Journal of Manufacturing Systems 69 (2023) 119–134

Contents lists available at ScienceDirect

Journal of Manufacturing Systems

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

Technical Paper

A novel priority dispatch rule generation method based on graph neural network and reinforcement learning for distributed job-shop scheduling

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

State Key Lab of Intelligent Manufacturing Equipment & Technology, Huazhong University of Science & Technology, Wuhan, 430074, P. R. China

ARTICLE INFO

Keywords:
Distributed job-shop
Scheduling
Reinforcement learning
Graph neural network

ABSTRACT

With the development of a global economy, distributed manufacturing becomes common in the industrial field. The Distributed Job-shop Scheduling Problem (DJSP), which is widespread in real-life production, is a hotspot in the academic field. The existing Priority Dispatch Rules (PDRs), which are used to assign a value to each waiting job according to some method and select the job with minimum or maximum "value" for next processing, are all relatively simple but lack self-learning ability, while the metaheuristics are all complex and with fixed evolutionary trajectory and cannot change with the manufacturing environment. This paper proposes a novel PDR generation method based on Graph Neural Network (GNN) and Reinforcement Learning (RL), which can self-learn and self-evolute by interacting with the scheduling environment. To combine DJSP with GNN closely, a new solution representation based on disjunctive graph is designed. DJSP is formulated as a Markov decision process, and the problem features and inner connections among the vertices of the disjunctive graph are fully explored by the GNN. An Actor-Critic RL method is applied to automatically train the network parameters to optimize the policy, so that it can be used to schedule the best action at each step. Comprehensive experiments on 240 test instances are conducted to evaluate the performance of the proposed method, and the results indicate that the proposed method shows greater effectiveness, generalizability and stability than other 8 classical PDRs, 5 metaheuristics and 3 RL-based methods.

1. Introduction

With the advent of Industry 4.0, distributed manufacturing which allows for geographically dispersed production is gaining attention. It can make full use of the resources from multiple companies or factories. Through rational allocation of resources, the products can be produced at a reasonable cost. In 2021, one paper in "Science" points that the distributed paradigm equips with rapid responsiveness and reliable resilience, and can cope with urgent production demands, promote customization and cost-effective, low-volume production, and reduce the dependence of production on the environment [1]. Recently, the distributed manufacturing problems have indeed drawn some attention [2,3]. The Distributed Job-shop Scheduling Problem (DJSP) is a typical representative of distributed manufacturing, and widely exists in the equipment manufacturing industry and other production scenes. Since DJSP is proposed [4], many researches have been investigated. Jia et al. [5] combine modified Genetic Algorithm (GA) and Gantt Chart to derive the factory combination and schedule. Naderi et al. [6] establish two Mixed Integer Linear Programs (MILPs), and design a factory assigning rule and three iterated greedy algorithms for DJSP. DJSP has been particularly important for the practical manufacturing and academic research. The classical Job Shop Problem (JSP) assumes that there is a single shop with m ($m \geq 2$) machines, and n ($n \geq 2$) jobs are operated on the machines in their own process route. It aims to schedule jobs on machines to minimize an objective, such as the maximum completion time of all jobs, *i.e.*, makespan ($C_{\max}$). JSP has been proved as a NP-hard problem, and the total number of all possible solutions is $(n!)^m$ [7]. DJSP with f identical factories is the generalization of JSP and is also a NP-hard problem [6]. DJSP aims to allocate n jobs to f factories, which has $\binom{n+f-1}{f-1}$ possible partitions. With the result, the total number of solutions of DJSP can be calculated by $\binom{n+f-1}{f-1}(n!)^m$. Specially, when f equals to 1, the case of JSP can be obtained. In the existing research, heuristics including Priority Dispatching Rules (PDRs) and metaheuristics show pretty good performance on such NP-hard problems. With the emergence of customized production, more and more

* Corresponding author.
E-mail address: lixinyu@mail.hust.edu.cn (X.-Y. Li).

https://doi.org/10.1016/j.jmsy.2023.06.007
Received 19 April 2023; Received in revised form 11 June 2023; Accepted 12 June 2023
Available online 18 June 2023

product types are brought, the rapidly changing production scenarios raise higher demands on the responsiveness and adaptability of the algorithms.

Graph Theory has an important application in solution representation for shop scheduling problems, and disjunctive graph is a classical way that shows the precedence constraints among operations of one job [8]. Chen and Tian [9] use a directed acyclic graph to represent each schedule, which clearly shows the dependency among the different jobs' schedules. Han and Yang [10] apply a 3D disjunction graph-based scheduling solution representation for FJSP. Moreover, in the field of deep learning, neural networks operated on graph data (called graph neural networks, or GNNs) have been developed for more than a decade [11]. Many GNN architectures have been proposed [12], such as graph convolutional networks, Graph Recurrent Neural Networks and so on. Among them, Graph Isomorphism Network (GIN) [13] is one of the simple and effective GNNs. The networks are practical in many areas such as decentralized wireless resource allocations [14], the prediction of remaining useful life of industrial equipment [15], and voltage stability control in power systems [16]. Even though the combination between the disjunctive graph and GNN is no longer a ground-breaking matter for solving the shop scheduling problems, the existing algorithms are always only for the problem with single factory. For example, Zhang et al. [8] have addressed the JSP by exploiting the disjunctive graph representation with the GNN, and the effectiveness have also been proved by the authors. However, all the existing methods take no consideration of the characteristics of the distributed paradigm, thus they may not work as well in solving distributed problems.

PDR is a classical heuristic method, and it has been widely applied in the real-word manufacturing facility such as the semiconductor manufacturing systems [17], the wafer fabrication factory [18] and the automatic guided vehicle systems [19]. Compared with the exact methods and the metaheuristics, PDR is intuitive, fast and time-saving, and they are widely applied to address all kinds of scheduling problems, especially for those with large size. For the complex scheduling situations lacking scheduling software, PDRs also have important application [20]. Many PDRs have been proposed for JSP, such as Short Processing Time (SPT), First In First Out (FIFO) [21]. However, a good PDR is based on rich domain knowledge and trial-and-error. Therefore, it is important to design a universal PDR generation method with self-learning and self-evolving ability, which can automatically mine problem features.

In recent years, the Deep Reinforcement Learning (DRL) technique has been an ideal tool for solving various issues, such as robotics and autonomous vehicles [22], combinatorial optimization problems [23], semiconductor wafer manufacturing [24], robotic flow shop scheduling [25] and Energy-Efficient Edge Computing for Internet of Vehicles [26], and so on. DRL is with considerably facilitated autonomy, and can learn the best action choice and react to the environment in real time. Besides, its strong generalization ability and fast speed of solving problem also deserve attention. More importantly, lots of algorithms based on DRL have been proposed for shop scheduling problems [27]. Inspired by the successful applications of DRL in different fields, in this paper, DRL is adopted to promote the self-learning and self-evolving ability of PDR generation method.

This paper proposes a PDR generation method based on GNN and RL for DJSP. In this method, first, by considering the characteristics of the DJSP, a simple heuristic is designed to make the factory assignment decisions, so as to achieve a factory load balance in a short time. Then, to effectively schedule the jobs in each factory, a complex **DRL** method based **G**NN (abbreviated as DRLG) is designed, where a GNN is applied to explore the connection among operation vertices of the solution's disjunctive graph. To improve the self-learning and self-evolving ability of the proposed method, DJSP is formulated as a Markov Decision Process (MDP), and an Actor-Critic RL method is used to automatically train the network parameters of GNN. To comprehensively observe valuable information in the environment, five critical features reflecting the static and dynamic characteristics during problem solving are recorded at each step, more details are presented in Section 4.1.1. In addition, the action space consisting of job operations and the state transition directly accelerate the method's responsiveness. To get a positive and reliable feedback from the environment, a reward mechanism based on distributed features and optimization goal is designed.

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

DJSP has gained much attention in the scheduling field since it was proposed [4]. Different approaches such as exact methods and metaheuristics have been proposed to address this problem. Jia et al. [5] present a GA based on Gantt Chart for solving JSP in distributed manufacturing systems. Naderi and Azab [28] propose a mathematical model and tackle the solution techniques for DJSP, and Simulated Annealing (SA) algorithms are developed to minimize makespan. An agent-based fuzzy constraint-directed negotiation mechanism is presented by Hsu et al. [29] to solve DJSP with minimization of makespan and average flow time. Chaouch et al. [30] propose an Ant Colony Optimization (ACO) algorithm to deal with DJSP with the object of minimizing the global makespan. The same authors propose a dynamic assignment rule for the same problem [31]. Jiang et al. [32] solves energy-efficient DJSP via multi-objective evolutionary algorithm with decomposition. Although many exact algorithms and metaheuristics have been proposed for DJSP, and they can get the optimal or near-optimal solutions. With the problem complexity of DJSP $\binom{n+f-1}{f-1}(n!)^m$ presented in the introduction, the number of the alternative solutions is in an explosive exponential growth with the size of the problem, and the difficulty to identify a better solution for DJSP is obvious. Thus, it is significant to explore an efficient method to solve the problem.

In practice, PDRs are often used to overcome the expensive computational cost. The first PDR is proposed by Jackson [33], and it is used to determine the priority of two flow processing machines. Naidu [34] presents a discuss on PDR for the single machine tardiness problem. All kinds of PDRs have been proposed to adapt to different scheduling problems [35,36]. Even PDRs are time-saving, their capacity of searching for the optimal solution is limited. Thus, they sometimes are used to generate the initial solution for the metaheuristics. Meng and Pan [3] propose six PDRs to initialize the population of the artificial bee colony algorithm for the distributed heterogeneous permutation flowshop scheduling problem (PFSP) with lot-streaming and carryover sequence-dependent setup time. Huang et al. [37] use a compose PDR to produce the initial solution for the improved iterated greedy algorithm. Besides, PDRs are also often combined with RL methods. Different PDRs are selected as the actions of RL methods [38,39]. The existing PDRs are simple, explainable and time-saving, and their performance has strong dependence on manufacturing scenes and problem constraints. Sometimes, manual experience can also lead to inappropriate selection of PDRs. Therefore, how to automatically generate efficient and adaptable PDRs has been a critical problem in the scheduling research.

Recent years, DRL has been an important choice to deal with different scheduling problems, and it can learn how to take the best action according to an observed state in a short time. Liu et al. [40] propose an Actor-Critic DRL to solve JSP, and different PDRs are treated as actions. Pan et al. [41] address PFSP with a DRL-based optimization algorithm, which escapes from the limitation of problem sizes. Han and Yang [10] propose an end-to-end DRL framework for FJSP. For the distributed shop scheduling problem, Yan et al. [42] design a deep Q

network (DQN) framework for distributed PFSP with flexible maintenance. For dynamic JSP, Zhao and Zhang [38] design a production control method with DRL, and present an explanation of the collaboration strategy. Wang et al. [43] take the dynamics and uncertainties of JSP into consideration, and propose a dynamic scheduling method based on DRL. For FJSP with random job arrivals, Chang et al. [39] apply a double DQN (DDQN) to optimize both earliness and tardiness. Liu et al. [44] propose a hierarchical and distributed architecture based on DDQN for dynamic FJSP, and facilitate a real-time scheduling. For PFSP with new job arrival, Yang et al. [45] adapt an Advantage Actor-Critic algorithm to train the scheduling agent. Hu et al. [46] solve a dynamic flexible manufacturing problem via DRL with graph convolutional network. Besides, DRL has also been used to address the production problems that have high real-time requirement. Lin et al. [47] investigate a smart manufacturing factory framework based on edge computing, and DRL is applied to reduce response time of production decisions. For a task allocation problem in human-machine manufacturing systems, Joo et al. [48] use a DRL-based approach to accommodate the unobservable characteristics of human operators and the stochastic nature of manufacturing systems. Park and Park [49] present a DRL method for scalable scheduling of semiconductor packaging facilities. Kong et al. [26] formulate energy-efficient edge computing for internet of vehicles into a RL problem, and propose a DRL-based method. As mentioned, DRL is a potential method and has been extensively studied in the scheduling field for all kinds of problems. First, DRL is the combination of deep learning and RL, which equips the ability to information-mining and self-learning simultaneously, and can learn an optimal strategy through the reward by interacting with the environment step by step. Second, if DRL sets some PDRs as its actions, it can be trained to automatically select the optimal PDR according to the observed state at each decision point, and produce a result in a short time, which surely performs better than a single PDR does [40,50]. Third, DRL can also work in an end-to-end way which can directly select the next processed job or operation, thus faster response and stronger generalizability can be ensured [10,51]. In comparison, a singe PDR that considers one characteristic during generating schedule can utilize limited scheduling information, which seems short-sighted [35]. While the metaheuristics search for solutions in an iterative way, which is time-consuming [28,30].

Okwudire et al. [1] point that distributed manufacturing is promising to improve the responsiveness and resilience of manufacturing to urgent production demands, where the efficient scheduling decision is important during production, and it can improve responsiveness of the system. Thus, higher demands on self-learning and self-evolving ability of scheduling methods have been raised. Besides, the manufacturing paradigm is changing from mass production with single-variety and high-volume to customization production with multi-variety and small-volume, which requires the enterprises manage production with diversified product configurations and trajectories of operations [44], where the scheduling problem is referred as JSP. DJSP as the typical representation of distributed manufacturing and the generalization of JSP, it takes the characteristics of distributed manufacturing and JSP into consideration simultaneously. Although some metaheuristics have been proposed for DJSP, they work in an iterative way which is time-consuming, and they can hardly evolve with changing environment which is with poor generalizability. While PDRs have been proved an intuitive and time-saving method. However, their capacity of searching for the optimal solution is limited, and there seems no specially-designed PDR for DJSP. DRL has a wide application in the shop scheduling problems and other manufacturing fields in recent years. The researches have shown the adaptivity of DRL in the manufacturing environment, and is with strong self-learning and self-evolving ability. However, the existing DRLs almost are proposed for the production scenario with one single factory, and take no consideration of the distributed properties of the distributed manufacturing problems. Therefore, it reasonable and essential to design a universal PDR generation method based DRL for DJSP.

Table 1
The processing times and executed machines of operations.

Jobs	J_1			J_2			J_3		
Operations	O_{11}	O_{12}	O_{13}	O_{21}	O_{22}	O_{23}	O_{31}	O_{32}	O_{33}
p_{ji}	2	3	2	3	3	2	4	3	2
$M(O_{ji})$	M_2	M_1	M_3	M_3	M_2	M_1	M_2	M_3	M_1
Jobs	J_4			J_5			J_6		
Operations	O_{41}	O_{42}	O_{43}	O_{51}	O_{52}	O_{53}	O_{61}	O_{62}	O_{63}
p_{ji}	3	3	2	3	3	4	2	3	4
$M(O_{ji})$	M_1	M_3	M_2	M_2	M_1	M_3	M_2	M_1	M_3

3. Problem description and solution representation

3.1. Problem description

The classical JSP assumes that there is only a single facility with a set of m machines. n independent jobs have their own process routes among the machines. As the extension of JSP, DJSP can be defined as follows. There are f identical factories, $F = \{F_1, F_2, \ldots, F_f\}$, each of which contains m machines, $M = \{M_1, M_2, \ldots, M_m\}$. A set of n jobs, $J = \{J_1, J_2, \ldots, J_n\}$, have to go through m machines in M. Any job J_j ($J_j \in J$) consists of m operations, $\{O_{j1}, O_{j2}, \ldots, O_{jm}\}$, each of which is processed on a certain machine. The processing time of O_{ji} ($1 \le j \le n$, $1 \le i \le m$) is p_{ji}, which is independent on the factory where the job is allocated. Any time, each machine can only process one job, and each job can only make one operation being processed. Once some job is placed into one factory, it cannot be reallocated to another factory. All jobs can be available at time 0. The capacity of machines is assumed unlimited, that is to say, all machines are capable of processing the jobs with no breakdowns. The process information of all jobs including the processing times and process routes are fixed and known ahead. The detailed MILP model of DJSP have been proposed by Naderi and Azab [28], which presents the definition of DJSP at the mathematical perspective. The factory assignment of jobs and the job scheduling in each factory should be optimized to minimize the maximum completion time of all the jobs as much as possible.

3.2. Solution representation

For JSP, the disjunctive graph is a classical solution representation [52]. The disjunctive graph is a directed graph $G = (V, C \cup D)$, where V denotes vertices set whose elements are corresponding to job operations, C is the conjunction set, wherein the precedence among operations of one job is constrained by conjunctive arcs. D is the disjunction set, wherein the operations that are executed on the same machine are connected by undirected arcs. Especially, V includes two extra dummy vertices, $\{S, T\}$, with zero processing time. All production tasks start from S and end with T. To get a complete schedule, the scheduler should turn each undirected disjunction to a directed conjunctive arc. Moreover, to ensure the optimization of the schedule, the longest path should be shortened as much as possible, because the longest path determines the makespan.

For DJSP, a common idea is that each factory takes a disjunctive graph to represent the scheduling of the jobs in it. In this way, a DJSP with f factories need f disjunctive graphs. To simplify the representation and facilitate the learning of the proposed algorithm (see Section 4), a stitched disjunctive graph is proposed, in which each vertex contains an added information, *i.e.*, the factory the corresponding operation is assigned. An example with 2 factories and 6 jobs is presented. The processing times and the executed machines are shown in Table 1. The raw information in Table 1 is expressed as a disjunctive graph in Fig. 1 (a), where black arrows are conjunctive arcs, the operations that are executed on the same machine are marked by the same color, and the

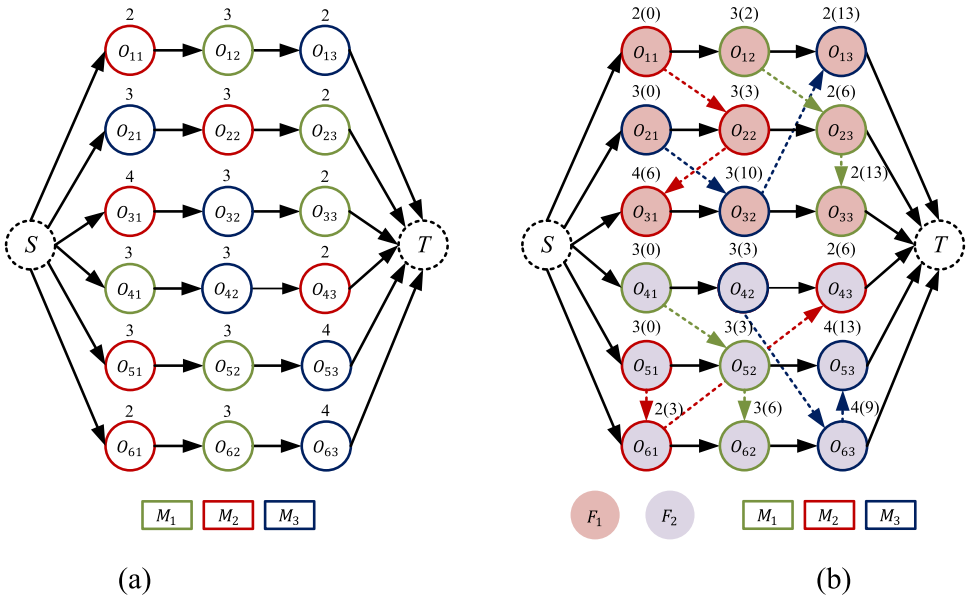


Fig. 1. The disjunctive graph of DJSP.

integers above the vertices are the processing times of the corresponding operations. Fig. 1(b) presents a complete solution to the example, where jobs 1, 2 and 3 are allocated in factory 1, and the corresponding operations are colored in pink, jobs 4, 5 and 6 are allocated in factory 2, and the corresponding operations are colored in purple. In each factory, the operations executed on the same machine are directed with dotted arrows with the same color, which describes the processing order. For example, operations O_{11}, O_{22} and O_{31} are executed in order on machine M_2, and directed with the red dotted arrow. The starting times of the scheduled operations are presented in bracket.

4. A PDR generation method based on GNN and RL for DJSP

For DJSP, two decisions should be made. One is allocating jobs to factories, and the other one is scheduling jobs in each factory [6]. A good job allocation method can provide a load balance among factories, and a good job sequencing method can greatly improve the production efficiency based on the balanced factory load. Additionally, due to the characteristics of DJSP that once a job is placed in one factory, it cannot be replaced into another one. It seems that the sequencing decisions play greater role in improving production efficiency if a balanced factory load can be provided. Inspired by these, in the paper, a factory assignment rule based iterative scheme is designed to allocate jobs to the factories, while scheduling decisions are learned by a complex DRL method. By doing so, an important compromise between algorithmic performance and computation time can be achieved.

The total processing time of jobs in each factory is an important indicator to measure the factory load [32]. Therefore, for the proposed Factory Assignment Rule, all jobs are sorted by the total processing times of each job in ascending order at first. Then the first f jobs are selected and allocated into each factory separately. Last, the remaining jobs are sequentially put into the factory with the smallest total processing time. The pseudo code of Factory Assignment Rule is presented in Algorithm 1.

Algorithm 1. Factory Assignment Rule.

1. **Procedure** Factory Assignment Rule
2. $\Pi \leftarrow \{\}$
3. $\tau \leftarrow$ Create a job sequence $(\tau_1, \tau_2, \ldots, \tau_n)$
4. **for** $j := 1$ **to** f **do**
5. Append job τ_j from τ to Π_j
6. **endfor**
7. **for** $j := f + 1$ **to** n **do**
8. **for** $l := 1$ **to** f **do**
9. $c_l \leftarrow$Calculate the total processing time of factory F_l
10. **endfor**
11. $l^* \leftarrow \arg(min_{l=1}^{f} c_l^*)$
12. Assign job τ_j from τ to the end of Π_{l^*}
13. **endfor**
14. **Output** Π

For the second decision of DJSP, a PDR generation method based on RL and GNN is proposed. First, the MDP model of DJSP is formulated. Then, a GNN is employed to explore the interaction among the vertices and arcs of the given graph. Final, a RL method is applied to train the proposed neural network.

4.1. Markov decision process formulation

RL is a powerful technique to address the sequential decision problems which can be modeled as a MDP [53]. MDP is a tuple (S, A, P, γ, R) with five elements. S is the state set, A is the action set, P is the state transition probability, γ is discount factor and $\gamma \in [0, 1]$, R is a reward function. At any decision point t, the agent, *i.e.*, the decision-maker (in the paper, it refers to the Actor network proposed in Section 4.2.2), considers the current state $s_t \in S$ and chooses an action $a_t \in A$ according to the policy $\pi(a_t|s_t)$. With transition probability $p(s_{t+1}|s_t, a_t) \in P$, the agent turns into a new state s_{t+1} and gains an immediate reward $r_t \in R$. The mentioned process continues until the agent reaches a terminal state and then it restarts. The accumulative reward is calculated by

$$R_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k} \tag{1}$$

where γ is the discount factor, it is used to regulate how long the agent considers when making decisions. The agent tries to maximize the expectation of the accumulative reward. DRL is an upgraded version of RL, in which the policy $\pi(a_t|s_t)$ consists of a deep neural network with parameters θ, and the parameters are updated during training.

The shop scheduling problem is a kind of sequential decision-making problem [10,40] and DJSP is not an exception. In this paper, a RL method based on GNN is employed to automatically generate PDRs to solve it. As mentioned, DJSP can be presented as a disjunctive graph, and the scheduling of DJSP can be achieved by directing the disjunctive arcs. Based on the disjunctive graph representation, the MDP model of DJSP is established as follows.

4.1.1. State

At decision point t, The disjunctive graph $G(t) = (V, C \cup D(t))$ reflects the current solution. As the definition shown in Section 3.2, V denotes vertices set whose elements are corresponding to job operations, C is the conjunction set, wherein the precedence among operations of one job is constrained. The disjunction set D is changeable step by step during problem solving. Until the decision point t, the disjunctions $D(t)$ are divided into the directed ones and the undirected ones, and represented as a 0–1 matrix with mn rows and mn columns, where mn is the total number of operations. Take an instance with 2 jobs, each of which contains 3 operations, as example, the initial disjunctions are represented as

$$D(0) = \begin{bmatrix} \mathbf{1} & 0 & 0 & 0 & 0 & 0 \\ 0 & \mathbf{1} & 0 & 0 & 0 & 0 \\ 0 & 0 & \mathbf{1} & 0 & 0 & 0 \\ 0 & 0 & 0 & \mathbf{1} & 0 & 0 \\ 0 & 0 & 0 & 0 & \mathbf{1} & 0 \\ 0 & 0 & 0 & 0 & 0 & \mathbf{1} \end{bmatrix} \tag{2}$$

where the diagonal elements are initialized to 1, they represent the connection of itself. For any vertex v in set V, five features are recorded, it is assumed that vertex v is corresponding to operation O_{ji}.

(1). The processing time of operation O_{ji}, *i.e.*, p_{ji}.

(2). A binary variable $b(O_{ji}, s_t)$, it is valued as 1 once operation O_{ji} is scheduled at point t, otherwise it keeps 0.

$$b(O_{ji}, s_t) = \begin{cases} 1, & \textit{if operation } O_{ji} \textit{ is scheduled} \\ 0, & \textit{otherwise} \end{cases} \tag{3}$$

(3). The factory that operation O_{ji} is assigned, $fac(O_{ji}, s_t)$. If job J_j is allocated into factory l, all operations belonging to job J_j are all featured by l, *i.e.*,

$$fac(O_{ji}, s_t) = l, \forall i \in [1, m], l \in [1, f] \tag{4}$$

(4). The estimated completion time of operation O_{ji}, $c_{LB}(O_{ji}, s_t)$. At decision point t, if operation O_{ji} is scheduled, $c_{LB}(O_{ji}, s_t)$ equals to its completion time $c(O_{ji}, s_t)$, otherwise, for any unscheduled operation O_{ji}, the value equals to the sum of the estimation completion time of operation $O_{j,i-1}$ and its own processing time p_{ji}, where $O_{j,i-1}$ is the operation before O_{ji}.

$$c_{LB}(O_{ji}, s_t) = \begin{cases} c(O_{ji}, s_t), & \textit{if operation } O_{ji} \textit{ is scheduled} \\ c_{LB}(O_{j,i-1}, s_t) + p_{ji}, & \textit{otherwise} \end{cases} \tag{5}$$

If $i = 1$, operation O_{ji} is the first one operation of job J_j, and the estimated completion time equals to the release time of job J_j plus the processing time of O_{ji}.

(5). The earliest start time of operation O_{ji}, $est(O_{ji}, s_t)$. At state s_0, the earliest start times of the first operation of each job are set as 0, and these of the remaining operations is estimated as

$$est(O_{ji}, s_0) = est(O_{j,i-1}, s_0) + p_{j,i-1} (j \in [1, n], i \in [2, m]) \tag{6}$$

At any state s_t ($t \neq 0$), the earliest start times of the scheduled operations are set as 0. For one unscheduled operation, if its previous operation $O_{j,i-1}$ has been done, the earliest start time equals to the completion time of operation $O_{j,i-1}$; For the unscheduled operation O_{ji} whose previous operation $O_{j,i-1}$ is not done, its earliest start time are estimated as the sum of the earliest start time and the processing time of operation $O_{j,i-1}$.

$$est(O_{ji}, s_t) = \begin{cases} 0, & \textit{if operation } O_{ji} \textit{ is done} \\ c(O_{j,i-1}, s_t), & \textit{if operation } O_{j,i-1} \textit{ is done, and operation } O_{ji} \textit{ is not done} \\ est(O_{j,i-1}, s_t) + p_{j,i-1}, & \textit{if both operations } O_{j,i-1} \textit{ and } O_{ji} \textit{ are not done} \end{cases} \tag{7}$$

Based on the mentioned features, five $[n, m]$ matrixes are obtained. Then these matrixes are all reshaped as $[n \times m, 1]$, and concatenated as a matrix $fea = [n \times m, 5]$. In order to eliminate the dimensional influence among the features and make the comparative evaluation fair, the raw features are normalized by the normalization formula

$$\overline{fea_i} = \frac{fea_i - \min(fea_i)}{\max(fea_i) - \min(fea_i)}, i = 1, 2, \ldots, 5 \tag{8}$$

4.1.2. Action

DJSP have $n \times m$ operations. The RL agent schedules one operation at a decision point. According to the definition of DJSP, at any time, each job can process only one operation. At state s_t, the action space A_t is composed of the successive operations of each job. The size of action space is not more than the number of jobs. As more and more jobs are completed, the action space becomes smaller and smaller. An action $a_t \in A_t$ is selected at decision point t. For the example shown in Section 3, the operations in action space are circled with red dot line. In Fig. 2(a), operations O_{11}, O_{21} and O_{31} have been scheduled, and the action space is $\{O_{12}, O_{22}, O_{32}, O_{41}, O_{51}, O_{61}\}$. Assume that operation O_{22} is selected at current step, then the action space turns into $\{O_{12}, O_{23}, O_{32}, O_{41}, O_{51}, O_{61}\}$ (see Fig. 2(b)). If operation O_{23} is scheduled at next step, the action

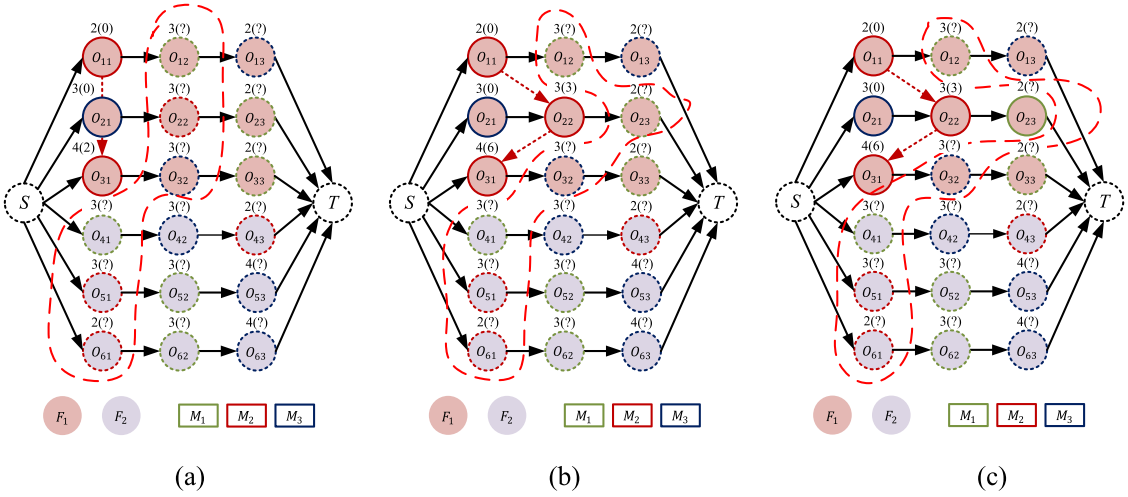


Fig. 2. Process diagram of action space.

space should be $\{O_{12}, O_{32}, O_{41}, O_{51}, O_{61}\}$ (see Fig. 2(c)), and the size of action space turns 6–5 with the completion of Job J_2.

4.1.3. State transition

Once an action has been taken, *i.e.*, an operation has been determined to dispatch, the most suitable start processing time on the required machine should be found out to make the completion time of the current operation minimized. Then the directions of the disjunctive graph should be updated. An example is shown in Fig. 3. The operations O_{11}, O_{21} and O_{31} that are marked with solid line have been scheduled, and the unscheduled operations are marked with dotted line. At state s_3 as shown in Fig. 3(a), action $a_3 = O_{22}$ (marked in orange) is selected from action space $\{O_{12}, O_{22}, O_{32}, O_{41}, O_{51}, O_{61}\}$. The earliest start processing time of O_{22} is 3 that is the completion time of O_{21}. On machine M_2, if O_{22} is allocated before O_{31}, the start processing time of O_{22} is minimized, and equals to 3. Therefore, at state s_4 as shown in Fig. 3(b), the directions of disjunctive arcs can be updated as $O_{11}\rightarrow O_{22}\rightarrow O_{31}$, the start processing time of O_{22} is 3, and that of O_{31} changes from 2 to 6.

4.1.4. Reward

The reward is a feedback signal given by environment, and it shows how well the agent performs a certain strategy at a certain step. The goal of the agent is to perform actions that maximize the accumulative reward in the long run. For DJSP, the completion times of each factory are all important, any factory with a poor schedule can affect the makespan. To dispatch operation one by one in the DJSP environment with the objective of minimizing makespan, a reward function based on the completion time of factory is designed. C_l is the completion time of factory l which equals to the maximum completion time of the operations in the factory, *i.e.*,

$$C_l = \max\left(c_{LB}\left(O_{ji}\right) | fac\left(O_{ji}\right) = l\right), l \in [1, f] \tag{9}$$

The makespan $C_{\max}$ equals to the maximum value among the completion times of factories, *i.e.*,

$$C_{\max} = \max(C_l) \tag{10}$$

To minimize the $C_{\max}$, the completion time of each factory C_l, $l \in [1, f]$ should be minimized as much as possible. Thus, the reward function is designed as

$$r(s_t, a_t) = C_l(s_t) - C_l(s_{t+1}) \tag{11}$$

where factory l is the factory that action a_t is in at state s_t. With the reward function, to maximize the reward, the completion time of factory l at state s_{t+1} should be minimized as small as possible. In the paper, the discount factor γ is set as 1. In a long term, the completion times of all factories can be optimal, so that the $C_{\max}$ can be minimized. Therefore, maximizing the accumulative reward equals to minimizing $C_{\max}$.

4.2. GNN-based policy

Graphs are a way to encode data in which the properties among the data can be clearly represented and mined. GNNs extend the existing neural network methods for processing the data represented in graph domains [11]. Several key terms for graphs are discussed to show how GNNs are formulated and work. ***Vertices*** are objects that contains some quantifiable attributes. In the proposed stitched disjunctive graph, vertices correspond to the job operations, by which the operation processing times, operated machines, the allocated factories and whether been done are presented. ***Edges*** characterize the relationships among objects, in the proposed problem, they indicate the precedence among operations of one job and the sequence of the operations that are executed on the same machine. ***Neighborhoods*** are subgraphs within a graph, and is usually a set of interrelated vertices or edges. ***Features***, in graph domains, are quantifiable attributes characterized by vertices and (or) edges, and they are represented as high-dimensional vectors in practice. ***Embeddings*** are compressed feature representations, which can turn large feature vectors into low dimensional data.

The traditional PDRs generally select jobs or operations based on the known information such as the processing times (SPT; Longest Processing Time, LPT), the number of the job operation (Least Operation Remaining, LOR) or the calculated values of the known information (Average Processing Time per Operation, AVPRO), they are treated as a policy that selects an action with probability 1 at a step. In this paper, a policy $\pi_\theta(a_t|s_t)$ based on GNN is proposed. The mentioned MDP formulation concludes the important static and dynamic information of DJSP such as the process information of each operation, the processing order on each machine and the relationship of the disjunctive arcs. By embedding the state features in the disjunctive graph, an effective PDR can be explored. In Fig. 4, The process of selecting one action is presented. First, the state features and the disjunction structure are set as the input; Second, the features are compressed by the GNN, *i.e.*, graph embedding is applied; Then, based on the graph embeddings, the information aggregation is performed by using an average pooling function; Last, the Actor network takes the embedding information as input and select one feasible action based on probability distribution. The details are presented as follows.

4.2.1. Graph embedding

Graph Isomorphism Network (GIN) [13] is a simple and effective

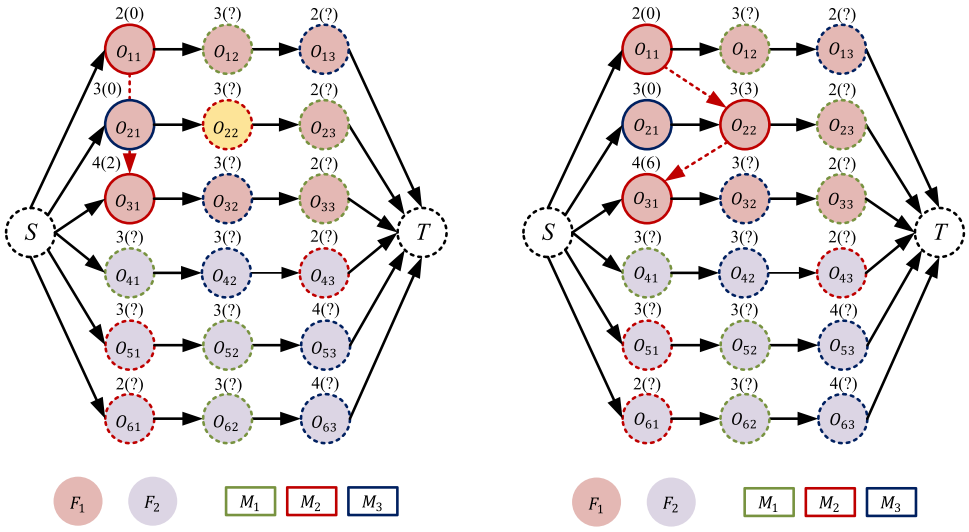


Fig. 3. The disjunctive graph of DJSP example.

GNN to extract feature embedding of vertices in a iterative and non-linear way, and it has been used to solve the scheduling problem successfully [8]. Inspired by the successful application, it is also used in the paper. For a given graph $G = (V, C \cup D)$, GIN computes an embedding for all vertices in set V by iterating K updates. GIN updates vertex representation as

$$h_v^k = MLP_{\theta_k}^k\left(\left(1+\epsilon^k\right) \bullet h_v^{k-1} + \sum_{u \in \mathbf{N}(v)} h_u^{k-1}\right) \tag{12}$$

where h_v^k is the compressed information of vertex v at iteration k, and h_v^0 is the raw input features of vertex v. $MLP_{\theta_k}^k$ is a Multi-Layer Perceptron (MLP) with parameters θ_k at iteration k. ϵ^k is a learnable parameter. $N(v)$ is the neighborhood of vertex v.

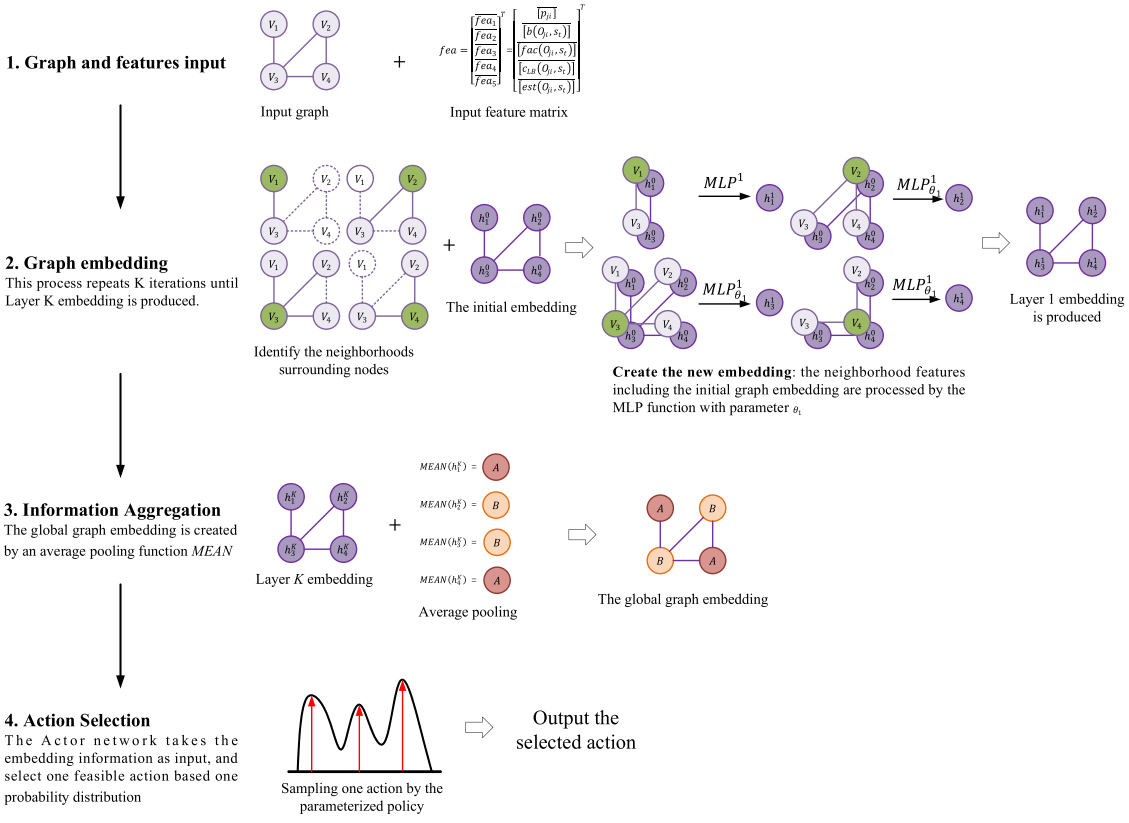


Fig. 4. Overall framework of how to select an action.

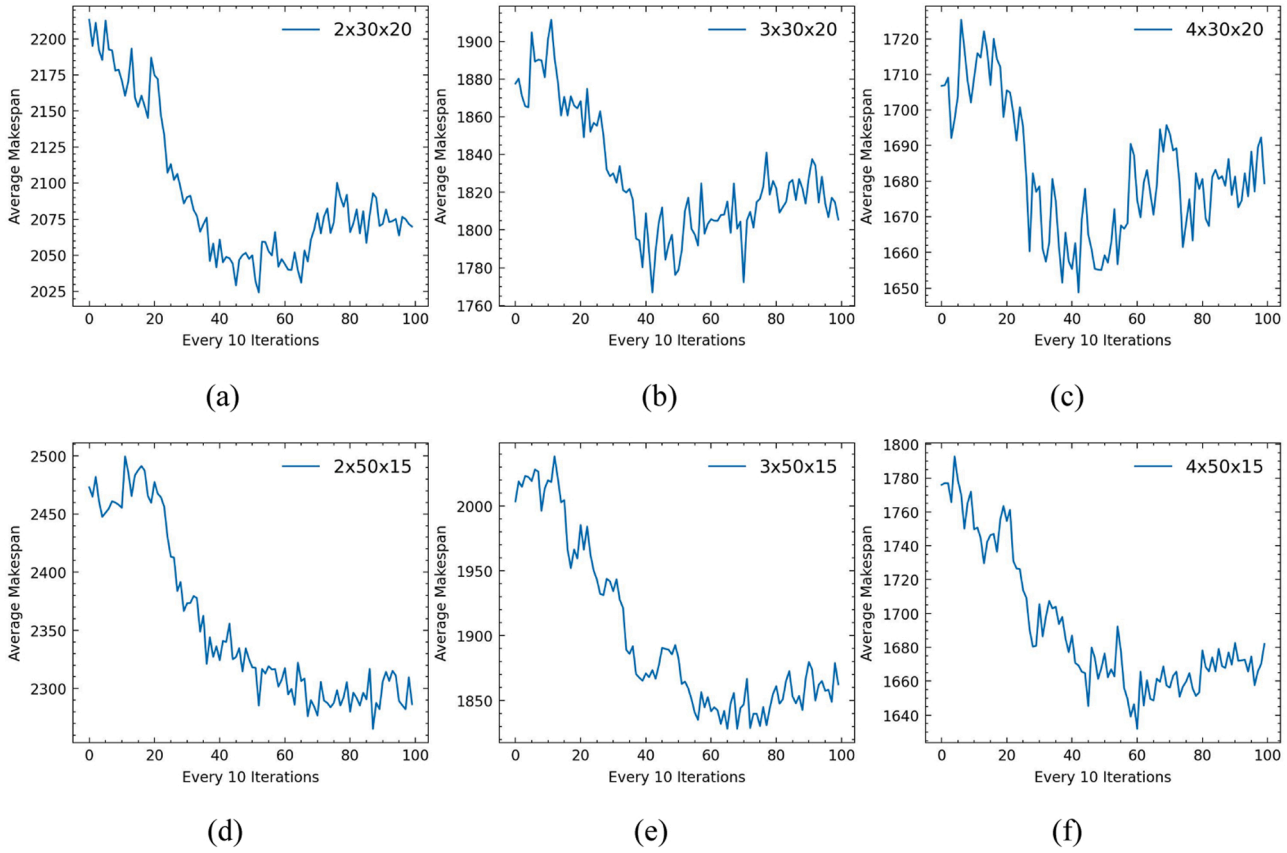


Fig. 5. The training curves of some instances.

After K iterations, the averages of the embeddings h_v^K $(v \in V)$ of all vertices are calculated by an average pooling function

$$MEAN\left(h_v^K, v \in V\right) = \frac{\sum_{v \in V} h_v^K}{|V|} \tag{13}$$

where $|V|$ is the number of vertices. At state s_t, for operation O_{ji} $(O_{ji} \in V)$, the raw features $h_{O_{ji}}^0(s_t)$ are represented as a 5-dimension vector $\left[\overline{p_{ji}}, \overline{b(O_{ji}, s_t)}, \overline{fac(O_{ji}, s_t)}, \overline{c_{LB}(O_{ji}, s_t)}, \overline{est(O_{ji}, s_t)}\right]$. After K iterations, the features of operation O_{ji} are represented as $h_{O_{ji}}^K(s_t)$. With the average pooling function (13), the global graph embedding $h_g(s_t)$ is calculated, and it is also a 5-dimension vector. For a DJSP with n jobs and m operations of each job, the value of $|V|$ is $m \times n$.

4.2.2. Action selection

After obtaining the vertex embeddings and the global embedding, a feasible action should be selected to make the schedule as optimal as possible. As mentioned above, each action in the state space corresponds to an unprocessed operation, and state s_t includes all information needed by current schedule. Here, an Actor network is applied to calculate the probability distribution over the feasible actions. To take full advantage of the local and the global information, for each action $a_t \in A_t$, the vertex embeddings and the global embedding are concatenated as $\left[h_{a_t}^K(s_t), h_g(s_t)\right]$. The Actor is a MLP with two layers which takes the concatenation as input, and scores for action a_t. Then a probability distribution $p(a_t)$ over the scores is calculated by a *softmax* function

$$p(a_t) = \frac{\exp\left(\left[h_{a_t}^K(s_t), h_g(s_t)\right]\right)}{\sum_{v \in A_t} \exp\left(\left[h_v^K(s_t), h_g(s_t)\right]\right)} \tag{14}$$

4.2.3. Reinforcement learning method

The Actor-Critic algorithm is an effective RL method which combines policy gradient and temporal difference learning. Actor network is a policy function $\pi_\theta(a|s)$ which learns a policy to get a reward as high as possible. Critic network refers to value function $V^\pi(s)$, and it estimates the current policy function, *i.e.*, evaluates how good the Actor network is. Based on the value function, the Actor-Critic method updates the parameters at one step, instead of waiting until an epoch is over. Furthermore, Asynchronous Advantage Actor-Critic (A3C) is the most famous variant, and it is structured with ϖ independent workers. Each worker samples, and is trained independently. The trained workers update the parameters of the global network periodically. In addition, the Critics share an identical network with the Actors, which is MLP with two layers, and take the global embedding $h_g(s_t)$ as input to evaluate the accumulative rewards at state s_t.

Table 2
The training times of each instance size (unit: h).

Instance sizes	15×15	20×15	20×20	30×15	30×20	50×15
$f = 2$	5.09	6.42	5.74	6.51	6.64 (for 6a)	7.47 (for 6d)
$f = 3$	5.17	6.51	5.75	6.49	5.44 (for 6b)	7.58 (for 6e)
$f = 4$	5.17	6.55	5.78	5.22	5.47 (for 6c)	7.44 (for 6f)

Table 3
Results on *Taillard's Benchmarks* with $f = 2$.

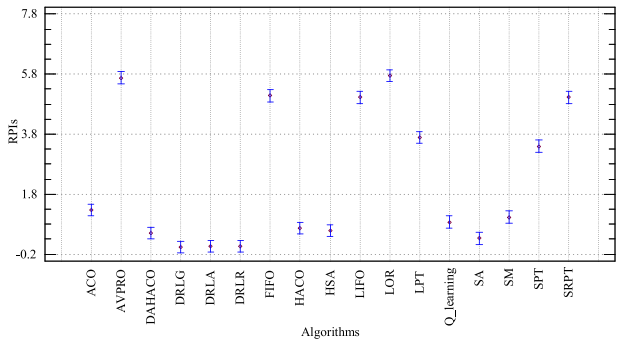
Algorithms	Instance sizes ($n \times m$)								
	15×15	20×15	20×20	30×15	30×20	50×15	***50 × 20***	***100 × 20***	means
FIFO	2.87	3.26	3.81	4.10	4.99	5.51	6.96	9.15	5.08
LIFO	2.70	3.29	3.76	4.15	4.91	5.38	6.97	9.03	5.02
SPT	1.90	2.24	2.62	2.80	3.39	3.59	4.62	6.04	3.40
LPT	2.02	2.45	2.79	3.14	3.70	3.86	5.00	6.54	3.69
SRPT	2.70	3.29	3.76	4.13	4.95	5.38	6.96	9.04	5.03
LOR	3.63	3.69	4.46	4.74	6.19	5.86	7.67	9.77	5.75
SM	0.56	0.61	0.80	0.78	0.88	1.17	1.51	1.98	1.04
AVPRO	3.37	3.87	4.52	4.89	5.84	5.99	7.15	9.80	5.68
SA	0.24	0.26	0.32	0.27	0.35	0.33	0.42	0.42	0.33
HSA	0.41	0.45	0.57	0.48	0.59	0.60	0.70	0.80	0.58
DAHACO	**0.02**	**0.05**	0.05	0.13	0.08	0.20	1.04	2.43	0.50
HACO	0.12	0.14	0.20	0.32	0.30	0.46	1.07	2.70	0.66
ACO	0.36	0.49	0.55	0.65	1.00	1.45	2.02	3.70	1.28
DRLG	0.08	0.07	**0.02**	**0.03**	**0.01**	**0.02**	0.09	0.04	**0.04**
DRLR	0.10	0.07	0.11	0.06	0.07	**0.02**	0.06	**0.00**	0.06
DRLA	0.07	0.10	0.08	0.09	0.07	0.07	**0.03**	0.06	0.07
Q-learning	0.58	0.87	0.77	0.84	0.84	1.01	1.02	1.04	0.87

Table 4
Results on *Taillard's Benchmarks* with $f = 3$.

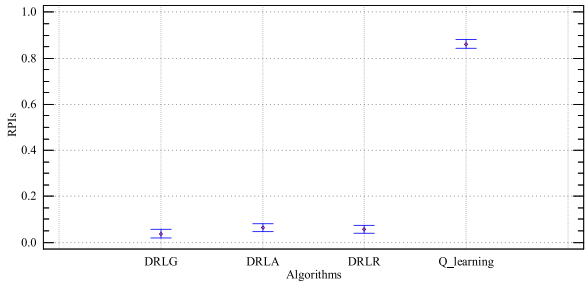
Algorithms	Instance sizes ($n \times m$)								
	15×15	20×15	20×20	30×15	30×20	50×15	***50 × 20***	***100 × 20***	means
FIFO	1.90	2.46	2.87	3.34	3.64	4.25	5.13	7.71	3.91
LIFO	1.88	2.45	2.83	3.27	3.63	4.32	5.23	7.61	3.90
SPT	1.25	1.72	1.89	2.24	2.54	2.69	3.43	5.14	2.61
LPT	1.57	1.86	2.11	2.49	2.93	3.09	3.77	5.63	2.93
SRPT	1.88	2.45	2.82	3.27	3.63	4.30	5.25	7.61	3.90
LOR	2.87	3.39	3.57	4.26	4.72	5.23	6.05	8.90	4.87
SM	0.37	0.58	0.57	0.66	0.65	0.80	1.10	1.79	0.81
AVPRO	2.47	3.50	3.89	4.24	4.53	5.21	6.08	8.92	4.86
SA	0.39	0.32	0.31	0.28	0.28	0.27	0.30	0.42	0.32
HSA	0.38	0.36	0.42	0.40	0.43	0.43	0.48	0.62	0.44
DAHACO	**0.00**	**0.04**	**0.03**	0.08	0.04	0.11	0.16	2.00	0.31
HACO	0.03	0.15	0.15	0.21	0.14	0.20	0.36	2.33	0.45
ACO	0.19	0.28	0.37	0.49	0.55	1.02	1.27	2.76	0.87
DRLG	0.10	0.06	0.04	**0.02**	**0.03**	**0.02**	**0.02**	0.06	**0.04**
DRLR	0.25	0.15	0.15	0.14	0.10	0.08	0.06	**0.01**	0.12
DRLA	0.13	0.08	0.07	0.11	0.07	0.09	**0.02**	0.09	0.08
Q-learning	0.88	0.89	0.85	1.05	1.03	1.16	1.13	1.40	1.05

Table 5
Results on *Taillard's Benchmarks* with $f = 4$.

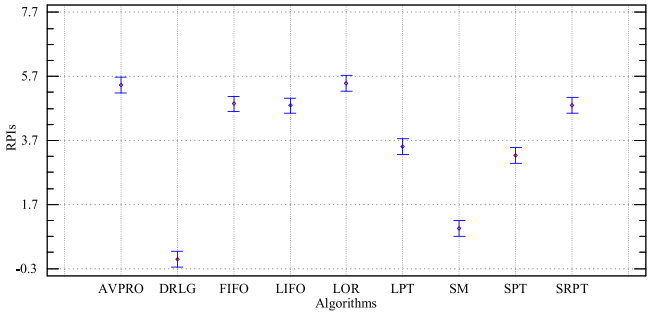
Algorithms	Instance sizes ($n \times m$)								
	15×15	20×15	20×20	30×15	30×20	50×15	***50 × 20***	***100 × 20***	means
FIFO	1.60	1.85	2.33	2.67	3.21	3.58	4.23	6.72	3.27
LIFO	1.49	1.82	2.30	2.56	3.13	3.53	4.26	6.64	3.22
SPT	1.08	1.40	1.62	1.80	2.09	2.43	3.00	4.47	2.24
LPT	1.23	1.53	1.81	1.97	2.49	2.63	3.21	4.87	2.47
SRPT	1.49	1.83	2.30	2.56	3.15	3.53	4.26	6.64	3.22
LOR	2.84	2.73	3.61	3.51	4.06	4.85	5.47	7.80	4.36
SM	0.36	0.39	0.52	0.49	0.59	0.72	0.87	1.48	0.68
AVPRO	2.36	2.94	3.42	3.73	4.37	4.44	5.28	7.91	4.31
SA	0.46	0.38	0.46	0.26	0.33	0.23	0.27	0.39	0.35
HSA	0.51	0.37	0.43	0.30	0.40	0.35	0.42	0.53	0.41
DAHACO	**0.00**	**0.01**	0.04	**0.02**	0.06	0.12	0.11	1.67	0.25
HACO	0.05	0.08	0.13	0.08	0.14	0.21	0.12	1.80	0.32
ACO	0.12	0.16	0.31	0.31	0.45	0.75	0.98	2.15	0.65
DRLG	0.14	0.08	0.12	0.05	0.07	0.04	0.04	0.05	0.07
DRLR	0.31	0.21	0.23	0.12	0.12	0.04	0.07	0.04	0.14
DRLA	0.17	0.16	**0.00**	0.07	**0.01**	**0.02**	**0.03**	**0.01**	**0.06**
Q-learning	0.95	0.93	1.02	1.12	1.24	1.37	1.31	1.66	1.20



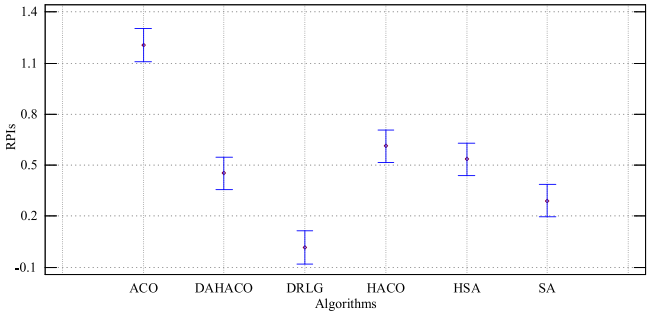
(a) Means Plots among All Compared Algorithms.



(b) Means Plots among DRLG and RL-based Methods.



(c) Means Plots among DRLG and PDRs.



(d) Means Plots among DRLG and Metaheuristics.

Fig. 6. Means plots with 95 % LSD Intervals with $f = 2$.

5. Experimental results and discussions

This session makes a detailed analysis of the method by sufficient experiments. Subsection 5.1 shows the model training details, including the parameter setting, training method, algorithm running environment and training results. Subsection 5.2 presents the results on *Taillard's Benchmarks*, and the number of factories is 2, 3 and 4. Subsection 5.3 analyzes the stability of the models. Subsection 5.4 designs an experiment to test the computational cost of the trained model. Finally, Subsection 5.5 makes a summary of the mentioned experimental analysis.

5.1. Training details

In this part, the hyper-parameters of the training method are presented, all of them are obtained by trial and error. The feature extraction network GIN is with three layers (including the input layer) of MLP. The number of the dimension of raw vertex features is five, therefore, the input dimension of the GIN is five. The hidden dimension of MLPs in the feature extraction network is 64. In addition, for the MLPs used in the Actor-Critic network is with two layers and the hidden dimension is 32. For the A3C, the number of workers is $\varpi = 4$, which means 4 independent environments are developed, the clipping parameter $\epsilon = 0.2$, the coefficients for value function, policy loss and entropy are 1.0, 2.0 and 0.01, respectively. The Adam optimizer is applied in model training, and *StepLR* which is the application programming interface provided by *Pytorch* [54] is used to adjust the learning rate. The initial learning rate is set as 2×10^{-5}. For *StepLR*, the learning rate decay interval *step_size* $=$ 100 and the multiplier for learning rate decay *gamma* $= 0.9$. Attentionally, other parameters of the model are all default as the set in *Pytorch*.

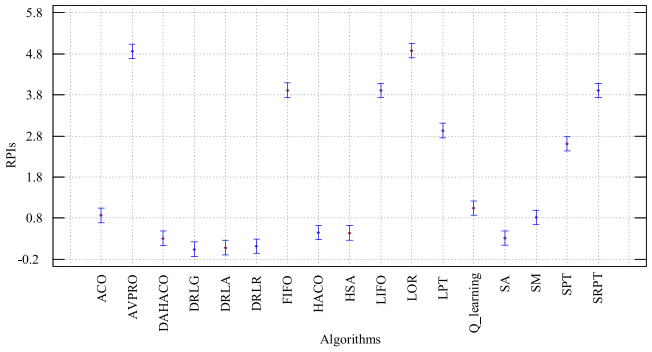
The well-known dataset *Taillard's Benchmarks* is used to test the algorithms designed for DJSP [31]. *Taillard's Benchmarks* have 80 instances, and these instances can be classified into 8 groups $\{15 \times 15, 20 \times 15, 20 \times 20, 30 \times 15, 30 \times 20, 50 \times 15, 50 \times 20, 100 \times 20\}$, and each group is with 10 instances. For DJSP, the number of factories is set as $\{2, 3, 4\}$. With the different number of factories, there are 240 instances for DJSP, and they can be classified into 24 instance groups, $\{2, 3, 4\} \times \{15 \times 15, 20 \times 15, 20 \times 20, 30 \times 15, 30 \times 20, 50 \times 15, 50 \times 20, 100 \times 20\}$. The processing time of each operation is between 1 and 99.

To train a model for solving a DJSP with a certain size, ϖ DJSP instances are generated randomly whose size is the same as that of the solved problem. For each instance size, 1000 iterations are run for training, and the policy model is updated at each iteration. For every 10 iterations, the model is validated on 50 instances that are generated ahead and fixed during training. All experiments are run on a machine with Intel(R) Core (TM) i7–6850 K CPU @ 3.60 GHz and Nvidia TITAN Xp GPU.

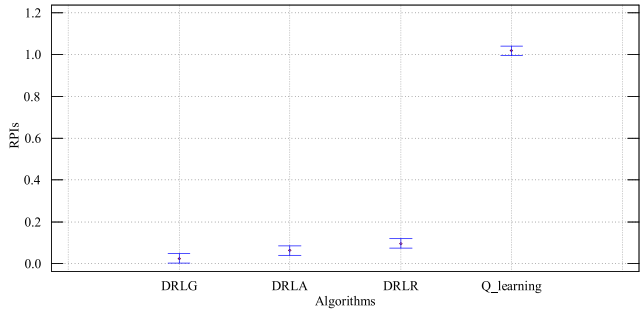
For the 24 instance groups, first, the policy for the instance groups whose sizes are not more than 50×15 is trained in the way mentioned above, and 18 models with different network parameters can be obtained. Then, the remaining instance groups $\{2, 3, 4\} \times \{50 \times 20, 100 \times 20\}$ are used to verify the generalizability of the trained models. Due to space limitation, only 6 training curves are shown in Fig. 5 where the horizontal axis denotes iteration times, and the vertical axis denotes the average makespan of the 50 fixed validation instances. The training times for each instance size are listed in Table 2, for the instances whose number of operations and factories is identical, the training time coincides with the number of jobs.

5.2. Results on Taillard's Benchmarks

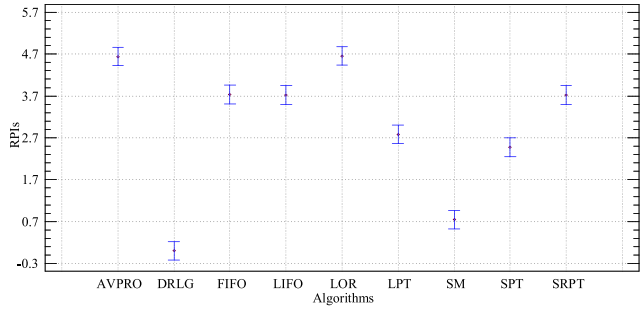
In the experiment, for the first 18 instance groups $\{2, 3, 4\} \times \{15 \times 15, 20 \times 15, 20 \times 20, 30 \times 15, 30 \times 20, 50 \times 15\}$, the trained models are used to address the corresponding instances. For example, the model trained by instance $2 \times 15 \times 15$ is applied to solve the instance groups $2 \times 15 \times 15$, and the model trained by instance $2 \times 20 \times 15$ is applied to solve the instance groups $2 \times 20 \times 15$, and so on. Without special statement, all trained models are collectively referred as DRLG in follows. For the remaining instance groups $\{2, 3, 4\} \times \{50 \times 20, 100 \times 20\}$, the 18 trained models are applied to solve them



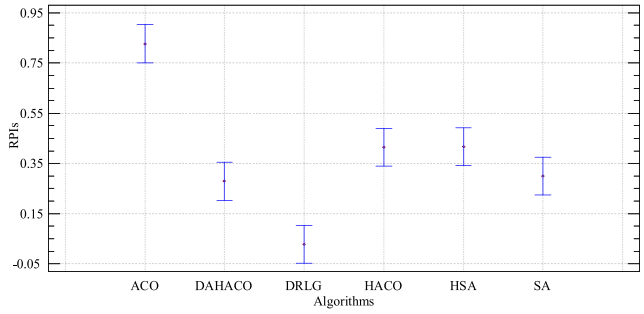
(a) Means Plots among All Compared Algorithms.



(b) Means Plots among DRLG and RL-based Methods.



(c) Means Plots among DRLG and PDRs.



(d) Means Plots among DRLG and Metaheuristics.

Fig. 7. Means plots with 95 % LSD intervals with $f = 3$.

respectively, and the experimental results are the averages of the results calculated by the 18 trained models. In Tables 3, 4 and 5, these results are marked with an underline.

To demonstrate the performance of the proposed DRLG, the trained models are compared with 8 classical PDRs [21], 5 metaheuristics and 3 RL-based methods. The compared PDRs are modified to solve DJSP, where the job assignment to each factory is initialized with the same method mentioned in Section 4, and then the jobs in each factory are scheduled with the following PDRs: FIFO, Last In First Out (LIFO), SPT, LPT, Shortest Remaining Processing Time (SRPT), LOR, Shortest Processing Time per Working Remaining (SPT/MWKR, SM) and AVPRO. 5 metaheuristics include SA [28], Hybrid SA (HSA) [28], ACO [31], Hybrid ACO (HACO) [31] and Dynamic Assignment method of jobs to factories with a HACO (DAHACO) [31]. Due to the lack of DRL methods for DJSP, here 3 RL-based methods for the related problems are adapted to solve DJSP, and they are a DRL method (DRLR) [8], a Dueling Double DQN (DRLA) [55], and a Q-learning method [56]. Specially, DRLR is based on disjunctive graph and GNN for JSP, DRLA is proposed for adaptive JSP which is based on disjunctive graph dispatching and deep convolutional neural network, both of them are based the graph structures, while Q_learning is a basic RL method for FSP. As the compared PDRs do, when comparing the RL methods, the job assignment to each factory is initialized with the same method mentioned in Section 4. In the original reference, SA and HSA are not used to address *Taillard's Benchmarks*, therefore they are recoded and are used to solve the benchmarks. The running time is set as $\frac{30\times n\times m\times f}{1000}$ seconds, and the parameters are the same as those provided in the original paper.

To have a clear identification of the experimental results, the relative percentage increase (RPI) is used as the response variable, and it is calculated as

$$RPI = \frac{Method_{sol} - Best_{sol}}{Best_{sol}} \times 100\% \tag{15}$$

where $Method_{sol}$ is the result of a certain instance obtained by some algorithm, and $Best_{sol}$ is the best result of the instance among all algorithms.

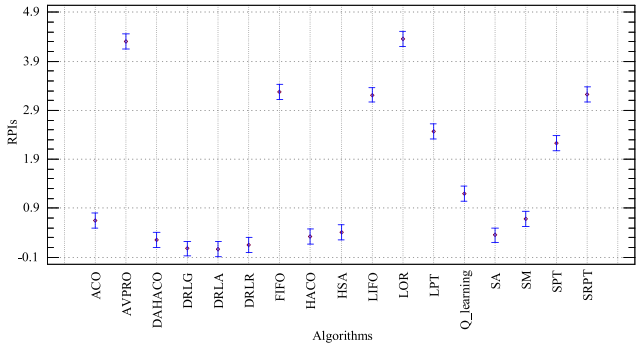
5.2.1. Results with $f = 2$

Table 3 shows the average RPIs of experimental results on *Taillard's Benchmarks* with 2 factories, and the best results are marked in bold. When the instance size is 15×15 and 20×15, DAHACO performs best with the smallest average RPI values 0.02 and 0.05. The proposed DRLG also shows pretty good performance with RPI values 0.08 and 0.07. However, the good performance of DAHACO comes at the time cost with 721 s and 1893 s respectively [31], while the running times of DRLG are fairly short with 0.363 s and 0.537 s respectively. When the instance sizes are 20×20, 30×15, 30×20 and 50×15, the smallest average RPI values are all from DRLG. When the instance sizes are 50×20 and 100×20, it seems that DRLG is slightly worse than DRLR and DRLA. However, in terms of generalizability, DRLG is the best. For the instance groups $\{2\} \times \{50 \times 20, 100 \times 20\}$, no model with corresponding size is specifically trained for them, but the mentioned 18 trained models with different sizes are used to address them. The results of the instance groups $\{2\} \times \{50 \times 20, 100 \times 20\}$ are the averages obtained by these models. Therefore, DRLG shows best performance whatever the quality of solutions and the generalizability of the models.

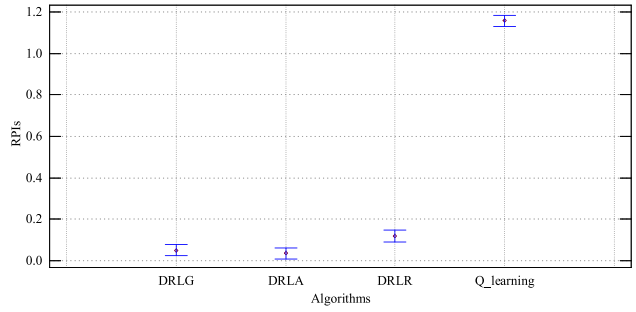
To have a clear identification of the results on *Taillard's Benchmarks* with 2 factories, an ANOVA analysis is done. Fig. 6 shows the means plots with 95 % LSD intervals. From Fig. 6(a), it can be found that RL-based methods excluding Q-learning show pretty good performance, and the metaheuristics also produce good results. By contrast, apart from SM which is even slightly better than ACO, the performance of other PDRs is relatively mediocre. Fig. 6(b) separately compares the RL-based methods, and demonstrates that DRLG is with the best performance. Fig. 6(c) and Fig. 6(d) compares DRLG with PDRs and metaheuristics respectively, and both of them tell that DRLG presents an absolute advantage over them.

5.2.2. Results with $f = 3$

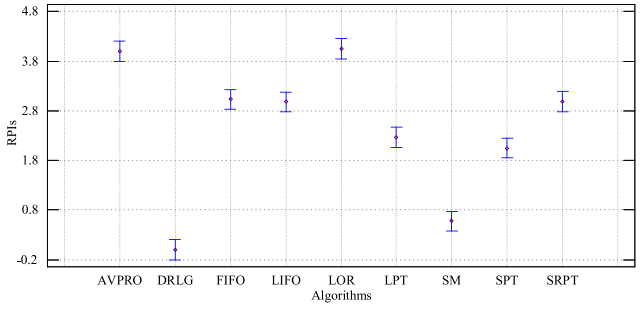
Table 4 shows the average RPIs of each instance size when the number of factories is 3 where the smallest average RPI value is marked



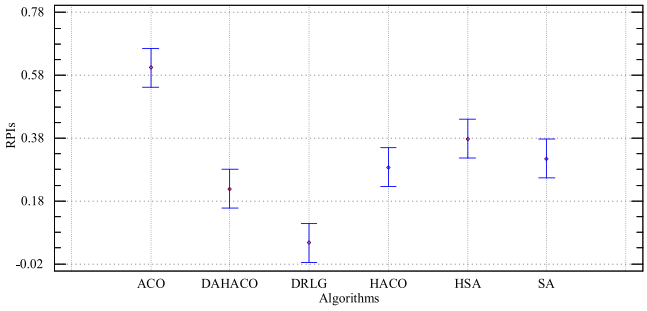
(a) Means Plots among All Compared Algorithms.



(b) Means Plots among DRLG and RL-based Methods.



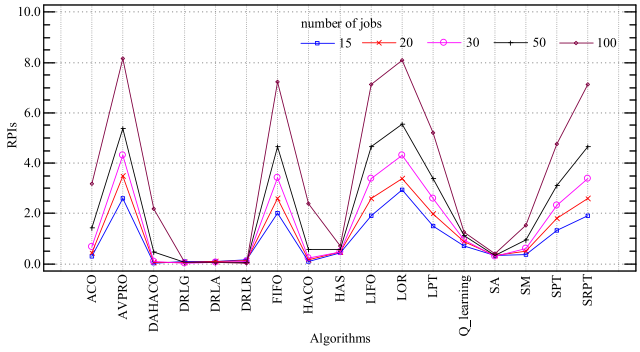
(c) Means Plots among DRLG and PDRs.



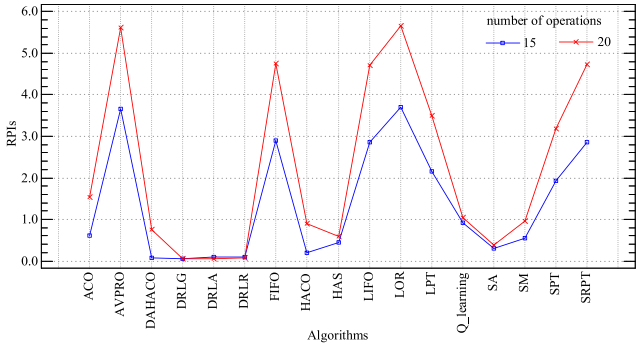
(d) Means Plots among DRLG and Metaheuristics.

Fig. 8. Means plots with 95 % LSD intervals with $f = 4$.

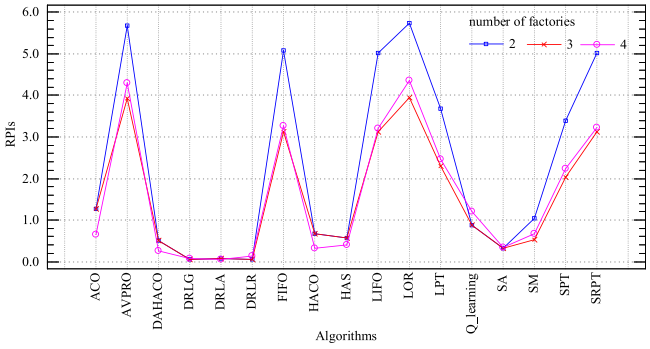
in bold. When the number of jobs is not more than 30, the best results are always obtained by the DAHACO. When the number of jobs is 15, the proposed DRLG is ranked third with average RPI value 0.10, and it occupies the second place when the instance sizes are 20×15 and 20×20. As the instance size grows, the advantages of the proposed DRLG are gradually emerging. When the sizes are 30×15, 30×20, 50×15, 50×20, the best average RPIs are all from it. DRLG (0.06) loses to DRLR



(a) Interaction Plots between Algorithms and Number of Jobs.



(b) Interaction Plots between Algorithms and Number of Operations.



(c) Interaction Plots between Algorithms and Number of Factories.

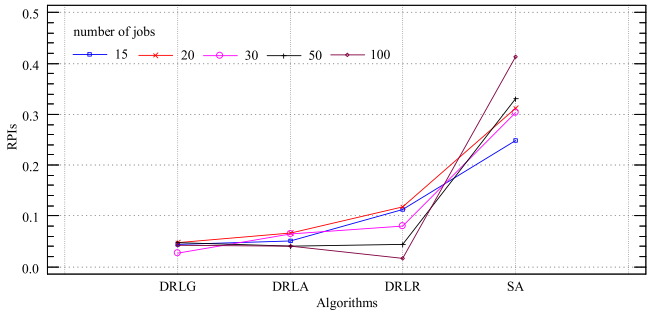
Fig. 9. Interaction plots with 95 % LSD intervals of the algorithm type and other factors.

(0.01) by a small margin when the instance size is 100×20. In terms of the overall mean, DRLG is still the best with the smallest average value of 0.04.

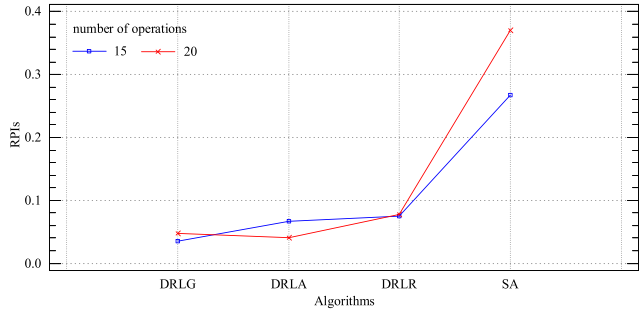
Here an ANOVA analysis is done based on the RPIs of all compared algorithms, and the means plots with 95 % LSD intervals are presented in Fig. 7. From Fig. 7, the conclusion can be obtained as that of Fig. 6. Fig. 7(a) tells a general conclusion that the best three algorithms are RL-based methods, DRLG, DRLR and DRLA. Then the metaheuristics such as SA, DAHACO, HSA and HACO also perform well. Among the PDRs, SM is with the best performance. Furthermore, Fig. 7(b), (c) and (d) show that when $f = 3$, DRLG is always the best algorithm when compared to the RL-based methods, PDRs or metaheuristics.

5.2.3. *Results with $f = 4$*

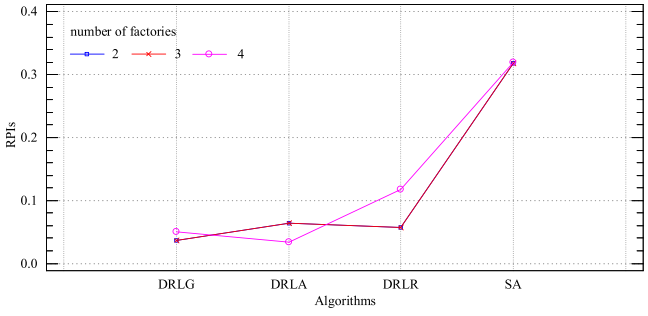
Table 5 presents the average RPI values of all results with 4 factories. When the instance size is 15×15, the smallest average RPI is produced by DAHACO, and the next two are from HACO and ACO, while DRLG just ranks forth. When the instance size is 20×15, DAHACO performs best, while both HACO and DRLG ranks second with average RPI 0.08.



(a) Interaction Plots between Algorithms and Number of Jobs.



(b) Interaction Plots between Algorithms and Number of Operations.



(c) Interaction Plots between Algorithms and Number of Factories.

Fig. 10. Interaction plots with 95 % LSD intervals of four best algorithms and other factors.

When the instance size is 20×20, the best result is from DRLA. Apart from DRLA, DRLG is with the average RPI value 0.12 which is after DAHACO with the average RPI 0.04. When the instance size is 30×15, DAHACO still shows the best performance with the smallest average RPI 0.02, while DRLG follows with the average RPI 0.05. When the instance size is 30×20, DRLA is with the smallest value 0.01, then DAHACO follows with the value of 0.06, DRLG is ranked third with the average 0.07. When the instance size is 50×15, DRLA produces the smallest average RPI 0.02, while DRLG and DRLR follow with the value 0.04. When the instance size is 50×20, the best average RPI is from DRLA with 0.03, while DRLG with the average value 0.04 which is slightly larger than that of DRLA. When the instance size is 100×20, the proposed DRLG ranks third with the average RPI 0.05. In terms of the average RPIs shown in Table 5, it seems that the performance of the DRLG is not optimal among the compared algorithms when the number of factories is 4, and it slightly worse than DRLA whose overall mean is 0.06, while that of DRLG is 0.07. However, except the RPIs, the generalizability should be taken into consideration. For DRLG, the results of instances with sizes of 50×20 and 100×20 are obtained by the models trained with instances with other sizes. Despite this, DRLG still shows good performance. It indicates that DRLG not only equips with high effectiveness but also good generalizability.

The compared results from ANOVA analysis based on the whole RPIs are shown in Fig. 8. From Fig. 8(a), it can be concluded that DRLG and DRLA are the two best algorithms among all compared algorithms, and then DRLR and DAHACO follow. HACO is slightly worse than DAHACO, and HSA is slightly worse than HACO and SA. ACO and SM show pretty close performance. Q-learning is worse than all above mentioned methods. Fig. 8(b) shows that among all RL-based methods, DRLG and DRLA are close, and both of them are better than other two methods. Fig. 8(c) indicates that DRLG outperforms all PDRs with significant margin. Fig. 8(d) tells that DRLG presents best performance among all the compared metaheuristics in general.

5.3. Stability analysis

An ANOVA analysis based on all obtained data is done where the factors are the type of compared algorithms, the number of jobs (n), the number of operations of each job (m) and the number of factories (f). Fig. 9 presents the interaction plots with 95 % LSD intervals of the algorithm types and other factors. From Fig. 9(a), it can be found that the stability of the most algorithms is affected significantly by the number of jobs, and the greater the number of jobs is, the worse the stability is. By contrast, DRLG, DRLA, DRLR and SA are less affected. Fig. 9(b) demonstrates that DRLG, DRLA, DRLR, HAS, Q-learning and SA are affected slightly by the number of operations. Fig. 9(c) shows that DRLG, DRLA, DRLR and SA are almost unaffected by the number of factories. Concluded from Fig. 9, DRLG, DRLA, DRLR and SA are the most stables. To have a detailed identification of the stability of the four algorithms, another comparison only among them is presented, and the interaction plots are shown in Fig. 10. It can be found that, among the four most stable algorithms, DRLG is least affected by the instance size, whatever the number of jobs, the number of operations of each job and the number of factories.

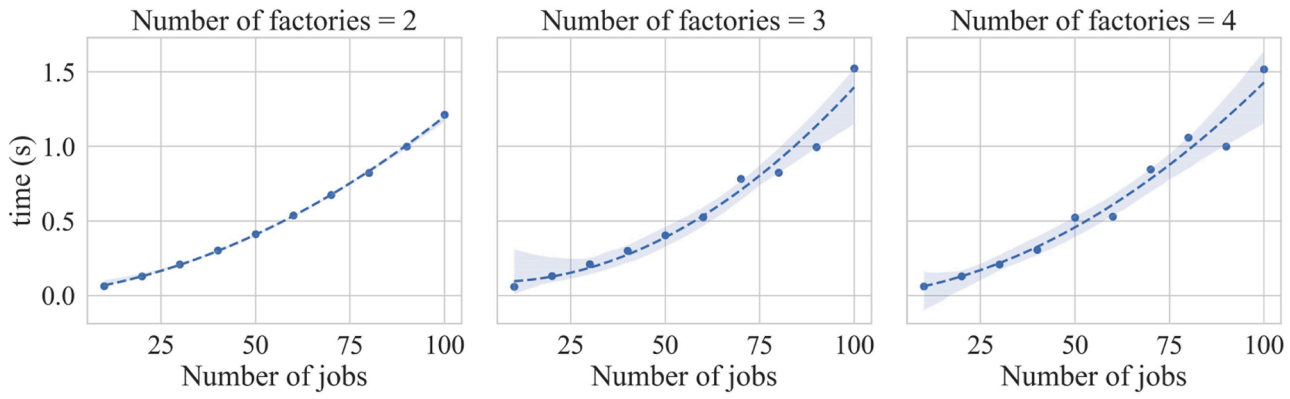
5.4. Computational cost

Computational cost is an important factor for an algorithm. To investigate the computational cost of the proposed method, three experiments are designed: (a) test how the computational time is affected by the number of jobs, and the test instance sizes are set as $\{2, 3, 4\} \times \{10, 20, 30, 40, 50, 60, 70, 80, 90, 100\} \times \{5\}$; (b) test how the computational time is affected by the number of operations, and the test instance sizes are set as $\{2, 3, 4\} \times \{50\} \times \{5, 10, 15, 20, 25, 30\}$; (c) test how the computational time is affected by the number of factories, and the test instance sizes are set as $\{2, 3, 4, 5, 6, 7\} \times \{50, 80, 100\} \times \{5\}$. Each size is with 10 instances whose processing times are generated between 1 and 99 randomly.

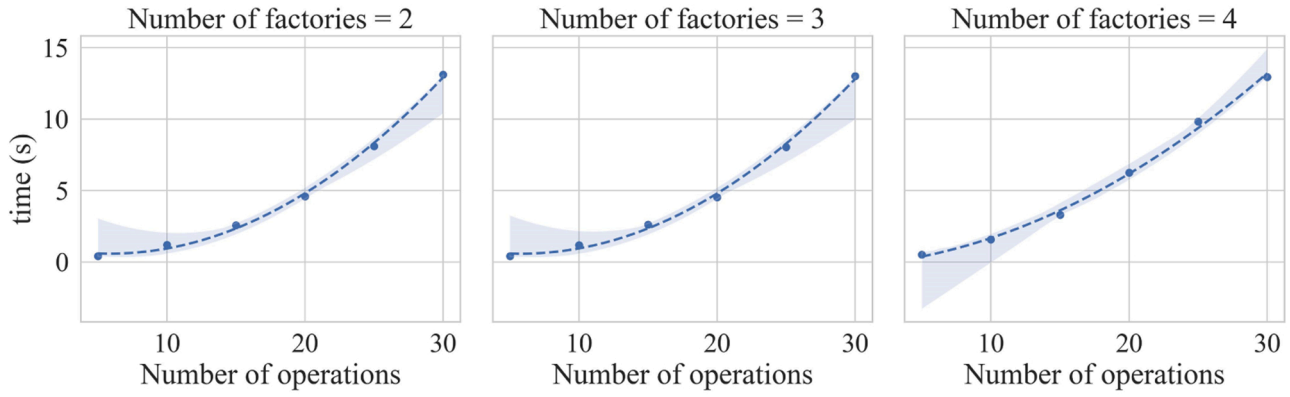
The average running times based on the instance sizes are recorded. Fig. 11 is obtained by polynomial fitting. Fig. 11(a) shows the second-order polynomial fitting between the number of jobs (n) and the running time, and it indicates that the running time is proportional to n^2. While Fig. 11(b) also shows the second-order polynomial fitting between the number of operations (m) and the running time, and obviously, the running time is also proportional to m^2. Fig. 11(c) is the first-order polynomial fitting between the number of factories (f) and the running time, it can be found that with the increase of factories, the running time is reduced, the trend is more pronounced when the number of jobs is large.

5.5. A summary of the experimental results

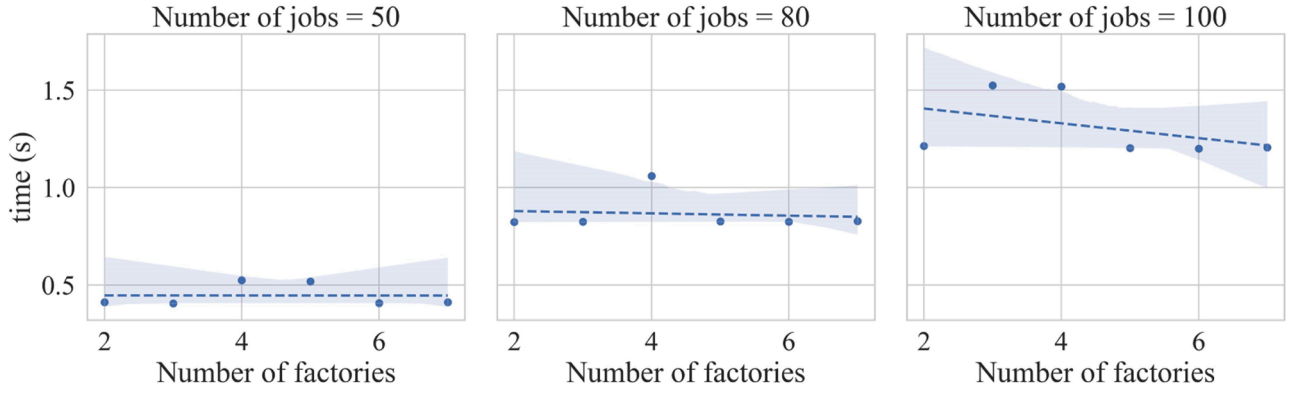
From the analyses mentioned above, the RL-based methods excluding Q-learning method show better ability to solving the problems whatever the instance sizes are, and then metaheuristics also perform pretty well. By comparison, the PDRs are a little less impressive. However, the SM deserves attention. When the number of factories is 2, it



(a) Computational Time Affected by the Number of Jobs.



(b) Computational Time Affected by the Number of Operations.



(c) Computational Time Affected by the Number of Factories.

Fig. 11. Computational cost analysis of DRLG.

even beats ACO, and is neck and neck with HACO and HSA, and is always better than Q-learning. Among the compared metaheuristics, DAHACO shows a good performance. From the presented results, it can be found that DAHACO performs better with the increase of number of factories. However, its performance seems to be deteriorating gradually as the scale of jobs and operations grew. Therefore, it can be concluded that DAHACO is better at addressing the problems that have fewer jobs in each factory. For the proposed DRLG, when the number of factories is 2 and 3, it shows absolute advantage whatever the quality of solutions and the generalizability of trained models. When the number of factories is 4, DRLG shows a similar performance with that of DRLA. When the instance sizes are 50×20 and 100×20, DRLG does not achieve the best average RPIs, which should have some impact on the overall performance evaluation of the algorithm, but it achieves a trade-off between effectiveness and generalizability. By an ANOVA analysis on all experimental data, the stability of DRLG is demonstrated. It is least affected by the instance size.

When the number of factories is 4, the proposed DRLG loses absolute advantage in terms of the average RPIs. For this matter, the following explanation is presented. First, in the proposed method, the factory

where the operations are assigned is a state feature that helps the method explore the connection among vertices of disjunctive graph. When the number of factories is larger, the features would be more scattered, which is not conductive to self-learning of the method. In addition, more factories mean greater productivity. For the instances with identical jobs, when the number of factories is larger, the number of jobs in each factory is smaller, the existing PDRs can also have a pretty good performance in this situation. For metaheuristics, the running time ($\frac{30\times n\times m\times f}{1000}$ seconds) increases as the number of factories increases, it is helpful for solution search of the algorithms. To some extent, the excellent results produced by metaheuristics benefit from the considerable running time. On these conditions, the power of the proposed method is weakened when the number of factories is 4.

6. Conclusion and future research

This paper proposes a PDR generation method based on GNN and RL for DJSP, with which PDRs can be obtained by training. The obtained PDR can schedule the jobs step by step by interacting with the observed environment. The comprehensive experiments have proved that it outperforms the three types of the mainstream methods in the scheduling field, including the PDRs, the metaheuristics and the RL-based methods. Except for the performance improvement, the major contributions of the paper are listed as follows.

(1) Solution representation. A stitched disjunctive graph representation is proposed, it connects the disjunctive graph of each factory together, and records the factory assignment information. The characterized design provides an important basis for the application of GNN in the method.
(2) Problem modeling. DJSP is a kind of sequential decision-making problem, and the specialized MDP is formulated. Five critical features reflecting the static and dynamic characteristics during problem solving are recorded at each decision point. An objective-oriented reward mechanism is shaped, whose value depends on the makespan increase of the current factory.
(3) Job priority. For the general PDRs for the related scheduling problems, the job priority is usually based on the job processing times (SPT, LPT), the number of the job operation (LOR) or the values calculated with the known information (AVPRO). The proposed method uses a trained GIN to extract the embedding features of the disjunctive graph, and then adopts a trained Actor network to score each feasible job. The priority among jobs is determined by the output scores. Through this way, a more comprehensive information can be observed, thus a wiser job selection can be achieved at each decision point.

Although this paper offers a new perspective on solving DJSP, there still are many points worth exploring. For example, the performance on the problem sizes with 4 factories is unremarkable. For this limitation, more connected features of DJSP should be explored, and more effective networks should be established. In addition, the practical production environment is always dynamic. The dynamic events such as the random job arrivals, the machine breakdowns and the uncertain processing time have a significant impact on the scheduling scheme. DRL can achieve a good performance on solving the dynamic events, and rapidly response to the changing environment. Based on these, the ability of DRL for addressing the distributed manufacturing with uncertain events should be explored.

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.

Acknowledgements

This work was supported in part by National Key R&D Program of China under Grant 2021YFB3301902, the National Science Foundation of China under Grants 51825502, U21B2029 and 52188102.

References

[1] Okwudire CE, Madhyastha HV. Distributed manufacturing for and by the masses. Science 2021;372(6540):341–2. https://www.science.org/doi/epdf/10.1126/science.abg4924.
[2] Li YZ, Pan QK, Li JQ, Gao L, Tasgetiren MF. An Adaptive Iterated Greedy algorithm for distributed mixed no-idle permutation flowshop scheduling problems. Swarm Evol Comput 2021;63:100874. https://doi.org/10.1016/j.swevo.2021.100874.
[3] Meng T, Pan QK. A distributed heterogeneous permutation flowshop scheduling problem with lot streaming and carryover sequence-dependent setup time. Swarm Evol Comput 2021;60:100804. https://doi.org/10.1016/j.swevo.2020.100804.
[4] Jia HZ, Nee AYC, Fuh JYH, Zhang YF. A modified genetic algorithm for distributed scheduling problems. J Intell Manuf 2003;14:351–62. https://doi.org/10.1023/A:1024653810491.
[5] Jia HZ, Fuh JYH, Nee AYC, Zhang YF. Integration of genetic algorithm and Gantt chart for job shop scheduling in distributed manufacturing systems. Comput Ind Eng 2007;53:313–20. https://doi.org/10.1016/j.cie.2007.06.024.
[6] Naderi B, Azab A. Modeling and heuristics for scheduling of distributed job shops. Expert Syst Appl 2014;41:7754–63. https://doi.org/10.1016/j.eswa.2014.06.023.
[7] Garey MR, Johnson DS, Sethi R. The complexity of flowshop and jobshop scheduling. Math OR 1976;1:117–29. https://doi.org/10.1287/moor.1.2.117.
[8] Zhang C, Song W, Cao Z, Zhang J, Tan PS, Chi X. Learning to dispatch for job shop scheduling via deep reinforcement learning. Adv Neural Inf Process Syst 2020;33:1621–32.
[9] Chen X, Tian Y. Learning to perform local rewriting for combinatorial optimization. Adv Neural Inf Process Syst 2019;32 (Curran Associates, Inc).
[10] Han BA, Yang JJ. A deep reinforcement learning based solution for flexible job shop scheduling problem. Int J Simul Model 2021;20:375–86. https://doi.org/10.2507/IJSIMM20-2-CO7.
[11] Scarselli F, Gori M, Tsoi AC, Hagenbuchner M, Monfardini G. The graph neural network model. IEEE Trans Neural Netw 2009;20:61–80. https://doi.org/10.1109/TNN.2008.2005605.
[12] Zhang Z., Cui P., Zhu W. Deep learning on graphs: a survey. ArXivOrg; 2018. Available from: https://arxiv.org/abs/1812.04202v3.
[13] Xu K., Hu W., Leskovec J., Jegelka S. How powerful are graph neural networks?; 2019. Available from: https://doi.org/10.48550/arXiv.1810.00826.
[14] Wang Z, Eisen M, Ribeiro A. Learning decentralized wireless resource allocations with graph neural networks. IEEE Trans Signal Process 2022;70:1850–63. https://doi.org/10.1109/TSP.2022.3163626.
[15] Kong Z, Jin X, Xu Z, Zhang B. Spatio-temporal fusion attention: a novel approach for remaining useful life prediction based on graph neural network. IEEE Trans Instrum Meas 2022;71:3515912. https://doi.org/10.1109/TIM.2022.3184352.
[16] Hossain RR, Huang Q, Huang R. Graph convolutional network-based topology embedded deep reinforcement learning for voltage stability control. IEEE Trans Power Syst 2021;36:4848–51. https://doi.org/10.1109/TPWRS.2021.3084469.
[17] Dabbas RM, Chen HN, Fowler JW, Shunk D. A combined dispatching criteria approach to scheduling semiconductor manufacturing systems. Comput Ind Eng 2001;39:307–24. https://doi.org/10.1016/S0360-8352(01)00008-0.
[18] Chen T. An effective dispatching rule for bi-objective job scheduling in a wafer fabrication factory—considering the average cycle time and the maximum lateness. Int J Adv Manuf Technol 2013;67:1281–95. https://doi.org/10.1007/s00170-012-4565-6.
[19] Koo PH, Jang J. Vehicle travel time models for AGV systems under various dispatching rules. Int J Flex Manuf Syst 2002;14:249–61. https://doi.org/10.1023/A:1015831711304.
[20] Jayamohan MS, Rajendran C. New dispatching rules for shop scheduling: a step forward. Null 2000;38:563–86. https://doi.org/10.1080/002075400189301.
[21] Sels V, Gheysen N, Vanhoucke M. A comparison of priority rules for the job shop scheduling problem under different flow time- and tardiness-related objective functions. Int J Prod Res 2012;50:4255–70. https://doi.org/10.1080/00207543.2011.611539.
[22] Nguyen TT, Nguyen ND, Nahavandi S. Deep reinforcement learning for multiagent systems: a review of challenges, solutions, and applications. IEEE Trans Cybern 2020;50:3826–39. https://doi.org/10.1109/TCYB.2020.2977374.
[23] Khalil E, Dai H, Zhang Y, Dilkina B, Song L. Learning combinatorial optimization algorithms over graphs. Adv Neural Inf Process Syst 2017;30 (Curran Associates, Inc).
[24] Wang J, Gao P, Zheng P, Zhang J, Ip WH. A fuzzy hierarchical reinforcement learning based scheduling method for semiconductor wafer manufacturing systems. J Manuf Syst 2021;61:239–48. https://doi.org/10.1016/j.jmsy.2021.08.008.
[25] Kim HJ, Lee JH. Look-ahead based reinforcement learning for robotic flow shop scheduling. J Manuf Syst 2023;68:160–75. https://doi.org/10.1016/j.jmsy.2023.02.002.
[26] Kong X, Duan G, Hou M, Shen G, Wang H, Yan X, et al. Deep reinforcement learning-based energy-efficient edge computing for internet of vehicles. IEEE Trans Ind Inf 2022;18:6308–16. https://doi.org/10.1109/TII.2022.3155162.

[27] Aydin ME, Öztemel E. Dynamic job-shop scheduling using reinforcement learning agents. Robot Auton Syst 2000;33:169–78. https://doi.org/10.1016/S0921-8890(00)00087-7.
[28] Naderi B, Azab A. An improved model and novel simulated annealing for distributed job shop problems. Int J Adv Manuf Technol 2015;81:693–703. https://doi.org/10.1007/s00170-015-7080-8.
[29] Hsu CY, Kao BR, Ho VL, Lai KR. Agent-based fuzzy constraint-directed negotiation mechanism for distributed job shop scheduling. Eng Appl Artif Intell 2016;53:140–54. https://doi.org/10.1016/j.engappai.2016.04.005.
[30] Chaouch I, Driss OB, Ghedira K. A modified ant colony optimization algorithm for the distributed job shop scheduling problem. Procedia Comput Sci 2017;112:296–305. https://doi.org/10.1016/j.procs.2017.08.267.
[31] Chaouch I, Driss OB, Ghedira K. A novel dynamic assignment rule for the distributed job shop scheduling problem using a hybrid ant-based algorithm. Appl Intell 2019;49:1903–24. https://doi.org/10.1007/s10489-018-1343-7.
[32] Jiang E, Wang L, Peng Z. Solving energy-efficient distributed job shop scheduling via multi-objective evolutionary algorithm with decomposition. Swarm Evolut Comput 2020;58:100745. https://doi.org/10.1016/j.swevo.2020.100745.
[33] Jackson J.R. Scheduling a production line to minimize maximum tardiness. Management science research projects; 1955.
[34] Naidu JT. A note on a well-known dispatching rule to minimize total tardiness. Omega 2003;31:137–40. https://doi.org/10.1016/S0305-0483(03)00020-3.
[35] Panwalkar SS, Iskander W. A survey of scheduling rules. Oper Res 1977;25:45–61. https://doi.org/10.1287/opre.25.1.45.
[36] Jones A, Rabel LC, Shalawi AT. Survey of job shop scheduling techniques. Wiley Encycl Electr Electron Eng 1999.
[37] Huang YY, Pan QK, Huang JP, Suganthan P, Gao L. An improved iterated greedy algorithm for the distributed assembly permutation flowshop scheduling problem. Comput Ind Eng 2021;152:107021. https://doi.org/10.1016/j.cie.2020.107021.
[38] Zhao Y, Zhang H. Application of machine learning and rule scheduling in a job-shop production control system. Int J Simul Model 2021;20:410–21. https://doi.org/10.2507/IJSIMM20-2-CO10.
[39] Chang J, Yu D, Hu Y, He W, Yu H. Deep reinforcement learning for dynamic flexible job shop scheduling with random job arrival. Processes 2022;10:760. https://doi.org/10.3390/pr10040760.
[40] Liu CL, Chang CC, Tseng CJ. Actor-critic deep reinforcement learning for solving job shop scheduling problems. IEEE Access 2020;8:71752–62. https://doi.org/10.1109/ACCESS.2020.2987820.
[41] Pan Z., Wang L., Wang J., Lu J. Deep reinforcement learning based optimization algorithm for permutation flow-shop scheduling. IEEE Trans Emerg Top Comput Intell; n.d. Available from: https://doi.org/10.1109/TETCI.2021.3098354.
[42] Yan Q, Wu W, Wang H. Deep reinforcement learning for distributed flow shop scheduling with flexible maintenance. Machines 2022;10:210. https://doi.org/10.3390/machines10030210.
[43] Wang L, Hu X, Wang Y, Xu S, Ma S, Yang K, et al. Dynamic job-shop scheduling in smart manufacturing using deep reinforcement learning. Comput Netw 2021;190:107969. https://doi.org/10.1016/j.comnet.2021.107969.
[44] Liu R, Piplani R, Toro C. Deep reinforcement learning for dynamic scheduling of a flexible job shop. Int J Prod Res 2022;0:1–21. https://doi.org/10.1080/00207543.2022.2058432.
[45] Yang S, Xu Z, Wang J. Intelligent decision-making of scheduling for dynamic permutation flowshop via deep reinforcement learning. Sensors 2021;21:1019. https://doi.org/10.3390/s21031019.
[46] Hu L, Liu Z, Hu W, Wang Y, Tan J, Wu F. Petri-net-based dynamic scheduling of flexible manufacturing system via deep reinforcement learning with graph convolutional network. J Manuf Syst 2020;55:1–14. https://doi.org/10.1016/j.jmsy.2020.02.004.
[47] Lin CC, Deng DJ, Chih YL, Chiu HT. Smart manufacturing scheduling with edge computing using multiclass Deep Q network. IEEE Trans Ind Inform 2019;15:4276–84. https://doi.org/10.1109/TII.2019.2908210.
[48] Joo T, Jun H, Shin D. Task allocation in human-machine manufacturing systems using deep reinforcement learning. Sustainability 2022;14:2245. https://doi.org/10.1109/TASE.2019.2956762; 10.3390/su14042245.
[49] Park I.B., Park J. Scalable scheduling of semiconductor packaging facilities using deep reinforcement learning. IEEE T Cybern; n.d. Available from: https://doi.org/10.1109/TCYB.2021.3128075.
[50] Luo S. Dynamic scheduling for flexible job shop with new job insertions by deep reinforcement learning. Appl Soft Comput 2020;91:106208. https://doi.org/10.1016/j.asoc.2020.106208.
[51] Lei K, Guo P, Zhao W, Wang Y, Qian L, Meng X, et al. A multi-action deep reinforcement learning framework for flexible Job-shop scheduling problem. Expert Syst Appl 2022;205:117796. https://doi.org/10.1016/j.eswa.2022.117796.
[52] Błażewicz J, Pesch E, Sterna M. The disjunctive graph machine representation of the job shop scheduling problem. Eur J Oper Res 2000;127:317–31. https://doi.org/10.1016/S0377-2217(99)00486-5.
[53] van Otterlo M, Wiering M. Reinforcement learning and Markov decision processes. In: Wiering M, van Otterlo M, editors. Reinforcement learning: state-of-the-art. Berlin, Heidelberg: Springer; 2012. p. 3–42. https://doi.org/10.1007/978-3-642-27645-3_1.
[54] Paszke A, Gross S, Massa F, Lerer A, Bradbury J, Chanan G, et al. PyTorch: an imperative style, high-performance deep learning library. Adv Neural Inf Process Syst 2019;32.
[55] Han BA, Yang JJ. Research on adaptive job shop scheduling problems based on dueling double DQN. IEEE Access 2020;8:186474–95. https://doi.org/10.1109/ACCESS.2020.3029868.
[56] Fonseca-Reyna YC, Martínez-Jiménez Y, NowéA Q. Learning algorithm performance for M-machine. N Jobs Flow Shop Sched Probl Minimize Make Invest Oper 2018;38(3):3.