


Article

An End-to-End Deep Learning Method for Dynamic Job Shop Scheduling Problem

Shifan Chen ¹, Zuyi Huang ¹  and Hongfei Guo ^{1,2,*}

¹ School of Intelligent Systems Science and Engineering, Jinan University, Zhuhai 519070, China; shifan@stu2019.jnu.edu.cn (S.C.); huangzuy@stu2020.jnu.edu.cn (Z.H.)

² Institute of Physical Internet, Jinan University, Zhuhai 519070, China

* Correspondence: ghf-2005@163.com

Abstract: Job shop scheduling problem (JSSP) is essential in the production, which can significantly improve production efficiency. Dynamic events such as machine breakdown and job rework frequently occur in smart manufacturing, making the dynamic job shop scheduling problem (DJSSP) methods urgently needed. Existing rule-based and meta-heuristic methods cannot cope with dynamic events in DJSSPs of different sizes in real time. This paper proposes an end-to-end transformer-based deep learning method named spatial pyramid pooling-based transformer (SPP-Transformer), which shows strong generalizability and can be applied to different-sized DJSSPs. The feature extraction module extracts the production environment features that are further compressed into fixed-length vectors by the feature compression module. Then, the action selection module selects the simple priority rule in real time. The experimental results show that the makespan of SPP-Transformer is 11.67% smaller than the average makespan of dispatching rules, meta-heuristic methods, and RL methods, proving that SPP-Transformer realizes effective dynamic scheduling without training different models for different DJSSPs. To the best of our knowledge, SPP-Transformer is the first application of an end-to-end transformer in DJSSP, which not only improves the productivity of industrial scheduling but also provides a paradigm for future research on deep learning in DJSSP.

Keywords: smart manufacturing; dynamic job shop scheduling problem; deep learning; transformer; spatial pyramid pooling; generalization



Citation: Chen, S.; Huang, Z.; Guo, H. An End-to-End Deep Learning Method for Dynamic Job Shop Scheduling Problem. *Machines* **2022**, *10*, 573. <https://doi.org/10.3390/machines10070573>

Academic Editor: Benoit Eynard

Received: 13 June 2022

Accepted: 14 July 2022

Published: 16 July 2022

Publisher's Note: MDPI stays neutral with regard to jurisdictional claims in published maps and institutional affiliations.



Copyright: © 2022 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<https://creativecommons.org/licenses/by/4.0/>).

1. Introduction

Smart manufacturing is an emerging concept with the development of Internet of Things technology [1], and scheduling plays a vital role in smart manufacturing since effective scheduling can improve profitability and resource utilization in the production [2]. Job shop scheduling problem (JSSP) is a classical scheduling problem in manufacturing, which is an NP-hard combinatorial optimization problem [3] that aims to find the optimal solution to production scheduling given a set of jobs, each with multiple operations processed by different machines. Most JSSP methods assume that the production environment is static, and the states of the production environment are known in advance. However, these JSSP methods have to reschedule the jobs when dynamic events such as machine breakdown and the insertion of new jobs occur in the production environment, causing the initial scheduling plans to be invalid. The extension of JSSP considering such dynamic events is called dynamic job shop scheduling problem (DJSSP), which effectively solved can better boost the productivity in the actual manufacturing.

DJSSP has been widely studied, and various methods are proposed. Dispatching rules are preferred in the production due to their simpler, easier implementation, and low computational complexity [4]. However, real-time selection of dispatching rules for scheduling is crucial because performances of different dispatching rules vary significantly in different DJSSPs. Simple priority rules are a kind of dispatching rule that uses a single job

or shop feature to prioritise jobs and process the job with the highest priority first. Human operators select simple priority rules based on their own experience in the production, which does not guarantee that the rules are optimal at each step. Moreover, selecting simple priority rules is intractable for human operators, failing to realize real-time scheduling. And using only one simple priority rule to schedule the jobs at each step is irrational since DJSSP often requires considering multi objects that cannot be achieved using a simple priority rule. Thus, two types of composite dispatching rules are proposed, which combine good features of different simple priority rules [5]. One is to deploy different simple priority rules on different machines; the other is to combine different simple priority rules to form a new dispatching rule using some operators such as addition, subtraction, multiplication, and division. But designing a stable composite dispatching rule is hard since different ways to combine the simple priority rules significantly affect the performance of the composite dispatching rule [6].

Meta-heuristic algorithm has been proven competitive in solving DJSSP. The most used ones in DJSSP are evolutionary algorithms (EAs) and swarm-based algorithms. EAs are algorithms inspired by biological evolution, which mimic the mechanism of biological evolution to generate better solutions, such as genetic algorithm (GA) [7] and differential evolutionary (DE) algorithm [8]. Whereas swarm-based algorithms mimic the behaviour of animal groups such as fish, birds, and wolf packs in nature and use the information exchange between groups to achieve optimization through simple and limited interactions between individuals, such as grey wolf optimizer (GWO) [9], ant colony optimization [10], and particle swarm optimization [11]. Although meta-heuristic algorithms can yield high-quality solutions, they often require massive iterations to get the optimal or near-optimal solution, and the parameters of the meta-heuristic algorithms are set empirically. Furthermore, meta-heuristic algorithms need to reschedule the jobs when dynamic events occur in the production environment, but rescheduling the jobs is impractical.

Reinforcement learning (RL) has been considered a powerful method for solving combinatorial optimization problems [12] and has been successfully used in various fields like AlphaGo wins the go game against one of the top human players [13], and AlphaFold predicts the protein structure with great accuracy [14]. The main component in RL is called agent, which takes actions to obtain maximum cumulative reward through interactions with the environment like a human being. Since the fast computation and ability to cope with dynamic events of RL [15], RL has achieved outstanding success in solving DJSSP. Q-learning is a classic algorithm of RL which chooses the action with the highest Q-value stored in the Q table. Q-learning not only improves multi-objective performance in JSSP but also effectively responds to dynamic events [16]. Considering that the Q table may become too large as the DJSSP size increases, causing the massive memory occupied and long computation time, deep reinforcement learning has been proposed. Deep reinforcement learning is also known as Deep Q-Network (DQN) and outperforms the popular dispatching rules in solving DJSSP [17]. Other RL methods such as proximal policy optimization (PPO) [18] and deep deterministic policy gradient [19] are also used in solving DJSSP and achieve great success. However, researchers have experimentally found that the optimal policy obtained by the RL agent is deterministic, but the observability of the unseen instance is partial. Thus RL models tend to generalize poorly [20], resulting in the models trained on a fixed-size instance can not cope with instances of different sizes.

Deep learning is a kind of representation-learning method in artificial intelligence, which contains a deeper network structure, helping it extract more accurate feature representation and map the feature representation with a particular output. Deep learning methods can generalize well on the unseen problem, and some researchers have successfully used deep learning to solve DJSSP. Weckman et al. [21] combine a multi-layer artificial neural network (ANN) with Giffler–Thomson (GT) algorithm to build a neural network scheduler, which can schedule the operations in real time. The ANN takes the real-time features related to the production environment as input and output the priorities of each operation. The priorities of each operation are further processed using GT algorithm to

get a scheduling plan that can deal with dynamic events. Sim et al. [22] further optimize the neural network scheduler, and they consider that the priorities of different operations output by the ANN might be the same for the same machine, causing the operations with the same priorities to be arbitrarily scheduled, so they propose to use dispatching rules to schedule the operations with the same priorities, but the GT algorithm is still needed to process the priorities of other operations to get a scheduling plan, failing to realize an end-to-end scheduling method. Zang et al. [23] design a hybrid deep neural network scheduler (HDNNS) combined with the two-dimensional convolution neural network (CNN), which can also extract the features of the production environment and output the priorities of each operation. But the ability to cope with dynamic events of HDNNS is not tested, and the CNN overlooks some critical features with time steps in the environment, which makes the performance of HDNNS still not satisfactory. Tian et al. [24] use the long short-term memory network (LSTM) that can retain the features with time steps to predict the production information and use the improved GA to achieve the scheduling. Shao et al. [25] also build a model based on LSTM named self-supervised long-short term memory (SS-LSTM), which can deal with dynamic events effectively, but the hyperparameters in SS-LSTM are difficult to determine. All the above-mentioned deep learning models either need one module to preprocess the input of the neural network or use other modules to process the output of the neural network to get the final scheduling plan, which may aggravate the errors between each module and lead to the suboptimal scheduling plan.

DJSSP can be viewed as a time series forecasting problem [26], and transformer is a deep learning-based method that is widely used for time-series forecasting [27]. Moreover, transformer with attention mechanism is a frontier model and some studies successfully use attention mechanism to solve JSSP. For example, Magalhães et al. [28] build a neural network with attention mechanism to solve dual resource constrained flexible job shop scheduling problem, Yang [29] utilizes attention mechanism to extract the representation of the production environment, and Chen et al. [30] use attention mechanism to allocate the processing resources to the more significant parts when solving the JSSP of large size. Most JSSP methods based on attention mechanism use attention mechanism to extract the features of the production environment, but there is no work to use the transformer based on attention mechanism to solve JSSP directly. In this paper, we build an end-to-end transformer-based deep learning model named spatial pyramid pooling-based transformer (SPP-Transformer) to solve different-sized DJSSPs. The disjunctive graph, which can represent the production environment features with time information [31], is adopted as the input of SPP-Transformer. SPP-Transformer utilizes a feature extraction module to extract the features in the disjunctive graph [32] and transforms them into vectors of different sizes. Then, a feature compression module compresses the different-sized feature vectors into fixed-length vectors. The fixed-length vectors are further input to the action selection module to select a simple priority rule. To the best of our knowledge, SPP-Transformer is the first work purely using a transformer-based model to solve DJSSP and achieve satisfactory results, providing a paradigm for further studies on deep learning in DJSSP. The contributions of this work are as follows:

1. An end-to-end transformer-based method is proposed, which takes the disjunctive graph as input and outputs the simple priority rule that can directly be applied in the production, providing a total data-driven method to solve DJSSP.
2. A feature compression module that contains a spatial pyramid pooling (SPP) [33] layer is presented to enhance the generalizability of the proposed model. The feature compression module enables the proposed model to be applied to DJSSP instances of different sizes and be trained on the various-sized instances represented by the disjunctive graph. Thus, there is no need to train the model instance-by-instance.
3. A sequence-to-action model that effectively deal with dynamic events is presented. Different from the conventional sequence-to-sequence deep learning models, the proposed sequence-to-action deep learning model reacts to the environment at each

step according to the real-time states of the production environment. Thus, dynamic events that happen at an uncertain step can be well tackled without extra measures.

The rest of this paper is organized as follows: Section 2 introduces the DJSSP with dynamic events such as machine breakdown and job rework, including the mathematical model and constraint conditions. Section 3 presents the whole structure of SPP-Transformer and gives the training details. Section 4 compares the performance of SPP-Transformer with exiting methods. Section 5 draws the conclusions finally.

2. Materials and Methods

2.1. Problem Description

The JSSP refers to allocating n jobs on m machines in a limited number of resources to optimize one or more objects, such as minimizing the makespan or achieving lower production costs. And DJSSP is an extension of JSSP, which considers the dynamic events that occur unexpectedly in the production and is more common than JSSP in the actual production. DJSSP can be described as follows: There are n jobs $J = \{J_1, J_2, \dots, J_n\}$ arrive at the shop successively, which will be processed on m machines $M = \{M_1, M_2, \dots, M_m\}$. Each job J_i consists of multiple operations $O = \{O_{i1}, O_{i2}, \dots, O_{ij}, \dots\}$. O_{ij} is the j th operation of job J_i , and it can only be processed on a specific machine. The dynamic events that may happen in the production are various, including machine breakdown, the insertion of new jobs, the change of the delivery dates and the processing time and job rework, etc. [34]. And DJSSP should satisfy the assumptions followed:

- (1) Each machine can only process one operation at the same time;
- (2) An operation is processed by only one machine at a time;
- (3) All jobs and machines are available at time 0;
- (4) An operation can not be interrupted if it has started unless machine breakdown;
- (5) The processing time of each operation may change during machining;
- (6) An operation cannot be processed until its preceding operations are completed.

In this paper, the dynamic events we considered include machine breakdown and job rework. We change the processing time and the machine order of job J_1 randomly to simulate machine breakdown or job rework. However, the number of machines and operations needed by J_1 are immutable. For simplicity, we only increase the processing time of the operation to represent machine breakdown and switch the machine order of the job to represent job rework. We use the parameter *Switch* to disorder the machine order to represent job rework. If *Switch* is "True", the machine order of the job is disordered. Moreover, we set a random rate f to represent the probability of machine breakdown, and the new processing time p'_{ij} of operation O_{ij} caused by machine breakdown is calculated as follows:

$$p'_{ij} = \begin{cases} p_{ij} + \min(1, \max(-1, \mathcal{N}(0, 0.1))) \cdot p_{ij}, & r < f \\ p_{ij}, & r > f \end{cases} \quad (1)$$

where p_{ij} is the original processing time of operation O_{ij} , $r \in [0, 1]$ is a random number, and $f \in [0, 1]$ is a hyperparameter. $\mathcal{N}(0, 0.1)$ is a normal distribution with mean 3 and variance 1.

SPP-Transformer is a sequence-to-action model and schedules the jobs at each step using the feature sequences from the disjunctive graph, which will change according to the production environment. And SPP-Transformer deals with dynamic events by taking actions depending on the input disjunctive graph without changing the scheduling procedure when dynamic events occur.

2.2. Mathematical Model

This study aims to minimize the makespan with the dynamic events, including machine breakdown and job rework that happened in the production. A mathematical model of DJSSP is formulated as follows with the notations listed in Table 1.

$$\begin{aligned} \min C_{\max} &= \min \left\{ \max \sum_{i=1}^n \sum_{j=1}^h (s_{ij} + p_{ij}) \right\} \\ \text{s.t. C1: } &p_{ij} > 0 \\ \text{C2: } &s_{ij} + p_{ij} \leq s_{i(j+1)} \\ \text{C3: } &e_{ij} - s_{ij} = p_{ij} \\ \text{C4: } &e_{ij} \neq e_{i'j'} \\ \text{C5: } &T = n \times m \end{aligned} \quad (2)$$

where the objective is minimizing the makespan as shown in Equation (2). C1 indicates that the processing time of all operations is positive. C2 shows that an operation can not be processed if its preceding operations are not completed. C3 represents that an operation cannot be interrupted while it is being processed since the operation O_{ij} and the operation $O_{i'j'}$ may be processed on the same machine. C4 is built to make sure that each machine can only process one operation at a time by ensuring the end time of O_{ij} and $O_{i'j'}$ is different. C5 makes sure that each operation is processed by only one machine at a time.

Table 1. DJSSP specific parameters table.

Parameters	Description
i	index of jobs, $i = 1, 2, \dots, n$
j	index of operations of a job, $j = 1, 2, \dots, h$
m	the number of machines
C_{\max}	the maximum completion time of all jobs
s_{ij}	the start time of operation O_{ij}
e_{ij}	the end time of operation O_{ij}
$e_{i'j'}$	the end time of operation $O_{i'j'}$
p_{ij}	the processing time of operation O_{ij}
T	the maximum total number of all planned operations at the current time

3. Proposed Method

3.1. Input and Output Representation

3.1.1. Input Representation

Considering the disjunctive graph can represent both the global and local features of the production and divide discrete-time steps in terms of operation assignment, it is adopted as the input of our model to help collect more features with time information. Disjunctive graph $G = (V, C \cup D)$ is a kind of directed graph, and is also used to represent the JSSP popularly. All operations of all jobs construct the set V , and these operations are represented as vertices. However, V includes two more dummy vertices called source and sink, representing the start and the end of the scheduling, respectively. Moreover, the processing time of the two dummy vertices is zero. The precedence constraints initially between every two consecutive operations are reflected by the directed conjunctive edges that construct the set C . Unordered operations that can be processed on the same machine are linked to each other with the undirected disjunctive edges, which construct the set D [35]. Thus, solving a JSSP means turning a disjunctive edge into a directed edge to get a directed acyclic graph and determine the operation sequences on each machine. A simple JSSP instance and its solution represented by the disjunctive graphs are shown in Figure 1.

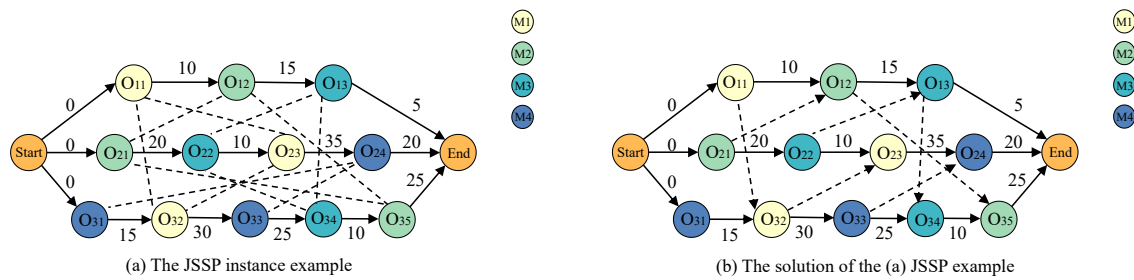


Figure 1. An example of JSSP instance and its solution representing by the disjunctive graphs.

Each vertex denotes an operation of a job, and the content O_{ij} in it denotes the j th operation of job i . The solid lines denote the conjunctive edges, while the dotted lines denote the disjunctive edges. The numbers near the edges denote the processing time needed for the operations where the edges begin. Furthermore, the vertices of the same colour mean that the operations represented by these vertices are processed on the same machine.

However, the disjunctive graphs only reflect the static features in JSSP, failing to represent the dynamic features in DJSSP. Therefore, several following attributes are added to the vertices in the disjunctive graph to represent the dynamic features [36]:

- (1) Operation identified number: The corresponding serial number of this operation in the job is added to the vertex;
- (2) Job identified number: If a job includes the operation, its serial number is added to the vertex;
- (3) Machine identified number: If a machine can process this operation, its serial number is added to the vertex. And 0 is added if there are no machines that can process this operation;
- (4) Node situation: 1, 0, and -1 represent that the operation is finished, being processed, and unfinished, respectively;
- (5) Completion rate: When this operation is finished, the completion rate of the entire job;
- (6) Number of subsequent operations: The number of remaining operations to be finished;
- (7) Time for waiting: The amount of time that has elapsed since the start before this operation can be processed;
- (8) Processing time: The time required to process the operation;
- (9) Remaining time: The remaining processing time of the operation (0 for unprocessed);
- (10) Doable: It is "True" if the operation is doable.

3.1.2. Output Representation

Because of the black-box nature of the neural network, a detailed scheduling operation output by a neural network may be unsafe. Therefore, dispatching rules, which are widely used in JSSP since they are easy to implement and compute fast, are adopted as the output of our model. Dispatching rules prioritise the jobs waiting to be processed according to shop features (e.g., machine utilization) or job features, and the machine will first process the job with the highest priority [37]. Simple priority rule is a kind of dispatching rule and some typical simple priority rules have shown their effectiveness in DJSSP. In our work, SPP-Transformer takes the following 8 simple priority rules as output:

- (1) First In First Out (FIFO): The machine will first process the job that first arrived at the queue of the machine;
- (2) Last In Last Out (LIFO): The machine will first process the job that last arrived at the queue of the machine;
- (3) Most Operations Remaining (MOR): The machine will first process the job that has the most remaining operations;
- (4) Least Operations Remaining (LOR): The machine will first process the job that has the least remaining operations;

- (5) Longest Processing Time (LPT): The machine will first process the job that has the longest processing time;
- (6) Shortest Processing Time (SPT): The machine will first process the job that has the shortest processing time;
- (7) Longest Total Processing Time (LTPT): The machine will first process the job that has the longest total processing time;
- (8) Shortest Total Processing Time (STPT): The machine will first process the job that has the shortest total processing time.

3.2. Transformer Layer

3.2.1. Feature Extraction Module

The feature extraction module of SPP-Transformer consists of 6 encoder layers, and each encoder layer contains stacked self-attention, a fully connected feed-forward network (FFN), residual connection [38], and layer normalization (LN) [39]. All operations $O = \{O_{i1}, O_{i2}, \dots, O_{ij}, \dots\}$ corresponding with the vertices in the disjunctive graph construct a feature sequence $X = \{x_{i1}, x_{i2}, \dots, x_{ij}, \dots\}$, which can be viewed as a vector with 10 dimensions because 10 attributes are added to each vertex. And the feature sequence is input to the feature extraction module and processed by the feature extraction module to get a feature vector so that the features of the production can be better extracted.

Self-attention computes the attention distribution over the input feature sequence [40] and maps the query and the set of key-value pairs to an output. Three different linear transformations are learned and performed on the input feature sequence to obtain the queries, keys, and values of all the operations. The query of an operation represents the type of the operation it concerns. The key of an operation represents its type, and the value of an operation contains the information about itself. Furthermore, the queries, keys, and values of all the operations construct the query matrix Q , the key matrix K , and the value matrix V , respectively. And $Q, K, V \in \mathbb{R}^{d_L \times d_F}$, where d_L represents the number of total operations while d_F represents the number of the attributes of each operation. Firstly, the dot products of the queries of each operation and the keys of the other operations are computed to obtain the matching scores over all operations, and the matching scores are further divided by $\sqrt{d_F}$ to avoid the attention over-focusing on the high-scores operations. Then, a softmax function is applied on the matching scores for obtaining the attention weights of the values, and we compute the $output \in \mathbb{R}^{d_L \times d_F}$ of a single attention function by finally conducting the matrix multiplication of the attention weights and values. And the attention function is shown in Equation (3). It is worth mentioning that such a process can be executed in parallel. Thus, high computational efficiency can be achieved.

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK'}{\sqrt{d_F}}\right)V \quad (3)$$

Further, self-attention is often combined with multi-head attention and the outputs of multiple attention functions are concatenated to obtain more information with different query weight matrices W_t^Q , key weight matrices W_t^K , value weight matrices W_t^V , as shown in Equation (4):

$$\begin{aligned} \text{MultiHead}(Q, K, V) &= \text{Concat}(\text{head}_1, \dots, \text{head}_H)W^{\text{multi}} \\ \text{where head}_t &= \text{Attention}\left(QW_t^Q, KW_t^K, VW_t^V\right) \end{aligned} \quad (4)$$

where H represents the number of self-attention heads that are applied on the input sequence, $W_t^Q, W_t^K, W_t^V \in \mathbb{R}^{d_F \times \frac{d_F}{H}}$ and $W^{\text{multi}} \in \mathbb{R}^{d_F \times d_F}$ represent the parameter matrices. By concatenating the outputs of different attention functions, we obtain a vector which contains more information about the input feature sequence.

Besides the multi-head attention sub-layer, a fully connected FFN is included in both the feature extraction module and the action selection module, which consists of a nonlinear activation function and two linear transformations in between to transform the input from a d_{Input} dimensions vector to a d_{Output} dimensions vector with the parameter W_1 , W_2 , b_1 , and b_2 as shown in Equation (5).

$$\text{FFN}(x) = \max(0, xW_1 + b_1)W_2 + b_2 \quad (5)$$

Moreover, residual connections are applied around FFN sub-layer and every multi-head attention, which helps prevent the gradient disappearance. We use LN after the residual connection to prevent internal covariate shifts since a single batch may contain several examples.

3.2.2. Feature Compression Module

Various-sized feature vectors are obtained after using the feature extraction module to process the different disjunctive graphs. And the feature compression module is proposed to compress these various-sized feature vectors into fixed-length. Various-sized feature vectors are input to the feature compression module, and the feature compression module outputs fixed-length feature vectors. The feature compression module is the SPP layer that is typically used in computer vision to aggregate the convolutional features of images by splitting images into a range of grids at each level in the pyramid and aggregating local features. Images can also be viewed as vectors with multiple dimensions, so the vectors output by the feature extraction module can be compressed into fixed-length vectors by the SPP layer in the feature compression module.

To sample the extracted features at different scales from the disjunctive graph, four different bins are used for adaptive pooling in the SPP layer. After various-sized feature vectors are input to the feature compression module, the four different-sized bins in the SPP layer pool the feature vectors to get four fixed-length vectors and all the four vectors are 64 dimensions. The sliding windows and the strides of the four bins are changing adaptively according to the input various-sized vectors. For a bin, the size and the stride of its sliding window are determined using Equations (6) and (7), respectively.

$$\text{win} = \text{ceil} \left(\frac{d_{In}}{d_{Out}} \right) \quad (6)$$

$$\text{str} = \text{floor} \left(\frac{d_{In}}{d_{Out}} \right) \quad (7)$$

where the “win” and the “str” represent the size and the stride of the sliding window, respectively. The “ceil(·)” and the “floor(·)” represent rounding up and rounding down, respectively. d_{In} and d_{Out} represent the sizes of the input vectors and the sizes of the output vectors, respectively. It is worth mentioning that the sizes of the input vectors are the same with the sizes of the vectors output by the feature extraction module.

Finally, four fixed-length vectors whose sizes are 64×1 , 32×2 , 16×4 , and 8×8 , respectively, are output by the four bins. Next, each vector is reshaped to 64×1 and connected to construct the final fixed-length feature vector of 256 dimensions. With the fixed-length feature vector, the generalizability of SPP-Transformer is achieved, enabling SPP-Transformer to be trained once using existing instances but be applied on different-sized DJSSPs instead of being trained instance-by-instance. The framework of the feature compression module in our work is shown in Figure 2.

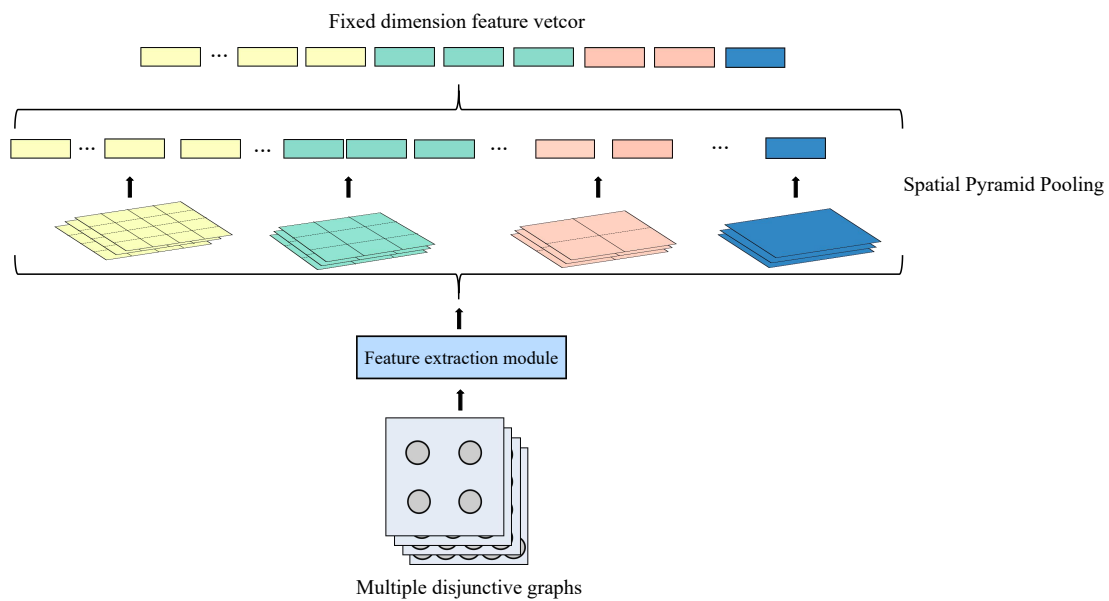


Figure 2. The framework of the feature compression module.

3.2.3. Action Selection Module

The fixed-length vectors of 256 dimensions output by the feature compression module are further input to the action selection module, and the action selection module outputs the simple priority rules using the input fixed-length vectors. The action selection module of SPP-Transformer consists of 6 decoder layers, a linear transformation and a softmax function. And each decoder layer is similar to the encoder layer in the feature extraction module, but contains a third sub-layer to perform multi-head attention over the output of the feature compression module. Moreover, masking is added to the self-attention sub-layer combined with the fact that the output of the feature compression module is offset by one position to avoid paying attention to the subsequent position [41].

And we utilize the linear transformation and softmax function to convert the output of the final decoder layer to predicted next-token probabilities. The linear transformation is a single layer composed of 16 neurons, which is used to transform the vectors of 256 dimensions to the vectors of 8 dimensions since there are 8 candidate simple priority rules to be selected. And each dimension of the vector represents each simple priority rules, respectively. Moreover, the specific expression of the softmax function used in this paper is shown in Equation (8) below.

$$\text{Softmax}(z_v) = \frac{e^{z_v}}{\sum_{v=1}^8 e^{z_v}} \quad (8)$$

where $v = 1, 2, \dots, 8$, and z_v represents the value of each dimension of the 8 dimensions vectors, corresponding to one of the eight simple priority rules. The probability of being selected of each simple priority rule is calculated using Equation (8).

3.3. Training Procedure

SPP-Transformer is built with the feature extraction module, the feature compression module and the action selection module, and its framework is shown in Figure 3. A DJSSP instance represented by the disjunctive graph is input to SPP-Transformer, and the feature extraction module extracts the initial features in the disjunctive graph and turns them into feature vectors of different sizes. Then, the feature compression module compresses the different-sized feature vectors into fixed-length vectors, which are 256 dimensions. Finally, the action selection module further transforms the 256 dimensions vectors to

8 dimensions vectors using the linear transformation and selects a simple priority rule from eight candidate simple priority rules at each step using a softmax function.

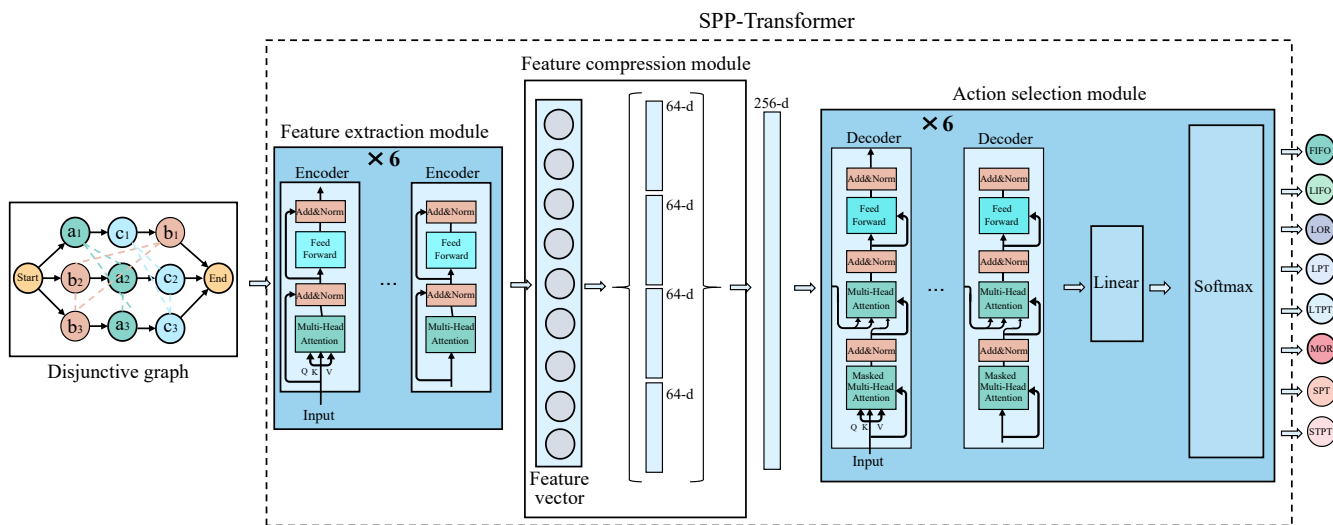


Figure 3. The whole framework of SPP-Transformer.

DJSSP in this paper can be modelled as a multi-classification problem since a specific instance represented by the disjunctive graph is related to one of the eight simple priority rules. The softmax function shown in Equation (8) is used as the output unit. And the cross-entropy loss function as shown in Equation (9) is used to calculate the cross-entropy loss between the simple priority rule output by our model and the ground truth from the expert to train SPP-Transformer.

$$L = -\frac{1}{N} \sum_c \sum_{v=1}^8 y_{cv} \log(p_{cv}) \tag{9}$$

where N represents the number of the total disjunctive graphs in the training dataset, v represents one of the eight simple priority rules, c represents each disjunctive graph, $y_{cv} \in \{0, 1\}$ is the label representing whether the disjunctive graph c is related to the simple priority rule v , and p_{cv} represents the probability that the disjunctive graph c is related to the simple priority rule v .

4. Numerical Experiments

4.1. Experimental Setup

We conduct the experiment based on the gymjsp, a benchmark whose data is from the OR-Library. And OR-Library is a dataset of various Operational Research (OR) problems, including JSSP [42]. Moreover, we test SPP-Transformer on the JSSP instances FT06, FT10, FT20, and LA01-20. The FT and LA instances are from different distributions, and the FT instances are proposed by Fisher and Henry [43], while the LA instances are proposed by Lawrence [44]. The FT and LA instances used in this paper are of different sizes, helping test if SPP-Transformer generalizes well on different-sized instances. The performance of SPP-Transformer significantly relies on the training samples. Therefore, the high-quality training samples from the expert whose solutions are viewed as optimal are required. The RL method named TOFA is the state-of-the-art method recently in solving DJSSP [45]. And the action selected by the TOFA agent at each step is one of the 8 simple priority rules, including FIFO, LIFO, MOR, LOR, LPT, SPT, LTPT, and STPT. We use the trained optimal TOFA agent to interact with the production environment and record the disjunctive graphs and the actions taken by the agent at each step. The collected disjunctive graphs are used as the original training data, and the collected actions are used as the training labels to obtain 100,000 training samples. To verify the generalizability of SPP-Transformer,

we test SPP-Transformer on the different-sized instances without further training. And we randomly change the machine order and the processing time of a job to represent job rework or machine breakdown when testing the ability of SPP-Transformer to tackle dynamic events. Moreover, SPP-Transformer is a sequence-to-action model and deals with dynamic events by taking actions according to the current environment without extra measures. The hyperparameters used in this paper are shown in Table 2, and we test our model on the machine with Intel(R) Xeon(R) Gold 6230 CPU @ 2.10GHz and a single NVIDIA GeForce RTX 3090 GPU.

Table 2. List of hyperparameters and their values.

Hyperparameter	Value
Number of encoder layers	6
Number of decoder layers	6
Number of feed-forward network layers	1
Random rate f	0.1
Switch	True
Number of heads	8
Number of training epochs	10,000
Number of test epochs	100
Optimizer	Adam
Mini-batch size	128
Dropout rate	0.1

4.2. Performance Analysis

In this subsection, we compare SPP-transformer with simple priority rules, composite dispatching rules, meta-heuristic methods, and RL methods. The results are obtained and shown in Table 3.

We use “the number of machines \times the number of jobs” to represent the size of each instance. Makespan is an important indicator in the production, which means the processing completion time, and the smaller makespan means better method performance. The “Opt” represents the optimal value obtained by TOFA. The best results obtained among these methods for comparison are highlighted in bold font. Further, the relative percentage deviation (RPD) of each method is calculated using Equation (10).

$$RPD = \frac{Alg - Opt}{Opt} \times 100\% \quad (10)$$

Table 3. The comparison between each method on makespan and RPD (%).

Instance	Size	Opt	Methods									
			Simple Priority Rules		Composite Dispatching Rules		Meta-Heuristic Methods		RL Methods		SPP-Transformer	
			Makespan	RPD	Makespan	RPD	Makespan	RPD	Makespan	RPD	Makespan	RPD
FT06	6 × 6	59	72	22.46	79	33.47	59	0	65	10.17	59	0
FT10	10 × 10	1002	1221	21.84	1169	16.62	1287	28.48	1288	28.57	1180	17.76
FT20	5 × 20	1205	1469	21.91	1383	14.79	1611	33.69	1440	19.5	1231	2.16
LA01	5 × 10	666	833	25.09	806	20.98	759	14.02	826	24.07	699	4.95
LA02	5 × 10	684	858	25.49	825	20.58	798	16.67	821	20.08	716	4.68
LA03	5 × 10	615	748	21.69	758	23.29	721	17.29	735	19.51	651	5.85
LA04	5 × 10	615	788	28.15	747	21.47	715	16.26	758	23.31	670	8.94
LA05	5 × 10	593	651	9.8	634	6.83	616	3.88	629	6.01	593	0
LA06	5 × 15	926	1090	17.66	1124	21.41	1016	9.68	1004	8.46	926	0
LA07	5 × 15	920	1079	17.28	1053	14.49	1056	14.75	1020	10.87	956	3.91
LA08	5 × 15	866	991	14.42	976	12.73	1014	17.13	1034	19.44	863	−0.35
LA09	5 × 15	952	1119	17.54	1094	14.89	1059	11.28	1039	9.17	951	−0.11
LA10	5 × 15	958	1085	13.3	1081	12.84	1018	6.23	1015	5.92	958	0
LA11	5 × 20	1222	1416	15.87	1430	16.98	1367	11.87	1359	11.21	1222	0
LA12	5 × 20	1039	1204	15.84	1214	16.8	1170	12.61	1149	10.62	1039	0
LA13	5 × 20	1151	1272	10.48	1285	11.62	1303	13.18	1247	8.34	1150	−0.09
LA14	5 × 20	1292	1451	12.34	1453	12.46	1350	4.49	1299	0.54	1292	0
LA15	5 × 20	1336	1500	12.29	1437	7.54	1491	11.63	1433	7.26	1321	−1.12
LA16	10 × 10	1108	1206	8.84	1200	8.28	1194	7.76	1155	4.21	1074	−3.07
LA17	10 × 10	844	976	15.61	994	17.74	1017	20.49	927	9.83	812	−3.79
LA18	10 × 10	942	1092	15.91	1031	9.45	1099	16.7	1088	15.46	942	0
LA19	10 × 10	952	1017	6.79	985	3.42	1130	18.7	1022	7.39	935	−1.79
LA20	10 × 10	1026	1154	12.47	1120	9.11	1179	14.95	1136	10.72	979	−4.58
Average		912	1056	16.66	1038	15.12	1045	13.99	1021	12.64	923	1.45

The “Alg” is the makespan obtained by each method on each instance. RPD reflects the distances between each method and the optimal method, and the smaller RPD means better method performance. Moreover, it can be seen from Equation (10) that the RPD is a negative value when the makespan obtained by the method is smaller than that obtained by TOFA, representing that this method even outperforms the state-of-the-art method. The average makespan and RPD of each method are shown in Table 3. Specifically, the “Makespan” and the “RPD” of the simple priority rules on each instance are, respectively, the average makespan and the average RPD of eight simple priority rules, including FIFO, LIFO, MOR, LOR, LPT, SPT, LTPT, and STPT. The “Makespan” and the “RPD” of the composite dispatching rules on each instance are, respectively, the average makespan and the average RPD of four composite dispatching rules, including MOR+LPT, LOR+SPT, FCFS*S, and SI/Q. The “Makespan” and the “RPD” of meta-heuristic methods on each instance are, respectively, the average makespan and the average RPD of three meta-heuristic methods, including GA, DE, and GWO. The “Makespan” and the “RPD” of RL methods on each instance are, respectively, the average makespan and the average RPD of three RL methods, including DQN, Rainbow, and PPO. MOR+LPT means processing the job with the most remaining operations and the longest processing time first, but LOR+SPT means processing the job with the least remaining operations and the shortest processing time first. FCFS*S means setting a specific value in advance, and if the number of waiting jobs is more than this value, using SPT; otherwise, using FIFO. SI/Q means using SPT mainly, but if the job in the queue is joining the queue of the next machine and the queue it will join beyond a specific length, this job is selected first. GA is a traditional meta-heuristic method to solve JSSP, while DE and GWO are relatively novel meta-heuristic methods. Rainbow is an extension of traditional DQN and performs well on atari games [46], and PPO is an effective policy gradient RL method [47]. The detailed results of each simple priority rule, each composite dispatching rule, each meta-heuristic method, and each RL method are listed in Appendix A.

And we depict the average RPD of each method to clearly show the differences between them. The result is shown in Figure 4.

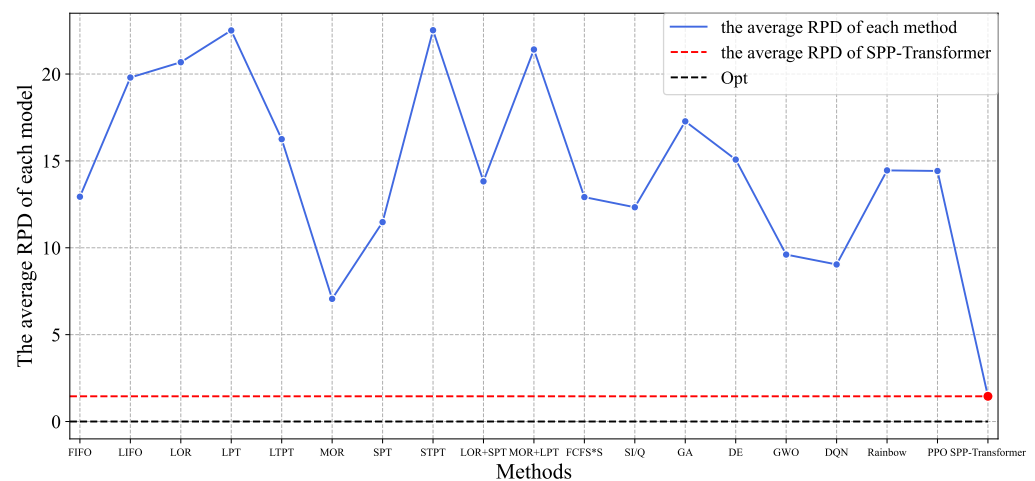


Figure 4. The average PRD(%) of each method on each instance.

The red dotted line represents the average RPD of SPP-Transformer and it can be found that the average RPDs of the other 18 methods are above the red dotted line, representing that the average RPDs of the other 18 methods are larger than that of SPP-Transformer and SPP-Transformer outperforms the other 18 methods. Moreover, it can be found from Table 3 that the RPDs of SPP-Transformer on LA08-09, LA13, LA15-17, LA19-20 are negative, representing that SPP-Transformer even gets better results than the optimal solutions on these instances. The makespan of SPP-Transformer is 11.67% smaller than the average makespan of the other 18 methods. Specifically, the makespan of SPP-Transformer is 14.30%

smaller than the average makespan of all dispatching rules, including simple priority rules and composite dispatching rules, representing that SPP-Transformer can select the effective dispatching rule at different steps according to the current environment. Moreover, MOR outperforms the other 17 methods except SPP-Transformer. The performances of other dispatching rules vary widely on different instances, reflecting that there is no dispatching rule that can always perform well under any situation. Thus, the dynamic selection of dispatching rules is needed. The composite dispatching rules do not achieve the best result among all dispatching rules, which shows that combining different simple priority rules to construct an effective composite dispatching rule is not easy and requires domain knowledge. The makespan of SPP-Transformer is 16.99% smaller than the average makespan of meta-heuristic methods, representing that SPP-Transformer can effectively deal with the dynamic events without extra measures while meta-heuristic methods always need to reschedule the jobs when dynamic events occur. Both SPP-Transformer and RL methods select dispatching rules at different steps, but the makespan of SPP-Transformer is 13.82% smaller than the average makespan of RL methods, representing that SPP-Transformer can select the more effective dispatching rules than RL methods.

And Figures 5–7 are the Gantt charts that represent the scheduling schemes obtained by SPP-Transformer to solve FT06, LA01, and LA20, respectively.

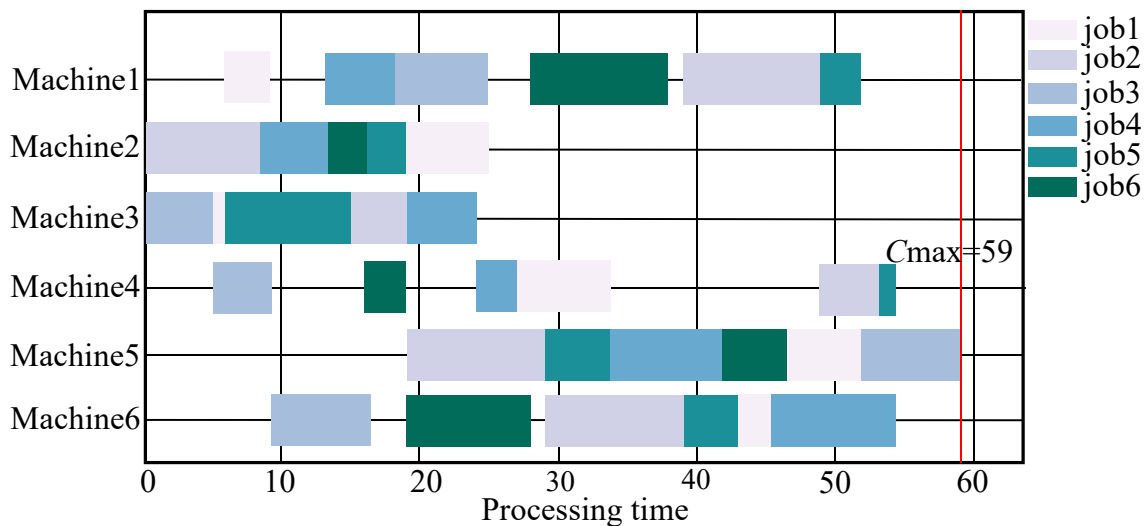


Figure 5. The Gantt chart of FT06.

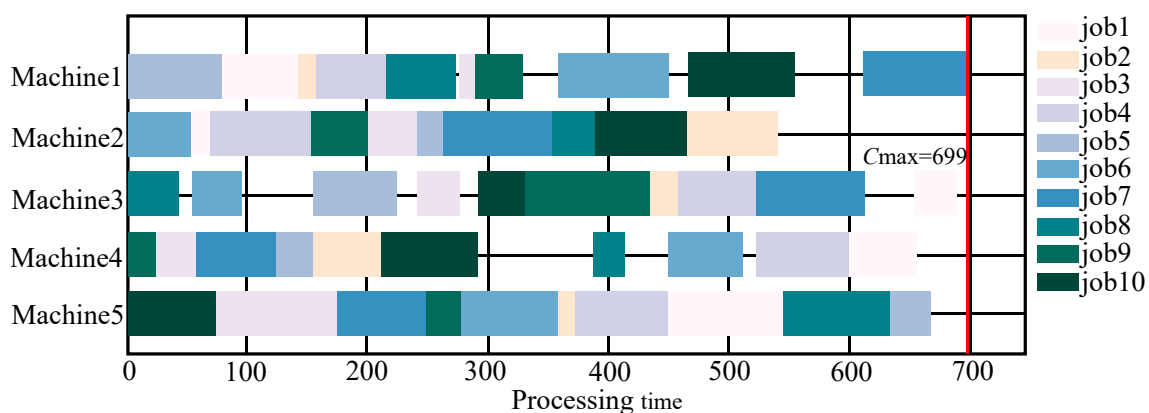


Figure 6. The Gantt chart of LA01.

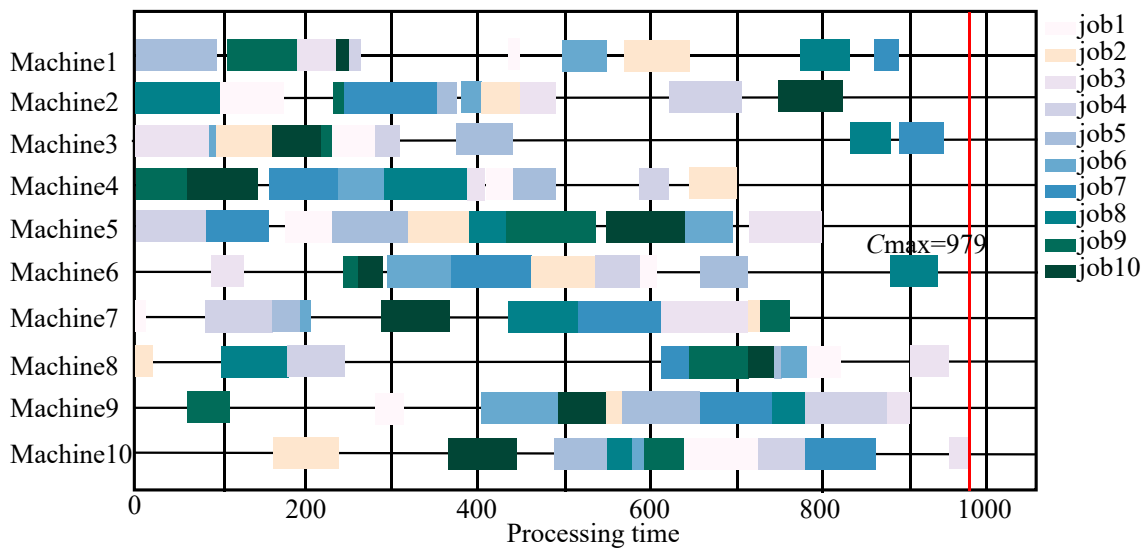


Figure 7. The Gantt chart of LA20.

The rectangles with different colours represent the different jobs, the horizontal axis represents the processing time, the vertical axis represents the different machines, and the C_{max} is related to the minimum makespan. The closer the rectangular areas are, the more balanced the load of the machine and the better the scheduling effect [48]. It can be found that the rectangles in the three Gantt charts are relatively close, reflecting that our model realizes satisfactory scheduling. The above analyses prove that using an end-to-end transformer to solve DJSSP is feasible and rather effective.

4.3. Data Ablation

Transformer-based models are known to require large amounts of data for training [49] and we verify the dependence of SPP-Transformer on the scale of datasets by doing the experiment that trains SPP-Transformer on different scale datasets. We randomly sample 1000 and 10,000 samples from the original dataset containing 100,000 samples to build two new subsets. For simplicity, other hyperparameters are kept fixed except for the scales of the training datasets. The line chart is depicted to show the RPD of each SPP-Transformer on each instance more clearly, as shown in Figure 8. And “ours-100000”, “ours-10000”, and “ours-1000” represent the SPP-Transformer trained on the 100,000, 10,000, and 1000 samples datasets, respectively. The detailed results of each model are listed in Appendix B.

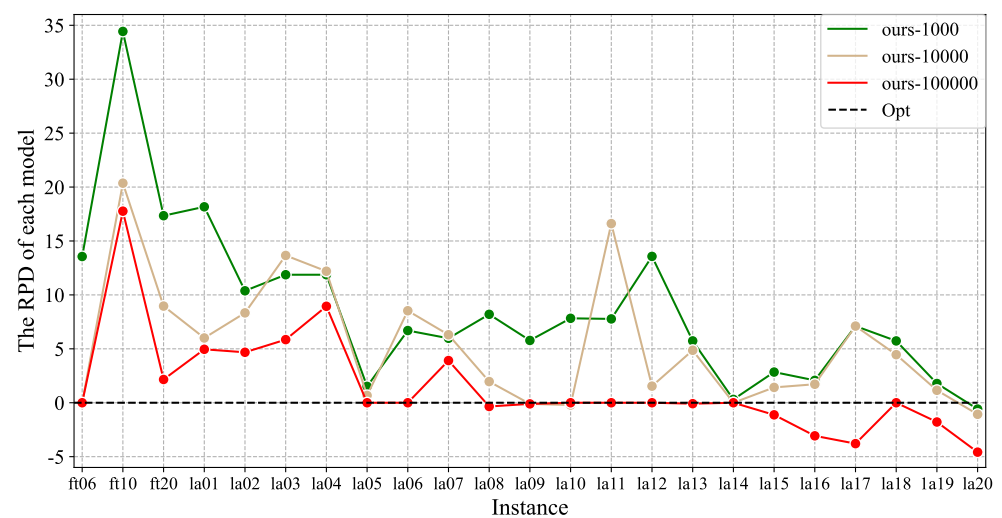


Figure 8. The RPD of each SPP-Transformer on each instance.

It can be found that except for the RPD on LA04, LA05, LA07, and LA14 being relatively closer, the performances of the three SPP-Transformers on other instances are still significantly different. Moreover, “ours-100000” performs best among the three models and its performance is even better than the optimal solutions on some instances, reflecting that a larger dataset helps improve the performance of SPP-Transformer. The fact that the improvement between “ours-100000” and “ours-10000” is more significant than that between “ours-10000” and “ours-1000” reasonably explains that the larger dataset can better improve the performance of the model, which provides a gist for us to further improve the performance of SPP-Transformer.

5. Conclusions

This paper introduces an end-to-end transformer-based deep learning model called SPP-Transformer to solve DJSSP of different sizes. SPP-Transformer is constructed by a feature extraction module, a feature compression module, and an action selection module. Firstly, a DJSSP instance represented by the disjunctive graph is input to the feature extraction module, and the feature extraction module extracts the features in the disjunctive graph and transforms them into various-sized feature vectors. Then, the feature compression module compresses the various-sized feature vectors into the fixed-length vectors. Finally, the action selection module selects the simple priority rule to schedule the job at each step to complete the scheduling. The experimental results show that the makespan of SPP-Transformer is 11.67% smaller than the average makespan of dispatching rules, meta-heuristic methods, and RL methods. Specifically, the makespan of SPP-Transformer is 14.30% smaller than the average makespan of dispatching rules, 16.99% smaller than the average makespan of meta-heuristic methods, and 13.82% smaller than the average makespan of RL methods. The experimental results prove that SPP-Transformer can generalize well on different-sized JSSP instances and effectively deal with dynamic events. Moreover, it is also proved that a larger training dataset improves the performance of SPP-Transformer. However, the training time of SPP-Transformer is relatively long since there are lots of parameters in SPP-Transformer and collecting enough data is not easy since it needs a large dataset, which should be further solved. And the dynamic events considered in this paper are limited, which can be further improved. To the best of our knowledge, this is the first study of an end-to-end transformer in DJSSP, providing a paradigm for future research on deep learning in DJSSP. Using the disjunctive graphs as input and dispatching rules as output, further research can be conducted to explore other end-to-end deep learning-based methods besides the transformer-based methods to solve DJSSP.

Author Contributions: Conceptualization, S.C.; methodology, S.C.; software, S.C.; validation, S.C., Z.H. and H.G.; formal analysis, S.C.; investigation, S.C. and Z.H.; resources, S.C. and Z.H.; data curation, S.C.; writing—original draft preparation, S.C. and Z.H.; writing—review and editing, S.C. and H.G.; visualization, S.C. and Z.H.; project administration, S.C.; funding acquisition, S.C. and H.G. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by Special Funds for the Cultivation of Guangdong College Students’ Scientific and Technological Innovation (“Climbing Program” Special Funds) (Grant No.pdjh2022b0059), the Guangdong Academic Degree and Graduate Education Reform Research Project (Grant No.2019JGXM15), Guangdong Province Higher Education Teaching Research and Reform Project (Grant No.2020059), the Guangdong Graduate Education Innovation Project (Grant No.82620516), Jinan University Off-campus Practice Teaching Base Construction Project (Grant No.55691207), Guangdong Higher Education Association “14th Five-Year Plan” Higher Education Research Topic (Grant No.21GZD09), Guangdong Province “Quality Engineering” Construction Project (Grant No.210308), Jinan University Graduate Education Teaching Achievement Cultivation Project (Grant No.2021YPY010), Jinan University Teaching Reform Research Project (Grant No.JG2022086).

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: All experiment datasets of solving DJSSP are available at <https://github.com/Yunhui1998/Gymjsp> (accessed on 17 May 2022).

Conflicts of Interest: The authors declare no conflict of interest.

Appendix A

Table A1. The comparison between simple priority rules and SPP-Transformer on makespan.

Instance	Size	Opt	Methods								
			FIFO	LIFO	LOR	LPT	LTPT	MOR	SPT	STPT	SPP-Transformer
FT06	6 × 6	59	65	70	68	77	68	59	88	83	59
FT10	10 × 10	1002	1230	1201	1352	1295	1190	1163	1074	1262	1180
FT20	5 × 20	1205	1658	1304	1487	1631	1495	1601	1267	1309	1231
LA01	5 × 10	666	848	772	941	822	835	763	751	933	699
LA02	5 × 10	684	821	799	982	990	881	812	821	761	716
LA03	5 × 10	615	734	765	758	825	696	726	672	811	651
LA04	5 × 10	615	737	827	745	818	887	706	711	874	670
LA05	5 × 10	593	593	681	704	693	658	593	610	677	593
LA06	5 × 15	926	1014	1246	1095	1125	1098	926	1200	1012	926
LA07	5 × 15	920	1105	1156	1137	1069	1025	1001	1034	1105	956
LA08	5 × 15	866	982	1101	995	1035	952	925	942	995	863
LA09	5 × 15	952	1011	1089	1149	1183	1304	951	1045	1220	951
LA10	5 × 15	958	1035	1132	1032	1132	1033	958	1049	1312	958
LA11	5 × 20	1222	1229	1488	1586	1467	1416	1222	1473	1446	1222
LA12	5 × 20	1039	1039	1329	1295	1240	1126	1039	1203	1358	1039
LA13	5 × 20	1151	1160	1519	1320	1230	1252	1150	1275	1267	1150
LA14	5 × 20	1292	1321	1505	1567	1434	1419	1292	1427	1646	1292
LA15	5 × 20	1336	1471	1519	1598	1612	1394	1436	1339	1632	1321
LA16	10 × 10	1108	1297	1306	1119	1229	1165	1108	1156	1268	1074
LA17	10 × 10	844	908	987	948	1082	993	844	924	1120	812
LA18	10 × 10	942	1057	1178	1154	1114	1198	942	981	1111	942
LA19	10 × 10	952	1062	992	1004	1062	1004	1088	940	981	935
LA20	10 × 10	1026	1243	1092	1207	1272	1086	1130	1000	1201	979
Average		912	1027	1089	1098	1106	1051	975	999	1104	923

Table A2. The comparison between simple priority rules and SPP-Transformer on RPD (%).

Instance	Size	Methods								
		FIFO	LIFO	LOR	LPT	LTPT	MOR	SPT	STPT	SPP-Transformer
FT06	6 × 6	10.17	18.64	15.25	30.51	15.25	0	49.15	40.68	0
FT10	10 × 10	22.75	19.86	34.93	29.24	18.76	16.07	7.19	25.95	17.76
FT20	5 × 20	37.59	8.22	23.4	35.35	24.07	32.86	5.15	8.63	2.16
LA01	5 × 10	27.33	15.92	41.29	23.42	25.38	14.56	12.76	40.09	4.95
LA02	5 × 10	20.03	16.81	43.57	44.74	28.8	18.71	20.03	11.26	4.68
LA03	5 × 10	19.35	24.39	23.25	34.15	13.17	18.05	9.27	31.87	5.85
LA04	5 × 10	19.84	34.47	21.14	33.01	44.23	14.8	15.61	42.11	8.94
LA05	5 × 10	0	14.84	18.72	16.86	10.96	0	2.87	14.17	0
LA06	5 × 15	9.5	34.56	18.25	21.49	18.57	0	29.59	9.29	0
LA07	5 × 15	20.11	25.65	23.59	16.2	11.41	8.8	12.39	20.11	3.91
LA08	5 × 15	13.39	27.14	14.9	19.52	9.93	6.81	8.78	14.9	-0.35
LA09	5 × 15	6.2	14.39	20.69	24.26	36.97	-0.11	9.77	28.15	-0.11
LA10	5 × 15	8.04	18.16	7.72	18.16	7.83	0	9.5	36.95	0
LA11	5 × 20	0.57	21.77	29.79	20.05	15.88	0	20.54	18.33	0

Table A2. Cont.

Instance	Size	Methods								
		FIFO	LIFO	LOR	LPT	LTPT	MOR	SPT	STPT	SPP-Transformer
LA12	5 × 20	0	27.91	24.64	19.35	8.37	0	15.78	30.7	0
LA13	5 × 20	0.78	31.97	14.68	6.86	8.77	−0.09	10.77	10.08	−0.09
LA14	5 × 20	2.24	16.49	21.28	10.99	9.83	0	10.45	27.4	0
LA15	5 × 20	10.1	13.7	19.61	20.66	4.34	7.49	0.22	22.16	−1.12
LA16	10 × 10	17.06	17.87	0.99	10.92	5.14	0	4.33	14.44	−3.07
LA17	10 × 10	7.58	16.94	12.32	28.2	17.65	0	9.48	32.7	−3.79
LA18	10 × 10	12.21	25.05	22.51	18.26	27.18	0	4.14	17.94	0
LA19	10 × 10	11.55	4.2	5.46	11.55	5.46	14.29	−1.26	3.05	−1.79
LA20	10 × 10	21.15	6.43	17.64	23.98	5.85	10.14	−2.53	17.06	−4.58
Average		12.94	19.8	20.68	22.51	16.25	7.06	11.48	22.52	1.45

Table A3. The comparison between composite dispatching rules and SPP-Transformer on makespan.

Instance	Size	Opt	Methods				
			LOR+SPT	MOR+LPT	FCFS*S	SI/Q	SPP-Transformer
FT06	6 × 6	59	85	68	78	84	59
FT10	10 × 10	1002	1100	1284	1176	1114	1180
FT20	5 × 20	1205	1280	1616	1358	1279	1231
LA01	5 × 10	666	761	818	869	775	699
LA02	5 × 10	684	796	952	773	778	716
LA03	5 × 10	615	745	807	770	711	651
LA04	5 × 10	615	703	865	722	698	670
LA05	5 × 10	593	620	668	623	623	593
LA06	5 × 15	926	1182	1122	1022	1171	926
LA07	5 × 15	920	1021	1087	1082	1023	956
LA08	5 × 15	866	959	1086	902	958	863
LA09	5 × 15	952	1122	1178	1015	1060	951
LA10	5 × 15	958	1090	1116	1041	1077	958
LA11	5 × 20	1222	1494	1471	1318	1435	1222
LA12	5 × 20	1039	1214	1245	1168	1227	1039
LA13	5 × 20	1151	1338	1278	1234	1289	1150
LA14	5 × 20	1292	1518	1405	1426	1463	1292
LA15	5 × 20	1336	1356	1596	1429	1366	1321
LA16	10 × 10	1108	1255	1219	1144	1181	1074
LA17	10 × 10	844	956	1074	1001	944	812
LA18	10 × 10	942	998	1099	1029	998	942
LA19	10 × 10	952	950	1068	963	957	935
LA20	10 × 10	1026	1022	1210	1223	1023	979
Average		912	1025	1101	1016	1010	923

Table A4. The comparison between composite dispatching rules and SPP-Transformer on RPD (%).

Instance	Size	Methods				
		LOR+SPT	MOR+LPT	FCFS*S	SI/Q	SPP-Transformer
FT06	6 × 6	44.07	15.25	32.2	42.37	0
FT10	10 × 10	9.78	28.14	17.37	11.18	17.76
FT20	5 × 20	6.22	34.11	12.7	6.14	2.16
LA11	5 × 10	14.26	22.82	30.48	16.37	4.95
LA12	5 × 10	16.37	39.18	13.01	13.74	4.68
LA13	5 × 10	21.14	31.22	25.2	15.61	5.85
LA14	5 × 10	14.31	40.65	17.4	13.5	8.94

Table A4. Cont.

Instance	Size	Methods				
		LOR+SPT	MOR+LPT	FCFS*S	SI/Q	SPP-Transformer
LA15	5 × 10	4.55	12.65	5.06	5.06	0
LA16	5 × 15	27.65	21.17	10.37	26.46	0
LA17	5 × 15	10.98	18.15	17.61	11.2	3.91
LA18	5 × 15	10.74	25.4	4.16	10.62	−0.35
LA19	5 × 15	17.86	23.74	6.62	11.34	−0.11
LA10	5 × 15	13.78	16.49	8.66	12.42	0
LA11	5 × 20	22.26	20.38	7.86	17.43	0
LA12	5 × 20	16.84	19.83	12.42	18.09	0
LA13	5 × 20	16.25	11.03	7.21	11.99	−0.09
LA14	5 × 20	17.49	8.75	10.37	13.24	0
LA15	5 × 20	1.5	19.46	6.96	2.25	−1.12
LA16	10 × 10	13.27	10.02	3.25	6.59	−3.07
LA17	10 × 10	13.27	27.25	18.6	11.85	−3.79
LA18	10 × 10	5.94	16.67	9.24	5.94	0
LA19	10 × 10	−0.21	12.18	1.16	0.53	−1.79
LA20	10 × 10	−0.39	17.93	19.2	−0.29	−4.58
Average		13.82	21.41	12.92	12.33	1.45

Table A5. The comparison between meta-heuristic methods and SPP-Transformer on makespan.

Instance	Size	Opt	Methods			
			GA	DE	GWO	SPP-Transformer
FT06	6 × 6	59	58	60	59	59
FT10	10 × 10	1002	1331	1312	1219	1180
FT20	5 × 20	1205	1634	1635	1564	1231
LA01	5 × 10	666	782	769	727	699
LA02	5 × 10	684	821	805	768	716
LA03	5 × 10	615	737	731	696	651
LA04	5 × 10	615	736	728	681	670
LA05	5 × 10	593	633	616	599	593
LA06	5 × 15	926	1040	1025	982	926
LA07	5 × 15	920	1085	1068	1014	956
LA08	5 × 15	866	1045	1024	974	863
LA09	5 × 15	952	1094	1063	1021	951
LA10	5 × 15	958	1048	1020	985	958
LA11	5 × 20	1222	1400	1371	1330	1222
LA12	5 × 20	1039	1195	1182	1133	1039
LA13	5 × 20	1151	1335	1312	1261	1150
LA14	5 × 20	1292	1377	1352	1321	1292
LA15	5 × 20	1336	1527	1498	1449	1321
LA16	10 × 10	1108	1238	1206	1138	1074
LA17	10 × 10	844	1072	1032	947	812
LA18	10 × 10	942	1149	1109	1040	942
LA19	10 × 10	952	1182	1142	1066	935
LA20	10 × 10	1026	1235	1185	1118	979
Average		912	1076	1054	1004	923

Table A6. The comparison between meta-heuristic methods and SPP-Transformer on RPD (%).

Instance	Size	Methods			
		GA	DE	GWO	SPP-Transformer
FT06	6 × 6	−1.69	1.69	0	0
FT10	10 × 10	32.83	30.94	21.66	17.76
FT20	5 × 20	35.6	35.68	29.79	2.16
LA01	5 × 10	17.42	15.47	9.16	4.95
LA02	5 × 10	20.03	17.69	12.28	4.68
LA03	5 × 10	19.84	18.86	13.17	5.85
LA04	5 × 10	19.67	18.37	10.73	8.94
LA05	5 × 10	6.75	3.88	1.01	0
LA06	5 × 15	12.31	10.69	6.05	0
LA07	5 × 15	17.93	16.09	10.22	3.91
LA08	5 × 15	20.67	18.24	12.47	−0.35
LA09	5 × 15	14.92	11.66	7.25	−0.11
LA10	5 × 15	9.39	6.47	2.82	0
LA11	5 × 20	14.57	12.19	8.84	0
LA12	5 × 20	15.01	13.76	9.05	0
LA13	5 × 20	15.99	13.99	9.56	−0.09
LA14	5 × 20	6.58	4.64	2.24	0
LA15	5 × 20	14.3	12.13	8.46	−1.12
LA16	10 × 10	11.73	8.84	2.71	−3.07
LA17	10 × 10	27.01	22.27	12.2	−3.79
LA18	10 × 10	21.97	17.73	10.4	0
LA19	10 × 10	24.16	19.96	11.97	−1.79
LA20	10 × 10	20.37	15.5	8.97	−4.58
Average		17.28	15.08	9.61	1.45

Table A7. The comparison between RL methods and SPP-Transformer on makespan.

Instance	Size	Opt	Methods			
			DQN	Rainbow	PPO	SPP-Transformer
FT06	6 × 6	59	65	63	67	59
FT10	10 × 10	1002	1289	1235	1341	1180
FT20	5 × 20	1205	1463	1378	1479	1231
LA01	5 × 10	666	785	935	759	699
LA02	5 × 10	684	809	804	851	716
LA03	5 × 10	615	750	742	713	651
LA04	5 × 10	615	684	812	779	670
LA05	5 × 10	593	592	660	634	593
LA06	5 × 15	926	984	1066	963	926
LA07	5 × 15	920	1035	991	1034	956
LA08	5 × 15	866	1014	964	1125	863
LA09	5 × 15	952	1010	1121	987	951
LA10	5 × 15	958	978	1045	1021	958
LA11	5 × 20	1222	1283	1480	1314	1222
LA12	5 × 20	1039	1123	1204	1121	1039
LA13	5 × 20	1151	1198	1300	1243	1150
LA14	5 × 20	1292	1283	1290	1324	1292
LA15	5 × 20	1336	1390	1393	1516	1321
LA16	10 × 10	1108	1102	1183	1179	1074
LA17	10 × 10	844	894	900	987	812
LA18	10 × 10	942	993	1147	1123	942
LA19	10 × 10	952	963	958	1146	935
LA20	10 × 10	1026	1047	1183	1178	979
Average		912	988	1037	1038	923

Table A8. The comparison between RL methods and SPP-Transformer on RPD (%).

Instance	Size	Methods			
		DQN	Rainbow	PPO	SPP-Transformer
FT06	6 × 6	10.17	6.78	13.56	0
FT10	10 × 10	28.64	23.25	33.83	17.76
FT20	5 × 20	21.41	14.36	22.74	2.16
LA01	5 × 10	17.87	40.39	13.96	4.95
LA02	5 × 10	18.27	17.54	24.42	4.68
LA03	5 × 10	21.95	20.65	15.93	5.85
LA04	5 × 10	11.22	32.03	26.67	8.94
LA05	5 × 10	-0.17	11.3	6.91	0
LA06	5 × 15	6.26	15.12	4	0
LA07	5 × 15	12.5	7.72	12.39	3.91
LA08	5 × 15	17.09	11.32	29.91	-0.35
LA09	5 × 15	6.09	17.75	3.68	-0.11
LA10	5 × 15	2.09	9.08	6.58	0
LA11	5 × 20	4.99	21.11	7.53	0
LA12	5 × 20	8.08	15.88	7.89	0
LA13	5 × 20	4.08	12.95	7.99	-0.09
LA14	5 × 20	-0.7	-0.15	2.48	0
LA15	5 × 20	4.04	4.27	13.47	-1.12
LA16	10 × 10	-0.54	6.77	6.41	-3.07
LA17	10 × 10	5.92	6.64	16.94	-3.79
LA18	10 × 10	5.41	21.76	19.21	0
LA19	10 × 10	1.16	0.63	20.38	-1.79
LA20	10 × 10	2.05	15.3	14.81	-4.58
Average		9.04	14.45	14.42	1.45

Appendix B

Table A9. The comparison between different SPP-Transformers on makespan.

Instance	Size	Opt	Methods		
			Ours-1000	Ours-10000	Ours-100000
FT06	6 × 6	59	67	59	59
FT10	10 × 10	1002	1347	1206	1180
FT20	5 × 20	1205	1414	1313	1231
LA01	5 × 10	666	787	706	699
LA02	5 × 10	684	755	741	716
LA03	5 × 10	615	688	699	651
LA04	5 × 10	615	688	690	670
LA05	5 × 10	593	602	597	593
LA06	5 × 15	926	988	1005	926
LA07	5 × 15	920	975	978	956
LA08	5 × 15	866	937	883	863
LA09	5 × 15	952	1007	951	951
LA10	5 × 15	958	1033	956	958
LA11	5 × 20	1222	1317	1425	1222
LA12	5 × 20	1039	1180	1055	1039
LA13	5 × 20	1151	1217	1207	1150
LA14	5 × 20	1292	1296	1292	1292
LA15	5 × 20	1336	1374	1355	1321
LA16	10 × 10	1108	1131	1127	1074
LA17	10 × 10	844	904	904	812
LA18	10 × 10	942	996	984	942
LA19	10 × 10	952	969	963	935
LA20	10 × 10	1026	1020	1015	979
Average		912	988	1037	923

Table A10. The comparison between different SPP-Transformers on RPD (%).

Instance	Size	Methods		
		Ours-1000	Ours-10000	Ours-100000
FT06	6 × 6	13.56	0	0
FT10	10 × 10	34.43	20.36	17.76
FT20	5 × 20	17.34	8.96	2.16
LA01	5 × 10	18.17	6.01	4.95
LA02	5 × 10	10.38	8.33	4.68
LA03	5 × 10	11.87	13.66	5.85
LA04	5 × 10	11.87	12.2	8.94
LA05	5 × 10	1.52	0.67	0
LA06	5 × 15	6.7	8.53	0
LA07	5 × 15	5.98	6.3	3.91
LA08	5 × 15	8.2	1.96	−0.35
LA09	5 × 15	5.78	−0.11	−0.11
LA10	5 × 15	7.83	−0.21	0
LA11	5 × 20	7.77	16.61	0
LA12	5 × 20	13.57	1.54	0
LA13	5 × 20	5.73	4.87	−0.09
LA14	5 × 20	0.31	0	0
LA15	5 × 20	2.84	1.42	−1.12
LA16	10 × 10	2.08	1.71	−3.07
LA17	10 × 10	7.11	7.11	−3.79
LA18	10 × 10	5.73	4.46	0
LA19	10 × 10	1.79	1.16	−1.79
LA20	10 × 10	−0.58	−1.07	−4.58
Average		8.69	5.41	1.45

References

- Ma, J.; Chen, H.; Zhang, Y.; Guo, H.; Ren, Y.; Mo, R.; Liu, L. A digital twin-driven production management system for production workshop. *Int. J. Adv. Manuf. Technol.* **2020**, *110*, 1385–1397. [\[CrossRef\]](#)
- Guo, H.; Zhu, Y.; Zhang, Y.; Ren, Y.; Chen, M.; Zhang, R. A digital twin-based layout optimization method for discrete manufacturing workshop. *Int. J. Adv. Manuf. Technol.* **2021**, *112*, 1307–1318. [\[CrossRef\]](#)
- Garey, M.R.; Johnson, D.S.; Sethi, R. The complexity of flowshop and jobshop scheduling. *Math. Oper. Res.* **1976**, *1*, 117–129. [\[CrossRef\]](#)
- Renke, L.; Piplani, R.; Toro, C. A Review of Dynamic Scheduling: Context, Techniques and Prospects. *J. Intell. Syst. Ref. Libr. Implement. Ind.* **2021**, *4*, 229–258. [\[CrossRef\]](#)
- Tay, J.C.; Ho, N.B. Evolving dispatching rules using genetic programming for solving multi-objective flexible job-shop problems. *Comput. Ind. Eng.* **2008**, *54*, 453–473. [\[CrossRef\]](#)
- Papakostas, N.; Chryssolouris, G. A scheduling policy for improving tardiness performance. *Asian Int. J. Sci. Technol.* **2009**, *2*, 79–89.
- Cheng, R.; Gen, M.; Tsujimura, Y. A tutorial survey of job-shop scheduling problems using genetic algorithms—I. Representation. *Comput. Ind. Eng.* **1996**, *30*, 983–997. [\[CrossRef\]](#)
- Wong, K.P.; Dong, Z.Y. Differential evolution, an alternative approach to evolutionary algorithm. In Proceedings of the 13th International Conference on Intelligent Systems Application to Power Systems, Arlington, VA, USA, 6–10 November 2005; pp. 73–83.
- Mirjalili, S.; Mirjalili, S.M.; Lewis, A. Grey wolf optimizer. *Adv. Eng. Softw.* **2014**, *69*, 46–61. [\[CrossRef\]](#)
- Zhou, R.; Nee, A.; Lee, H. Performance of an ant colony optimisation algorithm in dynamic job shop scheduling problems. *Int. J. Prod. Res.* **2009**, *47*, 2903–2920. [\[CrossRef\]](#)
- Wang, Z.; Zhang, J.; Yang, S. An improved particle swarm optimization algorithm for dynamic job shop scheduling problems with random job arrivals. *Swarm Evol. Comput.* **2019**, *51*, 100594. [\[CrossRef\]](#)
- Mazyavkina, N.; Sviridov, S.; Ivanov, S.; Burnaev, E. Reinforcement learning for combinatorial optimization: A survey. *Comput. Oper. Res.* **2021**, *134*, 105400. [\[CrossRef\]](#)
- Wang, F.Y.; Zhang, J.J.; Zheng, X.; Wang, X.; Yuan, Y.; Dai, X.; Zhang, J.; Yang, L. Where does AlphaGo go: From church-turing thesis to AlphaGo thesis and beyond. *IEEE/CAA J. Autom. Sin.* **2016**, *3*, 113–120. [\[CrossRef\]](#)
- Jumper, J.; Evans, R.; Pritzel, A.; Green, T.; Figurnov, M.; Ronneberger, O.; Tunyasuvunakool, K.; Bates, R.; Žídek, A.; Potapenko, A.; et al. Highly accurate protein structure prediction with AlphaFold. *Nature* **2021**, *596*, 583–589. [\[CrossRef\]](#) [\[PubMed\]](#)

15. Rinciog, A.; Meyer, A. Towards Standardizing Reinforcement Learning Approaches for Stochastic Production Scheduling. *arXiv* **2021**, arXiv:2104.08196. <https://doi.org/10.48550/arXiv.2104.08196>.
16. Zhou, T.; Tang, D.; Zhu, H.; Wang, L. Reinforcement learning with composite rewards for production scheduling in a smart factory. *IEEE Access* **2020**, *9*, 752–766. [[CrossRef](#)]
17. Turgut, Y.; Bozdog, C.E. Deep Q-network model for dynamic job shop scheduling problem based on discrete event simulation. In Proceedings of the 2020 Winter Simulation Conference (WSC), Orlando, FL, USA, 14–18 December 2020; pp. 1551–1559.
18. Wang, L.; Hu, X.; Wang, Y.; Xu, S.; Ma, S.; Yang, K.; Liu, Z.; Wang, W. Dynamic job-shop scheduling in smart manufacturing using deep reinforcement learning. *Comput. Netw.* **2021**, *190*, 107969. [[CrossRef](#)]
19. Liu, C.L.; Chang, C.C.; Tseng, C.J. Actor-critic deep reinforcement learning for solving job shop scheduling problems. *IEEE Access* **2020**, *8*, 71752–71762. [[CrossRef](#)]
20. Ghosh, D.; Rahme, J.; Kumar, A.; Zhang, A.; Adams, R.P.; Levine, S. Why Generalization in RL is Difficult: Epistemic POMDPs and Implicit Partial Observability. In Proceedings of the Advances in Neural Information Processing Systems, Virtual, 6–14 December 2021; pp. 25502–25515.
21. Weckman, G.R.; Ganduri, C.V.; Koonce, D.A. A neural network job-shop scheduler. *J. Intell. Manuf.* **2008**, *19*, 191–201. [[CrossRef](#)]
22. Sim, M.H.; Low, M.Y.H.; Chong, C.S.; Shakeri, M. Job Shop Scheduling Problem Neural Network Solver with Dispatching Rules. In Proceedings of the 2020 IEEE International Conference on Industrial Engineering and Engineering Management (IEEM), Singapore, 14–17 December 2020; pp. 514–518.
23. Zang, Z.; Wang, W.; Song, Y.; Lu, L.; Li, W.; Wang, Y.; Zhao, Y. Hybrid deep neural network scheduler for job-shop problem based on convolution two-dimensional transformation. *Comput. Intell. Neurosci.* **2019**, *2019*, 7172842. [[CrossRef](#)]
24. Tian, W.; Zhang, H. A dynamic job-shop scheduling model based on deep learning. *Adv. Prod. Eng. Manag.* **2021**, *16*, 23–36. [[CrossRef](#)]
25. Shao, X.; Kim, C.S. Self-Supervised Long-Short Term Memory Network for Solving Complex Job Shop Scheduling Problem. *KSII Trans. Internet Inf. Syst. (TIIS)* **2021**, *15*, 2993–3010. [[CrossRef](#)]
26. Morariu, C.; Borangiu, T. Time series forecasting for dynamic scheduling of manufacturing processes. In Proceedings of the 2018 IEEE International Conference on Automation, Quality and Testing, Robotics (AQTR), Cluj-Napoca, Romania, 24–26 May 2018; pp. 1–6.
27. Zhou, H.; Zhang, S.; Peng, J.; Zhang, S.; Li, J.; Xiong, H.; Zhang, W. Informer: Beyond Efficient Transformer for Long Sequence Time-Series Forecasting. In Proceedings of the AAAI Conference on Artificial Intelligence, Vancouver, BC, Canada, 2–9 February 2021; pp. 11106–11115.
28. Magalhães, R.; Martins, M.; Vieira, S.; Santos, F.; Sousa, J. Encoder-Decoder Neural Network Architecture for solving Job Shop Scheduling Problems using Reinforcement Learning. In Proceedings of the 2021 IEEE Symposium Series on Computational Intelligence (SSCI), Orlando, FL, USA, 5–7 December 2021; pp. 1–8.
29. Yang, S. Using Attention Mechanism to Solve Job Shop Scheduling Problem. In Proceedings of the 2022 2nd International Conference on Consumer Electronics and Computer Engineering (ICCECE), Guangzhou, China, 14–16 January 2022; pp. 59–62.
30. Chen, R.; Li, W.; Yang, H. A Deep Reinforcement Learning Framework Based on an Attention Mechanism and Disjunctive Graph Embedding for the Job Shop Scheduling Problem. *IEEE Trans. Ind. Inform.* **2022**, *1*. [[CrossRef](#)]
31. Shakhlevich, N.; Sotskov, Y.N.; Werner, F. Adaptive scheduling algorithm based on mixed graph model. *IEE Proc.-Control Theory Appl.* **1996**, *43*, 9–16. [[CrossRef](#)]
32. Gholami, O.; Sotskov, Y.N. Solving parallel machines job-shop scheduling problems by an adaptive algorithm. *Int. J. Prod. Res.* **2014**, *52*, 3888–3904. [[CrossRef](#)]
33. He, K.; Zhang, X.; Ren, S.; Sun, J. Spatial pyramid pooling in deep convolutional networks for visual recognition. *IEEE Trans. Pattern Anal. Mach. Intell.* **2015**, *37*, 1904–1916. [[CrossRef](#)]
34. Wang, Z.; Zhang, J.; Si, J. Dynamic job shop scheduling problem with new job arrivals: A survey. In Proceedings of the Chinese Intelligent Automation Conference, Zhenjiang, China, 20–22 September 2019; pp. 664–671.
35. Błażewicz, J.; Pesch, E.; Sterna, M. The disjunctive graph machine representation of the job shop scheduling problem. *Eur. J. Oper. Res.* **2000**, *127*, 317–331. [[CrossRef](#)]
36. Zeng, Y.; Liao, Z.; Dai, Y.; Wang, R.; Li, X.; Yuan, B. Hybrid intelligence for dynamic job-shop scheduling with deep reinforcement learning and attention mechanism. *arXiv* **2022**, arXiv:2201.00548. <https://doi.org/10.48550/arXiv.2201.00548>.
37. Chen, B.; Matis, T.I. A flexible dispatching rule for minimizing tardiness in job shop scheduling. *Int. J. Prod. Econ.* **2013**, *141*, 360–365. [[CrossRef](#)]
38. He, K.; Zhang, X.; Ren, S.; Sun, J. Deep residual learning for image recognition. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Las Vegas, NV, USA, 27–30 June 2016; pp. 770–778.
39. Ba, J.L.; Kiros, J.R.; Hinton, G.E. Layer normalization. *arXiv* **2016**, arXiv:1607.06450. <https://doi.org/10.48550/arXiv.1607.06450>.
40. Wang, Y.; Mohamed, A.; Le, D.; Liu, C.; Xiao, A.; Mahadeokar, J.; Huang, H.; Tjandra, A.; Zhang, X.; Zhang, F.; et al. Transformer-based acoustic modeling for hybrid speech recognition. In Proceedings of the ICASSP 2020—2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), Barcelona, Spain, 4–8 May 2020; pp. 6874–6878.
41. Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A.N.; Kaiser, L.u.; Polosukhin, I. Attention is All you Need. In Proceedings of the Advances in Neural Information Processing Systems, Long Beach, CA, USA, 4–9 December 2017; pp. 5998–6008.

42. Beasley, J.E. OR-Library: distributing test problems by electronic mail. *J. Oper. Res. Soc.* **1990**, *41*, 1069–1072. [[CrossRef](#)]
43. Fisher, H. *Probabilistic Learning Combinations of Local Job-Shop Scheduling Rules*; Prentice-Hall: Hoboken, NJ, USA, 1963; pp. 225–251.
44. Lawrence, S. *Resource Constrained Project Scheduling: An Experimental Investigation of Heuristic Scheduling Techniques (Supplement)*; Graduate School of Industrial Administration, Carnegie-Mellon University: Pittsburgh, PA, USA, 1984.
45. Available online: <https://github.com/Yunhui1998/TOFA> (accessed on 27 April 2022).
46. Hessel, M.; Modayil, J.; Van Hasselt, H.; Schaul, T.; Ostrovski, G.; Dabney, W.; Horgan, D.; Piot, B.; Azar, M.; Silver, D. Rainbow: Combining improvements in deep reinforcement learning. In Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, New Orleans, LA, USA, 2–7 February 2018; pp. 3215–3222.
47. Schulman, J.; Wolski, F.; Dhariwal, P.; Radford, A.; Klimov, O. Proximal policy optimization algorithms. *arXiv* **2017**, arXiv:1707.06347. <https://doi.org/10.48550/arXiv.1707.06347>.
48. Zhang, W.; Wen, J.; Zhu, Y.; Hu, Y. Multi-objective scheduling simulation of flexible job-shop based on multi-population genetic algorithm. *Int. J. Simul. Model.* **2017**, *16*, 313–321. [[CrossRef](#)]
49. Dosovitskiy, A.; Beyer, L.; Kolesnikov, A.; Weissenborn, D.; Zhai, X.; Unterthiner, T.; Dehghani, M.; Minderer, M.; Heigold, G.; Gelly, S.; et al. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv* **2020**, arXiv:2010.11929. <https://doi.org/10.48550/arXiv.2010.11929>.