applied sciences

Article

Solving Panel Block Assembly Line Scheduling Problem via a Novel Deep Reinforcement Learning Approach

Tao Zhou, Liang Luo *, Yuanxin He, Zhiwei Fan and Shengchen Ji

Ministry of Education Key Laboratory of High Performance Ship Technology, Wuhan University of Technology, Wuhan 420100, China; a15630455659@163.com (T.Z.); 305364@whut.edu.cn (Y.H.); fanzhiwei@whut.edu.cn (Z.F.); jsc@whut.edu.cn (S.J.)

* Correspondence: luoliang@whut.edu.cn

Abstract: The panel block is a quite important "intermediate product" in the shipbuilding process. However, the assembly efficiency of the panel block assembly line is not high. Therefore, rational scheduling optimization is of great significance for improving shipbuilding efficiency. Currently, the processing sequence of the panel blocks in the panel block assembly line is mainly determined using heuristic and metaheuristic algorithms. However, these algorithms have limitations, such as small problem-solving capacity and low computational efficiency. To address these issues, this study proposes an end-to-end approach based on deep reinforcement learning to solve the scheduling problem of the ship's panel block assembly line. First, a Markov decision model is established, and a disjunctive graph is creatively used to represent the current scheduling status of the panel block assembly line. Then, a policy function based on a graph isomorphism network is designed to extract information from the disjunctive graph's state and train it using Proximal Policy Optimization algorithms. To validate the effectiveness of our method, tests on both real shipbuilding data and publicly available benchmark datasets are conducted. We compared our proposed end-to-end deep reinforcement learning algorithm with heuristic algorithms, metaheuristic algorithms, and the unimproved reinforcement learning algorithm. The experimental results demonstrate that our algorithm outperforms other baseline methods in terms of model performance and computation time. Moreover, our model exhibits strong generalization capabilities for larger instances.

Keywords: panel block assembly line; deep reinforcement learning; disjunctive graph; graph isomorphism network

check for updates

Citation: Zhou, T.; Luo, L.; He, Y.; Fan, Z.; Ji, S. Solving Panel Block Assembly Line Scheduling Problem via a Novel Deep Reinforcement Learning Approach. *Appl. Sci.* **2023**, *13*, 8483. https://doi.org/10.3390/app13148483

Academic Editors: Wendong Xiao, Jin Guo and Wei Su

Received: 2 July 2023
Revised: 16 July 2023
Accepted: 19 July 2023
Published: 22 July 2023

1. Introduction

In recent years, with the continuous maturation of artificial intelligence technology, machine learning has provided new approaches for solving scheduling problems in complex and uncertain environments. Moreover, integrating machine learning algorithms with job scheduling problems aligns more closely with the core principles of intelligent manufacturing. Particularly, with the ongoing development of deep reinforcement learning (DRL) [1,2], intelligent agents learn optimal scheduling strategies through interaction with the environment guided by rewards. This approach has been widely applied to solve workshop scheduling problems. The panel block assembly line in shipbuilding is a specialized workshop, but its processing process will encounter bottlenecks such as block congestion [3]. To address these issues, we propose a DRL-based method to tackle the scheduling problem of the ship's panel block assembly line.

The structural composition of a vessel primarily consists of various profiles and steel materials. The hull of a ship is typically streamlined, but near the midship, the shape tends to become flat [4]. In modern shipbuilding practices, the entire structure of a ship is generally divided into several small sections, referred to as blocks. These individual blocks are then combined to form larger sections or ring sections, which are further assembled to create the complete structure of the vessel. A block serves as the most fundamental intermediate unit in ship structure manufacturing [5]. After the hull is divided into blocks, based on their internal structural characteristics, blocks with complex structures and large outer panel curvatures are referred to as curved blocks, while those with flat or nearly flat profiles are known as panel blocks, as depicted in Figure 1 [6]. For conventional vessels like bulk carriers and tankers, panel blocks can account for more than 60% of the total number of hull blocks. In some cases, the proportion of panel blocks can even reach around 80% for large and ultra-large oil tankers [7]. Furthermore, with the development of larger ships, such as large bulk carriers, oil tankers, and container ships, which commonly present the characteristics of longer midship, the demand for panel blocks has significantly increased.

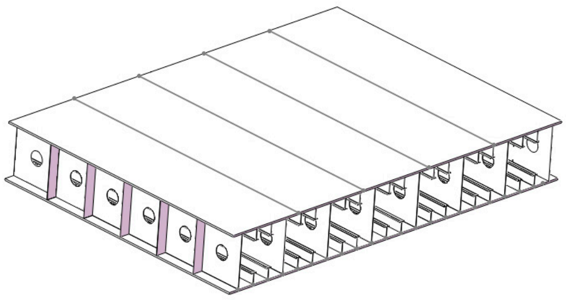


Figure 1. Panel block.

Due to the extensive processing required for a large number of panel blocks in the shipbuilding process, the manufacturing and assembly of panel blocks represent the primary bottleneck in the entire shipbuilding process. Furthermore, shipbuilding is an order-based industry, where the panel blocks of the ship for each order have different shapes and volumes, further increasing the complexity of scheduling for the panel block assembly line [8,9]. As a result, the assembly process of the panel block assembly line possesses a high degree of variability and system complexity [10].

However, the production efficiency of most panel block assembly lines is not high [11]. This is primarily due to the continued use of traditional on-site scheduling methods, which make it difficult to obtain optimized scheduling solutions. For instance, some shipyards directly prioritize the production of simple blocks before complex ones without employing means of optimized scheduling to reduce labor hours. Consequently, optimizing the production plan of the panel block assembly line to enhance the overall shipbuilding process efficiency is of paramount importance [12–15].

In light of these considerations, this paper proposes an end-to-end DRL approach to address the scheduling problem of panel block assembly lines in shipbuilding. Initially, we introduce a scheduling model based on the Markov Decision Process (MDP) [16] and creatively employ a disjunctive graph to represent the scheduling process of panel block assembly lines, thereby comprehensively and logically capturing the current state. This representation effectively integrates the dependencies between operations and the status of each workstation in the panel block assembly line, providing crucial information for generating optimal scheduling decisions. Subsequently, we utilize a Graph Isomorphism Network (GIN) to encode and embed the nodes in the disjunctive graph, enabling efficient computation of the policy network. Based on this approach, we design a policy network capable of handling instances of panel block assembly line scheduling of any size, effectively facilitating the generalization from model training to model deployment without the need for retraining. Finally, we employ the Proximal Policy Optimization (PPO) algorithm [17] to train our policy network. Extensive experiments are conducted using real shipyard data and publicly available benchmark datasets, demonstrating that our proposed method outperforms existing heuristic algorithms, metaheuristic algorithms, and reinforcement learning algorithms in terms of algorithmic performance and computation time. Furthermore, our method exhibits remarkable generalization capabilities when applied to larger-scale instances. The main contributions of this paper are as follows:

(1) We introduce an end-to-end reinforcement learning approach to learn scheduling rules, overcoming limitations such as poor model generalization. This method can effectively solve instances of any scale without the need for retraining;
(2) We present an MDP model for the panel block assembly line scheduling problem, providing a comprehensive definition of states, actions, and rewards within this MDP framework. The algorithms utilized for model training are also elaborated upon;
(3) We propose a graph embedding method that employs disjunctive graphs to represent the state information of the panel block assembly line. This approach directly extracts scheduling features from the disjunctive graph, marking the first instance of combining DRL with disjunctive graphs to address the scheduling problem in shipbuilding's panel block assembly lines.

The remaining sections of the paper are described as follows: Section 2 provides a comprehensive summary of the current research status in the relevant field. Section 3 presents the mathematical model for the scheduling problem and describes the background information on the technologies related to our research. Section 4 elucidates our research methodology, including the establishment of the MDP model, parameterized policy network, and the training process of the algorithm. Section 5 presents the experimental procedure and discusses the results obtained. Finally, in Section 6, we present the conclusions and outline future work for our research.

2. Literature Review

The scheduling process among the workstations in the panel block assembly line is modeled in this paper as a permutation flow shop scheduling problem with the objective of minimizing the maximum completion time [18]. Sriskandarajah and Hall [19] have proven the NP-hardness of such problems when the number of processing machines exceeds two ($m > 2$). Currently, heuristic algorithms, metaheuristic algorithms [20,21], and reinforcement learning algorithms are the mainstream research approaches for solving this type of problem. The following is a summary of the current research for each method.

2.1. Solving Scheduling Problem via Heuristics

Heuristic algorithms offer advantages such as high solution efficiency and fast computation speed. The Johnson algorithm [22] was the first constructed heuristic algorithm, which can be applied to the two-machine flow shop problem and yield an optimal solution. The Palmer algorithm [23] and the Gupta algorithm [24] employ the construction of processing time slopes to solve the permutation flow shop problem (PFSP). This approach involves converting the slopes of the jobs into function values, sorting them based on increasing or decreasing rules and arranging the job sequence accordingly. The NEH algorithm [25], considered one of the most efficient heuristic algorithms, follows the fundamental idea of prioritizing and allocating jobs based on their total processing time. It then achieves a complete schedule through consecutive job insertions. The priorities of the jobs and the insertion construction method are the two key aspects of the NEH algorithm. Framinan et al. [26] proposed three stages in the development of heuristic algorithms, namely index development, solution construction, and solution improvement. Through experiments, they concluded that the priority assignment method of the NEH algorithm is most effective for the permutation flow shop scheduling problem with objectives of maximum completion time and machine time constraints.

2.2. Solving Scheduling Problem via Metaheuristics

Conventional heuristic methods can only address small-scale permutation flow shop scheduling problems. To enhance computational performance, numerous scholars have employed metaheuristic algorithms to tackle various large-scale scheduling problems. Yu (2015) [27] proposed a block-based evolutionary algorithm for solving PFSP. They designed association rules to extract excellent genes, increasing solution diversity and enabling the generation of various blocks for artificial chromosome combinations, thereby improving convergence efficiency. Qun et al. (2015) [28] introduced a hybrid backtracking search algorithm to solve PFSP, establishing a mathematical model with the objective of minimizing the completion time. They devised new crossover and mutation strategies to incorporate simulated annealing mechanisms into random insertion local search, preventing premature convergence to local optima. Korhan et al. (2016) [29] presented an improved iterative greedy algorithm for PFSP, aiming to minimize total tardiness. They combined this algorithm with random search techniques, further enhancing the quality of the solutions. Suansh Deb et al. (2018) [30] designed a metaheuristic algorithm based on rhinoceros natural behavior to minimize the *makespan* in PFSP. They simplified the search operations and reduced the number of operation parameters required in the mathematical model. Morais et al. (2022) [31] devised three optimization algorithms based on discrete differential evolution to solve the *makespan* minimization problem in PFSP. While metaheuristic algorithms outperform heuristic algorithms in terms of solution quality, they still suffer from drawbacks such as excessive computation time for large-scale problem instances.

2.3. Solving Scheduling Problem via Reinforcement Learning

Considering the fixed structure of heuristic algorithms and metaheuristic algorithms, the search performance is somewhat constrained. Many researchers have attempted to utilize reinforcement learning algorithms to solve scheduling problems. Liu et al. [32] employed a parallel-trained Actor–Critic neural network model to address job shop scheduling problems. The Actor network learns actions under different circumstances, while the Critic assists in evaluating the value of those actions and returns feedback to the Actor network. This algorithm achieved promising results in job shop scheduling problems. Waschneck et al. [33] applied the DQN algorithm to production scheduling, utilizing deep neural networks to train in a reinforcement learning environment with flexible user-defined objectives to optimize production scheduling problems. Lin et al. [34] proposed an MDQN algorithm to solve job shop scheduling problems in an intelligent factory based on edge computing frameworks, and simulation results demonstrated its superior performance compared to other methods. Park et al. [35] introduced a framework that combines graph neural networks with reinforcement learning, yielding excellent results in job shop scheduling problems. Yang et al. [36] employed DRL to investigate dynamic PFSP for implementing intelligent decision-making in dynamic scheduling scenarios, achieving superior results compared to heuristic algorithms. Moreover, the trained network can generate a scheduling action within an average of 2.13 ms. Pan et al. [37] presented a DRL model based on heterogeneous networks to solve PFSP, and experimental results demonstrated its remarkable performance in PFSP problem-solving.

2.4. Research Gap

Both heuristic algorithms and metaheuristic algorithms serve as effective means of obtaining feasible solutions to scheduling problems. However, as the population size increases, the computational complexity of these algorithms also grows, resulting in significantly longer solution times. Additionally, heuristic algorithms and metaheuristic algorithms require retraining when tackling problems of different scales, leading to reduced efficiency. Moreover, even a minor adjustment in a parameter within heuristic and metaheuristic algorithms can potentially impact the final results, making them more suitable for small-scale scheduling problems. On the other hand, reinforcement learning algorithms only require training once to address problems of all scales without the need for retraining. However, in the application of reinforcement learning algorithms, most rely on Deep Q-Networks (DQN) to approximate action-value functions, which cannot directly optimize policies. Furthermore, the representation of environmental states often relies on mathematical models, which may not fully capture the scheduling state. To address these issues and minimize the influence of intermediate processes on the computational results, we adopt an end-to-end DRL approach to solve the scheduling problem in the context of shipyard panel block assembly, with the objective of minimizing the maximum completion time.

3. Preliminaries

3.1. Description of Scheduling Problem

3.1.1. Symbolic Representation

The symbols used to establish the mathematical model for the scheduling problem in shipyard panel block assembly are presented in Table 1.

Table 1. Nomenclature for various parameters.

Notation	Description
n	The number of blocks
m	The number of workstations
B	The set of blocks
S	The set of workstations
i	The number of blocks in set B
j	The process number of block i
B_i	The i-th block
S_j	The j-th workstation
O_{ij}	The operation of block B_i on workstation S_j
$p_{i,j}$	The processing time of block B_i on workstation S_j
π	The processing sequence of blocks
C_{max}	The maximum completion time
$C(\pi_i, j)$	The completion time of block π_i on workstation S_j

3.1.2. Problem Description

The research focuses on the scheduling problem in shipbuilding panel block assembly, where n panel blocks $B = \{B_1, B_2, \ldots, B_n\}$ undergo m assembly processes $\{Q_{i1}, Q_{i2}, \ldots, Q_{im}\}$ in a flow production manner across m workstations $S = \{S_1, S_2, \ldots, S_m\}$. In this study, the assembly processes in the shipyard panel block assembly line are designed to consist of seven sequential stages, as illustrated in Figure 2, resulting in a total of seven workstations (m = 7) and seven assembly processes per block. All blocks are processed in the same order at each workstation, with the constraint that each block is processed only once at each workstation. The processing times required for each block at each workstation are known, and infinite buffers exist between workstations [38]. The objective is to find an optimal scheduling scheme with the goal of minimizing the maximum completion time. It is assumed that the blocks are processed in the order of workstation 1 to m, denoted as the block processing sequence $\pi = \{\pi_1, \pi_2, \ldots, \pi_n\}$. The mathematical formulation of the problem is described in Equations (1) and (2).

$$\begin{cases} C(\pi_1, 1) = p_{\pi_1,1} \\ C(\pi_i, 1) = C(\pi_{i-1}, 1) + p_{\pi_1,1} \\ C(\pi_i, j) = C(\pi_1, j-1) + p_{\pi_1,j} \\ C(\pi_i, j) = max\{C(\pi_{i-1}, j), C(\pi_i, j-1)\} + p_{\pi_i,j} \end{cases} \tag{1}$$

$$makespan = C_{max}(\pi) = C(\pi_n, m) \tag{2}$$

where $i = 2, \ldots,$ n; $j = 2, \ldots,$ n; *makespan* is the maximum time to complete.

3.2. Reinforcement Learning

Reinforcement learning [39] comprises essential components such as agents, environments, states, actions, and rewards. The interaction between the agent and the environment occurs through states, actions, and rewards. MDP forms the core of reinforcement learning, consisting of elements $(S,\ A,\ P,\ \gamma,\ R)$, as defined in Equation (3). It encompasses actions

$a \in A$, states $s \in S$, reward function $r = R(s, a)$, state transition probabilities $P(s\prime|s, a)$, and discount factor γ. In this paper, the notations a_t, s_t, and r_t are employed to represent the action, state, and reward at step t, respectively. The objective of reinforcement learning is to maximize the expected return by learning an optimal scheduling strategy.

$$M =< S, A, P, \gamma, R > \quad (3)$$

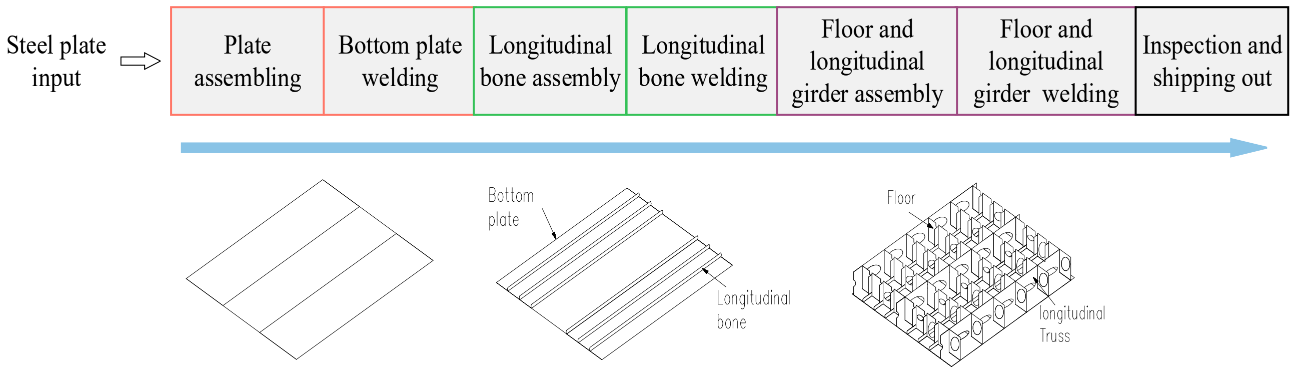

Steel plate input
Plate assembling
Bottom plate welding
Longitudinal bone assembly
Longitudinal bone welding
Floor and longitudinal girder assembly
Floor and longitudinal girder welding
Inspection and shipping out
Bottom plate
Longitudinal bone
Floor
longitudinal Truss

Figure 2. Ship panel block assembly process.

3.3. Disjunctive Graph

To express the scheduling state more comprehensively and logically, we utilize a disjunctive graph [40] to represent the scheduling process of the panel block assembly line. Let $N = \{O_{ij}|\forall i, j\} \cup \{S, T\}$ denote the set of all operation nodes, where S and T represent the virtual start and end nodes, respectively. Therefore, the disjunctive graph $G = (N, C, D)$ is a mixed graph with O as its vertex set. Here, C represents a set of conjunctive arcs (directed arcs) that depict the adjacent operation relationships determined by the process, usually denoted by solid lines. On the other hand, D represents a set of disjunctive arcs (undirected arcs), denoted by dashed lines, representing the disjunctive arcs between operations that can be processed on the same workstation. By determining the direction of each disjunctive arc, a solution for the planar segmented assembly line scheduling instance can be obtained, resulting in a directed acyclic graph (DAG) [41]. Figure 3a,b illustrate an example of a disjunctive graph and its solution for a panel block assembly line scheduling instance. In Figure 3a, which represents a scheduling instance with three blocks and three workstations, the black arrows depict the conjunctive arcs among operations within the same block, while the dashed lines represent the disjunctive arcs connecting operations that require the same workstation across different blocks. Figure 3b presents a complete solution to the scheduling problem, where each disjunctive arc has a direction, and each node is connected by at most two disjunctive arcs.

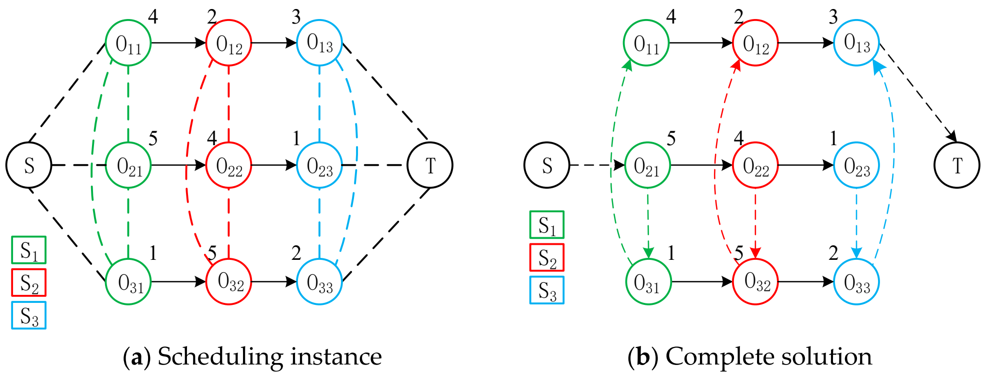
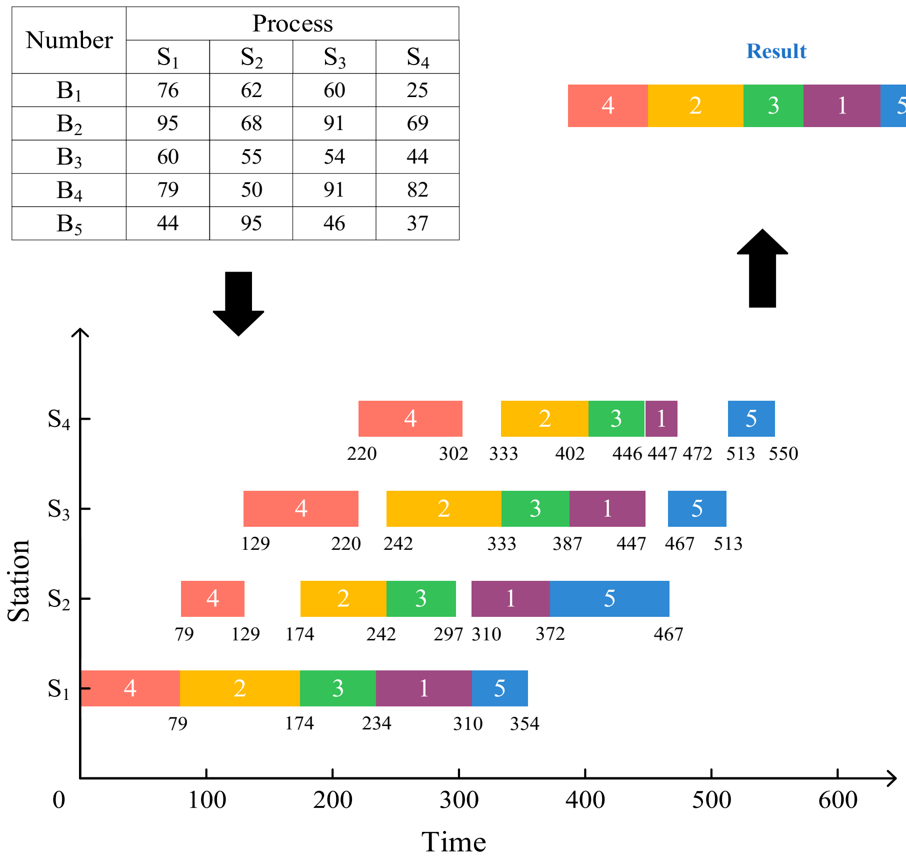


Figure 3. Exemplary disjunctive graph for the panel block assembly line scheduling problem and its solution.

To enhance the reader's understanding of the scheduling process for the panel block assembly line, we present an illustrative example in Figure 4. As depicted in the diagram, when the input instance consists of five blocks and four workstations, the Gantt chart output reveals that the optimal sequence for block input should follow the order of 4-2-3-1-5.



Number	Process			
	S_1	S_2	S_3	S_4
B_1	76	62	60	25
B_2	95	68	91	69
B_3	60	55	54	44
B_4	79	50	91	82
B_5	44	95	46	37

Figure 4. Scheduling instance of dimensions 5 × 4.

4. Methods

In this section, we present the fundamental principles of our approach. Firstly, we establish the MDP model for the scheduling problem. Subsequently, we devise a policy network based on GIN to address the task. Finally, we introduce the training algorithm employed and provide a comprehensive account of the training process.

4.1. MDP Model

The MDP model serves as a bridge between DRL and scheduling problems. Resolving the scheduling problem in panel block assembly can be viewed as determining the orientations of the disjunctive graphs. The underlying MDP model we establish is as follows:

State: Due to the limitations of existing methods in providing a comprehensive and rational representation of the current scheduling state in the planar segment assembly line and considering the inherent characteristics of the scheduling environment, we introduce the concept of disjunctive graphs for representation. Disjunctive graphs encompass all the information related to the scheduling environment, including the processing status of each workstation and the number of segments currently being assembled. The state consists of the processing status of each workstation and the blocks placed on the panel block assembly line. To capture the current scheduling state at step t, we utilize the disjunctive graph $G(t) = (N, C(t), D(t))$, where $C(t)$ comprises the set of all connecting arcs up to step t, and $D(t)$ represents the remaining disjunctive arcs in the graph. The initial state s_0 represents the initial state of the panel block assembly line before scheduling begins, while the terminal state s_t corresponds to a feasible solution when scheduling is complete, at which point $D(t) = \varnothing$, indicating that all disjunctive arcs have been assigned orientations. As the assembly process progresses, the disjunctive graph provides varying state descriptions of the current scheduling environment, and in turn, the changes in the environmental state lead to different disjunctive graphs.

Action: The actions of the intelligent agent involve selecting the block to be placed in the first workstation during the assembly process of the panel block assembly line. The design of the action space should thoroughly consider minimizing idle time across workstations and enhancing their utilization rate.

State transition: As the scheduling environment of the panel block assembly line constantly evolves, the scheduling state progresses from state s_t to the next state s_{t+1}. When determining the next action a_t to schedule, our first step is to identify the earliest feasible time to allocate a_t on the required workstation. Subsequently, we update the direction of the disjunctive arcs for the workstations based on the current temporal relationships, generating a new disjunctive graph as the new state s_{t+1}.

Reward: The rational definition of rewards serves as a prerequisite for successful learning in reinforcement learning. Rewards should be defined in direct relation to the objective of minimizing *makespan*. Therefore, we initially compute the difference between partial solutions at two consecutive steps, denoted as $U_t = C(s_{t+1}) - C(s_t)$, where $C(s_t)$ is defined as $C(s_t) = max_{ij}\{LB_t(O_{ij})\}$, representing a lower bound on the *makespan*. We assign the negative value of the difference U_t as the immediate reward for each step t, i.e., $R(a_t, s_t) = -U_t$. In other words, the cumulative reward corresponds to the negative makespan when all operations are scheduled.

Policy: For state s_t, the stochastic policy $\pi(a_t|s_t)$ generates a probability distribution over actions a_t, with action selection prioritized based on the probability distribution.

Graph embedding: Graph isomorphic network (GIN) [42] is a kind of deep neural network [43,44] capable of learning representations of graph-structured data, and it is the latest variant of graph neural network (GNN). The disjunctive graph we have constructed encompasses all the information regarding the scheduling states, including the processing times of blocks at each workstation and the order of block processing. To extract all the embedded states from the disjunctive graph, we parameterize the policy $\pi(a_t|s_t)$ as a GIN $\pi_\theta(a_t|s_t)$ with trainable parameters θ. Given a graph $G = (V, E)$, GIN calculates p-dimensional embeddings for each node $v \in V$ by performing k iterations of update steps.

Action selection: The disjunctive graph provides all the information of the scheduling environment at each decision step t. By transmitting the context information embedded in the disjunctive graph to a multi-layer perceptron network, the probability distribution of all actions at this step t is generated.

4.2. Learning Algorithm

In this paper, we employ the PPO algorithm to train our agent. PPO is an Actor–Critic algorithm. Detailed information about the PPO algorithm is provided in pseudocode in Algorithm 1. The Actor refers to the policy network π_θ described above, while the Critic and Actor utilize the same GIN.

We present the scheduling process of our proposed reinforcement learning method in the ship panel block assembly line in Figure 5. This reinforcement learning model consists of an Agent that determines the input order of panel blocks and an environment that captures the current state of the assembly line and the processing information of each station using a disjunctive graph. The environment provides feedback to the agent regarding the current processing status of the assembly line. Subsequently, the agent selects a block to input and makes decisions on the next block to input when certain operations on the block are completed.

Algorithm 1. PPO Algorithm for training our model

```
Input: update epoch $k$; PPO steps $M$; number of actors to compute reward and perform update $N$; actor network $\pi_\theta$; behavior actor network $\pi_{\theta_{old}}$; trainable parameters of actor network $\theta$; trainable parameters of behavior actor network $\theta_{old}$; critic network $V_\varnothing$; trainable parameters of critic network $\varnothing$; clipping ratio $\varepsilon$; policy loss coefficient $C_p$; value function loss coefficient $C_v$; entropy loss coefficient $C_e$;
1 Initialization: initialize parameter sets of $\pi_\theta$, $\pi_{\theta_{old}}$ and $V_\varnothing$;
2 for $m = 1, \cdots, M$, do;
3 Pick $N$ independent scheduling instances from distribution $D$;
4 for $n$=1, $\cdots$ , $N$, do;
5       for $t$=0, 1, 2, $\cdots$, do
6         sample $a_{n,t}$ based on $\pi_{\theta_{old}}(a_{n,t}|S_{n,t})$;
7         Receive reward $r_{n,t}$ and next state $S_{n,t+1}$;
8         $\hat{A}_{n,t} = \sum_0^t \Upsilon^t r_{n,t} - V_\varnothing(S_n, t)$; $r_{n,t}(\theta) = \frac{\pi_\theta(a_{n,t}|S_{n,t})}{\pi_{\theta_{old}}(a_{n,t}|S_{n,t})}$
9         if $S_{n,t}$ is terminal then
10              break;
11          end
12      end
13      $L^{CLIP}(\theta) = \sum_0^t min(r_{n,t}(\theta)\hat{A}_{n,t},\ clip(r_{n,t}(\theta),\ 1-\varepsilon,\ 1+\varepsilon)\hat{A}_{n,t})$
14      $L^E(\theta) = \sum_0^t Entropy(\pi_\theta(a_{n,t}|S_{n,t}))$
15      $L^{VF}(\varnothing) = \sum_0^t (V_\varnothing(S_{n,t}) - \hat{A}_{n,t})^2$
16      $L(\theta,\varnothing) = C_p L^{CLIP}(\theta) - C_v L^{VF}(\varnothing) + C_e L^E \theta$
17    end
18    for $k = 1, 2, \cdots, K$ do;
19    update actor and critic parameters $\theta, \varnothing$ by Adam optimizer
20    $\theta, \varnothing = argmax\left(\sum_{n=1}^N L_n(\theta,\varnothing)\right)$
21    end
22    $\theta_{old} \leftarrow \theta\ (update,\ \theta_{old})$
23 end
24 Output: Trained parameter set of $\theta$
```

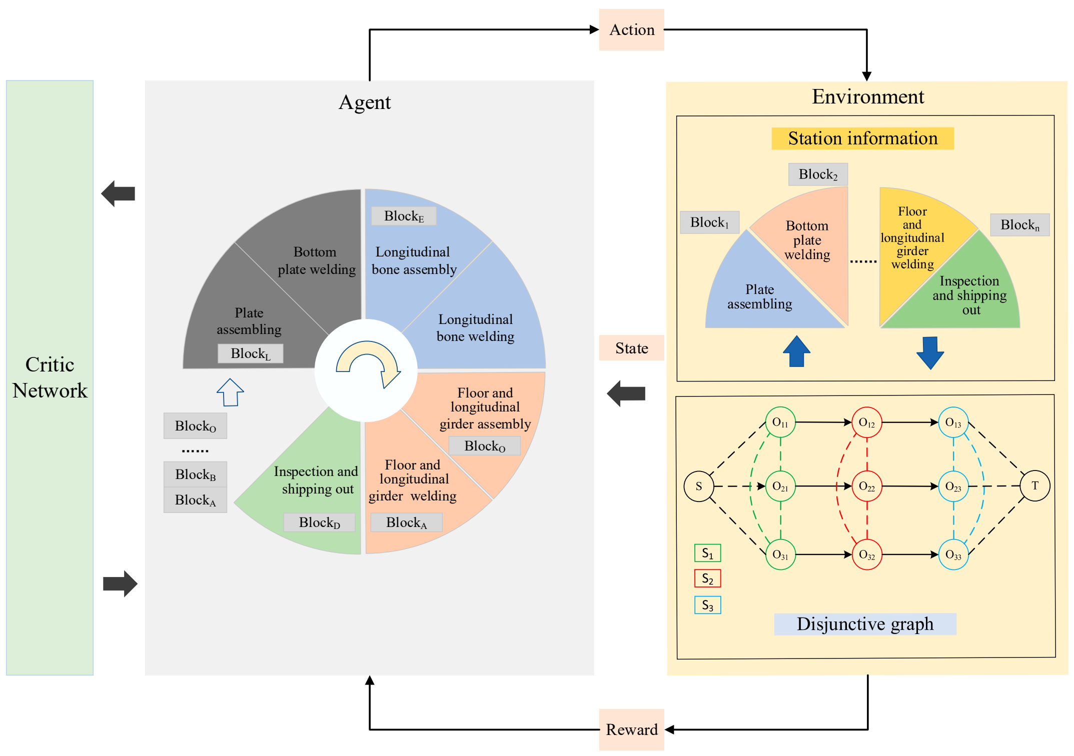


Figure 5. The algorithm structure of the panel block assembly line scheduling process.

5. Computational Experiment

In this section, we present the computational results of our method on instances of various scales. We demonstrate the effectiveness of our model through two validation approaches. Firstly, we conduct tests using real shipyard data, and secondly, we evaluate our model using the publicly available Taillard benchmark dataset [45]. To assess the performance of the proposed model, we compare it to six baseline methods, considering both the minimization of the maximum completion time and the training time of each algorithm.

Datasets: In this study, we trained our model using real processing data of ship hull blocks on a panel block assembly line. The dataset comprises 40 types of blocks, totaling 287, with their respective processing data across seven workstations. As shown in Table 2, we randomly selected six blocks to illustrate their processing times at each workstation in the assembly line. We use a random sampling method to select 50 blocks from several panel blocks to train our model. Subsequently, the trained model was tested on instances with 25, 50, 75, 100, and 125 planar segments, totaling five cases. To evaluate the effectiveness of our approach, the trained model was also subjected to nine test sets ranging from 20 $\times$ 5 to 100 $\times$ 20 using the Taillard benchmark.

Table 2. The processing times of randomly selected blocks at each workstation.

Workstation	Block Number					
	Block1	**Block2**	**Block3**	**Block4**	**Block5**	**Block6**
Plate assembling	2.7 h	3.0 h	3.0 h	3.0 h	3.0 h	2.7 h
Bottom plate welding	4.6 h	4.8 h	4.8 h	4.5 h	4.5 h	4.6 h
Longitudinal bone assembly	2.6 h	2.4 h	2.4 h	2.5 h	2.5 h	2.6 h
Longitudinal bone welding	3.3 h	3.0 h	3.0 h	3.4 h	3.4 h	3.3 h
Ribbed longitudinal Truss assembly	3.8 h	3.5 h	3.5 h	3.6 h	3.6 h	3.8 h
Ribbed longitudinal Truss welding	5.8 h	5.4 h	5.4 h	5.8 h	5.8 h	5.8 h
Inspection and shipping out	2.5 h	3.2 h	3.2 h	2.5 h	2.5 h	2.5 h

Baseline methods: In the two test cases, our model is compared to heuristic algorithms LPT, NEH, and metaheuristic algorithms GA, TS. Additionally, to demonstrate the effectiveness of our introduced disjunctive graph state representation approach, we contrast it with DDQN and DRL algorithm PPO which employs traditional state representation methods. Detailed descriptions of the six baseline methods are provided in Appendix A.

Experimental Settings: Our experiments were conducted entirely in Python 3.8, running on a computer equipped with an AMD Ryzen 7 5800H/3.20Ghz CPU and an NVIDIA RTX3050Ti GPU. The selection of appropriate parameters plays a crucial role in the success of our experiments. Table 3 presents the parameters utilized during the training process of our model.

Table 3. Hyperparameter configuration.

Hyperparameter	Value
Batch size	128
Learning rate	10^{-4}
Learning rate decay factor	0.98
Learning rate decay step	3000
Optimizer	Adam
The clipping parameter	0.2
The policy loss coefficient	2

Evaluation metrics: We employ the metrics of *makespan* and the computational time of the model to assess the performance of our method and the baseline approaches.

5.1. Computational Results of the Panel Block Assembly Line

Table 4 presents the computational results of six baseline methods and our approach, in which the time unit of *makespan* is hours, as shown by the character h in brackets in the title of the table. We test five groups of instances of different planar segments. By comparing them, we can conclude that our method yields a smaller *makespan* than all the baseline methods in all cases. Compared with the two models of reinforcement learning, our method can save 1–2% time, and it can save 10% time compared with LPT.

Table 4. *Makespan* of each algorithm on the panel block assembly line (h).

Number of Blocks	Heuristic		Metaheuristic		Reinforcement Learning		Ours
	LPT	NEH	GA	TS	DDQN	PPO	
25	180.5	162.7	164.6	163.4	160.7	159.3	157.4
50	327.7	299.8	301.3	299.6	295.1	293.4	289.7
75	457.5	419.4	421.7	420.9	414.3	412.2	407.1
100	613.7	564.6	566.5	564.7	561.6	557.6	550.4
125	753.3	696.1	698.4	694.1	689.5	685.2	674.9

To visually demonstrate the differences between our model and the baseline methods, we showcase the disparities by taking the differences between the baseline methods and our model's results in Table 5. Additionally, we depict a line graph in Figure 6 that intuitively illustrates how the magnitude of differences varies with the problem scale. From Table 5 and Figure 6, it can be observed that our model consistently outperforms the other methods across all problem instances. The metaheuristic algorithm shows superior performance compared to the heuristic algorithm, but the NEH algorithm and the metaheuristic algorithm perform similarly, with the NEH algorithm even slightly outperforming the metaheuristic algorithm. Overall, the reinforcement learning algorithm surpasses the other baseline methods. However, when compared to other reinforcement learning algorithms, such as DDQN and PPO, which do not use disjunctive graph representation for states, our algorithm achieves lower *makespan*. For instance, compared with the DDQN method, our method can save 3.3 h of processing time when the segment count is 15 and 14.6 h when the segment count is 125. Therefore, in real shipbuilding scenarios, our approach can significantly enhance the efficiency of panel block assembly.

Table 5. Comparative analysis of the differences in *makespan* among the algorithms (h).

Number of Blocks	Heuristic		Metaheuristic		Reinforcement Learning		Ours
	LPT	NEH	GA	TS	DDQN	PPO	
25	23.1	5.3	7.2	6	3.3	1.9	0
50	38	10.1	11.6	9.9	5.4	3.7	0
75	50.4	12.3	14.6	13.8	7.2	5.1	0
100	63.3	14.2	16.1	14.3	11.2	7.2	0
125	78.4	21.2	23.5	19.2	14.6	10.3	0

With the increase in problem scale, it is evident that the disparity between other algorithms and the algorithm proposed in this paper in terms of *makespan* is also growing. The advantages of our method are even more pronounced. For instance, compared to the PPO algorithm that does not incorporate disjunctive graph representation for states, the time saved by our algorithm has expanded from 1.9 h to 10.3 h. This signifies that in practical applications, as ship sizes grow and the demand for panel blocks increases, our algorithm will save even more time.

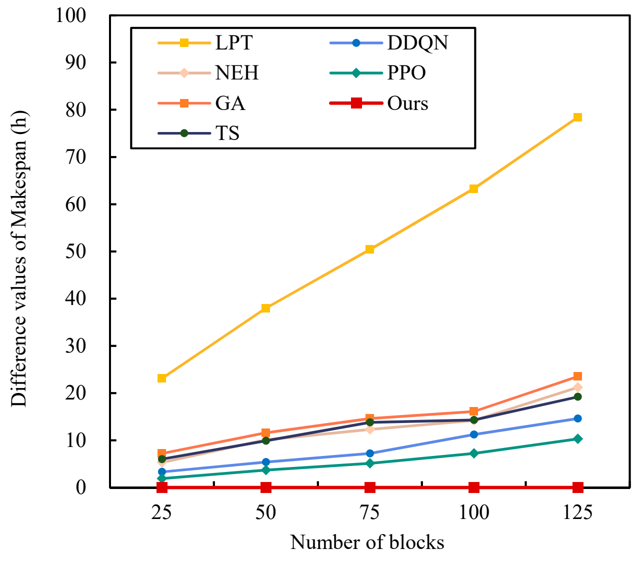


Figure 6. Line graph depicting the differences in *makespan* among the algorithms.

To provide a more comprehensive reflection of the model's performance, we further compared the computation times of each algorithm, as shown in Table 6, in which the time unit of computation time is seconds, as shown by the character s in parentheses in the title of the table. From Table 6, it can be observed that the LPT algorithm in the heuristic methods almost instantly yields scheduling results. However, compared to our model, the reinforcement learning algorithm, metaheuristic algorithm, and NEH algorithm require slightly longer computation times.

Table 6. Computation time of each algorithm on the panel block assembly line (s).

Number of Blocks	**Heuristic**		**Metaheuristic**		**Reinforcement Learning**		**Ours**
	LPT	**NEH**	**GA**	**TS**	**DDQN**	**PPO**	
25	0.00	2.12	4.36	7.51	1.45	1.53	1.21
50	0.00	2.87	6.32	10.12	1.78	1.85	1.54
75	0.00	4.35	7.94	13.10	2.77	3.02	2.32
100	0.00	7.43	10.22	15.62	4.34	4.67	3.83
125	0.00	10.52	12.53	18.33	5.68	6.04	4.99

Based on the results obtained from Tables 4 and 6, our algorithm achieves superior scheduling solutions with reduced computational time. Taking the case of a panel block size of 125 as an example, when evaluating the *makespan* criterion, the ordering of algorithms based on time, from longest to shortest, is as follows: LPT, GA, NEH, TS, DDQN, PPO, and our algorithm. On the other hand, when considering the computational time criterion, the ordering from longest to shortest is as follows: TS, GA, NEH, PPO, DDQN, our algorithm, and LPT. It is evident that although LPT exhibits the shortest computational time, it results in the longest *makespan*. Conversely, our method achieves the shortest *makespan* while also maintaining a computational time that is second only to LPT. Therefore, considering the comprehensive performance, our approach emerges as the optimal choice.

Similarly, we calculated the differences in computation times between the baseline methods and our model, as presented in Table 7. These differences were then plotted in a line graph, shown in Figure 7. Based on Table 7 and Figure 7, it is evident that, apart from the LPT algorithm, all other baseline methods have longer computation times than our approach, and as the instance scale increases, the gap in computation times widens. However, although the LPT algorithm can yield instant results, its processing time for *makespan* significantly exceeds that of our algorithm. Therefore, considering the overall performance, our model surpasses the other baseline methods and offers a more effective solution to the scheduling problem in ship panel block assembly lines.

Table 7. Comparative analysis of the differences in computation time among the algorithms (s).

Number of Blocks	Heuristic		Metaheuristic		Reinforcement Learning		Ours
	LPT	NEH	GA	TS	DDQN	PPO	
25	−1.21	0.91	3.15	6.30	0.24	0.32	0
50	−1.54	1.33	4.78	8.58	0.24	0.31	0
75	−2.32	2.03	5.62	10.78	0.45	0.70	0
100	−3.83	3.60	6.39	11.79	0.51	0.84	0
125	−4.99	5.53	7.54	13.34	0.69	1.05	0

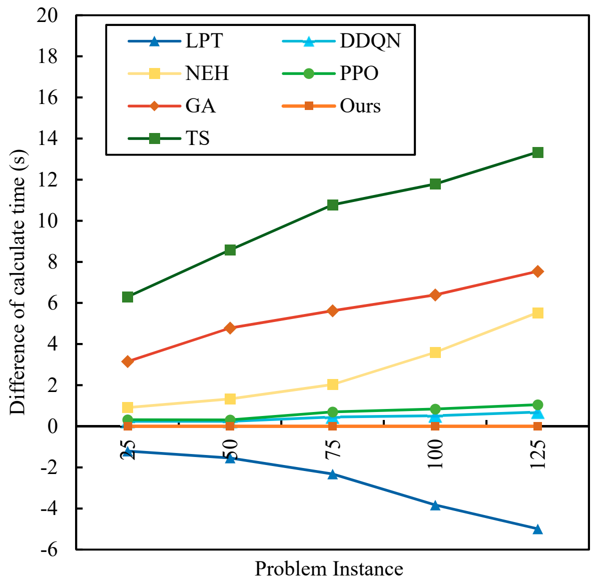


Figure 7. Line graph depicting the differences in computation time among the algorithms.

5.2. Computational Results of Benchmark Instances

In this section, we compare the performance of LPT, NEH, GA, TS, DDQN, and PPO without disjunctive graph representation and our approach to the Taillard benchmark dataset. Table 8 shows the *makespan* achieved by each algorithm for each problem scale. To highlight the differences in results, we have bolded the best results in the table. On the whole, reinforcement learning is better than metaheuristic and heuristic algorithms. But also, as a reinforcement learning algorithm, our method consistently achieves the smallest *makespan* across all problem instances, surpassing the performance of DDQN and PPO methods. Particularly, when the number of workstations is fixed at five and the number of blocks is 20, the disparities between different algorithms are minimal. However, as the number of blocks increases, our algorithm proves more efficient in time-saving. For instance, compared to the PPO algorithm, our algorithm can save 0 h, 13 h, and 33.5 h, respectively, when the number of blocks is 20, 50, and 100. With the increasing number of blocks, the superiority of our algorithm becomes increasingly prominent.

Table 9 presents the computational time of our approach and all the baseline methods. From Table 9, it can be observed that LPT remains instantaneous in solving instances of any scale. Although our method is slower than LPT in terms of computation speed, it outperforms other baseline methods, which is acceptable in practical industrial production. Furthermore, when the number of workstations is the same, our model demonstrates the ability to save more computational time compared to the other five methods, excluding the LPT approach, as the number of blocks increases. For instance, when compared to the PPO algorithm, which does not use disjunctive graph representation for states, our algorithm achieves respective reductions in computational time of 0.07 s, 0.23 s, and 0.54 s when the number of blocks is 20, 50, and 100. Moreover, the rate of time savings surpasses the rate of block increase significantly. This observation further underscores our model's capacity to economize computational time as the instance size expands.

Table 8. *Makespan* of each algorithm on the common benchmark (h).

Problem Instance	Size	Heuristic		Metaheuristic		Reinforcement Learning		Ours
		LPT	NEH	GA	TS	DDQN	PPO	
Ta010	20 × 5	1213.4	**1108**	**1108**	**1108**	**1108**	**1108**	**1108**
Ta020	20 × 10	1701	1689.7	1693	1691.3	1657.7	1646.3	**1640.2**
Ta030	20 × 20	2305.2	2286.1	2278.4	2259.8	2263.0	2256.5	**2247.4**
Ta040	50 × 5	2901.6	2884.5	2881.6	2875	2879.1	2874.8	**2861.8**
Ta050	50 × 10	3198	3175.4	3171.5	3162.6	3168.3	3159.5	**3145.1**
Ta060	50 × 20	3971.6	3951.8	3954	3946.5	3931.4	3903.4	**3887.4**
Ta070	100 × 5	5560	5483.2	5479.1	5451.5	5452.5	5429.7	**5396.2**
Ta080	100 × 10	6089.7	6004	5996.4	5989.1	5983.0	5971.8	**5949**
Ta090	100 × 20	6705.9	6670.8	6658.5	6649.4	6652.3	6649.4	**6627**

Table 9. Computation time of each algorithm on the common benchmark (s).

Problem Instance	Size	Heuristic		Metaheuristic		Reinforcement Learning		Ours
		LPT	NEH	GA	TS	DDQN	PPO	
Ta010	20 × 5	0	1.73	3.75	7.14	0.70	0.79	0.72
Ta020	20 × 10	0	2.14	4.26	7.58	1.19	1.35	**1.14**
Ta030	20 × 20	0	2.47	5.74	9.77	1.39	1.52	**1.31**
Ta040	50 × 5	0	2.61	6.22	10.26	1.41	1.58	**1.35**
Ta050	50 × 10	0	4.21	7.69	12.91	2.64	2.95	**2.38**
Ta060	50 × 20	0	6.72	8.94	14.59	4.62	5.35	**3.42**
Ta070	100 × 5	0	7.33	9.59	15.14	5.44	6.28	**3.74**
Ta080	100 × 10	0	10.87	12.94	17.93	7.65	9.32	**4.47**
Ta090	100 × 20	0	12.95	15.47	19.66	8.60	10.67	**6.61**

Based on the results presented in Tables 8 and 9, our method consistently exhibits the shortest computational time and achieves the smallest makespan across all test cases. Consequently, when considering the comprehensive performance of the model, it can be confidently concluded that our approach continues to surpass the baseline methods in the Tarillard benchmark test.

5.3. Discussion

By comparing our proposed model with six baseline methods in two test cases, our model outperforms other baseline methods in terms of computation results and computational time. Among them, the LPT algorithm demonstrates near real-time computation but exhibits a significant disparity in *makespan* results compared to other algorithms. Therefore, under specific conditions that prioritize computation time, the LPT algorithm is undoubtedly more suitable. However, when considering the overall performance of the model and computational time, our method holds a distinct advantage.

As our model's computational instances scale up from 25 to 125, it continues to exhibit superior performance compared to other algorithms, showcasing its remarkable generalization capabilities when handling larger-scale instances. With the development of larger ships, the demand for panel blocks increases significantly. Thanks to the robust generalization abilities of our model, our approach can effectively tackle this challenge. Additionally, it is noteworthy that our model outperforms the PPO algorithm, which does not utilize disjunctive graph representation of states, in both sets of test experiments.

In essence, the satisfactory performance of our method can be attributed to the end-to-end model based on DRL efficiently generating feasible scheduling sequences, coupled with the disjunctive graph-based state representation method, which provides a more comprehensive and rational depiction of the current scheduling information. Hence, our model stands as a competitive approach for solving the ship panel block assembly scheduling problem. In practical shipbuilding processes, our method can provide scientific and effective scheduling plans for panel block assembly, reducing construction time for blocks, and facilitating the execution of subsequent work plans, thereby shortening the shipbuilding cycle and lowering costs. Moreover, the scheduling method for panel block assembly lines can offer valuable insights for scheduling other product assembly lines, thereby comprehensively enhancing shipbuilding management and boosting enterprise competitiveness.

6. Conclusions and Future Work

This study investigates the scheduling problem of ship panel block assembly lines, aiming to minimize the assembly time of panel blocks by determining the sequence of incoming blocks. To address this problem, we propose an end-to-end scheduling method based on DRL. Initially, we establish an MDP model that conforms to the scheduling environment, creatively employing disjunctive graphs to capture the state of the current node and designing a reward function. In order to enhance the learning of information contained in the disjunctive graphs, we introduce a policy function based on GIN and train it using PPO algorithms. We compare our proposed model with heuristic algorithms LPT and NEH, metaheuristic algorithms GA and TS, as well as reinforcement learning algorithms DDQN and PPO without disjunctive graph state representation. Experimental results demonstrate that our algorithm surpasses other baseline methods in terms of both model performance and computational time considerations. Moreover, our model exhibits strong generalization ability when handling larger-scale instances. In the practical production of panel blocks, our approach enables shipyards to save significant time, and as the demand for panel blocks increases with the development of large-scale vessels, our model can effectively respond to these requirements.

However, it should be noted that our proposed DRL method, being an optimization approach, does not guarantee the discovery of a globally optimal solution. Due to limitations in available data, we were unable to test our model on larger-scale instances. Additionally, in order to simplify the problem, we have made simplifications to the actual process of panel block assembly in the assembly line. However, in real-world scenarios, scheduling issues can become more complex, and in the future, we aim to further investigate how to handle unforeseen events that may arise during the actual assembly process, such as workstation machine failures or the introduction of urgent tasks. To better address the scheduling problem in the panel block assembly line, we will enhance the robustness of our model and validate its scalability by testing on larger-scale instances. Furthermore, we will continue exploring improved heuristic and metaheuristic algorithms while conducting additional experiments to verify the consistency of our model across different datasets.

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

Funding: This research was funded by the Fundamental Research Funds for the Central Universities. (2023-vb-016).

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Not applicable.

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

Appendix A

Appendix A.1. LPT

The LPT algorithm involves arranging segments in non-decreasing order based on their processing duration and selecting the segment with the longest processing time as the first to be processed. When a workstation enters an idle state, the remaining task with the longest processing duration is assigned to that vacant workstation until all tasks have been sequentially dispatched.

Appendix A.2. NEH

The NEH algorithm is a heuristic algorithm. The steps are as follows: (1) Arrange the segments to be scheduled in non-increasing order of their total processing time. (2) Start by taking the first element from the sequence, then take the second element from the sequence and insert it before and after the first element. This creates two sub-sequences. Choose the sub-sequence with the smallest $makespan$ as the current sub-sequence. (3) Take the third element from the sequence and insert it at all possible insertion positions within the current sub-sequence. This creates three sub-sequences. Choose the sub-sequence with the smallest $makespan$ as the current sub-sequence. Repeat this process until all segments have been scheduled.

Appendix A.3. Genetic Algorithm (GA)

The Genetic Algorithm (GA) is a stochastic global search optimization method that emulates phenomena such as replication, crossover, and mutation observed in natural selection and genetics. It serves as a computational model for searching for optimal solutions, drawing inspiration from natural selection in evolutionary theory and the biological evolution process in genetics. Starting from any initial population, the algorithm generates a set of individuals that are better suited to the environment through random selection, crossover, and mutation operations. This allows the population to evolve towards increasingly favorable regions in the search space, ultimately converging into a group of individuals that are best adapted to the environment and yielding high-quality solutions to the problem.

Appendix A.4. Tabu Search Algorithm (TS)

The Tabu Search algorithm (TS) emulates the human memory function by utilizing a taboo list to block previously explored regions, thus avoiding redundant searches. It also grants clemency to certain promising states within the taboo region, ensuring search diversity and ultimately achieving global optimality. The algorithm begins with an initial feasible solution and enters a cyclic iterative process. In each iteration, the current solution undergoes a neighborhood search using a neighborhood selection strategy, generating neighboring solutions. The optimal solution within the neighborhood, selected based on an evaluation function, becomes the current solution. The algorithm repeats this iterative process while evaluating if the current solution surpasses the optimal solution, continuing until the termination condition is met.

Appendix A.5. DDQN

Random samples of scheduling experience are extracted from the scheduling experience pool and fed into the DDQN deep neural network model trainer. This trainer updates the network parameters of the deep neural network scheduling model. The implementation process is illustrated in Figure A1. In the scheduling experience pool, the scheduling experience samples take the form of $(s,\ d,\ s',\ r)$, where s represents the state of the panel block assembly line, d represents the scheduling plan, s' represents the new production state after applying scheduling plan d to the panel block assembly line, and r represents the reward obtained when applying scheduling plan d under state s. The DDQN consists of the Current Network, responsible for action retrieval, and the Target Network, responsible for action value computation. Both networks have identical structures. We treat the various system states s in the MDP as input values to the neural network. The output of the neural network is a Q-table for different actions, where each dimension of the Q-table maps to a specific action. The value stored at a particular index in the Q-table represents the Q-value of that action. A higher Q-value indicates greater value and rationality for the corresponding action.

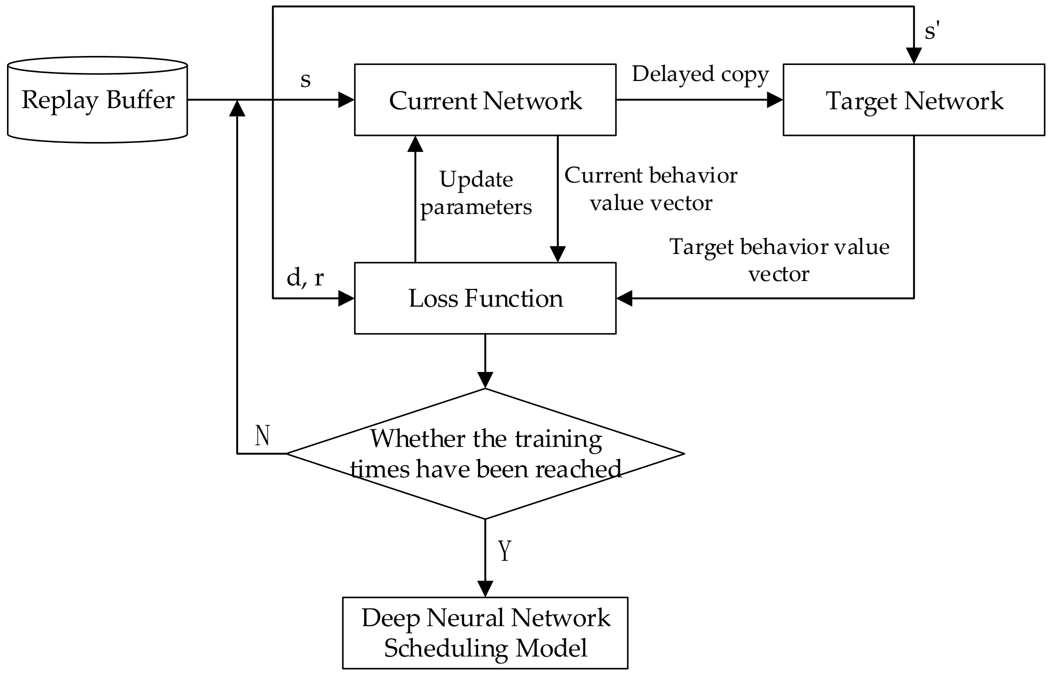

s'
Replay Buffer
s
Current Network
Delayed copy
Target Network
Update parameters
Current behavior value vector
Target behavior value vector
d, r
Loss Function
N
Whether the training times have been reached
Y
Deep Neural Network Scheduling Model

Figure A1. Line graph depicting the differences in computation time among the algorithms.

Appendix A.6. PPO

The PPO algorithm consists of two networks: a value function network and a policy network. The agent samples actions from the output of the policy network, while the value function network evaluates the actions. The parameters of both networks are updated alternately using gradient descent. The PPO algorithm employs a clipped surrogate performance function to address the reward cliff problem, as shown in Equation (A1).

$$L^{clicp}(\theta',\theta) = \sum\nolimits_{(s_t,\ a_t)} min\begin{cases} r_t(\theta)R_{\theta_{min}}(s_t, a_t) \\ clip(r_t(\theta),\ 1-\epsilon, 1+\epsilon\ )R_{\theta_{old}}(s_t, a_t) \end{cases} \tag{A1}$$

where θ' represents the new policy parameters, θ represents the old policy parameters, $r_t(\theta) = \frac{\pi(a_t|s_t,\ \theta')}{\pi(a_t|s_t,\ \theta)}$ denotes the importance sampling ratio that characterizes the similarity between the old and new policies, and ϵ is the clipping parameter. PPO utilizes gradient ascent to update the parameters, as shown in Equation (A2).

$$\theta' = \theta + \alpha\nabla_{\theta'}L^{clicp}(\theta',\theta) \tag{A2}$$

References

1. Shao, K.; Zhu, Y.H.; Zhao, D.B. StarCraft micromanagement with reinforcement learning and curriculum transfer learning. *IEEE Trans. Emerg. Top. Comput. Intell.* **2019**, *3*, 73–84. [CrossRef]
2. Guo, H.N.; Li, S.H.; Qi, K.Y.; Guo, Y.; Xu, Z.W. Learning automata based competition scheme to train deep neural networks. *IEEE Trans. Emerg. Top. Comput. Intell.* **2020**, *4*, 151–158. [CrossRef]
3. Cai, Q.; Jing, X.; Chen, Y.; Liu, J.; Kang, C.; Li, B. Online Monitoring of Ship Block Construction Equipment Based on the Internet of Things and Public Cloud: Take the Intelligent Tire Frame as an Example. *KSII Trans. Internet Inf. Syst.* **2021**, *15*, 3970–3990.
4. Salazar-Domínguez, C.M.; Hernández-Hernández, j.; Rosas-Huerta, E.D.; Iturbe-Rosas, G.E.; Herrera-May, A.L. Structural Analysis of a Barge Midship Section Considering the Still Water and Wave Load Effects. *J. Mar. Sci. Eng.* **2021**, *9*, 99. [CrossRef]
5. Hoosen, M.; Chalfant, J.S. Subdivision Blocks and Component Placement in Early-Stage Ship Design. In Proceedings of the 2021 IEEE Electric Ship Technologies Symposium (ESTS), Arlington, VA, USA, 4–6 August 2021.
6. Son, Y.B.; Nam, J.H. Creation of hierarchical structure for computerized ship block model based on interconnection relationship of structural members and shipyard environment. *Int. J. Nav. Arch. Ocean* **2022**, *14*, 100455. [CrossRef]
7. Zheng, Y.Q.; Mo, G.F.; Zhang, J. Blocking flowline scheduling of panel block in shipbuilding. *Comput. Integr. Manuf. Syst.* **2016**, *22*, 2305–2314.
8. Woo, J.H.; Oh, D. Development of simulation framework for shipbuilding. *Int. J. Comput. Integ. M* **2018**, *31*, 210–227. [CrossRef]
9. Lee, Y.G.; Ju, S.H.; Woo, J.H. Simulation-based planning system for shipbuilding. *Int. J. Comput. Integr. Manuf.* **2020**, *33*, 626–641. [CrossRef]
10. Kwak, D.H.; Woo, J.H.; Park, J.G. Analysis of master plan and procurement plan of shipbuilding based on queuing theory with variability. *J. Korean Inst. Ind. Eng.* **2020**, *46*, 673–682.
11. Yang, Z.; Liu, C. A hybrid multi-objective gray wolf optimization algorithm for a fuzzy blocking flow shop scheduling problem. *Adv. Mech. Eng.* **2018**, *10*, 2072045641. [CrossRef]
12. Ko, D. A Study on the Saving Method of Plate Jigs in Hull Block Butt Welding. IOP conference series. *Mater. Sci. Eng.* **2017**, *269*, 12089.
13. Kafali, M.; Aydin, N.; Genç, Y.; Çelebi, U.B. A two-stage stochastic model for workforce capacity requirement in shipbuilding. *J. Mar. Eng. Technol.* **2022**, *21*, 146–158. [CrossRef]
14. Guo, H.; Li, J.; Yang, B.; Mao, X.; Zhou, Q. Green scheduling optimization of ship plane block flow line considering carbon emission and noise. *Comput. Ind. Eng.* **2020**, *148*, 106680. [CrossRef]
15. Kolich, D.; Storch, R.L.; Fafandjel, N. Lean Methodology to Transform Shipbuilding Panel Assembly. *J. Ship Prod. Des.* **2017**, *33*, 317–326. [CrossRef]
16. Luo, S. Dynamic Scheduling for Flexible Job Shop with New Job Insertions by Deep Reinforcement Learning. *Appl. Soft. Comput.* **2020**, *91*, 106208. [CrossRef]
17. Ohn, S.; Filip, W.; Prafulla, D.; Alec, R.; Oleg, K. Proximal policy optimization algorithms. *arXiv* **2017**, arXiv:1707.06347.
18. Fernandez-Viagas, V.; Rubén, R.; Jose, M.F. A New Vision of Approximate Methods for the Permutation Flowshop to Minimise Makespan: State of the Art and Computational Evaluation. *Eur. J. Oper. Res.* **2017**, *257*, 707–721. [CrossRef]
19. Hall, N.G.; Sriskandarajah, C. A survey of machine scheduling problems with blocking and no-wait in process. *Oper. Res.* **1996**, *44*, 510–525. [CrossRef]
20. Oliveira, A.; Gordo, J.M. Lean Tools Applied to a Shipbuilding Panel Line Assembling Process. *Brodogradnja* **2018**, *69*, 53–64. [CrossRef]
21. Ryu, H.; Kang, S.; Lee, K. Numerical Analysis and Experiments of Butt Welding Deformations for Panel Block Assembly. *Appl. Sci.* **2020**, *10*, 1669. [CrossRef]
22. Johnson, S.M. Optimal two- and three-stage production schedules with setup times included. *Nav. Res. Logist. Q.* **1954**, *1*, 61–67. [CrossRef]
23. Palmer, D.S. Sequencing Jobs Through a Multi-Stage Process in the Minimum Total Time—A Quick Method of Obtaining a Near Optimum. *J. Oper. Res. Soc.* **2017**, *16*, 101–107. [CrossRef]
24. Gupta, J. A Functional Heuristic Algorithm for the Flowshop Scheduling Problem. *J. Oper. Res. Soc.* **1971**, *22*, 39–47. [CrossRef]
25. Singh, H.; Oberoi, J.S.; Singh, D. The taxonomy of dynamic multi-objective optimization of heuristics algorithms in flow shop scheduling problems: A systematic literature review. *Int. J. Ind. Eng.-Theory Appl. Pract.* **2020**, *27*, 429–462.
26. Framinan, J.M.; Leisten, J. A review and classification of heuristics for permutation flow-shop scheduling with makespan objective. *J. Oper. Res. Soc.* **2004**, *55*, 1243–1255. [CrossRef]
27. Hsu, C.Y.; Chann, C.P.; Hui, C.M. A link age miningin block-based evolutionary algorithm for permutation flowshop scheduling problem. *Comput. Ind. Eng.* **2015**, *83*, 159–171. [CrossRef]
28. Lin, Q.; Gao, L.; Li, X.; Zhang, C. A hybrid back tracking search algorithm for permutation flow-shop scheduling problem. *Comput. Ind. Eng.* **2015**, *85*, 437–446. [CrossRef]
29. Karabulut, K. A hybrid iterated greedy algorithm for total tardiness minimization in permutation flowshops. *Comput. Ind. Eng.* **2016**, *98*, 300–307. [CrossRef]
30. Deb, S.; Tian, Z.; Fong, S.; Tang, R.; Wong, R.; Dey, N. Solving permutation flow-shop scheduling problem by rhinoceros search algorithm. *Soft Comput.* **2018**, *22*, 6025–6034. [CrossRef]

31. De Fátima, M.M.; Ribeiro, M.H.D.M.; Silva R, G. Discrete differential evolution metaheuristics for permutation flow shop scheduling problems. *Comput. Ind. Eng.* **2022**, *166*, 107956.
32. 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]
33. Waschneck, B.; Reichstaller, A.; Belzner, L. Optimization of global production scheduling with deep reinforcement learning. *Procedia CIRP* **2018**, *72*, 1264–1269. [CrossRef]
34. Lin, C.C.; Deng, D.J.; Chih, Y.L. Smart manufacturing scheduling with edge computing using multiclass deep Q network. *IEEE Trans. Ind. Inform.* **2019**, *15*, 4276–4284. [CrossRef]
35. Park, J.; Chun, J.; Kim, S.H. Learning to schedule job-shop problems: Representation and policy learning using graph neural network and reinforcement learning. *Int. J. Prod. Res.* **2021**, *59*, 3360–3377. [CrossRef]
36. Yang, S.; Xu, Z.; Wang, J. Intelligent decision-making of scheduling for dynamic permutation flowshop via deep reinforcement learning. *Sensors* **2021**, *21*, 1019. [CrossRef]
37. Pan, R.; Dong, X.; Han, S. Solving permutation flowshop problem with deep reinforcement learning. In Proceedings of the Prognostics and Health Management Conference (PHM-Besançon), Besançon, France, 4–7 May 2020; pp. 349–353.
38. Yang, S.L.; Xu, Z.G. The Distributed Assembly Permutation Flow shop Scheduling Problem with Flexible Assembly and Batch Delivery. *Int. J. Prod. Res.* **2021**, *59*, 4053–4071. [CrossRef]
39. Ramírez, J.; Yu, W.; Perrusquía, A. Model-free reinforcement learning from expert demonstrations: A survey. *Artifcial Intell. Rev.* **2022**, *55*, 3213–3241. [CrossRef]
40. 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]
41. 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**, *19*, 1322–1331. [CrossRef]
42. Battaglia, P.W.; Hamrick, J.B.; Bapst, V.; Sanchez-Gonzalez, A.; Zambaldi, V.; Malinowski, M.; Tacchetti, A.; Raposo, D.; Santoro, A.; Faulkner, R.; et al. Relational inductive biases, deep learning, and graph networks. *arXiv* **2018**, arXiv:1806.01261.
43. Zhou, J.; Cui, G.; Zhang, Z.; Yang, C.; Liu, Z.; Wang, L.; Li, C.; Sun, M. Graph neural networks: A review of methods and applications. *arXiv* **2018**, arXiv:1812.08434. [CrossRef]
44. Lv, M.; Hong, Z.; Chen, L.; Chen, T.; Zhu, T.; Ji, S. Temporal multi-graph convolutional network for traffic flow prediction. *IEEE Trans. Intell. Transp. Syst.* **2020**, *22*, 3337–3348. [CrossRef]
45. Taillard, E.D. Benchmarks for basic scheduling problems. *Eur. J. Oper. Res.* **1993**, *64*, 278–285. [CrossRef]