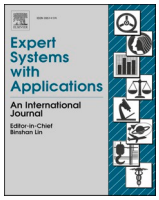
Expert Systems With Applications 238 (2024) 121756

Contents lists available at ScienceDirect

Expert Systems With Applications

journal homepage: www.elsevier.com/locate/eswa



An end-to-end deep reinforcement learning method based on graph neural network for distributed job-shop scheduling problem

Jiang-Ping Huang, Liang Gao, Xin-Yu Li *

State Key Laboratory of Intelligent Manufacturing Equipment and Technology, Huazhong University of Science and Technology, Wuhan 430074, China

ARTICLE INFO

Keywords:
Distributed job shop
Scheduling
Distributed scheduling
Reinforcement learning

ABSTRACT

Distributed Job-shop Scheduling Problem (DJSP) is a hotspot in industrial and academic fields due to its valuable application in the real-life productions. For DJSP, the available methods aways complete the job selection first and then search for an appropriate factory to assign the selected job, which means job selection and job assignment are done independently. This paper proposes an end-to-end Deep Reinforcement Learning (DRL) method to make the two decisions simultaneously. To capture the problem characteristics and realize the objective optimization, the Markov Decision Process (MDP) of DJSP is formulated. Specialised action space made up of operation-factory pairs is designed to achieve the simultaneous decision-making. A stitched disjunctive graph representation of DJSP is specially designed, and a Graph Neural Network (GNN) based feature extraction architecture is proposed to dig the state embedding during problem solving. A Proximal Policy Optimization (PPO) method is applied to train an action-selection policy. To further lead the agent to assign jobs to the factory with smaller makespan, a probability enhancement mechanism is designed. The experimental results on 240 test instances have shown that the proposed method outperforms 8 classical Priority Dispatching Rules (PDRs), 3 closely-related RL methods and 5 metaheuristics in terms of effectiveness, stability and generalization.

1. Introduction

With the prevalence of cooperative production among enterprises, distributed manufacturing has become a common production mode. It can take full advantage of the resources of multiple enterprises, and realize a rational allocation of resources, so that the efficient production can be achieved at a reasonable cost (Sabuncuoglu & Karpat, 2009). The manufacturing environment has been transformed from the single-factory workshop to the multi-factory workshop with decentralized structure. In the equipment manufacturing industry, Distributed Job-shop Scheduling Problem (DJSP) is a typical case, and has been a hot-spot (Şahman, 2021). It is an extension of JSP (Naderi & Azab, 2014), and is first investigated by Jia et al. (2003). Since JSP is a NP-hard problem (Liu et al., 2022), DJSP is also NP-hard. Due to the practicality and difficulty of DJSP, it is studied in the paper.

Industry 4.0 advocates building smart factories and promotes intelligent manufacturing, which raises greater demands on the automation and intelligent decision-making of manufacturing systems. The existed technologies and human expertise can hardly meet the scheduling demands in such manufacturing system. In the scheduling field, exact methods (Naderi & Ruiz, 2010), heuristics (Naderi & Azab, 2014) and metaheuristics (Huang et al., 2021) are the most widely studied methods, they are applicable to different scenarios, but their performance varies a lot for solving different problems. The exact methods can get the optimal solution for the problems with small size in a reasonable time, while their ability to solve the problems with big size is limited. The distributed scheduling problems are NP-hard, therefore the exact methods are not an ideal choice for them. The heuristics are direct and easy to implement, however the rational application of them raises high demands on rich human knowledge. Currently, the metaheuristics are the most dominant ones in the scheduling field for their good ability to search for the optimal solution. However, for some scenarios requiring fast responsiveness, they can hardly generate a scheduling scheme in a short time due to their iterative evolutionary mechanism. The development of Industry 4.0 is revolutionizing the diversity and availability of industrial data, and the accessibility of the industrial data is improved. The three mentioned methods can solve the most scheduling problems in a good manner, but they lack the ability to learn from the industrial data. To adapt to the new industrial environment, and facilitate the integration of scheduling algorithms with new information technologies, it is

* Corresponding author.
E-mail addresses: d202180294@hust.edu.cn (J.-P. Huang), gaoliang@mail.hust.edu.cn (L. Gao), lixinyu@mail.hust.edu.cn (X.-Y. Li).

https://doi.org/10.1016/j.eswa.2023.121756
Received 31 July 2023; Received in revised form 2 September 2023; Accepted 19 September 2023
Available online 27 September 2023

necessary to develop the methods with self-learning and self-evolving ability. In such background, Deep Reinforcement Learning (DRL) has been an ideal technique for various issues (Yu & Luo, 2023; Ruan et al., 2021; Wang & Deng, 2021). It's the combination of Deep Learning (DL) and Reinforcement Learning (RL), and it obtains the intelligent decision ability from RL and the feature extraction ability from DL. DRL can self-learn and self-evolute, and can make automatic decisions based on the environment. The application of DRL in the shop scheduling field has been tapped (Liu et al., 2022; Zhang et al., 2022; Zhang et al., 2022). Han and Yang (2021) propose an end-to-end DRL framework based on 3D disjunction graph dispatching for Flexible Job Shop Scheduling Problem (FJSP). Luo et al. (2021) develop a real-time scheduling method based on hierarchical multi-agent DRL for dynamic partial-no-wait multi-objective FJSP.

With the motivations mentioned above, this paper explores an end-to-end DRL method based on Graph Neural Network (GNN) (abbreviated as GDRL) to address DJSP with minimization of makespan. The contributions of the paper can be listed as follows: (1) A stitched disjunctive graph representation for DJSP is proposed to facilitate the feature extraction made by the GNN; (2) An action space composed by operation-factory pairs is presented to guide the agent to assign jobs to appropriate factories; (3) A probability enhancement mechanism based on factory makespan is proposed to avoid the excessive job accumulation in some factory.

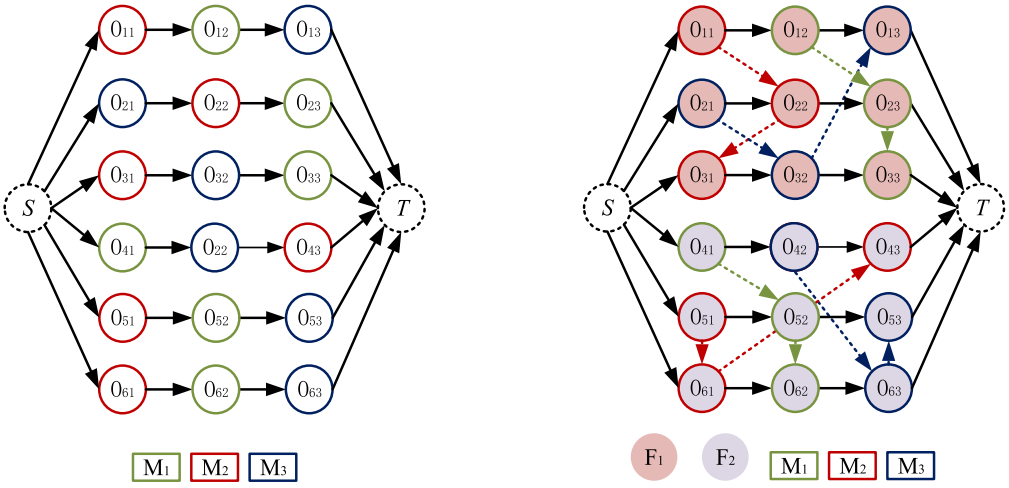
The paper is organized as follows. Section 2 reviews the closely related literature. Section 3 shows the definition of DJSP and the solution representation based on disjunctive graph. Section 4 describes the proposed algorithm in detail. Section 5 tells the experimental results and evaluates the algorithm's performance. Section 6 presents a conclusion of the paper and plans a picture for the further study.

2. Literature review

Distributed manufacturing has drawn much attention recent years. Okwudire and Madhyastha (2021) publish a paper in "*Science*" and point that the distributed paradigm improves the responsiveness and resilience of enterprise, and can cope with urgent production demands, promote customization and achieve energy saving. Even though DJSP is a typical case in the distributed manufacturing, only several related researches have been proposed. Naderi and Azab (2014) propose two Mix Integer Linear Programming models (MILPs) for DJSP, and address the large instances with three heuristics and three greedy heuristics. Chaouch et al. (2019) propose a hybrid ant-based algorithm based on a dynamic assignment rule to minimize the makespan. A energy-efficient DJSP (da Jiang et al., 2020) is solved by a multi-objective evolutionary algorithm with decomposition. Due to the limited research for DJSP, the research on the related distributed scheduling problems is also analyzed. For Distributed Permutation Flowshop Scheduling Problem (DPFSP), Naderi and Ruiz (2010) propose six MILPs, two factory assignment rules and 14 dispatching rules. Ruiz et al. (2019) apply an iterated greedy (IG) method for DPFSP with minimization of makespan. To optimizing the total flowtime in DPFSP, Pan et al. (2019) explore metaheuristics based on the frameworks of scatter search, discrete artificial bee colony, IG and iterated local search. DPFSP with different constraints are also widely studied, such as the customer order constraint (Meng et al., 2019), due windows (Jing et al., 2020) and energy-efficient constraint (Wang & Wang, 2020). The distributed blocking flow shop scheduling problem with energy-efficient constraints is also solved with Pareto-based discrete Jaya algorithm (Zhao, Zhang, et al., 2023) and a hyper-heuristic with Q-learning (Zhao, Di, et al., 2023). Distributed Flexible Job-shop Scheduling Problem (DFJSP) has also been studied by many researchers. De Giovanni and Pezzella (2010) propose a Genetic Algorithm (GA) for DFJSP to minimize the makespan. Chang and Liu (2017) apply a hybrid GA with adopted crossover and mutation operators for the same problem. Meng et al. (2020) address DFJSP with four MILPs and a constraint programming model, and achieve a good performance on both the small-size and the large-size instances. For DFJSP with transfers (Luo et al., 2020), a memetic algorithm is proposed to minimizie the makespan, maximum workload, and total energy consumption. Du et al. (2021) solve DFJSP with crane transportations constraint with a hybrid estimation of distributed algorithm. Through the investigation of the studies about distributed manufacturing, it can be found that the distributed manufacturing scenarios are complex, and it is difficult to find a method that performs well in all scenarios. In addition, almost all of the existing algorithms for the distributed shop scheduling are the exact methods, heuristics and metaheuristics, all of which are hardly satisfy the requirements of efficiency, and time-saving at the same time. Thus, it is necessary to explore different optimization methods for different scenarios to improve productivity.

DRL-based methods have been a popular tool for solving combinatorial optimization problems, such as Vehicle Routing Problem (VRP) (Lin et al., 2022), Traveling Salesman Problem (TSP) (Zhang et al., 2021), and Cloud Resource Scheduling (Guo et al., 2021). Shop scheduling as a classical combinatorial optimization problem, the applications of DRL on it also attract much attention, and they are in indirect or direct way. The indirect way combines the DRL agent with the dispatching rules, heuristics or metaheuristics, while the direct one extracts the state features by observing the environment and generates a scheduling scheme with the agent directly. The direct architecture is also called as "end-to-end". For the indirect application of DRL in the scheduling field, the Priority Dispatching Rules (PDRs) and local search operators are usually set as the agents' actions. Han and Yang (2020) view the scheduling process using a disjunction graph as a multi-stage sequential decision-making problem and use a deep convolutional neural network to approximate the state-action value, and the eighteen PDRs are set as the actions to be used to select the processed job. Zhao et al. (2022) propose a RL driven artificial bee colony algorithm for distributed heterogeneous no-wait flowshop scheduling problem with sequence-dependent setup times, where Q-learning is used to select neighborhood structures via empirical knowledge in the operation processes. Besides, DRL is also used to parameter tuning for metaheuristics. Zhao et al. (2021) address the distributed assembly no-idle flowshop scheduling problem with a water wave optimization algorithm, and they apply RL to balance the exploration and exploitation of the algorithm. Gu et al. (2023) use DQN-based population adaptive partitioning mechanism in the discrete salp swarm algorithm for the dynamic JSP to improve the self-learning ability of the algorithm. From the above literature, it can be found that the indirect application of DRL in the shop scheduling problems is combining DRL with the traditional methods such as PDRs and metaheuristics, where DRL is used to improve the performance of the traditional methods. Thus, it can't fully overcome the shortcomings of the traditional methods, for example, the running time of the metaheuristics can't be cut by DRL at a large extent. To take full advantage of DRL, the indirect application of DRL is not enough.

In the end-to-end way, the scheduling policy trained by DRL method can achieve efficient decisions during problem solving. Han and Yang (2021) propose a DRL framework with a modified pointer network to encode and decode FJSP. Palombarini and Martinez (2022) apply DRL to achieve an end-to-end on-line rescheduling at shop floor by observing the color-rich Gantt chart images. Lei et al. (2022) propose a multi-action DRL for FJSP by combining disjunctive graph and GNN, and realize a outperformance. Pan et al. (2021) minimize the maximum completion time of permutation flowshop scheduling problem with a DRL method to achieve an end-to-end output without consideration of the problem sizes. A DRL-based reactive policy for FJSP with random job arrivals is proposed to minimize the total tardiness (Zhao, Fan, et al., 2023). GNNs, as a popular artificial intelligent technique, also have applications in the shop scheduling field. Zhang et al. (2020) exploit the disjunctive graph with GNN to achieve an automatically PDRs learning by an end-to-end DRL agent. Wang and Gombolay (2020) use a graph attention network-based scheduler to automatically learn features of scheduling problems to generate high-quality solutions. Zhang et al.



(a) Initial information of DJSP instance (b) A feasible solution of the instance

Fig. 1. Stitched disjunctive graph of DJSP.

(2023) propose a DRL with multi-agent graphs for FJSP. From the literature, it can be found that the end-to-end DRL-based methods have shown good performance on the shop scheduling problems such as JSP, FJSP, etc. However, its application in the distributed scheduling problems is rarely studied, and the distributed characteristics are seldom considered. Therefore, it is reasonable and necessary to explore the end-to-end application of DRL in the distributed scheduling problems.

In the mentioned related works, various methods have been proposed for distributed scheduling problems including exact methods, heuristics and metaheuristics. The DRL-based methods also have important application in scheduling problems. In addition, to best of our knowledge, the research based on DRL for distributed scheduling is scarcely studied. Therefore, an end-to-end GNN-based DRL method is proposed to solve the DJSP.

3. Distributed scheduling environment

3.1. Problem description

DJSP is formed of a set of f identical factories, each of which contains m machines. There are a set of n jobs that should be processed in one of the f factories. Each job has its own process route, and any operation of the job should be processed on the specific machine. For DJSP, two decisions should be made, one is selecting a job and assigning it to an appropriate factory, and the other is scheduling jobs in each factory. Once a job starts its process in one factory, it is not allowed to transform it to another factory. For shop scheduling problem, the maximum completion time of jobs, *i.e.*, the makespan (C_{max}) is one of the most common objectives (Huang et al., 2021; Pan et al., 2022). For the distributed scheduling, minimizing makespan means minimizing the maximum makespan among factories. This paper solves DJSP to minimize the makespan.

3.2. Disjunctive graph model

In general, JSP can be represented by a disjunctive graph (Błażewicz et al., 2000) $G = (V, C \cup D)$, where V contains a set of vertices corresponding to job operations, C is a set of directed arcs that reflect the process constraints among the operations of each single job, D is used to store the disjunctive edges that reflect the execution order of operations of different jobs on the same machines. Initially, the edges in D is undirected, what we should do is find a method to direct the edges in order, so as to minimize the objective. Based on the definition of disjunctive graph, a specific disjunctive graph for DJSP is proposed in this paper. Compared to JSP, the job operations should be charactered by the factory where the operations are assigned to. Therefore, a stitched disjunctive graph representation $G = \left(V, (C_1 \cup D_1) \cup (C_2 \cup D_2) \cup \cdots \cup \left(C_f \cup D_f\right)\right)$ is proposed, where f is the number of factories. The stitched disjunctive graph and a feasible solution of a $2 \times 3 \times 3$ DJSP instance are illustrated in Fig. 1. Specially, the dummy vertices S and T is used to mark the start and the end of the scheduling process, and they are included in V. In Fig. 1(a), the process constraints of the same job are directed by the black arrows, and the same available machines for different operations are marked with the same color. In Fig. 1 (b), all disjunctive edges are directed, which shows the processing order of operations on each machine. In addition, the operation vertices are marked in the same color if they are assigned to the same factory. For this example, the operations of jobs 1, 2 and 3 are colored in pink, which means all of them are assigned to factory 1. The operations of jobs 4, 5 and 6 are colored in purple, which means all of them are assigned to factory 2.

4. GNN-based DRL method

In this section, the framework of GDRL is introduced. First, DJSP is conceptualized as a Markov Decision Process (MDP); Then a network model based on GNN is proposed, which acts the encoder and the decoder to extract the scheduling information from the stitched disjunctive graph and schedule a task at decision point; Last, a PPO method based on Actor-Critic framework is designed for training the proposed network model.

4.1. Markov decision process formulation

The scheduling of DJSP is formulated as a sequential decision-making process. For any DJSP instance, $n \times m$ consecutive decisions should be made to assign job tasks, where n is the number of jobs, and m is the number of operations of each job. At each decision point, the agent select an eligible action via a priority index which is output by a sampling or greedy strategy. MDP is defined as a tuple (S, A, P, R, γ), where S is the state set, A is the action set, P is the transition function to show the state transition, R is the reward function, and γ is the discount factor valued between 0 and 1. The details of tuple components of DJSP are listed as follows.

State. The state s_t at decision point t is a stitched disjunctive graph $G = \left(V, \left(C_{1t} \cup D^o_{1t} \cup D_{1t}\right) \cup \left(C_{2t} \cup D^o_{2t} \cup D_{2t}\right) \cup \cdots \cup \left(C_f \cup D^o_{ft} \cup D_{ft}\right)\right)$. D^o_{kt} contains the directed disjunctive edges till time t in factory k, while D_{kt} contains the remaining disjunctive edges with no directions in

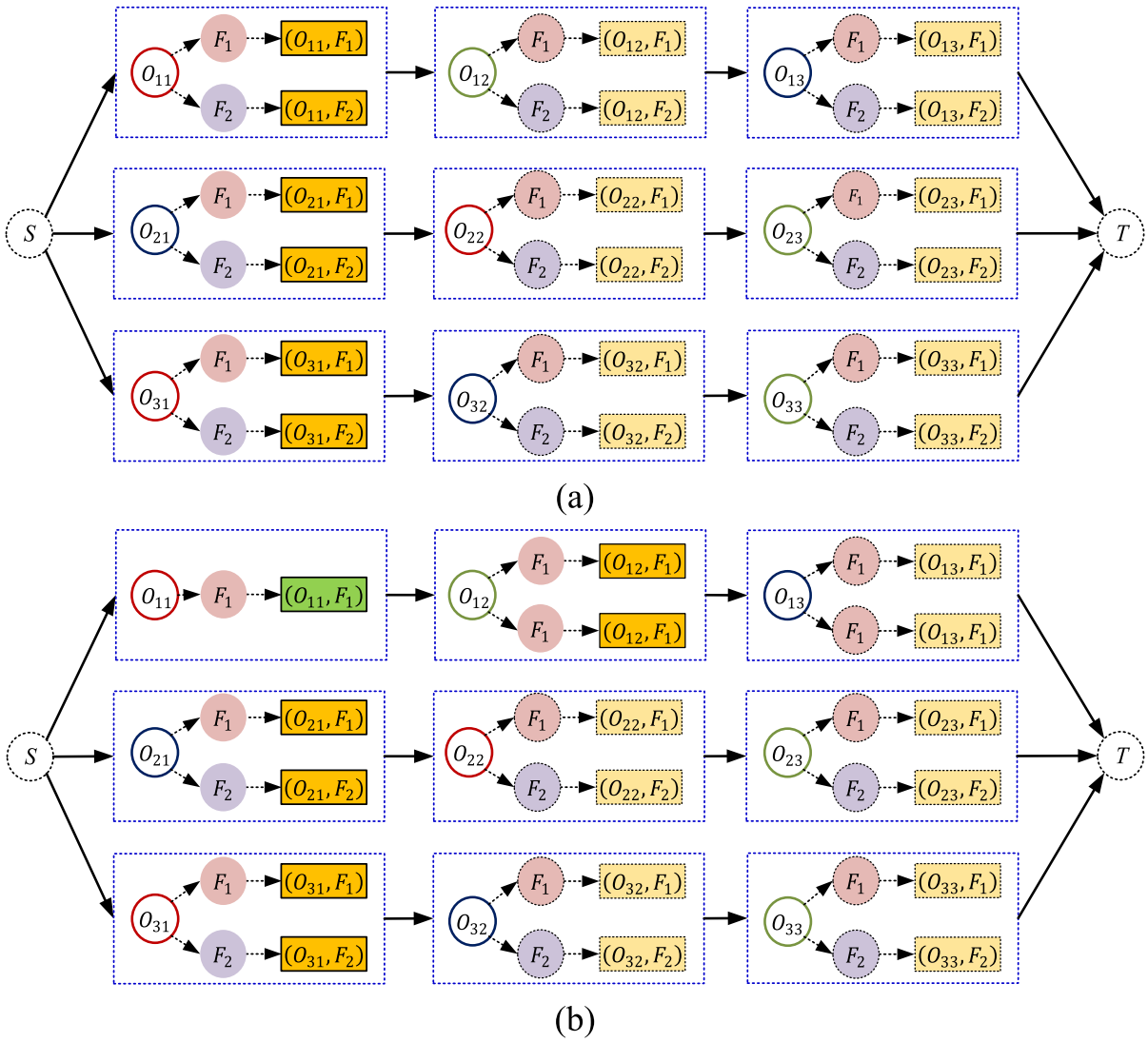


Fig. 2. The diagram of the change process of action space.

factory k, *i.e.*, $D_{kt}^{o} \cup D_{kt} = D_k$, $k \in [1, f]$. Any vertex v in set V corresponds to a certain job operation O_{ji} ($j \in [1, n]$, $i \in [1, m]$), if job operation O_{ji} is assigned to factory k, the variant $F_t^{O_{ji}}$ is valued as k. Each vertex v in V contains two features, which are the estimated lower bounder of completed time of the corresponding job operation O_{ji} and the 0–1 variant that marks the corresponding operation O_{ji} is scheduled or not. Specially, the estimated lower bounder of completed time $C_{LB}(O_{ji})$ is calculated by

$$C_{LB}(O_{ji}) = C_{LB}(O_{ji-1}) + p_{ji} \tag{1}$$

where p_{ji} is the processing time of O_{ji}. When operation O_{ji} has been processed, its estimated lower bounder of completed time refers to its actual makespan. When O_{ji} is the first operation of job j, (*i.e.*, $i = 1$), the estimated lower bounder equals to the release time of job j plus the processing time of operation O_{ji}. The 0–1 variant is $b(O_{ji})$, and it is masked as 1 if operation O_{ji} is scheduled, otherwise, it equals to 0.

Actions. For DJSP, job assigning and job sequencing are two critical decision-making processes, the decision-maker should select a job operation from the eligible operations and assign it to an appropriate factory. With the definition of DJSP, all the operations of one job should be processed in the same factory. Thus, for a complete scheduling scheme, each job operation belongs to a unique and certain factory. It is therefore straightforward to employ an operation-factory pair to represent the factory allocation for the operations. For example, operation-factory pair (O_{32}, F_2) means job operation O_{32} is assigned to factory 2. Therefore, in the paper, the job operations and factories are paired together to form the action space A, with which the selected job and its assigned factory at each decision point can be easily represented. At any decision point t, first, an action a_t is selected from action space A_t according to the current state. Then, the action space is transformed from A_t to A_{t+1} according to the selected action a_t. As the definition of DJSP, any job can make only one operation been processed at any time. Therefore, for a $f \times n \times m$ instance, the size of action space keeps $n \times f$ until some job is completed. As jobs are completed one by one, the action space becomes smaller and smaller. Take a $2 \times 3 \times 3$ instance as an example, the change process of action space is shown in Fig. 2, where the components of action space are colored in orange. If some job operation has been completed, the corresponding action is colored in green. As shown in Fig. 2(a), at begin, each available operation can be assigned to two different factories, and the action space is formed by operation-factory pairs (O_{11}, F_1), (O_{11}, F_2), (O_{21}, F_1), (O_{21}, F_2), (O_{31}, F_1), (O_{31}, F_2). If operation-factory pair (O_{11}, F_1) is selected, as Fig. 2(b) presented, the action space turns into $\{(O_{12}, F_1), (O_{12}, F_1), (O_{21}, F_1), (O_{21}, F_2), (O_{31}, F_1), (O_{31}, F_2)\}$. In Fig. 2(c), as operations O_{11}, O_{12} and O_{31} are done, the action space is transformed into $\{(O_{13}, F_1), (O_{13}, F_1), (O_{21}, F_1), (O_{21}, F_2), (O_{32}, F_2), (O_{32}, F_2)\}$. In Fig. 2(d), the last operation of job 1 is done, and the size of action space turns six to four. It can be

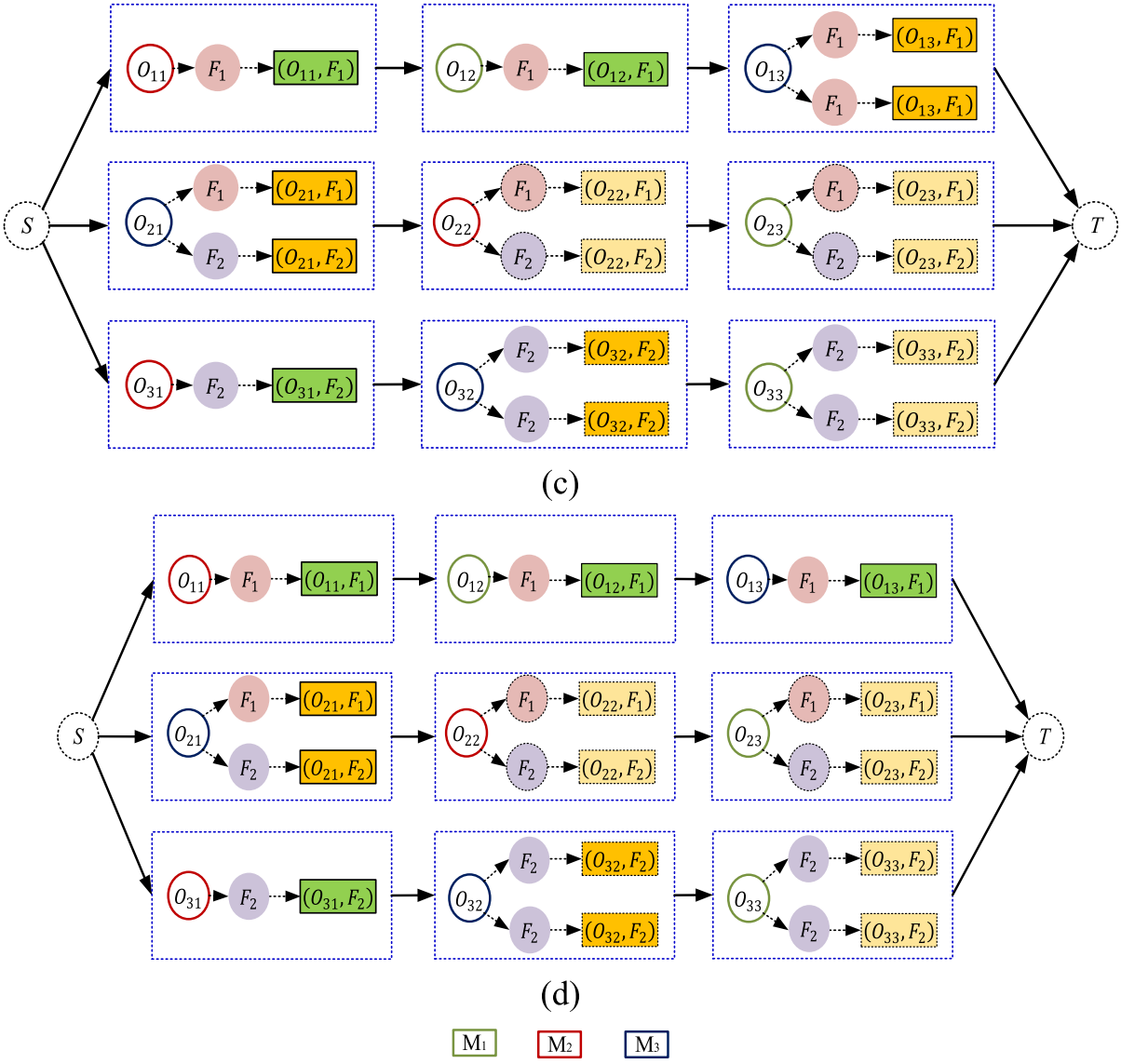


Fig. 2. (*continued*).

found that once one job is completed, the action space decreases by f, where f is the number of factories.

State Transition. At decision point t, the agent samples an action a_t, with which the selected job operation and its assigned factory can be obtained, and the directions of disjunctive arcs are determined based on the operation-factory pair corresponding to the sampled action. The vertex features are updated based on the updated disjunctive graph.

Reward. The reward is a significant information fed back by the environment, and it is used to evaluate whether the last action performed by the agent is good or not. Through a reasonable reward function, a better scheduling policy can be obtained. In this paper, the objective is to minimize the makespan. For DJSP, minimizing the makespan means minimizing the maximum makespan among factories. Therefore, a reward function based on factory makespan is designed. The makespan of factory k is C_k, and it equals to the maximum completed time of operations assigned into factory k, *i.e.*,

$$C_k = max\left(C_{LB}\left(O_{ji}^k\right)\right), k \in [1,f], j \in [1,n], i \in [1,m] \tag{2}$$

where O_{ji}^k denotes operation O_{ji} is processed in factory k, *i.e.*, $F_t^{O_{ji}} = k$. The makespan is calculated by

$$C_{max} = max(C_k), k \in [1,f] \tag{3}$$

The proposed reward function is defined as

$$R(s_t, a_t) = C_k(s_t) - C_k(s_{t+1}), k \in [1,f] \tag{4}$$

With the reward function, the reward value is inversely proportional to the makespan increment of the assigned factory. In a long run, the reward leads the agent to optimize the job schedule.

4.2. GNN-based policy

GNN is a kind of deep neural network, and it can learn the representation of graph-structured data. It has been successfully applied in many fields, such as decentralized wireless resource allocations (Wang et al., 2022), the prediction of remaining useful life of industrial equipment (Kong et al., 2022). The disjunctive graph representation is also a graph-structured data, therefore, it's reasonable to combine GNN with the disjunctive graph to explore an effective scheduling method for DJSP.

In this section, the details of scheduling a job operation are described. The entire framework is composed of three steps: (1) Generating a job task encoding according to the input disjunctive graph; (2) Selecting an action by calculating the probability distribution over the feasible actions according to the encoding, if at the testing stage, a probability enhancement mechanism is applied; (3) Scheduling the

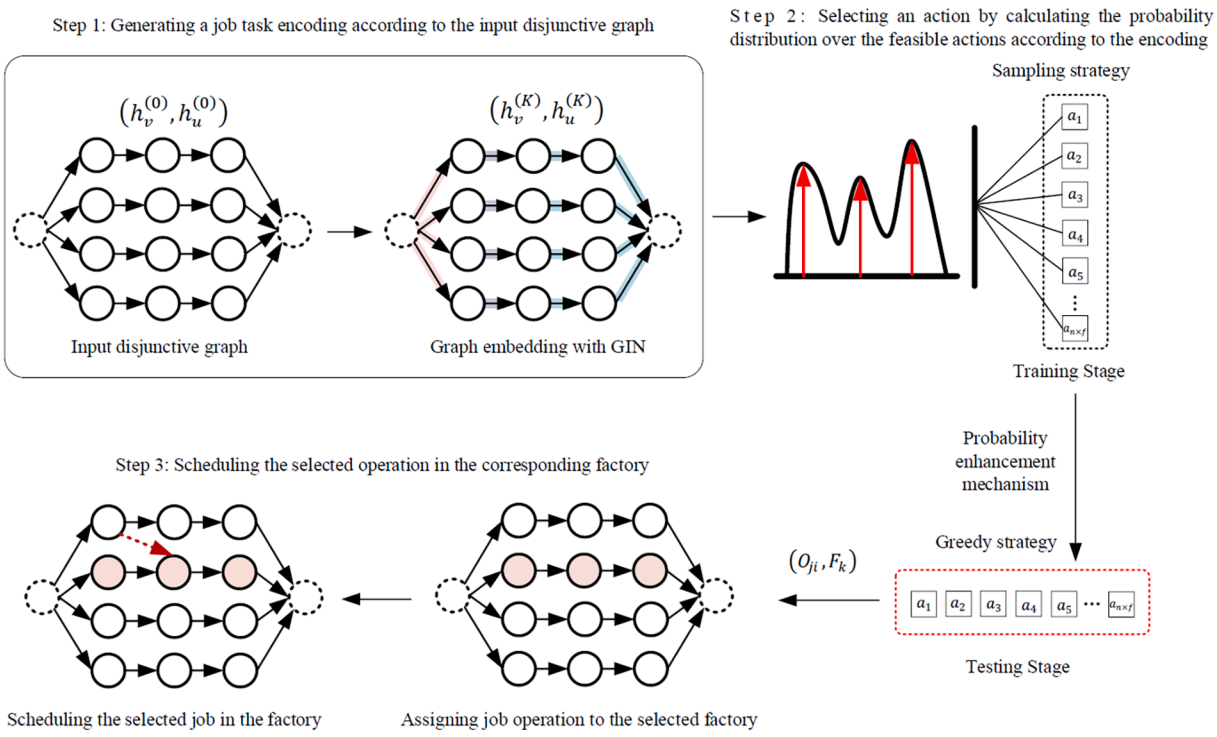


Fig. 3. Entire framework for scheduling a job operation.

selected operation in the corresponding factory. The three steps correspond to job task encoding, action selection and scheduling according to the selected action. The entire framework for scheduling a job operation is presented in Fig. 3.

4.2.1. Job task encoding

Job task encoding refers to embedding, and it is a compressed feature representation. Large feature vectors associated with vertices and edges can be reduced to low dimensional embeddings. The stitched disjunctive graph proposed in Section 3 has presented a comprehensive view of the scheduling states, and almost all necessary information can be obtained from the graph, including the precedence constraints among operations of one job, the processing order of operations on one machine, the available time of each machine and the processing time needed by each operation. Therefore, it inspires us to exploit the rich graph with a GNN.

To extract the feature embedding of each node, Graph Isomorphism Network (GIN) (Xu et al., 2019) is adopted, GIN is a powerful and simple variant of GNNs. The computation procedure of embedding layer in GIN is given as

$$h_v^{(k)} = MLP_{\theta_k}^{(k)}\left(\left(1+\varepsilon^{(k)}\right)\bullet h_v^{(k-1)} + \sum_{u\in N(v)} h_u^{(k-1)}\right) \tag{5}$$

where $h_v^{(k)}$ represents the node embedding of vertex v at iteration k, and $h_v^{(0)}$ donates the raw information carried by vertex v for input. $MLP_{\theta_k}^{(k)}$ is a Multi-Layer Perceptron (MLP) with parameters θ_k which is obtained through k iterations. ε is a parameter that can be learned. $N(v)$ contains the neighborhood of vertex v. After K iterations, a global representation for the entire disjunctive graph can be extracted by an average pooling function

$$h_g = 1/\left(nm\sum_{v\in V} h_v^K\right) \tag{6}$$

where nm is number of vertices. In this paper, the MLP used in GIN is with 2 layers and 64 neurons in each layer. As mentioned in Section 4.1, the state has a 2-demensional feature at decision point, therefore, the input dimension of the input layer is 2.

4.2.2. Action selection

The job task encoding based on the GIN is used to provide the action selection a valuable information reference, so as to achieve an optimal or near-optimal decision. To ensure the job task encoding can extract the critical state features, the GIN parameters are trained and the details is described in next section. In the MDP formulation for DJSP, action a_t taken at decision point t guides the agent to assign one available operation to an appropriate factory. To obtain a scheduling policy, an actor network $\pi(a_t|s_t)$ is introduced (described in Subsection 4.3). The actor produces a probability distribution over the available actions with

$$p(a_t) = \frac{exp\left(MLP_{\theta_\pi}\left(\left[h_{a_t}^{(K)}, h_g(s_t)\right]\right)\right)}{\sum_{a_t\in A_t} exp\left(MLP_{\theta_\pi}\left(\left[h_{a_t}^{(K)}, h_g(s_t)\right]\right)\right)} \tag{7}$$

where θ_π is the parameters of the actor network. The stochastic and greedy policies are used to select an action from the action space in the training and testing stage respectively. The stochastic policy means sampling an action by referring to the probability distribution so as to construct a feasible solution, and helps to explore the environment to achieve a good training effect. While the greedy policy choses an action with the highest probability at each decision point to form an effective solution, and it is conductive to realizing the effect of the trained models.

For DJSP, the factory assignment of jobs is critical to achieve an optimal scheduling scheme. Therefore, a skillful probability enhancement mechanism is explored and used in the testing stage. As mentioned, minimizing the makespan means minimizing the maximum makespan among factories. In the testing stage, through experiments, it is found that when a probability distribution over the action space is generated by observing the current scheduling environment, the greedily chosen operations can cause the excessive job accumulation in some factory, which is not beneficial for the makespan minimization. To alleviate such case, the makespans of different factories are also taken into consideration. Based on the probability distribution generated by the agent, the reciprocal of the factory makespan is plus to form the probability distribution. For example, the action space shown in Fig. 2 (b) is operation-factory pairs (O_{12},F_1), (O_{12},F_1), (O_{21},F_1), (O_{21},F_2), (O_{31},F_1), (O_{31},F_2), and the probability over the actions assumes as

(0.167, 0.167, 0.154, 0.173, 0.155, 0.184). The makespan of factory 1 is 126, and that of factory 2 is 158, with the probability enhancement mechanism, the probability distribution is transformed to (0.167 + 1/126, 0.167 + 1/126, 0.154 + 1/126, 0.173 + 1/158, 0.187 + 1/126, 0.184 + 1/158), *i.e.*, (0.174, 0.174, 0.162, 0.179, 0.195, 0.190). In this way, the selected action is turned from (O_{31}, F_2) with probability 0.184 to (O_{31}, F_1) with probability 0.195. Obviously, the probability enhancement mechanism helps the agent to assign the jobs to the factory with smaller makespan, and thus a better decision can be made.

4.3. Proximal policy optimization

A PPO (Schulman et al., 2017) is a promising variant of policy gradient method, and it can improve the network training efficiency. PPO is also a kind of actor-critic architecture, and is employed to train the proposed model. It applies *MLP* s to calculate the probability distribution over the feasible actions and the evaluation value of action that is taken. The actor is formed by a *MLP* with 2 hidden layers of 32 neurons, and each layer is with a *tanh* activation function. The critic shares the same *MLP* network architecture with the actor, and it takes the global information $h_g(s_t)$ as input to get a value that evaluates how good the action taken at state s_t.

4.4. Training details

PPO is used to train the proposed agent in this paper. As the original paper (Schulman et al., 2017), ϖ parallel actors are established, each of which addresses a DJSP instance that are generated randomly. The data collected by the ϖ actors are used to update the parameters of the network. Assume Ψ iterations are performed during training, and each actor runs Γ timesteps at each iteration. After Ψ iterations, one actor can collect $\Gamma\Psi$ timesteps of data. The surrogate loss on these $\Gamma\Psi$ timesteps of data are constructed, and it is optimized by Adam optimizer. In this paper, the trained method is the same as that of paper (Zhang et al., 2020), and more details are also available from the original paper of PPO (Schulman et al., 2017). The training method is presented is Algorithm 1, especially for the aggregate loss in Line 16, it consists of three parts, the policy surrogate $L_t^{CLIP}(\theta)$, the value function error $L_t^{VF}(\phi)$ and the entropy bonus $S[\pi_\theta](s_t)$, and c_p, c_v, and c_e are the policy loss coefficient, the value function loss coefficient and the entropy loss coefficient, respectively.

Algorithm 1: Training Method: PPO, Actor-Critic Style

Algorithm 1: Training Method: PPO, Actor-Critic Style

Input: number of iterations: Ψ; number of training steps: Γ; discounting factor: γ; update epoch: ρ; policy loss coefficient: c_p; value function loss coefficient: c_v; entropy loss coefficient: c_e; clipping ratio: ϵ.

Output: actor network π_θ; critic network π_ϕ;

1 Initialize network parameters: actor network π_θ; behavior actor network $\pi_{\theta_{old}}$; critic network π_ϕ
2 **for** $iteration = 1$ **to** Ψ **do**
3 Generate ϖ DJSP instances randomly
4 **for** $actor = 1$ **to** ϖ **do**
5 $t = 0$
6 **while** no one DJSP instance is done **do**
7 Sample a_t^{actor} based on policy $\pi_{\theta_{old}}(a_t^{actor}|s_t^{actor})$
8 Update the reward r_t^{actor} and the next state s_{t+1}^{actor}
9 Calculate advantage estimates by $\hat{A}_t^{actor} = \sum_0^t \gamma^t r_t^{actor} - \pi_\phi(s_t^{actor})$
10 $t = t + 1$
11 **if** s_t^{actor} is done **then**
12 **break**
13 **endif**
14 **endwhile**
15 Calculate the policy surrogate $L_t^{CLIP}(\theta)$, value function error $L_t^{VF}(\phi)$ and entropy bonus $S[\pi_\theta](s_t)$
16 Calculate aggregate losses: $L_t^{CLIP+VF+S}(\theta) = c_p L_t^{CLIP}(\theta) - c_v L_t^{VF}(\phi) + c_e S[\pi_\theta](s_t)$
17 **endfor**
18 Optimize network parameters with ρ epochs
19 $\theta_{old} \leftarrow \theta$
20 **Endfor**

Table 1
Parameters of the training method.

Parameters	Values
number of iterations:Ψ	1000
number of training steps:Γ	100
discounting factor:γ	1.0
update epoch: ρ	5
policy loss coefficient:c_p	2.0
value function loss coefficient:c_v	1.0
entropy loss coefficient: c_e	0.01
clipping ratio:ϵ	0.2

For the training method, the algorithm parameters are obtained by trial and error, and they are listed in Table 1. Other parameters of the model are all default as the set in *Pytorch*.

5. Experimental results and discussions

In this section, the details of training process are presented at first. Then, by referring to the related work, PDRs, RL-based methods and metaheuristics are the three most mainstream and efficient methods for scheduling problems. Therefore, to verify the effectiveness and generalization of the trained models, a comprehensive comparison among the proposed models and other PDRs, RL methods and metaheuristics are presented. The test benchmarks is Taillard's Benchmarks which is a classical test dataset for JSP, and it is also suitable for testing DJSP (Chaouch et al., 2019). Taillard's Benchmarks are divided into 8 groups $\{15\times15, 20\times15, 20\times20, 30\times15, 30\times20, 50\times15, 50\times20, 100\times20\}$, and each of which contains 10 instances. In the paper, the factory set is $\{2, 3, 4\}$. Therefore, a total of 240 ($8\times10\times3$) instances are used to test the performance of the proposed models, and the processing time of each job operation is between 1 and 99.

All experiments are operated on Intel(R) Xeon(R) W-3365 CPU @ 2.70 GHz and NVIDIA GeForce RTX 3090 Ti. The relative percentage increase (RPI) is used as the responsive variable to show the difference of the experimental results provided by different methods. It is defined as

$$RPI = \frac{Method_{sol} - Best_{sol}}{Best_{sol}} \times 100\% \tag{8}$$

where $Method_{sol}$ is the makespan provided by some method, and $Best_{sol}$ is the optimal result among all obtained results for the same instance. For the methods with random factors, such as the compared metaheuristics, ten repeats are performed, and the averages of the ten repeats are set as the final results from the methods.

5.1. The training process of GDRL

The proposed GDRL model is trained by randomly generated instances. 1000 iterations are performed for each model, and every 10 iterations, the model is validated on 10 instances which are generated ahead and fixed during training. The model would be saved if the average of the validation instances is smaller than the smallest average during training. 3 models are trained by instances with sizes $2\times15\times15$, $3\times15\times15$ and $4\times15\times15$, and the training times are 4.61 *h*, 5.49 *h* and 6.05 *h* respectively. The validation curves are shown in Fig. 4, which demonstrates the good convergence of GDRL.

5.2. Comparison with priority dispatching rules

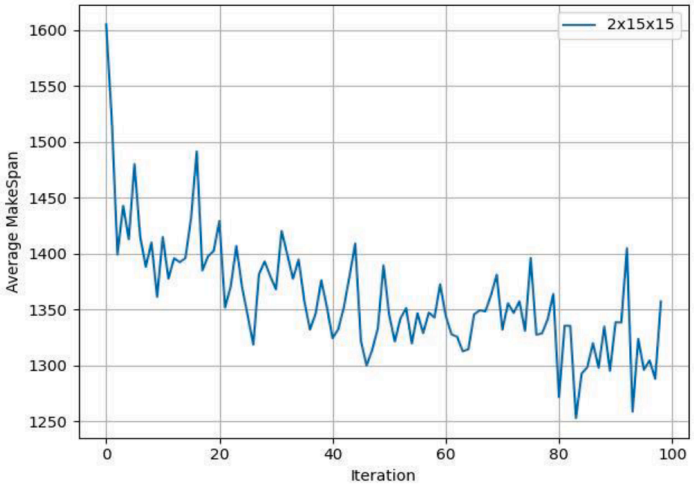
To confirm the effectiveness of the proposed GDRL, it is compared with other eight well-known PDRs (Sels et al., 2012): First In First Out (FIFO), Last In First Out (LIFO), Shortest Processing Time (SPT), Longest Processing Time (LPT), Shortest Remaining Processing Time (SRPT), Least Operation Remaining (LOR), Shortest Processing Time per Working Remaining (SPT/MWKR, SM) and Average Processing Time per Operation (AVPRO). These PDRs are originally proposed for JSP, therefore, a factory assignment rule is adopted to allocate jobs to each factory. Then the PDRs are used to schedule in each factory. For the factory assignment rule, all jobs are sorted in the ascending order of the total processing times of each job. Then the first f jobs are allocated in the f factories. Last, the remaining jobs are assigned to the factory with the minimal total processing time one by one.

As mentioned, Taillard's Benchmarks contains 10 instances for each instance size. In Table 2, for each compared method, the average RPI values of the 10 instances with the same size are calculated and recorded. The GDRL models trained with different instance sizes $2\times15\times15$, $3\times15\times15$ and $4\times15\times15$ are marked as GDRL $2\times15\times15$, GDRL $3\times15\times15$ and GDRL $4\times15\times15$ respectively. The best three average RPIs are in bold, and the best averages are in bold and italics. From Table 2, it is clear that the best three averages are produced by the three trained GDRL models. Especially, the GDRL model trained by instances with size $4\times15\times15$ performs best, and it brings the best averages for almost all of the instances except $2\times50\times15$ and $2\times100\times20$ whose best results are produced by the GDRL model $2\times15\times15$. To further analyze the experimental data, an ANOVA analysis is presented. The means and interaction plots with 95 % LSD intervals of the algorithms and other factors are shown in Fig. 5. Fig. 5(a) indicates that the trained GDRL models are the best three methods compared with the PDRs. Fig. 5 (b) shows that the performance of the proposed methods is seldom affected by the number of factories, and the model with four factories is the most stable one. The compared PDRs all get worse with the number of factories decreases. Fig. 5(c) tells how the number of jobs affects the algorithms' performance. Obviously, the trained models are hardly affected by the number of jobs. While other PDRs perform worse with the increasement of the number of jobs. Fig. 5(d) indicates that the proposed models bring the best results whatever the number of operations is. From the interaction plots among the algorithms and other factors, it can be concluded that the proposed models are the most stables, and they always occupy the first three places when compared with other PDRs.

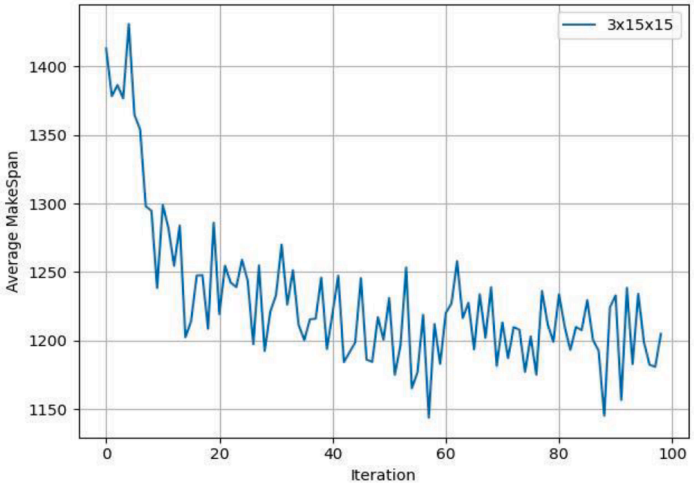
5.3. Comparison with RL methods

To see the superiority of the proposed GDRL, three RL methods are compared. Due to the absence of RL methods specifically designed for DJSP, the three methods which are originally for the close relative problems are adapted to solve DJSP. The first compared DRL (Zhang et al., 2020) called DRLZ in the paper is proposed for JSP, and it is also based on the GNN scheme. The second compared method is the Dueling Double DQN (DDQN) (Han & Yang, 2020), it combines convolutional neural network and disjunctive graph and is also proposed for JSP. The third method is a Q-learning algorithm (Fonseca-Reyna et al., 2018), and it is originally used to solve the flowshop scheduling problem.

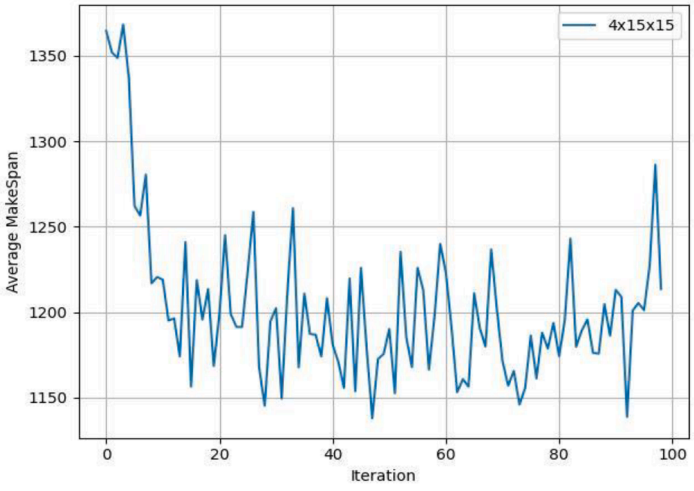
As in Table 2, Table 3 also records the average RPI values of the 10 instances with the same size for each compared RL method. The best three average RPIs are in bold, and the best results are in bold and italics. Whatever the instance sizes are, the GDRL model $4\times15\times15$ always ranks in the top three. Compared with GDRL model $4\times15\times15$, the performance of the other two GDRL models is a bit inferior. Although GDRL model $2\times15\times15$ gets worse when the number of factories is four, it is second only to GDRL model $4\times15\times15$ in terms of the overall performance. To have a detailed perception of the experimental data, an ANOVA analysis is done. The means and interaction plots of the compared algorithms and other factors are shown in Fig. 6. Fig. 6(a) is the means plot among the RL methods and the trained GDRL models, and it indicates the GDRL model $4\times15\times15$ is the best method, then the GDRL model $2\times15\times15$ follows. DDQN is slightly better than the GDRL model $3\times15\times15$. Fig. 6(b), (c) and (d) are the interactions plots among the compared algorithms and other factors. Fig. 6(b) shows that the model $4\times15\times15$ is least affected by the number of factories. Fig. 6(c) and (d) indicates that the number of jobs and the number of operations have little effect on the trained models' performance, especially for the



(a) $2 \times 15 \times 15$



(b) $3 \times 15 \times 15$



(c) $4 \times 15 \times 15$

Fig. 4. The training curves of 3 different instances.

Table 2
Average RPI values of the trained GDRL models and other PDRs.

Instance size	$n \times m$	Algorithms										
		FIFO	LIFO	SPT	LPT	SRPT	LOR	SM	AVPRO	GDRL $2 \times 15 \times 15$	GDRL $3 \times 15 \times 15$	GDRL $4 \times 15 \times 15$
$f = 2$	15×15	2.996	2.811	2.002	2.123	2.811	3.792	0.609	3.513	**0.073**	**0.117**	***0.005***
	20×15	3.409	3.438	2.344	2.573	3.440	3.852	0.668	4.056	**0.042**	**0.118**	***0.017***
	20×20	3.961	3.921	2.743	2.913	3.921	4.642	0.859	4.712	**0.063**	**0.184**	***0.005***
	30×15	4.350	4.410	2.996	3.350	4.392	5.054	0.867	5.197	***0.013***	**0.059**	**0.014**
	30×20	5.174	5.093	3.529	3.850	5.135	6.438	0.939	6.050	**0.063**	**0.124**	***0.010***
	50×15	6.025	5.889	3.958	4.239	5.891	6.406	1.343	6.556	***0.009***	**0.066**	**0.018**
	50×20	7.176	7.190	4.783	5.159	7.181	7.907	1.577	7.379	**0.028**	**0.108**	***0.017***
	100×20	9.922	9.786	6.572	7.102	9.798	10.588	2.207	10.605	***0.013***	**0.103**	**0.014**
	Mean	5.377	5.317	3.616	3.914	5.321	6.085	1.134	6.009	**0.038**	**0.110**	***0.013***
$f = 3$	15×15	1.851	1.829	1.212	1.525	1.829	2.783	0.346	2.413	**0.081**	**0.081**	***0.010***
	20×15	2.429	2.422	1.704	1.842	2.422	3.362	0.568	3.479	**0.060**	**0.212**	***0.004***
	20×20	2.775	2.732	1.817	2.036	2.729	3.466	0.529	3.780	**0.095**	**0.161**	***0.026***
	30×15	3.306	3.239	2.212	2.465	3.239	4.224	0.646	4.185	**0.083**	**0.160**	***0.009***
	30×20	3.630	3.614	2.527	2.914	3.614	4.701	0.644	4.518	**0.083**	**0.193**	***0.004***
	50×15	4.586	4.665	2.928	3.355	4.645	5.635	0.915	5.642	**0.038**	**0.158**	***0.010***
	50×20	5.432	5.541	3.652	4.007	5.557	6.411	1.202	6.436	**0.125**	**0.275**	***0.013***
	100×20	8.049	7.946	5.372	5.888	7.944	9.293	1.897	9.272	**0.054**	**0.155**	***0.007***
	Mean	4.007	4.000	2.678	3.004	3.997	4.984	0.843	4.966	**0.077**	**0.174**	***0.010***
$f = 4$	15×15	1.458	1.364	0.969	1.113	1.364	2.639	0.282	2.188	**0.079**	**0.141**	***0.009***
	20×15	1.753	1.732	1.316	1.448	1.739	2.607	0.345	2.805	**0.058**	**0.139**	***0.019***
	20×20	1.902	1.879	1.292	1.454	1.879	3.032	0.320	2.833	**0.079**	**0.094**	***0.011***
	30×15	2.645	2.528	1.779	1.949	2.528	3.478	0.485	3.694	**0.106**	**0.190**	***0.000***
	30×20	2.888	2.811	1.859	2.226	2.837	3.681	0.470	3.959	**0.090**	**0.165**	***0.006***
	50×15	3.681	3.631	2.506	2.702	3.631	4.987	0.756	4.557	**0.059**	**0.197**	***0.003***
	50×20	4.114	4.143	2.909	3.116	4.143	5.355	0.831	5.143	**0.146**	**0.341**	***0.009***
	100×20	6.973	6.894	4.651	5.061	6.894	8.090	1.561	8.207	**0.146**	**0.294**	***0.000***
	Mean	3.177	3.123	2.160	2.384	3.127	4.234	0.631	4.173	**0.095**	**0.195**	***0.007***

model $4 \times 15 \times 15$. Therefore, the results among the RL methods and the trained GDRL models tell that the proposed GDRL is the best algorithm in terms of the effectiveness and stability.

5.4. Comparison with metaheuristics

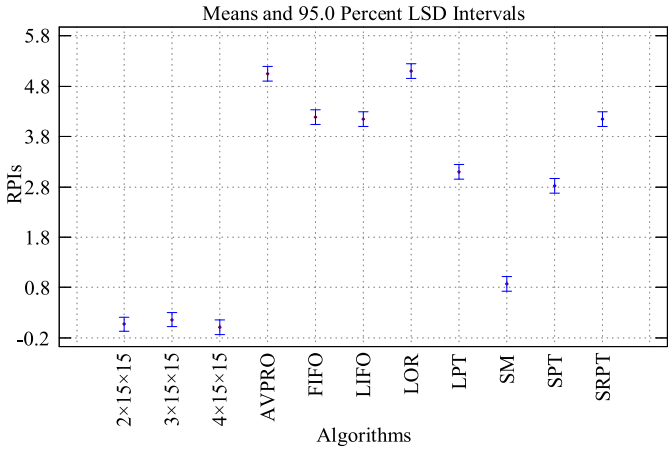
Metaheuristics are a kind of representative methods for solving combinatorial optimization problems, and they can solve the problems with near-optimal and even optimal solutions. Several metaheuristics have been proposed for DJSP. To further verify the effectiveness of the proposed GDRL, five state-of-art metaheuristics that are proposed for DJSP are compared. The first two are Simulated Annealing (SA) algorithm and Hybrid SA (HSA), both of them are from paper (Naderi & Azab, 2015). The remaining three are Ant Colony Optimization (ACO), Hybrid ACO (HACO) and Dynamic Assignment method of jobs to factories with a HACO (DAHACO), and they are proposed by Chaouch et al. (Chaouch et al., 2019). ACO, HACO, DAHACO are all used to solve Taillard's Benchmarks in the original paper, and the results of them are directly compared with the results from the trained GDRL models. SA and HSA do not address Taillard's Benchmarks by the authors, therefore they are recoded to solve the instances. The running time is set as $\frac{30 \times n \times m \times f}{1000}$ seconds. The algorithm parameters are referred to the original paper.

The data in Table 4 are obtained by the same way as in Table 2 and Table 3. In Table 4, for all instances, at least one of the trained GDRL models can get the top three results. When the number of factories is 2 and 3, the best results are all produced by the trained GDRL models. Especially, the GDRL model $4 \times 15 \times 15$ performs best. Although DAHACO shows good performance on a few instances $2 \times 15 \times 15$, $2 \times 20 \times 15$, $2 \times 20 \times 20$, $3 \times 20 \times 15$ and $3 \times 50 \times 20$, its performance is still not as good as that of the trained GDRL models. When the number of factories is 4, the trained GDRL model $4 \times 15 \times 15$ still shows the best advantage. Although the other two GDRL models are not as good as HACO and DAHACO in some instances, but their means ($2 \times 15 \times 15$: 0.128; $3 \times 15 \times 15$: 0.230) is still smaller than those of HACO (0.325) and DAHACO (0.254). Fig. 7 shows the results of ANOVA analysis of the experimental data. The means plot of the compared metaheuristics and the proposed models is shown in Fig. 7(a), on the whole, the three trained models are still the best three. Fig. 7(b), (c) and (d) indicates the proposed models are affected little by the number of factories, the number of jobs and the number of operations, especially, the model $4 \times 15 \times 15$ is the most stable one. In contrast, the performance of other metaheuristics is more affected by the instance scales.

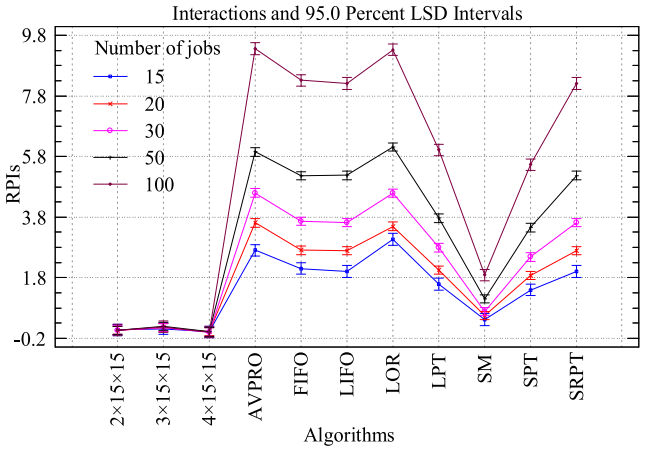
5.5. Discussions

From the analyses mentioned above, it can be concluded that the trained GDRL models are the most efficient, and they always produce the best results when compared with the PDRs, RL methods and the metaheuristics. By referring to the interaction plots with 95 % LSD intervals of the algorithms and other factors, the trained GDRL models are also the most stable ones, and their performance is seldom affected by the instance scales. In addition, whatever the instance size used for training the GDRL models is, the trained GDRL models can be applied to solve the instances with arbitrary sizes, it demonstrates that the proposed GDRL method is with a significant generalization. As a whole, the proposed GDRL method is the best one in terms of the effectiveness, the stability and the generalization.

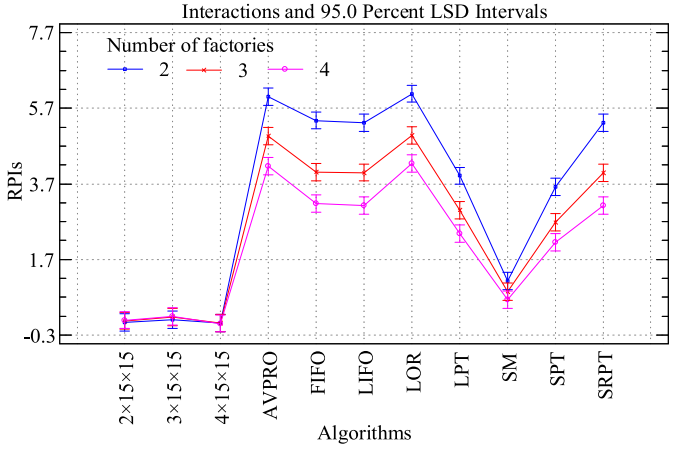
Some reasons for the good performance of the proposed method are presented. First, the stitched disjunctive graph representation and the vertex features lay a groundwork for the problem characteristics embedding. Then, the application of the GNN during the feature extraction also plays an important role, which provides a critical reference for the agent to make a better decision, and it also weakens the impact of instance size on algorithm performance, and ensures the generalization of the GDRL model. Third, the design of the MDP of DJSP is also creative. Especially, the actions based on operation-factory pairs make the job selection and the job assignment been simultaneously done, which facilitates the decision process. Fourth, the probability enhancement mechanism is used to weaken the ability of the agent to allocate jobs to the factory with larger makespan, and avoids the excessive job accumulation in some factory. Last, the sophisticated



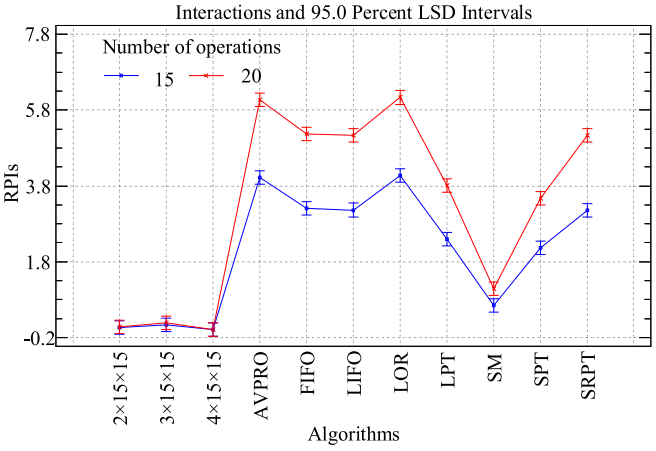
(a) Means plot of PDRs and the proposed models



(c) Interaction plots between algorithms and number of Jobs



(b) Interaction plots between algorithms and number of factories

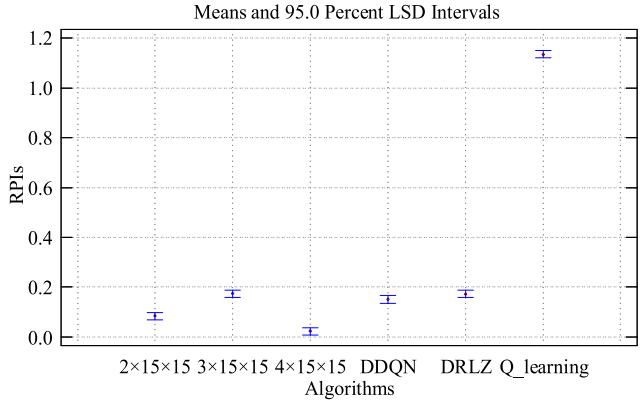


(d) Interaction plots between algorithms and number of operations

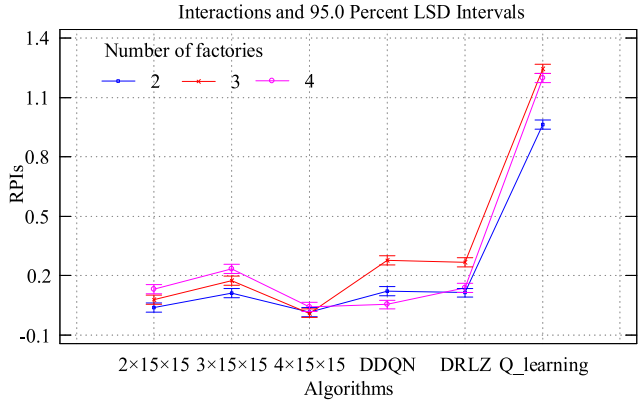
Fig. 5. Means and interaction plots with 95% LSD intervals of PDRs and the proposed models.

Table 3
Average RPI values of the trained GDRL models and other RL methods.

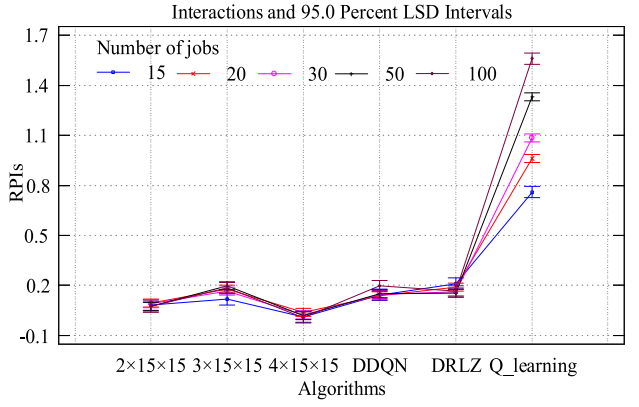
Instance size	$n \times m$	Algorithms					
		DRLZ	DDQN	Q-learning	GDRL $2 \times 15 \times 15$	GDRL $3 \times 15 \times 15$	GDRL $4 \times 15 \times 15$
$f = 2$	15×15	0.142	**0.113**	0.641	**0.080**	0.125	***0.012***
	20×15	**0.114**	0.137	0.946	**0.047**	0.124	***0.023***
	20×20	0.151	**0.111**	0.827	**0.063**	0.184	***0.005***
	30×15	0.121	0.149	0.935	***0.013***	**0.059**	**0.014**
	30×20	**0.105**	0.107	0.900	**0.063**	0.124	***0.010***
	50×15	0.103	0.161	1.170	***0.009***	**0.066**	**0.018**
	50×20	**0.091**	0.057	1.079	**0.031**	0.112	***0.020***
	100×20	**0.080**	0.137	1.200	***0.013***	0.103	**0.014**
	Mean	0.113	0.122	0.962	**0.039**	**0.112**	***0.015***
$f = 3$	15×15	0.243	0.213	0.788	**0.081**	**0.081**	***0.010***
	20×15	0.234	0.262	1.153	**0.060**	**0.212**	***0.004***
	20×20	0.240	0.195	0.966	**0.095**	**0.161**	***0.026***
	30×15	0.291	0.323	1.234	**0.083**	**0.160**	***0.009***
	30×20	0.219	0.214	1.096	**0.083**	**0.193**	***0.004***
	50×15	0.311	0.377	1.578	**0.038**	**0.158**	***0.010***
	50×20	0.269	**0.228**	1.415	**0.125**	0.275	***0.013***
	100×20	0.336	0.406	1.720	**0.054**	**0.155**	***0.007***
	Mean	0.268	0.277	1.244	**0.077**	**0.174**	***0.010***
$f = 4$	15×15	0.245	**0.106**	0.851	**0.082**	0.144	***0.012***
	20×15	0.174	**0.120**	0.864	**0.058**	0.139	***0.019***
	20×20	**0.232**	***0.000***	1.016	0.238	0.253	**0.161**
	30×15	0.119	**0.067**	**1.116**	0.118	0.206	***0.011***
	30×20	**0.118**	***0.008***	1.233	0.174	0.259	**0.085**
	50×15	0.070	**0.043**	1.434	**0.064**	0.203	***0.008***
	50×20	**0.074**	**0.036**	1.316	0.176	0.378	***0.036***
	100×20	**0.077**	**0.051**	1.760	0.151	0.299	***0.004***
	Mean	0.139	**0.054**	1.199	**0.133**	0.235	***0.042***



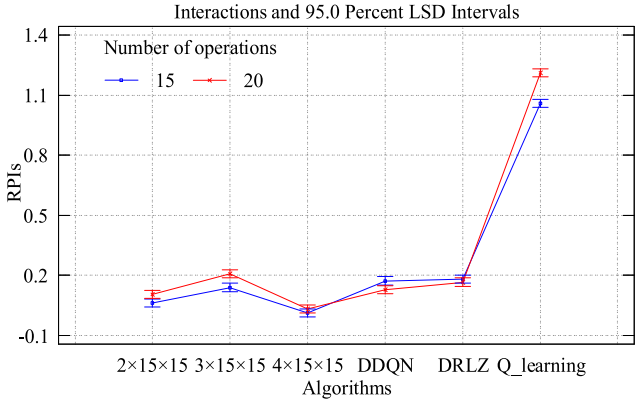
(a) Means plot of other RL methods and the proposed models



(b) Interaction plots between algorithms and number of factories



(c) Interaction plots between algorithms and number of jobs

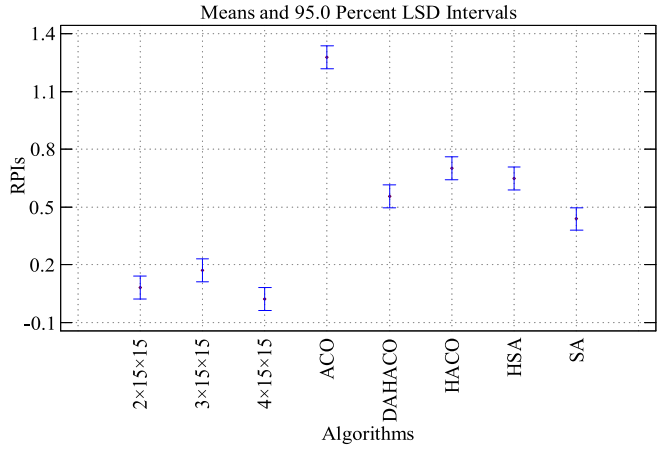


(d) Interaction plots between algorithms and number of operations

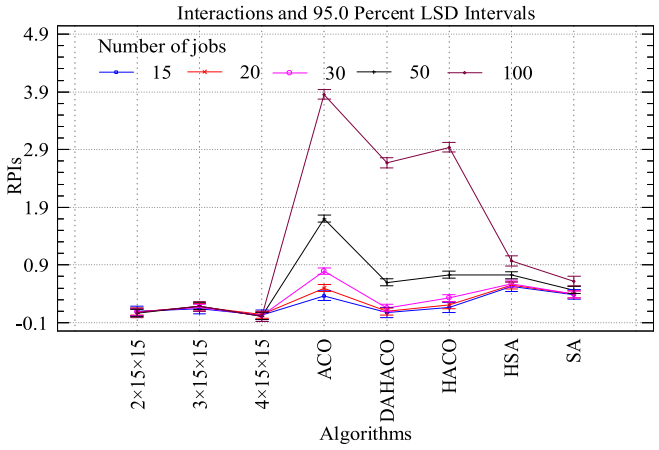
Fig. 6. Means and interaction plots with 95% LSD intervals of the RL methods and the proposed models.

Table 4
Average RPI values of the trained GDRL models and other metaheuristics.

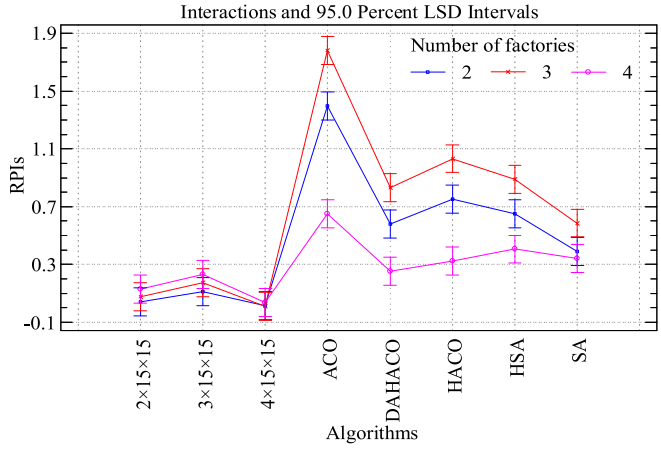
Instance size	$n \times m$	Algorithms							
		SA	HSA	ACO	HACO	DAHACO	GDRL $2 \times 15 \times 15$	GDRL $3 \times 15 \times 15$	GDRL $4 \times 15 \times 15$
$f = 2$	15×15	0.299	0.476	0.430	0.172	**0.070**	**0.089**	0.133	***0.020***
	20×15	0.299	0.501	0.540	0.177	**0.087**	**0.042**	0.118	***0.017***
	20×20	0.362	0.617	0.604	0.238	**0.086**	**0.063**	0.184	***0.005***
	30×15	0.340	0.564	0.733	0.380	0.187	***0.013***	**0.059**	**0.014**
	30×20	0.392	0.633	1.064	0.342	**0.117**	**0.063**	0.124	***0.010***
	50×15	0.431	0.734	1.640	0.577	0.292	***0.009***	**0.066**	**0.018**
	50×20	0.454	0.746	2.099	1.126	1.094	**0.028**	**0.108**	***0.017***
	100×20	0.530	0.937	4.060	2.990	2.694	***0.013***	**0.103**	**0.014**
	Mean	0.388	0.651	1.396	0.750	0.578	**0.040**	**0.112**	***0.014***
$f = 3$	15×15	0.404	0.594	0.548	0.267	0.157	**0.081**	**0.081**	***0.010***
	20×15	0.450	0.673	0.715	0.313	**0.209**	**0.060**	0.212	***0.004***
	20×20	0.465	0.741	0.729	0.334	0.170	**0.095**	**0.161**	***0.026***
	30×15	0.548	0.804	0.999	0.593	0.370	**0.083**	**0.160**	***0.009***
	30×20	0.534	0.798	1.270	0.477	0.229	**0.083**	**0.193**	***0.004***
	50×15	0.702	1.060	2.140	0.883	0.536	**0.038**	**0.158**	***0.010***
	50×20	1.036	2.613	1.466	1.430	**0.125**	**0.275**	***0.013***	1.036
	100×20	0.893	1.391	5.252	3.927	3.563	**0.054**	**0.155**	***0.007***
	Mean	0.629	1.084	1.640	1.028	0.670	***0.096***	**0.142**	**0.138**
$f = 4$	15×15	0.457	0.510	0.122	**0.051**	***0.001***	0.139	0.208	**0.068**
	20×15	0.379	0.370	0.161	**0.078**	***0.007***	0.095	0.178	**0.056**
	20×20	0.402	0.375	0.253	**0.082**	***0.002***	0.189	0.203	**0.115**
	30×15	0.269	0.311	0.315	**0.080**	**0.030**	0.121	0.209	***0.015***
	30×20	0.271	0.335	0.385	**0.085**	***0.013***	0.127	0.207	**0.042**
	50×15	0.258	0.379	0.791	0.233	**0.142**	**0.059**	0.197	***0.003***
	50×20	0.247	0.383	0.933	**0.096**	**0.086**	0.146	0.341	***0.009***
	100×20	0.434	0.580	2.251	1.894	1.752	**0.146**	**0.294**	***0.000***
	Mean	0.340	0.405	0.651	0.325	0.254	**0.128**	**0.230**	***0.039***



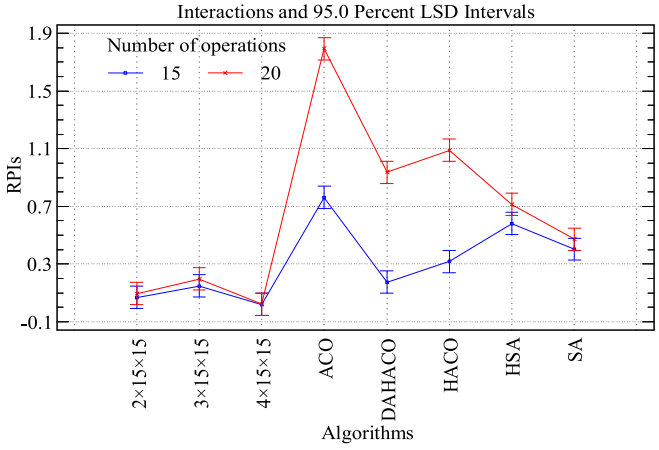
(a) Means plot of other metaheuristics and the proposed models



(c) Interaction plots between algorithms and number of jobs



(b) Interaction plots between algorithms and Number of Factories



(d) Interaction plots between algorithms and number of operations

Fig. 7. Means and interaction plots with 95% LSD intervals of the metaheuristics and the proposed models.

training process helps a lot for the models' good performance.

6. Conclusion and further work

In this paper, an end-to-end GNN-based DRL method is proposed for DJSP. Considering the distributed characteristics of DJSP, a stitched disjunctive graph representation is proposed, and a MDP for DJSP is designed based on the solution representation. The GNN-based policy network can fully extract the inner connections among the disjunctive graph and achieve a wise action selection at each decision point, and it is with good effectiveness, stability and generalization. The explorations made in the paper show that the GNN has an excellent feature extraction ability for the data with graph structure. In the shop scheduling field, the disjunctive graph is a classical and important solution representation method, and it facilitates the application of the GNN in the shop scheduling problems. With the facilitation provided by the disjunctive graph, to achieve better performance, a careful design of the vertex features based the specific problem is also critical, which provides the GNN with an intuitive insight on the problem solving, and the more comprehensive view can be obtained by the GNN, and thus a wiser decision can be made by the agent.

Though the proposed GDRL method has been proved effective, stable and generalizable, it has something for improvement. In terms of its performance on the instances with four factories, it can be found that the best results are not all obtained by the proposed GDRL models. This prompts us to explore more efficient DRL methods for distributed scheduling problems. In addition, in the real production systems, there are many dynamic events, such as job arrivals and random machine breakdowns, and so on. The proposed framework also shows a potential for solving real-time scheduling problems. Therefore, it would be interesting to extend the proposed GNN-based policy to the dynamic distributed shop scheduling problems.

CRediT authorship contribution statement

Jiang-Ping Huang: Conceptualization, Methodology, Writing – original draft. **Liang Gao:** Supervision, Project administration. **Xin-Yu Li:** Supervision, Writing – review & editing, Project administration.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

Data will be made available on request.

Acknowledgements

This work was supported in part by the National Science Foundation of China under Grants 51825502, U21B2029 and 52188102, and the Key R&D Program of Hubei Province under Grant No. 2021AAB001.

References

Błażewicz, J., Pesch, E., & Sterna, M. (2000). The disjunctive graph machine representation of the job shop scheduling problem. *European Journal of Operational Research, 127*(2), 317–331. https://doi.org/10.1016/S0377-2217(99)00486-5

Chang, H. C., & Liu, T. K. (2017). Optimisation of distributed manufacturing flexible job shop scheduling by using hybrid genetic algorithms. *Journal of Intelligent Manufacturing, 28*(8), 1973–1986. https://doi.org/10.1007/s10845-015-1084-y

Chaouch, I., Driss, O. B., & Ghedira, K. (2019). A novel dynamic assignment rule for the distributed job shop scheduling problem using a hybrid ant-based algorithm. *Applied Intelligence, 49*(5), 1903–1924. https://doi.org/10.1007/s10489-018-1343-7

De Giovanni, L., & Pezzella, F. (2010). An improved genetic algorithm for the distributed and flexible job-shop scheduling problem. *European Journal of Operational Research, 200*(2), 395–408. https://doi.org/10.1016/j.ejor.2009.01.008

Du, Y., Li, J. Q., Luo, C., & Meng, L. L. (2021). A hybrid estimation of distribution algorithm for distributed flexible job shop scheduling with crane transportations. *Swarm and Evolutionary Computation, 62*, Article 100861. https://doi.org/10.1016/j.swevo.2021.100861

Fonseca-Reyna, Y. C., Martínez-Jiménez, Y., & Nowé, A. (2018). Q-learning algorithm performance for m-machine, n-jobs flow shop scheduling problems to minimize makespan. *Investigación Operacional, 38*(3), 3.

Palombarini, J. A., & Martinez, E. C. (2022). End-to-end on-line rescheduling from Gantt chart images using deep reinforcement learning. *International Journal of Production Research, 60*(14), 4434–4463. https://doi.org/10.1080/00207543.2021.2002963

Gu, Y. M., Chen, M., & Wang, L. (2023). A self-learning discrete salp swarm algorithm based on deep reinforcement learning for dynamic job shop scheduling problem. *Applied Intelligence, 53*, 18925–18958. https://doi.org/10.1007/s10489-023-04479-7

Guo, W. X., Tian, W. H., Ye, Y. F., Xu, L. X., & Wu, K. (2021). Cloud resource scheduling with deep reinforcement learning and imitation learning. *IEEE Internet of Things Journal, 8*(5), 3576–3586. https://doi.org/10.1109/JIOT.2020.3025015

Han, B. A., & Yang, J. J. (2020). Research on adaptive job shop scheduling problems based on dueling double dqn. *IEEE Access, 8*, 186474–186495. https://doi.org/10.1109/ACCESS.2020.3029868

Han, B. A., & Yang, J. J. (2021). A deep reinforcement learning based solution for flexible job shop scheduling problem. *International Journal of Simulation Modelling, 20*(2), 375–386. https://doi.org/10.2507/IJSIMM20-2-CO7

Huang, J. P., Pan, Q. K., Miao, Z. H., & Gao, L. (2021). Effective constructive heuristics and discrete bee colony optimization for distributed flowshop with setup times. *Engineering Applications of Artificial Intelligence, 97*, Article 104016. https://doi.org/10.1016/j.engappai.2020.104016

Huang, Y. Y., Pan, Q. K., Huang, J. P., Suganthan, P., & Gao, L. (2021). An improved iterated greedy algorithm for the distributed assembly permutation flowshop scheduling problem. *Computers & Industrial Engineering, 152*, Article 107021. https://doi.org/10.1016/j.cie.2020.107021

Jia, H. Z., Nee, A. Y. C., Fuh, J. Y. H., & Zhang, Y. F. (2003). A modified genetic algorithm for distributed scheduling problems. *Journal of Intelligent Manufacturing, 14*(3), 351–362.

da Jiang, E., Wang, L., Peng, & Ping, Z. (2020). Solving energy-efficient distributed job shop scheduling via multi-objective evolutionary algorithm with decomposition. *Swarm and Evolutionary Computation, 58*, Article 100745.

Jing, X. L., Pan, Q. K., Gao, L., & Wang, Y. L. (2020). An effective Iterated Greedy algorithm for the distributed permutation flowshop scheduling with due windows. *Applied Soft Computing, 96*, Article 106629.

Kong, Z. Q., Jin, X. H., Xu, Z. G., & Zhang, B. (2022). Spatio-temporal fusion attention: A novel approach for remaining useful life prediction based on graph neural network. *IEEE Transactions on Instrumentation and Measurement, 71*, 3515912.

Lei, K., Guo, P., Zhao, W. C., Wang, Y., Qian, L. M., Meng, X. Y., & Tang, L. S. (2022). A multi-action deep reinforcement learning framework for flexible Job-shop scheduling problem. *Expert Systems with Applications, 205*, Article 117796.

Lin, B., Ghaddar, B., & Nathwani, J. (2022). Deep reinforcement learning for the electric vehicle routing problem with time windows. *IEEE Transactions on Intelligent Transportation Systems, 23*(8), 11528–11538.

Liu, R., Piplani, R., & Toro, C. (2022). Deep reinforcement learning for dynamic scheduling of a flexible job shop. *International Journal of Production Research*, 1–21.

Luo, Q., Deng, Q. W., Gong, G. L., Zhang, L. K., Han, W. W., & Li, K. X. (2020). An efficient memetic algorithm for distributed flexible job shop scheduling problem with transfers. *Expert Systems with Applications, 160*, Article 113721.

Luo, S., Zhang, L. X., & Fan, Y. S. (2021). Real-time scheduling for dynamic partial-no-wait multiobjective flexible job shop by deep reinforcement learning. *IEEE Transactions on Automation Science and Engineering*, 1–19.

Meng, L. L., Zhang, C. Y., Ren, Y. P., Zhang, B., & Lv, C. (2020). Mixed-integer linear programming and constraint programming formulations for solving distributed flexible job shop scheduling problem. *Computers & Industrial Engineering, 142*, Article 106347.

Meng, T., Pan, Q. K., & Wang, L. (2019). A distributed permutation flowshop scheduling problem with the customer order constraint. *Knowledge-Based Systems, 184*, Article 104894.

Naderi, B., & Azab, A. (2014). Modeling and heuristics for scheduling of distributed job shops. *Expert Systems with Applications, 41*(17), 7754–7763.

Naderi, B., & Azab, A. (2015). An improved model and novel simulated annealing for distributed job shop problems. *The International Journal of Advanced Manufacturing Technology, 81*(1), 693–703.

Naderi, B., & Ruiz, R. (2010). The distributed permutation flowshop scheduling problem. *Computers & Operations Research, 37*(4), 754–768.

Okwudire, C. E., & Madhyastha, H. V. (2021). Distributed manufacturing for and by the masses. *Science, 372*, 341–342.

Pan, Q. K., Gao, L., & Wang, L. (2022). An effective cooperative co-evolutionary algorithm for distributed flowshop group scheduling problems. *IEEE Transactions on Cybernetics, 52*(7), 5999–6012.

Pan, Q. K., Gao, L., Wang, L., Liang, J., & Li, X. Y. (2019). Effective heuristics and metaheuristics to minimize total flowtime for the distributed permutation flowshop problem. *Expert Systems with Applications, 124*, 309–324.

Pan, Z. X., Wang, L., Wang, J. J., & Lu, J. W. (2021). Deep reinforcement learning based optimization algorithm for permutation flow-shop scheduling. *TEEE Transactions on Emerging Topics in Computational Intelligence, 7*(4), 983–994.

Ruan, J. H., Wang, Z. X., Chan, F. T. S., Patnaik, S., & Tiwari, M. K. (2021). A reinforcement learning-based algorithm for the aircraft maintenance routing problem. *Expert Systems with Applications, 169*, Article 114399.

Ruiz, R., Pan, Q. K., & Naderi, B. (2019). Iterated greedy methods for the distributed permutation flowshop scheduling problem. *Omega, 83*, 213–222.

Sabuncuoglu, I., & Karpat, Y. (2009). Process planning and scheduling for distributed manufacturing. *International Journal of Production Research, 47*(4), 1151–1152.

Şahman, M. A. (2021). A discrete spotted hyena optimizer for solving distributed job shop scheduling problems. *Applied Soft Computing, 106*, Article 107349.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., & Klimov, O. (2017). *Proximal policy optimization algorithms* (arXiv:1707.06347). arXiv.

Sels, V., Gheysen, N., & Vanhoucke, M. (2012). A comparison of priority rules for the job shop scheduling problem under different flow time- and tardiness-related objective functions. *International Journal of Production Research, 50*(15), 4255–4270.

Wang, D., & Deng, H. (2021). Multirobot coordination with deep reinforcement learning in complex environments. *Expert Systems with Applications, 180*, Article 115128.

Wang, J. J., & Wang, L. (2020). A knowledge-based cooperative algorithm for energy-efficient scheduling of distributed flow-shop. *IEEE Transactions on Systems, Man, and Cybernetics: Systems, 50*(5), 1805–1819.

Wang, Z. Y., Eisen, M., & Ribeiro, A. (2022). Learning decentralized wireless resource allocations with graph neural networks. *IEEE Transactions on Signal Processing, 70*, 1850–1863.

Wang, Z. Y., & Gombolay, M. (2020). Learning scheduling policies for multi-robot coordination with graph attention networks. *IEEE Robotics and Automation Letters, 5*(3), 4509–4516.

Xu, K., Hu, W. H., Leskovec, J., & Jegelka, S. (2019). *How powerful are graph neural networks?* (arXiv:1810.00826). arXiv.

Yu, X., & Luo, W. (2023). Reinforcement learning-based multi-strategy cuckoo search algorithm for 3D UAV path planning. *Expert Systems with Applications, 223*, Article 119910.

Zhang, C., Song, W., Cao, Z. G., Zhang, J., Tan, P. S., & Chi, X. (2020). Learning to dispatch for job shop scheduling via deep reinforcement learning. *Advances in Neural Information Processing Systems, 33*, 1621–1632.

Zhang, G. H., Lu, X. X., Liu, X., Zhang, L. T., Wei, S. W., & Zhang, W. Q. (2022). An effective two-stage algorithm based on convolutional neural network for the bi-objective flexible job shop scheduling problem with machine breakdown. *Expert Systems with Applications, 203*, Article 117460.

Zhang, J. D., He, Z. X., Chan, W. H., & Chow, C. Y. (2023). DeepMGA: Deep reinforcement learning with multi-agent graphs for flexible job shop scheduling. *Knowledge-Based Systems, 259*, Article 110083.

Zhang, Y., Zhu, H., Tang, D., Zhou, T., & Gui, Y. (2022). Dynamic job shop scheduling based on deep reinforcement learning for multi-agent manufacturing systems. *Robotics and Computer-Integrated Manufacturing, 78*, Article 102412.

Zhang, Z., Liu, H., Zhou, M. C., & Wang, J. H. (2021). Solving dynamic traveling salesman problems with deep reinforcement learning. *IEEE Transactions on Neural Networks and Learning Systems*, 1–14.

Zhao, F. Q., Di, S. L., & Wang, L. (2023). A hyperheuristic with Q-learning for the multiobjective energy-efficient distributed blocking flow shop scheduling problem. *IEEE Transactions on Cybernetics, 53*(5), 3337–3350.

Zhao, F. Q., Wang, Z. Y., & Wang, L. (2022). A reinforcement learning driven artificial bee colony algorithm for distributed heterogeneous no-wait flowshop scheduling problem with sequence-dependent setup times. *IEEE Transactions on Automation Science and Engineering*, 1–16.

Zhao, F. Q., Zhang, H., & Wang, L. (2023). A Pareto-based discrete Jaya algorithm for multiobjective carbon-efficient distributed blocking flow shop scheduling problem. *IEEE Transactions on Industrial Informatics, 19*(8), 8588–8599.

Zhao, L. L., Fan, J. X., Zhang, C. J., Shen, W. M., & Zhuang, J. (2023). A DRL-based reactive scheduling policy for flexible job shops with random job arrivals. *IEEE Transactions on Automation Science and Engineering*, 1–12.

Zhao, F. Q., Zhang, L. X., Cao, J., & Tang, J. X. (2021). A cooperative water wave optimization algorithm with reinforcement learning for the distributed assembly no-idle flowshop scheduling problem. *Computers & Industrial Engineering, 153*, Article 107082.