CORA Cork Open Research Archive
Cartlann Taighde Oscailte Chorcaí

Title	Local search methods for the cumulative flow shop scheduling problem
Authors	Missaoui, Ahmed;Neufeld, Janis S.;O'Sullivan, Barry
Publication date	2026-07-01
Original Citation	Missaoui, A, Neufeld, J S & O'Sullivan, B 2026, 'Local search methods for the cumulative flow shop scheduling problem', Operations Research Perspectives, vol. 17, 100403, pp. 1-16. https://doi.org/10.1016/j.orp.2026.100403
Type of publication	Article (peer-reviewed)
Link to publisher's version	https://www.scopus.com/pages/publications/105043365267 - 10.1016/j.orp.2026.100403
Rights	© 2026, authors. Published by Elsevier Ltd. This is an open access article under the CC BY license (http://creativecommons.org/licenses/by/4.0/) - https://creativecommons.org/licenses/by/4.0/
Download date	2026-11-14 00:07:35
Item downloaded from	https://hdl.handle.net/10468/19615

UCC
University College Cork, Ireland
Coláiste na hOllscoile Corcaigh

Operations Research Perspectives 17 (2026) 100403

Contents lists available at ScienceDirect

Operations Research Perspectives

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

Local search methods for the cumulative flow shop scheduling problem

Ahmed Missaoui [a], Janis S. Neufeld [b,*], Barry O'Sullivan [a]

[a] University College Cork, Cork, Ireland
[b] Otto-von-Guericke University Magdeburg, 39106 Magdeburg, Germany

ARTICLE INFO

Keywords:
Flow shop scheduling
Cumulative scheduling
Metaheuristics
Machine capacity
Multiple objectives

ABSTRACT

Scheduling problems are typically studied under the assumption that each resource can process only one job at a time. However, in many real-world settings, resources are capable of processing multiple jobs simultaneously, leading to cumulative scheduling problems, where each resource can process several jobs up to a specified capacity. While cumulative scheduling has been investigated in job shops and parallel-machine contexts, it remains unexplored in flow shop environments. In this work, we introduce the cumulative flow shop scheduling problem as a generalization of the classical flow shop. We propose a mixed-integer programming formulation along with two metaheuristic approaches – an iterated local search algorithm and an iterated greedy algorithm – designed to minimize total flow time. Both methods incorporate problem-specific features, such as an adapted initial solution and a rescheduling mechanism to improve resource utilization. With this, they demonstrate strong performance in a comprehensive computational study. Additionally, we analyze trade-offs between total flow time and resource utilization, and investigate how varying job sizes and resource capacities affect outcomes. Our findings provide actionable managerial insights and open new avenues for research on cumulative scheduling problems.

1. Introduction

The flow shop scheduling problem has received significant attention in research due to its high relevance for planning efficient production systems [1,2]. It is characterized by a set of jobs that need to be processed on a number of machines. Thereby, the sequence of machines is given and identical for all jobs. Flow shops are typically found in flowline manufacturing systems, forming one of the most common production layouts. The sequencing and scheduling of the jobs is a key planning task to increase machine utilization, lower work-in-progress, and ensure timely deliveries [3,4]. However, in some practical applications, resources can handle several jobs simultaneously up to a certain capacity. An example forms the planning in growing facilities of agricultural technology companies [5]. Here, newly developed plants must be scheduled in multiple greenhouses with different environmental characteristics and capacities. This problem is referred to as cumulative scheduling [6]. Similar environments exist, for example, in thermal production processes, where several products may be processed in an oven simultaneously. In contrast to batch scheduling, each job may enter and leave a resource individually. Further applications are reported from project scheduling [7]. Independent of the application area, cumulative machines lead to very different problem characteristics compared to the classical flow shop; the parallel processing of jobs adds a second dimension to the sequencing of jobs, which is similar to a packing problem. Besides its practical applications, the cumulative flow shop is also compelling for theoretical research as it constitutes a generalization of the classical flow shop. Nevertheless, cumulative scheduling problems have only rarely been considered in the scheduling literature. While they have been introduced in parallel machine environments [7] and job shops [6], to the best of our knowledge, the cumulative flow shop has not been studied until now.

Our study aims to close this gap and provides the following contributions:

- We introduce the cumulative flow shop and propose efficient solution methods for its solution. The iterated greedy- and iterated local search-based heuristics include problem-specific components that are proven to improve solution quality, namely an adapted rule for generating an initial solution, a block operator, and a rescheduling mechanism that improves resource utilization during the decoding.
- Based on a computational study, we provide a deeper understanding of the distinct characteristics of the cumulative flow shop problem, such as the relationship between the total flow time objective and machine utilization. Besides, the impact of varying job sizes and resource capacities is analyzed.

* Corresponding author.
E-mail address: janis.neufeld@ovgu.de (J.S. Neufeld).

https://doi.org/10.1016/j.orp.2026.100403
Received 11 March 2026; Received in revised form 20 May 2026; Accepted 20 June 2026
Available online 30 June 2026
2214-7160/© 2026 The Authors. Published by Elsevier Ltd. This is an open access article under the CC BY license (http://creativecommons.org/licenses/by/4.0/).

- The findings enable us to derive managerial implications and lay the foundation for further investigation into cumulative scheduling problems.

The remainder of the paper is structured as follows. Section 2 summarizes the related literature. Based on the problem definition, we present a linear mixed-integer program for the problem at hand in Section 3. Section 4 explains the procedure of the developed solution methods. In Section 5, a computational study proves their effectiveness. A further problem analysis is provided in Section 6, which discusses the relationship between total flow time and machine utilization, and the influence of capacity changes. Based on these results, important managerial insights are derived. Finally, we conclude our study and give an outlook on promising research directions in Section 7.

2. Related literature

To the best of our knowledge, we are the first to study the cumulative flow shop scheduling problem. But even for other machine environments, cumulative scheduling has hardly been considered in the scientific literature. In the following, we summarize the relevant literature in the field and also highlight recent methodological developments in the field of flow shop scheduling.

The cumulative scheduling problem has originally been introduced for job shop environments by Nuijten and Aarts [8], who refer to it as *multiple capacitated job shop scheduling problem*. It is formulated and solved as a constraint satisfaction problem. Carlier and Néron [9] and Baptiste et al. [7] study cumulative single-machine and cumulative parallel-machine problems. While Carlier and Néron [9] develop several lower bounds, Baptiste et al. [7] derive conditions for feasible schedules and use these within a branch-and-bound procedure. Apart from these theoretical works, Godard et al. [6] minimize makespan by a large neighborhood search algorithm in the cumulative job shop scheduling problem. The constraint programming-based approach is proven to solve instances of this problem effectively. After their work, cumulative scheduling did not receive any attention in the literature for several years. Only recently, the cumulative job shop problem was extended by the order acceptance decision by Linß et al. [5,10], who focus on an application in agribusiness. They propose an MILP to tackle a bi-objective planning problem, namely the maximization of accepted jobs and the minimization of waiting times. Besides, cumulative scheduling problems can also be modeled as resource-constrained scheduling problems [7]. According to Hartmann and Briskorn [11], the problem at hand can be defined as a special case of the resource-constrained multi-project scheduling problem. A survey of recent approaches for solving this problem is provided by Sánchez et al. [12]. However, the studied problems in this field do not match the characteristics of a cumulative flow shop problem with total flow time minimization. Hence, to the best of our knowledge, there are no studies focusing on the cumulative flow shop yet.

In general, the flow shop scheduling problem has received considerable attention in research and practice [13]. From a methodological perspective, partly very complex solution approaches have been proposed. At the same time, rather simple local search-based methods and iterated greedy algorithms are proven to solve flow shop problems very effectively [13,14, e.g.,]. A big advantage of these methods is that they are easy to implement, and the simple framework allows more iterations within the given computation time [15]. Lourenço et al. [16] demonstrated the impressive effectiveness of ILS across various scheduling problems, including single machine, flow shop, and job shop. Iterated Greedy (IG) algorithms have been successfully applied to a variety of scheduling problems, often achieving state-of-the-art performance, including permutation flow shops [17], hybrid flow shops [18], parallel machine problems [19], etc. Recently, Gelvez et al. [20] embedded new neighborhood structures into an MIP-based matheuristics to solve the permutation flow shop with makespan objective. It significantly improves results from a conventional local search algorithm. However, no comparison to metaheuristics had been conducted.

Another development is the integration of machine learning into a metaheuristics framework. For example, for the permutation flow shop with makespan objective, Du et al. [21] combine an artificial neural network with an iterated greedy framework. For the same problem, Yang et al. [22] cluster a population of solutions and apply a parallel Q-learning search method to identify promising regions of the solution space. Similarly, Li et al. [23] embed Q-learning into an artificial bee colony optimization to enhance the choice of neighborhood structures. Interestingly, the aforementioned methods have not been compared to iterated greedy algorithms in the respective experimental studies.

The mentioned literature shows that local search-based approaches and iterated greedy algorithms still provide state-of-the-art performance for flow shop scheduling. This is also true for extensions of the flow shop problem, such as permutation flow shops with sequence-dependent setups [21], distributed flow shops [24], or assembly flow shops with no-wait constraints [25]. While the main focus of this study is not on the development of new methods, it is still critical to use effective heuristic approaches in order to solve and analyze a novel problem such as the capacitated flow shop. Therefore, we apply iterated local search as well as an iterated greedy algorithm with problem-specific components to study the problem at hand.

3. The cumulative flow shop scheduling problem

3.1. Problem definition

The studied problem can formally be defined as follows. Given is a set of jobs $j \in J$ and a set of machines $i \in I$. The time horizon H is defined by the set of time periods $t \in T$. Each job must be processed on all machines in an identical technological order with a processing time p_{ji}, i.e., a flow shop is considered. The cumulative characteristic states that each machine can process more than one job simultaneously. The quantity is limited by a constant machine capacity Q_i, while the required capacity per job is defined by its size q_j, which we assume to be constant on all machines. However, without loss of generality, the job size could be supposed to be variable per machine. The total size of the jobs processed simultaneously on a machine must not exceed its capacity. Jobs cannot be processed partly, meaning that the complete size of a job needs to be processed at once and without interruption, or the job needs to be postponed, for example, if a machine's capacity is insufficient at the respective moment. A special case of the problem with a job size of $q_j = 1, \forall i, j$, and a capacity of $Q_i = 1, \forall i$, reduces the cumulative flow shop to the well-known NP-hard flow shop scheduling problem. The objective is to minimize the total flow time, i.e., the sum of flow times of all jobs.

We provide an illustrative example to describe how to obtain the flow time value for a small instance of five jobs and three machines. The sizes of jobs and the processing time per machine are given in Table 1 while the capacity of each machine is assumed to be 10. The resulting Gantt chart job sequence $\pi = (1, 2, 3, 4, 5, 6)$ is shown in Fig. 1. Considering the completion times of all jobs on the last machine this results in a total flow time of $10+15+17+19+22+27 = 110$. Note that several jobs are processed in parallel on each machine. However, for example, on Machine 2, Job J_6 cannot be processed immediately after its completion on Machine 1 because of the limited machine capacity of Machine 2. Only after Job J_5 has been completed, J_6 can be started at time unit 10. At the same time, both jobs are partly processed in parallel to Job J_1 on Machine 2. This shows the effect of the machine capacity in the cumulative flow shop.

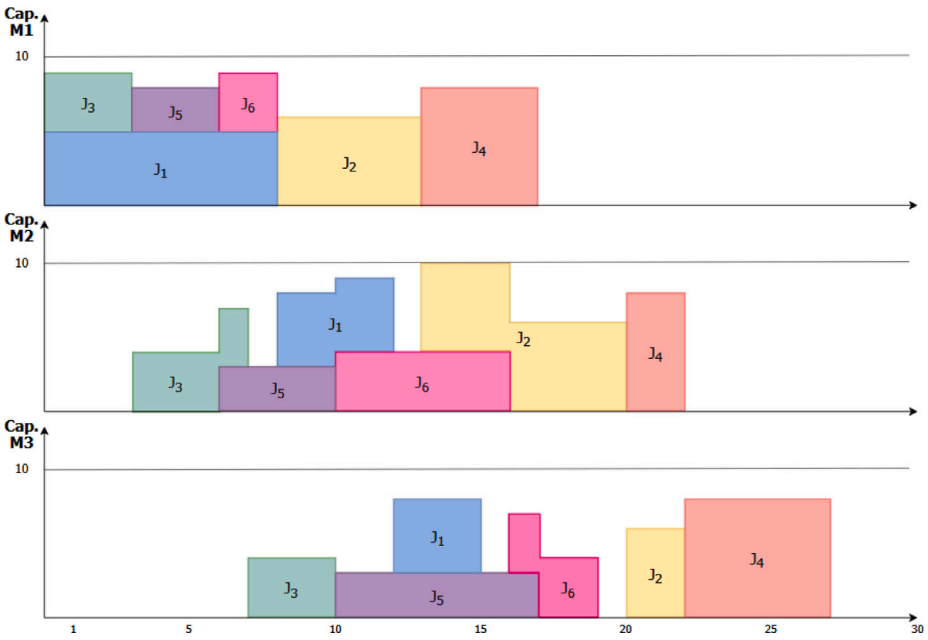


Fig. 1. Gantt chart for job sequence π.

Table 1
Sizes of jobs and processing times per machine.

Jobs j	Size q_j	p_{j1}	p_{j2}	p_{j3}
J_1	5	8	4	3
J_2	6	5	7	2
J_3	4	3	4	3
J_4	8	4	2	5
J_5	3	3	4	7
J_6	4	2	6	3

Table 2
Notation.

Sets and parameters	
n	Total number of jobs $j, k \in J$
m	Total number of machines $i \in I$
H	Time horizon in time units $t \in T$
p_{ji}	Processing time of job j on machine i
q_j	Required capacity (size) of job j on all machines
Q_i	Available capacity on machine i
Decision variables	
TFT	Total flow time
S_{ji}	Starting time of job j on machine i
X_{jit}	Binary decision variable; 1 if job j starts at time t on machine i, 0 otherwise

3.2. Mathematical formulation

To ensure compliance with the capacity restriction at all times, a time-indexed decision variable X_{jit} indicates whether a job j is started on machine i at time period t. A time-indexed variable enables a straightforward modeling of the capacity constraint at any time t. At the same time, the number of variables may become enormous for large time horizons H. Besides, the starting time of job j on machine i is defined by S_{ji}. The notation is summarized in Table 2 and includes the parameters introduced in Section 3.1. With this, a mixed-integer programming model is formulated as follows.

$$\min TFT \tag{1}$$

$$S_{ji} = \sum_{t=1}^{H} t \cdot X_{jit}, \qquad \forall j \in J, i \in I \tag{2}$$

$$\sum_{t=1}^{H-p_{ji}} X_{jit} = 1, \qquad \forall j \in J, i \in I \tag{3}$$

$$S_{ji} + p_{ji} \leq S_{ji+1}, \qquad \forall j \in J, i \in I \setminus \{m\} \tag{4}$$

$$TFT \geq \sum_{j=1}^{n} \left(S_{jm} + p_{jm}\right), \tag{5}$$

$$\sum_{j=1}^{n} \sum_{t'=\max(1,t-p_{ji}+1)}^{t} X_{jit'} \cdot q_j \leq Q_i, \qquad \forall i \in I, \forall t \in T \tag{6}$$

$$X_{jit} \in \{0,1\}, \qquad \forall j \in J, \forall i \in I, \forall t \in T \tag{7}$$

$$S_{ji} \geq 0, \qquad \forall j \in J, \forall i \in I \tag{8}$$

Eq. (1) is the objective function minimizing the total flow time. Constraint (2) links the two decision variables X_{jit} and S_{ji}, and defines the starting times of the jobs with the help of the time index t. Constraint (3) specifies that each job starts exactly once. The overlapping of two subsequent operations of a job is prevented by Constraint (4) by defining that an operation can only start if the job has been completed on the previous machine. Constraint (5) determines the total flow time as the sum of completion times of all jobs on the last machine. Constraint (6) ensures that the total required capacity is less than or equal to machine capacity at time t. Here, all jobs processed simultaneously on a machine are considered with their respective sizes q_j. While X_{jit} only indicates the starting time of each operation, the definition of index t' ensures that all periods during which the operation is processed are accounted for. Finally, Eqs. (7) and (8) define the domains of the variables.

4. Solution methods

Unfortunately, to date, no solution approach has been developed for the cumulative flow shop scheduling problem. Due to its NP-hard nature, we propose two metaheuristic algorithms to solve instances

of reasonable size efficiently. As shown in Section 2, local search-based methods and iterated greedy algorithms are proven to solve flow shop problems very effectively. Hence, we propose both an iterated local search algorithm (ILS) and an iterated greedy algorithm (IG) that include a new rescheduling mechanism to adapt to the characteristics of the cumulative flow shop.

Even though several jobs may be processed in parallel on a machine in the cumulative flow shop, both algorithms use a simple job sequence as the solution representation. This encoding enables the use of efficient neighborhood structures known from flow shop scheduling. In the decoding procedure, the jobs are scheduled according to the defined job sequence. Iteratively and machine by machine, each operation of the current job is added at the end of the incumbent schedule. If both the machine's capacity and the completion time of the job's previous operation allow an earlier start, the current operation is brought forward and started as early as possible. However, the starting and ending times of already scheduled jobs and operations are not changed. This procedure is referred to as *standard insertion* and used for both algorithms. The detailed steps of the algorithms are described in the following.

4.1. Iterated local search algorithm

Iterated local search ILS is an iterative approach that starts with an initial solution and then refines it using local search to reach a local optimum. To avoid getting stuck in this local optimum, a perturbation phase is added, allowing the exploration of new solutions. Despite its simplicity, this approach demonstrates great efficiency in solving optimization problems. Lourenço et al. [16] demonstrated the impressive effectiveness of ILS across various scheduling problems, including single machine, flow shop, and job shop. Based on this, the ILS is selected for the resolution of the cumulative flow shop scheduling problem. The general procedure of the iterated local search is depicted in Algorithm 1 and described in detail in the following sections. Based on an initial solution (see Section 4.1.1), the local search procedure described in Section 4.1.2 is applied. To meet the specifics of the cumulative flow shop, a rescheduling step further improves the generated solutions (Section 4.1.4). Within the main loop of the ILS, the incumbent solution is iteratively perturbed to reach new areas of the solution space (see Section 4.1.3), before again local search and rescheduling are applied. The acceptance criterion for newly generated solutions is detailed in Section 4.1.5. We choose a time limit as termination criterion.

Algorithm 1: Iterated Local Search (ILS)

```
π ← Initial solution;
π* ← LocalSearch(π);
π* ← Rescheduling(π*);
π_best ← π*;
while termination criterion is not met do
    π' ← Perturbation(π*);
    π'' ← LocalSearch(π');
    π''' ← Rescheduling(π'');
    π* ← AcceptanceCriterion(π*, π''', π_best);
end
```

4.1.1. Initial solution

As with most metaheuristics, the quality of the initial solution affects the overall effectiveness. Here, two simple dispatching rules are used. In scheduling problems with a flow time criterion, the Shortest Processing Time (SPT) heuristic has proven to be highly efficient in getting competitive initial solutions [26,27]. The principle of SPT is to sort jobs in ascending order according to their sum of processing times.

The second heuristic, referred to as SPTS, accounts for the distinctive characteristic of cumulative scheduling in the sequencing, i.e., differing job sizes. Here, jobs are sorted according to the product of their processing time and job size in ascending order, $p_{ij} \cdot q_j$. Both heuristics are applied, and the best one is selected as the initial solution. Besides SPT and SPTS, we tested six other heuristics. The first two are based solely on the jobs' sizes: the smallest job size rule (SJS), which sorts jobs in ascending order of their size, and the largest job size rule (LJS), which sorts jobs in descending order. The third heuristic is the Shortest Processing Times at Stage Zero (SPT0), while the fourth sorts the jobs according to the Smallest Ratio of Processing Time and Sizes (SRPS), $\frac{p_{ij}}{q_j}$. Finally, two variants of NEH were tested: the first is the basic NEH, which uses only processing times for job sorting, while the second variant (NEH-PS) uses the product of processing times and job sizes for job sorting. All heuristics were tested in a preliminary experiment with a benchmark of 50 instances. The results indicate that SPT and SPTS outperformed the others significantly, achieving the best results in 42% (SPT) and 54% (SPTS) of all cases, respectively. Since the other rules did not lead to comparable results, they are not considered in the following.

4.1.2. Local search

Once the initial solution is generated, the local search procedure is started to improve solution quality. In this phase, two distinct simple local search methods are explored. The first approach, proposed in Naderi et al. [28], uses a simple insertion method. Starting with the first job, each job is inserted into a random position in the sequence. This process is stopped after the first improvement or if all jobs have been considered.

The second proposed local search is inspired by the first one. However, instead of selecting a single job each time, a block of BL jobs is selected and reinserted each time in a random position outside the block. This reflects a key characteristic of cumulative scheduling problems: high-quality solutions should utilize machine capacities efficiently. Accordingly, the total job size of concurrently processed jobs (i.e., jobs appearing consecutively in the sequence) should be close to the available machine capacity. These structures are intended to be preserved by moving blocks of jobs rather than individual jobs. The block size BL is set to the average number of jobs that can be processed simultaneously on one machine. It is calculated by dividing the average job size over all jobs and machines $\bar{q}_j$ by the average capacity of all machines $\bar{Q}_i$:

$$BL = \frac{\bar{Q}_i}{\bar{q}_j}$$

Algorithm 2 shows the pseudo-code of the second local search procedure.

Algorithm 2: Block Insertion

```
Input: schedule π, block length BL
Output: new schedule π'
improvement := true ;
while improvement == true do
    improvement := false;
    for k := 1 to n − BL do
        block_start = k, block_end = k + BL;
        π' = Insert block in a random position;
        if Fit(π') < Fit(π) then
            improvement := True;
            π' := π;
            Break;
        end
    end
end
```

The two local search procedures are evaluated in the calibration experiments to determine the most effective operator.

4.1.3. Perturbation

The perturbation phase is a key operator of the ILS approach [16]. After an intensification phase using local search, the perturbation leads to a different region of the solution space to avoid getting stuck in a local optimum. Starting from the current solution, a set of $PerL$

random perturbation moves is applied. These are with equal probability either the insertion of a random job or the swap of two jobs. After each perturbation, the obtained solution is evaluated and accepted for the next iteration if it is better than the current one. This process is repeated $PerSol$ times. The solution with the best flow time is used in the next iteration. The number of iterations $PerL$ and the number of changes within each iteration are calibrated. Algorithm 3 shows the pseudo-code for the perturbation operator.

After each perturbation phase, again, the local search described in Section 4.1.2 is applied to refine the newly generated solution.

Algorithm 3: Perturbation Procedure

```
Input: schedule π, PerL, PerSol
Output: new schedule π*
bestValue := ∞;
for k := 1 to PerSol do
    π' := π;
    for i := 1 to PerL do
        op := rand() mod 2;
        if op = 0 then
            Apply Insertion move on π';
        else
            Apply Interchange move on π';
        end
    end
    value := Evaluate(π');
    if value < bestValue then
        bestValue := value;
        π* := π';
    end
end
π := π*;
```

4.1.4. Rescheduling

The idea of rescheduling is to maximize the utilization of idle machine capacity after a solution is decoded. In many cases, even when an available idle time on a machine has enough capacity to process the next job in the sequence, its duration may not be sufficient for the required processing time. As a result, these capacities remain unused, potentially decreasing resource utilization. Therefore, we introduce a rescheduling mechanism to better exploit available capacity. Its main steps are presented in Algorithm 4

The rescheduling mechanism uses the first available space that can accommodate a job. Based on the job sequence of the current solution, the jobs are iteratively added to an initially empty schedule as early as possible. If the remaining machine capacity at time t is sufficient to process the current job for at least one time unit, two options are considered:

- **Job insertion and rescheduling:** the current job j is assigned to the available idle time, and all remaining jobs (i.e., those that have not yet started at time t) are rescheduled accordingly.
- **Standard insertion:** The job is added at the end of the schedule where both machine capacity and the length of the time slot are sufficient. This option equals the standard decoding procedure.

After assigning each operation according to these two options, the total flow time for both solutions is evaluated, and the option yielding the better value is selected. An illustrative example is provided in Fig. 2 where 6 jobs are processed on a machine of the flow shop. The first Gantt chart shows the processing of $\pi = (1, 2, 3, 4, 5, 6)$ without rescheduling resulting in a total flow time of $TFT = 30$. This schedule corresponds to the standard insertion. Assume that Job 6 is now rescheduled. At time $t = 1$, there is a slot with sufficient capacity for processing Job 6. However, the slot is too short to accommodate the whole processing time. Therefore, in the rescheduling procedure, Job 6 is inserted at time $t = 1$, and Job 4 and Job 5 are rescheduled and started later. This new solution leads to an improved total flow time of $TFT = 28$, and is therefore accepted. Because changes during the rescheduling cannot be encoded in the solution representation, the rescheduling mechanism is only applied at the end of each iteration of the main loop of ILS (see Algorithm 1).

Algorithm 4: Rescheduling

```
Input : initial schedule π_0, Fit_0
Output: new schedule π, Fit
for m = 1 to machine do
    for j = 1 to jobs do
        Set Start = 0 and Is_scheduled = false;
        while Is_scheduled = false do
            if Check capacity(j, start) = true then
                Is_scheduled = True;
                π.insert(j);
                Fit = Fitness(π);
                Update capacity;
            end
            else
                if there is available capacity at t then
                    S0 = Insert j at the end of π;
                    S1 = Insert j at t = start;
                    F0 = Fitness(S0);
                    F1 = Fitness(S1);
                    if F0 < F1 then
                        π = S0;
                        Break;
                        else
                            π = S1;
                            Break;
                        end
                    end
                    Update capacity;
                end
            end
            Start++;
        end
    end
    Sort jobs based on Completion time
end
```

4.1.5. Acceptance criterion

Each iteration of the ILS algorithm can produce a new solution that differs from the current one. The resulting sequence can be either an improvement or a deterioration compared to the initial one. Dealing only with better solutions in subsequent steps could lead the search to become stuck in a local optimum, preventing it from converging to a global optimum. Therefore, the Simulated Annealing operator is applied to decide whether the incumbent solution will be accepted. In case the solution is worse, it is accepted with a probability given by

$$\exp\left(\frac{TFT(\pi') - TFT(\pi)}{Temp}\right), \tag{9}$$

where

$$Temp = T \cdot \frac{\sum_{i=1}^{n}\sum_{j=1}^{m} p_{ij}}{n \cdot m \cdot 10}, \tag{10}$$

and T is a parameter that has to be calibrated as shown by Ruiz and Stützle [13]

4.2. Iterated Greedy algorithm

Iterated Greedy (IG) algorithms have been successfully applied to a variety of scheduling problems, including flow shops. This paper introduces an adaptation of the IG algorithm for the considered cumulative flow shop scheduling problem. IG is closely related to the

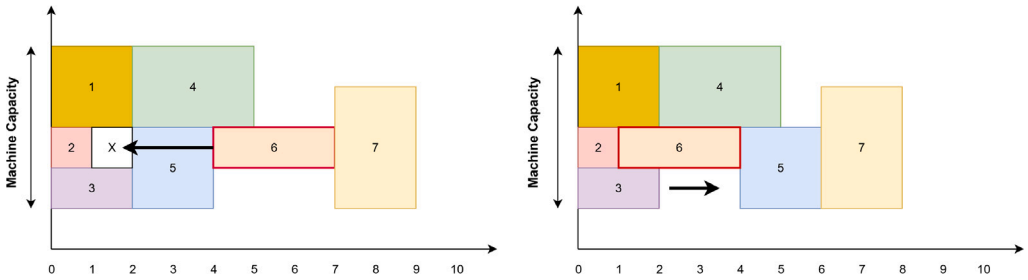


Fig. 2. Example of rescheduling Job 6 on one machine.

ILS framework and only replaces the perturbation step of ILS with a destruction reconstruction, which is the key operator of this approach. In the destruction phase, a set of d random jobs is removed from the current solution, where d is a parameter that has to be calibrated. After deconstruction, the reconstruction starts with the insertion of all removed jobs one by one in the part sequence. Each job is inserted into every possible position in the sequence. From these, the position gaining the lowest TFT value is selected. This process continues until all jobs have been reinserted. Apart from the difference in destruction and reconstruction, the remaining steps of IG are identical to those of ILS and shown in Algorithm 5. Similar to ILS, a time limit is used as the termination criterion.

Algorithm 5: Iterated Greedy Algorithm (IG)

$\pi \leftarrow$ Initial solution;
$\pi^* \leftarrow$ LocalSearch(π);
$\pi^* \leftarrow$ Rescheduling(π^*);
$\pi_{best} \leftarrow \pi^*$;
while *termination criterion is not met* **do**
  $\pi' \leftarrow$ Destruction: remove d jobs from π^* to π^d;
  for $i = 1$ *to* d **do**
    $\pi' \leftarrow$ Construction: insert job $\pi^d(i)$ to π' with minimum TFT;
  end
  $\pi'' \leftarrow$ LocalSearch(π');
  $\pi''' \leftarrow$ Rescheduling(π'');
  $\pi^* \leftarrow$ AcceptanceCriterion($\pi^*, \pi''', \pi_{best}$);
end

5. Computational study

5.1. Experimental setting

We conducted a comprehensive computational study to test the proposed solution approaches and to gain insights into the characteristics of the cumulative flow shop scheduling problem. Furthermore, in Section 5.4 ILS and IG are compared against several metaheuristics from the literature originally developed for related flowshop scheduling problems. Two data sets, namely *small* and *large*, have been generated, similar to Vallada et al. [29] and Linß et al. [5]. The *small* instance set is used to test the MILP formulation and compare its results with those of the metaheuristics ILS and IG. It includes instances with the number of jobs $n \in \{10, 15\}$ and the number of machines $m \in \{3, 5, 10\}$. For each combination of n and m, we generated 10 instances, resulting in a total of 60 instances in this set. The *large* instance set is used to compare the metaheuristic approaches and comprises instances with $n \in \{20, 30, 50, 100, 200, 500\}$ and the number of machines $m \in \{3, 5, 10\}$.

For both test sets, the processing times are uniformly chosen from the range $[1, 50]$. The machine capacities are tested at three levels $Q \in \{20, 50, 100\}$. Job sizes, i.e., the required capacity q_j, were randomly drawn from a uniform distribution between either $[1, 20]$ or $[5, 15]$. This distinction allows comparison of results based on job-size variance. For each combination of n, m, Q, and q_j, 10 instances were generated, resulting in a total of 1080 test problems.

All algorithms were implemented in C++, while the MILP was solved using IBM CPLEX 20.1.0. Each instance was solved 5 timeswith each algorithm on an Intel(R) Xenon(R) CPU E5-4627 with a 3.3 GHz clock speed and 768 GB RAM.

To evaluate the solution quality of the proposed methods, the relative percentage deviation (RPD) is calculated by

$$RPD = \frac{\overline{TFT} - TFT_{best}}{TFT_{best}} \cdot 100, \tag{11}$$

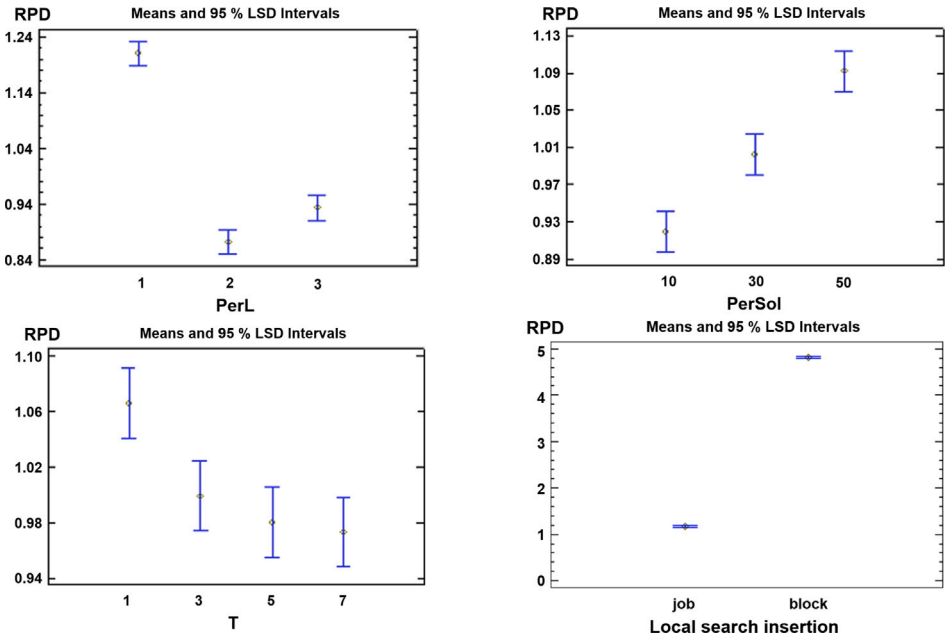
with TFT_{best} representing the best-known total flow time of all methods and $\overline{TFT}$ being the average total flow time by the respective method. Similar to Missaoui and Ruiz [18], the time limit was set to $(n \,.\, m \,.\, \pi_{CPU})$ milliseconds, knowing that π_{CPU} is tested at two different levels $\{30 \; and \; 60\}$. Let us consider $\pi_{CPU} = 30$, this rule will give 9 s for an instance of 100 jobs and 3 machines.

5.2. Calibration

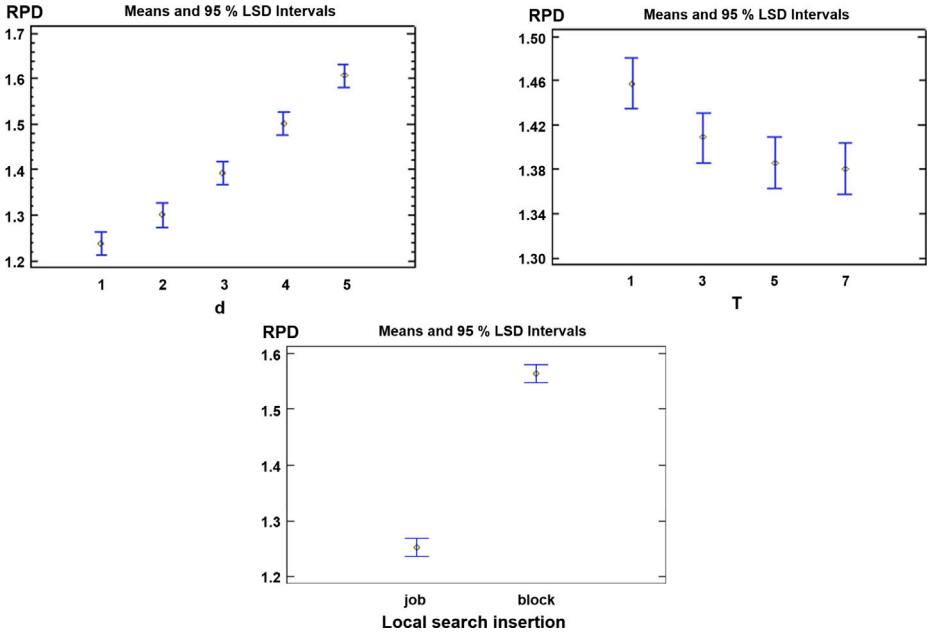
The proposed ILS and IG approaches were calibrated to determine optimal parameter settings using a design of experiments framework. The calibration testbed consists of 90 instances of varying size, created using the procedure described for the *large* instance set above. It is ensured that an equal number of instances is drawn from each job size distribution. For ILS, three parameters were tested: perturbation length (PerL $\in \{1, 2, 3\}$), perturbed solution size (PerSol $\in \{10, 30, 50\}$), and temperature factor ($T \in \{1, 3, 5, 7\}$), yielding 36 parameter combinations. For IG, two parameters were examined: destruction size ($d \in \{1, 2, 3, 4, 5\}$) and temperature factor ($T \in \{1, 3, 5, 7\}$), resulting in 20 combinations. All ranges were carefully chosen based on preliminary tests, where larger ranges of the parameters were applied for a smaller set of instances. For both algorithms, according to Section 4.1.2, a deterministic block size BL is used for the block insertion-based local search procedure. For both approaches, the two local search procedures, i.e., the single job insertion and the block insertion, are tested.

Each configuration was executed five times over the 90 calibration instances, generating $32,400$ results for ILS and $18,000$ for IG. The relative percentage deviation (RPD) served as the response variable, and results were analyzed using Analysis of Variance (ANOVA).

Fig. 3 presents the parameter plots for ILS and IG, respectively. Based on these results, the ILS parameters are set to PerL = 2, PerSol = 10, and $T = 7$. Surprisingly, the block insertion local search shows significantly worse performance compared to the single job insertion local search. A reason for this might be that even though subsets of jobs might form a promising block on one machine, this might not be valid for the same block on the other machines. Hence, only the single job insertion local search is used for the ILS in the following experiments. Besides, the choice of the perturbation-related parameters PerL and PerSol shows a significant influence on the algorithm's performance. For IG, the destruction size is set to $d = 1$ and the temperature factor to $T = 7$, even though no statistically significant difference can be observed compared to $T = 5$, and this would also be a valid choice. Again, the block insertion local search does not show a comparable



(a) Means plots for the ILS parameters



(b) Means plots for the IG parameters

Fig. 3. Results of parameter calibration.

performance and is omitted in the following. All other competing methods tested in Section 5.4 are calibrated using the same approach. The results of the calibration are available as supplementary material.

5.3. Results for small instances

For the small instances set, Table 3 displays the results with job sizes q_j following $\mathcal{U}[1, 20]$, while Table 4 depicts job sizes from $\mathcal{U}[5, 15]$.

Only very small problems with up to 15 jobs and 5 machines can be solved to optimality by the MILP. Already for instances with 15 jobs and ten machines, the CPLEX solver shows extensive RPD values (580.28% for $q_j \sim \mathcal{U}[1, 20]$) and average gap values of up to 86.85%; for instances with $q_j \sim \mathcal{U}[5, 15]$, the MILP cannot even find a solution within the time

Table 3
Average RPD values in % for (1–20 job sizes) instances.

n	m	IG	ILS	MILP	Avg. time	Ave. gap
10	3	0.00	0.00	0.00	56	0.00
	5	0.16	0.16	0.00	3600	0.0295
	10	0.00	0.00	7.50	3600	0.1292
Avg. ($n = 10$)		**0.05**	**0.05**	**2.50**		
15	3	0.78	0.05	0.00	3600	0.0325
	5	0.00	0.00	32.70	3600	0.3123
	10	0.11	0.03	580.48	3600	0.8685
Avg. ($n = 15$)		**0.30**	**0.03**	**204.39**		

Table 4
Average RPD values in % for (5–15 job sizes) instances.

n	m	IG	ILS	MILP	Avg. time	Ave. gap
10	3	0.50	0.50	0.00	115	0.0000
	5	0.00	0.00	4.37	3600	0.1126
	10	0.00	0.00	12.04	3600	0.1817
Avg. ($n = 10$)		**0.17**	**0.17**	**8.99**		
15	3	0.04	0.01	7.86	3600	0.1443
	5	0.03	0.00	20.66	3600	0.2267
	10	0.00	0.00	NA	3600	NA
Avg. ($n = 15$)		**0.03**	**0.01**	**14.26**		

limit. On the one hand, this proves the necessity for heuristic solution approaches to solve problems of reasonable size. On the other hand, it indicates that problems with more evenly distributed job sizes are more difficult to solve. The reason for this could be twofold: an uneven distribution of job sizes may lead to some jobs being nearly irrelevant due to very small processing times. These instances can be easier to solve. Besides, many similar jobs can result in convergence problems in the solution process due to symmetry, i.e., many different solutions may lead to the same objective function value [30,31]. This may happen more often if several jobs of similar or even equal size exist.

Both proposed metaheuristics are able to find the best or even optimal solutions for most instances, and the average RPD values are always below 1%, with a maximum of 0.78% for IG in Table 3. This demonstrates the algorithms' suitability to solve small instances effectively. While ILS seems to perform better for $n = 15$ and instances with a wider range of job sizes times ($q_j \sim \mathcal{U}[1, 20]$), larger instances need to be taken into account for a comprehensive comparison of algorithms.

5.4. Results for large instances

To compare the proposed algorithms, we solved the large-instance set using IG, ILS, and several well-known metaheuristics. Besides, the variants IGw and ILSw are variants are added, which exclude the rescheduling procedure described in Section 4.1.4 to validate its effectiveness. Since the cumulative flowshop problem has not previously been studied, no benchmark algorithm is directly applicable. To enable a broad comparison across metaheuristic approaches, we selected a hybrid simulated annealing (HSA, 28), a genetic algorithm (GA, 32), and an Artificial Bee Colony Algorithm (ABC, 33). All algorithms have been adapted to meet the requirements of the problem at hand, integrating the newly developed components if possible, such as the initial solution and the local search proposed in Section 4.

Based on the results for the instance set large with $q_j \sim \mathcal{U}[1, 20]$, reported in Table 5, IG and ILS algorithms yield similar results. For shorter computation times with $\pi_{CPU} = 30$ the overall averages are 1.30% for both methods. At the same time, the comparison algorithms result in total average RPD values between 6.67% (HSA) and 9.76% (ABC). While for small instances, especially HSA can still achieve similar results, with increasing n, the gap between the approaches becomes larger. This can also be seen in Fig. 4, which presents boxplots for all algorithms dependent on the job number. Particularly for very large instances ($n \geq 200$), IG and ILS yield 15% to 20% lower average RPD values.

While neither of the proposed methods, ILS and IG, consistently dominates the other, ILS outperforms IG for small instances. The boxplots in Fig. 4 show that, especially, the spread of RPD values is significantly smaller for small instances when comparing ILS and IG. In contrast, for $n \geq 50$ the results converge with IG becoming slightly preferable regarding average and mean RPD values. However, the boxplots do not show any statistical difference. The number of machines does not appear to have a significant effect on the relative performance of the two algorithms in Table 5. The same effects can be observed with larger computational time ($\pi_{CPU} = 60$). While ILS tends to be more effective for smaller instances, IG achieves better overall results for $n \geq 50$. Overall, both methods perform similarly, suggesting that IG benefits slightly more from longer run times.

The impact of rescheduling becomes more pronounced for large-scale instances, while for small n, its influence is negligible. For example, considering $\pi_{CPU} = 30$, while IG and IGw show nearly equal average RPD values for $n = 20$ (0.27% and 0.26%), for $n = 500$ the RPD differences are enormous for IGw compared to IG (16.86% to 1.91%). Similarly, ILS outperforms ILSw for $n = 500$ with 1.92% compared to 16.18%. This proves that the rescheduling procedure is able to significantly improve the effectiveness of the proposed methods. At the same time, the performance difference between IG and IGw as well as ILS and ILSw is similar across all m, i.e., no effect related to the number of machines was observed. Only for $n \geq 200$ and $m = 3$, the variants without rescheduling show slightly lower RPD. In general, rescheduling has a slightly stronger impact on IG compared to ILS, likely because the standard IG variant is weaker without it, resulting in a higher potential for improvement by enhanced algorithm components. Similarly, an increasing computation time decreases the difference between variants with and without the rescheduling procedure. At the same time, ILSw and IGw still outperform the comparison algorithms GA, HSA, and ABC.

Table 6 shows the results for instance set *large* with $q_j \sim \mathcal{U}[5, 15]$, i.e., more evenly distributed job sizes. Again, ILS and IG significantly outperform the comparison algorithms, even though the gap becomes smaller with total average RPD values between 3.68% (HSA for $\pi_{CPU} = 60$) and 6.96% (ABC for $\pi_{CPU} = 30$). In contrast to previous results, for instances with more uniformly distributed job sizes, ILS outperforms IG even for large-scale problems (e.g., 2.63% compared to 2.72% RPD for $n = 500$ and $\pi_{CPU} = 30$). In fact, even the variant without rescheduling ILSw achieves a partly better average performance than IG. Exemplarily, the average RPD with $\pi_{CPU} = 60$ is 0.91% for ILSw compared to 1.05% for IG for $n = 50$. In these instances, the benefit of rescheduling is generally lower compared to other job size distributions.

This finding is also supported by the boxplots in Fig. 5, which do not show a significant difference between the algorithms with and without rescheduling for $n < 200$. From a scheduling perspective, this behavior is expected. The rescheduling procedure attempts to shift jobs forward to fill existing idle times on the machines. Such reinsertion moves are more likely to succeed when operations are short, since shorter processing times can more easily fit into available gaps. When $q_j \sim \mathcal{U}[5, 15]$, however, every operation requires at least five time units, which substantially reduces the likelihood that an operation can be reinserted into existing idle intervals.

6. Problem analysis and discussion

6.1. Hierarchical optimization of total flow time and machine utilization

Flow time is an important objective in scheduling in general, and particularly in our application of the cumulative flow shop in growing facilities. However, given the central role of resource capacities in cumulative scheduling problems, the impact of different solutions on their utilization is especially relevant. Therefore, to facilitate a better understanding of the problem, we analyze the makespan C_{max} as a secondary objective. Makespan is a classical objective in scheduling, closely related to maximizing resource utilization. To provide a valid analysis, we focus on small instances that can be solved to optimality in the following.

By substituting the objective function (2) of the proposed model in Section 3.2 by

$$min C_{max} \tag{12}$$

and adding the following constraint

$$C_{\max} \geq S_{jm-1} + p_{jm-1}, \qquad \forall j \in J, \tag{13}$$

Table 5
Average RPD values in % for (1-20 job sizes) instances.

π_{CPU}	n	m	GA	HSA	ABC	ILSw	IGw	ILS	IG
30	20	3	1.21	0.24	1.10	0.18	0.37	0.16	0.41
		5	0.89	0.08	0.86	0.11	0.29	0.09	0.27
		10	0.72	0.13	0.70	0.10	0.19	0.06	0.22
	Avg. 20		0.94	0.15	0.89	0.13	0.28	0.11	0.30
	30	3	2.39	1.08	3.50	0.72	1.01	0.65	0.96
		5	2.18	1.00	3.27	0.59	0.92	0.56	0.81
		10	1.85	0.79	2.51	0.65	0.86	0.61	0.69
	Avg.30		2.14	0.96	3.09	0.65	0.93	0.61	0.82
	50	3	5.00	2.36	8.51	1.64	1.84	1.37	1.31
		5	4.75	2.16	7.35	1.62	1.75	1.37	1.28
		10	3.72	2.02	5.59	1.42	1.87	1.03	1.14
	Avg.50		4.49	2.18	7.15	1.56	1.82	1.26	1.25
	100	3	9.41	5.97	13.46	4.00	5.45	2.40	2.19
		5	8.75	5.36	11.95	3.84	5.22	1.98	2.08
		10	7.41	6.00	10.43	4.16	5.41	1.68	1.62
	Avg.100		8.52	5.78	11.95	4.00	5.36	2.02	1.96
	200	3	13.63	11.80	14.97	8.63	9.93	2.24	1.88
		5	14.17	13.24	16.71	9.34	11.59	1.99	1.57
		10	12.77	11.61	14.38	8.77	11.31	1.36	1.18
	Avg.200		13.52	12.21	15.35	8.91	10.94	1.86	1.54
	500	3	18.01	16.29	18.01	14.00	14.32	1.75	1.68
		5	20.95	19.64	20.95	17.25	17.97	2.42	2.40
		10	21.41	20.35	21.41	17.28	18.29	1.61	1.66
	Avg.500		20.12	18.76	20.12	16.18	16.86	1.92	1.91
30 Total			**8.29**	**6.67**	**9.76**	**5.24**	**6.03**	**1.30**	**1.30**
60	20	3	1.05	0.16	0.95	0.14	0.31	0.11	0.32
		5	0.86	0.05	0.75	0.06	0.19	0.04	0.19
		10	0.63	0.06	0.54	0.07	0.16	0.04	0.18
	Avg.20		0.85	0.09	0.75	0.09	0.22	0.06	0.23
	30	3	2.19	0.90	2.83	0.55	0.81	0.49	0.76
		5	2.01	0.75	2.76	0.43	0.71	0.44	0.66
		10	1.59	0.53	2.09	0.45	0.68	0.40	0.54
	Avg.30		1.93	0.73	2.56	0.48	0.73	0.45	0.65
	50	3	4.50	1.88	7.68	1.24	1.41	1.09	0.99
		5	4.48	1.74	6.63	1.26	1.36	1.04	0.98
		10	3.28	1.23	4.95	1.05	1.28	0.75	0.83
	Avg.50		4.09	1.62	6.42	1.18	1.35	0.96	0.93
	100	3	8.63	4.76	13.08	3.17	4.21	1.86	1.41
		5	8.04	3.90	11.51	3.07	4.09	1.51	1.42
		10	6.95	3.80	9.98	3.11	4.48	1.29	1.16
	Avg.100		7.87	4.15	11.52	3.12	4.26	1.55	1.33
	200	3	13.02	9.97	14.47	7.11	8.85	1.66	1.28
		5	13.69	11.16	16.37	7.84	10.28	1.52	1.04
		10	12.33	9.23	14.02	7.58	10.27	1.03	0.81
	Avg.200		13.01	10.12	14.95	7.51	9.80	1.40	1.04
	500	3	18.01	16.29	18.01	13.53	14.02	1.15	1.02
		5	20.76	19.60	20.91	16.37	17.47	1.42	1.40
		10	20.91	20.15	21.20	16.44	17.81	0.96	0.93
	Avg.500		19.89	18.68	20.04	15.44	16.43	1.18	1.12
60 Total			**7.94**	**5.90**	**9.37**	**4.64**	**5.47**	**0.93**	**0.88**

an adapted model minimizing makespan can be formulated. Based on these two formulations, the epsilon constraint method can be used to generate Pareto fronts that show the relationship between total flow time and the resource utilization represented by the makespan (see, for example, 34). For this, we first solve the MILP minimizing makespan, resulting in an objective function value C^*_{max}. Based on this solution, the MILP is solved again, minimizing total flow time including Eq. (13) and the additional constraint

$$C_{\max} \leq C^*_{\max}. \tag{14}$$

The resulting solution represents one extreme point of the Pareto front, i.e., the solution with minimal total flow time while ensuring optimal makespan. In the basic form of the epsilon constraint method, afterward, the model is solved again several times, iteratively increasing C^*_{max} in Eq. (14) by one or ϵ units. Whenever a new non-dominated solution is found, it is added to the Pareto front. The procedure terminates as soon as a solution with the optimal total flow time for the original model without Eq. (14) is found. Note that the sequence in which the objectives are considered may be changed without loss of generality. However, this choice influences the required number of iterations of the epsilon constraint method and the computation time. Although the epsilon method can be time-consuming, it provides valuable insights into the relationships between the considered objectives, thereby deepening the understanding of the problem.

Table 6
Average RPD values in % for (5-15 job sizes) instances.

π_{CPU}	n	m	GA	HSA	ABC	ILSw	IGw	ILS	IG
30	20	3	1.16	0.33	0.95	0.16	0.32	0.16	0.31
		5	1.01	0.22	0.87	0.12	0.28	0.13	0.35
		10	0.90	0.19	0.80	0.14	0.33	0.12	0.38
	Avg 20		1.02	0.25	0.87	0.14	0.31	0.14	0.35
	30	3	2.34	1.21	3.40	0.82	1.05	0.78	0.96
		5	2.38	0.96	3.09	0.77	0.94	0.67	0.88
		10	2.31	1.00	3.08	0.78	1.01	0.74	0.99
	Avg 30		2.34	1.06	3.19	0.79	1.00	0.73	0.94
	50	3	4.47	2.04	6.95	1.37	1.25	1.31	1.24
		5	4.13	1.81	6.78	1.16	1.47	1.15	1.32
		10	3.37	1.86	4.95	1.21	1.63	1.22	1.55
	Avg 50		3.99	1.90	6.23	1.25	1.45	1.23	1.37
	100	3	7.77	3.67	10.34	2.02	2.64	1.75	2.03
		5	7.79	4.44	9.94	2.58	3.33	2.11	2.22
		10	6.53	3.41	8.05	2.24	3.22	1.69	2.19
	Avg 100		7.37	3.84	9.44	2.28	3.07	1.85	2.15
	200	3	10.74	8.28	11.28	3.94	4.59	2.70	2.58
		5	9.48	7.25	9.85	4.24	5.19	2.62	2.56
		10	9.02	7.43	9.58	4.63	6.06	1.86	1.98
	Avg 200		9.74	7.65	10.24	4.27	5.28	2.39	2.37
	500	3	10.33	9.48	10.33	5.59	5.69	3.16	3.16
		5	12.22	11.42	12.22	7.82	8.24	2.75	2.79
		10	12.81	12.23	12.81	8.12	8.70	1.99	2.20
	Avg 500		11.78	11.04	11.78	7.18	7.54	2.63	2.72
30 Total			**6.04**	**4.29**	**6.96**	**2.65**	**3.11**	**1.49**	**1.65**
60	20	3	1.06	0.24	0.81	0.10	0.22	0.12	0.25
		5	0.85	0.16	0.76	0.09	0.22	0.07	0.27
		10	0.82	0.12	0.74	0.07	0.28	0.07	0.31
	Avg 20		0.91	0.17	0.77	0.09	0.24	0.09	0.27
	30	3	2.20	0.96	3.03	0.60	0.84	0.58	0.79
		5	2.23	0.73	2.61	0.57	0.79	0.54	0.71
		10	1.96	0.59	2.53	0.55	0.73	0.55	0.74
	Avg 30		2.13	0.76	2.72	0.57	0.79	0.56	0.74
	50	3	4.12	1.55	6.38	1.00	0.95	0.97	0.96
		5	3.75	1.49	6.25	0.86	1.17	0.85	0.98
		10	2.97	1.15	4.46	0.88	1.20	0.87	1.22
	Avg 50		3.61	1.40	5.70	0.91	1.11	0.90	1.05
	100	3	7.16	2.63	9.89	1.50	1.95	1.30	1.47
		5	7.34	3.45	9.60	2.04	2.73	1.61	1.76
		10	6.05	2.13	7.63	1.80	2.70	1.24	1.65
	Avg 100		6.85	2.74	9.04	1.78	2.46	1.38	1.63
	200	3	10.14	6.60	10.84	3.16	3.97	2.14	2.02
		5	9.09	5.75	9.45	3.48	4.42	2.07	1.73
		10	8.69	5.74	9.22	3.93	5.26	1.36	1.61
	Avg 200		9.31	6.03	9.83	3.52	4.55	1.86	1.79
	500	3	10.33	9.48	10.33	5.14	5.34	1.93	1.89
		5	12.09	11.41	12.14	7.30	7.77	1.65	1.69
		10	12.55	12.04	12.68	7.60	8.25	1.43	1.60
	Avg 500		11.65	10.98	11.71	6.68	7.12	1.67	1.73
60 Total			**5.74**	**3.68**	**6.63**	**2.26**	**2.71**	**1.07**	**1.20**

6.2. Pareto fronts for illustrative examples

Fig. 6 shows the resulting Pareto fronts for two exemplary instances *Example I* with 10 jobs and 5 machines 6(a), and *Example II* with 10 jobs and 10 machines 6(b), respectively. We observe a clear trade-off between the two objectives. Especially at the edges of the Pareto front, a small increase in one objective can lead to a significant improvement in the other objective. For example, the solution with the minimum total flow time in Example II is characterized by $TFT = 1890$ and $C_{max} = 288$. Compared to the fifth solution in the Pareto front ($TFT = 1910$; $C_{max} = 275$), this means that by allowing a deterioration of TFT by only 1.1%, C_{max} can be lowered by 4.5%. Likewise, comparing the two solutions with the lowest makespan ($TFT = 2110/C_{max} = 271$ and $TFT = 1930/C_{max} = 273$), a decrease of makespan by only 0.7% results in an improvement of total flow time by even 8.5%. Similar results can be observed for Example I. These disproportional change in objective values shows that considering only the objective total flow time may lead to unfavorable effects on resource utilization. However, it has to be noted that the trade-off between the two objectives is not so clear for all instances. While the presented instances are exemplary, for some instances, the Pareto front even consists of only a single solution minimizing both total flow time and makespan/machine utilization at the same time.

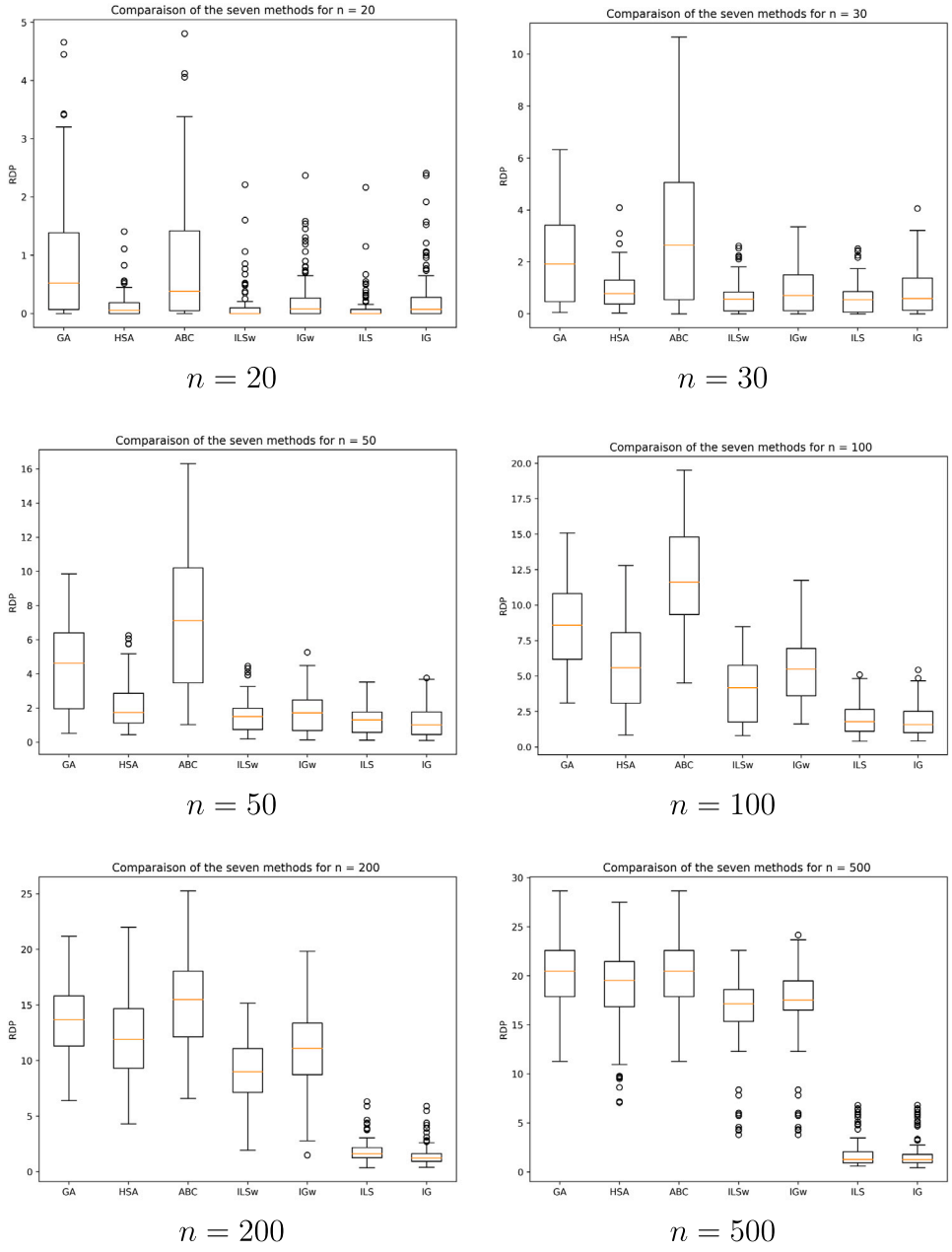


Fig. 4. Comparison of methods for different problem sizes and 1..20 job sizes.

For Example I, Figs. 7 and 8 show the Gantt charts and machine utilizations charts of the two extreme solutions with the minimum total flow time and minimum makespan, respectively. A cascading resource utilization can be observed that is typical for a flow shop problem. While Machine 1 is only used until time unit 47, Machine 5 only starts processing after 27 time units (see Fig. 7). This cascading effect makes it difficult to explicitly measure the average resource utilization. Nevertheless, it can be seen that in times when several jobs are processed in parallel on a resource, the average machine utilization is mostly above 50%. As expected, the chart is compressed, and machine utilization levels are on average even higher when minimizing makespan, see Fig. 8. At the same time, the machine utilization rarely attains 100% because the processed job sizes hardly ever add up to exactly meet the machine capacities. This clearly demonstrates that the cumulative flow shop scheduling problem extends beyond mere sequencing problems; in scheduling, the relationship between job sizes and machine capacities needs to be considered, drawing parallels to packing problems. Further, the charts illustrate and underline the trade-off between the minimization of total flow time and the maximization of resource utilization again.

6.3. Impact of machine capacities

To investigate how resource capacities influence total flow time, we conducted a sensitivity analysis by scaling resource capacities with a factor $f \in [1.2; 1.4; 1.6; 1.8; 2.0]$. Using $f \geq 1$ (i.e., only increasing the original capacities) guarantees feasibility, since a reduction in capacity could lead to cases in which a capacity might not be sufficient for processing the largest job size, i.e., $Q_i < \max_{\forall j} q_j$. For the sensitivity analysis, we employed the ILS algorithm for the instances with $n \in [20, 30, 50, 100]$ due to its good performance in our computational experiment. The computation time factor is set to $\pi_{CPU} = 30$. Table 7 shows the results of the sensitivity analysis. To ensure comparability, the reported average RPD values are calculated based on the TFT_{best} of

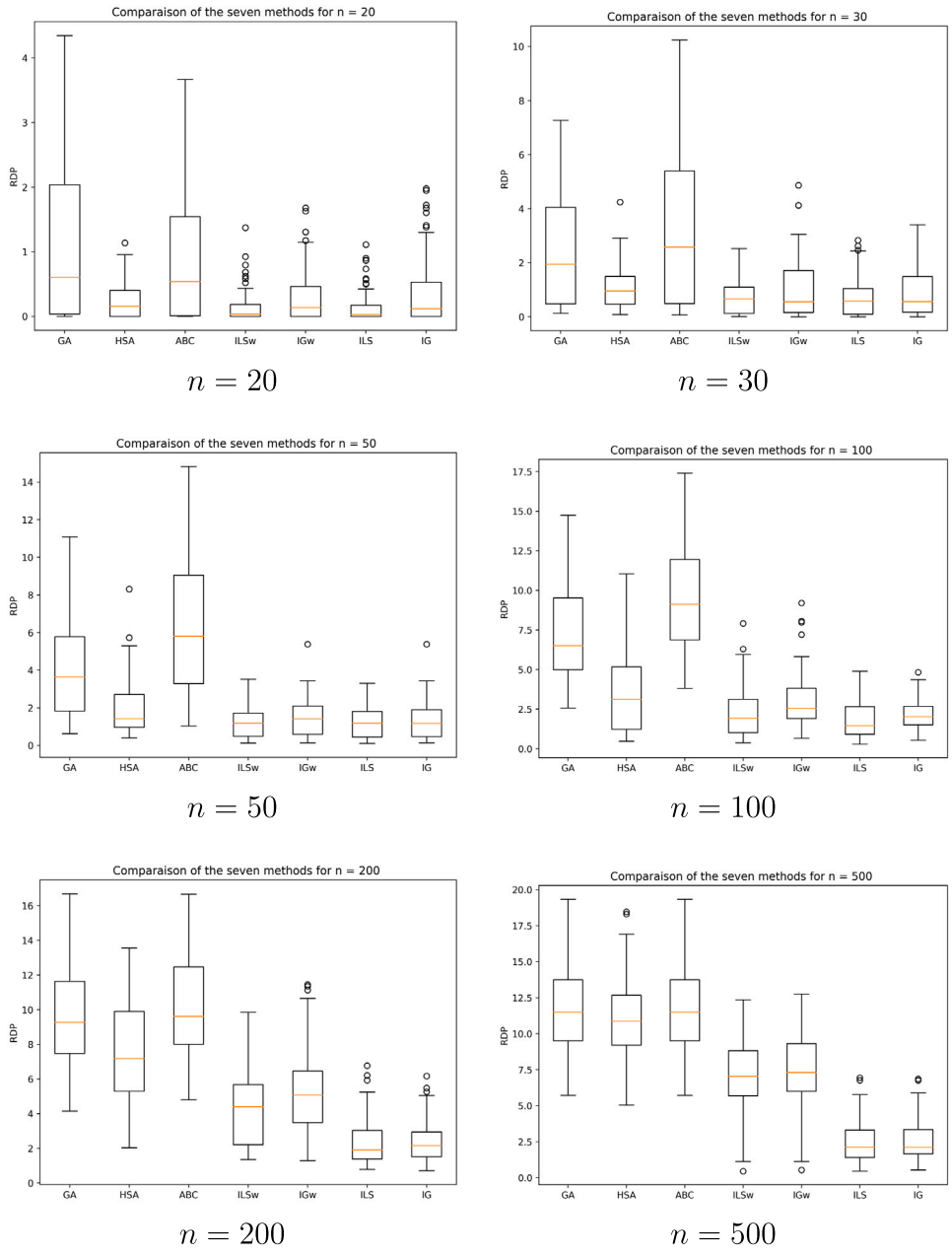


Fig. 5. Comparison of methods for different problem sizes and 5..15 job sizes.

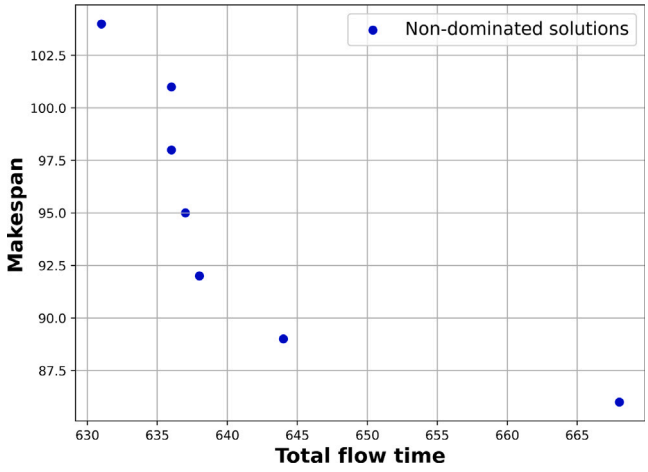
the best-known solution with unchanged capacity, i.e., $f = 1.0$. Hence, the values reported in the respective column equal those for ILS shown in Table 6, see Section 5.4.

The results in Table 7 demonstrate that a substantial reduction in total flow time can be achieved by increasing machine capacities. For instance, raising the capacity by 20% ($f = 1.2$) yields an average total flow time reduction of 8.05% compared to the best-known solutions at $f = 1.0$. As expected, larger capacity factors lead to greater improvements, reaching nearly 40% average relative percentage deviation (RPD) when capacities are doubled. However, the improvements exhibit diminishing marginal returns. While the initial increase from $f = 1.0$ to $f = 1.2$ results in an average total flow time reduction of approximately 9% (from RPD 0.96% to 8.05%), the increase from $f = 1.8$ to $f = 2.0$ yields only a 3.5% reduction (from RPD 21.46% to 24.00%). Overall, the relative improvement in total flow time is consistently smaller than the relative increase in machine capacity. For example, a 60% increase in capacity translates, on average, into less than a 20% reduction in total flow time. These findings highlight the distinctive interplay between sequencing decisions and machine capacity in the cumulative scheduling problem.

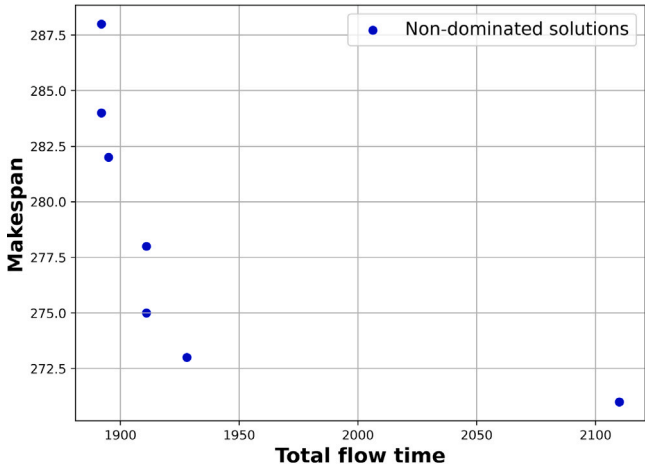
Furthermore, the results indicate that the benefit of additional capacity is more pronounced for instances with a larger number of jobs. At $f = 1.2$, the average total flow time reduction is 5.96% for instances with 20 jobs, but increases to 10.82% for instances with 100 jobs. Similar trends are observed for other values of f. This effect can be attributed to the higher likelihood, in larger job sets, of identifying tasks whose sizes fit well with the additional capacity, thereby enabling more efficient machine utilization. In contrast, increasing the number of machines diminishes the relative improvement. For example, at $f = 1.6$ and 50 jobs, the total flow time reduction is 23.8% with three machines, but decreases to 15.0% with ten machines. Thus, capacity increases are especially beneficial for instances with many jobs but relatively few machines.

Table 7
Average RPD values of ILS depending on the capacity factor f.

n	m	$f = 1.0$	1.2	1.4	1.6	1.8	2.0
20	3	0.14	−7.54	−12.19	−15.56	−17.93	−19.79
	5	0.13	−5.97	−9.81	−12.36	−14.37	−15.80
	10	0.12	−4.39	−7.13	−9.03	−10.39	−11.35
Avg. (n = 20)		**0.13**	**−5.96**	**−9.71**	**−12.32**	**−14.23**	**−15.65**
30	3	0.75	−8.68	−14.92	−19.17	−22.58	−25.20
	5	0.64	−7.20	−12.29	−15.76	−18.40	−20.43
	10	0.72	−4.74	−8.61	−10.96	−12.86	−14.40
Avg. (n = 30)		**0.71**	**−6.87**	**−11.94**	**−15.30**	**−17.95**	**−20.01**
50	3	1.28	−10.46	−18.33	−23.80	−28.20	−31.64
	5	1.09	−8.88	−15.68	−20.32	−23.95	−26.86
	10	1.14	−6.28	−11.72	−15.00	−17.68	−19.85
Avg. (n = 50)		**1.17**	**−8.54**	**−15.24**	**−19.71**	**−23.28**	**−26.11**
100	3	1.75	−12.92	−22.76	−29.75	−35.35	−39.81
	5	2.09	−10.68	−19.79	−25.92	−30.69	−34.66
	10	1.66	−8.85	−16.22	−21.11	−25.07	−28.24
Avg. (n = 100)		**1.84**	**−10.82**	**−19.59**	**−25.60**	**−30.37**	**−34.24**
Total Avg.		**0.96**	**−8.05**	**−14.12**	**−18.23**	**−21.46**	**−24.00**



(a) Example I: Pareto front for an instance with $n = 10$ and $m = 5$



(b) Example II: Pareto front for an instance with $n = 10$ and $m = 10$

Fig. 6. Exemplary Pareto fronts considering total flow time and makespan.

6.4. Managerial insights

The computational analysis provides several insights into the structural characteristics of the cumulative flow shop scheduling problem and its solution. While the solution methods can solve the problem effectively, several results indicate the relevance of paying attention to the specific parameters of an instance for its solution. While the IG algorithm performs well for large instances with high job size variance, the ILS is more robust and generally preferable for decision makers across problem sizes. The analysis confirms that the rescheduling procedure significantly improves solution quality. The differences highlight the importance of considering the variability of job sizes and the integration of problem-specific elements when selecting or designing scheduling approaches. The number of machines shows no substantial impact on the relative performance of the algorithms. From a managerial perspective, this suggests that scheduling policies can be applied consistently, independent of the number of machines available.

For the cumulative flow shop scheduling problem, results from the bi-objective analysis demonstrate that a sole focus on one objective, in our case, total flow time, can be misleading. Small deviations from the minimum flow time may yield disproportionately higher machine utilization. Hence, a joint perspective is desirable, and bi- or multi-objective approaches may support better managerial decisions.

The sensitivity analysis reveals that capacity increases yield clear benefits, but their extent depends on problem characteristics. Here, the number of machines has a relevant impact. Overall, the results suggest that investments in higher capacity deliver the highest returns in environments with many jobs and few machines. For the greenhouse application, this corresponds to settings with a high diversity of plant types. While capacity increases reduce total flow time, the relative improvements are smaller than the proportional rise in capacity. This implies that investments in capacity should be carefully evaluated against their cost, since beyond a certain point, additional resources provide only limited operational benefit.

7. Conclusion

In this study, we introduced the cumulative flow shop scheduling problem, characterized by machines that can process multiple jobs in parallel up to a given capacity. The problem is motivated by real-world applications such as greenhouse operations and represents a generalization of the classical flow shop scheduling problem. We proposed a mixed-integer linear programming formulation for minimizing total flow time and developed two metaheuristics, Iterated Local Search (ILS) and Iterated Greedy (IG), both enhanced with tailored components

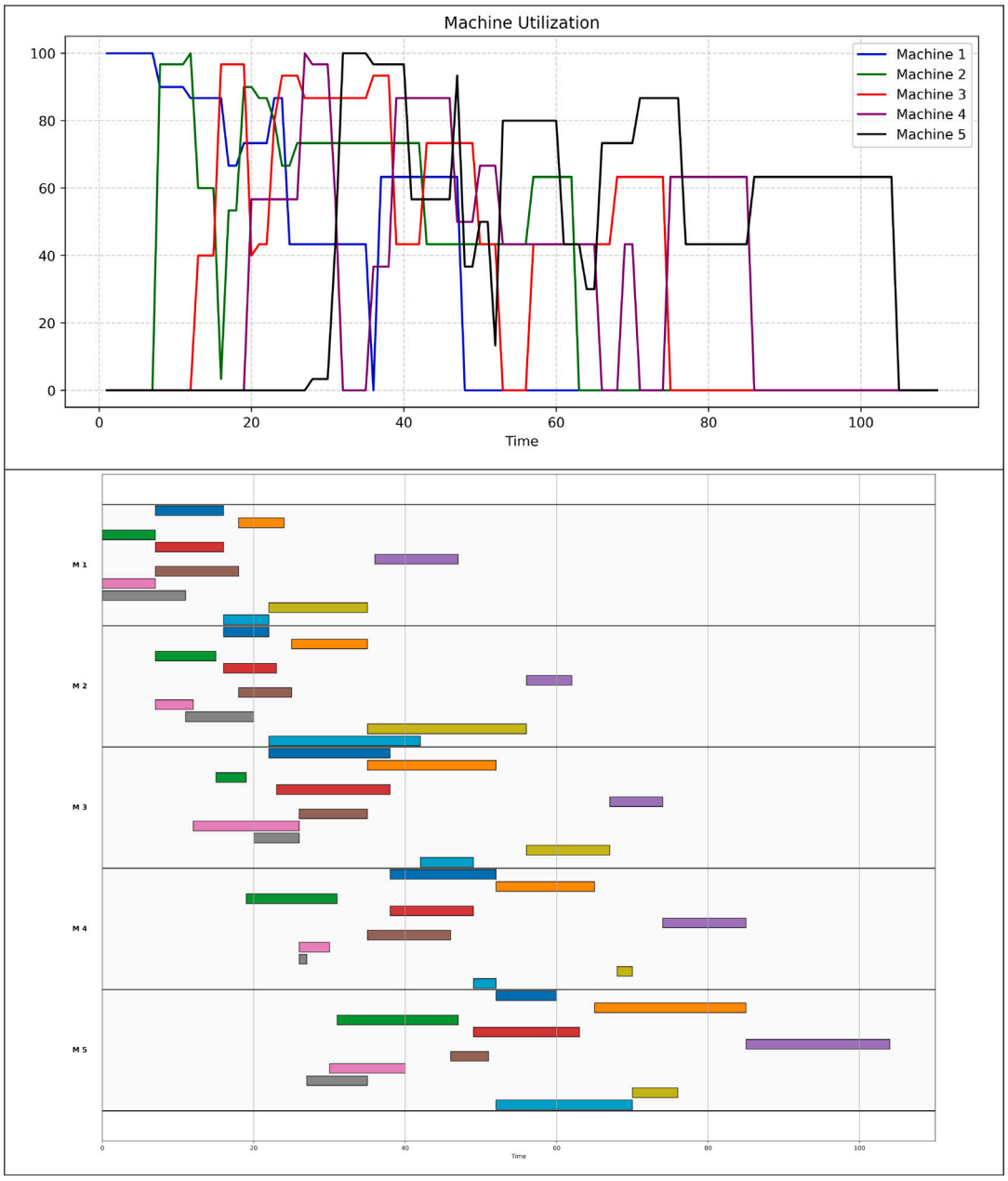


Fig. 7. Example II: Gantt chart and machine utilization for solution minimizing total flow time.

such as a rescheduling procedure. A comprehensive computational study demonstrated the effectiveness of these approaches and provided insights into the structural characteristics of the cumulative flow shop scheduling problem.

The results highlight that scheduling strategies should be adapted to specific problem settings. In particular, the distribution of job sizes plays an important role, although this factor has often been overlooked in the scheduling literature. Moreover, the interplay between total flow time and machine utilization reflects the distinct role of capacity usage in cumulative scheduling. Even small relaxations in flow time can yield disproportionately higher utilization, suggesting that a bi-objective perspective is more appropriate than focusing solely on flow time. In addition, the sensitivity analysis quantifies the impact of capacity changes on solution quality, thereby enabling a more informed cost-benefit assessment for cumulative scheduling environments.

Despite these contributions, several research questions remain open. A theoretical analysis of cumulative scheduling problems under diverse objectives is still lacking. In particular, similarities and differences with related problems, such as hybrid flow shops or flow shops with batch-processing machines, warrant further investigation. As cumulative scheduling integrates aspects of both scheduling and packing problems, a combination of solution techniques from both fields may also provide valuable insights. Furthermore, multi-objective approaches deserve greater attention, extending beyond total flow time and makespan to incorporate additional performance criteria. Finally, the proposed metaheuristics represent only a first step toward solving cumulative flow shop problems; matheuristics and decomposition-based approaches appear especially promising avenues for future research.

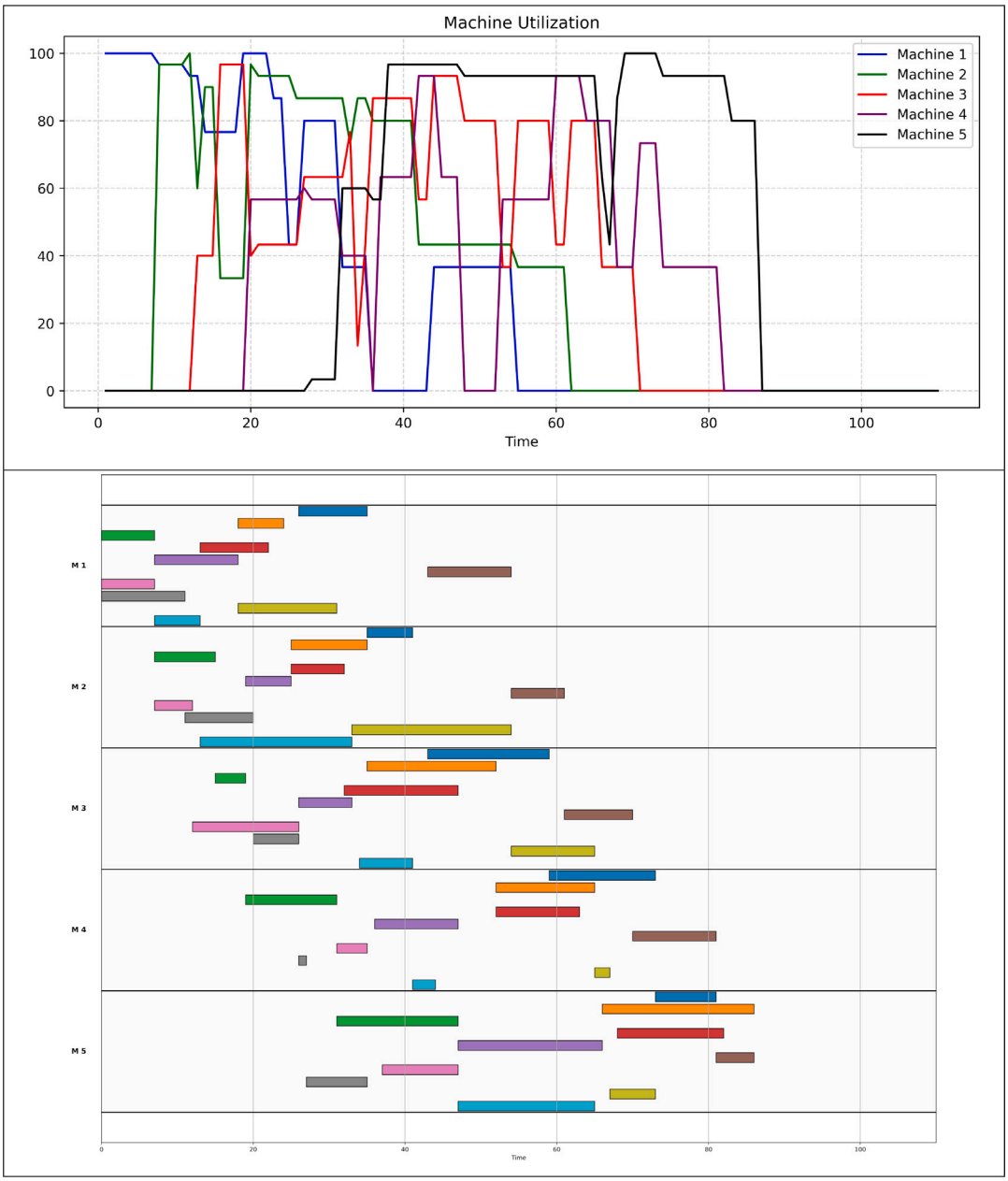


Fig. 8. Example II: Gantt chart and machine utilization for solution minimizing makespan.

CRediT authorship contribution statement

Ahmed Missaoui: Writing – review & editing, Writing – original draft, Visualization, Validation, Software, Methodology, Investigation, Formal analysis, Data curation. **Janis S. Neufeld:** Writing – review & editing, Writing – original draft, Validation, Supervision, Resources, Project administration, Methodology, Investigation, Formal analysis, Conceptualization. **Barry O'Sullivan:** Writing – review & editing, Supervision, Resources, Project administration, Methodology, Conceptualization.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Appendix A. Supplementary data

Supplementary material related to this article can be found online at https://doi.org/10.1016/j.orp.2026.100403.

Data availability

Data will be made available on request.

References

[1] Rossit DA, Tohmé F, Frutos M. The non-permutation flow-shop scheduling problem: a literature review. Omega 2018;77:143–53.
[2] Farahmand Rad S. Pixelation of time matrices for solving permutation flowshop scheduling problem. Int J Prod Res 2024;62(23):8505–17.
[3] De Snoo C, Van Wezel W, Jorna RJ. An empirical investigation of scheduling performance criteria. J Oper Manage 2011;29(3):181–93.

[4] Reza Hejazi S, Saghafian S. Flowshop-scheduling problems with makespan criterion: a review. Int J Prod Res 2005;43(14):2895–929.
[5] Linß F, Hewitt M, Neufeld JS, Buscher U. Order acceptance and scheduling in capacitated job shops. In: Voigt G, Fliedner M, Haase K, Brüggemann W, Hoberg K, Meissner J, editors. Operations Research Proceedings 2023. Cham: Springer Nature Switzerland; 2025, p. 341–7.
[6] Godard D, Laborie P, Nuijten W. Randomized large neighborhood search for cumulative scheduling. In: Proceedings of the fifteenth international conference on international conference on automated planning and scheduling. 2005, p. 81–9.
[7] Baptiste P, Le Pape C, Nuijten W. Satisfiability tests and time-bound adjustments for cumulative scheduling problems. Ann Oper Res 1999;92:305–33.
[8] Nuijten WP, Aarts EH. A computational study of constraint satisfaction for multiple capacitated job shop scheduling. European J Oper Res 1996;90(2):269–84.
[9] Carlier J, Néron E. A new LP-based lower bound for the cumulative scheduling problem. European J Oper Res 2000;127(2):363–82.
[10] Linß F, Hewitt M, Neufeld JS, Buscher U. Multi-objective optimization with order acceptance for the cumulative job shop scheduling problem in agribusiness. Omega 2026;141:103504.
[11] Hartmann S, Briskorn D. An updated survey of variants and extensions of the resource-constrained project scheduling problem. European J Oper Res 2022;297(1):1–14.
[12] Sánchez MG, Lalla-Ruiz E, Gil AF, Castro C, Voß S. Resource-constrained multi-project scheduling problem: A survey. European J Oper Res 2023;309(3):958–76.
[13] Ruiz R, Stützle T. A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem. European J Oper Res 2007;177(3):2033–49.
[14] Pan Q-K, Ruiz R. Local search methods for the flowshop scheduling problem with flowtime minimization. European J Oper Res 2012;222(1):31–43.
[15] Ramalhinho H, Stützle T. Iterated local search, iterated greedy and applications:. Top 2025;33(2):229–61.
[16] Lourenço HR, Martin OC, Stützle T. Iterated local search: Framework and applications. In: Handbook of metaheuristics. Springer; 2018, p. 129–68.
[17] Ribas I, Companys R, Tort-Martorell X. An iterated greedy algorithm for the flowshop scheduling problem with blocking. Omega 2011;39(3):293–301.
[18] Missaoui A, Ruiz R. A parameter-less iterated greedy method for the hybrid flowshop scheduling problem with setup times and due date windows. European J Oper Res 2022;303(1):99–113.
[19] Mecler D, Abu-Marrul V, Martinelli R, Hoff A. Iterated greedy algorithms for a complex parallel machine scheduling problem. European J Oper Res 2022;300(2):545–60.
[20] Gelvez SC, Dang TH, Letchford AN. MIP-based local search for permutation flowshop scheduling with makespan objective. Comput Oper Res 2026;107309.
[21] Du Z-S, Li J-Q, Song H-N, Gao K-Z, Xu Y, Li J-K, Zheng Z-X. Solving the permutation flow shop scheduling problem with sequence-dependent setup time via iterative greedy algorithm and imitation learning. Math Comput Simulation 2025;234:169–93.
[22] Yang Y-Y, Qian B, Li Z, Hu R, Wang L. Q-learning based hyper-heuristic with clustering strategy for combinatorial optimization: A case study on permutation flow-shop scheduling problem. Comput Oper Res 2025;173:106833.
[23] Li H, Gao K, Duan P-Y, Li J-Q, Zhang L. An improved artificial bee colony algorithm with Q-learning for solving permutation flow-shop scheduling problems. IEEE Trans Syst Man, Cybern: Syst 2022;53(5):2684–93.
[24] Han B, Pan Q-K, Gao L. A cooperative iterated greedy algorithm for the serial distributed permutation flowshop scheduling problem. Int J Prod Res 2024;62(12):4245–72.
[25] Zhao F, Xu Z, Wang L, Zhu N, Xu T, Jonrinaldi J. A population-based iterated greedy algorithm for distributed assembly no-wait flow-shop scheduling problem. IEEE Trans Ind Inform. 2022;19(5):6692–705.
[26] Rajendran C, Holthaus O. A comparative study of dispatching rules in dynamic flowshops and jobshops. European J Oper Res 1999;116(1):156–70.
[27] Jayamohan M, Rajendran C. New dispatching rules for shop scheduling: a step forward. Int J Prod Res 2000;38(3):563–86.
[28] Naderi B, Zandieh M, Balagh AKG, Roshanaei V. An improved simulated annealing for hybrid flowshops with sequence-dependent setup and transportation times to minimize total completion time and total tardiness. Expert Syst Appl 2009;36(6):9625–33.
[29] Vallada E, Ruiz R, Framinan JM. New hard benchmark for flowshop scheduling problems minimising makespan. European J Oper Res 2015;240(3):666–77.
[30] Liberti L. Reformulations in mathematical programming: automatic symmetry detection and exploitation. Math Program 2012;131(1):273–304.
[31] Lima RM, Novais AQ. Symmetry breaking in MILP formulations for unit commitment problems. Comput Chem Eng 2016;85:162–76.
[32] Ruiz R, Maroto C. A genetic algorithm for hybrid flowshops with sequence dependent setup times and machine eligibility. European J Oper Res 2006;169(3):781–800.
[33] Pan Q-K, Wang L, Mao K, Zhao J-H, Zhang M. An effective artificial bee colony algorithm for a real-world hybrid flowshop problem in steelmaking process. IEEE Trans Autom Sci Eng 2012;10(2):307–22.
[34] Schulz S, Buscher U, Shen L. Multi-objective hybrid flow shop scheduling with variable discrete production speed levels and time-of-use energy prices. J Bus Econ 2020;90(9):1315–43.