Flexible Job Shop Scheduling via Dual Attention Network-Based Reinforcement Learning

Runqing Wang, Gang Wang, *Senior Member, IEEE*, Jian Sun, *Senior Member, IEEE*, Fang Deng, *Senior Member, IEEE*, and Jie Chen, *Fellow, IEEE*

***Abstract*— Flexible manufacturing has given rise to complex scheduling problems such as the flexible job shop scheduling problem (FJSP). In FJSP, operations can be processed on multiple machines, leading to intricate relationships between operations and machines. Recent works have employed deep reinforcement learning (DRL) to learn priority dispatching rules (PDRs) for solving FJSP. However, the quality of solutions still has room for improvement relative to that by the exact methods such as OR-Tools. To address this issue, this article presents a novel end-to-end learning framework that weds the merits of self-attention models for deep feature extraction and DRL for scalable decision-making. The complex relationships between operations and machines are represented precisely and concisely, for which a dual-attention network (DAN) comprising several interconnected operation message attention blocks and machine message attention blocks is proposed. The DAN exploits the complicated relationships to construct production-adaptive operation and machine features to support high-quality decision-making. Experimental results using synthetic data as well as public benchmarks corroborate that the proposed approach outperforms both traditional PDRs and the state-of-the-art DRL method. Moreover, it achieves results comparable to exact methods in certain cases and demonstrates favorable generalization ability to large-scale and real-world unseen FJSP tasks.**

***Index Terms*— Deep reinforcement learning (DRL), flexible job-shop scheduling, graph attention networks (GATs), self-attention mechanism.**

I. Introduction

INDUSTRY 4.0 is transforming the way companies manufacture, improve, and distribute their products, by moving toward rapid, intelligent, and flexible manufacturing, which leads to a fundamental change in production capabilities of enterprises [3], [21]. The flexible job-shop scheduling problem (FJSP) is a classic problem that represents the typical scenarios faced by flexible manufacturing. FJSP has gained increasing attention in diverse fields, such as cyber–physical manufacturing and intelligent transportation [6], [19], [27]. The job-shop scheduling problem (JSP), which is a simpler and easier instance of FJSP, is also a core issue in the manufacturing industry [44]. JSP involves a set of jobs and machines, where each job consists of multiple operations that must be processed on a given machine in a pre-defined order. The goal is to generate a processing sequence of operations that achieves a meaningful production goal, such as minimum completion time, lateness/tardiness, or production cost. Compared to JSP, FJSP is a more challenging problem as it allows each operation to be processed on multiple different machines. In addition to the operation sequencing problem in JSP, the machine assignment problem adds further manufacturing flexibility, rendering FJSP a more complex topology and a much larger solution space.

In fact, it has been shown that FJSP is a strongly NP-hard problem [39]. The combinatorial nature of FJSP makes it challenging to find (near-)optimal solutions using traditional operation research methods such as constraint programming [22]. These methods have intractable computation costs that increase dramatically with the problem size, making them impractical for large-scale applications [5]. To strike a balance between the solution quality and the computation cost, research in the field has gradually shifted from traditional heuristic and meta-heuristic methods to intelligent methods such as data-driven deep learning [12], [44] and deep reinforcement learning (DRL) [10], [19], [24], [41].

Meta-heuristics, including genetic algorithms [26], [37], [40], particle swarm optimization [43], differential evolution [7], and artificial bee colony [18], have been widely employed for scheduling problems, and they often find high-quality solutions by means of a complex solution search procedure. On the contrary, rule-based heuristics, such as priority dispatching rules (PDRs), are more practical due to their ease of implementation and high efficiency [9]. PDRs repeatedly select the operation or machine with the highest priority based on some prescribed rules until a complete plan is generated. Nonetheless, designing effective PDRs often

Manuscript received 30 March 2023; revised 20 June 2023; accepted 15 August 2023. This work was supported in part by the National Key Research and Development Program of China under Grant 2021YFB1714800; in part by the National Natural Science Foundation of China under Grant 62173034, Grant 61925303, Grant 62025301, and Grant 62088101; in part by the CAAI-Huawei MindSpore Open Fund; and in part by the Chongqing Natural Science Foundation under Grant 2021ZX4100027. *(Corresponding author: Jian Sun.)*

Runqing Wang, Gang Wang, Jian Sun, and Fang Deng are with the National Key Laboratory of Autonomous Intelligent Unmanned Systems, School of Automation, Beijing Institute of Technology, Beijing 100081, China, and also with the Beijing Institute of Technology Chongqing Innovation Center, Chongqing 401120, China (e-mail: rqwang@bit.edu.cn; gangwang@bit.edu.cn; sunjian@bit.edu.cn; dengfang@bit.edu.cn).

Jie Chen is with the Department of Control Science and Engineering, Tongji University, Shanghai 201804, China, and also with the National Key Laboratory of Autonomous Intelligent Unmanned Systems, Beijing Institute of Technology, Beijing 100081, China (e-mail: chenjie@bit.edu.cn).

Color versions of one or more figures in this article are available at https://doi.org/10.1109/TNNLS.2023.3306421.

Digital Object Identifier 10.1109/TNNLS.2023.3306421

requires significant expertise and research effort, and they may only perform well in specific tasks.

DRL methods have emerged as promising approaches to solve JSP and FJSP [36], which model the scheduling process as a Markov decision process (MDP). In these methods, a parameterized neural network model is designed to receive information about the production environment as state and output the priority of each feasible scheduling action, such as assigning an operation to a machine, forming an end-to-end learning approach [35]. Through training on a set of production process data, the DRL model learns to adaptively select the best action at a state to maximize the total reward, which is related to the production goal. However, the effectiveness and efficiency of these methods heavily depend on the design of the state representation, which is a challenging task. As the problem size increases, the amount of production information grows significantly. Therefore, representing this information and complex constraints in the scheduling environment in a sufficiently and minimally expressive manner poses key challenges for designing the state representation. However, the trained DRL-based model is expected to be capable of solving problems of different sizes based on a single unifying architecture. Additionally, the structural information of the problem, which comprises diverse sets of constraints, also plays a critical role in making JSP and FJSP challenging. The priority of operations or machines is strongly related to these constraints, making it essential for the model to appropriately express and exploit them.

Several attempts have been made to address these challenges. One approach is to use aggregated features of operations or machines, such as average machine workload, to represent the production state instead of separately representing each operation or machine, to achieve a uniform representation for problems of varying sizes. For example, this approach is used in [20] to characterize the production status of dynamic partial-no-wait FJSP. They designed 20 extracted features as the state and developed a hierarchical DRL-based framework to learn from these features and choose rules from a set of PDRs for the operation and machine selection independently. Similarly, in [8], a similar approach for state representation is used to describe FJSP with crane transportation and setup times, and a non-fully connected deep Q-network was employed for selecting compound rules to solve this problem. Although this state representation approach is generic for various production scenarios, such as FJSP with additional production constraints, and simplifies the depiction of FJSP, it compresses the information about the production state and makes little use of the structural information.

Alternatively, the well-known disjunctive graph [2] has been widely used in many studies to represent the state of FJSP. This graph-based model represents operations as nodes and uses directed edges between nodes to indicate the processing order of operations, allowing it to describe features of each operation and represent the structural information of FJSP. Recent research has shown promising results in leveraging representation learning methods such as graph neural networks (GNNs) [38] and self-attention mechanisms [32] to handle combinatorial optimization problems with complicated constraints, as these methods have the ability to capture the inherent structure of the problem and are size-agnostic.

Integration of these methods with DRL has been explored in various domains such as vehicle routing [15], chip design [23], and production scheduling [16], and has demonstrated success in solving extremely large-scale problems. In the domain of JSP, for example, a learning framework that incorporates GNN and DRL was proposed in [42] for generating the priority of operations at each step of the scheduling process using a sparse graph as the state representation. Similarly, the work of [25] incorporated dynamic attributes in node features and designed a customized GNN for message passing to learn operation embeddings. In [4], graph embedding techniques were utilized to extract features of the disjunctive graph for downstream decision-making, and an attention-based model was proposed to generate solutions for JSP.

Recent works have extended these solutions to handle FJSP by considering additional machine features and improving decision-making models to handle both operation sequencing and machine assignment. For instance, the work of [17] introduced a two-step decision-making framework that uses a disjunctive graph for learning operation embeddings and a multi-layer perceptron (MLP) for passing machine messages. However, this approach may result in limited exploration during training since the priority of machines is only computed for the selected operation. On the other hand, [31] designed a heterogeneous graph that considers operations and machines as heterogeneous nodes and connects each machine to operations that it can process. They proposed a two-stage GNN for learning machine and operation embeddings and a decision-making model for selecting operations and machines simultaneously.

In general, albeit these DRL-based solutions yield better performance than traditional PDRs, two major challenging problems remain to be tackled for further improving the model's performance and narrowing the gap with exact methods. First, a more precise state representation is required which avoids incorporating information that is irrelevant to decision-making. For instance, completed operations have no contributions to subsequent decision-making at a specific scheduling step, for they will not affect the production status. However, existing methods consider all operations at each step, which may affect the performance and also reduce the efficiency [11]. Second, diverse relationships between operations and machines require to be represented and exploited more rationally. Although existing methods have considered the operation-operation or operation-machine relationships, they have not modeled the relationship between machines yet. The relationship between machines can be viewed as the competition for remaining unscheduled operations which is crucial for discriminating high-priority machines.

In response to these challenges, we propose a novel end-to-end learning framework for standard FJSP aiming to minimize the completion time. First, a tight state representation is introduced in the MDP formulation, which incorporates relevant information about decision-relevant operations and machines. Second, a dual attention network (DAN), comprising several operation and machine message attention blocks, is proposed to express the complicated relationships between operations and machines concisely. The operation message attention blocks learn to extract operation features by simply exploiting their precedence constraints, and the machine message attention blocks are customized for ingeniously capturing and leveraging the competitive relationship between machines, which is overlooked by existing methods. Finally, a size-agnostic decision-making framework is designed to simultaneously address the two critical subproblems in FJSP: operation sequencing and machine assignment. Two different types of information are incorporated as inputs to the policy network to ensure high-quality decision-making. These two types of information are the deep features extracted by the DAN and raw features of compatible operation–machine pairs. Experiment results show that our approach outperforms traditional PDRs and the state-of-the-art DRL method by a considerable margin on two synthetic data with different distributions. The performance is even comparable to exact methods (with time limits) on some tasks. Moreover, it exhibits favorable generalization ability, as models trained on small-scale instances achieve high-quality solutions when directly applied to solve large-scale or out-of-distribution instances.

The contributions of this work are summarized as follows.

1) A tight state representation for describing operations and machines in FJSP that is minimal and sufficient for downstream decision-making with the state space decreasing as scheduling proceeds.
2) A DAN consisting of several operation and machine message attention blocks for deep feature extraction of operations and machines.
3) A size-agnostic DRL-based approach with (markedly) improved performance and generalization capability compared to conventional PDRs and the state-of-the-art DRL method.

The remainder of this article is structured as below. Section II introduces the necessary basics and the FJSP. Section III provides the detailed formulation of the proposed method. Section IV reports the experimental results with Section V concluding this article.

II. Preliminaries and Problem Formulation

A. FJSP and Disjunctive Graph

The FJSP can be formally stated as follows. Consider a set of n jobs and m machines, represented by $\mathcal{J}$ and $\mathcal{M}$, respectively. We assume that all jobs arrive simultaneously at system production time $T_s = 0$. Each job $J_i \in \mathcal{J}$ consists of n_i operations which must be assembled in a specific order (i.e., precedence constraints) described by $\mathcal{O}_i = \{O_{i1}, O_{i2}, \ldots, O_{in_i}\}$. The set of all operations across all jobs is denoted by $\mathcal{O} = \bigcup_i \mathcal{O}_i$. Each operation O_{ij} can be processable by multiple machines, but can only be processed on a single machine from the set of available and compatible machines $\mathcal{M}_{ij} \subseteq \mathcal{M}$. The associated processing time on machine $M_k \in \mathcal{M}_{ij}$ is given by $p_{ij}^k > 0$. FJSP seeks to design a schedule that determines for each operation an appropriate processing machine and the start time while respecting the following constraints: c1) the operations of J_i must be processed in the order in $\mathcal{O}_i$ (i.e., precedence constraints); c2) each operation must be assigned to exactly one compatible machine; and c3) each machine can process at most an operation at a time. The goal of FJSP is to minimize the maximum completion time of all jobs, that is, minimizing the total makespan.

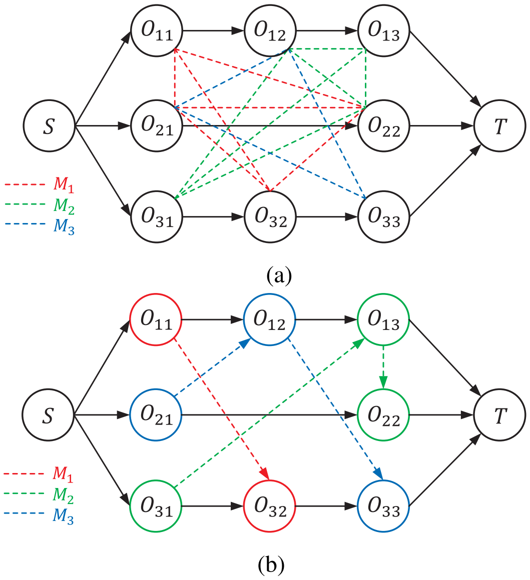


Fig. 1. Disjunctive graph for FJSP. (a) FJSP instance. (b) Possible solution.

The disjunctive graph is a well-documented technique for representing the scheduling problems such as JSP and FJSP. In FJSP, a disjunctive graph G can be described by a tuple $(\mathcal{V} = \mathcal{O} \cup \{\text{Start}, \text{End}\}, \mathcal{C}, \mathcal{D})$, see Fig. 1. The node set $\mathcal{V}$ is composed of all operations (i.e., nodes) and two dummy nodes (signifying the start and completion of production). The edge set $\mathcal{C} = \{\langle O_{ij}, O_{i(j+1)}\rangle | 1 \leq i \leq |J|, 1 \leq j < n_i\}$ comprises all directed edges called conjunctions, which models constraint c1) above, and $\mathcal{D}$ is the set of undirected edges or disjunctions, which consists of m groups, denoted by $\mathcal{D}_1, \ldots, \mathcal{D}_m$, respectively. Disjunctions in group $\mathcal{D}_k$ connect operations that machine M_k can process. Consequently, an operation can be linked to disjunctions in different groups according to their flexibility. A solution to FJSP can be obtained by updating these disjunctions. Once an operation O_{ij} is scheduled to be processed on a machine M_k, all disjunctions linked to O_{ij} except for the ones in $\mathcal{D}_k$ are then deleted. Meanwhile, disjunctions that link scheduled operations are converted into directed edges, whose direction indicates the processing sequence of operations on corresponding machines. As a result, the disjunctive graph becomes a directed acyclic graph when scheduling is completed.

B. Graph Attention Network

Graph attention networks (GATs) [33] are one of the most popular and effective architectures in the field of GNNs. The graph attention layer (GAL) is the core block of GATs which uses attention mechanisms to aggregate neighboring information and compute node embeddings. Roughly speaking, a GAL accepts as input a graph with a set $\mathcal{H} = \{\vec{h}_i \in \mathbb{R}^F\}_{i=1}^N$ of nodal feature vectors and outputs for each node a new feature vector $\vec{h}_i' \in \mathbb{R}^{F'}$.

Letting $\mathcal{N}_i$ collect the first-hop neighbors of node i (including node i itself), a GAL first computes the attention coefficients e_{ij} for each node i and $j \in \mathcal{N}_i$ as follows:

$$e_{ij} = \text{LeakyReLU}\big(\vec{\mathbf{a}}^{\top}\big[(\mathbf{W}\vec{h}_i)\|(\mathbf{W}\vec{h}_j)\big]\big) \tag{1}$$

where $\mathbf{W} \in \mathbb{R}^{F' \times F}$ is a linear transformation shared by all nodes, the operation $[\cdot\|\cdot]$ concatenates two vectors to form a larger vector, and $\vec{\mathbf{a}} \in \mathbb{R}^{2F' \times 1}$ is the weight vector of a single-layer neural network with leaky rectified linear unit (LeakyReLU) as the activation function [34]. These attention coefficients are further normalized using the softmax function as follows:

$$\alpha_{ij} = \text{softmax}_j(e_{ij}) = \frac{\exp(e_{ij})}{\sum_{k \in \mathcal{N}_i} \exp(e_{ik})} \quad \forall i. \tag{2}$$

Finally, the weighted sum of transformed neighboring features using the normalized attention coefficients is computed, which is then fed into a nonlinear activation function $\sigma : \mathbb{R} \to \mathbb{R}$ (e.g., the exponential linear unit [ELU]) to yield a new feature vector

$$\vec{h}'_i = \sigma\left(\sum_{j \in \mathcal{N}_i} \alpha_{ij} \mathbf{W} \vec{h}_j\right) \quad \forall i. \tag{3}$$

A GAT can have multiple GALs connected one by one forward similar to feedforward neural networks. The node embeddings at the last layer are employed in downstream tasks.

III. Proposed Method

In this section, we present our main results including the MDP formulation and a new end-to-end approach for standard FJSP. To this aim, we first formulate FJSP as a MDP by defining its states, actions, state transitions, and rewards. Then, we introduce the proposed learning approach for FJSP, which is termed DAN based reInforcEment Learning and DANIEL for short. It builds on GAT-style self-attention mechanisms and DRL. The workflow of the proposed method is depicted in Fig. 2. Specifically, our framework consists of: 1) a DAN for deep feature extraction of and between operations and machines and 2) a DRL-based decision-making network that considers operation-selection and machine-selection as a whole and outputs a probability distribution prioritizing available operation-machine pairs. At last, we demonstrate how to train the proposed model.

A. MDP Formulation of FJSP

The scheduling process can be understood as dynamically assigning a ready operation to a compatible and idle machine. In this way, the decision point t is the production system time $T_s(t)$ when there is at least a compatible operation-machine pair (O_{ij}, M_k) such that O_{ij} can be processed on machine M_k at time $T_s(t)$. At step t, the DRL model receives a state s_t from the environment and takes an action a_t that assigns a compatible pair to start processing immediately at time $T_s(t)$, for which the FJSP environment returns a reward r_t related to the makespan. A solution of FJSP can be obtained by repeating this procedure $|\mathcal{O}|$ times for the entire set of operations in the task. The MDP is defined as follows.

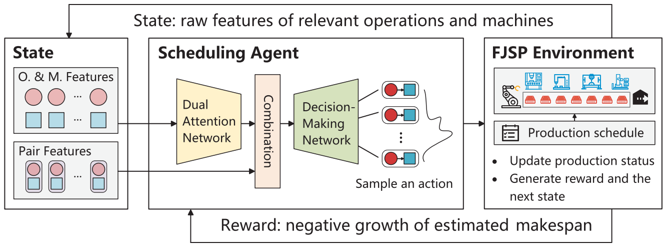


Fig. 2. Overview of the proposed method.

1) State: We propose a tight state representation that sufficiently characterizes the operations and machines relevant to decision-making. Specifically, the operations can be categorized into three groups according to their status at the time, namely completed operations, being processed operations, and unscheduled operations. Since the decision at step t only depends on the production status at system time $T_s(t)$, all operations, except for the completed ones which will not affect subsequent scheduling, are referred to as relevant operations. Likewise, the machines that cannot process any of the remaining unscheduled operations are also termed irrelevant to subsequent scheduling. Therefore, information about irrelevant operations and machines are not meaningful for scheduling and will not be recorded in the state. Let $\mathcal{O}_u(t)$ and $\mathcal{M}_u(t)$ be the set of relevant operations and machines at step t, and $\mathcal{A}(t)$ the set of compatible operation–machine pairs. The state s_t is a set of feature vectors, including $h_{O_{ij}} \in \mathbb{R}^{10}$ for each operation $O_{ij} \in \mathcal{O}_u(t)$, $h_{M_k} \in \mathbb{R}^8$ for each machine $M_k \in \mathcal{M}_u(t)$, and $h_{(O_{ij},M_k)} \in \mathbb{R}^8$ for each compatible operation-machine pair $(O_{ij}, M_k) \in \mathcal{A}(t)$. They are carefully handcrafted to provide a minimal yet sufficient description about the static and dynamic properties of scheduling-relevant entities; see the appendix for more details. Here, pair features provide information about operation and machine combinations (such as the processing time p_{ij}^k), distinct from individual features of operations or machines, which are also crucial for decision-making. It is worth stressing that, different from existing works, the state space gets smaller as the production/scheduling proceeds (until it becomes empty meaning that all operations have been scheduled and processing ends), making the representation computationally appealing for real-time and large-scale applications.

2) Action: The set $\mathcal{A}(t)$ of all compatible operation-machine pairs defines the action space at step t, which becomes smaller as more operations get scheduled for processing across time. We also refer to the operations in $\mathcal{A}(t)$ as the candidates, all of which form the candidate set $J_c(t)$.

3) State Transition: An action a_t corresponds to the processing of an operation on a machine. Upon taking the action, the environment (i.e., the production status of all operations and machines) changes. Then, the sets $\mathcal{O}_u(t)$, $\mathcal{M}_u(t)$ and $\mathcal{A}(t)$ as well as features of relevant entities are updated, and a new state s_{t+1} is obtained.

4) Reward: The reward r_t should be designed to guide the agent to choose actions that are helpful in reducing the maximum completion time of all operations in the task, namely the makespan, denoted by $C_{\max}$. Inspired by the design in [42], we estimate a lower bound of the completion time $\underline{C}(O_{ij}, s_t)$ for each operation O_{ij} at step t. If O_{ij} has been scheduled, the value equals its actual completion time C_{ij} which is accurately known and can be computed. For all unscheduled operations, the lower bound can be approximated by iteratively running the recursion $\underline{C}(O_{ij}, s_t) = \underline{C}(O_{i(j-1)}, s_t) + \min_{k \in M_{ij}} p_{ij}^k$ assuming without loss of generality that $\underline{C}(O_{i0}) = 0$. The estimated maximum completion time $\max_{O_{ij} \in \mathcal{O}} \underline{C}(O_{ij}, s_t)$ over all operations can naturally be used as a measure of the makespan at step t. The reward r_t is defined as the difference between the estimated makespan values at step t and $t+1$

$$r_t = \max_{O_{ij} \in \mathcal{O}} \underline{C}(O_{ij}, s_t) - \max_{O_{ij} \in \mathcal{O}} \underline{C}(O_{ij}, s_{t+1}). \tag{4}$$

When the discounting factor $\gamma = 1$, one can obtain the cumulative reward $\sum_{t=0}^{|\mathcal{O}|-1} r_t = \max_{O_{ij} \in \mathcal{O}} \underline{C}(O_{ij}, s_0) - C_{\max}$, where $\max_{O_{ij} \in \mathcal{O}} \underline{C}(O_{ij}, s_0)$ is a constant value representing the estimated makespan at the initial state. In this way, maximizing the cumulative reward is equivalent to minimizing the makespan.

5) Policy: We employ a stochastic policy $\pi(a_t|s_t)$ whose distribution is generated by an DRL algorithm (an actor–critic network in our simulations) with trainable parameters Θ. Given a state s_t, this distribution returns the probability of choosing each action $a_t \in \mathcal{A}(t)$.

B. Dual Attention Network

Self-attention models can relate elements of input sequences and find the significant ones by exploiting their relationships, which are suitable for finding high-priority operations and/or machines. In addition, self-attention models can handle length-varying sequences (as inputs), which is well-tailored for solving FJSP instances across different scales. These considerations have motivated us to use self-attention models to extract features for operations and machines. The disjunctive graph of FJSP shows that there are multiple types of connections among operations, such as conjunctions and multiple groups of disjunctions. Each type can be seen as a kind of relationship. Therefore, it is inappropriate to relate all operations through a uniform attention model. Moreover, the competitive relationship between machines is not explicitly explored in a disjunctive graph, but it is intuitive that they are crucial for modeling the priority of machines.

To bypass these drawbacks, this article proposes the DAN, an attention-based model for feature extraction of and between operations and machines in FJSP. The overall framework of DAN is shown in 3. DAN decomposes the complex relationships of operations and machines into two parts and handles them separately using two different attention blocks, which are called operation message attention block and machine message attention block, respectively. The former receives operation features as input and simply relates them through precedence constraints. The latter explicitly models the dynamic competitive relationships between machines, which handles the production state of each machine as well as its processing eligibility of the unscheduled operations. The connected two blocks altogether constitute a dual attention layer. Features of operations and machines are refined through L layers in turn, which exhibit similar structures but with a different number of attention heads and parameters. Let ω collect all the parameters of DAN. Details for updating the lth layer's parameters at step t are given as below. In the following, the subscripts t and l are omitted when clear from context.

1) Operation Message Attention Block: This block aims to relate operations in the same job so as to find the most significant operations via their inherent properties. Concretely, for each $O_{ij} \in \mathcal{O}_u$ with input features $h_{O_{ij}} \in \mathbb{R}^{d_o}$, this block models the relationships between O_{ij}, its predecessor $O_{i(j-1)}$, and its successor $O_{i(j+1)}$ (if it exists), by computing their attention coefficients as follows:

$$e_{i,j,p} = \text{LeakyReLU}\big(\vec{\mathbf{a}}^\top\big[(\mathbf{W}h_{O_{ij}}) \| (\mathbf{W}h_{O_{ip}})\big]\big) \tag{5}$$

where $\vec{\mathbf{a}}^\top \in \mathbb{R}^{2d_o'}$ and $\mathbf{W} \in \mathbb{R}^{d_o' \times d_o}$ are linear transformations and for all $|p - j| \leq 1$.

Note that the computations are similar to those in GAT, but we have narrowed its scope. Due to the fact that the predecessor (or successor) of some operations may not exist or will be removed at some step, a dynamic mask on the attention coefficients of these predecessors and successors is performed. A softmax function is employed to normalize all $e_{i,j,p}$ across choices of p, obtaining normalized attention coefficients $\alpha_{i,j,p}$. Finally, the output feature vector $h'_{O_{ij}} \in \mathbb{R}^{d_o'}$ is obtained by a weighted linear combination of transformed input features $\mathbf{W}h_{O_{i(j-1)}}$, $\mathbf{W}h_{O_{ij}}$ and $\mathbf{W}h_{O_{i(j+1)}}$ followed by a nonlinear activation function σ

$$h'_{O_{ij}} = \sigma\Bigg(\sum_{p=j-1}^{j+1} \alpha_{i,j,p} \mathbf{W} h_{O_{\text{ip}}}\Bigg). \tag{6}$$

By connecting multiple operation message attention blocks one by one, the messages of O_{ij} can be propagated to all operations in J_i.

2) Machine Message Attention Block: Two machines compete for the unscheduled operations that they can both process. This competitive relationship may change dynamically as production proceeds. We define C_{kq} as the set of operations that machines M_k and M_q compete for. Let $\mathcal{N}_k = \{q \mid C_{kq} \neq \emptyset\}$ denote the set of competing machines with M_k. Additionally, we employ $c_{kq} = \sum_{O_{ij} \in C_{kq} \cap J_c} h_{O_{ij}}$ as a measure of the intensity of competition between M_k and M_q, which makes sense as the competition becomes more severe when their competed candidates are more important ($h_{O_{ij}}$ is simplified from $h_{O_{ij}}^{(l)}$). The machine message attention block leverages c_{kq} to compute the attention coefficients u_{kq} (The "filter" operator in Fig. 3 is used for obtaining c_{kq} for each M_k and M_q). For each $M_k \in \mathcal{M}_u$ with input features $h_{M_k} \in \mathbb{R}^{d_m}$, the attention coefficients u_{kq} for all $q \in \mathcal{N}_k$ are computed as follows:

$$u_{kq} = \text{LeakyReLU}\big(\vec{\mathbf{b}}^\top\big[(\mathbf{Z}^1 h_{M_k}) \| (\mathbf{Z}^1 h_{M_q}) \| (\mathbf{Z}^2 c_{kq})\big]\big) \tag{7}$$

where $\mathbf{Z}^1 \in \mathbb{R}^{d_m' \times d_m}$ and $\mathbf{Z}^2 \in \mathbb{R}^{d_m' \times d_o}$ are weight matrices and $\vec{\mathbf{b}}^\top \in \mathbb{R}^{3d_m'}$ is a linear transformation.

Note that C_{kk} is the set of unscheduled operations that M_k can process and k is always in $\mathcal{N}_k$. Therefore, (7) can be applied to compute u_{kk}, for c_{kk} can be seen as a measure of the processing capacity of M_k. When $C_{kq} \cap J_c$ is empty, we fill c_{kq} by zeros. Then, analogous softmax normalization (across

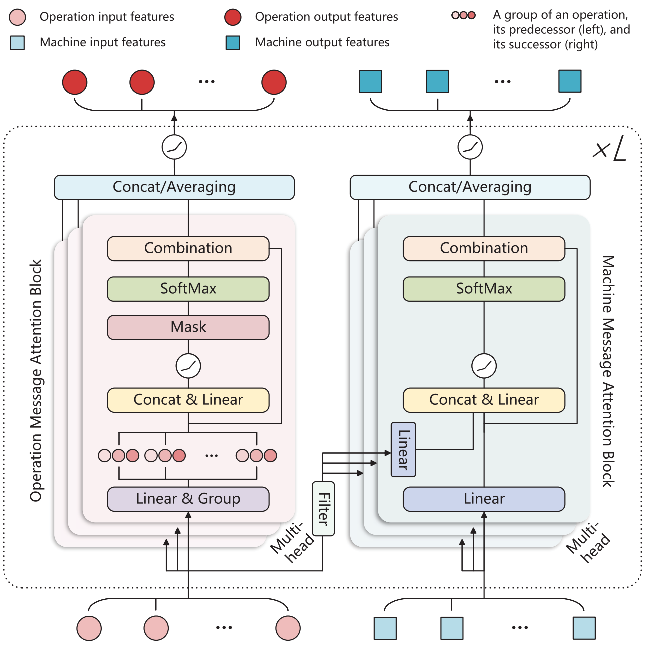

Operation input features
Machine input features
Operation output features
Machine output features
A group of an operation, its predecessor (left), and its successor (right)
×L
Concat/Averaging
Combination
SoftMax
Mask
Concat & Linear
Linear & Group
Filter
Linear
Multi-head
Operation Message Attention Block
Machine Message Attention Block

Fig. 3. Architecture of the DAN.

choices of q), combination, and activation steps are conducted to obtain the output features $h'_{M_k} \in \mathbb{R}^{d'_m}$.

3) Multi-Head Attention: We apply multiple attention heads to both blocks for the purpose of learning a variety of relationships between entities. This technique has been shown effective in strengthening the learning ability of attention models [32]. We specify its use in the operation message attention block, which is the same for the machine message attention block and thus omitted for brevity. Let H be the number of attention heads in a dual attention layer and H attention mechanisms with different parameters are applied. First, the computation of attention coefficients and the combination are performed in parallel. Then, their outputs are fed into an aggregation operator for integration. Adopted from GAT, the aggregation operator refers to a concatenation except for the last layer which uses an averaging operator. Finally, the activation function σ is applied to obtain the layer's output.

4) Pooling: Let the raw (handcrafted) features of operation O_{ij} and machine M_k denoted by $h_{O_{ij}}^{(0)}$ and $h_{M_k}^{(0)}$, respectively. Upon passing $h_{O_{ij}}^{(0)}$ and $h_{M_k}^{(0)}$ through L dual attention layers, the learned features $h_{O_{ij}}^{(L)}$ and $h_{M_k}^{(L)}$ are ready for downstream decision-making tasks. Similar to [15], we first apply an averaging pooling to the operation features and machine features separately, whose results are concatenated to form the global features of an FJSP instance, that is,

$$h_G^{(L)} = \left[\left(\frac{1}{|\mathcal{O}_u|}\sum_{O_{ij}\in\mathcal{O}_u} h_{O_{ij}}^{(L)}\right)\middle\|\left(\frac{1}{|\mathcal{M}_u|}\sum_{M_k\in\mathcal{M}_u} h_{M_k}^{(L)}\right)\right]. \tag{8}$$

5) Analysis: Compared with the prior art [17], [31], our method has advantages in the following aspects. First, we model the time-varying competition between machines explicitly use and rely on this relationship to construct machine features. On top of this, we employ the sum of processable candidate features to represent the intensity of competition and consider its influence on the machine's priority. Moreover, our design incurs a much smaller computational overhead, since the learning process can be understood as handling two (much) smaller graphs, one for operations and the other for machines. The graph for operations can be seen as a modified disjunctive graph with the node set $\mathcal{O}_u(t)$, where conjunctions are maintained but replaced by undirected edges and disjunctions are removed. The graph for machines is used for describing information incorporated in disjunctions, which takes compatible machines as nodes and characterizes their competitions by means of incorporating dynamic edge features. Such a representation features simplicity as well as intuition, for the complex connections have been grouped into two categories with $O(3|\mathcal{O}| + |\mathcal{M}|^2)$ connections in total. Since $|\mathcal{O}|$ is often much larger than $|\mathcal{M}|$ in practice, the number of connections is significantly reduced relative to existing disjunctive graph-based results [17] and [31]. Last but not least, irrelevant operations and machines get removed as scheduling proceeds, contributing to a decreasing state space and bringing computational efficiency.

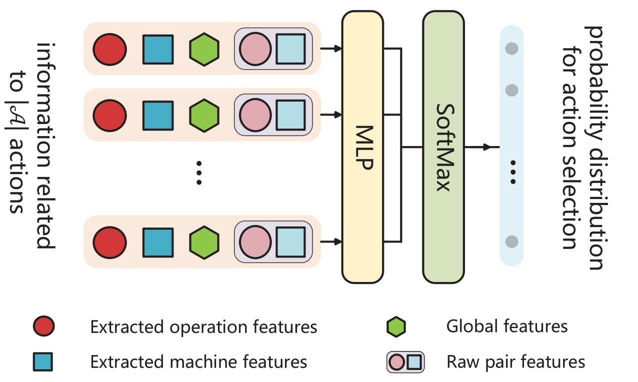

information related to |𝒜| actions
MLP
SoftMax
probability distribution for action selection
Extracted operation features
Extracted machine features
Global features
Raw pair features

Fig. 4. Architecture of the actor network.

C. Decision-Making Module

We design the decision-making network based on actor–critic RL which maintains the size-agnostic property of attention models. Two MLPs are used as the actor and critic, whose parameters are denoted by θ and ϕ, respectively. The actor-network aims to generate a stochastic policy $\pi_\theta(a_t|s_t)$ in two steps: it first produces a scalar $\mu(a_t|s_t)$ for each $a_t \in \mathcal{A}(t)$ and uses the softmax function to output the desired distribution, see Fig. 4. We concatenate all information related to $a_t = (O_{ij}, M_k)$ (including the extracted features of O_{ij} and M_k, the global features, and the compatible operation–machine pair features) in a single vector which is subsequently fed into MLP_θ to yield

$$\mu(a_t|s_t) = \text{MLP}_\theta\left[h_{O_{ij}}^{(L)}\middle\|h_{M_k}^{(L)}\middle\|h_G^{(L)}\middle\|h_{(O_{ij},M_k)}\right]. \tag{9}$$

Then, the probability of choosing action a_t is given by

$$\pi_\theta(a_t \mid s_t) = \frac{\exp(\mu(a_t|s_t))}{\sum_{b_t\in\mathcal{A}(t)}\exp(\mu(b_t|s_t))}. \tag{10}$$

Algorithm 1 Training DANIEL for FJSP

```
Input: DANIEL with initial parameters $\Theta = \{\omega, \theta, \phi\}$, behavior actor network $\theta_{\text{old}}$, pre-sampled training data $\mathcal{E}_{\text{tr}}$, and fixed validation data $\mathcal{E}_{\text{val}}$;
for $n_{\text{ep}} = 1, 2, \ldots, N_{\text{ep}}$ do
    $\theta_{\text{old}} \leftarrow \theta$
    for $i = 1, 2, \ldots, |\mathcal{E}_{\text{tr}}|$ do
        for $t = 0, 1, \ldots, T$ do
            Sample $a_{i,t}$ using $\pi_{\theta_{\text{old}}}(\cdot \mid s_{i,t})$;
            Receive the reward $r_{i,t}$ and the new state $s_{i,t+1}$;
            Collect the transition $(s_{i,t}, a_{i,t}, r_{i,t}, s_{i,t+1})$;
            Update $s_{i,t} \leftarrow s_{i,t+1}$;
        end for
        Compute the generalized advantage estimates $\hat{A}_{i,t}$ for $t = 0, 1, \ldots, T$ using collected transitions;
    end for
    for $k = 1, 2, \ldots, K$ do
        Compute the total loss $\sum_{i=1}^{|\mathcal{E}_{\text{tr}}|} \ell_i^{\text{PPO}}(\Theta)$;
        Update all parameters $\Theta$;
    end for
    if Every $N_r$ episodes then
        Resample $|\mathcal{E}_{\text{tr}}|$ instances to form the training data;
    end if
    if Every $N_{\text{val}}$ episodes then
        Validate $\pi_\theta$ on $\mathcal{E}_{\text{val}}$;
    end if
end for
```

The critic network is an estimator that takes the global features $h_G^{(L)}$ as input to produce a scalar $v_\phi(s_t)$, as an estimate of the state value.

D. Training Procedure

We employ the proximal policy optimization with the clipped surrogate objective (PPO-Clip) [29] to train the proposed scheduling model. The generalized advantage estimation technique [28] is utilized to enhance the performance of DANIEL. Two normalization operators are implemented to stabilize the training process. One is to divide each processing time p_{ij}^k by $\max_{i,j,k} p_{ij}^k$, which is the maximum processing time. The other is batch normalization across the collected advantages of the same instance within each training iteration. The training procedure is summarized in Algorithm 1. We train the model using N_{ep} episodes. For each episode n_{ep}, the agent interacts with a batch of same-scale FJSP environments $\mathcal{E}_{\text{tr}}$ in parallel and collects the transition data, which are used for updating the model parameters Θ. The environments are resampled every N_r episodes, according to a fixed distribution. The policy is validated on some fixed validation data $\mathcal{E}_{\text{val}}$ (with the same distribution as the training data) every N_{val} episodes. Two strategies for the action-selection are considered in our experiments. One is a greedy strategy that always chooses the action with the highest probability $\pi_\theta(a_t|s_t)$, used in validation. The other is an action-sampling strategy, i.e., sampling an action from the distribution π_θ, used during training for sufficient exploration.

IV. Experiments

We numerically compare the proposed DANIEL algorithm with several baselines including several popular PDRs, the exact method Google OR-Tools,[1] a genetic algorithm [26], and the DRL-based method [31] in this section. Both synthetically generated instances as well as popularly used FJSP benchmarks are used to verify its effectiveness in scheduling performance, generalization capability, and computational efficiency.

A. Datasets

An FJSP instance with n jobs and m machines is denoted as "$n \times m$" in short hereafter (the number of operations varies in different instances). We consider two types of synthetic data with distinct distributions to examine the learning as well as the generalization performance of the proposed DANIEL algorithm.

The first is adapted from [31], which allows jobs to have a varying number of operations, denoted by SD_1 in Table I. We refer interested readers to the original text [31] for details. The second is a task with a wider range for the random processing time of operations, denoted by SD_2 in Table I. Specifically, for an $n \times m$ FJSP instance generated for SD_2, each job has m operations. For each O_{ij}, the values $|\mathcal{M}_{ij}|$ and p_{ij}^k are integers sampled from $U(1, m)$ and $U(1, 99)$, respectively. For a fair comparison with the approach in [31], we consider six different scales of FJSP instances using the two data: 10×5, 20×5, 15×10, 20×10, 30×10, and 40×10. We train our model on four smaller sizes of each data with randomly generated instances (eight models in total), while testing is performed on two extra larger ones in addition to those. Both the testing data (unseen) and the validation data are generated in advance each having 100 instances. Furthermore, we evaluate the trained models on four groups of public benchmarks to explore their capabilities on cross-distribution tasks, including *mk*1-10 instances in [2] and three groups of *la*1-40 instances (except sdata) in [13].

B. Configurations

The training configurations were kept the same as in [31] for comparison, with $N_{\text{ep}} = 1,000$, $|\mathcal{E}_{\text{tr}}|$, $N_r = 20$, and $N_{\text{val}} = 10$. Hyperparameters of algorithms were tuned on SD_2 instances of size 10×5 and kept the same for all eight models. For a trade-off between performance and computational efficiency, our DAN used $L = 2$ dual attention layers and each block used $H = 4$ attention heads per layer with ELU as activation function σ. The output dimensions of each attention head in the two blocks are $d_o^{(1)} = d_m^{(1)} = 32$ for the first layer and $d_o^{(2)} = d_m^{(2)} = 8$ for the second layer. Both MLP_θ and MLP_ϕ have two hidden layers of dimension 64 with tanh as activation. For the PPO parameters, the policy, value, and entropy coefficient in the loss function were set to be 1, 0.5, and 0.01. The clipping parameter ϵ, GAE parameter λ, and discounting factor γ were set to be 0.2, 0.98, and 1, respectively. During training, we updated the network for

[1] https://developers.google.cn/optimization

TABLE I
Performance on Two Synthetic Data of Small to Medium Training Size

			PDRs				Greedy strategy		Sampling strategy		
	Size		FIFO	MOPNR	SPT	MWKR	[31]	DANIEL	[31]	DANIEL	OR-Tools[1]
SD_1	10 × 5	Objective	119.40	115.38	129.82	113.23	111.67	**106.71**	105.59	**101.67**	96.32 (5%)
		Gap	24.06%	19.87%	34.76%	17.58%	16.03%	**10.87%**	9.66%	**5.57%**	
		Time (s)	0.16	0.16	0.16	0.16	0.45	0.45	1.11	0.74	
	20 × 5	Objective	216.08	214.16	230.48	209.78	211.22	**197.56**	207.53	**192.78**	188.15 (0%)
		Gap	14.87%	13.85%	22.56%	11.51%	12.27%	**5.03%**	10.31%	**2.46%**	
		Time (s)	0.32	0.32	0.32	0.32	0.90	0.94	2.36	1.87	
	15 × 10	Objective	184.55	173.15	198.33	171.25	166.92	**161.28**	160.86	**153.22**	143.53 (7%)
		Gap	28.65%	20.68%	38.22%	19.41%	16.33%	**12.42%**	12.13%	**6.79%**	
		Time (s)	0.51	0.51	0.50	0.50	1.43	1.35	3.98	3.89	
	20 × 10	Objective	233.48	219.80	255.17	216.11	215.78	**198.50**	214.81	**193.91**	195.98 (0%)
		Gap	19.22%	12.20%	30.25%	10.30%	10.15%	**1.31%**	9.64%	**-1.03%**	
		Time (s)	0.71	0.71	0.71	0.71	1.91	1.85	6.23	6.35	
SD_2	10 × 5	Objective	569.41	557.48	514.39	549.28	553.61	**408.40**	483.90	**366.74**	326.24 (96%)
		Gap	76.47%	72.52%	57.96%	70.01%	71.42%	**25.68%**	49.71%	**12.57%**	
		Time (s)	0.16	0.16	0.16	0.16	0.46	0.44	1.11	0.88	
	20 × 5	Objective	1045.83	1045.94	835.94	1026.03	1059.04	**671.03**	962.90	**629.94**	602.04 (0%)
		Gap	74.59%	74.58%	38.91%	71.31%	76.79%	**11.52%**	60.70%	**4.66%**	
		Time (s)	0.33	0.33	0.33	0.33	0.94	0.90	2.37	1.84	
	15 × 10	Objective	871.14	845.16	703.07	830.53	807.47	**591.21**	756.07	**521.83**	377.17 (28%)
		Gap	132.23%	125.32%	86.74%	121.45%	115.26%	**57.16%**	101.52%	**38.70%**	
		Time (s)	0.52	0.51	0.51	0.51	1.50	1.36	4.16	3.83	
	20 × 10	Objective	1088.05	1059.68	829.14	1040.69	1045.82	**610.16**	990.37	**552.64**	464.16 (1%)
		Gap	135.27%	129.09%	78.82%	124.98%	126.12%	**31.58%**	114.15%	**19.13%**	
		Time (s)	0.70	0.70	0.70	0.71	1.95	1.79	6.03	5.97	

[1] For OR-Tools, the solution and the ratio of optimally solved instances are reported.

$K = 4$ epochs per episode via the Adam optimizer [14] with the learning rate $\mathrm{lr} = 3 \times 10^{-4}$. All experiments were carried out on a machine with an Intel Xeon Platinum 8163 CPU and a single NVIDIA Tesla T4 GPU. The code is available.[2]

C. Baselines and Performance Metrics

First, we selected four PDRs for operation sequencing that have been shown to yield good performance in [30] and generalized them to solve FJSP. In the experiments, we reported the results averaged over five independent runs for each PDR due to their stochastic nature. Specifics regarding the implementation are stated as follows.

1) *First in First Out (FIFO):* selecting the earliest ready candidate operation and the earliest ready compatible machine.
2) *Most Operations Remaining (MOPNR):* selecting the candidate operation with the most remaining successors and a machine that can immediately process it.
3) *Shortest Processing Time (SPT):* selecting the compatible operation–machine pair with the SPT.
4) *Most Work Remaining (MWKR):* selecting the candidate operation with the most remaining successor average processing time and a machine that can immediately process it.

Second, the results from OR-Tools and an advanced genetic algorithm were chosen as a reference line for time-consuming but high-performance methods. As an authoritative constrained programming solver, OR-Tools was employed to generate (near-)optimal solutions with 1800 s set as the time limit. In [26], a two-stage genetic algorithm (2SGA) was designed to solve FJSP with improved performance and efficiency than regular genetic algorithms, whose results on two groups of benchmarks were directly imported due to a lack of open-source implementations. Last but not least, we compared DANIEL with the state-of-the-art DRL method recently proposed in [31]. We used their open-source code[3] to perform training and testing following their default settings. We directly imported their results on SD_1, including the trained models and validation instances of four smaller sizes as well as testing instances of all six sizes. We reported the results of DANIEL and DRL for both greedy and sampling-based action-selection strategies in testing as done in [31]. Specifically, the sampling strategy uses action-sampling to solve an instance for 100 times in parallel and records the best one, to improve the solution quality at the price of an acceptable computation burden. Model's performance was evaluated in terms of the average makespan as well as the relative gap between its makespan and the best-known solution, which is either the solution of OR-Tools for synthetic instances or the best result

[2] https://github.com/wrqccc/FJSP-DRL

[3] https://github.com/songwenas12/fjsp-drl

TABLE II
PERFORMANCE ON TWO SYNTHETIC DATA OF LARGER-SIZES

			Top PDRs		Greedy strategy				Sampling strategy				
					[31]	[31]	DANIEL	DANIEL	[31]	[31]	DANIEL	DANIEL	
	Size		SPT	MWKR	10 × 5	20 × 10	10 × 5	20 × 10	10 × 5	20 × 10	10 × 5	20 × 10	OR-Tools[1]
SD_1	30 × 10	Objective	350.07	312.93	314.71	313.04	288.61	**281.49**	308.55	312.59	286.77	**279.20**	274.67 (6%)
		Gap	27.47%	13.96%	14.61%	14.01%	5.10%	**2.50%**	12.36%	13.49%	4.43%	**1.67%**	
		Time (s)	1.09	1.09	2.86	2.84	2.78	2.76	12.79	12.80	12.37	12.37	
	40 × 10	Objective	445.17	414.82	417.87	416.18	379.28	**371.45**	410.76	415.25	379.71	**370.48**	365.96 (3%)
		Gap	21.66%	13.37%	14.21%	13.75%	3.65%	**1.52%**	12.26%	13.49%	3.77%	**1.14%**	
		Time (s)	1.50	1.50	3.82	3.81	3.77	3.77	24.54	24.40	22.58	21.38	
SD_2	30 × 10	Objective	1105.99	1539.67	1564.57	1543.69	794.62	**774.56**	1486.56	1461.16	757.48	**725.27**	692.26 (0%)
		Gap	59.74%	122.89%	126.55%	123.57%	14.85%	**11.95%**	115.21%	111.51%	9.47%	**4.80%**	
		Time (s)	1.10	1.10	2.93	2.93	2.80	2.75	12.88	12.75	12.48	12.17	
	40 × 10	Objective	1357.16	2037.65	2048.96	2032.54	983.37	**962.58**	1976.25	1945.53	951.21	**914.02**	998.39 (0%)
		Gap	38.74%	108.66%	109.87%	108.12%	0.52%	**-1.67%**	102.45%	99.26%	-2.74%	**-6.60%**	
		Time (s)	1.50	1.51	3.87	3.93	3.76	3.74	24.55	24.50	21.45	21.09	

[1] For OR-Tools, the solution and the ratio of optimally solved instances are reported.

reported in [1] for public benchmarks. In our experiments, performance of the two strategies was compared separately and the best results for the DRL-based methods and PDRs are emphasized in bold for each problem.

D. Results on Synthetic Data

In Table I, we reported each model's average makespan and gap on testing instances which are generated from the same scales using the same distribution as its training instances. It is clear from the bold numbers that, for both data across all problem sizes, DANIEL not only outperforms all PDRs by a significant margin but also exhibits a notable improvement relative to the DRL solution in [31] for both action-selection strategies. Moreover, the optimality gap of DANIEL is less than 5% in three tasks when using the sampling strategy. Most excitingly, DANIEL even beats OR-Tools by 1.03% on the 20 × 10 FJSP instances on SD_1 data, while all instances in this data weren't solved optimally by OR-Tools within the time limits. The advantage of DANIEL becomes more pronounced on SD_2 data. When the processing time range gets larger, the performance of both PDRs and the DRL [31] are affected, resulting in a considerable gap in the best solution. In this case, DANIEL still performs well, especially when using the sampling strategy. When fixing the number of machines, DANIEL achieves a smaller gap as the number of jobs increases, despite the increase in problem size. This may be due to the fact that the larger number of jobs allows the model to better capture the variations in competition intensity among different machines, leading to improved performance. By means of the proposed tight state representation, DANIEL has a runtime comparable to PDRs and is computationally efficient in general. Its runtime is close to that of [31], although implementations of the simulation environment of FJSP in the two algorithms are different. Another interesting phenomenon from the results of PDRs is that SPT consistently performs the worst for all problem sizes on SD_1 data but performs the best for all sizes on SD_2, demonstrating its unstable performance over varying tasks. This may be caused by the very different processing time ranges, since a compatible operation–machine pair with the SPT may have a high priority on SD_2 but a low priority on SD_1 data.

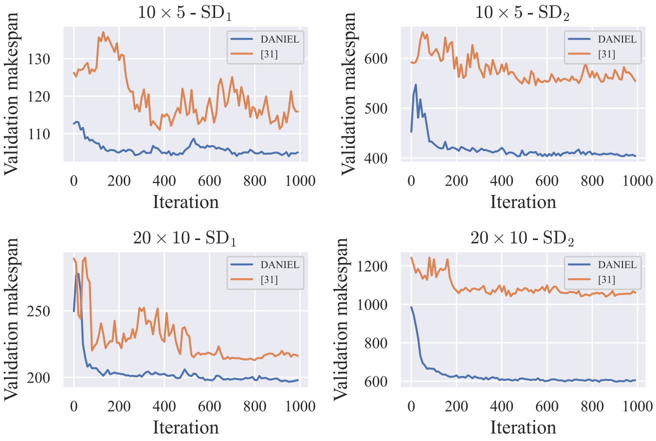


Fig. 5. Training curve of the DRL [31] and DANIEL algorithms on 10 × 5 and 20 × 10 FJSP instances using SD_1 and SD_2.

The training curves of DANIEL and DRL [31] on 10 × 5 and 20 × 10 FJSP instances using SD_1 and SD_2 are presented in Fig. 5, where the averaged makespan over 100 validation instances are shown. It is worth mentioning that our method and [31] used the same training data, validation frequency, and validation data. It can be observed that our method converges to a better solution more smoothly for both data, corroborating its powerful and stable performance.

Furthermore, we examined the generalization performance of the models trained over 10 × 5 and 20 × 10 tasks on two data using 30 × 10 and 40 × 10 testing instances, respectively. DANIEL was compared with [31] and the best PDRs in SD_1 and SD_2. As shown in Table II, DANIEL consistently generates high-quality solutions for large-scale problems with acceptable computation costs, which are considerably better than all baseline methods and occasionally beat OR-Tools. Especially for the 40 × 10 FJSP instances on SD_1, the model

TABLE III
PERFORMANCE ON PUBLIC BENCHMARKS

		Top PDR	Greedy stategy				Sampling stategy					
			[31]	[31]	DANIEL	DANIEL	[31]	[31]	DANIEL	DANIEL		
Benchmark		MWKR	10 × 5	15 × 10	10 × 5	15 × 10	10 × 5	15 × 10	10 × 5	15 × 10	2SGA	OR-Tools
mk	Objective	201.74	201.40	198.50	185.70	**184.40**	190.30	190.60	**180.80**	180.90	175.20	174.20
	Gap	28.91%	28.52%	26.77%	13.58%	**12.97%**	18.56%	19.0%	9.53%	**8.95%**	3.17%	1.5%
	Time (s)	0.49	1.26	1.25	1.29	1.30	4.13	4.13	4.12	4.08	57.60	1447.08
la (rdata)	Objective	1053.10	1030.83	**1031.33**	1031.63	1040.05	985.30	988.38	**978.28**	983.33	-	935.80
	Gap	13.86%	11.15%	**11.14%**	11.42%	12.07%	5.57%	5.95%	**4.95%**	5.49%		0.11%
	Time (s)	0.52	1.40	1.40	1.37	1.36	4.81	4.82	4.71	4.73		1397.43
la (edata)	Objective	1219.01	1187.48	1182.08	1194.98	**1175.53**	**1116.68**	1119.43	1122.60	1119.73	-	1028.93
	Gap	18.6%	15.53%	15.0%	16.33%	**14.41%**	**8.17%**	8.69%	9.08%	8.72%		-0.03%
	Time (s)	0.52	1.40	1.40	1.37	1.38	4.91	4.87	4.73	4.70		899.60
la (vdata)	Objective	952.01	955.90	954.33	**944.85**	948.73	930.80	931.33	**925.40**	925.68	812.20[1]	919.60
	Gap	4.22%	4.25%	4.02%	**3.28%**	3.75%	1.32%	1.34%	**0.69%**	0.72%	0.39%	-0.01%
	Time (s)	0.52	1.37	1.37	1.37	1.37	4.71	4.72	4.77	4.75	51.43	639.17

[1] The makespan and gap of 2SGA on la(vdata) benchmark are computed on la1-30 instances, as reported in [26].

trained on 20 × 10 instances with the sampling strategy outperforms OR-Tools by 6.60%. These results show that DANIEL can learn general knowledge by training on small-scale tasks, which can be employed to solve unseen large-scale instances. Note that the model trained on 20 × 10 consistently outperforms that trained on 10 × 5 instances, which is expected because its size is closer to that of the testing data.

E. Results on Benchmarks

Cross-distribution application is critical for a model in practical use, as real-world problems may come from unknown distributions. Therefore, we further explore the performance of models (trained on synthetic data) on four groups of public benchmarks whose distributions are completely different from the training instances. Each benchmark comprises instances having different problem sizes. For example, the number of jobs and machines in the mk benchmark ranges from 10 to 20 and from 5 to 15, respectively. We selected the models trained on 10 × 5 and 15 × 10 using SD_1 data for testing, which achieved the best results on these benchmarks in [31]. The results of MWKR (the best among the four PDRs), the OR-Tools, the GA baseline in [26], and the models of [31] (with greedy and sampling strategies) trained on the same data are presented in Table III. It can be seen that both OR-Tools and 2SGA are generally far ahead in performance but take quite a long time for computations. On the contrary, the two DRL methods achieve good performance which is better than the best PDR with acceptable runtime. Again, DANIEL surpasses [31] in most cases and exhibits comparable performance in the remaining ones (the testing of greedy strategy on la (rdata) and sampling strategy on la (edata), where the difference is less than 0.6%). Particularly, DANIEL outperforms [31] by a big margin for both action-selection strategies on mk instances. These results showcase that DANIEL can genuinely capture the inherent structural information of FJSP and discriminate compatible pairs with high priority rather than only learning the regularity behind a specific data distribution. We believe that DANIEL can perform better on real-world problems with unknown distributions upon a small amount of fine-tuning, as DANIEL can outperform the exact methods on tasks with known distributions, which is left for future research.

V. CONCLUSION

This study proposes DANIEL, a novel end-to-end learning framework for addressing standard FJSPs. DANIEL combines the attention mechanism and DRL while maintaining the size-agnostic property, i.e., training on small-size problems and deploying to problems of any size. DANIEL builds on a concise representation of the complex FJSP structure, by employing two attention blocks to explore relationships between operations within the job as well as between the competing machines. The DAN receives abundant information about decision-relevant operations and machines as input and extracts their features through GAT-style attention mechanisms. The downstream decision-making network generates composite decisions for the operation sequencing and machine assignment problems simultaneously using actor–critic reinforcement learning, trained by the PPO-Clip algorithm. Substantial numerical results on synthetic data as well as publicly available benchmarks show that DANIEL performs significantly better than traditional PDRs and the state-of-the-art DRL method while being computationally efficient. In several cases, DANIEL even beats the OR-Tools, which has never been reported for end-to-end learning approaches in the literature. Moreover, DANIEL demonstrates excellent generalization performance to large-scale and real-world instances when trained using small-size FJSP data relative to existing DRL methods. In the future, extending DANIEL to address dynamic FJSP with uncertainty, such as new job insertions, and to the multi-agent setting constitutes interesting topics for future research.

VI. Appendix

Here, details for the three types of feature vectors in the state s_t (t is omitted below) are provided.

1) Features of Operations: Four static attributes and six dynamic properties are recorded for each $O_{ij} \in \mathcal{O}_u$.

1) Minimum processing time among all machines.
2) Average processing time among all machines.
3) Span of processing time among all machines.
4) Proportion of machines that O_{ij} can be processed on.
5) *Scheduling Tag:* the value is 0 if O_{ij} is unscheduled otherwise 1.
6) *Estimated Lower Bound of the Completion Time:* $\underline{C}(O_{ij})$ defined in the text.
7) *Job Remaining Number of Operations:* the number of unscheduled operations in J_i.
8) *Job Remaining Workload:* the sum of average processing time of unscheduled operations in J_i.
9) *Waiting Time:* the time from the ready time until T_s (0 for O_{ij} that is not ready yet).
10) *Remaining Processing Time:* the time from T_s until the completion time (0 for unscheduled O_{ij}).

2) Features of Machines: Each machine $M_k \in \mathcal{M}_u$ owns a feature vector with eight elements and the first two are static.

1) Minimum processing time among all operations.
2) Average processing time of operations that M_k can process.
3) Number of unscheduled operations that M_k can process.
4) Number of candidates that M_k can process.
5) *Free Time:* the moment when M_k is free.
6) *Waiting Time:* the time from the free time until T_s (0 for working M_k).
7) *Working Tag*: the value is 0 if M_k is free otherwise 1.
8) *Remaining Processing Time:* the time from T_s until the free time (0 for free M_k).

3) Features of Compatible Operation–Machine Pairs: When a compatible pair (O_{ij}, M_k) is considered, we record eight features for it and the first two are static.

1) Processing time p_{ij}^k.
2) Ratio of p_{ij}^k to the maximum processing time of O_{ij}.
3) Ratio of p_{ij}^k to the maximum processing time of candidates that M_k can process.
4) Ratio of p_{ij}^k to the maximum processing time of unscheduled operations.
5) Ratio of p_{ij}^k to the maximum processing time of unscheduled operations that M_k can process.
6) Ratio of p_{ij}^k to the maximum processing time of compatible pairs.
7) Ratio of p_{ij}^k to remaining workload of J_i.
8) Summation of waiting time of O_{ij} and M_k.

References

[1] D. Behnke and M. J. Geiger, "Test instances for the flexible job shop scheduling problem with work centers," Arbeitspapier/Research Paper/Helmut-Schmidt-Universität, Lehrstuhl für Betriebswirtschaftslehre, insbes, Logistik-Management, Hamburg, Germany, Tech. Rep., 2012.

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

[3] J. Chen, J. Sun, and G. Wang, "From unmanned systems to autonomous intelligent systems," *Engineering*, vol. 12, pp. 16–19, May 2022.

[4] R. Chen, W. Li, and H. Yang, "A deep reinforcement learning framework based on an attention mechanism and disjunctive graph embedding for the job-shop scheduling problem," *IEEE Trans. Ind. Informat.*, vol. 19, no. 2, pp. 1322–1331, Feb. 2023.

[5] Y. Demir and S. K. İşleyen, "Evaluation of mathematical models for flexible job-shop scheduling problems," *Appl. Math. Model.*, vol. 37, no. 3, pp. 977–988, Feb. 2013.

[6] K. Ding, F. T. S. Chan, X. Zhang, G. Zhou, and F. Zhang, "Defining a digital twin-based cyber-physical production system for autonomous manufacturing in smart shop floors," *Int. J. Prod. Res.*, vol. 57, no. 20, pp. 6315–6334, Jan. 2019.

[7] W. Du, W. Zhong, Y. Tang, W. Du, and Y. Jin, "High-dimensional robust multi-objective optimization for order scheduling: A decision variable classification approach," *IEEE Trans. Ind. Informat.*, vol. 15, no. 1, pp. 293–304, Jan. 2019.

[8] 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.*, early access, Oct. 10, 2022, doi: 10.1109/TNNLS.2022.3208942.

[9] R. Haupt, "A survey of priority rule-based scheduling," *Oper. Res. Spektrum*, vol. 11, no. 1, pp. 3–16, Mar. 1989.

[10] F. Hou, J. Sun, Q. Yang, and Z. Pang, "Deep reinforcement learning for optimal denial-of-service attacks scheduling," *Sci. China Inf. Sci.*, vol. 65, no. 6, Jun. 2022, Art. no. 162201.

[11] G. Huang, Z. Liu, G. Pleiss, L. van der Maaten, and K. Q. Weinberger, "Convolutional networks with dense connectivity," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 44, no. 12, pp. 8704–8716, Dec. 2022.

[12] W. Huang and L. Gao, "A time wave neural network framework for solving time-dependent project scheduling problems," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 31, no. 1, pp. 274–283, Jan. 2020.

[13] J. Hurink, B. Jurisch, and M. Thole, "Tabu search for the job-shop scheduling problem with multi-purpose machines," *Oper. Res. Spektrum*, vol. 15, no. 4, pp. 205–215, Dec. 1994.

[14] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2014, *arXiv:1412.6980*.

[15] W. Kool, H. Van Hoof, and M. Welling, "Attention, learn to solve routing problems!" in *Proc. Int. Conf. Learn. Represent.*, 2018, pp. 1–25.

[16] Y.-D. Kwon, J. Choo, I. Yoon, M. Park, D. Park, and Y. Gwon, "Matrix encoding networks for neural combinatorial optimization," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 34, 2021, pp. 5138–5149.

[17] K. Lei et al., "A multi-action deep reinforcement learning framework for flexible job-shop scheduling problem," *Expert Syst. Appl.*, vol. 205, Nov. 2022, Art. no. 117796.

[18] J.-Q. Li et al., "Hybrid artificial bee colony algorithm for a parallel batching distributed flow-shop problem with deteriorating jobs," *IEEE Trans. Cybern.*, vol. 50, no. 6, pp. 2425–2439, Jun. 2020.

[19] Y. Li, X. Wang, J. Sun, G. Wang, and J. Chen, "Data-driven consensus control of fully distributed event-triggered multi-agent systems," *Sci. China Inf. Sci.*, vol. 66, no. 5, May 2023, Art. no. 152202.

[20] S. Luo, L. Zhang, and Y. Fan, "Real-time scheduling for dynamic partial-no-wait multiobjective flexible job shop by deep reinforcement learning," *IEEE Trans. Autom. Sci. Eng.*, vol. 19, no. 4, pp. 3020–3038, Oct. 2022.

[21] S. Mao, B. Wang, Y. Tang, and F. Qian, "Opportunities and challenges of artificial intelligence for green manufacturing in the process industry," *Engineering*, vol. 5, no. 6, pp. 995–1002, Dec. 2019.

[22] 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, Apr. 2020, Art. no. 106347.

[23] A. Mirhoseini et al., "A graph placement methodology for fast chip design," *Nature*, vol. 594, no. 7862, pp. 207–212, Jun. 2021.

[24] V. Mnih et al., "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, Feb. 2015.

[25] J. Park, J. Chun, S. H. Kim, Y. Kim, and J. Park, "Learning to schedule job-shop problems: Representation and policy learning using graph neural network and reinforcement learning," *Int. J. Prod. Res.*, vol. 59, no. 11, pp. 3360–3377, Jun. 2021.

[26] D. Rooyani and F. M. Defersha, "An efficient two-stage genetic algorithm for flexible job-shop scheduling," *IFAC-PapersOnLine*, vol. 52, no. 13, pp. 2519–2524, 2019.

[27] S. Satunin and E. Babkin, "A multi-agent approach to intelligent transportation systems modeling with combinatorial auctions," *Expert Syst. Appl.*, vol. 41, no. 15, pp. 6622–6633, Nov. 2014.

[28] J. Schulman, P. Moritz, S. Levine, M. Jordan, and P. Abbeel, "High-dimensional continuous control using generalized advantage estimation," 2015, *arXiv:1506.02438*.

[29] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," 2017, *arXiv:1707.06347*.

[30] V. Sels, N. Gheysen, and M. Vanhoucke, "A comparison of priority rules for the job shop scheduling problem under different flow time- and tardiness-related objective functions," *Int. J. Prod. Res.*, vol. 50, no. 15, pp. 4255–4270, Aug. 2012.

[31] W. Song, X. Chen, Q. Li, and Z. Cao, "Flexible job-shop scheduling via graph neural network and deep reinforcement learning," *IEEE Trans. Ind. Informat.*, vol. 19, no. 2, pp. 1600–1610, Feb. 2023.

[32] A. Vaswani et al., "Attention is all you need," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 30, 2017, pp. 1–11.

[33] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio, "Graph attention networks," in *Proc. Int. Conf. Learn. Represent.*, 2017, pp. 1–12.

[34] G. Wang, G. B. Giannakis, and J. Chen, "Learning ReLU networks on linearly separable data: Algorithm, optimality, and generalization," *IEEE Trans. Signal Process.*, vol. 67, no. 9, pp. 2357–2370, May 2019.

[35] L. Wang et al., "Dynamic job-shop scheduling in smart manufacturing using deep reinforcement learning," *Comput. Netw.*, vol. 190, May 2021, Art. no. 107969.

[36] L. Wang, Z. Pan, and J. Wang, "A review of reinforcement learning based intelligent optimization for manufacturing scheduling," *Complex Syst. Model. Simul.*, vol. 1, no. 4, pp. 257–270, Dec. 2021.

[37] M. Wang and B. Xin, "A genetic algorithm for solving flexible flow shop scheduling problem with autonomous guided vehicles," in *Proc. IEEE 15th Int. Conf. Control Autom. (ICCA)*, Jul. 2019, pp. 922–927.

[38] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and P. S. Yu, "A comprehensive survey on graph neural networks," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 32, no. 1, pp. 4–24, Jan. 2021.

[39] J. Xie, L. Gao, K. Peng, X. Li, and H. Li, "Review on flexible job shop scheduling," *IET Collaborative Intell. Manuf.*, vol. 1, no. 3, pp. 67–77, Sep. 2019.

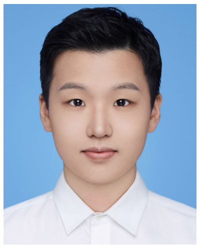
[40] Y. Xie, H. Wang, G. Liu, and H. Lu, "Just-in-time precast production scheduling using dominance rule-based genetic algorithm," *IEEE Trans. Neural Netw. Learn. Syst.*, early access, Nov. 4, 2022, doi: 10.1109/TNNLS.2022.3217318.

[41] Y. Xu, Z.-G. Wu, W.-W. Che, and D. Meng, "Reinforcement learning-based unknown reference tracking control of HMASs with nonidentical communication delays," *Sci. China Inf. Sci.*, vol. 66, no. 7, Jul. 2023, Art. no. 170203.

[42] C. Zhang, W. Song, Z. Cao, J. Zhang, P. S. Tan, and X. Chi, "Learning to dispatch for job shop scheduling via deep reinforcement learning," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 33, 2020, pp. 1621–1632.

[43] G. Zhang, X. Shao, P. Li, and L. Gao, "An effective hybrid particle swarm optimization algorithm for multi-objective flexible job-shop scheduling problem," *Comput. Ind. Eng.*, vol. 56, no. 4, pp. 1309–1318, May 2009.

[44] J. Zhang, G. Ding, Y. Zou, S. Qin, and J. Fu, "Review of job shop scheduling research and its new perspectives under industry 4.0," *J. Intell. Manuf.*, vol. 30, no. 4, pp. 1809–1830, Apr. 2019.



Runqing Wang received the bachelor's degree from the School of Mathematics, Hefei University of Technology, Hefei, China, in 2021. He is currently pursuing the master's degree with the School of Automation, Beijing Institute of Technology, Beijing, China.

His research interests include deep reinforcement learning and neural combinatorial optimization.

Gang Wang (Senior Member, IEEE) received the B.Eng. degree in automatic control and the Ph.D. degree in control science and engineering from the Beijing Institute of Technology, Beijing, China, in 2011 and 2018, respectively, and the Ph.D. degree in electrical and computer engineering from the University of Minnesota, Minneapolis, MN, USA, in 2018.

Since July 2020, he has been a Post-Doctoral Researcher with the University of Minnesota. Since August 2020, he has been a Professor with the School of Automation, Beijing Institute of Technology. His research interests include signal processing, control, and reinforcement learning with applications to cyber–physical systems and multi-agent systems.

Dr. Wang was a recipient of the Best Paper Award from the FITEE in 2021, the Excellent Doctoral Dissertation Award from the Chinese Association of Automation in 2019, the Eusipco Best Student Paper Award in 2017, and the Best Conference Paper at the 2019 IEEE Power and Energy Society General Meeting. He is currently an Associate Editor of *Signal Processing* and the IEEE TRANSACTIONS ON SIGNAL AND INFORMATION PROCESSING OVER NETWORKS, and an Early Career Advisory Board Member of the IEEE/CAA JOURNAL OF AUTOMATICA SINICA.

Jian Sun (Senior Member, IEEE) received the bachelor's degree in electric engineering from the Jilin Institute of Technology, Changchun, China, in 2001, the master's degree in mechatronics engineering from the Changchun Institute of Optics, Fine Mechanics and Physics, Chinese Academy of Sciences (CAS), Changchun, in 2004, and the Ph.D. degree in control theory and control engineering from the Institute of Automation, CAS, Beijing, China, in 2007.

He was a Research Fellow with the Faculty of Advanced Technology, University of Glamorgan, Pontypridd, U.K., from 2008 to 2009. He was a Post-Doctoral Research Fellow with the Beijing Institute of Technology, Beijing, from 2007 to 2010. In 2010, he joined the School of Automation, Beijing Institute of Technology, where he has been a Professor since 2013. His current research interests include networked control systems, time-delay systems, and security of cyber–physical systems.

Dr. Sun is an Editorial Board Member of IEEE TRANSACTIONS ON SYSTEMS, MAN, AND CYBERNETICS: SYSTEMS, *Journal of Systems Science and Complexity*, *Science China: Information Sciences*, and *Acta Automatica Sinica*.

Fang Deng (Senior Member, IEEE) received the B.E. and Ph.D. degrees in control science and engineering from the Beijing Institute of Technology, Beijing, China, in 2004 and 2009, respectively.

He is currently a Professor with the School of Automation, Beijing Institute of Technology. His current research interests include nonlinear estimation, fault diagnosis, control of renewable energy resources, and wireless sensor networks.

Jie Chen (Fellow, IEEE) received the B.Sc., M.Sc., and Ph.D. degrees in control theory and control engineering from the Beijing Institute of Technology, Beijing, China, in 1986, 1996, and 2001, respectively.

He was the President of Tongji University, Shanghai, China, from 2018 to 2023. He is currently a Professor in control science and engineering with the Beijing Institute of Technology and Tongji University, where he was the Director of the National Key Laboratory of Autonomous Intelligent Unmanned Systems (KAIUS). His research interests include complex systems, multiagent systems, multiobjective optimization and decision, and constrained nonlinear control.

Prof. Chen is a fellow of IFAC and a member of the Chinese Academy of Engineering. He is currently the Editor-in-Chief of *Unmanned Systems and the Journal of Systems Science and Complexity*. He has served on the editorial boards for several journals, including the IEEE TRANSACTIONS ON CYBERNETICS, *International Journal of Robust and Nonlinear Control*, and *Science China Information Sciences*.