

Near-Optimal Dynamic Scheduling for Complex Manufacturing: A Constraint-Based Method with Real-Time Adaptation

Vincent Boyer^{1*}, M. Angélica Salazar-Aguilar²

¹ Tecnológico de Monterrey,
School of Engineering and Sciences, Monterrey,
Mexico

² Universidad Autónoma de Nuevo León,
Facultad de Ingeniería Mecánica y Eléctrica, San Nicolás de los Garza,
Mexico

vincent.boyer@tec.mx, maria.salazaragl@uanl.edu.mx

Abstract. This work addresses Dynamic Scheduling with Real-Time Order Release in Complex Manufacturing Environments. Our approach employs a constraint programming model to dynamically integrate newly released orders into existing schedules without modifying the deadlines of previously scheduled tasks. An order comprising a set of jobs is scheduled to ensure its earliest possible delivery. Our approach consists of a backtracking heuristic that initially constructs a feasible solution, which is subsequently refined using a constraint programming solver. We developed a simulator to evaluate our method and adapted static instances from the literature of the generalized flexible job shop scheduling problem for testing. Our results demonstrate that within one minute, our algorithm achieves near-optimal solutions every time a new order is released. We also assess the quality of our solutions against a deterministic approach, where job release times are known in advance. This comparison highlights the efficacy of our dynamic scheduling method in complex, constraint-rich manufacturing settings.

Keywords. Dynamic flexible job-shop scheduling problem, constraint programming, time lag constraints, machine capacity, holding times.

1 Introduction

Production scheduling refers to planning, organizing, and controlling the sequence and timing of operations in a manufacturing environment to ensure efficient use of resources. In operations

research, various scheduling problems have been extensively studied due to their critical role in optimizing both tactical and strategic decisions. In make-to-order (MTO) environments, the complexity of scheduling increases as new orders arrive dynamically, requiring constant updates to the production schedule. Scheduling problems are fundamental for improving operational control and solving strategic optimization challenges, including equipment selection, delivery date setting, and process planning [6]. In dynamic production environments, the arrival of new orders complicates scheduling further, requiring decision-makers to adapt schedules to maintain continuous efficiency.

The literature on production scheduling spans a broad range of problem types, from single-machine environments to more complex flexible job shop scheduling problems (FJSP). For instance, [15] studied a single-machine scheduling problem with unavailability constraints and proposed a mixed-integer linear programming (MILP) model and a variable neighborhood search (VNS) heuristic to minimize earliness and tardiness. Similarly, [13] developed a time window periodic maintenance strategy for a single-machine environment, using MILP and ant colony optimization (ACO) to minimize tardy jobs.

In flow shop environments, [8] explored scheduling with urgent jobs and limited waiting times,

minimizing tardiness and makespan through simulated annealing (SA) and iterated greedy (IG) algorithms. In heterogeneous parallel machine scheduling, [2] proposed MILP, SA, and VNS models to minimize tardiness, with SA outperforming existing methods (BKRG-DE [9], ABC-TS [10], and IG [1]). For flow shops with waiting time constraints, [7] addressed scheduling in semiconductor manufacturing, minimizing total tardiness using MILP and heuristic approaches such as SA and IG.

The flexible job shop scheduling problem (FJSP), which allows multiple machines to process the same operation, represents a significant extension of classical job shop scheduling. It is an NP-hard combinatorial optimization problem with wide applicability in real-world manufacturing. [17] introduced a constraint programming (CP) formulation to solve the FJSP, achieving new lower and upper bounds. Recently, some studies have applied deep reinforcement learning (DRL) to address both the classical FJSP [16] and dynamic FJSPs, as shown in [11] and [4]. For a detailed review of FJSP variants and solution methods developed over the past 30 years, we refer the reader to the survey by [6].

In more complex environments, the Generalized Flexible Job Shop Scheduling Problem (GFJSP) further extends the FJSP by introducing additional complexities such as machine capacity constraints, setup times, holding times, and time lags [3]. Traditional methods for solving GFJSP, such as the CP approach of [3], have been shown to outperform mixed integer linear programming (MILP) models in static settings.

Dynamic job shop scheduling, as classified by [12], can involve various types of events, including workpiece-related, machine-related, and process-related events. The framework proposed in this paper specifically addresses workpiece-related events, focusing on the dynamic arrival of new orders in MTO environments.

As each new order consists of multiple jobs, the scheduling process must be constantly refined to integrate the new orders without disrupting the current production schedule.

In addition, [14] discusses manufacturing scheduling approaches within the Industry 4.0

framework, while [5] reviews the role of smart scheduling in the generation and adaptation of schedules in response to dynamic events.

In practice, production scheduling operates on a rolling horizon, where schedules are updated as new jobs arrive. Incorporating these dynamic elements into the scheduling process is crucial for maintaining efficiency in MTO environments, where timely completion of orders is critical to meeting customer demands.

The dynamic GFJSP is particularly relevant for industries where complex production processes, tight deadlines, and fluctuating demand require adaptable scheduling solutions.

Therefore, this work proposes a framework to solve a dynamic GFJSP, which begins with a backtracking heuristic to generate an initial feasible schedule as a new order arrives.

Then, a constraint programming model is used to refine the schedule by integrating the new jobs and re-optimizing the overall production plan. The objective is to minimize the makespan of the new order by keeping the deadlines of undergoing orders.

The proposed framework contributes to the existing FJSP literature by providing a flexible approach to dynamic GFJSP that can quickly respond to the arrival of new orders without altering the deadlines of previously scheduled jobs.

This adaptability is essential in MTO environments that require scheduling tools that can dynamically adapt to the arrival of new jobs. The proposed framework addresses this need by minimizing the makespan across all orders and improving resource allocation through heuristic and constraint programming methods.

We organize the remainder of this paper as follows. In Section 2, we provide a general overview of the GFJSP and introduce a constraint programming (CP) model. In Section 3, we describe the proposed approach in detail. Section 4 presents the computational experiments and analyzes the performance of the approach. Section 5 discusses the findings and the algorithm effectiveness in dynamic settings. Finally, Section 6 draws conclusions and outlines future research.

2 Problem Definition

We consider a set of jobs J , a set of operations O , and a set of machines M . Each job $j \in J$ consists of an ordered set of operations $O_j \subset O$, which must be processed in sequence on a set of compatible machines. Jobs are grouped into orders or production requests R , which are released at different times within the planning horizon H . The release time of an order $r \in R$ is denoted by t_r , and is unknown in advance.

Each operation $o \in O$ can be processed by a subset of compatible machines $M_o \subseteq M$. Operations within the same job are subject to precedence constraints: An operation o can only start after the completion of its predecessor operation o' . Furthermore, time lag constraints impose a minimum time $\underline{l}_{oo'}$ and a maximum time $\bar{l}_{oo'}$ between the completion of o' and the start of o . These time lags can lead to negative cycles in the scheduling graph, which complicates finding feasible solutions.

Each machine $m \in M$ has a load capacity c_m that limits the number of operations it can process simultaneously. For any operation o processed on a machine m , there is an associated weight w_{om} representing the load imposed on the machine, as well as a processing time p_{om} . Furthermore, the setup time $a_{oo'm}$ represents the time required to prepare machine m to process operation o after operation o' . A maximum holding time h_{om} defines how long an operation can remain on a machine after completion.

For example, if a machine m can process only one operation at a time, its load capacity is $c_m = 1$, and the weight $w_{om} = 1$ for all operations o . However, for machines such as conveyor belts that can process multiple operations simultaneously, $c_m > 1$ and w_{om} may exceed 1.

The problem we address involves dynamically assigning jobs from newly released requests, such that the completion time of the last operation is minimized. Operations scheduled prior to the release of a new request cannot be modified, and the completion times of previous requests must remain unaffected.

We propose a Constraint Programming (CP) model to integrate new requests into the current

schedule. The model accounts for precedence constraints, time lags, machine capacity, setup times, and holding times. Let r denote the new request, and O represent the set of operations in r . The set of previously scheduled requests is denoted by R^* , with O^* representing their operations. For operations $o \in O^*$ from requests $r^* \in R^*$, we denote by s_o^* the start time of the operation o , by c_o^* the completion time, and by m_o^* the assigned machine of operation o derived from the current schedule. For these operations, the deadlines d_o^* are known in advance and cannot be modified.

Our decision variables are as follows:

- s_o : start time of operation o .
- c_o : completion time of operation o .
- y_{om} : binary variable, equal to 1 if operation o is processed on machine m .

The objective is to minimize the completion time of the last operation in the request r . The model is formulated as follows:

$$\min \max_{o \in O} c_o, \quad (1)$$

s.t.

$$(y_{om} = 1) \Rightarrow (p_{om} \leq c_o - s_o \leq p_{om} + h_{om}) \quad \forall o \in O \cup O^*, \quad (2)$$

$$\begin{aligned} s(y_{om} = 1 \wedge y_{o'm} = 1) \Rightarrow \\ (s_o \geq c_{o'} + a_{o'o'm} \vee s_{o'} \geq c_o + a_{oo'm}) \quad \forall o, o' \in O \cup O^*, \end{aligned} \quad (3)$$

$$\underline{l}_{oo'} \leq s_o - c_{o'} \leq \bar{l}_{oo'} \quad \forall o, o' \in O \cup O^*, \quad (4)$$

$$\sum_{m \in M_o} y_{om} = 1 \quad \forall o \in O \cup O^*, \quad (5)$$

$$\sum_{o \in O \cup O^*} w_{om} y_{om} \mathcal{U}(s_o, c_o) \leq c_m \quad \forall m \in M, \quad (6)$$

$$(s_o^* \leq t_r) \Rightarrow (s_o = s_o^* \wedge y_{m_o^*} = 1) \quad \forall o \in O^*, \quad (7)$$

$$(c_o^* \leq t_r) \Rightarrow (c_o = c_o^*) \quad \forall o \in O^*, \quad (8)$$

$$c_o \leq d_o^* \quad \forall o \in O^*, \quad (9)$$

$$s_o \geq t_r \quad \forall o \in O, \quad (10)$$

$$s_o \geq 0 \quad \forall o \in O \cup O^*, \quad (11)$$

$$c_o \geq 0 \quad \forall o \in O \cup O^*, \quad (12)$$

$$y_{om} \in \{0, 1\} \quad \forall o \in O \cup O^*, m \in M. \quad (13)$$

The objective function (1) minimizes the completion time of the last operation in the request r . Constraints (2) ensure that the processing times and holding times are respected. Constraints (3) enforce setup times between successive operations on the same machine. Time lag constraints (4) guarantee that precedence relationships are respected, with appropriate time intervals between operations. Notice that when the operations are from different jobs, the minimum and maximum time lag constraints are set to $-\infty$ and $+\infty$, respectively. Constraints (5) ensure that each operation is assigned to one machine. Constraints (6) ensure machine capacity is not exceeded, where $\mathcal{U}(a, b)$ is a step function that is equal to 1 in the interval $[a, b]$ and 0 otherwise. Constraints (7) and (8) ensure that previously scheduled operations are not altered. Constraints (9) enforce the pre-set deadlines of previously scheduled operations. Finally, constraints (10) enforce that operations from the new request cannot start before the release time t_r , and constraints (11-13) define the nature of the decision variables.

3 Materials and Methods

Our approach consists of a Backtracking Heuristic (BH), which initially generates a feasible solution and then refines it using a Constraint Programming (CP) model. This method enables quickly integrating new orders into the existing schedule without altering the deadlines of previously scheduled tasks.

The purpose of the heuristic is to rapidly provide an initial solution for the CP solver, which then optimizes the schedule to minimize the makespan of the new order. This two-stage approach not only ensures the generation of a feasible solution but also accelerates the convergence of the CP solver towards an optimal or near-optimal solution.

The BH algorithm iteratively assigns operations to machines while respecting precedence and time lag constraints, machine capacities, setup times, and holding times. During the assignment process, the algorithm may encounter situations where it

cannot assign an operation to a machine without violating maximum time lag constraints, which are detected by identifying negative cycles in the scheduling graph.

In such cases, the algorithm backtracks, reverting to a previous operation and attempting a different assignment. To avoid redundant searches, the algorithm maintains a list of forbidden assignment sequences that lead to negative cycles.

By only appending operations at the end of machine sequences, we limit the search space, thus reducing the number of potential assignments. Additionally, the operations assignments from previously scheduled requests remain fixed. The key steps of the BH algorithm are outlined in Algorithm 1.

Once the BH algorithm provides a feasible schedule, the CP model is employed to refine it. The CP solver optimizes the schedule to ensure the earliest possible delivery of the new order. While the BH algorithm may not yield an optimal solution, it offers a feasible starting point, significantly improving the CP solver's efficiency in reaching high-quality solutions. To assess the performance of our approach, we developed a simulator that generates random releases of new orders within a planning horizon. We use benchmark instances from the literature, which include static instances of up to 150 jobs, released at the start of the planning horizon.

The simulator groups jobs into requests and releases them randomly during the horizon. Each time a new request is released, our approach is used to incorporate it into the ongoing schedule. The planning horizon is set to one week, reflecting real-world data from a manufacturing company. Indeed, the makespan of the instances with 150 jobs is over one week when all jobs are released at the start. This allows us to evaluate the performance of our approach in dynamic environments of varying complexity, with the largest instances representing high production demand. We compare our approach against a deterministic method, where all job release times are known in advance. The deterministic method uses the same CP model but with fixed release times and deadlines for all jobs.

Algorithm 1 Backtracking Heuristic

```

1: Initialize Current Schedule  $S$ , New Request  $r$ ,
   Release Time  $t_r$ 
2: Create Updated Schedule  $S'$  by adding  $r$ 
3: Set  $O$  to the operations of  $r$ 
4: Set  $O^*$  to the operations of previously
   scheduled requests in  $S$ 
5: Set Forbidden Assignment List  $F$ 
6: Set  $S' = S$ 
7: for  $o \in O^*$  do
8:   Fix start time  $s_o$  and machine  $m_o$  for  $o$  if
      $s_o \leq t_r$ 
9:   Fix completion time  $c_o$  for  $o$  if  $c_o \leq t_r$ 
10:  Fix deadline  $d_o$  for  $o$ 
11: end for
12: while  $O \neq \emptyset$  do
13:   Identify the set of candidate operations  $C$ 
     that can be assigned to a machine
14:   Remove operations in  $C$  that yield a
     forbidden assignment in  $F$ 
15:   Select the operation/machine pair  $(o, m) \in$ 
      $C \times M$  with the earliest release time
16:   if no suitable operation/machine pair is
     found then
17:     Add current assignment sequence to  $F$ 
18:     Backtrack to the previously assigned
     operation
19:     Continue to the next iteration
20:   end if
21:   Assign selected operation  $o$  to machine  $m$ 
     in  $S'$ 
22:   if  $S'$  is not feasible then
23:     Add the current assignment sequence
     to  $F$ 
24:     Remove the selected operation  $o$  and its
     predecessors
25:     Add all removed operations back to  $O$ 
26:   else
27:     Update operations list:  $O = O \setminus \{o\}$ 
28:   end if
29: end while
30: Return Updated Schedule  $S'$ 

```

The goal is to maximize total earliness, which is the difference between job completion times and deadlines. This provides a benchmark to assess the quality of our dynamic scheduling solutions.

4 Results

We implemented our approach in C++ using the IBM ILOG CPLEX Optimization Studio. The experiments were conducted on a workstation with an Intel Core i7-11850H CPU and 32 GB of RAM. The BH algorithm and the CP model were developed as standalone programs that generate schedules for new requests, which are generated randomly by a simulator. This simulator creates new requests at random times within the planning horizon and evaluates the performance of our approach by comparing it to a deterministic method.

4.1 Benchmark Instances

Table 1 describes the benchmark instances used for testing. The column $\#_Inst$ indicates the number of cases for each class, $\#_Jobs$ denotes the number of jobs, $\#_Request$ represents the average number of requests generated, and $\#_Operations$ shows the average number of operations. The instances are classified based on the number of jobs, ranging from 10 to 150.

4.2 Performance of the BH Algorithm

Table 2 presents the results of the BH algorithm, showing the CPU time in seconds required to generate a schedule for a new request and the percentage improvement in fitness achieved by the CP model. The percentage improvement is calculated as the difference between the fitness of the CP model and that of the BH algorithm, divided by the fitness of the CP model.

The results demonstrate that the BH algorithm efficiently generates schedules for new requests within a short time frame. The CPU time required to create a schedule is consistently less than two seconds for all instances, with a maximum CPU time of 1.45 seconds for the *gfjsp_150* class. The percentage improvement in fitness achieved by the CP model is noteworthy, with a minimum of 0.00% and a maximum of 63.27%. However, the average percentage improvement is relatively modest, indicating that the BH algorithm effectively provides a good feasible starting point for the CP solver.

Table 1. Description of the benchmark Instances

Class	#_Inst	#_Jobs	#_Request	#_Operations
gfjsp_10	20	10	6.75	446.95
gfjsp_25	20	25	13.90	204.60
gfjsp_50	20	50	26.55	405.60
gfjsp_100	20	100	51.45	811.00
gfjsp_150	20	150	76.60	1212.90

Table 2. BH results

Class	CPU Time (s)			Improvement (%)		
	Min	Average	Max	Min	Average	Max
gfjsp_10	0.00	0.00	0.00	0.00%	7.76%	43.89%
gfjsp_25	0.00	0.00	0.00	0.00%	10.38%	50.13%
gfjsp_50	0.00	0.00	0.05	0.00%	12.73%	57.06%
gfjsp_100	0.00	0.02	0.45	0.00%	14.61%	60.05%
gfjsp_150	0.00	0.09	1.45	0.00%	15.85%	63.27%

4.3 Performance of the CP Model

Table 3 presents the results of the CP model, including the number of jobs lost, the CPU time in seconds required to generate a schedule for a new request, and the optimality gap reported by the CP solver. The results are shown for each class of instances, detailing the minimum, average, and maximum CPU times and the minimum, average, and maximum optimality gaps.

The CP model consistently generates near-optimal solutions within one minute for all instances, with a maximum CPU time of 60 seconds for the gfjsp_150 class. The average optimality gap remains low, below 2% for all instances, with the maximum gap reaching 29.16% for the gfjsp_150 class. This indicates that the CP model can quickly generate high-quality schedules for new requests, ensuring the earliest possible delivery.

Notably, the number of jobs lost is zero across all instances. This confirms that our approach successfully integrates new requests into the current schedule without modifying the deadlines of previously scheduled tasks. The proposed BH

is crucial in providing a feasible solution under various circumstances. Furthermore, it is evident that the CP model effectively refines the schedule to guarantee the earliest delivery of new orders, even for the largest instances, while adhering to the 60-second time limit.

4.4 Comparison with the Deterministic Method

Table 4 summarizes the results of the deterministic method. It presents the total earliness, the CPU time (in seconds) required to generate a schedule, the optimality gap reported by the CP solver, and the percentage improvement in fitness compared to the deterministic method. For each class of instances, we provide the minimum, average, and maximum values for total earliness, CPU time, and optimality gap. The percentage improvement in fitness is similarly detailed.

Our approach generates schedules comparable to those produced by the deterministic method, with a maximum fitness improvement of 36.75% for the gfjsp_100 class. The lower improvement in the gfjsp_150 class can be attributed to the high complexity of the instances, which increases the

Table 3. CP model results

Class	Jobs lost	CPU Time (s)			Optimality Gap (%)		
		Min	Average	Max	Min	Average	Max
gfjsp_10	0	0.00	0.00	0.00	0.00%	0.00%	0.01%
gfjsp_25	0	0.00	0.00	0.00	0.00%	0.00%	0.01%
gfjsp_50	0	0.00	0.61	3.00	0.00%	0.00%	0.01%
gfjsp_100	0	1.00	6.15	24.00	0.00%	0.00%	0.17%
gfjsp_150	0	3.00	23.88	60.00	0.00%	1.98%	29.16%

convergence time of the CP solver. We imposed a time limit of 600 seconds for the CP solver, which may not be sufficient to reach optimal solutions for the largest instances. This understanding is crucial in interpreting our results.

The reported total earliness is relatively high, indicating the potential for significant improvements in delivery times if the release times were known in advance. However, when considering the relative improvement in fitness individually, the total earliness remains relatively low, suggesting that the schedules generated by our approach are close to optimal solutions.

This underscores the effectiveness of our dynamic scheduling method in complex, constraint-rich manufacturing environments, where integrating new orders into an existing schedule without altering the deadlines of previously scheduled tasks is crucial.

5 Discussion

The results of our computational experiments demonstrate the effectiveness of our approach in generating schedules for new requests in dynamic manufacturing environments. The Backtracking Heuristic (BH) efficiently produces feasible schedules for new requests, which are subsequently refined by the Constraint Programming (CP) model to ensure the earliest possible delivery of each order. Our findings indicate that the CP model consistently generates near-optimal solutions within a short time frame, resulting in schedules that closely approximate optimal solutions.

Notably, our approach yields schedules that are competitive with the deterministic method, in particular for the gfjsp_100 class. This result highlights the efficacy of our dynamic scheduling method in complex, constraint-rich manufacturing settings, where it is critical to integrate new orders into existing schedules without altering the deadlines of previously scheduled tasks.

Introducing maximum time lag constraints adds complexity to the scheduling problem, making it crucial to follow these constraints closely. Classical greedy algorithms often struggle in these situations, sometimes leading to negative cycles in the scheduling graph. In contrast, our BH algorithm effectively manages these constraints by using backtracking techniques. This allows the algorithm to return to previous operations and try different assignments, helping it avoid negative cycles and meet the time lag requirements. Our findings show that mathematical programming models, particularly CP models, can effectively tackle complex scheduling issues in real-world manufacturing settings. However, to improve reliability and robustness, it is important to combine these mathematical models with heuristic methods that can quickly produce feasible solutions. This hybrid approach leads to more efficient and flexible scheduling strategies in dynamic environments.

6 Conclusions and Future Work

This study proposed a flexible framework for the Dynamic Generalized Flexible Job Shop

Table 4. Deterministic results

Class	Earliness	CPU Time (s)	Gap (%)	Improvement (%)		
				Min	Average	Max
gfjsp_10	1.00	0.00	0.01%	0.00%	0.06%	7.30%
gfjsp_25	44.20	2.00	0.01%	0.00%	0.07%	5.25%
gfjsp_50	286.70	600.00	0.58%	0.00%	0.30%	19.49%
gfjsp_100	864.65	600.15	2.96%	0.00%	0.75%	36.75%
gfjsp_150	394.50	600.55	10.91%	0.00%	0.16%	8.57%

Scheduling Problem in Make-to-Order environments, combining a rapid backtracking heuristic with a constraint programming model.

The results demonstrate that new orders can be seamlessly integrated into ongoing schedules in real time without affecting previously committed deadlines.

The approach guarantees full job acceptance, delivers high-quality schedules within a one-minute time frame, and effectively handles complex operational constraints such as setup times and time lags.

These findings highlight the potential of hybrid CP-based approaches to support responsive and reliable decision-making in Industry 4.0 contexts.

Future research can extend this framework in several directions.

First, incorporating additional sources of uncertainty, such as machine breakdowns or process disruptions, would enhance system robustness.

Second, improving scalability for large-scale instances may require advanced pruning strategies or hybridization with metaheuristics.

Finally, extending the model to a multi-objective setting, including criteria such as energy consumption, maintenance costs, or tardiness, would better reflect real-world manufacturing trade-offs.

Acknowledgments

This research was supported by the Secretariat of Science, Humanities, Technology, and Innovation of Mexico through grant CBF-2025-I-2005.

References

1. Arroyo, J. E. C., Leung, J. Y.-T., Tavares, R. G. (2019). An iterated greedy algorithm for total flow time minimization in unrelated parallel batch machines with unequal job release times. *Engineering Applications of Artificial Intelligence*, Vol. 77, pp. 239–254. DOI: <https://doi.org/10.1016/j.engappai.2018.10.012>.
2. Beldar, P., Moghtader, M., Giret, A., Ansaripoor, A. H. (2022). Non-identical parallel machines batch processing problem with release dates, due dates and variable maintenance activity to minimize total tardiness. *Comput. Ind. Eng.*, Vol. 168, pp. 108135.
3. Boyer, V., Vallikavungal, J., Cantú Rodríguez, X., Salazar-Aguilar, M. A. (2021). The generalized flexible job shop scheduling problem. *Computers & Industrial Engineering*, Vol. 160, pp. 107542. DOI: <https://doi.org/10.1016/j.cie.2021.107542>.
4. Chen, Z., Zhang, L., Wang, X., Gu, P. (2022). Optimal design of flexible job shop scheduling under resource preemption based on deep reinforcement learning. *Complex System Modeling and Simulation*, Vol. 2, No. 2, pp. 174–185. DOI: 10.23919/CSMS.2022.0007.
5. Chorghe, S., Kumar, R., Kulkarni, M. S., Pandhare, V., Lad, B. K. (2024). Smart scheduling for next generation manufacturing systems: a systematic literature review. *Journal of Intelligent Manufacturing*.

6. Dautère-Pères, S., Ding, J., Shen, L., Tamssaouet, K. (2024). The flexible job shop scheduling problem: A review. *European Journal of Operational Research*, Vol. 314, No. 2, pp. 409–432.
7. Han, J.-H., Lee, J.-Y. (2023). Scheduling for a flow shop with waiting time constraints and missing operations in semiconductor manufacturing. *Engineering Optimization*, Vol. 55, pp. 1742 – 1759.
8. Jeong, B., Han, J.-H., Lee, J.-Y. (2021). Metaheuristics for a flow shop scheduling problem with urgent jobs and limited waiting times. *Algorithms*, Vol. 14, pp. 323.
9. Kong, M., Liu, X., Pei, J., Cheng, H., Pardalos, P. M. (2020). A brkga-de algorithm for parallel-batching scheduling with deterioration and learning effects on parallel machines under preventive maintenance consideration. *Annals of Mathematics and Artificial Intelligence*, Vol. 88, No. 1, pp. 237–267.
10. Lu, S., Liu, X., Pei, J., T. Thai, M., M. Pardalos, P. (2018). A hybrid abc-ts algorithm for the unrelated parallel-batching machines scheduling problem with deteriorating jobs and maintenance activity. *Applied Soft Computing*, Vol. 66, pp. 168–182. DOI: <https://doi.org/10.1016/j.asoc.2018.02.018>.
11. Luo, S. (2020). Dynamic scheduling for flexible job shop with new job insertions by deep reinforcement learning. *Applied Soft Computing*, Vol. 91, pp. 106208. DOI: <https://doi.org/10.1016/j.asoc.2020.106208>.
12. Mohan, J., Lanka, K., Rao, A. N. (2019). A review of dynamic job shop scheduling techniques. *Procedia Manufacturing*, Vol. 30, pp. 34–39.
13. Qamhan, A. A., Ahmed, A., Al-harkan, I., Badwelan, A., Al-Samhan, A., Hidri, L. (2020). An exact method and ant colony optimization for single machine scheduling problem with time window periodic maintenance. *IEEE Access*, Vol. 8, pp. 44836–44845.
14. Varela, L., Putnik, G. D., Alves, C. F., Lopes, N., Cruz-Cunha, M. M. (2022). A systematic review of manufacturing scheduling for the industry 4.0. Matt, D. T., Vidoni, R., Rauch, E., Dallasega, P., editors, *Managing and Implementing the Digital Transformation*, Springer International Publishing, Cham, pp. 237–249.
15. Yazdani, M., Khalili, S. M., Babagolzadeh, M., Jolai, F. (2017). A single-machine scheduling problem with multiple unavailability constraints: A mathematical model and an enhanced variable neighborhood search approach. *J. Comput. Des. Eng.*, Vol. 4, pp. 46–59.
16. Yuan, E., Wang, L., Cheng, S., Song, S., Fan, W., Li, Y. (2024). Solving flexible job shop scheduling problems via deep reinforcement learning. *Expert Systems with Applications*, Vol. 245, pp. 123019. DOI: <https://doi.org/10.1016/j.eswa.2023.123019>.
17. Yuraszeck, F., Montero, E., Canut-De-Bon, D., Cuneo, N., Rojel, M. (2023). A constraint programming formulation of the multi-mode resource-constrained project scheduling problem for the flexible job shop scheduling problem. *IEEE Access*, Vol. 11, pp. 144928–144938. DOI: 10.1109/ACCESS.2023.3345793.

Article received on 21/02/2025; accepted on 10/06/2026.

*Corresponding author is Vincent Boyer.