Self-Adaptive Differential Evolution-Enhanced Quality-Diversity Optimization for Distributed Flexible Job-Shop Scheduling With Transportation

Haoxiang Qin, Yi Xiang, Yuyan Han, Yuting Wang, Chunguo Wu, and Fujian Feng

Abstract—**Distributed Flexible Job Shop Scheduling with Transportation Constraints (DFJSP-T) is widely used in manufacturing workshops across various industries, as it more accurately represents real-world production scenarios. However, many existing studies do not fully leverage critical problem features or domain knowledge, limiting the ability of current algorithms to find high-quality solutions for complex large-scale problems. To address these challenges, this paper proposes a Self-Adaptive Differential Evolution Enhanced Quality-Diversity Optimization (SADE-QD). The SADE-QD incorporates domain knowledge in three main ways: (1) it models the problem using machine idle and job transportation features, allowing the algorithm to retain solutions with diverse behaviors; (2) it introduces a knowledge-guided heuristic search strategy based on critical path to discover more high-quality solutions; and (3) it applies an self-adaptive differential evolution search method that leverages machine idle and transportation features to explore a broader range of solutions. This helps valuable information from the feature space contribute directly to the search process. Experiments on small, medium, and large scale benchmarks show that SADE-QD achieves an average makespan reduction of 12% compared to several recent state-of-the-art algorithms.**

Index Terms—**Distributed flexible job shop scheduling, transportation, quality-diversity optimization, differential evolution.**

Received 7 December 2025; accepted 29 January 2026. This work was supported in part by Guangdong Basic and Applied Basic Research Foundation under Grant 2024A1515030022, in part by the Fundamental Research Funds for the Central Universities, JLU under Grant 93K172024K03 and Grant 2024ZYGXZR097, in part by the National Natural Science Foundation of China under Grant 61906069, Grant 62566012, and Grant 62541317, and in part by the Natural Science Foundation of Shandong province under Grant ZR2023MF022 and Grant ZR2024MF112. *(Corresponding author: Yi Xiang.)*

Haoxiang Qin is with the School of Software Engineering, South China University of Technology, Guangzhou 510006, China (e-mail: 987352978@qq.com).

Yi Xiang is with the School of Software Engineering, South China University of Technology, Guangzhou 510006, China, and also with the Key Laboratory of Symbolic Computation and Knowledge Engineering of Ministry of Education, Jilin University, Changchun 130012, China (e-mail: xiangyi@scut.edu.cn).

Yuyan Han and Yuting Wang are with the School of Computer Science, Liaocheng University, Liaocheng 252059, China (e-mail: hanyuyan@lcu-cs.com; wangyuting@lcu-cs.com).

Chunguo Wu is with the Key Laboratory of Symbolic Computation and Knowledge Engineering of Ministry of Education, College of Computer Science and Technology, Jilin University, Changchun 130012, China (e-mail: wucg@jlu.edu.cn).

Fujian Feng is with the College of Data Science and Information Engineering, Guizhou Minzu University, Guiyang 550025, China (e-mail: fujian_feng @gzmu.edu.cn).

This article has supplementary downloadable material available at https://doi.org/10.1109/TETCI.2026.3670689, provided by the authors.

Recommended for acceptance by Y. Wang.

Digital Object Identifier 10.1109/TETCI.2026.3670689

I. Introduction

The Distributed Flexible Job Shop Scheduling Problem (DFJSP) extends the traditional Flexible Job Shop Scheduling Problem (FJSP) and is widely applied in modern manufacturing [1], [2]. It integrates job assignment and transportation scheduling across independent machines, enabling flexible resource allocation and greater adaptability [3], [4]. Each job comprises multiple operations that can be processed by various machines [5], and jobs are distributed among multiple factories to better respond to demand fluctuations, shorten production cycles, and enhance customer satisfaction [6], [7], [8]. In DFJSP, processing and transportation are interdependent: operations cannot begin until the required transport is completed, and subsequent transport tasks depend on prior operations. Therefore, to optimize the overall production system, processing and transportation must be addressed in a coordinated and integrated manner [9].

In modern manufacturing scheduling [10], [11], [12], [13], incorporating problem features and knowledge is critical for improving machine utilization and overall production efficiency. Neglecting these factors can lead to unrealistic models, suboptimal resource allocation, and reduced performance [14]. For example, Qin et al. [15] demonstrated that identifying job transportation and machine idleness as key features, and designing heuristics based on transportation characteristics, can enhance scheduling flexibility while reducing energy consumption. Similarly, Pan et al. [16] highlighted the importance of minimizing machine idle time and transport delays by leveraging domain knowledge to adapt to dynamic production conditions. Additionally, in practice, schedulers must also consider the flexible demands of actual production. One effective approach is to generate solutions with diverse characteristics [17]. Studies have shown that real-world environments, such as machining plants in Nanjing, China [18], and coal machinery workshops [19], often require a variety of scheduling strategies to cope with changing operational conditions. Li et al. [20] further demonstrated that offering a range of tailored solutions enables schedulers to reduce idle times and adapt decisions to specific scenarios. By exploring diverse solution sets, decision-makers can better balance trade-offs and select schedules aligned with varying production needs [21], [22], [23].

As scheduling problems grow in complexity, manual approaches can no longer meet the demands of modern production [24]. To enable automated scheduling, many studies have formulated FJSP and DFJSP as mixed-integer linear programming (MILP) models [25], [26], [27], [28]. However, these models are limited to small-scale instances due to combinatorial explosion, making it challenging to obtain efficient or even feasible solutions for larger problems within reasonable time. To overcome this, meta-heuristic algorithms have been widely adopted and have shown strong performance in optimizing scheduling decisions [21], [29], [30], [31], [32], [33], [34], [35], [36], [37], [38], [39], [40], [41], [42], [43], [44], [45]. Despite their success, these methods often overlook important problem-specific features and domain knowledge, such as machine idle time and job transportation. As a result, their adaptability to the diverse and dynamic requirements of real-world production remains limited [19].

To address the limitations of existing methods and better meet the flexibility demands of real-world manufacturing, Quality-Diversity (QD) optimization [46], [47] offers a promising avenue. By leveraging domain-specific features and problem knowledge, QD guides the search toward high-quality solutions while encouraging behavioral diversity, thus promoting cooperation among solutions and helping escape local optima. However, traditional QD frameworks often lack effective variation strategies, which restricts their exploitation ability. To overcome this, we integrate differential evolution (DE) operators [48] into QD. DE exploits promising individuals to generate offspring and facilitates collaborative, targeted evolution across the feature space. Combined with a self-adaptive mechanism that dynamically adjusts the search based on population feedback, the approach becomes well suited for the complex DFJSP-T.

To better solve DFJSP-T, we propose a Self-Adaptive Differential Evolution-enhanced Quality-Diversity Optimization (SADE-QD). The approach uses the widely adopted MAP-Elites framework [49], which has demonstrated strong performance across robotics [50], [51], gaming [52], [53], and software testing [54], [55]. Unlike existing DFJSP-T algorithms, which mainly rely on traditional metaheuristics that operate only in the decision space, our method reformulates DFJSP-T as a QD optimization task in feature space. This changes the goal of the search. Instead of finding a single optimal schedule, the algorithm maintains and evolves a set of behaviorally diverse schedules at the same time. Existing most DFJSP-T approaches do not build or utilize the feature space, and they do not exploit interactions across different behavioral regions. In contrast, SADE-QD introduces a theoretically supported feature space, a MAP-Elites grid archive, and a self-adaptive DE collaboration mechanism that guides evolution based on behavioral information. These designs create a fundamentally different optimization process compared with previous DFJSP-T research.

The core difficulty in extending MAP-Elites to DFJSP-T is the design of meaningful features. Prior work [17] suggested using job transportation and machine idle times for FJSP-T, but provided no theoretical justification. This study fills that gap by formally proving the rationality of these two features and by building the QD model upon this theoretical foundation. Based on this, our contributions focus on three aspects:

- *QD Reformulation with Proven Behavioral Features:* This work models DFJSP-T as a QD optimization problem, providing a novel perspective on its formulation. We theoretically formalize and prove the necessity and sufficiency of using machine idle time and job transportation times as behavior features in QD optimization for DFJSP-T. This provides a principled feature space for MAP-Elites, enabling the algorithm to preserve and explore structurally distinct scheduling solutions that conventional methods cannot represent.
- *Feature-Diverse Evolutionary Framework:* Unlike traditional algorithms that explore toward the search space, the proposed QD paradigm maintains diverse features and exploits cooperation among feature space. This diversity forms a multi-modal search landscape that improves exploration and avoids premature convergence. Our analysis further shows that most successful and high-quality updates arise from neighboring feature solutions rather than from the solution itself, highlighting the cross-cell evolutionary synergy enabled by QD.
- *Self-Adaptive Feature-Guided DE Co-Evolution:* We introduce a self-adaptive differential evolution mechanism that co-evolves the solution vectors under the guidance of MAP-Elites' transportation–idle (T&I) feature space. The strategy no longer depends solely on stochastic mutation. Instead, it progressively shifts solutions toward elite behavioral feature regions with lower transportation and machine idle times. This creates a closed feedback loop: vector perturbations change the behavioral features, and the resulting elites guide the next round of perturbations. The synergy across different behavioral regions enables targeted exploitation while preserving diversity, with 74.8%–89.8% of the obtained solutions outperform state-of-the-art methods.

The remainder of this paper is organized as follows. Section II reviews related work on FJSP, DFJSP, and QD optimization. Section III performs feature selection and validation, and models DFJSP-T as a QD optimization problem. In Section IV, the proposed SADE-QD algorithm is described. Section V shows the results and analysis of SADE-QD. Finally, Section VI concludes the paper and suggests future research directions.

II. Related Work

To contextualize the necessity and novelty of the proposed algorithm, this section critically reviews existing literature on complex FJSPs and DJSPs, then subsequently explores the potential of QD optimization in addressing these recognized challenges.

A. Flexible Job Shop Scheduling

FJSP has been studied for decades [24], [29], [56]. Within FJSP, each operation has the flexibility to be executed on any machine from the available set [30]. To address this problem, researchers have formulated it as an optimization problem, either single-objective or multi-objective, and applied various meta-heuristic algorithms. Meta-heuristic approaches such as Particle Swarm Optimization (PSO) combined with Genetic Algorithms (GA) [31], [32], hybrid GA [40], [57], and hybrid variable neighborhood search (VNS) [37] are widely utilized for the FJSP. Additionally, algorithms utilizing small populations, including the two-individual master-apprentice Evolutionary Algorithm (EA) [33] and the single-individual iterative greedy algorithm [3], have demonstrated promising performance for solving FJSP. Several studies have also explored FJSP under specific constraints. For example, Zhang et al. [34] investigated the FJSP with assembly constraints, integrating multiple flexible and assembled stages. For other complex constraints, such as the mating operation in FJSP, Zhang et al. [35] suggested a knowledge-based cuckoo search approach. To optimize multiple objectives simultaneously in the presence of job arrivals, An et al. [41] leverages a non-dominated sorting genetic algorithm-III approach. Furthermore, numerous high-performing evolutionary algorithms have been developed to handle FJSP with fuzzy constraints, aiming to achieve high-quality solutions while minimizing energy consumption [37], [42], [58]. A comprehensive review of FJSP methods can be found in [59].

Although these approaches have achieved remarkable success, they fail to take into account the transportation constraints and distributed scheduling environment of FJSPs. In addition, the above methods have difficulty in continuing to generate more diverse solutions in the later stages of evolution, thus making it difficult to meet the flexibility requirements of organizations.

B. *FJSP With Distributed Environment and Transportation Constraints*

With the shift from centralized to distributed production, the DFJSP has emerged as an extension of FJSP to address the added complexity of distributed environments. Various methods have been proposed to solve DFJSP, including an improved genetic algorithm that enhances solution quality by refining the most promising individuals [36]. In [26], four mathematical models of DFJSP were suggested to optimize the makespan. Luo et al. [43] considered job transfers across factories and presented the Memetic Algorithm (MA) to optimize multiple tasks. Later, they addressed worker arrangement constraints and applied an enhanced MA to address the DFJSP [21]. In [22], [39], Deep Q-networks Co-Evolution (DQCE) and a Surprisingly Popular Based Adaptive MA (SPAMA) were proposed for addressing the DFJSP. Additionally, some studies have incorporated both distributed environments and lag time considerations in DFJSP [38], [44]. It should be noted that during transportation and loading process, the machine's state is often assumed to be idle by default, whereas in practice, lag time should be accounted for to improve scheduling accuracy [14], [60].

It is crucial to highlight that, for the above studies, the utilization of domain knowledge and problem features (e.g., distributed factories, the state of machine transportation, and machine idle time) are limited. Addressing these limitations requires more flexible and adaptable optimization approaches that can accommodate the evolving requirements of real-world distributed production systems.

C. *Quality-Diversity Optimization*

As discussed earlier, QD optimization [46], [47] represents a paradigm in evolutionary algorithms, focusing on simultaneously exploring the feature space and optimizing objective functions measured by user-defined metrics [61]. These metrics, known as features or behavior descriptors, are numerical vectors that characterize solution properties and define a corresponding feature space. For example, in robotics, the objective might be speed, while features could include weight and height, forming a three-dimensional space. The goal of QD is to identify the best-performing solution for each region within this feature space. MAP-Elites [49], a widely used QD algorithm, discretizes the feature space into a grid and aims to fill each cell with the highest-quality solution for its corresponding features. Starting from a randomly generated population, solutions are evaluated and placed into the grid. Iteratively, selected solutions undergo mutation to produce new candidates, which are added to the grid if they occupy an empty cell or outperform the current occupant. This process promotes both diversity and local refinement within each cell. Although MAP-Elites has been applied successfully in fields such as robotics, gaming, and software testing, QD optimization remains largely unexplored in job shop scheduling. This paper pioneers its application to DFJSP-T, leveraging solution diversity and adaptability to address the complexity of distributed production environments.

TABLE I
SYMBOL DEFINITIONS OF DFJSP-T

Symbols	Descriptions
k	Index of machine
j	Index of operation
i	Index of job
f	Index of factory
l	Total number of factories
n	Total number of jobs
$n_{\max}$	Total number of operations across all jobs
m	Total number of machines in each factory
w_i	Number of operations associated with job i
F	Set of factories, where $F = \{1, \ldots, f, \ldots, l\}$
I	Set of jobs, where $I = \{1, \ldots, i, \ldots, n\}$
M	Set of machines, where $M = \{1, \ldots, k, \ldots, m\}$
J_i	Set of operations for job i, where $J_i = \{1, \ldots, j, \ldots, w_i\}$
M_f	Set of machines in factory f, $M_f = \{1, \ldots, k, \ldots, m_f\}$
$O_{i,j}$	Operation j in job i
$T_{i,j,f,k}$	Processing time for $O_{i,j}$ on machine k in f
$M_{i,j,f}$	Set of machines available for operation $O_{i,j}$ in f
$TR_{i,j,k,k'}$	Transportation time between machines k and k' for $O_{i,j}$
$C_{i,j}$	Completion time of $O_{i,j}$
C_{max}	Makespan
$S_{f,i,j}$	Start time for operation $O_{i,j}$ in factory f

III. QD Optimization Model for DFJSP-T

This section begins with a description of the DFJSP-T. When applying QD to DFJSP-T, besides the objective function, the definition of the feature space is crucial as it directly affects the quality and diversity of the solutions. However, the selection of features should not be arbitrary and needs to be supported by a specific properties or theories. To justify the use of job transportation and machine idle times as features, we verify three formal properties of the DFJSP-T. On this basis, it lays the theoretical foundation for the subsequent design of algorithmic frameworks.

A. *Problem Description*

To better describe the problem, Table I lists the following symbols. The optimization objective is to minimize the makespan C_{max} (completion time of the last operation). In DFJSP-T, there are n jobs to be processed in l factories with different processing capabilities. All jobs must be assigned to one of the l factories. For each operation j of job i (denoted $O_{i,j}$), a machine k must be selected form a set of available machines within the assigned factory. Each factory is an a FJSP containing m machines. The standard processing time $T_{i,j,f,k}$ of $O_{i,j}$ in factory f on each machine k for different operations is known in advance. Setup times are either negligible or assumed to be included in the processing time. Preemption or interruption of an operation in progress is not allowed. After an operation is completed on a machine, it may need to be transported to another machine within the same factory for further processing. The transportation time $TR_{i,j,k,k'}$ is determined by the distance between machines.

B. Preliminaries for QD Optimization

QD optimization focuses on identifying a set of diverse and high-quality solutions within a given feature space. Let $\mathcal{D}$ and $\mathcal{B}$ represent the decision and feature space, respectively. In this context, the goal is to minimize the objective function $f(x)$. The general mathematical formulation for QD optimization is as follows:

$$\begin{aligned} &\forall\, \boldsymbol{b} \in \mathcal{B},\ x^* = arg \min_{x\in\mathcal{D}} f(x), \\ &s.t.\ \boldsymbol{b}_x = \boldsymbol{b} \end{aligned} \tag{1}$$

where $\boldsymbol{b_x} = (b_1(x), b_2(x), \ldots, b_r(x))$ denotes the feature vector of solution $x \in D$, $b_q(x)$, $q = 1, \ldots, r$ represents the q-th feature; $r \geq 1$ indicates the dimensionality of the feature space. As outlined in [47], QD seeks to identify the optimal solution x^* for each cell within $\mathcal{B}$. For DJFSP-T, $\mathcal{D}$ represents the set of operation sequences, machine selections, and factory allocation orderings. In the following subsections, this paper illustrate how to instantiate the main components of the QD optimization model and apply it to DFJSP-T. This includes specifying the $f(x)$ and defining the feature space $\mathcal{B}$.

$f(x)$ quantifies how the solution x addresses the DFJSP-T. In this study, $f(x)$ is defined as the makespan, which is expressed as follows:

$$f(x) = \max_{i\in\{1,2,\ldots,n\}} C_{i,w_i} \tag{2}$$

where C_{i,w_i} represents the completion time of the final operation w_i in job i. $\max\limits_{i\in\{1,2,\ldots,n\}} C_{i,w_i}$ indicates finding the makespan for the last operation of all jobs.

C. The Defination of Feature Space

The feature space comprises one or more user-defined behaviors that characterize how a solution addresses the problem. In this paper, we define the number of machine idle times (num_idle) and job transfers (num_trans) as the two features constituting the feature space B. These features are selected because the completion time of an operation is jointly influenced by processing time, transportation time, and machine idle time. Since processing time is fixed, variations in completion time primarily result from differences in transportation and idle times caused by scheduling decisions.

Focusing on only one feature, such as machine idle time, may not accurately reflect its influence on completion time. This will be formally validated in the following properties. In general, more job transfers and idle periods tend to increase completion time and prolong the makespan [17]. However, in extreme cases where all operations are assigned to a single machine to minimize transfers, other machines may remain idle, leading to an excessively long makespan. Therefore, considering both machine idle time and job transportation is essential for capturing their combined effect on completion time. We now present and prove the key propositions that establish this relationship.

Property 1. Necessary conditions for job completion: The completion time $C_{i,j}$ depends on both machine idle time and job transportation time. These features are tightly coupled and jointly determine when processing can begin. Neglecting either renders the computation of $C_{i,j}$ incomplete or incorrect.

Proof: We prove by contradiction that machine idle time and job transportation time are jointly necessary for correctly computing the $C_{i,j}$:

- *Ignoring machine availability:* Assume the computation of $O_{i,j}$ excludes machine available time M_t. Let $C_{i,j-1}$ be the completion time of the preceding operation and $TR_{i,j,k,k'}$ be the transportation time to machine k'. If $M_t > C_{i,j-1} + TR_{i,j,k,k'}$, where M_t is the earliest available time of machine k, then the actual start time $S_{i,j}$ must satisfy $S_{i,j} \geq M_t$. Omitting M_t would imply $S_{i,j} = C_{i,j-1} + TR_{i,j,k,k'}$, which violates machine availability. Thus, idle time is essential for determining $C_{i,j}$.
- *Ignoring job transportation:* Assume transportation time $TR_{i,j,k,k'}$ is omitted. This implies that the job is considered instantly available at machine k' upon completion of $O_{i,j-1}$, i.e., $S_{i,j} = C_{i,j-1}$. However, if $TR_{i,j,k,k'} > 0$, this leads to an infeasible start time, since the job has not yet arrived. In reality, $O_{i,j}$ cannot start before $C_{i,j-1} + TR_{i,j,k,k'}$. Therefore, ignoring transportation time violates the constraint $S_{i,j} \geq C_{i,j-1} + TR_{i,j,k,k'}$, making it essential for correctly determining $C_{i,j}$.

Property 2. Sufficient conditions for job completion: Given the machine idle time and job transportation time, along with the known processing time, the completion time $C_{i,j}$ of operation $O_{i,j}$ is uniquely determined. That is, once the earliest available time of the machine and the arrival time of job are known, $C_{i,j}$ can be computed without ambiguity.

Proof: Under the assumptions of non-preemptive scheduling, no setup time, and active scheduling (i.e., operations are not delayed beyond their earliest feasible start), the start time of $O_{i,j}$ is governed only by the two constraints discussed (machine availability and job transportation). Once these constraints are satisfied, $O_{i,j}$ can begin immediately.

Under our assumptions, operation $O_{i,j}$ can begin only when both conditions are satisfied, no additional condition conflicts exist. Thus, the start time is:

$S_{i,j} = \max\{M_t, C_{i,j-1} + TR_{i,j,k,k'}\}$

Given the known $T_{i,j,k}$, the completion time is:

$C_{i,j} = S_{i,j} + T_{i,j,k} = \max\{M_t, C_{i,j-1} + TR_{i,j,k,k'}\} + T_{i,j,k'}$

All components on the right-hand side are either known inputs or derived from prior operations. No other factors affect $S_{i,j}$ under the given assumptions. Hence, machine idle time (through M_t) and job transportation time $TR_{i,j,k,k'}$ are sufficient to uniquely determine $C_{i,j}$.

Property 3. Quantitative relationship for job completion: Given the predecessor completion time $C_{i,j-1}$, machine availability time M_t, and transportation time $TR_{i,j,k,k'}$, the completion time of $O_{i,j}$ can be computed as $C_{i,j} = \max\{M_t, C_{i,j-1} + TR_{i,j,k,k'}\} + T_{i,j,k'}$.

This formula quantitatively shows the contribution of any machine idle time and transportation time to $C_{i,j}$. Let the machine idle time before $O_{i,j}$ be defined as $I_O = \max\{0, (C_{i,j-1} + TR_{i,j,k,k'}) - M_t\}$. The duration which machine k remains idle while waiting for the job to arrive. If the job arrives after the available machine, $I_O > 0$; otherwise, $I_O = 0$. Similarly, define the job waiting time as: $W_O = \max\{0, M_t - (C_{i,j-1} + TR_{i,j,k,k'})\}$, which indicates the time the job waits at machine k' for it to become available. If the machine is already available upon job arrival, then $W_O = 0$. By definition, at most one of I_O or W_O is positive, and they never overlap. Using these quantities, the start and completion time can be reformulated as:

$S_{i,j} = M_t + I_O = (C_{i,j-1} + TR_{i,j,k,k'}) + W_O$,

$C_{i,j} = S_{i,j} + T_{i,j,k'} = M_t + I_O + T_{i,j,k'} = (C_{i,j-1} + TR_{i,j,k,k'}) + W_O + T_{i,j,k'}$.

This equation clearly reveals the dependency structure. If the job arrives after the machine becomes available ($I_O > 0$, $W_O = 0$), then $C_{i,j} = (C_{i,j-1} + TR_{i,j,k,k'}) + T_{i,j,k'}$, and the machine experienced an idle period I_O (which does not delay $C_{i,j}$ beyond the arrival time). Conversely, if the machine becomes available after the job arrives ($W_O > 0, I_O = 0$), then $C_{i,j} = M_t + T_{i,j,k'}$, and the completion is pushed out by the waiting time W_O (the job waits until time M_t). In both cases, any increase in $TR_{i,j,k,k'}$ (longer transportation) directly increases the waiting/idle interval and thus delays $C_{i,j}$, any delay in machine availability (larger M_t relative to $C_{i,j-1} + T_{i,j,k'}$) also directly delays $C_{i,j}$. Quantitatively, this relationship is expressed as:

- *Monotonicity:* $C_{i,j}$ is non-decreasing in both M_t and $TR_{i,j,k,k'}$. An increase in transportation time $TR_{i,j,k,k'}$ either directly increases $C_{i,j}$ (if the machine was waiting) or simply increases I_O (if the machine was free early, $C_{i,j}$ doesn't change until $TR_{i,j,k,k'}$ exceeds $M_t - C_{i,j-1}$). Similarly, a delay in machine free time M_t either directly increases $C_{i,j}$ (if the job is waiting) or increases W_O (if the job arrives early).
- *Additive impact:* The difference between $C_{i,j}$ and the sum $(C_{i,j-1} + T_{i,j,k'})$ (which would be the completion time if there are no transportation or machine idle needed) is exactly the effective waiting time $\max I_O, W_O$ caused by these factors. In fact: $C_{i,j} - (C_{i,j-1} + T_{i,j,k'}) = \max\{M_t - C_{i,j-1}, TR_{i,j,k,k'}\}$, which is either the machine-induced delay or the transportation delay, whichever is larger. This underscores that $C_{i,j}$ is extended beyond $C_{i,j-1} + T_{i,j,k'}$ precisely by the presence of a transport time and/or a machine busy delay.

Proof: The core relationship $C_{i,j} = \max\{M_t, C_{i,j-1} + T_{i,j,k'}\} + T_{i,j,k'}$ is derived earlier from the fundamental scheduling constraints. Here we provide a clear derivation and interpretation:

Derivation of the Formula: By definition, $S_{i,j} = \max\{M_t, C_{i,j-1} + TR_{i,j,k,k'}\}$. This is simply a restatement of the fact that $O_{i,j}$ cannot start until both the machine is ready at M_t and the job has arrived at $C_{i,j-1} + TR_{i,j,k,k'}$. Thus, whichever of these two time is later will decide $S_{i,j}$. Once started, $O_{i,j}$ completes after its processing time $T_{i,j,k'}$, giving $C_{i,j} = S_{i,j} + T_{i,j,k'}$ immediately. This establishes $C_{i,j} = \max\{M_t, C_{i,j-1} + TR_{i,j,k,k'}\} + T_{i,j,k'}$ as claimed.

We introduce I_O and W_O to distinguish two scenarios:

- If $M_t < C_{i,j-1} + TR_{i,j,k,k'}$, the machine is idle first and waits for the job. In this case $I_O = (C_{i,j-1} + TR_{i,j,k,k'}) - M_t > 0$ and $W_O = 0$. The start time $S_{i,j} = C_{i,j-1} + TR_{i,j,k,k'}$ (job arrival), and the machine idle time I_O does not add to the timeline beyond $S_{i,j}$ (it's a gap before $O_{i,j}$ starts). The completion time simplifies to $C_{i,j} = (C_{i,j-1} + TR_{i,j,k,k'} + T_{i,j,k'}$. Here $TR_{i,j,k,k'}$ directly delays $C_{i,j}$ while the machine idle do not further delay completion (it only indicates the unused time of the machine).
- If $M_t \geq C_{i,j-1} + TR_{i,j,k,k'}$, the job arrives first and waits for the machine. In this case $W_O = M_t - (C_{i,j-1} + TR_{i,j,k,k'}) \geq 0$ and $I_O = 0$. The start time $S_{i,j} = M_t$ (machine ready time). The waiting time W_O represents the delay caused by the machine not being available when the job is ready. It directly leads to the delay of $C_{i,j}$. The completion time becomes $C_{i,j} = M_t + T_{i,j,k'}$, which can be rewritten as $C_{i,j} = (C_{i,j-1} + TR_{i,j,k,k'}) + W_O + T_{i,j,k'}$. Here the transportation time $T_{i,j,k'}$ does not delay $C_{i,j}$ beyond M_t (since the job is ready earlier), but the busy processing of machine delays $C_{i,j}$ by W_O.

Combined impact: The formula $C_{i,j} = M_t + I_O + T_{i,j,k'} = (C_{i,j-1} + TR_{i,j,k,k'}) + W_O + T_{i,j,k'}$ unifies both scenarios. It explicitly includes both $TR_{i,j,k,k'}$ and either an idle interval I_O or a waiting interval W_O (one of which will be zero). This gives a quantitative measure of how much later $C_{i,j}$ occurs compared to the ideal case of starting immediately after $O_{i,j-1}$ on an idle machine. As noted, the maximum of M_t and $C_{i,j-1} + TR_{i,j,k,k'}$ determines the delay. For example, if the transportation time $TR_{i,j,k,k'}$ increases while all other factors remain fixed, the completion time $C_{i,j}$ may increase—especially when the machine is available and idle, as the job takes longer to arrive. However, if the machine is the bottleneck (i.e., still busy), $C_{i,j}$ may remain unaffected until the increased transportation time exceeds the original job waiting time W_O. Similarly, if the machine becomes available later (i.e., M_t increases), this leads to a longer W_O if the job has already arrived. But if the transportation time is the bottleneck, $C_{i,j}$ remains unchanged until M_t surpasses the arrival time of the job, which is $C_{i,j-1} + TR_{i,j,k,k'}$. In all cases, $C_{i,j}$ is exactly $T_{i,j,k'}$ plus the larger of the two delay factors $M_t - C_{i,j-1}$ and $TR_{i,j,k,k'}$.

Thus, **Property 3** provides the explicit formula relating $C_{i,j}$ to machine idle and transportation time, and the reasoning above confirms that this formula correctly captures the influence

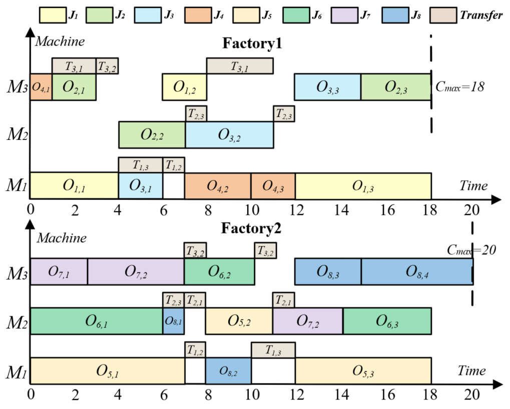


Fig. 1. Gantt chart of the DFJSP-T.

(or lack thereof) of each factor in all cases. This quantitative relationship completes the understanding of how $C_{i,j}$ is determined by these two critical features.

Based on the above properties, we choose with machine idle and job transportation as the key features for constructing the QD optimization problem. For the convenience of discretized representation, we use their times as a measure of feature variance. An unbounded feature space can lead to significant computational overhead, potentially rendering QD optimization intractable. To manage this, we specify lower and upper bounds lb, ub for the feature space $\mathcal{B}$. Specifically, the features num_idle and num_trans are constrained within the range $[lb, ub]$, where $lb = 0$ and $ub = n_{max}$. As a result, the feature space $\mathcal{B}$ is discretized into $(ub\text{-}lb)^2$ grid cells, and the algorithm maintains the best-performing solution in each cell. The values of lb and ub are set manually, depending on the decision maker's preference or the problem scale. For instance, in a problem with 50 operations, the maximum number of machine idle or transportation times cannot exceed 50. Therefore, $ub = 50$ is a reasonable upper bound, while the lower bound remains 0, as neither machine idle time nor job transportation time can be negative.

D. Illustrative Example for QD Optimization

The following example illustrates the problem, helping to clarify the decision space, objective function, and feature space.

Consider a scenario with $l = 2$ factories, where each factory is equipped with a FJSP structure comprising 3 machines. There are $n = 8$ jobs, including operations with a total of $n_{max} = 25$ that need to be processed on machines in these two factories. Fig. 1 illustrates the Gantt chart for the DFJSP-T. Operations for the same job are depicted in the same color, while light brown represents transportation time of jobs. The feature space consists of two features: num_idle and num_trans. From the Gantt chart, num_idle is 8, num_trans is 14. Therefore, the corresponding storage coordinate of this scheduling scheme in the feature space is (8,14). Additionally, the maximum completion time $C_{\max}$ (determined by the critical factory 2) for this schedule is 20.

Algorithm 1: The framework of SADE-QD.

```
    Input: $batch, lb, ub$
    Output: grid $\mathcal{A}(includes\ \mathcal{P}\ and\ \mathcal{X})$
 1: $\mathcal{A} \leftarrow$ Generate_Grid($lb, ub$)
 2: for $i \leftarrow 1$ to $batch$ do
 3:     $x \leftarrow$ random_solution()
 4:     Add_to_grid($\mathcal{P}, \mathcal{X}, x$)
 5: end for
 6: while The stopping criterion is not met do
 7:     for $i \leftarrow 1$ to $batch$ do
 8:         $x' \leftarrow SADE-CM(\mathcal{A})$
            // Apply self-adaptive DE-enhanced collaborative
            mutation strategy to generate diverse and high-quality
            solutions.
 9:         Add_to_grid($\mathcal{P}, \mathcal{X}, x'$)
            // Evaluate and update the grid archive $\mathcal{A}$.
10:     end for
11: end while
```

IV. SADE-QD FOR THE DFJSP-T

Based on the properties derived in the preceding section, this section will construct a feature space using num_idle and num_trans, and design the SADE-QD algorithm framework accordingly. The proposed SADE-QD employs a grid archive to maintain diverse solutions within the bounded feature space $\mathcal{B}$, as defined in Section III-C. $\mathcal{B}$ is discretized into multiple cells, each corresponding to a behavioral niche. A high-level overview of SADE-QD is provided in Algorithm 1, with detailed components explained in the following sections.

A. Initialization

At the beginning of the procedure, the batch size and feature space bounds $[lb, ub]$ are provided as inputs. In Line 1 of Algorithm 1, an empty archive $\mathcal{A}$ (includes performance set $\mathcal{P}$ and coordinate set $\mathcal{X}$) is initialized as a 2D matrix of size $(ub\text{-}lb)^2$, consistent with the discretized feature space defined in Section III-C. All entries are initialized as null, which indicates that no solutions have been stored in the grid. In Lines 3-4, a $batch$ of individuals is randomly generated and added to the grid. Each solution x consists of three concatenated vectors: 1) Operation Sequence (OS, π), 2) Machine Selection (MS, μ), and 3) Factory Assignment (FA, τ). The vectors π and μ share the same length, which equals the total number of operations $n_{\max}$. The vector τ has length n, corresponding to the number of jobs. As illustrated in Fig. 2, the OS vector determines the order in which operations enter the schedule, the MS vector specifies the machine chosen for each operation, and the FA vector assigns each job to a single factory.

Feasibility of solutions is ensured by the encoding and decoding rules. The assigned factory of each job is determined by FA, so all operations belonging to the same job are forced to stay in the same factory. The machine assignment is controlled by MS, and the decoding step only allows machines that can process the corresponding operation. Even when a search operator modifies the machine index, the index is always restricted to the set

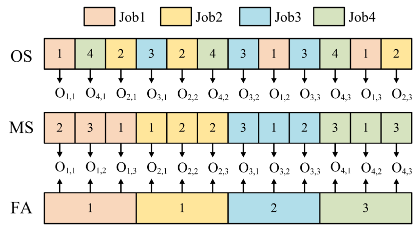


Fig. 2. An example of encoding scheme.

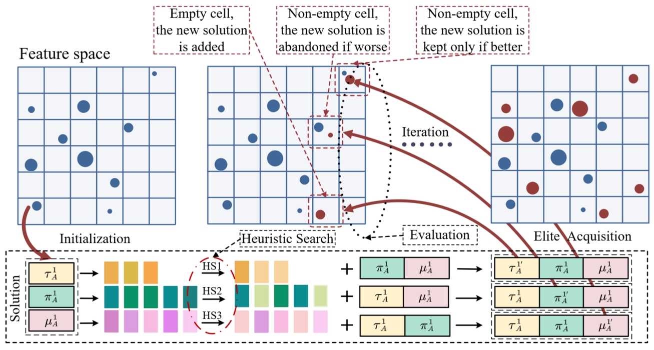

Empty cell, the new solution is added
Non-empty cell, the new solution is abandoned if worse
Non-empty cell, the new solution is kept only if better
Feature space
Iteration
Initialization
Heuristic Search
Evaluation
Elite Acquisition
Solution
HS1
HS2
HS3

Fig. 3. Illustration of three ways of update in grid archive. The figure illustrates three scenarios for updating solutions in the feature space after applying the HS operators. The solution is updated in two scenarios: 1) If the corresponding cell is empty, the newly generated solution is archived directly to the cell. 2) If the targeted cell is already occupied, the new solution will only replace the existing one if it offers a better performance; otherwise, the new solution will be ignored.

of feasible machines. Therefore, the encoding and decoding scheme prevents infeasible schedules. Furthermore, the specific decoding process can be found in the *supplementary materials*.

B. Self-Adaptive Differential Evolution Collaboration and Mutation Strategy

As shown in Lines 8-9 of Algorithm 1, SADE-QD performs the *Self-adaptive Differential Evolution Collaboration and Mutation Strategy* (SADE-CM) to obtain solutions with high quality and diverse behavioral features. In the pre-evolutionary stage, SADE-CM focus on expanding the domain of the feature space using problem-specific *knowledge-assisted heuristic search strategy* (KAHS), whereas in the mid- to late-evolutionary stage, DE is utilized to enhance co-mutation of KAHS.

In the early stage of evolution, the selection of solutions continues to be conducted from the grid archive with diverse behavioral features. This aligns with the original intention of pre-QD to increase the diversity of solutions as much as possible [46]. While expanding the feature space by continuously populating the grid with solutions exhibiting novel behavioral features, we propose the KAHS to enhance the quality of newly generated solutions. The pseudocode of KAHS is shown in Algorithm 2.

The essence of using KAHS is to reduce the length of the *Critical Path*, and thereby reducing the makespan of job sequence. It fully leverages the domain knowledge of DFJSP-T

Algorithm 2: KAHS $(x, \mathcal{P})$.

```
Input: $x, \mathcal{P}$
Output: $x'$
1: $x$ is composed of $\tau$, $\pi$, and $\mu$
2: Allocate jobs to each factory $f$ according to $\tau$
3: Find the critical factory $f\_critical$
4: $ope\_path \leftarrow$ Calculate the Critical Path of $f\_critical$
5: Randomly select one $O_{i,j}$ from $ope\_path$
6: Change the item at the position corresponding to $O_{i,j}$
   with a randomly selected index in one of the $\tau$, $\pi$, and $\mu$
7: A new solution $x'$ is generated
return $x'$
```

Algorithm 3: Add _ to_grid$(\mathcal{P}, \mathcal{X}, x, bool)$.

```
Input: $\mathcal{P}, \mathcal{X}, x, bool$
Output: grid $\mathcal{A}(includes\ \mathcal{P}\ and\ \mathcal{X})$
1: Calculate the objective value $f(x)$ and the
   corresponding cell coordinate $\boldsymbol{b}_x$ of solution $x$
2: if $\mathcal{P}(\boldsymbol{b}_{\boldsymbol{x}}) = \emptyset$ or $\mathcal{P}(\boldsymbol{b}_{\boldsymbol{x}}) > f(x)$ then
3:     $\mathcal{P}(\boldsymbol{b}_{\boldsymbol{x}}) \leftarrow f(x)$
4:     $\mathcal{X}(\boldsymbol{b}_{\boldsymbol{x}}) \leftarrow x$
5: end if
```

to evolve solutions. As shown in Algorithm 2, the process of KAHS involves three knowledge-assisted heuristics (for τ, π, and μ, respectively) based on the *Critical Path*. The *Critical Path* refers to the sequence of operations that has the longest duration. The factory with the longest *Critical Path* is known as the *Critical Factory* [39], [62]. It is crucial that there is no idle time between adjacent critical operations. The duration of the *Critical Path* directly impacts the overall completion time of the final operation. Before executing the three knowledge-assisted heuristics, it is necessary to identify the *Critical Factory* and its corresponding *Critical Path* (Lines 2–4). Then, a critical operation $O_{i,j}$ is selected from the *Critical Path*, and one of its attributes—processing machine, operation sequence, or factory assignment index—is randomly modified (only one is changed per iteration, Lines 5–6). Specifically, if KAHS is applied to π, a operation $O_{i,j}$ is chosen from *Critical Path*, and the positions of $O_{i,j}$ and random operation $O_{i',j'}$ are swapped. If KAHS targets μ, the machine assigned to $O_{i,j}$ is randomly selected from the available machine set. For the τ, the job i associated with $O_{i,j}$ is swapped with a randomly chosen job i' from a non-critical factory. Finally, the newly generated solution is then used to update the current grid archive.

The process of updating the grid archive is straightforward, with details provided in Algorithm 3. Similar to traditional QD, The purpose of this update is to populate all cells in the grid $\mathcal{A}$ with top-performing solutions. In Line 1, the value of $f(x)$ is calculated along with the corresponding cell coordinate $\boldsymbol{b}_x$ of solution x. According to Lines 2-5 in Algorithm 3, if the position in grid is empty, x is directly stored in the grid. Otherwise, x and the existing solution in the grid compare their objective values. If $f(x)$ is better than the existing solution, x replaces it; otherwise,

x is discarded. Fig. 3 visually illustrates the process of updating solutions in the feature space.

An illustration of update is given in Fig. 3, showing how the proposed algorithm takes the boundary and termination conditions of the grid as inputs and outputs a diverse set of solutions. The scheduling decision maker can then select the most appropriate solution from this set. We apply KAHS to τ, π, and μ as examples and illustrate three different update ways in the grid archive. After generations of evolution and updating, the grid archive is gradually populated, which indicates the acquisition of high-quality solutions with diverse behavioral features.

After iterating for thousands or even tens of thousands of generations, most cells in the grid archive become filled. At this stage, a key issue emerges: solutions located at the edges of the grid are less likely to be selected as parents, since their proportion in the population decreases. This leads to stagnation in grid coverage expansion, limiting the discovery of behaviorally diverse solutions. To overcome this, the paper introduces further improvements. Inspired by collaborative DE strategies [48], we propose a self-adaptive DE approach that leverages domain knowledge to help the algorithm select promising solutions located at the edges of the grid archive, guiding the KAHS process for further solution evolution. This enhances the exploration of the algorithm while preserving its exploitation. Experiments in [17], [63] have shown that solutions with fewer job transportation and machine idle times usually possess greater potential for improvement and higher quality. Based on this insight, we design a knowledge-based self-adaptive DE collaboration mechanism, which is responsible for selecting suitable collaborators before updating τ, π, and μ to enhance both solution quality and diversity.

To make the collaboration between KAHS and DE truly adaptive, SADE-QD employs a self-adjusting mechanism based on the evolution status of the grid archive. In early generation, the algorithm monitors two indicators: (1) the increase in grid coverage, and (2) the improvement of objective values within the archive. When both indicators show clear progress, KAHS is preferred because the search still benefits from intensive exploitation around the *Critical Path*. When grid expansion slows down and no quality improvement are observed for a predefined number of iterations, the algorithm automatically leverages DE-guided operators. In this way, KAHS dominates the early stage, while DE collaboration gradually becomes the main driver in the mid and late stages.

The knowledge-based self-adaptive DE collaboration mechanism is shown below:

DE/rand-to-T&I-guided-edge/1

$$\mathbf{x}_{new} = \mathbf{x}_r + F \times (\mathbf{x}_t + \mathbf{x}_i - 2\mathbf{x}_r) \tag{3}$$

DE/rand-to-T&I-guided-edge/2

$$\mathbf{x}_{new} = \mathbf{x}_r + F \times (\mathbf{x}_r - \mathbf{x}_t) \tag{4}$$

DE/rand-to-T&I-guided-edge/3

$$\mathbf{x}_{new} = \mathbf{x}_r + F \times (\mathbf{x}_r - \mathbf{x}_i) \tag{5}$$

where F is the scaling factor. $\mathbf{x}_{new}$ denotes a newly generated solution with guidance. $\mathbf{x}_r$ indicates a random solution from the grid archive. $\mathbf{x}_t$ denotes a solution with minimal job transportation times. To avoid extreme cases, it is randomly selected from the top 20 solutions with the lowest transportation times in each generation. Similarly, $\mathbf{x}_i$ represents a solution with minimal machine idle times and is also randomly chosen from the top 20 solutions with the lowest idle times per generation.

The proposed DE operators are designed to assist KAHS for adjusting τ, π, and μ to improve both the quality and diversity of solutions by explicitly leveraging problem-specific features. As shown in Fig. 4, during DE-guided collaboration, the corresponding vector is extracted from one of the three vector archives (FA, OS, and MS), while the remaining two are randomly selected from the assistant archive within the grid. ***DE/rand-to-T&I-guided-edge/1*** perturbs the factory assignment vector τ, as factory allocation directly affects both job transportation and machine idle times. Suboptimal assignments often cause excessive inter-factory movement, increasing makespan. This operator therefore guides the current solution $\mathbf{x}_r$ toward a promising region by combining the behavioral characteristics of solutions with minimal transportation time ($\mathbf{x}_t$) and minimal idle time ($\mathbf{x}_i$). ***DE/rand-to-T&I-guided-edge/2*** adjusts the operation sequence vector π, which strongly influences transportation patterns. By directing $\mathbf{x}_r$ toward $\mathbf{x}_i$, which exhibits minimal transportation delays, the operator encourages the construction of compact schedules and helps reduce the critical path length. ***DE/rand-to-T&I-guided-edge/3*** refines the machine assignment vector μ, addressing imbalance in machine workloads. Guiding the solution toward $\mathbf{x}_i$ helps alleviate bottlenecks and improves overall machine utilization. After applying the corresponding KAHS adjustment to τ, π, or μ, a new solution is generated. This solution is compared with its associated vector archive, which includes both the current vector and its collaborators (e.g., φ is the collaborator of τ), as well as the assistant archive. If the new solution outperforms the current one, it replaces it in the archive. Together, these guided DE operators enhance grid coverage and promote the generation of high-quality solutions with diverse behaviors.

V. Experiments

This section will provide a detailed performance evaluation of the SADE-QD for DFJSP-T.

A. Benchmarks of DFJSP-T

In real-world multi-factory machining systems, such as those used for metal parts production, jobs are often processed across different factories, and transferring them between sites incurs additional transportation time. Such settings naturally give rise to the DFJSP-T problem. This article therefore abstracts the core decision-making challenges in such environments. Following [39], we constructed all benchmark instances based on this setting. A total number of 20 instances were tested. For each instance, 20 independent experiments were conducted, resulting in a total of 400 runs per algorithm (all experiments used the same instances). The number of factories was chosen

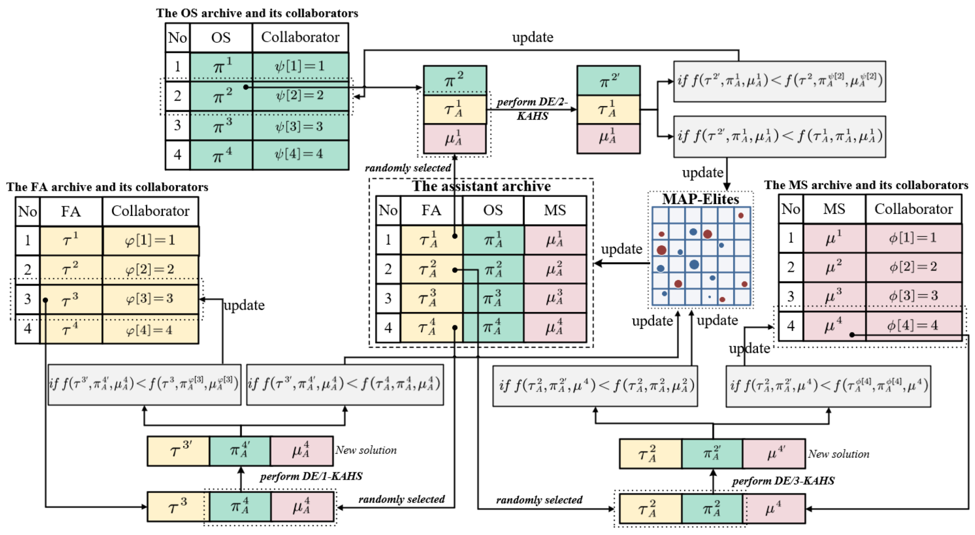


Fig. 4. Collaborative evolution for solutions in the grid archive.

from the set $f \in \{2, 3, 4, 5, 6, 7\}$, with each factory consisting of 5 machines. The total number of jobs, i, varied across the set 10, 20, 30, 40, 50, 100, 150, 200, and for each job i, the number of operations w_i was fixed at 5. The processing time for each operation $T_{i,j,f,k}$ was within the range $[5, 20]$. Machines were considered non-idle due to factors such as machine settings, cleaning, inspection, and operation transportation, as outlined in [64]. Additionally, the maximum evaluations for all tests was: $Max\_Iter = 200 \cdot \sum_1^n w_i$. In practical production scheduling, decision-makers must define appropriate termination conditions. Whether based on the number of evaluations or other performance metrics, these parameters can be flexibly adjusted to meet specific scheduling requirements.

B. Configuration and Evaluation Metric

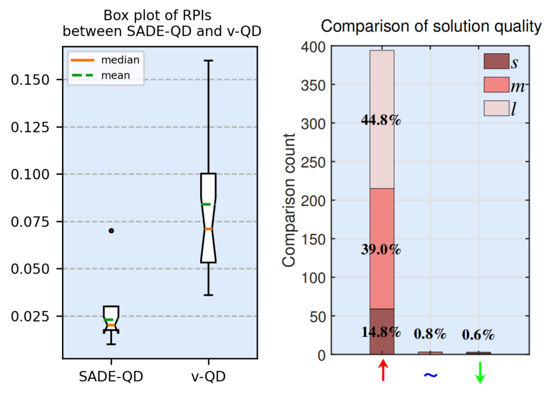
The experiments were performed on a system running a 64-bit Windows 11 OS, equipped with an Intel Core i7-13790F processor clocked at 2.10 GHz and 16.0 GB of RAM. Python was used for the algorithm implementations, with the development carried out in the PyCharm 2023 IDE. We assess the performance of algorithm using the relative percentage increment (RPI) metric [65], [66]. The RPI quantifies the difference between the makespan of the current solution and the best-known solution, defined as:

$$RPI_a = (c_a - c_{best}) / c_{best} \times 100\% \tag{6}$$

where c_a represents the makespan produced by algorithm a, and c_{best} is the minimum makespan found across all algorithms. A lower RPI_a value signifies a better performance by algorithm a.

C. Validation of the QD Optimization for DFJSP-T

The framework of the proposed algorithm is the core concept of this paper. To validate the effectiveness of the QD optimization, we conducted a dedicated experiment to evaluate the performance of the SADE-QD. As shown in Fig. 5, we compare



(a) (b)

Fig. 5. Comparison with and without the use of the QD optimization framework.

the proposed SADE-QD with a variant that does not use the QD optimization framework (stored in a traditional population way and does not take into account the diversity of solution features, denoted as v-QD). As depicted in the RPI box plot of Fig. 5(a), the overall RPI value of the SADE-QD is approximately 64.3% lower than that of v-QD, and both the mean and median are also lower compared to v-QD.

To provide more details on the superiority of SADE-QD over v-QD across all instances, we further performed statistical experiments: job quantities of 10 and 20 are classified as small-scale (s) cases, while 30, 40, and 50 are considered medium-scale (m) cases, and 100, 150, and 200 are designated as large-scale (l) cases (represented by shades of pink indicating varying depths). The analysis of the solutions within feature space involved the following criteria: for solutions better than those of the compared algorithms, we incremented the count indicated by the red "↑\color{red}{$\uparrow$}"; for solutions that were on par with the best solutions of comparison algorithms, we added the tally marked by the blue "∼"; and for cases where solutions obtained

TABLE II
PARAMETER SETTINGS

SADE-QD	IGSA
batch: 100 *cells size*: n_{max}×n_{max} *N*: 2-dimensional	*ps*: 1 *destruction size*: 2 *Temperature*: 0.3

DCGA	SPAMA	DQNMA
ps: 100 *pc*: 0.9 *pm*: 0.15	*ps*: 100 *history memory*: 30 *reward step*: 0.15	*ps*: 100 *batch*: 16 *experience pool*: 512

from SADE-QD were outperformed by the best solutions from comparison algorithms, we added the count designated by the green “↓\color{green}{$\downarrow$}”.

As shown in Fig. 5(b), in 400 runs, approximately 98.6% of the solutions obtained by SADE-QD are better than v-QD. This further validates that incorporating the diversity of solution features to find solutions is far superior to just storing a group of solutions.

The superiority of SADE-QD lies in its capacity to produce a broad range of high-quality solutions, each with distinct features. These diverse solutions provide decision makers with a wider range of choices, which can be better adapted to different machines, transportation equipment, and the various needs of actual scheduling scenarios. SADE-CM strategy guides the evolution of collaborative solutions in the direction of smaller features conducive to solution improvement in the later stages of the iteration. In addition, this characterization of the generated diversified solutions prevents the algorithm from falling into a local optimum to some extent. In the developed QD, the objective function and feature space can be customized according to the specific scheduling environment, enabling the approach to more accurately capture key performance indicators of interest. This flexibility greatly supports the decision-making process.

D. Comparison Experiment and Analysis

This subsection conducted a comparison between SADE-QD and four prevalent algorithms. The selected algorithms address FJSP and related problems: dual population collaborative GA (DCGA) [23], the IG and simulated annealing algorithm (IGSA) [3], SPAMA [22], and DQN based memetic algorithm (DQNMA) [67]. DCGA was chosen because it is based on the classic GA and represents the latest study on solving FJSP with transportation constraints. IG and IGSA have shown strong performance in FJSP, with IGSA being an improved version applied to FJSP with transportation constraints, which are similar to the problem addressed in this paper. SPAMA and DQNMA are modern algorithms for DFJSP with transportation. SPAMA focuses on collaborative solutions, while DQNMA utilizes deep reinforcement learning techniques, representing the state-of-the-art in this field. All algorithms were implemented following their original descriptions, using the encoding and decoding schema of SADE-QD. The objective value calculations were adapted for other algorithms, and their strategies were modified to solve DFJSP-T. Table II presents the main parameters for all compared algorithms. Notably, SADE-QD had fewer parameters compared to the other algorithms. We conducted an independent evaluation of the DE scaling factor F. As shown in Fig. 6,

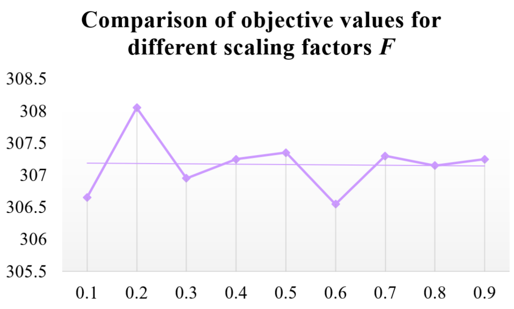


Fig. 6. Comparison of different scaling factor F settings.

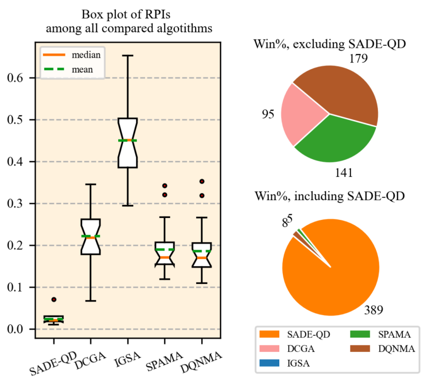


Fig. 7. Comparison among SADE-QD and other algorithms.

the smallest average objective value across all instances was achieved when F=0.6. For all algorithms except IGSA, the batch size, analogous to the population size ps, was fixed at 100.

Table III shows the performance results for all algorithms under comparison. We evaluate the RPI, MEAN, BEST, and standard deviation (STD) metrics. The symbol † denotes a statistically significant difference between SADE-QD and the other algorithms. It is evident from Table III that SADE-QD outperforms other algorithms across all instances, with significant differences observed in each comparison. A box plot and win rate pie charts are utilized to visualize the distribution of solutions based on these results. To assess the statistical significance of the performance differences, a Friedman test with a significance level of $\alpha = 0.05$ was conducted across all instances. As shown in Fig. 7, the boxplots reveal that SADE-QD exhibits the lowest median and mean RPI values among all algorithms, together with the smallest interquartile range. This indicates both superior performance and high consistency. In contrast, the other four algorithms show larger medians, wider spreads, and more outliers, suggesting greater variability and inferior solution quality.

In addition to the boxplots, Fig. 7 also presents two pie charts that summarize the win percentage across all test comparisons. The upper pie chart excludes SADE-QD and reports the win counts among the remaining four algorithms. In this case, no single method dominates: DQNMA wins 179 tests, SPAMA wins 141 tests, and DCGA wins 95 tests, indicating a relatively balanced competition when the proposed algorithm is removed. The lower pie chart incorporates SADE-QD into the comparison.

TABLE III
NUMERICAL RESULTS OF ALL COMPARISON ALGORITHMS

Instance	RPI					MEAN					BEST					STD				
	SADE-QD	DCGA	IGSA	SPAMA	DQNMA	SADE-QD	DCGA	IGSA	SPAMA	DQNMA	SADE-QD	DCGA	IGSA	SPAMA	DQNMA	SADE-QD	DCGA	IGSA	SPAMA	DQNMA
10J2F	**0.07**	0.35†	0.47†	0.34†	0.35†	**97.2**	122.40	133.95	122.15	123.05	**91**	110	122	111	106	**2.745**	7.096	5.862	4.030	5.916
20J2F	**0.03**	0.20†	0.38†	0.19†	0.18†	**180.35**	209.40	241.10	207.85	206.55	**175**	188	223	195	190	**2.254**	13.248	9.375	5.905	6.549
20J3F	**0.03**	0.28†	0.49†	0.26†	0.26†	**129.7**	161.45	187.20	158.45	158.20	**126**	144	177	152	148	**2.250**	8.900	7.317	4.559	5.764
30J2F	**0.03**	0.15†	0.33†	0.14†	0.15†	**261.6**	293.10	338.35	289.65	291.10	**254**	269	311	275	279	**2.854**	13.345	13.743	7.307	5.982
30J3F	**0.03**	0.21†	0.45†	0.16†	0.16†	**182.05**	213.75	255.45	204.50	204.25	**176**	194	236	194	198	**2.685**	9.968	9.417	4.513	4.621
40J2F	**0.03**	0.13†	0.32†	0.14†	0.12†	**340.95**	376.75	439.15	379.70	372.65	**332**	352	403	369	356	**3.927**	12.578	13.812	5.243	7.922
40J3F	**0.02**	0.17†	0.40†	0.14†	0.14†	**240.35**	275.10	328.15	267.05	267.50	**235**	253	314	257	254	**2.961**	9.408	9.544	5.969	6.420
40J4F	**0.02**	0.26†	0.48†	0.16†	0.16†	**187**	229.75	270.25	213.00	212.95	**183**	213	252	204	200	**2.714**	9.031	9.014	3.880	4.407
50J3F	**0.01**	0.16†	0.35†	0.10†	0.10†	**296.2**	338.85	394.00	321.85	322.45	**292**	321	373	305	303	**2.441**	12.500	12.342	9.444	7.997
50J4F	**0.02**	0.21†	0.43†	0.15†	0.13†	**228.65**	272.00	322.85	258.60	255.10	**225**	251	299	246	246	**3.216**	10.819	9.783	5.698	5.139
50J5F	**0.03**	0.24†	0.53†	0.17†	0.17†	**189.35**	228.65	281.00	215.40	215.15	**184**	215	259	208	208	**2.540**	9.337	10.657	3.747	5.363
100J4F	**0.02**	0.14†	0.37†	0.10†	0.08†	**439.65**	494.00	591.20	474.15	468.70	**432**	467	564	458	453	**3.167**	15.871	16.745	8.197	8.367
100J5F	**0.01**	0.16†	0.40†	0.08†	0.09†	**359.05**	411.25	497.35	384.95	385.85	**355**	399	466	373	375	**2.114**	9.678	18.015	5.306	7.206
100J6F	**0.02**	0.24†	0.50†	0.16†	0.16†	**292.55**	356.40	433.25	334.30	333.60	**288**	338	411	314	318	**2.235**	13.068	11.285	7.568	6.402
100J7F	**0.02**	0.16†	0.40†	0.15†	0.15†	**285.35**	323.05	390.85	321.25	320.60	**279**	299	375	308	308	**3.183**	11.014	11.032	6.576	6.652
150J5F	**0.02**	0.18†	0.41†	0.15†	0.13†	**504.4**	587.00	701.65	570.50	562.95	**496**	567	665	558	547	**3.899**	13.992	20.466	8.192	8.835
150J6F	**0.01**	0.07†	0.30†	0.01†	0.01†	**478.75**	507.95	616.20	480.30	481.65	**475**	486	581	457	470	**2.337**	17.647	13.976	8.945	6.226
150J7F	**0.01**	0.09†	0.34†	0.11†	0.10†	**413.6**	447.50	549.25	455.55	451.25	**409**	428	525	439	431	**2.854**	11.086	16.105	10.257	10.417
200J6F	**0.02**	0.06†	0.29†	0.11†	0.10†	**625.6**	652.95	792.10	684.35	678.55	**616**	634	754	642	655	**4.547**	12.784	19.344	15.932	15.000
200J7F	**0.01**	0.09†	0.31†	0.11†	0.11†	**543.85**	586.55	705.20	593.70	594.60	**537**	564	664	566	569	**2.961**	18.429	14.771	13.985	15.816
AVG	**0.023**	0.178	0.398	0.147	0.143	**313.81**	354.39	423.43	346.86	345.34	**308**	334.6	398.7	331.55	330.7	**2.894**	11.990	12.630	7.263	7.550

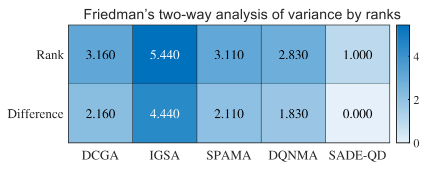


Fig. 8. Friedman Test of all comparative algorithms.

Here, SADE-QD achieves 389 wins in all tests, while the best-performing competitor wins only 8 tests. This overwhelming dominance further confirms that SADE-QD consistently outperforms all competing methods across almost all instances.

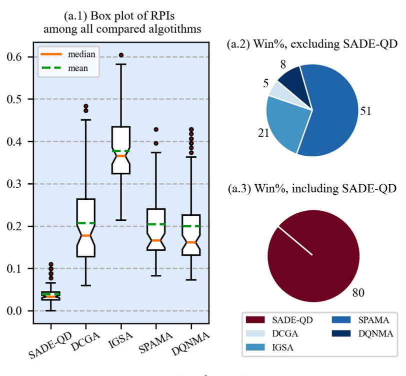
The pairwise p-values obtained from the Friedman test confirm these observations. For each comparison involving SADE-QD, the p-values are below 0.05, indicating statistically significant differences. This reflects that the improvements achieved by SADE-QD are not due to random fluctuations but arise from meaningful algorithmic advantages. Fig. 8 further illustrates the average ranks and differences among all algorithms: SADE-QD consistently ranks 1st, while the remaining algorithms obtain larger ranks. Although DQNMA ranks 2nd, its performance is still statistically worse than SADE-QD ($p < 0.05$).

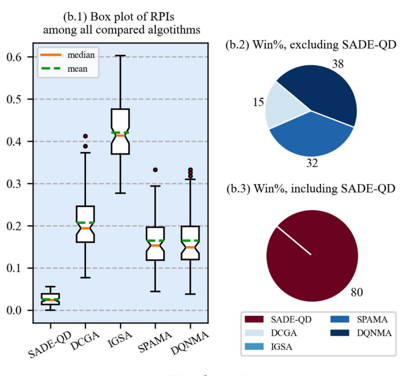
To illustrate the differences between the algorithms from multiple perspectives, error bar charts with a 95% confidence interval are presented. These visual representations offer a deeper insight into the reliability of results. Fig. 9 shows the comparison of RPI values across different numbers of factories for all algorithms, as well as the win rate of SADE-QD. Compared to other algorithms, SADE-QD consistently exhibits lower mean and median values, indicating more stable solutions with fewer outliers. When SADE-QD is excluded from the comparison, DQNMA and SPAMA perform the best. However, when MAP-Elites is included, it consistently produces the optimal solution across all test instances. These findings underscore the effectiveness of the proposed method, highlighting its superior performance and reliability in producing robust and optimal solutions across various scenarios.

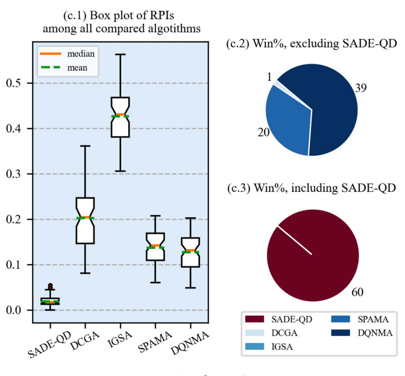
SADE-QD achieves the best performance in terms of RPI, MEAN, BEST, and rank, while also presenting the lowest STD values across runs. These advantages stem not only from the exploration capability of the QD framework but more importantly from the intrinsic synergy introduced by the SADE-CM strategy. In the early evolutionary stage, the critical-path-based KAHS provides knowledge-driven exploitation by selectively modifying τ, π, or μ on critical operations. This directly eliminates scheduling bottlenecks such as long waiting sequences, inefficient factory assignments, or imbalanced machine loads. Such targeted improvements cannot be achieved by random variation operators, allowing SADE-QD to accelerate makespan reduction by systematically shortening the *Critical Path* from the outset.

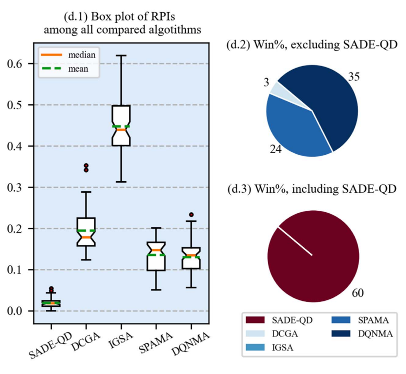
As evolution progresses and the archive becomes densely populated, exploration gradually loses its ability to expand the feature space. This leads to a risk of stagnation. At this stage, the self-adaptive DE collaboration becomes essential. The algorithm uses solutions with minimal transfer and machine idle times as directional guidance. The ***DE/rand-to-T&I-guided-edge***operators introduce controlled and knowledge-based perturbations. These perturbations move solutions toward promising regions. At the same time, they preserve feasibility. This mechanism helps maintain diversity at the boundaries of the feature space. Traditional operators rarely generate offspring in these regions, so the guided DE strategy becomes important. It enables the discovery of novel behavioral patterns and improves grid coverage and solution quality. The alternating cycle of KAHS exploiting the *Critical Path* and DE-guided collaboration expanding the feature-space frontier creates an adaptive balance between exploration and exploitation. This balance explains the superior performance, robustness, and stability of SADE-QD across all benchmark instances.

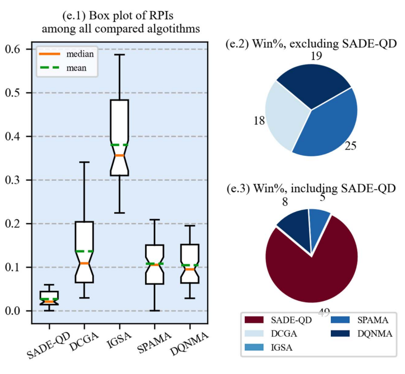
This work explores the main advantage of the proposed algorithm over existing optimization methods: its ability to yield diverse and high-performing solutions. This raises the question: within the feature space, how many solutions outperform those generated by the compared algorithms, and what is their proportion? To this end, we conducted the following statistical experiments: pairwise comparisons between SADE-QD and other algorithms, including DCGA, IGSA, SPAMA, and DQNMA. The experimental setup was the same as in Section V.C. Fig. 10


(a) $f = 2$


(b) $f = 3$


(c) $f = 4$


(d) $f = 5$


(e) $f = 6$

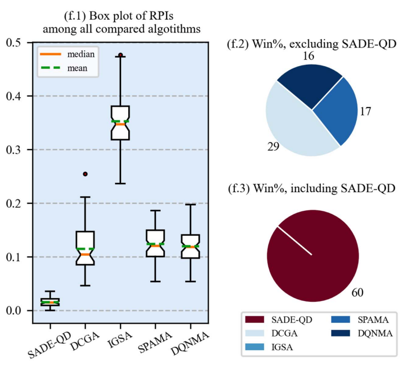
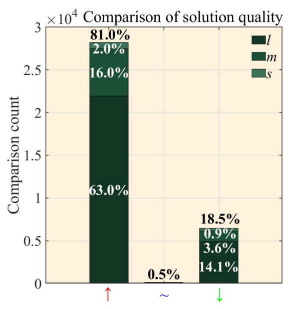
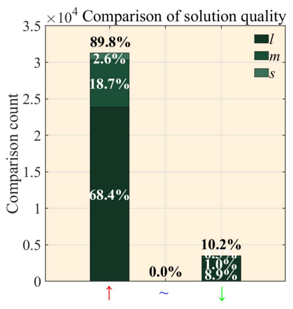
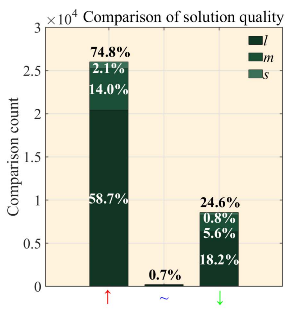

(f) $f = 7$

Fig. 9. Comparative analysis of SADE-QD and other algorithms across different factory settings.


(a) $SADE\text{-}QD\ \ vs\, DCGA$


(b) $SADE\text{-}QD\ \ vs\, IGSA$


(c) $SADE\text{-}QD\ \ vs\, SPAMA$

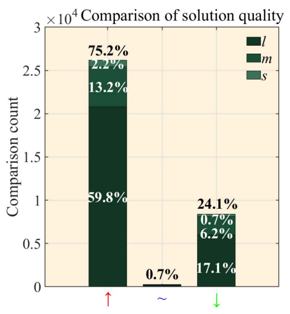
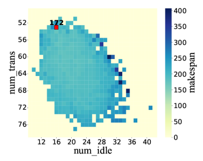
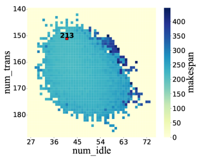

(d) $SADE\text{-}QD\ \ vs\, DQNMA$

Fig. 10. Summary of the statistic for solution quality comparisons between SADE-QD and other algorithms under different scale instances.

shows that within the feature space, the solution set derived by SADE-QD contains 74.8% to 89.8% of solutions with quality superior to the best solutions found by the comparison algorithms, and these solutions exhibit a diverse range of behavioral characteristics. Only 10.2% to 24.6% of the solutions are inferior to the best solutions of the comparison algorithms. This confirms that QD optimization not only yields better solutions but also provides a more diverse range and a greater number of superior solutions.

To illustrate the distribution within feature space and explore how features affect solution quality, we selected three instances—20J2F (small), 50J4F (medium), and 100J6F (large)—and visualized the grid in Fig. 11. The x-axis corresponds to the number of machine idle times, while the y-axis indicates the number of operation transfers. The color intensity of each cell in the grids indicates the corresponding makespan value, with darker colors representing larger makespan values. It is observed that higher numbers of machine idle times or


(a) 20J2F


(b) 50J4F

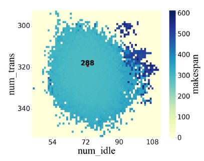

(c) 100J6F

Fig. 11. The grid returned by SADE-QD in three instances.

TABLE IV
NUMERICAL RESULTS OF ALL VARIANTS

Instance	RPI			MEAN			BEST		
	SADE-QD	V-CM	V-DE	SADE-QD	V-CM	V-DE	SADE-QD	V-CM	V-DE
10J2F	**0.064**	0.322†	0.080†	**95.8**	118.95	97.2	91	112	**90**
20J2F	**0.031**	0.157†	**0.031**	175.35	196.7	**175.3**	**170**	186	**170**
20J3F	**0.040**	0.261†	0.069†	**124.85**	151.3	128.3	**120**	143	123
30J2F	0.057	0.151	**0.055**	254.65	277.3	**254.2**	248	266	**241**
30J3F	**0.046**	0.190†	0.050	**176.8**	201.1	177.5	170	189	**169**
40J2F	**0.029**	0.116†	0.036	**333.45**	361.65	335.55	**324**	349	325
40J3F	**0.033**	0.142†	0.044†	**234.5**	259.2	237.1	**227**	247	230
40J4F	**0.041**	0.113†	0.048	**182.15**	194.85	183.35	**175**	189	177
50J3F	**0.033**	0.128†	0.044†	**289.3**	315.8	292.45	283	303	**280**
50J4F	**0.034**	0.080†	0.053†	**223.3**	233.35	227.45	**216**	228	221
50J5F	**0.039**	0.188†	0.065†	**176.7**	202	181	**170**	196	174
100J4F	**0.020**	0.084†	0.035†	**414.2**	440.15	420.4	**406**	428	412
100J5F	**0.031**	0.117†	0.045†	**337.1**	365.1	341.6	331	357	**327**
100J6F	**0.022**	0.120†	0.041†	**287.15**	314.684	292.5	**281**	305	286
100J7F	**0.013**	0.123†	0.036†	**280.5**	310.95	286.95	**277**	306	282
150J5F	**0.012**	0.076†	0.025†	**498.95**	530.4	505.3	494	516	**493**
150J6F	**0.011**	0.206†	0.150†	**421.65**	503	479.6	**417**	493	469
150J7F	**0.011**	0.087†	0.025†	**408.3**	439	413.9	**404**	431	406
200J6F	**0.019**	0.073†	0.033†	**618.75**	651.15	626.95	**612**	644	615
200J7F	**0.017**	0.079†	0.037†	**537.05**	569.5	547.35	**528**	563	534
AVG	**0.030**	0.141	0.050	**303.525**	331.807	310.198	**297.2**	322.55	301.2

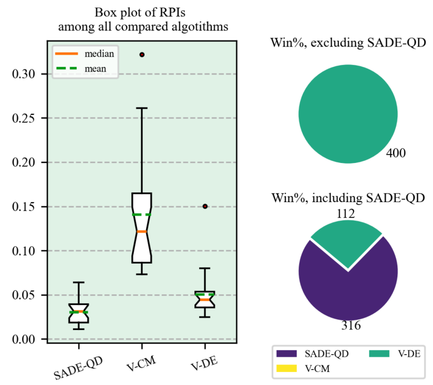


Fig. 12. Comparisons among SADE-QD and its variants.

operation transfers (the right region of the graphs) generally lead to larger makespan values (darker colors). Therefore, devising search strategies that perturb the order of solutions to generate solutions with different feature values can effectively improve the makespan. Additionally, the red squares in the grids indicate the positions of the best objective values. For the small, medium, and large-scale instances, the corresponding minimum makespan values are 172, 213, and 288. These optimal values mainly appear in the upper-left or central region of the grids in Fig. 11, rather than the right region, further supporting our previous conjecture.

E. Ablation Analysis

To evaluate the impact of different components in SADE-QD, we compare it against two variants: 1) V-CM, which omits the *SADE-CM* strategy, and 2) V-DE, which excludes the self-adaptive DE strategy. The results for SADE-QD and its variants are presented in Table IV. Besides RPI, we report the mean (MEAN) and best (BEST) makespan values across 20 runs for each instance. The final row (AVG) represents the overall average across all instances, with the best values are marked in 'bold'. 'J' and 'F' denote the number of jobs and factories, respectively. As indicated in Table IV, SADE-QD consistently surpasses both V-CM and V-DE, achieving 95% (19/20) of the best RPI values, whereas V-CM and V-DE achieve 0% (0/20) and 5% (2/20), respectively. For both MEAN and BEST values, SADE-QD achieves 90% (18/20) and 70% (14/20) of the minimum values, respectively. The AVG values further confirm the superior performance of SADE-QD, yielding the best RPI (0.03), MEAN (303.525), and BEST (297.2) among all algorithms, indicating the effectiveness of both components. To assess whether the performance differences between SADE-QD and its variants are statistically significant, we conducted a Wilcoxon signed-rank test at a significance level of $\alpha = 0.05$. In Table IV, the symbol † indicates that SADE-QD significantly outperforms the variants, with most of p-values below 0.05, confirming the statistical significance of the results.

To illustrate the distribution and performance differences among the solutions, we present box plots and win rate pie charts based on RPI values. As shown in Fig. 12, SADE-QD consistently achieves the best values, demonstrating its superior performance. In the absence of SADE-QD, V-DE wins100% of the outcomes. This indicates that *SADE-CM*, plays a critical role in enhancing the performance of the algorithm. In contrast, removing only the DE strategy (which is primarily responsible for guidance) has a less pronounced impact than *SADE-CM*, but it still contributes to improving the overall performance of the algorithm. Notably, when the complete SADE-QD is incorporated, it outperforms both V-CM and V-DE in approximately 74% of cases, further confirming its effectiveness in optimizing DFJSP-T.

The superior performance of SADE-QD can be attributed to its QD optimization modeling. Incorporating job transfers and machine idle times as feature dimensions within the feature space effectively promotes solution diversity, preventing premature convergence to local optima. QD optimization ensures a broad exploration of the solution space, maintaining diversity throughout the search process. The improvement observed with the inclusion of *SADE-CM* is likely due to its domain knowledge-based DE operators, which facilitate the discovery of diverse and high-performing solutions for DFJSP-T. The combination of diversity maintenance and targeted perturbations applied to the FA, OS, and MS vectors enhances the exploration abilities, leading to superior scheduling outcomes.

F. Discussion

This section presents our findings on the solution update process using SADE-QD. We selected all these instances and recorded the number of successful updates for each feature space cell. A successful update for a cell occurs when a newly generated solution is either placed directly into the cell or replaces an existing solution due to having a better objective value. Successful updates can originate from either the solution within the cell itself or its neighboring solutions. Fig. 13 shows the percentage of successful updates contributed by the solutions themselves and their neighboring solutions. As illustrated in Fig. 13, neighboring solutions contribute significantly more successful updates compared to the solutions themselves. This finding highlights the importance of co-evolution facilitated by QD optimization. One of the primary assumptions of QD is that evolving a set of solutions is generally more effective than

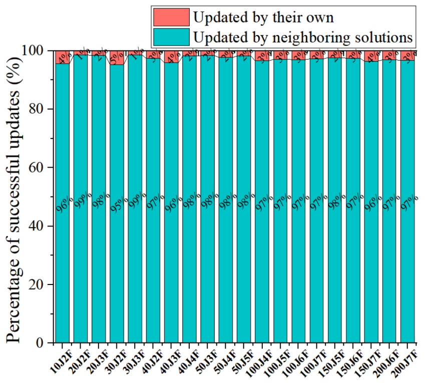


Fig. 13. Percentage of successful updates in all instances.

evolving each solution independently [46]. Our experimental results confirm that this assumption holds true in our context.

VI. Conclusion and Future Work

This paper presents the SADE-QD for addressing the DFJSP-T. Unlike conventional optimization methods, SADE-QD generates a set of high-performing solutions with diverse behaviors, offering greater flexibility for decision-makers. In the early stages of iteration, SADE-QD focuses on exploring solutions with diverse behaviors. In the later stages, once a sufficiently diverse set of solutions has been discovered, the self-adaptive differential evolution strategy intentionally guides the search toward more promising regions of the feature space by leveraging solutions with better-performing features.

The results demonstrate that SADE-QD provides significant advantages over other optimization methods. The solutions obtained by SADE-QD outperform those generated by state-of-the-art algorithms in 74.8% to 89.8% of instances, validating its effectiveness. Furthermore, analysis reveals that neighboring solutions play a key role in successful updates, emphasizing the importance of co-evolution in the optimization process.

The application of SADE-QD optimization in DFJSP-T opens several promising avenues for future research. One potential direction is expanding the feature space to incorporate additional industrial constraints and operational characteristics. Further exploration is needed to assess the scalability of QD optimization in more complex multi-objective scheduling problems. Additionally, future work could investigate the integration of QD with advanced techniques such as evolutionary strategies [68], reinforcement learning [17], [69], and neural networks [70] to further enhance its performance and adaptability in real-world scheduling applications.

References

[1] Z. Pan, L. Wang, J. Zheng, J.-f. Chen, and X. Wang, "A learning-based multi-population evolutionary optimization for flexible job shop scheduling problem with finite transportation resources," *IEEE Trans. Evol. Comput.*, vol. 27, no. 6, pp. 1590–1603, Dec. 2023.

[2] J. Ding, S. Dauzère-Pérès, L. Shen, and Z. Lü, "A novel evolutionary algorithm for energy-efficient scheduling in flexible job shops," *IEEE Trans. Evol. Comput.*, vol. 27, no. 5, pp. 1470–1484, Oct. 2023.

[3] J.-Q. Li et al., "A hybrid iterated greedy algorithm for a crane transportation flexible job shop problem," *IEEE Trans. Autom. Sci. Eng.*, vol. 19, no. 3, pp. 2153–2170, Jul. 2022.

[4] Y. Du, J. Li, C. Li, and P. Duan, "A reinforcement learning approach for flexible job shop scheduling problem with crane transportation and setup times," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 35, no. 4, pp. 5695–5709, Apr. 2024.

[5] 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 Trans. Cybern.*, vol. 49, no. 5, pp. 1944–1955, May 2019.

[6] J.-J. Wang and L. Wang, "A cooperative memetic algorithm with learning-based agent for energy-aware distributed hybrid flow-shop scheduling," *IEEE Trans. Evol. Comput.*, vol. 26, no. 3, pp. 461–475, Jun. 2022.

[7] X.-L. Jing, Q.-K. Pan, L. Gao, and L. Wang, "An effective iterated greedy algorithm for a robust distributed permutation flowshop problem with carryover sequence-dependent setup time," *IEEE Trans. Syst., Man, Cybern. Syst.*, vol. 52, no. 9, pp. 5783–5794, Sep. 2022.

[8] 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 Trans. Evol. Comput.*, vol. 27, no. 3, pp. 610–620, Jun. 2023.

[9] B. Xin et al., "Simultaneous scheduling of processing machines and automated guided vehicles via a multi-view modeling-based hybrid algorithm," *IEEE Trans. Autom. Sci. Eng.*, vol. 21, no. 3, pp. 4753–4767, Jul. 2024.

[10] M. Jensen, "Generating robust and flexible job shop schedules using genetic algorithms," *IEEE Trans. Evol. Comput.*, vol. 7, no. 3, pp. 275–288, Jun. 2003.

[11] Q. Zhang, H. Manier, and M.-A. Manier, "A genetic algorithm with tabu search procedure for flexible job shop scheduling with transportation constraints and bounded processing times," *Comput. Operations Res.*, vol. 39, no. 7, pp. 1713–1723, 2012.

[12] J. Li, Y. Han, K. Gao, X. Xiao, and P. Duan, "Bi-population balancing multi-objective algorithm for fuzzy flexible job shop with energy and transportation," *IEEE Trans. Autom. Sci. Eng.*, vol. 21, no. 3, pp. 4686–4702, Jul. 2024.

[13] 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 Trans. Ind. Informat.*, vol. 20, no. 1, pp. 1007–1018, Jan. 2024.

[14] F. M. Defersha and D. Rooyani, "An efficient two-stage genetic algorithm for a flexible job-shop scheduling problem with sequence dependent attached/detached setup, machine release date and lag-time," *Comput. Ind. Eng.*, vol. 147, 2020, Art. no. 106605.

[15] H. Qin, Y. Xiang, Y. Han, and X. Yan, "Optimizing energy-efficient flexible job shop scheduling with transportation constraints: A Q-learning enhanced quality-diversity algorithm," in *Proc. 6th Int. Conf. Data-Driven Optim. Complex Syst.*, 2024, pp. 373–378.

[16] Z. Pan, L. Wang, J. Wang, Y. Yu, and R. Li, "Distributed energy-efficient flexible manufacturing with assembly and transportation: A knowledge-based bi-hierarchical optimization approach," *IEEE Trans. Autom. Sci. Eng.*, vol. 22, pp. 7463–7479, 2025.

[17] H. Qin, Y. Xiang, F. Liu, Y. Han, and Y. Wang, "Enhancing quality-diversity algorithm by reinforcement learning for flexible job shop scheduling with transportation constraints," *Swarm Evol. Comput.*, vol. 93, 2025, Art. no. 101849. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2210650225000070

[18] M. Dai, D. Tang, A. Giret, and M. A. Salido, "Multi-objective optimization for energy-efficient flexible job shop scheduling problem with transportation constraints," *Robot. Comput.- Integr. Manuf.*, vol. 59, pp. 143–157, 2019.

[19] Y. Yao et al., "A novel mathematical model for the flexible job-shop scheduling problem with limited automated guided vehicles," *IEEE Trans. Autom. Sci. Eng.*, vol. 22, pp. 7449–7462, 2024.

[20] Z. Li and Y. Chen, "Dynamic scheduling of multi-memory process flexible job shop problem based on digital twin," *Comput. Ind. Eng.*, vol. 183, 2023, Art. no. 109498.

[21] Q. Luo, Q. Deng, G. Gong, X. Guo, and X. Liu, "A distributed flexible job shop scheduling problem considering worker arrangement using an improved memetic algorithm," *Expert Syst. Appl.*, vol. 207, 2022, Art. no. 117984.

[22] R. Li, W. Gong, L. Wang, C. Lu, and X. Zhuang, "Surprisingly popular-based adaptive memetic algorithm for energy-efficient distributed flexible job shop scheduling," *IEEE Trans. Cybern.*, vol. 53, no. 12, pp. 8013–8023, Dec. 2023.

[23] X. Han et al., "A dual population collaborative genetic algorithm for solving flexible job shop scheduling problem with AGV," *Swarm Evol. Computation*, vol. 86, 2024, Art. no. 101538.

[24] S. Dauzère-Pérès, J. Ding, L. Shen, and K. Tamssaouet, "The flexible job shop scheduling problem: A review," *Eur. J. Oper. Res.*, vol. 314, pp. 409–432, 2023.

[25] B. Tutumlu and T. Saraç, "A milp model and a hybrid genetic algorithm for flexible job-shop scheduling problem with job-splitting," *Comput. Operations Res.*, vol. 155, 2023, Art. no. 106222.

[26] L. Meng, C. Zhang, Y. Ren, B. Zhang, and C. Lv, "Mixed-integer linear programming and constraint programming formulations for solving distributed flexible job shop scheduling problem," *Comput. Ind. Eng.*, vol. 142, 2020, Art. no. 106347.

[27] L. Meng, C. Zhang, B. Zhang, K. Gao, Y. Ren, and H. Sang, "MILP modeling and optimization of multi-objective flexible job shop scheduling problem with controllable processing times," *Swarm Evol. Computation*, vol. 82, 2023, Art. no. 101374. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2210650223001475

[28] L. Meng, W. Cheng, C. Zhang, K. Gao, B. Zhang, and Y. Ren, "Novel CP models and CP-assisted meta-heuristic algorithm for flexible job shop scheduling benchmark problem with multi-AGV," *IEEE Trans. Syst., Man, Cybern. Syst.*, vol. 55, no. 11, pp. 8455–8468, Nov. 2025.

[29] C. Lin, Z. Cao, and M. Zhou, "Learning-based grey wolf optimizer for stochastic flexible job shop scheduling," *IEEE Trans. Autom. Sci. Eng.*, vol. 19, no. 4, pp. 3659–3671, Oct. 2022.

[30] P. Brandimarte, "Routing and scheduling in a flexible job shop by tabu search," *Ann. Operations Res.*, vol. 41, no. 3, pp. 157–183, 1993.

[31] T. Jamrus, C.-F. Chien, M. Gen, and K. Sethanan, "Hybrid particle swarm optimization combined with genetic operators for flexible job-shop scheduling under uncertain processing time for semiconductor manufacturing," *IEEE Trans. Semicond. Manuf.*, vol. 31, no. 1, pp. 32–41, Feb. 2018.

[32] L. Sun, L. Lin, M. Gen, and H. Li, "A hybrid cooperative coevolution algorithm for fuzzy flexible job shop scheduling," *IEEE Trans. Fuzzy Syst.*, vol. 27, no. 5, pp. 1008–1022, May 2019.

[33] J. Ding, Z. Lü, C.-M. Li, L. Shen, L. Xu, and F. Glover, "A two-individual based evolutionary algorithm for the flexible job shop scheduling problem," in *Proc. AAAI Conf. Artif. Intell.*, vol. 33, no. 1, pp. 2262–2269, 2019.

[34] S. Zhang and S. Wang, "Flexible assembly job-shop scheduling with sequence-dependent setup times and part sharing in a dynamic environment: Constraint programming model, mixed-integer programming model, and dispatching rules," *IEEE Trans. Eng. Manag.*, vol. 65, no. 3, pp. 487–504, Aug. 2018.

[35] Z. Cao, C. Lin, and M. Zhou, "A knowledge-based cuckoo search algorithm to schedule a flexible job shop with sequencing flexibility," *IEEE Trans. Autom. Sci. Eng.*, vol. 18, no. 1, pp. 56–69, Jan. 2021.

[36] L. De Giovanni and F. Pezzella, "An improved genetic algorithm for the distributed and flexible job-shop scheduling problem," *Eur. J. Oper. Res.*, vol. 200, no. 2, pp. 395–408, 2010.

[37] D. Lei, M. Li, and L. Wang, "A two-phase meta-heuristic for multiobjective flexible job shop scheduling problem with total energy consumption threshold," *IEEE Trans. Cybern.*, vol. 49, no. 3, pp. 1097–1109, Mar. 2019.

[38] Z.-Q. Zhang, F.-C. Wu, B. Qian, R. Hu, L. Wang, and H.-P. Jin, "A Q-learning-based hyper-heuristic evolutionary algorithm for the distributed flexible job-shop scheduling problem with crane transportation," *Expert Syst. Appl.*, vol. 234, 2023, Art. no. 121050.

[39] 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 Trans. Syst., Man, Cybern. Syst.*, vol. 54, no. 1, pp. 201–211, Jan. 2024.

[40] I. Kacem, S. Hammadi, and P. Borne, "Approach by localization and multiobjective evolutionary optimization for flexible job-shop scheduling problems," *IEEE Trans. Syst., Man, Cybern., Part C (Appl. Rev.)*, vol. 32, no. 1, pp. 1–13, Feb. 2002.

[41] Y. An, X. Chen, K. Gao, Y. Li, and L. Zhang, "Multiobjective flexible job-shop rescheduling with new job insertion and machine preventive maintenance," *IEEE Trans. Cybern.*, vol. 53, no. 5, pp. 3101–3113, May 2023.

[42] Z. Pan, D. Lei, and L. Wang, "A bi-population evolutionary algorithm with feedback for energy-efficient fuzzy flexible job shop scheduling," *IEEE Trans. Syst., Man, Cybern. Syst.*, vol. 52, no. 8, pp. 5295–5307, Aug. 2022.

[43] Q. Luo, Q. Deng, G. Gong, L. Zhang, W. Han, and K. Li, "An efficient memetic algorithm for distributed flexible job shop scheduling problem with transfers," *Expert Syst. Appl.*, vol. 160, 2020, Art. no. 113721.

[44] Y. Du, J. qing Li, C. Luo, and L. lei Meng, "A hybrid estimation of distribution algorithm for distributed flexible job shop scheduling with crane transportations," *Swarm Evol. Comput.*, vol. 62, 2021, Art. no. 100861.

[45] W. Cheng, L. Meng, B. Zhang, K. Gao, and H. Sang, "Imitation learning-assisted evolutionary algorithm for energy-efficient flexible job shop scheduling problem with automated guided vehicles," *IEEE Trans. Evol. Comput.*, vol. 30, no. 1, pp. 171–185, Feb. 2026.

[46] J. K. Pugh, L. B. Soros, and K. O. Stanley, "Quality diversity: A new frontier for evolutionary computation," *Front. Robot. AI*, vol. 3, 2016, Art. no. 40.

[47] K. Chatzilygeroudis, A. Cully, V. Vassiliades, and J.-B. Mouret, "Quality-diversity optimization: A novel branch of stochastic optimization," in *Black Box Optimization, Machine Learning, and No-Free Lunch Theorems*, P. M. Pardalos, V. Rasskazova, and M. N. Vrahatis, Eds., Cham, Switzerland: Springer, 2021, pp. 109–135.

[48] H. Qin, W. Bai, Y. Xiang, F. Liu, Y. Han, and L. Wang, "A self-adaptive collaborative differential evolution algorithm for solving energy resource management problems in smart grids," *IEEE Trans. Evol. Comput.*, vol. 28, no. 5, pp. 1427–1441, Oct. 2024.

[49] J.-B. Mouret and J. Clune, "Illuminating search spaces by mapping elites," 2015, *arXiv:1504.04909*.

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

[51] L. Grillotti and A. Cully, "Unsupervised behavior discovery with quality-diversity optimization," *IEEE Trans. Evol. Comput.*, vol. 26, no. 6, pp. 1539–1552, Dec. 2022.

[52] A. Ecoffet, J. Huizinga, J. Lehman, K. O. Stanley, and J. Clune, "First return, then explore," *Nature*, vol. 590, no. 7847, pp. 580–586, Feb., doi: 10.1038/s41586-020-03157-9.

[53] J. Schrum, B. Capps, K. Steckel, V. Volz, and S. Risi, "Hybrid encoding for generating large scale game level patterns with local variations," *IEEE Trans. Games*, vol. 15, no. 1, pp. 46–55, Mar. 2023.

[54] Y. Xiang, H. Huang, M. Li, S. Li, and X. Yang, "Looking for novelty in search-based software product line testing," *IEEE Trans. Softw. Eng.*, vol. 48, no. 7, pp. 2317–2338, Jul. 2022.

[55] Y. Xiang, H. Huang, S. Li, M. Li, C. Luo, and X. Yang, "Automated test suite generation for software product lines based on quality-diversity optimization," *ACM Trans. Softw. Eng. Methodol.*, vol. 33, pp. 1–52, Oct. 2023.

[56] Y. Yuan and H. Xu, "Multiobjective flexible job shop scheduling using memetic algorithms," *IEEE Trans. Autom. Sci. Eng.*, vol. 12, no. 1, pp. 336–353, Jan. 2015.

[57] N. Zribi, I. Kacem, A. E. Kamel, and P. Borne, "Assignment and scheduling in flexible job-shops by hierarchical optimization," *IEEE Trans. Syst., Man, Cybern.*, vol. 37, no. 4, pp. 652–661, Jul. 2007.

[58] Y. Du, J.-q. Li, X.-l. Chen, P.-y. Duan, and Q.-k. Pan, "Knowledge-based reinforcement learning and estimation of distribution algorithm for flexible job shop scheduling problem," *IEEE Trans. Emerg. Topics Comput. Intell.*, vol. 7, no. 4, pp. 1036–1050, Aug. 2023.

[59] A. P. A. Agarwal and D. Kumar, "Job shop and flexible job shop scheduling problems: Critical analysis of literature review and their basic technology," *Int. J. Ind. Eng. Des.*, vol. 6, pp. 48–69, Jul. 2020.

[60] Y. Wang and Q. Zhu, "A hybrid genetic algorithm for flexible job shop scheduling problem with sequence-dependent setup times and job lag times," *IEEE Access*, vol. 9, pp. 104864–104873, 2021.

[61] Y. Zhang, M. C. Fontaine, A. K. Hoover, and S. Nikolaidis, "Deep surrogate assisted map-elites for automated hearthstone deckbuilding," in *Proc. Genet. Evol. Comput. Conf.*, 2022, pp. 158–167.

[62] C. Lu, L. Gao, J. Yi, and X. Li, "Energy-efficient scheduling of distributed flow shop with heterogeneous factories: A real-world case from automobile industry in China," *IEEE Trans. Ind. Informat.*, vol. 17, no. 10, pp. 6687–6696, Oct. 2021.

[63] H. Qin, Y. Xiang, Y. Han, Y. Wang, J. Li, and Q. Pan, "A knowledge region selection enhanced quality-diversity algorithm for real-world flexible job shop scheduling with automated guided vehicles transportation," *Eng. Appl. Artif. Intell.*, vol. 164, 2026, Art. no. 113352.

[64] G. U. Umit Bilge, "A time window approach to simultaneous scheduling of machines and material handling system in an FMS," *Operations Res.*, vol. 43, no. 6, pp. 1058–1070, 1995.

[65] H.-X. Qin et al., "An improved iterated greedy algorithm for the energy-efficient blocking hybrid flow shop scheduling problem," *Swarm Evol. Computation*, vol. 69, 2022, Art. no. 100992.

[66] H. Qin, Y. Han, Y. Wang, Y. Liu, J. Li, and Q. Pan, "Intelligent optimization under blocking constraints: A novel iterated greedy algorithm for the

hybrid flow shop group scheduling problem," *Knowl.-Based Syst.*, vol. 258, 2022, Art. no. 109962.

[67] X. Zhou, F. Wang, B. Wu, Y. Li, and N. Shen, "Deep reinforcement learning-based memetic algorithm for solving dynamic distributed green flexible job shop scheduling problem with finite transportation resources," *Swarm Evol. Computation*, vol. 94, 2025, Art. no. 101885.

[68] S. Yang, H. Huang, F. Luo, Y. Xu, and Z. Hao, "Local-diversity evaluation assignment strategy for decomposition-based multiobjective evolutionary algorithm," *IEEE Trans. Syst., Man, Cybern. Syst.*, vol. 53, no. 3, pp. 1697–1709, Mar. 2023.

[69] H. Qin et al., "Energy-efficient iterative greedy algorithm for the distributed hybrid flow shop scheduling with blocking constraints," *IEEE Trans. Emerg. Topics Comput. Intell.*, vol. 7, no. 5, pp. 1442–1457, Oct. 2023.

[70] J.-P. Huang, L. Gao, X.-Y. Li, and C.-J. Zhang, "A novel priority dispatch rule generation method based on graph neural network and reinforcement learning for distributed job-shop scheduling," *J. Manuf. Syst.*, vol. 69, pp. 119–134, 2023.

Haoxiang Qin received the B.S. and M.S. degrees from the School of Computer Science, Liaocheng University, Liaocheng, China. He is currently working toward the Ph.D. degree in software engineering with the School of Software Engineering, South China University of Technology, Guangzhou, China. His research interests include quality-diversity optimization, flexible job shop scheduling, and deep reinforcement learning.

Yi Xiang received the B.Sc. and M.Sc. degrees in mathematics from Guangzhou University, Guangzhou, China, in 2010 and 2013, respectively, and the Ph.D. degree in computer science from Sun Yat-sen University, Guangzhou, in 2018. He is currently an Associate Professor with the School of Software Engineering, South China University of Technology, Guangzhou. His research interests include software product lines, search-based software engineering, and multi-objective optimization.

Yuyan Han received the M.S. degree from the School of Computer Science, Liaocheng University, Liaocheng, China, in 2012, and the Ph.D. degree in control theory and control engineering from the China University of Mining and Technology, Xuzhou, China, in 2016. Since 2016, she has been an Associate Professor with the School of Computer Science, Liaocheng University. She has authored more than 72 refereed papers. Her research interests include evolutionary computation, multiobjective optimization, and flow-shop scheduling.

Yuting Wang received the master's degree in computer software and theory from the China University of Petroleum Beijing, China, in 2005. Since 2013, he has been an Associate Professor with the School of Computer Science, Liaocheng University, Liaocheng, China. He has authored or coauthored more than 20 papers. His research interests include mathematical modeling, intelligent optimization algorithms, and software development technology.

Chunguo Wu received the B.S. and M.S. degrees from the Department of Mathematics, and the Ph.D. degree from the College of Computer Science and Technology, Jilin University, Changchun, China, in 2006. He is currently a Professor with Jilin University. He has authored or coauthored two books, two inventions, and more than 60 articles in *Information Sciences* and *Physical Review E*. His research interests include machine learning, evolutionary computation, adaptive systems, mathematical modeling, intelligent optimization algorithms, and software development technology.

Fujian Feng received the B.S. degree in computer science and technology from the Shandong Technology and Business University, Yantai, China, in 2009, the M.S. degree in probability theory and mathematical statistics from Guizhou Minzu University, Guiyang, China, in 2013, and the Ph.D. degree in software engineering from the South China University of Technology, Guangzhou, China, in 2022. He is currently a Professor with the Guizhou Key Laboratory of Pattern Recognition and Intelligent System, Guizhou Minzu University, Guiyang, China. His research interests include evolutionary computation and micro-computation.