Scheduling Heuristic Learning via Genetic Programming for Dynamic Flexible Job Shop Scheduling with Heterogeneous Batch Arrivals

Luyao Zhu[1], Fangfang Zhang[1(✉)], Yi Mei[1], Mengjie Zhang[1], and Ruibin Bai[2]

[1] Centre for Data Science and Artificial Intelligence and School of Engineering and Computer Science, Victoria University of Wellington, Wellington 6140, New Zealand
fangfang.zhang@vuw.ac.nz

[2] School of Computer Science, University of Nottingham Ningbo China, Ningbo, China

Abstract. Dynamic flexible job shop scheduling (DFJSS) is a challenging combinatorial optimisation problem that requires effective decision-making under dynamic environments. Although genetic programming (GP) has shown success in automatically learning scheduling heuristics, existing research predominantly addresses dynamic events involving single job arrivals, which does not always reflect real-world situations. In practice, heterogeneous batch arrivals where different jobs arrive simultaneously, are quite common and introduce a new decision-making challenge: handling multiple routing decisions (machine assignments) concurrently. However, this problem has received little attention in the literature. To fill this gap, we first formulate the DFJSS problem with heterogeneous batch arrivals. Furthermore, to effectively coordinate simultaneous routing decisions introduced by batch arrivals, GP is used to evolve scheduling heuristics to prioritise hybrid operation-machine pairs instead of jobs. An update strategy is incorporated to reflect the latest system status after each routing assignment, improving decision quality under dynamic changes. Experimental results across 18 scenarios demonstrate that using routing rules to prioritise pairs achieves the best average rank and outperforms compared methods. The effectiveness of the update strategy is also verified. In general, a single routing rule can be effectively applied to both concurrent and individual routing decisions.

Keywords: Scheduling heuristic · Genetic programming · Dynamic flexible job shop scheduling · Batch arrivals

1 Introduction

Dynamic flexible job shop scheduling (DFJSS) [1] is a fundamental combinatorial optimisation problem for resource allocation. Each job consists of a fixed sequence of operations, and each operation can be processed on alternative machines. Hence, DFJSS simultaneously requires machine assignment and operation sequencing under dynamic conditions. Existing studies mainly consider random single-job arrivals, whereas in practice jobs frequently arrive in batches. Batch arrivals occur when multiple jobs are released simultaneously, either as a customer order or accumulated workload. Unlike prior work on batch processing [2], which often assumes homogeneous jobs, this study focuses on heterogeneous batches, where jobs differ in processing times and routing options. Such scenarios are common in customised manufacturing, e.g., bearings production [3].

Note that heterogeneous batch arrivals increase decision complexity. Single-job arrivals involve one routing decision at a time, but batch arrivals require concurrent routing of multiple jobs. A common method is to randomly assign or use manual rules to determine the routing order of jobs in a batch. Genetic Programming (GP) [4] has been successfully applied to learn scheduling rules for DFJSS with single-job arrivals. Extending GP to batch routing is therefore natural. However, this method focuses mainly on individual job attributes and only partially accounts for machine status, lacking a global perspective on jobmachine interactions. Moreover, since one routing decision alters subsequent ones through changes in machine status, incorporating real-time updates is essential. This study proposes a GP-based global prioritisation framework that assigns operationmachine pairs during batch routing. Main objectives are:

– Formulate a DFJSS model with heterogeneous batch arrivals.
– Design a GP-evolved prioritisation strategy with dynamic updates.
– Evaluate algorithm performance and verify the update mechanism that incorporates system status into each routing decision.

2 Background

2.1 Problem Description

In DFJSS, a set of machines processes batches of jobs. Each batch consists of multiple jobs, which arrive dynamically in a batch-by-batch manner, with their details revealed only upon arrival. Each job is composed of an ordered set of operations. An operation may be executed on multiple eligible machines. We consider two optimisation perspectives. In the *job-level* case, jobs within a batch are independent and represent different orders. In the *batch-level* case, jobs in the same batch correspond to a single order, sharing a common weight and due date, as customers are only concerned with the completion of the entire batch.

2.2 GP for DFJSS

In GP for DFJSS, the GP algorithm starts with randomly initialised population. Each individual is then evaluated to determine its fitness value. If the stopping criterion is not met, a parent selection method is applied to choose individuals for generating offspring through genetic operators (i.e., crossover, mutation, and reproduction) thereby forming new population. Otherwise, the algorithm outputs the best individuals (i.e., scheduling rules) from the current generation.

2.3 Related Works on Job Shop Scheduling with Batch Arrival

JSS with batch arrivals is a complex problem in manufacturing environments where jobs arrive in groups rather than individually. Existing research mainly considers machine processing modes, which are classified into parallel batch scheduling [5] and serial batch scheduling [6]. In parallel batch scheduling problems, jobs within a batch are typically identical or have the same processing time, while in serial batch scheduling a strict intra-batch sequence is required, as in electrical discharge machining [7].

However, in either case, the job order within a batch is typically not considered. Batch arrivals can be seen as a special case of workflow scheduling [8], but existing studies often overlook this and apply random routing. In contrast, our study focuses on a more flexible and realistic setting where batch arrivals are frequent, and jobs within a batch may differ in routing paths, processing times, and have no fixed order. This is common in mixed-order manufacturing, custom assembly, and third-party logistics, where jobs with varying requirements are grouped for efficiency. In such environments, the intra-batch job order can significantly affect machine load, queue dynamics, and system performance. Therefore, determining this order is essential to improving scheduling effectiveness.

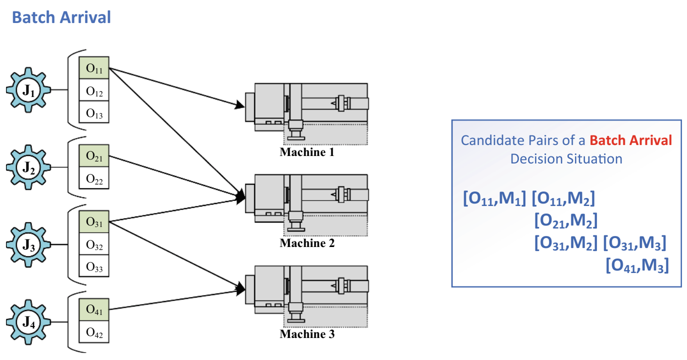

Fig. 1. An example of hybrid routing decision upon batch arrival.

3 Proposed Algorithm

3.1 Main Idea

We propose a fine-grained scheduling strategy that directly prioritises operationmachine pairs upon batch arrival. Unlike conventional methods that first determine a job routing order and then assign operations sequentially, our approach constructs a pool of feasible operationmachine pairs and evaluates them simultaneously. As shown in Fig. 1, each initial operation in the batch may have multiple eligible machines (e.g., operation O_{11} of job J_1 can be processed on

Machine 1 or Machine 2, forming two candidate pairs). A GP-evolved priority rule then assigns priority values to each pair based on system features. This pair-based approach removes the need for predefined job orderings, enabling globally coordinated routing decisions that improve overall system performance.

3.2 Update Strategy

A key aspect of the proposed method is the dynamic update of candidate pairs after each assignment. Since some pairs may involve already-assigned operations, recalculating all priority values is unnecessary. Moreover, each assignment immediately changes machine status (e.g., workload, availability), which influences subsequent decisions. To handle this, we design an update strategy where operationmachine pairs are selected sequentially according to their priority values. After each assignment, two updates are performed: (i) removing all conflicting pairs involving the same operation, and (ii) updating the status of the selected machine and recalculating the priority values of its related pairs.

As shown in Fig. 2, once $[O_{11}, M_2]$ is selected, all pairs involving O_{11} (e.g., $[O_{11}, M_1]$) are removed, and only the priority values of pairs related to M_2 (e.g., $[O_{21}, M_2]$, $[O_{31}, M_2]$) are updated. This iterative selectassignupdate process continues until all operations in the batch are assigned.

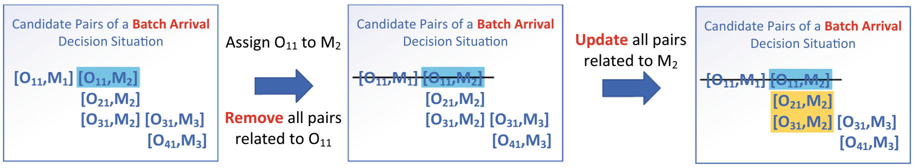
Candidate Pairs of a Batch Arrival Decision Situation
[O11,M1] [O11,M2]
[O21,M2]
[O31,M2] [O31,M3]
[O41,M3]
Assign O11 to M2
Remove all pairs related to O11
Candidate Pairs of a Batch Arrival Decision Situation
~~[O11,M1] [O11,M2]~~
[O21,M2]
[O31,M2] [O31,M3]
[O41,M3]
Update all pairs related to M2
Candidate Pairs of a Batch Arrival Decision Situation
~~[O11,M1] [O11,M2]~~
[O21,M2]
[O31,M2] [O31,M3]
[O41,M3]

Fig. 2. An example of the update strategy.

In addition, a key question is whether a new rule should be evolved to prioritise operationmachine pairs, or if existing routing and sequencing rules can be repurposed. To evaluate their effectiveness in concurrent routing, we compare three approaches: (i) evolving a new routing rule, (ii) reusing the routing rule, and (iii) reusing the sequencing rule.

4 Experiment Design

4.1 Simulation Model

To model the DFJSS environment, we adopt a simulation-based modelling technique [9], representing a system of 10 machines operating at a target utilisation of 0.85. A key determinant of system complexity is batch size: larger batches typically lead to longer queues and increased system load. In this paper, three batch size categories are defined—small ($[1, 9]$), medium ($[10, 20]$), and large

([20, 30])—with the number of jobs per batch sampled from a uniform distribution within each range. Each simulation run processes 6000 jobs. For small batches, this results in approximately 1200 batches, with the final 1000 batches used for performance evaluation after a 200-batch warm-up phase. Batch arrivals are modelled as a Poisson process. More details can be found in [10].

4.2 Design of Comparisons

To ensure a comprehensive comparison, six algorithms are considered. Since cooperative coevolution GP (CCGP) generally outperforms multi-tree GP (MTGP) when evolving two rules, while MTGP is more effective with more than two [11], we use CCGP for learning two rules and MTGP for three. Each algorithm is trained 30 times and tested on 50 unseen simulations, with performance measured by the average objective value of the generated schedules.

– **CCGP-Ran**: random routing order for jobs.
– **CCGP-SPT**: shortest processing time rule for jobs.
– **MTGP-Ordering-Jobs**: an evolved ordering rule for jobs.
– **MTGP-New-Routing-Pairs**: a new routing rule for pairs with update.
– **CCGP-Sequencing-Pairs**: original sequencing rule for pairs with update.
– **CCGP-Routing-Pairs**: the original routing rule for pairs with update.

4.3 Parameter Settings

For a fair comparison, all algorithms have 600 evaluations per generation. CCGP uses two subpopulations of 300 individuals each, with 5 elites per subpopulation. MTGP employs a single population of 600 individuals, 10 elites. The evolution runs for 100 generations. Initial programs are generated via ramped half-and-half with depth 26; the maximum depth during evolution is 8. Offspring are created using crossover (80%), mutation (15%), and reproduction (5%). Each individuals consists of terminals and functions {+, -, *, /}, where "/" is protected division (returns one when dividing by zero). The terminal set can be seen in [12].

5 Results and Analysis

5.1 Test Performance of Using Different Rules to Prioritise Pairs

First, we investigate which rule is most suitable for prioritising pairs. Table 1 shows the objective values obtained with different rules. CCGP-Routing-Pairs achieves the best average rank and performs significantly better than MTGP-New-Routing-Pairs and CCGP-Sequencing-Pairs in 9 and 4 of the 18 test scenarios, respectively. This advantage arises because concurrent routing is essentially a special case of routing, providing more decision points for the routing rule during training. As the decision involves multiple routing assignments, applying routing rules naturally outperforms sequencing rules. Surprisingly, MTGP-New-Routing-Pairs, which evolves a dedicated rule for this decision type, performs significantly worse than CCGP-Sequencing-Pairs and CCGP-Routing-Pairs in 5 and 9 scenarios, respectively, likely because the hybrid nature of pair decisions makes it difficult to directly learn effective heuristics. Therefore, we adopt the original evolved routing rule for pair prioritisation in the subsequent sections.

Table 1. The mean (standard deviation) of the objective values obtained by MTGP-New-Routing-Pairs, CCGP-Sequencing-Pairs, and CCGP-Routing-Pairs across eighteen scenarios based on 30 independent runs.

Scenario	MTGP-New-Routing-Pairs	CCGP-Sequencing-Pairs	CCGP-Routing-Pairs
⟨Job, Fmean, Small⟩	429.02 (2.37)	428.61 (1.29) (≈)	**428.51** (1.40) (≈)(≈)
⟨Job, Fmean, Medium⟩	**746.12** (3.12)	747.24 (5.47) (≈)	747.69 (3.01) (≈)(≈)
⟨Job, Fmean, Large⟩	**1044.40** (8.07)	1045.31 (7.77) (≈)	1047.15 (5.88) (≈)(≈)
⟨Job, WTmean, Small⟩	218.49 (3.50)	218.84 (4.36) (≈)	**216.68** (2.41) (↑)(↑)
⟨Job, WTmean, Medium⟩	761.96 (17.03)	763.18 (16.65) (≈)	**758.23** (5.61) (≈)(≈)
⟨Job, WTmean, Large⟩	1281.78 (14.90)	1290.92 (34.59) (≈)	1282.81 (10.29) (≈)(≈)
⟨Job, Profit, Small⟩	6463.55 (42.17)	6460.37 (33.52) (≈)	**6497.15** (24.90) (↑)(↑)
⟨Job, Profit, Medium⟩	5236.45 (136.00)	5264.64 (109.44) (≈)	**5348.91** (56.72) (↑)(↑)
⟨Job, Profit, Large⟩	3802.61 (199.16)	3899.40 (108.52) (≈)	**3941.18** (119.52) (↑)(≈)
⟨Batch, Fmean, Small⟩	600.38 (2.66)	**599.08** (2.34) (↑)	599.23 (2.37) (≈)(≈)
⟨Batch, Fmean, Medium⟩	1149.97 (9.63)	1144.72 (9.08) (↑)	**1143.06** (7.20) (↑)(≈)
⟨Batch, Fmean, Large⟩	**1718.09** (34.06)	1723.73 (28.23) (≈)	1719.59 (28.63) (≈)(≈)
⟨Batch, WTmean, Small⟩	221.67 (10.26)	222.23 (10.52) (≈)	**216.28** (5.67) (↑)(↑)
⟨Batch, WTmean, Medium⟩	934.03 (44.60)	929.73 (49.47) (≈)	**909.30** (18.42) (↑)(≈)
⟨Batch, WTmean, Large⟩	1601.93 (169.10)	1600.44 (164.38) (≈)	**1537.64** (55.57) (≈)(≈)
⟨Batch, Profit, Small⟩	6203.82 (60.94)	6245.98 (51.81) (↑)	**6276.16** (40.01)(↑)(↑)
⟨Batch, Profit, Medium⟩	4394.28 (108.31)	4398.09 (189.11) (↑)	**4439.60** (98.11)(≈)(≈)
⟨Batch, Profit, Large⟩	3375.45 (388.156)	3534.17 (134.55) (↑)	**3551.5808** (232.50)(↑)(≈)
Win/Draw/Lose	**9/9/0**	**4/14/0**	–
Average Rank	2.32	2.01	**1.68**

5.2 Test Performance of Compared Algorithms

Table 2 presents the objective values of the compared algorithms. CCGP-Routing-Pairs achieves the best average rank and the lowest mean objective values in half of the test scenarios. An interesting observation is that CCGP-SPT performs relatively well in job-level scenarios but poorly in batch-level ones. This is because batch-level objectives depend on the completion time of the longest job in a batch, where scheduling longer jobs earlier can reduce flow time or tardiness, while SPT always prioritises the shortest job. In contrast, algorithms with GP-evolved rules perform robustly across all scenarios, showing greater adaptability. Moreover, compared with MTGP-Ordering-Jobs, CCGP-Routing-Pairs not only performs better in one scenario but also relies on two rules instead of three, thereby enhancing interpretability.

Table 2. The mean (standard deviation) of the objective values obtained by CCGP-Ran, CCGP-SPT, MTGP-Ordering-Jobs, and CCGP-Routing-Pairs in 18 scenarios based on 30 independent runs.

Scenario	CCGP-Ran	CCGP-SPT	MTGP-ordering-Jobs	CCGP-Routing-Pairs
⟨Job, Fmean, Small⟩	429.16 (1.35)	428.45 (1.36) (↑)	**428.28** (2.03) (↑)(≈)	428.51 (1.40) (≈)(≈)(≈)
⟨Job, Fmean, Medium⟩	749.41 (5.01)	**746.80** (2.54) (≈)	748.58 (8.99) (≈)(≈)	747.69 (3.01) (≈)(≈)(≈)
⟨Job, Fmean, Large⟩	1049.91 (6.46)	1047.80 (8.97) (≈)	1049.33 (10.06) (≈)(≈)	**1047.15** (5.88) (≈)(≈)(≈)
⟨Job, WTmean, Small⟩	217.95 (1.98)	217.73 (2.56) (≈)	**215.65** (2.46) (↑)(↑)	216.68 (2.41) (↑)(≈)(≈)
⟨Job, WTmean, Medium⟩	759.28 (5.73)	759.28 (13.57) (≈)	**755.07** (6.53) (↑)(≈)	758.23 (5.61) (≈)(≈)(≈)
⟨Job, WTmean, Large⟩	1288.42 (20.67)	1283.15 (13.67) (≈)	1287.35 (20.78) (≈)(≈)	**1282.81** (10.29) (≈)(≈)(≈)
⟨Job, Profit, Small⟩	6458.63 (30.07)	**6498.97** (30.40) (↑)	6485.55 (36.68) (↑)(≈)	6497.15 (24.90) (↑)(≈)(≈)
⟨Job, Profit, Medium⟩	5272.21 (58.20)	**5356.98** (54.92) (↑)	5329.24 (77.24) (↑)(≈)	5348.91 (56.72) (↑)(≈)(≈)
⟨Job, Profit, Large⟩	3792.91 (90.10)	3924.56 (70.27) (↓)	3887.99 (163.72) (↑)(≈)	**3941.18** (119.52) (↑)(≈)(≈)
⟨Batch, Fmean, Small⟩	600.72 (3.12)	602.30 (2.18) (↓)	**598.78** (2.35) (↑)(↑)	599.23 (2.37) (↑)(↑)(≈)
⟨Batch, Fmean, Medium⟩	1147.69 (7.68)	1149.97 (8.07) (≈)	1145.51 (8.46) (≈)(↑)	**1143.06** (7.20) (↑)(↑)(≈)
⟨Batch, Fmean, Large⟩	1723.24 (37.80)	1724.24 (31.68) (≈)	**1718.36** (46.94) (≈)(↑)	1719.59 (28.63) (≈)(≈)(≈)
⟨Batch, WTmean, Small⟩	217.69 (5.76)	221.78 (9.23) (≈)	219.02 (8.19) (≈)(≈)	**216.28** (5.67) (≈)(↑)
⟨Batch, WTmean, Medium⟩	**909.06** (19.39)	923.00 (25.73) (↓)	919.23 (38.50) (≈)(≈)	909.30 (18.42) (≈)(↑)(≈)
⟨Batch, WTmean, Large⟩	1550.03 (42.88)	1559.85 (81.18) (≈)	1566.07 (128.71) (≈)(≈)	**1537.64** (55.57) (≈)(≈)(≈)
⟨Batch, Profit, Small⟩	6273.30 (29.03)	6265.00 (35.78) (≈)	6243.31 (52.56) (↓)(↓)	**6276.16** (40.01) (≈)(≈)(↑)
⟨Batch, Profit, Medium⟩	4438.09 (91.82)	4404.93 (79.04) (≈)	4396.25 (174.34) (≈)(≈)	**4439.60** (98.11) (≈)(≈)(≈)
⟨Batch, Profit, Large⟩	3518.65 (244.60)	3552.48 (108.10) (≈)	3493.60 (269.38) (≈)(≈)	**3551.58** (232.50) (≈)(≈)(≈)
Win/Draw/Lose	6/12/0	4/14/0	1/17/0	–
Average Rank	2.66	2.49	2.59	**2.27**

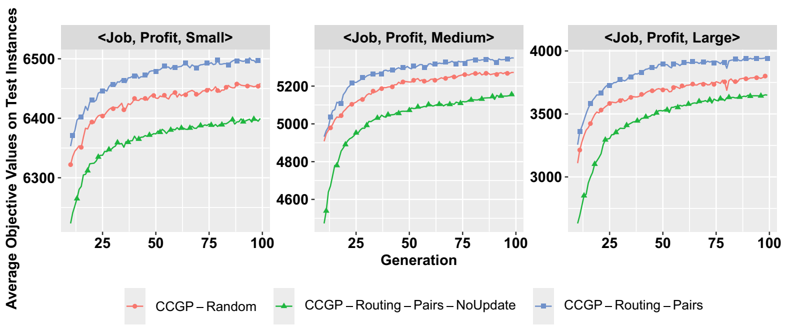

Fig. 3. Curves of average objective values on test instances of CCGP-Routing-Pairs with and without updating in nine scenarios according to 30 independent runs.

5.3 The Effectiveness of Update Strategy in CCGP-Routing-Pairs

A key component in CCGP-Routing-Pairs is the update strategy, which recalculates the priority values of affected pairs. Figure 3 presents results: higher values are preferred for profit-based scenarios. Omitting the update step leads to inferior performance in all cases, since outdated machine states cause inefficient assignments. Incorporating updates enables adaptive decisions, and although only part of the results are shown, other scenarios exhibit the same pattern.

6 Conclusions and Future Work

The goal of this paper is to use GP to learn effective scheduling heuristics to address multiple routing challenges introduced by batch arrivals. This goal has been successfully achieved by designing a global prioritisation strategy for operationmachine pairs based on routing rule, combined with a dynamic update strategy that reflects the real-time system status during batch routing.

Results show that when dealing with concurrent routing decisions introduced by batch arrivals, it is recommended to use the original evolved routing rule to handle these multiple decisions simultaneously. This is because concurrent routing decisions are essentially a special case of standard routing. Therefore, a single routing rule is sufficient to manage both concurrent routing under batch arrivals and individual routing decisions throughout the scheduling process. In addition, the effectiveness of the update strategy highlights the importance of using the latest system status.

Several interesting directions can be explored in future work. One promising direction is to incorporate GP with local search [13] to balance the exploration and exploitation capabilities of GP algorithms in this problem.

References

1. Zhang, F., Mei, Y., Nguyen, S., Zhang, M.: Survey on genetic programming and machine learning techniques for heuristic design in job shop scheduling. IEEE Trans. Evolutionary Comput. **28**(1), 147–167 (2023)
2. Mönch, L., Fowler, J.W.: A survey of scheduling with parallel batch (p-batch) processing. European J. Oper. Res. **298**(1), 1–24 (2022)
3. Wang, Y., Shi, J., Wang, W., Li, C.: Re-entrant green scheduling problem of bearing production shops considering job reworking. Machines **12**(4), 281 (2024)
4. John, R.K.: Genetic programming: a paradigm for genetically breeding populations of computer programs to solve problems, vol. 34. Stanford University, Department of Computer Science Stanford (1990)
5. Jun, X., Wang, J.-Q., Liu, Z.: Parallel batch scheduling: impact of increasing machine capacity. Omega **108**, 102567 (2022)
6. Wahl, S., Gahm, C., Tuma, A.: Serial-and hierarchical-batch scheduling: a systematic review and future research directions. Int. J. Prod. Res., 1–31 (2024)
7. Xiong, J., Xing, L., Chen, Y.: Robust scheduling for multi-objective flexible job-shop problems with random machine breakdowns. Int. J. Produ. Econ. **141**(1), 112–126 (2013)
8. Yang, Y., Chen, G., Ma, H., Hartmann, S., Zhang, M.: Dual-tree genetic programming with adaptive mutation for dynamic workflow scheduling in cloud computing. IEEE Trans. Evolutionary Comput., 1–1 (2024). https://doi.org/10.1109/TEVC.2024.3392968
9. Kiran, A.S., Smith, M.L.: Simulation studies in job shop sheduling–i a survey. Comput. Industr. Eng. **8**(2), 87–93 (1984)
10. Zhu, L., Zhang, F., Zhu, X., Chen, K., Zhang, M.: Phenotype and genotype based sample aware surrogate-assisted genetic programming in dynamic flexible job shop scheduling. IEEE Trans. Artif. Intell. (2025). https://doi.org/10.1109/TAI.2025.3562161

11. Zhang, F., Mei, Y., Nguyen, S., Zhang, M.: Learning strategies on scheduling heuristics of genetic programming in dynamic flexible job shop scheduling. In: Proceedings of the IEEE Congress on Evolutionary Computation, pp. 1–8. IEEE (2022)
12. Zhu, L., Zhang, F., Feng, M., Chen, K., Zhu, X., Zhang, M.: Sample-aware surrogate-assisted genetic programming for scheduling heuristics learning in dynamic flexible job shop scheduling. In: Proceedings of the Genetic and Evolutionary Computation Conference, pp. 384–392 (2023)
13. Jin, C., Bai, R., Zhou, Y., Chen, X., Tan, L.: Enhancing online yard crane scheduling through a two-stage rollout memetic genetic programming. Memetic Comput. **16**(3), 467–489 (2024)