2024 6th International Conference on Data-driven Optimization of Complex Systems (DOCS)

Optimizing Energy-Efficient Flexible Job Shop Scheduling with Transportation Constraints: A Q-Learning Enhanced Quality-Diversity Algorithm

Haoxiang Qin
School of Software Engineering
South China University of Technology
Guangzhou, China
987352978@qq.com

Yi Xiang*
School of Software Engineering
South China University of Technology
Guangzhou, China
*Corresponding author. xiangyi@scut.edu.cn

Yuyan Han
School of Computer Science
Liaocheng University
Liaocheng, China
hanyuyan@lcu-cs.com

Xueming Yan
School of Information Science and Technology
Guangdong University of Foreign Studies
Guangzhou, China
yanxm@gdufs.edu.cn

***Abstract*—The Flexible Job Shop Scheduling Problem (FJSP), particularly one with transportation constraints, is prevalent in the intelligent manufacturing field. Leveraging the intricacies of these transportation constraints is recognized for its potential to enhance problem-solving efficacy. Despite this, there has been a dearth of research focusing on this approach. This paper posits that integrating transportation conditions into local search operators can significantly bolster the ability to solve such problems. To make well-informed decisions among local search operators, we have implemented a reinforcement learning technique known as Q-learning. Furthermore, we design a Quality-Diversity (QD) algorithm aimed at preserving solution diversity within a tailored feature space. This space is designed in accordance with the unique attributes of transportation constraints. The empirical results from testing on 20 instances indicate that our proposed algorithm shows great promise, achieving an average 6% reduction in the optimization objective when compared to existing state-of-the-art algorithms.**

***Index Terms*—Flexible job shop scheduling with transportation constraints, energy-efficient, Q-learning, Quality-Diversity algorithm**

I. Introduction

The Flexible Job Shop Scheduling Problem (FJSP) is a significant challenge in the field of combinatorial optimization, being both NP-hard and widely prevalent in the intelligent manufacturing [1], [2]. Characterized by a diverse range of operations for each job, FJSP offers the flexibility to assign these operations to any of the available machines for processing [3]. To optimize job processing, companies often utilize an array of equipment, including cranes, conveyors, and Automated Guided Vehicles (AGVs), to facilitate the transportation of jobs [4]. Integrating transportation constraints into the FJSP (thereafter referred to as FJSP-T) is deemed crucial as it substantially augments the applicability and potency of the developed solutions, thereby warranting in-depth investigation.

The FJSP-T is a multifaceted challenge that necessitates the simultaneous consideration of job sequencing, machine allocation, and AGV transportation. While mathematical methods theoretically ensure solution quality, they often struggle to tackle large-scale instances of FJSP-T within reasonable timeframes, a limitation stemming from the problem's NP-hard nature [5]. Evolutionary algorithms offer a heuristic approach to address FJSP-T within the constraints of limited time [6]. Nonetheless, these algorithms might not fully capitalize on the potential of local search operators, a deficiency that could adversely impact their overall performance [7]. To bolster the efficacy of evolutionary algorithms, Q-Learning can be employed to inform strategic decisions within the algorithmic process [8].

The Quality-Diversity (QD) algorithm is a significant paradigm within evolutionary computation [9]. Distinct from conventional methods, QD algorithms concentrate on uncovering a diverse array of solutions that excel across a spectrum of performance metrics within a defined feature space—a space constituted by a collection of features [10]. Indeed, the QD algorithm has garnered success across a myriad of domains, such as robotics [11], gaming [12]. However, to our best knowledge, no studies have yet delved into harnessing the QD framework to tackle the FJSP-T. This gap presents an innovative avenue for approaching FJSP-T, marking it as a compelling and significant area of research.

In recent years, Reinforcement Learning (RL) has been successfully implemented in various shop scheduling scenarios [13], [14]. In [13], a parameter selection mechanism grounded

This work was supported by the Guangdong Basic and Applied Basic Research Foundation (2024A1515030022); National Natural Science Foundation of China (61906069).

979-8-3503-7784-2/24/$31.00 ©2024 IEEE

in RL was introduced to facilitate sound decision-making, which subsequently enhanced the diversity of the solutions. Another work incorporated a neighborhood structure to delve into the solution space during evolution [14]. Throughout the search process, Q-Learning was employed to choose the appropriate neighborhood structure, effectively bolstering the algorithm's local search abilities. These studies collectively illustrate that evolutionary algorithms can effectively strike a balance between exploration and exploitation by integrating RL techniques [14]. Indeed, the integration of current RL methods into evolutionary algorithms has proven to be immensely beneficial for decision-making processes.

This paper proposes an improved algorithm that combines the QD algorithm with Q-Learning, referred to as QQD.

The main contributions are given as follows:

- In this paper, a new decoding method of the energy-efficient FJSP-T is proposed.
- The QD algorithm considers the diversity of solutions within the feature space, which aids in preventing the algorithm from getting trapped in local optima. Q-Learning method intelligently choose appropriate local search operators, thereby significantly enhancing the exploitation of the QD algorithm.
- Q-Learning smartly utilizes these local search operators through strategic selection, particularly those tailored to meet transportation conditions. This approach guides the algorithm to evolve in a more promising direction, steering clear of futile search spaces.

II. The Encoding and Decoding of the FJSP-T

It should be mentioned that the contribution of this paper is not in the mathematic modeling, but in the decoding rules. For ease of description, the parameters and variables can be seen in appendix.

The objective of FJSP-T are shown as follows:

$$\text{Minimize obj} \ = TP + TI + TT \tag{1}$$

$$TP = \sum_{i \in I} \sum_{j \in J_i} \sum_{k \in M_{i,j}} TP_{i,j,k} \tag{2}$$

$$TI = \sum_{k \in M} TI_k \tag{3}$$

$$TT = \sum_{i \in I} \sum_{j \in J_i} \sum_{k,k' \in M_{i,j}} TT_{i,j,k,k'} \tag{4}$$

$$TP_{i,j,k} = T_{i,j,k} \cdot PP_k \quad \forall i \in I, j \in J_i, k \in M_{i,j} \tag{5}$$

$$\begin{aligned} TI_k = (C_{i,j-1} - P_k + TR_{i,j,k,k'}) \cdot PI_k \\ \forall i \in I, j \in J_i, k, k' \in M_{i,j} \end{aligned} \tag{6}$$

$$TT_{i,j,k,k'} = TR_{i,j,k,k'} \cdot PT_{i,j,k,k'} \quad \forall i \in I, j \in J_i, k, k' \in M_{i,j} \tag{7}$$

Additionally, the constraints followed by this problem are:

(1) Jobs cannot be interrupted once they start processing.

(2) Each operation can only be processed by one machine, and each machine can only process one operation at a time.

(3) All machines are available at moment 0.

(4) The processing time of a job is greater than or equal to 0.

(5) Conditions such as machine setup time are included in the processing time.

This section details the encoding and decoding processes for the Flexible Job Shop Scheduling Problem with Transportation constraints (FJSP-T). As depicted in Fig. 1, an integer encoding schema is utilized, which is sourced from [15]. This schema encompasses two primary vectors: 1) the Operation Sequence OS; and 2) the Machine Selection MS. Both the OS and MS vectors are of a length equivalent to the total count of operations, denoted as n_{max}.

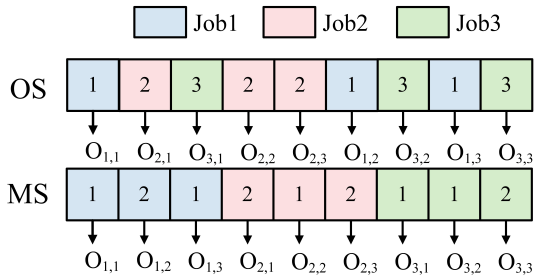

Fig. 1. The encoding vectors of one solution.

As outlined in Algorithm 1, a decoding procedure, denoted as *Decoding(x)*, has been crafted to assess a given solution. This procedure yields two principal outputs: obj_x and $\boldsymbol{b}_x$. Specifically, obj_x represents the objective value associated with the solution x, whereas $\boldsymbol{b}_x$ signifies the coordinates of x within the feature space. These coordinates are ascertained by two distinct features: num_idle and num_trans. The former corresponds to the total number of machine idle instances, and the latter denotes the total number of transportation occurrences for all jobs.

Before delving into the specifics of the *Decoding(x)*, it is essential to present some critical formulas. The $P_{U_{i,j}}$ variable represents the availability time of machine $U_{i,j}$. If $P_{U_{i,j}}$ is less than $C_{i,j-1}$, it indicates that machine $U_{i,j}$ is idle. Equation (8) is utilized to determine $P_{U_{i,j}}$:

$$P_{U_{i,j}} = C_{i,j-1} + TR_{i,j,U_{i,j-1},U_{i,j}} + T_{i,j,U_{i,j}} \tag{8}$$

If $P_{U_{i,j}} \geqslant C_{i,j-1}$, $U_{i,j}$ is not in idle state. When $j = 1$, Equation (9) is used to calculate $P_{U_{i,j}}$; otherwise, either Equation (8) or (9) is adopted to calculate $P_{U_{i,j}}$ (Lines 18-24).

$$P_{U_{i,j}} = P_{U_{i,j}} + T_{i,j,U_{i,j}} \tag{9}$$

As depicted in Algorithm 1, lines 4-5 identify the selected machine for operation $O_{i,j}$ from the MS vector. Lines 7-13 outline the calculation for $TP_{i,j,U_{i,j}}$, $TI_{U_{i,j}}$, and $TT_{i,j,U_{i,j-1},U_{i,j}}$, and count the number of num_idle and num_trans when machine $U_{i,j}$ is idle. Lines 15-32 describe the calculation for $TP_{i,j,U_{i,j}}$, $TI_{U_{i,j}}$, and $TT_{i,j,U_{i,j-1},U_{i,j}}$, and

Algorithm 1 Decoding (x)

Input: x
Output: The objective value obj_x of x and its corresponding coordinates in feature space $\boldsymbol{b}_x$

1: x can be represented by a combination of OS *and* MS
2: $num\_idle \leftarrow 0,\ num\_trans \leftarrow 0$
3: **for** $pos \leftarrow 1$ **to** n_{max} **do**
4: $O_{i,j} \leftarrow OS_{pos}$
5: Find $U_{i,j}$ from MS *according to the position of* $O_{i,j}$
6: **if** $P_{U_{i,j}} < C_{i,j-1}$ **then**
// Machine $U_{i,j}$ is idle
7: $num\_idle \leftarrow num\_idle + 1$
8: Calculate the $P_{U_{i,j}}$ using Eq.(8)
9: $C_{i,j} \leftarrow P_{U_{i,j}}$
10: **if** $U_{i,j-1} != U_{i,j}$ **then**
11: $num\_trans \leftarrow num\_trans + 1$
12: **end if**
13: Calculate $TP_{i,j,U_{i,j}}$, $TI_{U_{i,j}}$, and $TT_{i,j,U_{i,j-1},U_{i,j}}$ using Eqs.(5-7), respectively.
14: **else**
// Machine $U_{i,j}$ is not idle
15: **if** $j == 1$ **then**
16: Calculate $P_{U_{i,j}}$ using Eq.(9)
17: **else**
18: **if** $P_{U_{i,j}} < C_{i,j-1} + TR_{i,j,U_{i,j-1},U_{i,j}}$ **then**
19: Calculate $TI_{U_{i,j}}$ using Eq.(6)
20: $num\_idle \leftarrow num\_idle + 1$
21: Calculate $P_{U_{i,j}}$ using Eq.(8)
22: **else**
23: Calculate $P_{U_{i,j}}$ using Eq.(9)
24: **end if**
25: **if** $U_{i,j-1} != U_{i,j}$ **then**
26: $num\_trans \leftarrow num\_trans + 1$
27: **end if**
28: Calculate $TT_{i,j,U_{i,j-1},U_{i,j}}$ using Eq.(7)
29: **end if**
30: $C_{i,j} \leftarrow P_{U_{i,j}}$
31: Calculate $TP_{i,j,U_{i,j}}$ using Eq.(5)
32: **end if**
33: **end for**
34: Calculate TP, TI, and TT using Eqs.(2-4), respectively.
35: $obj_x \leftarrow TP + TI + TT$
36: $\boldsymbol{b}_x \leftarrow (num\_idle, num\_trans)$
return $\{obj_x, \boldsymbol{b}_x\}$

count the number of num_idle and num_trans when $U_{i,j}$ is not idle. In line 35, the total energy consumption, denoted as obj_x, is calculated. Line 36 obtains the coordinates of two features.

III. The Proposed Algorithm

Algorithm 2 presents the framework for the QQD. The input parameters encompass: the grid dimension N, batch size $batch$, maximum evaluations Max_Iter, learning rate α, discount factor γ, and greedy factor ϵ. The algorithm's output yields a feature-performance grid consisting of $\mathcal{P}$ and $\mathcal{X}$. P represents the objective values, while $\mathcal{X}$ signifies the set of solutions within the grid.

A. Framework of the QQD

Lines 3-4 list the initialization operations of the QQD. The reward values are not involved in the computation of the initialization phase. In lines 6-7, the Q-table is initialized with a default value of 0. The state s_1 has an initial value of 1, which indicates the first solution. The value of the state in the Q-table indicates the number of a solution. In Line 10, a solution is randomly selected from the grid. Line 11 of the algorithm utilizes the Q-Learning algorithm for strategic decision-making. It describes the selection of action a_t through Equation 10. In line 12, the value of ϵ is decremented. By randomly selecting actions for trial and error, the Q-table accumulates increasingly valuable information. Lines 13-14 execute a local search operator based on action a_t and yield a new solution x'. This new solution is subsequently placed into the grid for evaluation. Lines 15-19 update the subsequent state s_t and the count of solutions. Line 20 revises the value of $Q(s_t, a_t)$ according to Equation 11. In Equation 11, the learning rate α is gradually reduced as per Equation 12 (line 21).

Algorithm 2 Q-Learning based QD Algorithm

Input: N, $batch$, Max_Iter, α, γ, ϵ
Output: feature-performance grid $(\mathcal{P}\ and\ \mathcal{X})$

1: $\mathcal{P} \leftarrow \emptyset$, $\mathcal{X} \leftarrow \emptyset$
2: **for** $i \leftarrow 1$ **to** *batch* **do**
3: $x \leftarrow$ random_solution ()
4: Add_to_grid $(\mathcal{P}, \mathcal{X}, x)$ // *Algorithm* 3
5: **end for**
6: $iter \leftarrow 1,\ s_1 \leftarrow 1$
7: Initialize Q table
8: **while** $iter \leqslant Max\_Iter$ **do**
9: **for** $t \leftarrow 1$ **to** *batch* **do**
10: Randomly select one solution x from $\mathcal{X}$
11: Select a_t using Eq.10
12: $\varepsilon \leftarrow \varepsilon \cdot 0.999$
13: $x' \leftarrow$ Execute LSS for x according to a_t
14: Add_to_grid $(\mathcal{P}, \mathcal{X}, x')$ // *Algorithm* 3
15: **if** $t = batch$ **then**
16: $s_{t+1} \leftarrow 0$
17: **else**
18: $s_{t+1} \leftarrow s_t + 1$
19: **end if**
20: Update $Q(s_t, a_t)$ using Eq.11
21: Update α using Eq.12
22: **end for**
23: **end while**
return feature-performance grid $(\mathcal{P}\ and\ \mathcal{X})$

$$a_t = \begin{cases} \underset{a \in A}{\operatorname{argmax}} Q(s_t, a), \tau \geqslant \varepsilon \\ a_{\text{random}}, \tau < \varepsilon \end{cases} \tag{10}$$

where a_t represents an action that determines the strategy to be taken. $Q(s_t, a)$ represents a particular Q value in Q table for state s_t. In this paper, a state is defined as a solution, and t represents the specific index of the solution. A denotes the set of possible actions. We employ various local search operators to represent these different actions. $\underset{a \in A}{argmax} Q(s_t, a)$ signifies the selection of the action with the highest Q-value among all actions in state s_t. τ is randomly sampled within the range [0,1].

$$Q(s_t, a_t) = Q(s_t, a_t) + \alpha \cdot \left[r_t + \gamma \cdot \arg\max_{a \in A} Q(s_{t+1}, a) - Q(s_t, a_t) \right] \tag{11}$$

where $Q(s_t, a_t)$ indicates the Q-value associated with performing action a_t in state s_t. γ is the the discount factor. $arg\max_{a \in A} Q(s_{t+1}, a)$ denotes the maximum Q-value for the subsequent state s_{t+1}.

$$\alpha = \alpha - (\alpha - 0.01) \cdot iter / Max\_Iter \tag{12}$$

where α represents the learning rate. $iter$ denotes the current iteration number. Max_Iter signifies the total number of iterations.

B. Evaluation function

The objective of Algorithm 3 is to calculate the value of the objective function for the new solution and to update the feature space grid. The feedback signal r_t represents the reward and is utilized to assess the impact of taking an action in each state. Line 1 outlines the computation of $\boldsymbol{b_x}$ and its objective value obj_x. In Lines 3-5, if $\boldsymbol{b_x}$ is empty, the new solution x is directly placed into the cell, and the reward r_t is set to 1. If $\boldsymbol{b_x}$ is not empty and x is superior to the existing solution, the old solution is replaced, and r_t is calculated as Line 9; if not, x is rejected, and the reward r_t is set to 0.

Algorithm 3 Add_to_Grid $(\mathcal{P}, \mathcal{X}, x)$

Input: $\mathcal{P}, \mathcal{X}, x$
Output: feature-performance grid ($\mathcal{P}$ *and* $\mathcal{X}$)
1: $\{obj_x, \boldsymbol{b_x}\} \leftarrow Decoding(x)$ // *Algorithm* 1
2: **if** $\mathcal{P}(\boldsymbol{b_x}) = \emptyset$ **then**
3: $\quad \mathcal{P}(\boldsymbol{b_x}) \leftarrow obj_x$
4: $\quad \mathcal{X}(\boldsymbol{b_x}) \leftarrow x$
5: $\quad r_t \leftarrow 1$
6: **else if** $\mathcal{P}(\boldsymbol{b_x}) < obj_x$ **then**
7: $\quad \mathcal{P}(\boldsymbol{b_x}) \leftarrow obj_x$
8: $\quad \mathcal{X}(\boldsymbol{b_x}) \leftarrow x$
9: $\quad r_t \leftarrow (\mathcal{P}(\boldsymbol{b_x}) - obj_x)/\mathcal{P}(\boldsymbol{b_x})$
10: **else**
11: $\quad r_t \leftarrow 0$
12: **end if**
return feature-performance grid ($\mathcal{P}$ *and* $\mathcal{X}$)

C. Local Search Strategy

This subsection presents an overview of the six local search operators within the Local Search Strategy (LSS), as outlined in Line 13 of Algorithm 2.

LS1: See [16], identify the machine with the smallest load *Minload*. Subsequently, iterate through all operations on the other machines. If an operation can be processed on the *Minload* machine, it is relocated to this machine.

LS2: Randomly select a job within the scheduling sequence and identify the operation with the longest transportation time. Then, alter the machine assignment for that operation to ensure it differs from the original machine.

LS3: Identify all critical operations on the critical path. Subsequently, it locates the operation with the longest transportation time. Finally, it modifies the machine assignment for that operation.

LS4: See [15], identify all critical operations. Randomly select two distinct critical operations that are not part of the same job sequence. Swap their positions within the schedule.

LS5: See [15], identify all critical operations and then alter the machine assignments for one of the critical operations.

LS6: See [15], randomly select one critical operation and a non-critical operation. Determine their current positions in the schedule. Insert the non-critical operation ahead of the critical operation.

IV. Simulation experiment

A. Parameter settings

The experiments were conducted on a Windows 11 operating system, utilizing an Intel Core i7 processor at 2.10 GHz with a frequency of 2.10 GHz and 16.0 GB of RAM.

Following the setup in [13], the number of machines was set as $m \in \{5, 6, 7, 8\}$ and the number of jobs as $i \in \{20, 30, 40, 50, 100\}$. In total, there are 20 distinct instance combinations, and each instance was subjected to 20 independent tests. The maximum number of evaluations for all tests is defined as $Max\_Iter = 20 \cdot n_{max} \cdot m$. For each job, the processing time $T_{i,j,k}$ of its operation varies within the range $[5, 20]$. We refer to the test set in [17] to establish the transportation time for all operations.

Refer to [7] for the use of the Relative Percentage Increment (RPI) to assess the quality of the algorithm. It is defined as follows:

$$RPI_a = (c_a - c_{best}) / c_{best} \times 100\% \tag{13}$$

where c_a represents the energy consumption obtained from algorithm a, and c_{best} is the minimum energy consumption observed across all algorithms. The lower the RPI_a, the better the performance of the algorithm.

B. Comparison experiment and analysis

We compared QQD with five state-of-the-art algorithms. We selected the classical QD algorithm [18]. Additionally, we included hybrid VNS-GA [16], IGSA [3], LRVMA [13], and DQCE [15]. To ensure fairness, we implemented all the comparison algorithms as they were originally described, including their parameter settings. Table I presents the results of these comparative algorithms. The evaluation metrics we utilized are RPI, MEAN, and BEST. 'AVG' indicates the average value of each metric.

In addition, we also drew box plots, as shown in the Fig. 2, QQD got the best RPI value. The distribution of solutions is stable and dense with no high outliers. By comparing with the other five state-of-the-art algorithms, it can be seen that the proposed QQD algorithm is the most effective in solving FJSP-T.

TABLE I
RPI, MEAN AND BEST OBJECTIVE VALUES OF ALGORITHMS.

Instance	RPI						MEAN						BEST					
	QQD	QD	VNS-GA	IGSA	LRVMA	DQCE	QQD	QD	VNS-GA	IGSA	LRVMA	DQCE	QQD	QD	VNS-GA	IGSA	LRVMA	DQCE
20J5M	**0.04**	0.21	0.14	0.16	0.18	0.18	**4720.7**	5494.2	5152.8	5273.7	5330.3	5340.1	**4535**	5412	4846	4854	5172	5152
20J6M	**0.04**	0.23	0.15	0.18	0.19	0.20	**4639.1**	5465.6	5123.4	5225.5	5269.3	5345.9	**4445**	5373	4774	4943	5007	4963
20J7M	**0.04**	0.27	0.18	0.17	0.22	0.24	**4416.9**	5370.3	5010.8	4970.0	5159.5	5247.5	**4230**	5278	4746	4619	5002	5103
20J8M	**0.03**	0.27	0.18	0.19	0.21	0.21	**4367.4**	5392.0	5027.1	5074.1	5150.9	5162.2	**4257**	5307	4714	4863	5036	4980
30J5M	**0.04**	0.19	0.12	0.13	0.15	0.16	**7310.0**	8389.2	7887.5	7972.6	8117.8	8178.3	**7060**	8255	7265	7589	7914	7821
30J6M	**0.03**	0.21	0.13	0.13	0.17	0.18	**7113.3**	8365.3	7784.2	7780.4	8092.8	8169.2	**6896**	8227	7244	7344	7874	7935
30J7M	**0.03**	0.22	0.14	0.14	0.18	0.19	**6910.5**	8156.5	7646.4	7644.3	7848.6	7955.6	**6679**	7985	7140	6893	7709	7685
30J8M	**0.03**	0.25	0.15	0.17	0.19	0.19	**6649.1**	8096.2	7400.0	7565.0	7713.2	7709.8	**6455**	7960	7031	7103	7509	7454
40J5M	**0.04**	0.17	0.08	0.13	0.12	0.14	**9720.3**	10946.4	10154.2	10608.2	10556.9	10687.6	**9385**	10842	9500	10062	10352	10480
40J6M	**0.02**	0.18	0.10	0.12	0.13	0.14	**9428.5**	10864.9	10150.6	10309.8	10436.9	10510.3	**9199**	10681	9601	10004	10242	9988
40J7M	**0.05**	0.21	0.12	0.17	0.15	0.17	**9154.3**	10624.9	9826.1	10203.4	10088.3	10233.8	**8751**	10314	9422	9535	9855	9881
40J8M	**0.04**	0.25	0.15	0.19	0.18	0.18	**8893.1**	10664.4	9793.3	10151.1	10056.3	10035.2	**8520**	10477	9172	9639	9937	9673
50J5M	**0.04**	0.16	0.10	0.14	0.14	0.14	**12226.2**	13624.0	12916.7	13352.3	13315.3	13371.9	**11705**	13421	12308	12966	13104	12941
50J6M	**0.03**	0.17	0.09	0.13	0.13	0.13	**11953.1**	13585.0	12627.7	13110.6	13117.5	13193.7	**11635**	13464	11889	12465	12922	12850
50J7M	**0.04**	0.20	0.11	0.14	0.15	0.15	**11558.4**	13309.4	12323.3	12729.0	12761.4	12799.8	**11120**	13013	11809	12047	12445	12481
50J8M	**0.03**	0.21	0.12	0.16	0.14	0.14	**11172.5**	13189.4	12192.0	12598.7	12417.9	12395.4	**10863**	12888	11558	12145	12288	11939
100J5M	**0.02**	0.10	0.05	0.07	0.08	0.09	**25519.8**	27528.1	26179.0	26795.0	26818.0	27069.0	**24937**	27199	25331	26042	26483	26467
100J6M	**0.04**	0.13	0.07	0.11	0.08	0.10	**24784.7**	27006.2	25491.9	26456.5	25932.6	26384.9	**23914**	26762	24524	25484	25576	25807
100J7M	**0.03**	0.15	0.07	0.12	0.08	0.11	**23399.1**	26135.5	24228.8	25464.3	24556.3	25154.8	**22653**	25528	23212	24582	24224	24716
100J8M	**0.03**	0.20	0.10	0.15	0.10	0.13	**22452.7**	25994.7	23763.8	24870.5	23845.9	24532.4	**21699**	25745	22387	23325	23605	23979
AVG	**0.04**	0.20	0.12	0.15	0.15	0.16	**11319.5**	12910.1	12034.0	12407.7	12329.3	12473.8	**10947**	12706.6	11423.7	11825.2	12112.8	12114.8

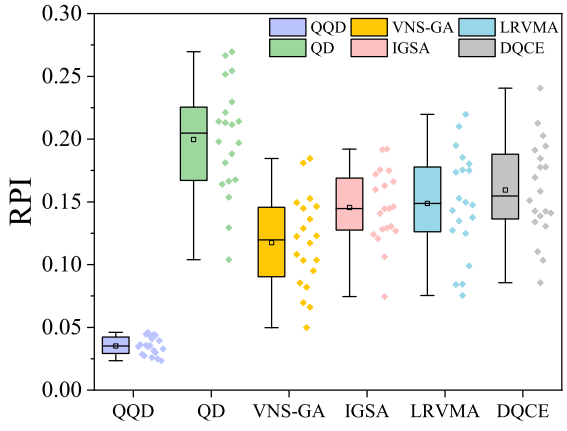

Fig. 2. The bov plot of all algorithms.

V. CONCLUSION

In this paper, we focus on the Flexible Job Shop Problem with Transportation constraints, prioritizing energy consumption as the optimization objective. We propose a Q-Learning enhanced Quality-Diversity algorithm. This integration not only ensures the diversity of solutions but also enhances the local search performance of the algorithm. In our experiments, comparisons with five state-of-the-art algorithms substantiate the effectiveness of QQD. Future research may explore the application of the QD framework in the domain of intelligent manufacturing, as well as investigate the potential of integrating QD with advanced techniques such as neural networks.

ACKNOWLEDGMENT

This work was supported by the Guangdong Basic and Applied Basic Research Foundation (2024A1515030022); National Natural Science Foundation of China (61906069). We are also grateful for Guangyue Young Scholar Innovation Team of Liaocheng University under grant number LCUGYTD2022-03.

REFERENCES

[1] K. Lei, P. Guo, Y. Wang, J. Zhang, X. Meng, and L. Qian, "Large-scale dynamic scheduling for flexible job-shop with random arrivals of new jobs by hierarchical reinforcement learning," *IEEE Transactions on Industrial Informatics*, vol. 20, no. 1, pp. 1007–1018, 2024.

[2] H. Wang, Y. Jiang, H. Wang, and H. Luo, "An online optimization scheme of the dynamic flexible job shop scheduling problem for intelligent manufacturing," in *2022 4th International Conference on Industrial Artificial Intelligence (IAI)*, 2022, pp. 1–6.

[3] J.-Q. Li, Y. Du, K.-Z. Gao, P.-Y. Duan, D.-W. Gong, Q.-K. Pan, and P. N. Suganthan, "A hybrid iterated greedy algorithm for a crane transportation flexible job shop problem," *IEEE Transactions on Automation Science and Engineering*, vol. 19, no. 3, pp. 2153–2170, 2022.

[4] M. Saidi-Mehrabad, S. Dehnavi-Arani, F. Evazabadian, and V. Mahmoodian, "An ant colony algorithm (aca) for solving the new integrated model of job shop scheduling and conflict-free routing of agvs," *Computers & Industrial Engineering*, vol. 86, pp. 2–13, 2015.

[5] J.-J. Wang and L. Wang, "A bi-population cooperative memetic algorithm for distributed hybrid flow-shop scheduling," *IEEE Transactions on Emerging Topics in Computational Intelligence*, vol. 5, no. 6, pp. 947–961, 2021.

[6] K. Gao, F. Yang, M. Zhou, Q. Pan, and P. N. Suganthan, "Flexible job-shop rescheduling for new job insertion by using discrete jaya algorithm," *IEEE Transactions on Cybernetics*, vol. 49, no. 5, pp. 1944–1955, 2019.

[7] H. Qin, Y. Han, Q. Chen, L. Wang, Y. Wang, J. Li, and Y. Liu, "Energy-efficient iterative greedy algorithm for the distributed hybrid flow shop scheduling with blocking constraints," *IEEE Transactions on Emerging Topics in Computational Intelligence*, vol. 7, no. 5, pp. 1442–1457, 2023.

[8] H. Li, K. Gao, P.-Y. Duan, J.-Q. Li, and L. Zhang, "An improved artificial bee colony algorithm with q-learning for solving permutation flow-shop scheduling problems," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 53, no. 5, pp. 2684–2693, 2023.

[9] J. K. Pugh, L. B. Soros, and K. O. Stanley, "Quality diversity: A new frontier for evolutionary computation," *Frontiers in Robotics & Ai*, vol. 3, 2016.

[10] A. Cully and Y. Demiris, "Quality and diversity optimization: A unifying modular framework," *IEEE Transactions on Evolutionary Computation*, vol. 22, no. 2, pp. 245–259, 2018.

[11] A. Cully, J. Clune, D. Tarapore, and J.-B. Mouret, "Robots that can adapt like animals," *Nature*, vol. 521, no. 7553, p. 503–507, May 2015. [Online]. Available: http://dx.doi.org/10.1038/nature14422

[12] A. Ecoffet, J. Huizinga, J. Lehman, K. O. Stanley, and J. Clune, "First return, then explore," *Nature*, vol. 590, no. 7847, p. 580–586, Feb. 2021. [Online]. Available: http://dx.doi.org/10.1038/s41586-020-03157-9

[13] R. Li, W. Gong, C. Lu, and L. Wang, "A learning-based memetic algorithm for energy-efficient flexible job-shop scheduling with type-2 fuzzy processing time," *IEEE Transactions on Evolutionary Computation*, vol. 27, no. 3, pp. 610–620, 2023.
[14] F. Zhao, Z. Wang, and L. Wang, "A reinforcement learning driven artificial bee colony algorithm for distributed heterogeneous no-wait flowshop scheduling problem with sequence-dependent setup times," *IEEE Transactions on Automation Science and Engineering*, vol. 20, no. 4, pp. 2305–2320, 2023.
[15] R. Li, W. Gong, L. Wang, C. Lu, and C. Dong, "Co-evolution with deep reinforcement learning for energy-aware distributed heterogeneous flexible job shop scheduling," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, pp. 1–11, 2023.
[16] G. Zhang, L. Zhang, X. Song, Y. Wang, and C. Zhou, "A variable neighborhood search based genetic algorithm for flexible job shop scheduling problem," *Cluster Computing*, pp. 1–12, 2018. [Online]. Available: https://api.semanticscholar.org/CorpusID:4548038
[17] B. Ümit and U. Gündüz, "A time window approach to simultaneous scheduling of machines and material handling system in an fms," *Operations Research*, vol. 43, no. 6, p. 1058–1070, 1995.
[18] J.-B. Mouret and J. Clune, "Illuminating search spaces by mapping elites," *arXiv preprint arXiv:1504.04909.*, 2015.

APPENDIX

i: The index of job
j: The index of operation
k: The index of machine
n: The total number of jobs
w_i: The number of operations of job i
n_{max}: The total number of operations of all jobs
m: The total number of machines
I: The set of jobs, $I = \{1, ...i, ...n\}$
J_i: The operation set of job i, $J_i = \{1, ...j, ...w_i\}$
M: The set of machines, $M = \{1, ...k, ...m\}$
$O_{i,j}$: The operation j of job i
$M_{i,j}$: The set of selectable machines of $O_{i,j}$
$T_{i,j,k}$: Processing time of $O_{i,j}$ on machine k
$TR_{i,j,k,k'}$: Transportation time of $O_{i,j}$ from machine k to k'
$C_{i,j}$: The completion time of $O_{i,j}$
P_k: The available time set of machine k
$U_{i,j}$: The machine that processes $O_{i,j}$
TP: Total processing EC
TI: Total machine idle EC
TT: Total transportation EC
$TP_{i,j,k}$: Processing EC of operation $O_{i,j}$ on machine k
TI_k: machine idle EC on machine k
$TT_{i,j,k,k'}$: Transportation EC of operation $O_{i,j}$ from machine k to k'
PP_k: The power of machine k processing $O_{i,j}$ per unit time.
PI_k: The power of machine k in idle state per unit time.
$PT_{i,j,k,k'}$: The power of transferring $O_{i,j}$ from machine k to k' per unit time.