordered and proportionate flow shop problems". *Computers & Industrial Engineering* 107, 105 –108.

Panwalkar, S.S., Smith, Milton L., and Koulamas, Christos (2013). "Review of the ordered and proportionate flow shop scheduling research". *Naval Research Logistics (NRL)* 60(1), 46–55.

Park, Myoung-Ju and Choi, Byung-Cheon (2015). "Min-max regret version of an m-machine ordered flow shop with uncertain processing times." *Management Science & Financial Engineering* 21(1).

Ponnambalam, S. G., Aravindan, P., and Chandrasekaran, S. (2001). "Constructive and improvement flow shop scheduling heuristics: An extensive evaluation". *Production Planning & Control* 12(4), 335 –344.

Röck, Hans (1984). "The three-machine no-wait flow shop is NP-complete". *Journal of the Association Computing Machinery* 31(2), 336–345.

Ruiz, R. and Stützle, T. (2007). "Robust scheduling of a two machine flow shop with uncertain processing times". *European Journal of Operational Research* 177, 2033 –2049.

Ruiz, Rubn and Maroto, Concepcin (2005). "A comprehensive review and evaluation of permutation flowshop heuristics". *European Journal of Operational Research* 165(2). Project Management and Scheduling, 479 –494.

Şen, Alper, Topaloğlu, Engin, and Benli, Ömer S. (1998). "Optimal streaming of a single job in a two-stage flow shop". *European Journal of Operational Research* 110(1), 42 –62.

Smith, M. L., Panwalkar, S. S., and Dudek, R. A. (1975). "Flowshop sequencing problem with ordered processing time matrices". *Management Science* 21(5), 544–549.

Smith, M. L., Panwalkar, S. S., and Dudek, R. A. (1976). "Flowshop sequencing problem with ordered processing time matrices: A general case". *Naval Research Logistics Quarterly* 23(3), 481–486.

Smith, Milton L (1968). "A critical analysis of flow-shop sequencing". PhD thesis. Texas Tech University.

Stützle, Thomas (1998). "Applying iterated local search to the permutation flow shop problem". *FG Intellektik, TU Darmstadt, Darmstadt, Germany.*

Stützle, Thomas and Ruiz, Rubén (2018). "Iterated Local Search". *Handbook of Heuristics.* Ed. by Rafael Martí, Panos M. Pardalos, and Mauricio G. C. Resende. Springer International Publishing: Cham, 579–605.

Taillard, E. (1990). "Some efficient heuristic methods for the flow shop sequencing problem". *European Journal of Operational Research* 47(1), 65 –74.

Taillard, E. (1993). "Benchmarks for basic scheduling problems". *European Journal of Operational Research* 64(2). Project Management anf Scheduling, 278 –285.

Talbi, E. G. (2009). *Metaheuristics: From design to implementation.* John Wiley & Sons.

Tseng, Fan T., Stafford Jr., Edward F., and Gupta, Jatinder N. D. (2004). "An empirical analysis of integer programming formulations for the permutation flowshop". *Omega* 32, 285 –293.

Vallada, Eva, Ruiz, Rubn, and Framinan, Jose M. (2015). "New hard benchmark for flowshop scheduling problems minimising makespan". *European Journal of Operational Research* 240(3), 666 –677.

Watson, Jean-Paul, Barbulescu, Laura, Whitley, L. Darrell, and Howe, Adele E. (2002). "Contrasting structured and random permutation flow-shop scheduling problems: Search-space topology and algorithm performance". *INFORMS Journal on Computing* 14(2), 98–123.

Wilson, J. M. (1989). "Alternative formulations of a flow-shop scheduling problem". *Journal of the Operational Research Society* 40(4), 395–399.