Self-adaptive Iterated Greedy Algorithm for Two-Stage Hybrid Flow Shop with a Cumulative Machine

1st Ahmed Missaoui
Insight Centre for Data Analytics
University College Cork
Cork, Ireland
ahmed.missaoui@insight-centre.org

2nd Janis S. Neufeld
Operations Management
Otto von Guericke University Magdeburg
Magdeburg, Germany
janis.neufeld@ovgu.de

3rd Barry O'Sullivan
School of Computer Science & IT
University College Cork
Cork, Ireland
b.osullivan@cs.ucc.ie

***Abstract*—This work addresses a realistic production system inspired by industry, involving a two-stage scheduling problem where a set of jobs is processed first on a group of parallel machines in the first stage and subsequently on a cumulative resource-constrained machine. A detailed mixed-integer linear programming (MILP) model is proposed to show the problem's constraints and objectives. Due to the model's computational complexity for larger instances, we also propose an efficient Self-adaptive iterated greedy (SAIG) algorithm to provide high-quality solutions in significantly reduced computation time. The proposed mathematical model and iterated greedy algorithm are evaluated on a benchmark set of more than 160 instances with varying job sizes and characteristics. Computational results are compared against a set of three well-selected metaheuristics from the literature. The results demonstrate that the MILP model yields optimal solutions for small instances, while our iterated greedy algorithm achieves near-optimal performance on larger instances with substantial time savings.**

***Index Terms*—Hybrid Flow Shop, Cumulative Machine, MILP Modeling, Iterated Greedy**

I. INTRODUCTION

Effectively responding to fluctuating market demands and global competition requires the design of production systems that not only minimize costs but also maximize the resource utilization rate. For the sustainability of companies, managers must make crucial decisions and adopt various strategies. In the short and medium term, scheduling emerges as one of the most effective approaches to ensure an efficient production system capable of reducing costs and increasing productivity.

In this context, each production system can be viewed as a scheduling problem, taking into account its specific characteristics and resource configuration. Previous studies have classified scheduling problems into three main categories:

- **Single-machine problems:** Involving a single resource used to process a group of jobs.
- **Single-stage multi-machine problems:** Involving a set of parallel machines used to process a set of jobs.
- **Multi-stage multi-machine problems:** Involving multiple production stages, each consisting of one or more resources. This category includes frameworks such as flow shops, hybrid flow shops, and job shops.

Among these, the third category represents practical production environments and closely reflects real-world industrial systems. The Hybrid Flow Shop (HFS) scheduling problem has attracted significant attention from researchers and academics due to its strong relevance to modern industrial practices [1]. An HFS system is characterized by multiple production stages, with at least one stage containing two or more parallel resources. Within this system, a set of jobs must be processed sequentially through all stages. In the HFS system, the sequencing of jobs and their allocation to available machines have a significant impact on reducing overall completion time. However, depending on the type of industry and the characteristics of the resources, some machines are capable of processing multiple tasks simultaneously with respect to a given capacity. Processing on this kind of machine is known in the literature as cumulative scheduling [2].

There is little study in the literature on cumulative scheduling, and only a few works have tackled this kind of machine as an independent production system [3]. It is clear that cumulative and batch scheduling share a common feature, the ability to process multiple jobs simultaneously. However, in cumulative machines, a job can be released as soon as its processing is complete, without depending on the completion of other jobs being processed on the same machine.

In real life, cumulative scheduling can be found in many domains. For example, in manufacturing, some production stations (such as ovens in steel-making or painting booths in automotive plants) are able to process multiple tasks simultaneously, subject to certain constraints like machine capacity or energy consumption limits [4]. These types of problems can be modelled as cumulative scheduling problems. In the agriculture sector, newly developed plants must be allocated to multiple greenhouses with different environmental characteristics and capacities. Each greenhouse can be considered as a single cumulative machine [5]. As illustrated, cumulative scheduling appears in many sectors. Depending on the underlying resource framework, various variants of the cumulative scheduling problem can be identified. Although cumulative scheduling arises in many real-world applications,

it has been less studied than classical HFS problems due to its increased modelling and computational complexity.

Summing up, this paper addresses a two-stage HFS problem with identical parallel machines in the first stage and a single cumulative machine in the second stage, with the objective of minimizing the makespan. We refer to this problem as CHFS. To the best of our knowledge, this specific configuration, involving a cumulative machine in the second stage, has not yet been studied in the existing literature.

The remainder of the paper is structured as follows. Section 2 summarizes prior research. Section 3 outlines the problem statement and the corresponding MILP formulation. Section 4 details the proposed solution methods. Section 5 provides the computational results. Finally, Concluding remarks and future research directions are presented in Section 6.

II. RELATED WORKS

For many decades, the HFS has been extensively studied from various perspectives and according to multiple performance criteria. To address these complex scheduling problems, numerous solution approaches have been developed, including mathematical models as well as heuristic and metaheuristic methods. In [6], the HFS problem with dedicated machines and delivery times is addressed with the objective of minimizing the makespan, using an improved simulated annealing (SA) algorithm. The two-stage HFS under no-wait constraints is investigated in [7], where the authors propose an improved genetic algorithm. Furthermore, [8] introduces an iterated local search algorithm to solve the two-stage HFS with no-wait constraints, focusing on makespan and energy consumption minimization.

A wide range of methods has been proposed in the literature to address HFS scheduling problems, including both iterative and population-based metaheuristics. Among these, the iterated greedy (IG) algorithm of [9] stands out as one of the most effective methods. In [10], an enhanced IG approach with an improved destruction–reconstruction operator was developed to address the HFS with blocking constraints. Similarly, [11] employed an improved IG algorithm incorporating a new initialization strategy to tackle the HFS with makespan minimization as the primary objective. More recently, [12] introduced a novel collaborative IG algorithm designed for HFS with batch processing machines, also focusing on the makespan criterion.

A common aspect of the aforementioned studies is that they all assume each machine is able to handle only one task at a time. However, this assumption does not always reflect real-world industrial settings, where a single machine may be able to process multiple tasks simultaneously without strict constraints on start or completion times. This more general and flexible framework is referred to as cumulative scheduling. Cumulative scheduling problems involve resources that can handle several tasks at the same time, provided that the total resource consumption does not exceed a predefined capacity. Despite their high practical relevance, these problems have received relatively limited attention in the scheduling literature. In [13], the cumulative job shop scheduling problem with release dates and deadlines is addressed, and a large neighborhood search approach is proposed for makespan minimization. In [14], the single cumulative machine scheduling problem is studied, and a novel linear programming formulation is introduced to solve it. The cumulative parallel machine scheduling problem is presented in [15], where the authors developed three deductive algorithms to adjust activity time bounds. In [16], the parallel cumulative scheduling problem with release dates and a makespan criterion is tackled using Jackson's pseudo-preemptive schedule (JPPS). Finally, [17] proposes an iterative relaxation method to address large-scale cumulative scheduling problems.

As can be observed, only a limited number of studies in the literature address cumulative scheduling. Especially (hybrid) flow shop environments have not been explored so far, despite their high practical relevance. Therefore, there is a clear opportunity to further explore and incorporate this aspect in scheduling research.

III. PROBLEM STATEMENT

A. Problem Definition

The scheduling problem under consideration can be formally described as follows: We are given a set of independent jobs that must be processed sequentially through two production stages. The first stage consists of m identical parallel machines, while the second stage is composed of a single cumulative machine with limited processing capacity.

Each job j is characterized by three parameters:

- p_{1j}: processing time required on a first-stage machine,
- p_{2j}: processing time required on the second-stage machine,
- s_j: job size, which represents the capacity requirement of the job during processing on the second-stage machine.

In the first stage, each machine can process at most one job at a time. In contrast, during the second stage, the cumulative machine is capable of processing multiple jobs simultaneously, provided that the sum of their sizes at any given time does not exceed the machine's total capacity C. Formally, for any time t

$$\sum_{j \in J_t} s_j \leq C,$$

where J_t denotes the set of jobs being processed at time t on the second-stage machine.

This capacity constraint introduces additional complexity, as it requires determining not only the sequence of jobs but also the optimal grouping of jobs in the second stage.

B. Mathematical model

Sets and Indices

- $J = \{1, \ldots, n\}$: Set of n jobs
- $P = \{1, \ldots, m\}$: Set of m parallel machines in Stage 1
- $T = \{0, \ldots, H-1\}$: Time horizon for cumulative scheduling

Parameters

- p_{j1}: Processing time of job j on Stage 1
- p_{j2}: Processing time of job j on Stage 2
- s_j: Size of job j on cumulative machine
- C: Capacity of cumulative machine
- H: Time horizon
- M: A large constant

Decision Variables

- $x_{jm} \in \{0,1\}$: 1 if job j is assigned to machine m in Stage 1
- $S_{j1} \geq 0$: Start time of job j in stage 1
- $C_{j1} \geq 0$: Completion time of job j in stage 1
- $S_{j2} \geq 0$: Start time of job j in stage 2
- $C_{j2} \geq 0$: Completion time of job j in stage 2
- $z_{jk} \in \{0,1\}$: Binary precedence variable (1 if j precedes k)
- $y_{jt} \in \{0,1\}$: 1 if job j starts at time t in stage 2
- $C_{\max} \geq 0$: Makespan

Objective

$$\min C_{\max} \tag{1}$$

Constraints

$$\sum_{m \in P} x_{jm} = 1 \qquad \forall j \in J \tag{2}$$

$$C_{j1} = S_{j1} + p_{j1} \qquad \forall j \in J \tag{3}$$

$$S_{j1} + p_{j1} \leq S_{k1} + M\Big(1 - (x_{jm} + x_{km} - 1) + (1 - z_{jk})\Big), \quad \forall j,k \in J,\ j < k,\ m \in P \tag{4}$$

$$S_{k1} + p_{k1} \leq S_{j1} + M\Big(1 - (x_{jm} + x_{km} - 1) + z_{jk}\Big), \quad \forall j,k \in J,\ j < k,\ m \in P \tag{5}$$

$$S_{j2} = \sum_{t=0}^{H-1} t\, y_{jt} \qquad \forall j \in J \tag{6}$$

$$C_{j2} = S_{j2} + p_{j2} \qquad \forall j \in J \tag{7}$$

$$\sum_{t=0}^{H-p_{j2}} y_{jt} = 1 \qquad \forall j \in J \tag{8}$$

$$S_{j2} \geq C_{j1} \qquad \forall j \in J \tag{9}$$

$$\sum_{j \in J} \sum_{\tau=\max(0,\, t-p_{j2}+1)}^{t} s_j\, y_{j\tau} \leq C, \quad \forall t \tag{10}$$

TABLE I
PROCESSING TIMES AND JOB SIZES

Job	Processing Times ($p_{j,i}$)		Sizes
	i = 1	i = 2	i = 2
1	3	3	5
2	2	3	4
3	4	3	3
4	3	4	7
5	4	2	2
6	3	4	5
7	5	2	2
8	2	3	5

$$C_{\max} \geq C_{j2} \quad \forall j \in J \tag{11}$$

The objective of the model is to minimize the overall makespan $C_{\max}$, as defined in Equation (1). Assignment constraints, given in Equation (2), ensure that each job j is assigned to exactly one of the parallel machines in Stage 1. The completion time of a job in Stage 1, C_{j1}, is defined in Equation (3) as its start time S_{j1} plus the processing time p_{j1}. Equations (4) and (5) prevent overlap between jobs assigned to the same machine.

In Stage 2, the start time S_{j2} of each job is determined by Equation (6), and the completion time C_{j2} is computed by adding the processing time p_{j2} to S_{j2}, as shown in Equation (7). Each job must start exactly once in Stage 2, as enforced by Equation (8), and it cannot begin until its Stage 1 processing has been completed, as stated in Equation (9). Equation (10) ensures that the cumulative machine's capacity is not exceeded at any time t by summing the resource consumption s_j of all jobs active at that time. Finally, the makespan $C_{\max}$ is formally defined in Equation (11).

C. Illustrative Example

To illustrate the problem, consider an example with 8 jobs, 3 machines in the first stage, and a cumulative machine with a total capacity of 10 in the second stage. All relevant job data are provided in Table I. Let us consider the sequence $\text{seq} = (1,2,3,4,5,6,7,8)$. Figure 1 shows the corresponding Gantt chart for this sequence, highlighting the allocation of jobs across the first-stage machines and the cumulative processing on the second stage machine.

IV. PROPOSED SELF-ADAPTIVE ITERATED GREEDY

The IG algorithm was initially introduced by [9] for solving permutation flowshop problems with the makespan criterion. Like most metaheuristic approaches, IG begins with an initial solution and iteratively improves it until a predefined stopping criterion is satisfied. During each iteration, three operators are applied successively with the aim of enhancing the objective function. The main steps of the algorithm are summarized in Algorithm 1. We refer to our proposal as SAIG because it automatically adapts the selection of the destruction and construction operators. For more details about the IG method, readers can refer to the literature review provided in [18]

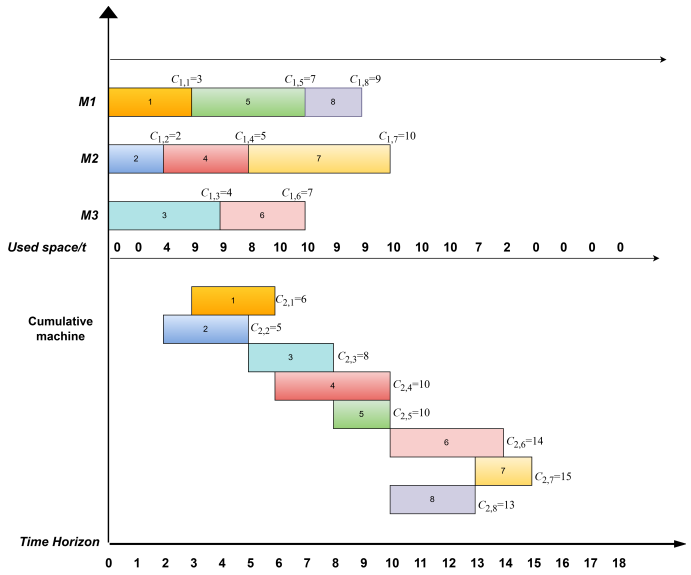

Fig. 1. Gantt chart

Algorithm 1 Self-adaptive Iterated Greedy

$S := GenerateInitialSolution()$
$S := LocalSearch(S)$
$S^* := S$
while *The termination criterion is not reached* **do**
  $S' := DestructionConstruction(S)$
  $S' := LocalSearch(S')$
  $S := AcceptanceCriterion(S, S')$
  if S is better than S^* **then**
    $S^* := S$
  end if
end while
Return S^*

A. Initial solution

In metaheuristics, the initial solution plays a crucial role in enhancing the overall effectiveness of the algorithm. The choice of heuristics is closely linked to the characteristics of the objective function under study. In this work, the NEH heuristic is employed to generate the initial solution for the IG algorithm. This heuristic has been shown to be highly effective in producing high-quality initial solutions in several scheduling problems [6]. A simple permutation of jobs is used as the solution representation. In the first stage, jobs are allocated one by one to the first available machine, while in the second stage, each job starts only when there is sufficient capacity to accommodate its entire processing time.

B. Local search

The local search (LS) operator is applied at two stages: initially, immediately after generating the initial solution, and subsequently, at each iteration following the destruction-reconstruction procedure. In the LS process, a randomly selected job is removed from the incumbent solution and systematically reinserted into all feasible positions within the sequence, from the beginning to the end. At each insertion, the makespan is evaluated. If an improvement is achieved, the search terminates, and the improved solution replaces the incumbent.

C. Destruction reconstruction

The destruction–reconstruction (DR) operator constitutes the core component of the IG algorithm. Its underlying principle is straightforward: during the destruction phase, a predefined number of jobs are removed from the current sequence. In the subsequent reconstruction phase, the removed jobs are reinserted one by one into all feasible positions within the partial sequence. For each job, the insertion position yielding the best makespan is selected, and this process is repeated until all removed jobs have been reinserted.

In this study, we adopt the auto-calibrated DR operator proposed by [19], which differs from the basic version in two main aspects: (i) the adaptive selection of the destruction size d, and (ii) the job insertion strategy. We first describe the mechanism for selecting d. Consistent with existing studies on IG algorithms for HFS problems, the parameter d was varied over the range $\{1, 2, 3, 4, 5\}$.

Initially, all candidate values of d have an equal probability of being selected. At each iteration, one value of d is randomly chosen, and the resulting makespan is compared with that of the incumbent solution. If an improvement is observed, the probability of selecting this value in future iterations is increased by adding an additional occurrence of d to the selection pool. This adaptive mechanism allows the algorithm to progressively favor the most promising values of d, thereby improving its search efficiency over time.

The insertion strategy also differs from the basic version. For each removed job, the insertion begins at the first possible position $i = 0$. The makespan obtained from this insertion is compared with that of the incumbent solution. If an improvement is observed, the process continues by inserting the job into the next position $i = i + 1$. Otherwise, the insertion position is advanced by two steps $i = i + 2$, effectively skipping one intermediate position. This adaptive insertion mechanism reduces the number of unnecessary evaluations, thereby increasing the number of iterations that can be performed within the same computational budget and improving the overall search efficiency.

D. Acceptance criterion

Inspired by the genetic algorithms in scheduling problem resolution, the tournament selection is used as an acceptance criterion in this work. The method relies on a list-based approach: at each iteration, a new solution is generated, which may either improve the current best sequence or be worse. If it improves the results, it becomes the new incumbent solution. Otherwise, the acceptance criterion determines which sequence should remain as the incumbent. All non-improving solutions are stored in a dynamic list R, whose maximum size equals the number of iterations. At each step, a tournament is

conducted by randomly selecting two individuals from R, and the best among them becomes the new incumbent. This simple approach balances intensification and diversification.

V. COMPUTATIONAL STUDY

A. Experimental settings

To evaluate the effectiveness of the proposed IG approach, a comprehensive benchmark was generated. The benchmark design considers several key parameters. First, the number of jobs was varied across five levels: $n \in \{20, 30, 50, 80, 100\}$. The job sizes s_j were uniformly distributed in $U[5, 20]$, while the processing times for both stages followed a uniform distribution $U[1, 50]$. The number of machines at the first stage was tested at three levels: $m \in \{3, 5, 8\}$. Finally, the capacity of the cumulative machine was considered at two levels: cap $\in \{20, 50\}$. Each scenario is repeated five times to get a total of 150 instances.

In addition, to evaluate the performance of the MILP model and compare it with the proposed IG approach, a smaller benchmark consisting of 18 instances was generated. In this case, the number of jobs was tested at four levels: $n \in \{10, 15, 20\}$, while all other parameters were kept the same as in the previous benchmark, with one replication per configuration.

This experimental design ensures a wide range of problem sizes and complexity, allowing for a rigorous assessment of the algorithm's performance.

To assess the performance of the developed algorithm, we compared its performance against three well-established metaheuristic methods from the scheduling literature. Specifically, we considered the Simulated Annealing (SA) algorithm of [6], the Iterated Local Search (ILS) algorithm, and the IG algorithm proposed by [19]. These three approaches were carefully adapted to address the characteristics of the studied scheduling problem.

To ensure a fair and robust comparison, all competing methods were parameterized through a systematic calibration process based on the Design of Experiments (DoE) methodology, considering the relative percentage deviation RPD = $(Cmax - Cmax_{best})$ / $Cmax_{best} \cdot 100$ as the response variable. This calibration allows each algorithm to operate at its best-performing configuration.

To ensure a fair comparison among the metaheuristics, we applied two stopping criteria based on the instance size, with a CPU parameter set to either 60 or 90. The stopping time for each algorithm was computed in milliseconds as

$$\text{Stopping Time} = \text{CPU} \times n \times 2,$$

The stopping times of 30 and 60 were chosen to represent two different computational budgets, enabling a fair comparison of the methods under both time-constrained and more relaxed conditions. For the MILP model, a time limit of 1 hour was set for each small instance.

The proposed SAIG algorithm, as well as all competing metaheuristic methods, were implemented in C++ using Microsoft Visual Studio 2022. The MILP model was implemented and solved using CPLEX 20.1.0. All computational experiments were carried out on a Windows-based laptop equipped with an Intel(R) processor and 16 GB of RAM.

TABLE II
AVERAGE RPD VALUES FOR DIFFERENT JOB SIZES AND METHODS

n	m	SA	ILS	IG	SAIG	MILP
10	3	0.23	0.08	**0.00**	**0.00**	**0.00**
	5	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**
	8	0.30	0.30	0.30	0.30	**0.00**
10 Avg		0.18	0.12	0.10	0.10	**0.00**
15	3	0.46	**0.00**	0.08	**0.00**	**0.00**
	5	1.30	**0.00**	**0.00**	**0.00**	**0.00**
	8	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**
15 Avg		0.59	**0.00**	0.03	**0.00**	**0.00**
20	3	1.50	0.35	0.23	**0.12**	8.66
	5	2.39	0.71	0.53	**0.09**	3.98
	8	1.23	**0.54**	0.58	0.71	6.82
20 Avg		1.71	0.53	0.45	**0.30**	6.49

B. Results

For the small benchmark, Table II presents the comparison of all resolution methods, including the MILP model, our proposed SAIG, and the three competing methods. It is clear that the MILP model is able to outperform all other methods for instances with $n = 10$ and $n = 15$, where it achieved an RPD value of 0.00. For the same instance sizes, the methods based on SAIG and LS also provided good results, while SA gave the worst results, with RPD values of 0.18 and 0.59 for $n = 10$ and $n = 15$, respectively. For the instances with 20 jobs, the proposed SAIG method achieved the best results, while the MILP model produced the worst RPD values for all m.

To test the efficiency of our proposed SAIG against the competing methods, we used the large benchmark. All results are given in Table III. For CPU = 60, our proposed SAIG yields the best RPD average value of 0.37, while the nearest value is given by ILS 0.58. Such as with the small benchmark, SA gave the worst results with an RPD = 1.81.

For the case with a CPU parameter of 90, the proposed SAIG method achieved the best results, with an RPD of 0.28 and an improvement of 31.58% compared to the nearest RPD value obtained by ILS. In contrast, the worst results were obtained by the SA algorithm.

When increasing the CPU parameter from 60 to 90, ILS improves its solution quality by an average of only 0.06. The RPDs of the other algorithms are further improved to nearly the same extent (0.09 or 0.10), despite SAIG starting from a better solution and, therefore, having less potential for further RPD reduction. This demonstrates that SAIG can utilize additional computational time more efficiently.

Finally, it can be observed that the SAIG method produces results superior to all competing methods across all CPU times. Moreover, the average RPD achieved by our proposed IG with a CPU time of 60 is better than the average RPD obtained by the competing methods, even with a CPU time of 90.

TABLE III
AVERAGE RPD VALUES FOR DIFFERENT JOB SIZES AND METHODS

CPU	n	m_1	SA	ILS	IG	SAIG
60	20	3	0.59	0.15	0.19	**0.08**
		5	2.06	0.45	0.64	**0.26**
		8	1.62	0.27	0.36	**0.25**
	20 avg		1.42	0.29	0.40	**0.20**
	30	3	0.34	**0.00**	**0.00**	**0.00**
		5	2.64	0.52	0.83	**0.31**
		8	2.69	1.04	1.18	**0.58**
	30 avg		1.89	0.52	0.67	**0.30**
	50	3	1.55	0.63	0.74	**0.28**
		5	1.83	0.58	0.68	**0.32**
		8	2.97	0.99	1.22	**0.56**
	50 avg		2.12	0.73	0.88	**0.38**
	80	3	1.20	0.49	0.53	**0.16**
		5	1.80	0.47	0.78	**0.41**
		8	2.03	0.65	0.85	**0.48**
	80 avg		1.68	0.54	0.72	**0.35**
	100	3	1.61	0.75	0.82	**0.40**
		5	2.08	0.87	1.01	**0.77**
		8	2.11	0.79	0.97	**0.63**
	100 avg		1.93	0.81	0.93	**0.60**
60 average			1.81	0.58	0.72	**0.37**
90	20	3	0.56	0.14	0.16	**0.08**
		5	1.94	0.37	0.54	**0.16**
		8	1.45	0.22	0.28	**0.25**
	20 avg		1.32	0.24	0.33	**0.16**
	30	3	0.29	**0.00**	**0.00**	**0.00**
		5	2.45	0.49	0.69	**0.21**
		8	2.67	0.92	1.02	**0.53**
	30 avg		1.80	0.47	0.57	**0.25**
	50	3	1.45	0.58	0.69	**0.22**
		5	1.75	0.50	0.58	**0.21**
		8	2.67	0.94	1.05	**0.48**
	50 avg		1.96	0.68	0.77	**0.30**
	80	3	1.15	0.43	0.47	**0.13**
		5	1.73	0.47	0.70	**0.30**
		8	1.92	0.60	0.73	**0.38**
	80 avg		1.60	0.50	0.63	**0.27**
	100	3	1.55	0.66	0.70	**0.28**
		5	2.04	0.70	0.94	**0.51**
		8	2.04	0.73	0.90	**0.42**
	100 avg		1.87	0.69	0.85	**0.40**
90 average			1.71	0.52	0.63	**0.28**
Total average			1.83	0.57	0.74	**0.39**

VI. CONCLUSION

In this work, we studied a new variant of the hybrid flowshop scheduling problem by considering a cumulative machine in the second stage. This problem is relevant to many practical settings in real production shops and, to the best of our knowledge, has not been studied before. A mixed-integer linear programming model was developed for the problem with the objective of minimizing the makespan. Furthermore, we proposed a new self-adaptive iterated greedy approach. To demonstrate the effectiveness of our proposal, we compared its results with those of three well-selected methods from the literature using two benchmarks comprising more than 160 instances. The results show that our proposed iterated greedy method outperforms all competing approaches.

For future work, considering additional practical constraints such as no-wait or setup times appears promising. Moreover, the developed methods could be adapted for the resolution of other scheduling problems.

ACKNOWLEDGMENT

This publication was conducted with the financial support of Science Foundation Ireland under Grant number 12/RC/2289-P2 at Insight the SFI Research Centre for Data Analytics at UCC and 22/NCF/DR/11264, National Challenge Fund, Digital for Resilience Challenge.

REFERENCES

[1] J. S. Neufeld, S. Schulz, and U. Buscher, "A systematic review of multi-objective hybrid flow shop scheduling," *European Journal of Operational Research*, vol. 309, no. 1, pp. 1–23, 2023.

[2] Y. Caseau and F. Laburthe, "Cumulative scheduling with task intervals." in *JICSLP*, 1996, pp. 363–377.

[3] P. Baptiste, C. Le Pape, and W. Nuijten, "Satisfiability tests and time-bound adjustmentsfor cumulative scheduling problems," *Annals of Operations research*, vol. 92, no. 0, pp. 305–333, 1999.

[4] M.-L. Lackner, C. Mrkvicka, N. Musliu, D. Walkiewicz, and F. Winter, "Exact methods for the oven scheduling problem," *Constraints*, vol. 28, no. 2, pp. 320–361, 2023.

[5] F. Linß, M. Hewitt, J. S. Neufeld, and U. Buscher, "Multi-objective optimization with order acceptance for the cumulative job shop scheduling problem in agribusiness," *Available at SSRN 5210102*, 2025.

[6] M. K. Hajji, O. Hamlaoui, and H. Hadda, "A simulated annealing metaheuristic approach to hybrid flow shop scheduling problem," *Advances in Industrial and Manufacturing Engineering*, vol. 9, p. 100144, 2024.

[7] S. Wang and M. Liu, "A genetic algorithm for two-stage no-wait hybrid flow shop scheduling problem," *Computers & Operations Research*, vol. 40, no. 4, pp. 1064–1075, 2013.

[8] A. Missaoui and B. O'Sullivan, "Multi-objective energy-efficient scheduling in two-stage hybrid flowshop under consideration of no-wait," in *International Conference on Industrial, Engineering and Other Applications of Applied Intelligent Systems*. Springer, 2025, pp. 464–475.

[9] R. Ruiz and T. Stützle, "A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem," *European journal of operational research*, vol. 177, no. 3, pp. 2033–2049, 2007.

[10] A. Missaoui and Y. Boujelbene, "An effective iterated greedy algorithm for blocking hybrid flow shop problem with due date window," *RAIRO-Operations Research*, vol. 55, no. 3, pp. 1603–1616, 2021.

[11] C. Lu, J. Zheng, L. Yin, and R. Wang, "An improved iterated greedy algorithm for the distributed hybrid flowshop scheduling problem," *Engineering Optimization*, vol. 56, no. 5, pp. 792–810, 2024.

[12] C. Li, Y. Han, B. Zhang, Y. Wang, J. Li, and K. Gao, "A novel collaborative iterative greedy algorithm for hybrid flowshop scheduling problem with batch processing machines and variable sublots," *International Journal of Production Research*, vol. 62, no. 11, pp. 4076–4096, 2024.

[13] D. Godard, P. Laborie, and W. Nuijten, "Randomized large neighborhood search for cumulative scheduling." in *ICAPS*, vol. 5, 2005, pp. 81–89.

[14] J. Carlier and E. Néron, "A new LP-based lower bound for the cumulative scheduling problem," *European Journal of Operational Research*, vol. 127, no. 2, pp. 363–382, 2000.

[15] P. Baptiste, C. Le Pape, and W. Nuijten, "Satisfiability tests and time-bound adjustmentsfor cumulative scheduling problems," *Annals of Operations Research*, vol. 92, no. 0, pp. 305–333, 1999.

[16] J. Carlier and E. Pinson, "Jackson's pseudo-preemptive schedule and cumulative scheduling problems," *Discrete Applied Mathematics*, vol. 145, no. 1, pp. 80–94, 2004.

[17] L. Michel and P. Van Hentenryck, "Iterative relaxations for iterative flattening in cumulative scheduling." in *ICAPS*, vol. 4, 2004, pp. 200–208.

[18] A. Missaoui, C. Ozturk, and B. O'Sullivan, "Iterated greedy algorithms for combinatorial optimization: a systematic literature review," in *2023 20th ACS/IEEE International Conference on Computer Systems and Applications (AICCSA)*. IEEE, 2023, pp. 1–7.

[19] A. Missaoui and R. Ruiz, "A parameter-less iterated greedy method for the hybrid flowshop scheduling problem with setup times and due date windows," *European Journal of Operational Research*, vol. 303, no. 1, pp. 99–113, 2022.