applied sciences

MDPI

Article

Load Balancing of Two-Sided Assembly Line Based on Deep Reinforcement Learning

Guangpeng Jia [1], Yahui Zhang [2,*], Shuqi Shen [1], Bozu Liu [1], Xiaofeng Hu [3] and Chuanxun Wu [2]

[1] Process Research Institution, China National Heavy Duty Truck Group Co., Ltd., Jinan 250100, China
[2] 5G Intelligent Manufacturing Research Center, Institute of Marine Equipment, Shanghai Jiao Tong University, Shanghai 200240, China
[3] Institute of Intelligent Manufacturing and Information Engineering, School of Mechanical Engineering, Shanghai Jiao Tong University, Shanghai 200240, China
* Correspondence: emily187@sjtu.edu.cn

Abstract: In the complex and ever-changing manufacturing environment, maintaining the long-term steady and efficient work of the assembly line is the ultimate goal pursued by relevant enterprises, the foundation of which is a balanced load. Therefore, this paper carries out research on the two-sided assembly line balance problem (TALBP) for load balancing. At first, a mathematical programming model is established with the objectives of optimizing the line efficiency, smoothness index, and completion time smoothness index of the two-sided assembly line (TAL). Secondly, a deep reinforcement learning algorithm combining distributed proximal policy optimization (DPPO) and the convolutional neural network (CNN) is proposed. Based on the distributed reinforcement learning agent structure assisted by the marker layer, the task assignment states of the two-sided assembly and decisions of selecting tasks are defined. Task assignment logic and reward function are designed according to the optimization objectives to guide task selection and assignment. Finally, the performance of the proposed algorithm is verified on the benchmark problem.

Keywords: two-sided assembly line; load balancing; deep reinforcement learning; distributed multiple processes

Citation: Jia, G.; Zhang, Y.; Shen, S.; Liu, B.; Hu, X.; Wu, C. Load Balancing of Two-Sided Assembly Line Based on Deep Reinforcement Learning. *Appl. Sci.* **2023**, *13*, 7439. https://doi.org/10.3390/app13137439

Academic Editors: Zoran Jurković, David Ištoković and Janez Gotlih

Received: 20 May 2023
Revised: 12 June 2023
Accepted: 20 June 2023
Published: 23 June 2023

1. Introduction

An assembly line (AL) is an arrangement of workstations in a streamlined manner according to the product assembly process sequence for the organization and the arrangement of production. A workstation is an assembly unit that focuses on a specific segment of production, and the workpiece is assembled in all the workstations to form a complete product. Due to the assembly line adopting the flow operation mode, the assembly operation process is standardized, and the assembly workers are generally fixed in a single station or several adjacent stations for repeated operations, which increases the utilization rate of workers, thus greatly improving production efficiency [1].

In the two-sided assembly line (TAL), the left and right sides of the same workstation can independently execute assembly of the same product with different processes in parallel, as shown in Figure 1. These are called mated stations [2]. Compared with the one-sided AL, the TAL can effectively shorten the length of theAL, improve the utilization rate of auxiliary tools, reduce the time loss caused by the movement of workers between various stations, and reduce the transportation cost of assembly parts [2].

To balance the TAL is to distribute a group of tasks evenly to each station as far as possible under certain constraints to pursue the optimization of one or more objectives [3]. However, after the assembly line is put into operation, the original balance parameters, such as cycle time, operation content, operation time, and assembly process, may change with the improvement of assembly workers' technology, product upgrades, customer demand change, etc. In such cases, the original assembly line balance may be disrupted, prohibiting

the assembly line from being a stable, efficient, and high-quality operation, and thus greatly reducing the economic benefits of the relevant enterprises. Therefore, it is necessary to optimize and improve the original balancing scheme, which involves reassigning tasks to achieve the assembly line load balance and improve the operation efficiency of the TAL.

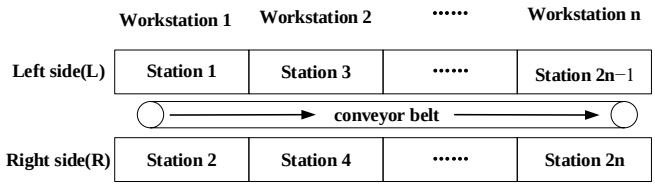


Figure 1. Two-sided assembly line.

TALBP is one of the NP-hard problems belonging to combinatorial optimization [4]. Since it was proposed in 1993 [5], it has been widely studied. The main research methods include exact algorithm, heuristic algorithm, and meta-heuristic algorithm [1,2]. Although the exact algorithm can obtain the optimal solution, its solving speed is slow and can can be used for small-sized problems; the heuristic algorithm is fast, simple, and efficient, but the solution results cannot reach the global optimal; and the meta-heuristic algorithm is relatively fast and effective, but the iterative search process is usually time-consuming and needs to be solved iteratively for each problem case. Moreover, these traditional optimization algorithms rarely make use of historical information to adjust behavior and cannot effectively use historical solving experience for learning; hence, there is great room for improvement in terms of solving large-scale problems.

In addition, different cases of problems have the same combinatorial optimization structure (the objective function or the coefficient of constraint conditions); there are only differences in specific values; for example, improving assembly workers' skills will reduce the working time of tasks, but the correlation between tasks does not change. As an artificial intelligence algorithm closer to the way of human thinking, the deep reinforcement learning algorithm has deep association learning ability compared with the traditional optimization algorithm. When solving the cases at a same scale, historical experience of different cases—that is, existing task assignment schemes—can be associated and learned, the essential information of problems can be mined, and task assignment strategies can be obtained and automatically updated to achieve efficient solutions to similar cases. Moreover, the deep reinforcement learning algorithm has higher adaptability to the complex and ever-changing production environment, which is easy to adjust so as to alter the solution. In addition, although deep reinforcement learning algorithms have begun to be tried in solving such AL balancing problems [6,7], they are currently used to optimize simulation models and resource allocation. Therefore, this paper carries out the study on load balancing of TAL based on deep reinforcement learning for the first time.

The remaining sections are introduced as follows. Section 2 provides a literature review for the load balancing-oriented TALBP and deep reinforcement learning. Section 3 shows the mathematical model of load balancing-oriented TALBP. Section 4 describes the deep reinforcement learning algorithm combining distributed proximal policy optimization (DPPO) and convolutional neural network (CNN). The experimental verification is carried out in Section 5 and conclusions and future work avenues are presented in Section 6.

2. Literature Review

2.1. Two-Sided Assembly Line Balance Oriented to Load Balancing

According to different production stages and research objectives, TALBPs are mainly divided into two categories [8]. The first type of balancing problem occurs in the design and planning stage, the aim was to explore that how to use the minimum number of workstations and/or stations to achieve production under the premise of a given cycle time. For this type problem and its variants, many scholars have designed exact algorithms [9,10], heuristic algorithms [11,12], and meta-heuristic algorithms [13,14]. With the

assembly line officially put to use, some potential problems (task assignment, resource allocation, etc.) that could not be predicted in time in the design stage of the assembly line become increasingly obvious. Moreover, assembly workers' skills, product configuration, and customer demands may also change, and thus the task content and/or operation time changes and the original balancing plan needs to be further optimized. The second type of balance problem occurs in this stage, which is the problem of how to obtain a better cycle time by optimizing task assignment under the premise of definite workstations [15–17]. However, the above studies did not consider load balancing, which is the initial goal of balance research [5] and the source of maintaining the stable and efficient operation of the assembly line [18,19].

Considering the importance of load balancing, some scholars have specifically studied this kind of problem and defined it as the third type of balance problem—that is, how to optimize task assignment and balance the load of all stations as much as possible under the premise of fixed cycle time and line length. Ozcan and Toklu [20] used the assembly line smoothness index to measure load balancing and combined it with assembly line efficiency as optimization objectives to build a mathematical model of TAL load balancing. The minimum deviation method (MDM) was used to combine multi-objective problems into single-objective problems, and a tabu search algorithm was proposed. Lan and Yee [21] designed a nonlinear integer programming model to maximize the smoothness of the line for the third-type TALBP and solved it using Lingo. Purnomo and Wee [22] designed a harmonic search (HS), combining non-inferior sorting rules for TALBP considering zone constraints. The goal was to optimize the production efficiency and balance the load of stations. Li et al. [23] used an improved version of teaching and learning optimization (ITLBO) to balance multi-constraint and multi-objective TAL. The goals were the optimization of assembly line efficiency, assembly line smoothness, and total associated unit production costs. Li et al. [24] proposed an iterative local search algorithm (ILS) to realize the load balancing of TALs. A heuristic algorithm was applied to obtain the better initial population; the local search and disturbance factors were used iteratively until a local optimal solution was found. This algorithm also applies the priority decoding method based on the combination of orientation selection rules and task selection rules to reduce the load difference between the left and right stations of a workstation as far as possible so as to ensure that the sequence-related waiting time is reduced to a certain extent. Wu et al. [25] used a hybrid algorithm based on variable neighborhood search and gravity search for the TALBP considering zone constraints to achieve load balancing among workstations and reduce the series-dependent completion time of tasks as much as possible. Buyukozkan et al. [26] provided the dictionary bottleneck hybrid TALBP, balancing the load of all workstations by gradually minimizing the weighted load of the workstations with the maximum load and finally achieving the load balancing of all workstations on the assembly line. Yadav and Agrawal [27] measured load balancing by the idle time length on the workstations and established the related mathematical model. A branch and bound algorithm was programmed in the Lingo solver for this problem, and load balancing schemes of various benchmark problems were explored. After that, the workload maximization of stations was used as the measure of load balance [28], an exact algorithm was designed, and its effectiveness was verified through an engineering project. Abdullah Make and Ab Rashid [29] carried out a study on load balancing of TALs in automobile assembly workshops and designed a particle swarm optimization algorithm. To obtain a better task assignment scheme, in addition to cycle time and stations, the influences and constraints of workers' skill levels, tools, and equipment on assembly task assignment were also considered.

To sum up, current research on the load balancing of TALs is still mainly focused on the optimization design and solution of the traditional algorithm (i.e., exact, heuristic, and meta-heuristic algorithms), and as far as we know, there is no research on load balancing-oriented TALBP based on deep reinforcement learning, which is more suitable for complex and variable manufacturing processes. In addition, for multi-objective optimization, most of them are converted into single objectives by mathematical programming methods. The

operation involved weighted coefficient or unified dimensions; that is, the search direction is delimited artificially, which reduces the search space and cannot guarantee the optimality of the solution.

2.2. Deep Reinforcement Learning

The reinforcement learning algorithm (RLA) could be used to obtain optimal strategies for sequencing decision problems [30]. As shown in Figure 2, the agent of RLA interacts with the environment continuously, observes the environment state s_t, makes a decision action a_t, and obtains the feedback (reward) r_t from the environment, then adjusts the strategy according to the feedback information from the environment so that the subsequent output decision action can meet the expectation.

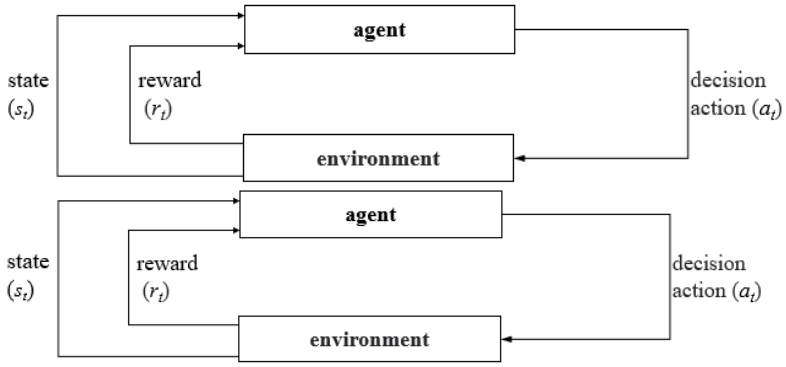


Figure 2. Solving process of reinforcement learning.

The deep learning algorithm (DLA) is a method by which to gradually acquire the whole picture of things through multi-level learning and abstracting from simple to complex [31]. The deep reinforcement learning algorithm (DRLA) generated by the combination of reinforcement learning (RL) and deep learning (DL) shows strong data processing and environment interaction abilities in terms of self-adaptation and self-learning, has rapid decision-making abilities combined with offline training and online decision-making, and highly versatile generalization ability, which can better solve complex problems [32].

In 2015, DeepMind [33] proposed the first DRLA—the Deep Q-learning Network (DQN). It combined deep neural networks with reinforcement learning and applied them to the research of games, Go (i.e., weiqi) and other fields, achieving impressive results. Since then, research on various deep reinforcement learning methods has been rapidly carried out, and the applications have expanded from games to other fields [34–36]. At present, for combinatorial optimization, deep reinforcement learning algorithms have already penetrated into the classic travel salesman problem [37], the path optimization problem [38], the packing problem [39], the maximum cutting problem [40], and other operational research problems. At the same time, some scholars, aiming to solve practical production problems, have also begun to introduce the deep reinforcement learning algorithm into the research of manufacturing problems [41]. In the area of AL balancing, Li et al. [6] focused on the research of balancing AL in the digital domain, and designed a DRLA with the support of deep deterministic policy gradient (DDPG) to enhance the operation and simulation effect of the assembly line digital twin model. Lv et al. [7] combined the sequencing problem with the assembly line balance problem and proposed a new version of DRLA on the basis of DDPG, in which an iterative interaction mechanism between task assembly time and station load were designed to achieve production task sequencing and worker allocation layer by layer. The objective was minimizing the work overload.

Although there have been some preliminary achievements in solving combinatorial optimization problems by DRLA, to the best of our knowledge, there are no studies of the deep reinforcement learning algorithm for the TALBP so far.

3. Mathematical Model for Load Balancing-Oriented TALBP

3.1. Assumptions

(1) The assembly line task assignment scheme is available, and the specific task assignment is known;

(2) The product variety is unique, and the processes are determined;

(3) Cycle time is deterministic;

(4) Execution time of takes and precedence relationship between tasks are known;

(5) Buffers, parallel stations and tasks are not considered.

3.2. Parameters

ns: Number of workstations.

nm: Number of stations.

I: Task set, $I = \{1, 2, \ldots, i, \ldots, m\}$.

J: Workstation set, $J = \{1, 2, \ldots, j, \ldots, n\}$.

(j,k): the specific station of the workstation j, i.e., the operating orientation of the station, k = 1, represents left station, k = 2, represents right station.

A_L: Task set that can only be assembled in the left station, $A_L \subset I$.

A_R: Task set that can only be assembled in the right station, $A_R \subset I$.

A_E: Task set that can be assembled in both left and right stations, $A_E \subset I$.

$P(i)$: Task set that contains all immediate precedence tasks of task i.

$P_a(i)$: Task set that contains all precedence tasks of task i.

$S(i)$: Task set that contains all immediate successor tasks of task i.

$S_a(i)$: Task set that contains all successor tasks of task i.

P_C: Set of tasks without precedence tasks.

$C(i)$: Task set opposite to operating orientation of task i. $C(i) = A_L, i \in A_R; C(i) = A_R, i \in A_L; C(i) = \Phi, i \in A_E$.

$K(i)$: Set of operating orientation indication of a task i. $K(i) = \{1\}, i \in A_L; K(i) = \{2\}, i \in A_R; K(i) = \{1,2\}, i \in A_E$.

t_i^s: Start time of the task i.

t_i^f: Finish time of the task i.

t_i: Operation time of the task i, $t_i = t_i^f - t_i^s$.

ct: Cycle time.

μ: A constant with a larger value.

$x_{ijk} = \{0,1\}$: If the task i is assigned to the workstation (j,k), the value is 1, otherwise it is 0.

$z_{ip} = \{0,1\}$: If task i and task p are assigned to the same workstation, the task i is assigned earlier than task p, the value is 1, otherwise 0.

3.3. Mathematical Model

Equations (1)–(3) are the objective functions, and their calculation formulae are shown in Equations (4)–(6). $ST_{jk} = \sum_{i \in S_{jk}} t_i$, is the total working time of tasks which are assigned to station (j,k), and $ST_{max} = max\{ST_{jk}\}$ is the maximum one of them. $Ct(j,k)$ represents the completion time of station (j,k), and $Ct_{max} = max\{Ct(j,k)\}$ is the maximum thereof. Equation (7) indicates that any task can only be assigned to one station. Equations (8) and (9) show cycle time constraints—that is, the completion time of each station must be less than cycle time. Equation (10) represents the precedence constraint. Equations (11)–(13) represent sequence-dependent constraints. Equations (14)–(17) give the definition of each variable.

$$\max LE \tag{1}$$

$$\min SI \tag{2}$$

$$\min CSI \tag{3}$$

$$LE = \frac{\sum_{i=1}^{nm} t_i}{ct * nm} \times 100\% \tag{4}$$

$$SI = \sqrt{\frac{\sum_{j\in J}\sum_{k=1}^{2}(ST_{\max} - ST_{jk})^2}{nm}} \tag{5}$$

$$CSI = \sqrt{\frac{\sum_{j\in J}\sum_{k=1}^{2}(Ct_{\max} - Ct(j,k)^2)}{nm}} \tag{6}$$

$$\sum_{j\in J}\sum_{k\in K(i)} x_{ijk} = 1, \forall i \in I \tag{7}$$

$$t_i^f \le ct, \forall i \in I \tag{8}$$

$$t^f \ge t_i, \forall i \in I \tag{9}$$

$$\sum_{g\in J}\sum_{k\in K(i)} gx_{hjk} \le \sum_{j\in J}\sum_{k\in K(i)} jx_{ijk}, \forall i \in I - P_0, h \in P(i) \tag{10}$$

$$t^f - t_h^f + \mu(1 - \sum_{k\in K(h)} x_{hjk}) + \mu(1 - \sum_{k\in K(i)} x_{ijk}) \ge t_i, \forall i \in I - P_0, h \in P(i), j \in J \tag{11}$$

$$\begin{gathered} t_p^f - t_i^f + \mu(1 - x_{pjk}) + \mu(1 - x_{ijk}) + \mu(1 - z_{ip}) \ge t_p, \forall i \in I, \\ p \in \{r | r \in I - (P_a(i) \cup S_a(i) \cup C(i)), i \prec r\}, j \in J, k \in K(i) \cup K(p) \end{gathered} \tag{12}$$

$$\begin{gathered} t_i^f - t_p^f + \mu(1 - x_{pjk}) + \mu(1 - x_{ijk}) + \mu z_{ip} \ge t_i, \forall i \in I, \\ p \in \{r | r \in I - (P_a(i) \cup S_a(i) \cup C(i)), i \prec r\}, j \in J, k \in K(i) \cup K(p) \end{gathered} \tag{13}$$

$$x_{ij1} = \{0,1\}, i \in A_L, j \in J \tag{14}$$

$$x_{ij2} = \{0,1\}, i \in A_R, j \in J \tag{15}$$

$$x_{ijk} = \{0,1\}, i \in A_E, j \in J \tag{16}$$

$$z_{ip} = \{0,1\}, \forall i \in I, p \in \{r | r \in I - (P_a(i) \cup S_a(i) \cup C(i)), i \prec r\} \tag{17}$$

4. Deep Reinforcement Learning Algorithm Based on DPPO and CNN (DPPO–CNN)

4.1. DPPO–CNN Agent

The architecture of DPPO–CNN with distributed multiple processes is shown Figure 3; the main process and m subprocess are turned on simultaneously. The main process is responsible for network training and update (gradient calculation and update), while the subprocess is only responsible for data acquisition without gradient calculation. The main process includes the experience pool and the main Actor–Critic network, and each subprocess includes the child Actor–Critic network. The Actor network is used to make task assignment decisions a_t based on the environmental status s_t of the TAL, while the

Critic network is used to evaluate the quality of task allocation decisions a_t. Actor–Critic network structures are shown in Figures 4 and 5.

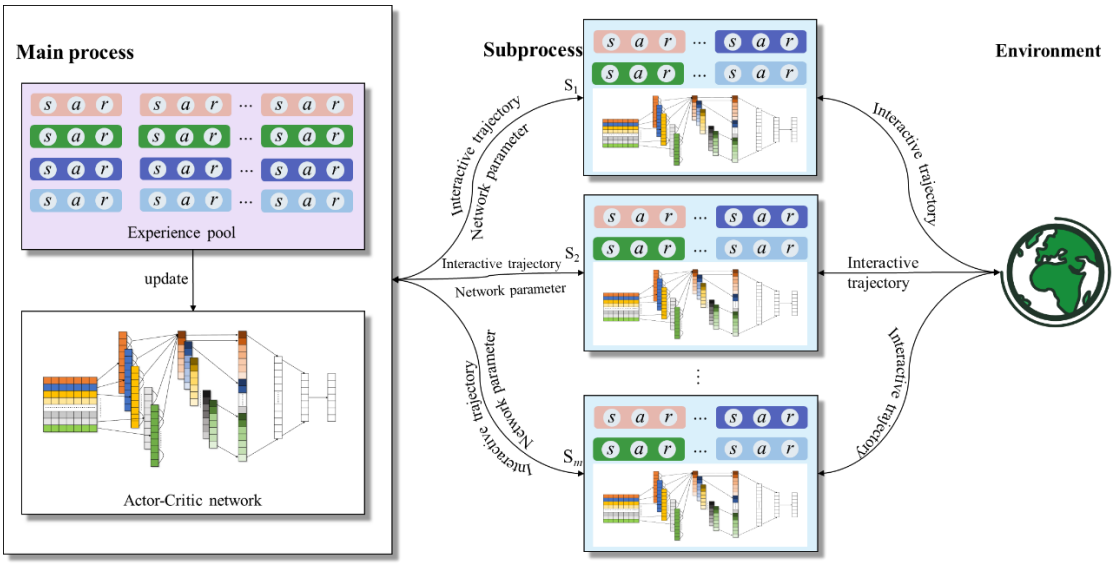


Figure 3. The architecture of DPPO–CNN.

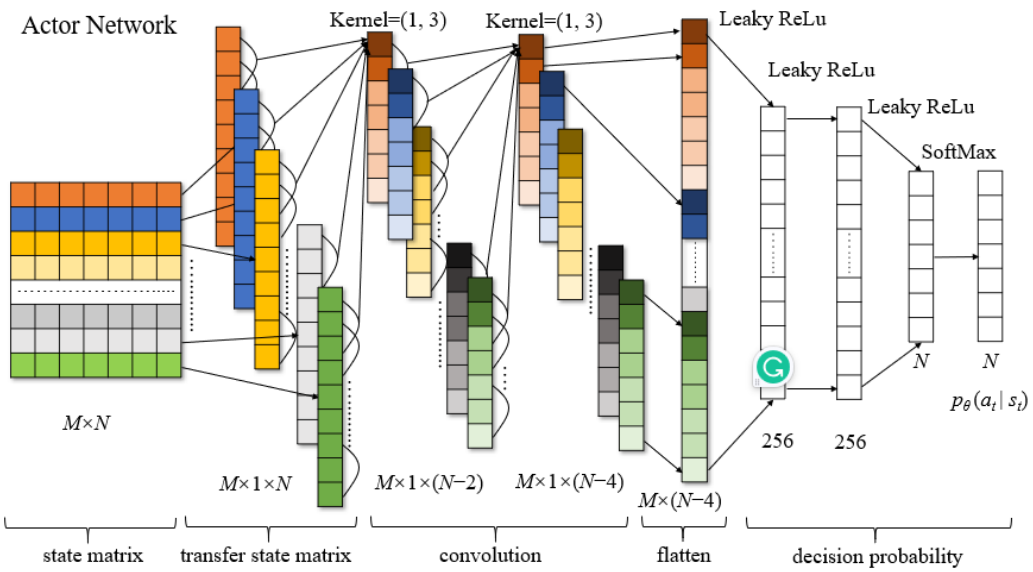


Figure 4. Actor network.

(1) Initialize the main Actor–Critic network of the main process and send its parameters to each subprocess through an orbit.

(2) The child Actor–Critic network of the subprocess loads the main Actor–Critic network and then interacts with the environment and transmits interactive trajectory (state matrix s, task assignment decisions a, reward function r) to the main process through an orbit.

(3) The main process stores the interaction experience of all subprocesses in the experience pool. When the amount of experience stored in the experience pool exceeds the capacity of the experience pool, it is packaged as a training set to train the main Actor–Critic network.

(4) Transfer the updated main Actor–Critic network to each subprocess again and go back to (1).

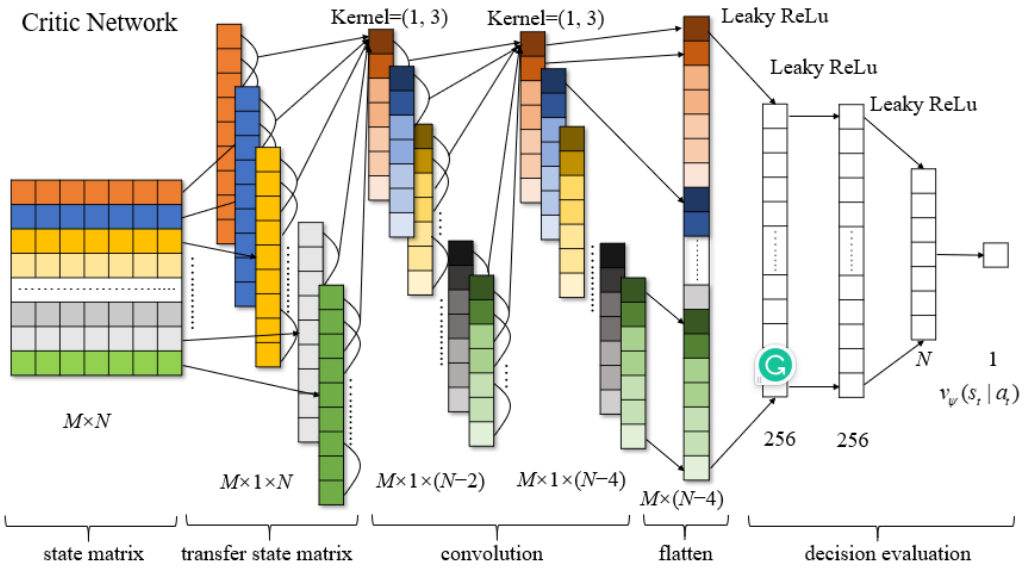


Figure 5. Critic network.

The Actor network is the strategy network, which approximates the optimal task assignment strategy $p_\theta(a_t|s_t)$ by using the neural network with parameter θ. The network structure is shown in Figure 4, including a two-layer convolutional network and a three-layer fully connected network. The dimensional matrix $M \times N$ is the input of the network at time t corresponding to the environmental status s_t. M is the number of feature vectors, and N is the number of total assembly tasks. After the matrix is input, two layers of convolution operations are carried out on it. The convolution Kernel size is as follows: Kernel = (1, 3). The feature vector obtained after convolution is flattened and input into the three fully connected layers. The number of nodes in the first two layers is 256, and the number of nodes in the last layer is N. Then, it is normalized by the SoftMax function and outputs $p_\theta(a_t|s_t)$, which is the probability of output task to be assigned a_t when the Actor policy network is in the status s_t.

The Critic network is evaluation network, which approximates the optimal strategy evaluation value $v_\Psi(s_t|a_t)$ by using the neural network with parameter Ψ. The network structure is shown in Figure 5. In this paper, the first several layers of the Critic network structure are the same as that of the Actor network structure, but the last layer is the linear regression layer; that is, the SoftMax layer in the Actor network is replaced with $v_\Psi(s_t|a_t) = f(h(t); \Psi) = \omega * h(t) + b$, where $v_\Psi(s_t|a_t)$ is the output of the Critic network at time t and $h(t)$ is the output of the previous fully connected layer. Ψ is the parameter of the internal unit node of the network, including the weight ω and the bias item b.

In each subprocess, the learning process is shown in Figure 6. The agent observes environmental status s_t of task assignment, makes the selection decision, and outputs the task to be assigned a_t and updates the status and gives back the reward r_t after the assignment of task a_t. The agent can solve the problem through continuous interaction with the environment. After each problem is solved, the trajectory of the interaction between the agent and the environment (including status, decision task, and reward) is stored in the experience pool. These dates are preprocessed to obtain training data in the experience pool. The interaction between the agent and the environment is suspended when the amount of data stored in the experience pool reaches its capacity limit. Parts of the training data are selected randomly by the agent from the experience pool, the network (task allocation strategy) is updated, and the experience is learned. The higher reward value is obtained through repeated trial and error, and eventually, the maximization of the cumulative reward and the optimization of the task allocation strategy can both be realized.

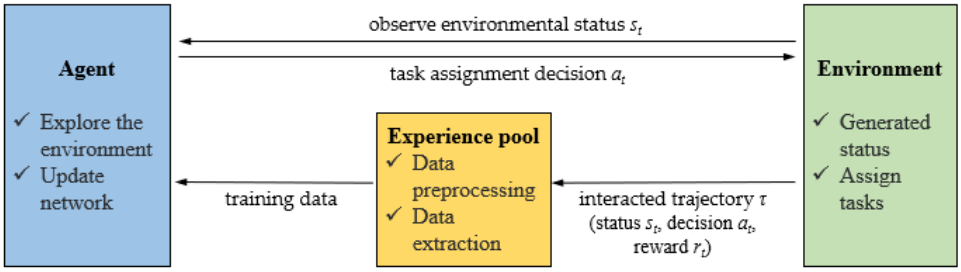


Figure 6. Learning process of each subprocess.

4.2. Task Assignment State of TAL

As shown in Table 1, there are 18 task assignment state features for the load balancing-oriented TALBP considered. Values of features 1–5 represent the related sequence numbers tasks; features 6–18 represent the overall situation of task assignment scheme of TAL; in particular, values of state features 11–16 are rounded.

Table 1. Task assignment state of TAL.

No.	Features	Description
1	PTime	A task with the longest operation time in the set of tasks without precedence tasks
2	MFlow	A task with the largest spare time caused by matched task in the set of tasks without precedence tasks
3	SucNum	A task with the largest number of successor tasks in the set of tasks without precedence tasks
4	AllSucNum	A task with the largest number of successor tasks
5	MTNum	A task with the smallest number of matched tasks in the set of tasks without precedence tasks
6	Side	The operating orientation of a task with the smallest sequence number in the set of tasks without precedence tasks (0 is E type of tasks; 1 is non-E type of tasks)
7	FAN	The number of tasks can be selected at present
8	NPN	The number of tasks without predecessors at present
9	NER	The number of remaining E type of tasks
10	NR	The number of remaining non-E type of tasks
11	LRDiffoverAE	The load difference of remaining non-E type of tasks/the average time of remaining E type of tasks
12	RToverAveptO	Remaining spare time of current station/the average time of tasks of current station
13	RToverAveptT	Remaining spare time of matched station/the average time of tasks of matched station
14	OPToverRemain	The operation time/remaining time of tasks
15	ARPW	Improved position weight
16	RRPW	Reverse position weight
17	PreNum	A task with the largest number of immediate successor tasks in the set of tasks without precedence tasks
18	AllPreNum	A task with the smallest number of precedence tasks

Original state feature information is abstracted and preprocessed by one-hot encoding to give it two-dimensional arrangement feature information, which is suitable for subsequent processing by convolutional neural network. If the number of tasks in TAL is N, the matrix with dimension $18 \times N$, as the environment state s_t, can be obtained after preprocessing.

Figure 7 shows the two types of the initial state matrix of P16 (Figure 8), and the state feature value of the initial state is shown in Table 2 (Figure 9).

Feature No. \ Task No.	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
6	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
7	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
8	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
9	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
11	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
12	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
13	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
14	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
15	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
16	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
17	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
18	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

(a)

Feature No. \ Task No.	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
6	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
7	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
8	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
9	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
11	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
12	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
13	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
14	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
15	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
16	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
17	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
18	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

(b)

Figure 7. State matrix of P16: (**a**) state matrix s_1 generated for third column of Table 2; (**b**) state matrix s_2 generated for fourth column of Table 2.

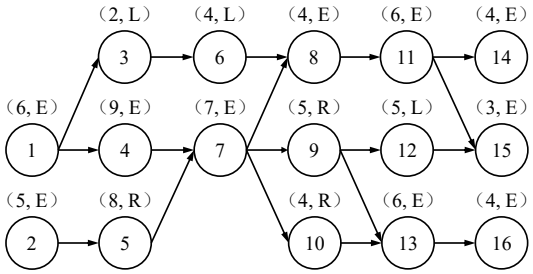


Figure 8. P16.

Table 2. Each state feature value of the initial state for P16.

No.	Features	State Feature Value (Task Assignment Scheme I)	State Feature Value (Task Assignment Scheme II)
1	PTime	1	1
2	MFlow	1	1
3	SucNum	1	1
4	AllSucNum	1	1
5	MTNum	1	1
6	ARPW	1	1
7	RRPW	1	1
8	PreNum	1	1
9	AllPreNum	1	1
10	Side	0	0
11	FAN	2	1
12	NPN	2	2
13	NER	10	9
14	NR	6	6
15	LRDiffoverAE	1	1
16	RToverAveptO	3	3
17	RToverAveptT	3	2
18	OPToverRemain	1	1

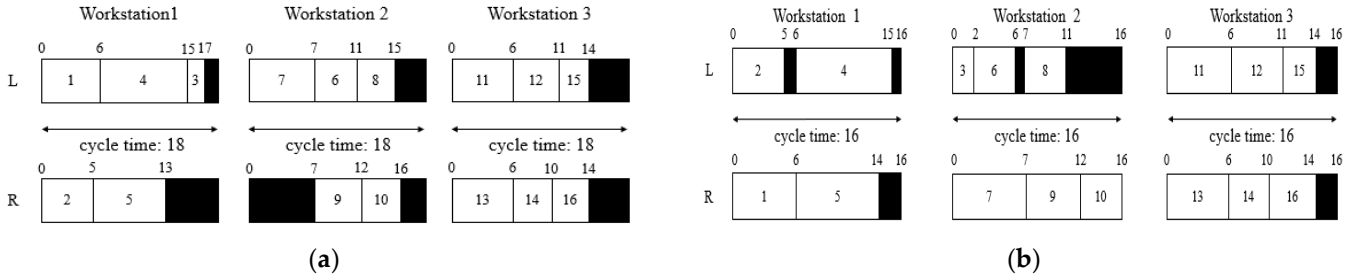


Figure 9. Task assignment schemes of P16: (**a**) task assignment scheme I; (**b**) task assignment scheme II.

4.3. Decision of Selecting Tasks

The mask layer is introduced to ensure that only the tasks that meet the constraints of precedence, operation orientation, and sequence dependence can be selected. Take the P16 problem as an example; the network parameters θ of the agent Actor strategy adopt orthogonal initialization. In the state s_1 of task assignment in the TAL environment, the output $p_\theta(a_1|s_1)$ is shown in Figure 9. At this time, if sampling is conducted according to the probability distribution, the task to be assigned is 3, which does not satisfy the precedence constraint. After processing at the mark layer—as shown in Figure 10—only task 1 and task 2 satisfy constraints, i.e., they can be selected. At this time, if sampling is conducted according to probability distribution, task 2 can be selected to be assigned.

probability	0.0604	0.0677	0.0727	0.0598	0.0653	0.0647	0.0634	0.0622	0.0650	0.0546	0.0594	0.0606	0.0607	0.0580	0.0639	0.0616
task	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
								mask								
probability	0.0604	0.0677	0.0727	0.0598	0.0653	0.0647	0.0634	0.0622	0.0650	0.0546	0.0594	0.0606	0.0607	0.0580	0.0639	0.0616
task	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16

Figure 10. Mask layer process.

4.4. Task Assignment

Task assignment procedure is shown in Figure 11 and the contents are as follows.

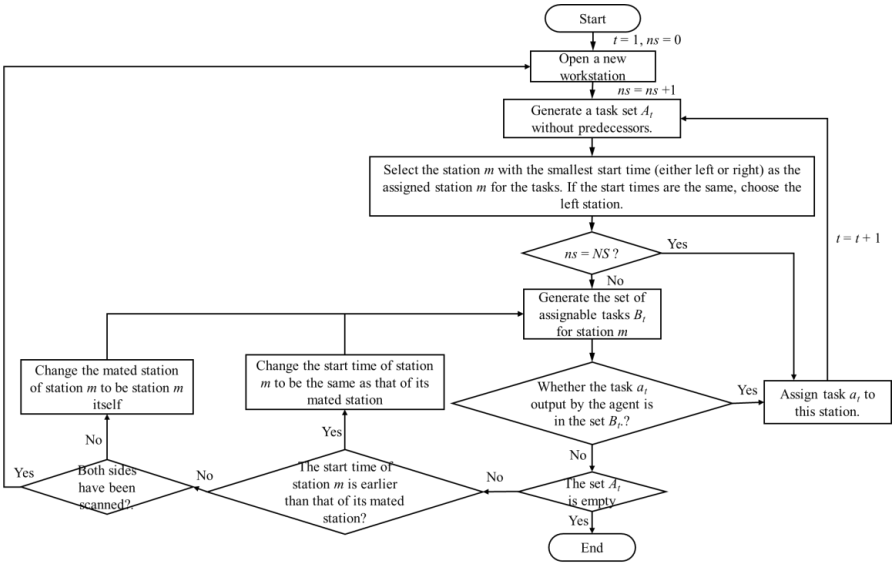


Figure 11. Task assignment.

Step 1: Initialize to generate the initial state of the TAL environment.

Step 2: Start a new workstation and set the earliest assignable task time of to 0.

Step 3: Generate a task set A_t without precedence tasks; alternatively, all of its precedence tasks have already been assigned, according to precedence constraint.

Step 4: Select the station (left or right) with a smallest start time as the assigned station m. If the start time is the same, choose the left station.

Step 5: Check whether the current workstation is the last one. If yes, perform Step 6; otherwise, turn to Step 7.

Step 6: Generate the set of assignable tasks B_t for station m.

Generate rules are as follows: (1) select the task without real-time predecessors; (2) select the task for which the completion time of its immediate predecessors is less than the earliest task-assignable time of the station; (3) select the task whose operating orientation is the same as the operating edge of station m.

Step 7. Check whether the task output by the agent is in the set B_t. If it is, turn to Step 11; otherwise, go to Step 8.

Step 8. If A_t is an empty set, stop this algorithm; otherwise, perform Step 9.

Step 9. If the earliest task assignable time of station (t_1) is earlier than the earliest task assignable time of the mated station (t_2), $t_1 = t_2$, and return to Step 5; otherwise, perform Step 10.

Step 10. If both sides of the station have been scanned, start a new workstation and return to Step 2; otherwise, scan the other side and return to Step 5.

Step 11. Assign the selected task by DPPO–CNN agent.

Step 12. Update the number of immediate predecessors and their related completion time of all the unassigned tasks, the earliest task assignable time of station m, and the assembly line state; then, return to Step 3.

4.5. Reward Function

Reinforcement learning agents can achieve the optimal task assignment strategy $p_\theta(a_t|s_t)$ by maximizing cumulative rewards (r_{sum}) and then realize the optimization objectives. Sparse reward is adopted by the traditional deep reinforcement learning algorithm, i.e., rewards $r_1, r_2, \ldots, r_{n-1}$ are all equal to 0 until all tasks have been assigned. The environment gives feedback reward r_n to the agent; then, the accumulate reward $r_{sum} = r_n$. This is an easy way to converge the algorithm because for many tasks, the agent can obtain positive samples with a certain probability by conducting random exploration in the environment, and the positive samples occupy a relatively significant proportion of the total samples at the early stage of learning.

However, with the increase in task complexity, the probability of obtaining positive samples through random exploration becomes small, and the sparse feedback signals cannot indicate the exploration direction for the agent. The algorithm will be difficult to converge or the convergence speed will be very slow. To overcome this problem, it is necessary to add other reward items or punishment items to make the reward function become dense, and to guide the agent to explore the environment more efficiently [39]. In this paper, for objective *LE*, if both the operation time of the tasks and the number of stations are determined, the smaller the *ct*, the higher the *LE*. For objective *SI*, if the workloads of tasks which are assigned to a station are substantially equal to cycle time of that station, the difference between the $ST_{\max}$ and the ST_i will be small, and so will the *SI*. Similarly, for objective *CSI*, if the completion time of a station is substantially equal, the value of *CSI* is small, which means the operation of the assembly line is stable and balanced. Therefore, the values of SI and *CSI* will decrease if the idle time of each station is reduced.

There are two types of idle time that can be generated on a two-sided assembly line due to its parallel structure and the sequence-dependent relationship of its tasks, as shown in Figure 12: (1) End idle time—If the completion time of the current station will exceed the cycle time when the new task is assigned to the current station, the new task has to be assigned to the next station. In this case, the completion time of the current station will be

less than the cycle time, and the time difference between the completion time of the current station and cycle time is the end idle time. As shown in Figure 12, on workstation 1, after task 5 is completed, task 4 needs to be assigned, but if the operation time of task 4 is 9, whether it is assigned to the left or right stations of workstation 1, the completion time will exceed cycle time 18; therefore, task 4 has to be assigned to workstation 2, and there has to be idle time at the end of the left and right stations of workstation 1. (2) Sequence-dependent idle time—If sequence-related tasks are assigned to the left and right stations of the same workstation, due to the absolute sequence constraint between them, this may lead to a situation in which some tasks may have to sit idle and wait, and this kind of idle time inside stations is called sequence-dependent idle time. As shown in Figure 12, on workstation 2, task 7, assigned to right station, can not be started from time 0 because its immediate precedent, task 4, is assigned to left station of the same workstation. Task 7 has to wait before task 4 is completed; therefore, there are nine time unitsof sequence-dependent idle time before task 7, which is caused by task 4.

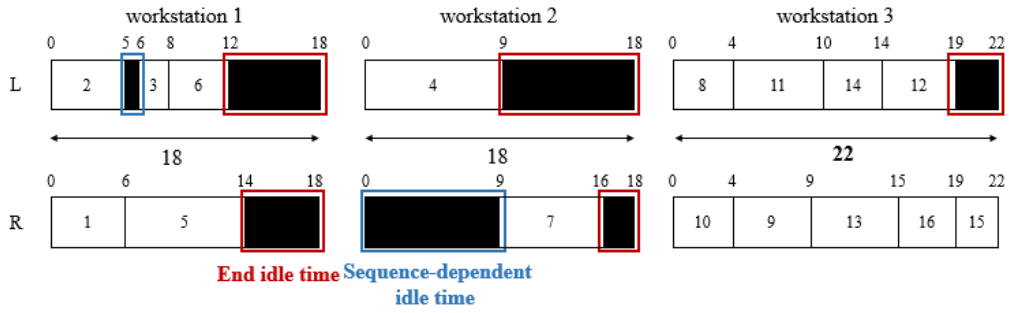


Figure 12. A task assignment scheme of P16.

Moreover, the logic of task assignment in this paper may cause the cycle time of the last workstation to be much larger than that of all other workstations, as shown in Figure 12. Due to fewer tasks being assigned to the first two stations, more tasks are assigned to the last workstation (3), resulting in the overloading of the completion time of workstation 3. Therefore, the difference between the actual cycle time of the last station (i.e., its completion time) and the ideal cycle time should be controlled.

Under these conditions, the reward function is set as follows during the task assignment process:

(1) If a task is the last task at the current station and there is end idle time at the current station, the reward function r is calculated as Equation (18):

$$r = -\left(end\ idle\ time\ of\ the\ current\ station\right); \tag{18}$$

(2) If there is idle time in the station caused by a between thesequence-dependent relationship of tasks, i.e., sequence-dependent idle time, the reward function r is calculated as Equation (19):

$$r = -\left(sequence - dependent\ idle\ time\right); \tag{19}$$

(3) If all tasks have been assigned in the last station, the reward function r is calculated as Equation (20):

$$r = -\left(\frac{le}{LE} + \frac{SI}{si} + \frac{CSI}{csi} + ct_{actual} - ct_{ideal}\right), \tag{20}$$

where ct_{actual} represents the actual cycle time of the last station (i.e., its completion time) and ct_{ideal} represents the ideal cycle time;

(4) The other state is equal to 0, as shown in Equation (21):

$$r = 0. \tag{21}$$

4.6. Overall Flow of DPPO–CNN

The overall flow of the proposed DPPO–CNN for TALBP-BL (Algorithm 1) is as follows.

Algorithm 1 Load balancing-oriented TALBP based on DPPO–CNN

1: Initializes the Actor–Critic network parameters θ, φ for the main process, maximum iteration number, experience pool capacity buffer size, maximum experience pool capacity max buffer size, batch data size, number of network updates of each round epoch; Subprocess Actor–Critic network parameters θ, φ.
2: **for** each *episode* **do**:
3: $t = 1$.
4: The main process empties the experience pool, sends its Actor–Critic network parameters θ, φ to each child process; each sub-process bilateral assembly line environment 1, 2, 3, ... , m is initialized, generates states S_t^1, S_t^2, S_t^1, ... , S_t^m; each agent 1, 2, 3, ... , m loads the network parameters of the main process.
5: **while** *buffer size* < *max buffer size* **do:**
6: **while** all tasks of m's respective bilateral assembly lines have not been allocated **do**:
7: Agent 1, 2, 3, ... , m observes the environment status S_t^1, S_t^2, S_t^1, ... , S_t^m, respectively, and takes the tasks a_t^1, a_t^2, a_t^1, ... , a_t^m to be assigned according to strategy $p_\theta(a_t|s_t)$.
8: Environment 1, 2, 3, ... , m allocates tasks a_t^1, a_t^2, a_t^1, ... , a_t^m separately, and feedback reward r_t^1, r_t^2, r_t^1, ... , r_t^m.
9: $t = t + 1$.
10: Environment 1, 2, 3, ... , m update states S_t^1, S_t^2, S_t^1, ... , S_t^m.
11: **end while**
12: Agent 1, 2, 3, ... , m stores the interactive trajectory (solving experience) τ_1, τ_2, τ_3, ... , τ_m, respectively, in the experience pool and packages it as training data.
13: **end while**
14: **for** epoch in {1, 2, ... , *epochs*} **do:**
15: Randomly extract the training data with the size of batch size from the experience pool.
16: Calculate the network loss function of the actor strategy of the master process; evaluate the critic loss function of the master process.
17: Update the main process Actor's policy network $p_\theta(a_t|s_t)$.
18: Update the main process Critic's evaluation network $v_\varphi(s_t, a_t)$.
19: **end for**
20: $\theta_{old}, \varphi_{old} \leftarrow \theta, \varphi$.
21: **end for**

5. Experimental Verification and Discussions

5.1. Implementation of the DPPO–CNN

In this paper, 59 instances of the benchmark problem (P9 [42] (Figure A1), P12 [42] (Figure A1), P16 [6] (Figure 8), P24 [42] (Figure A1), P65 [6] (Figure A2), P148 [5,6] (Table A1), and P205 [6] (Table A2)) are utilized to test the performance of the proposed DPPO–CNN. In addition, the DPPO–CNN algorithm is programmed in Python 3.6 and runs on a personal computer with Ubuntu 20.04LTS, 2.90 GHz CPU frequency, and the 16 g memory.

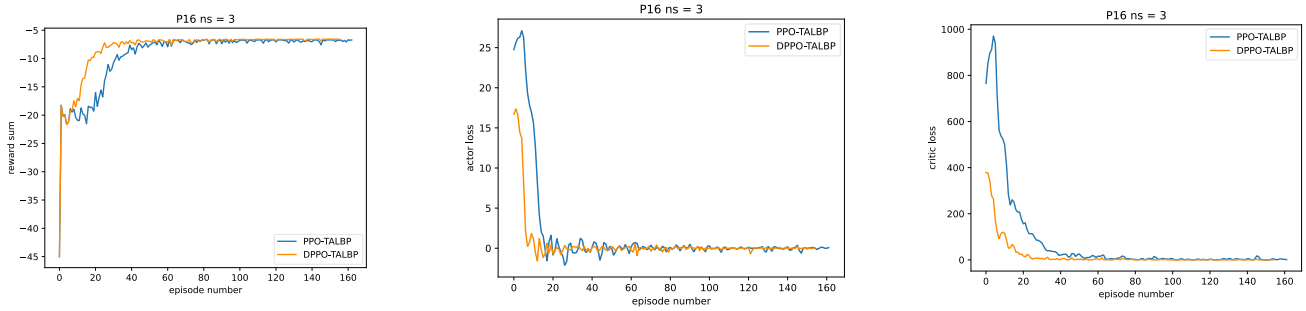
The parameters of DPPO–CNN are mainly decided according to the empirical values and the actual data of the agent interaction process, as listed in Table 3.

Table 3. Parameters of DPPO–CNN.

Parameter	Value	Parameter	Value
Number of hidden layers on the Actor and Critic network	256	Learning rate of the Actor network	10^{-4}
Activation function of hidden layers on the Actor and Critic network	Leaky Relu	Learning rate of the Critic network	2×10^{-4}
Activation function of output layers on the Actor network	Softmax	Sample size	256
Activation function of output layers on the Critic network	Leaky Relu	Maximum capacity of the experience pool	4096
Convolution kernel of convolution layers on the Actor and Critic network	(1, 3)	Study times per round	8
Initialize network parameters	Orthogonal initialization	Return discount factor	0.99
Optimizer	Adam	Cutting coefficient	0.2

5.2. Verification of DPPO–CNN in Term of Model Training

To verify the performance of distributed multiple processes of DPPO–CNN, the model training results of DPPO–CNN are compared with the deep reinforcement learning with single process, named the PPO–CNN, which uses the same parameters, state matrix, and reward function. To clarify the effect of distributed multiple processes, P16 is used as the representative of small-scale cases and P65 as the representative of large-scale cases for detailed explanation, as shown in Figures 13 and 14.



(**a**) (**b**) (**c**)

Figure 13. Comparison between DPPO–CNN and PPO–CNN in term of model training (P16): (**a**) cumulative reward curves; (**b**) loss curves of the Actor network; (**c**) loss curves of the Critic network.

Figure 13a shows the cumulative reward curves of DPPO–CNN and PPO–CNN for P16; we can see that: (1) the cumulative rewards of both DPPO–CNN and PPO–CNN are increasing gradually, and they converge after 60 rounds of training; this verifies the effectiveness of our algorithm indirectly; (2) the cumulative reward curve of DPPO–CNN algorithm is rising faster than that of PPO–CNN algorithm, which shows that DPPO–CNN is better than PPO–CNN; (3) the gap is not obvious because of P16 is relatively simple and both algorithms can obtain a good solution strategy quickly. However, from the related results of large-scale case P65 (Figure 14a,b), the advantages of DPPO–CNN are very prominent; the convergance of PPO–CNN needs 1200 rounds of training, whereas that of DPPO–CNN only needs 600 rounds. The study concludes that the distributed architecture designed in this paper is helpful in solving large-scale cases. A similar conclusion can be also obtained from the comparison between DPPO–CNN and PPO–CNN in terms of loss curves in P16 and P65, respectively, as shown in Figure 13b,c and Figure 14c,d.

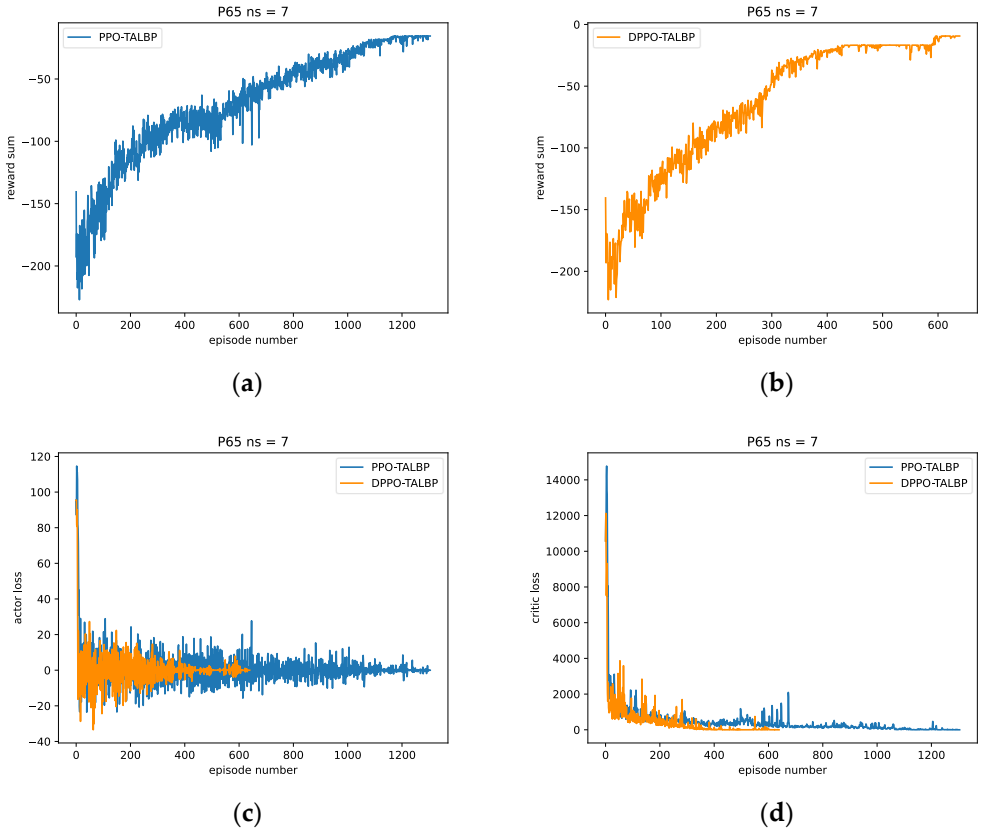


Figure 14. Comparison between DPPO–CNN and PPO–CNN in term of model training (P65): (**a**) cumulative reward curve of PPO–CNN; (**b**) cumulative reward curve of DPPO–CNN; (**c**) loss curves of the Actor network; (**d**) loss curves of the Critic network.

5.3. Verification of DPPO–CNN in Term of Solutions

The trained Actor–Critic network model of the DPPO–CNN agent is saved and utilized to solve the load balancing-oriented TALBP of all 59 instances with different scales. Both DPPO–CNN and PPO–CNN algorithms are ran 20 times for problem instance and the best results are record and exhibited in Table 4.

As can be seen from Table 4, the solution of the DPPO–CNN algorithm is superior to that of the PPO-TALBP algorithm in 49 cases out of all 59 test cases, which shows the absolute advantage of DPPO–CNN. Furthermore, (1) both DPPO–CNN and PPO–CNN perform well in solving the load balancing-oriented TALBP, especially in small-scale cases (P9, P12, P16, and P24), which shows that the main architecture of the deep reinforcement learning algorithm combining distributed proximal policy optimization (DPPO) and the convolutional neural network (CNN) is tenable; (2) DPPO–CNN has outstanding performance in solving large-scale cases (P65, P148, and P205). Take P148 with cycle time 255 as an example, LE = 91.34%, SI = 34.17, and CSI = 26.78 obtained by PPO–CNN, while LE = 99.53%, SI = 1.98, and CSI = 1.98 obtained by DPPO–CNN. Obviously, both SI and SCI decrease significantly, which means the load is more balanced. In addition, through calculation, it is found that in large-scale cases (P65, P148, and P205) that the solutions obtained by DPPO–CNN have an average increase of 7.89% in LE, an average decrease of 76.71% in SI, and an average decrease of 74.92% in CSI compared to the solutions obtained by PPO–CNN. (3) Although both DPPO–CNN and PPO–CNN perform excellently in terms of calculation time, the solution speed of DPPO–CNN is significantly better than that of PPO–CNN. For example, each kind of P65 instance can be resolved by DPPO–CNN within 0.04 s, while PPO–CNN resolves them 0.1 s; the calculation time here refers to the online solving time of the instance after the training is complete. Considering the fact that model training time makes up the majority of the running time of

deep reinforcement learning algorithms, the comparison of the offline training time of the proposed DPPO–CNN and PPO–CNN is shown in Table 5. We can see that compared with PPO–CNN, DPPO–CNN also performs better in terms of offline training time.

Table 4. Result comparison.

Instance	*ct*	*ns*	PPO-CNN				DPPO–CNN			
			LE	*SI*	*CSI*	*Time(s)*	*LE*	*SI*	*CSI*	*Time(s)*
P9	3	3	94.44%	0.41	0.41	0.004	94.44%	0.41	0.41	0.001
	4	3	85.00%	1.87	1.87	0.004	94.44%	0.41	0.41	0.001
	5	2	85.00%	1.12	1.12	0.004	85.00%	1.12	1.12	0.001
	6	2	70.83%	2.50	1.58	0.005	85.00%	1.12	1.12	0001
	7	2	60.71%	3.57	3.57	0.005	85.00%	1.12	1.12	0.001
P12	4	4	78.13%	1.37	1.37	0.007	78.13%	1.37	1.37	0.002
	5	3	83.33%	1.23	0.91	0.007	83.33%	1.23	0.91	0.002
	6	3	83.33%	2.97	2.97	0.008	83.33%	1.23	0.91	0.002
	7	2	89.29%	1.12	1.12	0.008	89.29%	1.12	1.12	0.003
	8	2	78.13%	1.94	1.23	0.008	89.29%	1.12	1.12	0.003
	9	2	69.44%	3.35	1.87	0.009	89.29%	1.12	1.12	0.003
P16	15	4	78.095%	7.18	7.09	0.009	78.095%	7.18	7.09	0.003
	16	3	85.42%	2.94	2.52	0.009	85.42%	2.94	2.52	0.004
	18	3	75.93%	4.97	2.55	0.010	85.42%	2.94	2.52	0.004
	19	3	71.73%	5.86	2.38	0.011	85.42%	2.94	2.52	0.006
	20	3	68.33%	6.81	2.83	0.011	85.42%	2.94	2.52	0.006
	21	3	65.08%	8.56	6.18	0.012	85.42%	2.94	2.52	0.007
	22	2	93.18%	1.87	1.58	0.013	93.18%	1.87	1.58	0.007
P24	18	4	97.22%	0.71	0.61	0.014	97.22%	0.71	0.61	0.009
	20	4	87.5%	2.92	2.03	0.015	97.22%	0.71	0.61	0.009
	24	3	97.22%	1.00	0.71	0.017	97.22%	1.00	0.71	0.010
	25	3	95.33%	2.52	2.52	0.017	97.22%	1.00	0.71	0.010
	30	3	77.78%	7.92	5.21	0.020	97.22%	1.00	0.71	0.013
	35	2	100%	0.00	0.00	0.020	100%	0.00	0.00	0.014
	40	2	87.50%	6.82	2.55	0.020	100%	0.00	0.00	0.014
P65	326	8	97.76%	9.99	9.94	0.076	98.36%	8.79	8.79	0.024
	381	7	95.59%	27.42	25.03	0.079	98.98%	6.10	6.10	0.027
	435	6	97.68%	12.37	11.58	0.083	99.28%	4.59	4.59	0.030
	490	6	86.72%	102.11	40.07	0.087	99.28%	4.59	4.59	0.032
	512	5	99.59%	2.70	2.70	0.092	99.59%	2.70	2.70	0.037
	544	5	93.73%	43.76	36.76	0.100	99.59%	2.70	2.70	0.038
P148	204	13	96.61%	9.52	9.52	0.175	99.53%	1.66	1.66	0.094
	228	12	93.64%	22.61	17.38	0.181	99.30%	2.43	2.43	0.097
	255	11	91.34%	34.17	26.78	0.193	99.53%	1.98	1.98	0.101
	306	9	93.03%	32.58	20.76	0.211	99.35%	2.08	2.08	0.113
	357	8	89.71%	57.34	52.67	0.242	99.46%	2.18	2.18	0.118
	378	7	96.83%	20.44	5.09	0.256	99.73%	2.00	2.00	0.123
	408	7	89.71%	72.79	31.40	0.263	99.73%	2.00	2.00	0.127
	454	6	94.05%	40.96	13.82	0.271	99.77%	1.58	1.58	0.136
	459	6	93.03%	41.95	33.69	0.281	99.77%	1.58	1.58	0.144
	510	6	83.73%	133.08	128.98	0.297	99.77%	1.58	1.58	0.149
P205	1133	11	93.66%	103.255	76.98	0.761	98.44%	24.44	16.49	0.391
	1275	10	91.55%	157.90	108.27	0.783	98.34%	26.75	19.20	0.420
	1322	9	98.11%	35.00	29.71	0.789	98.70%	29.50	12.09	0.438
	1455	9	89.14%	202.97	119.14	0.821	98.70%	29.50	12.09	0.445
	1510	8	96.63%	63.64	34.84	0.843	98.85%	25.94	17.83	0.461
	1650	8	88.43%	265.00	96.17	0.855	98.85%	25.94	17.83	0.474
	1699	7	98.15%	46.30	19.52	0.857	98.79%	30.93	13.99	0.480
	1888	7	88.32%	374.85	253.90	0.864	98.79%	30.93	13.99	0.499
	1920	7	86.85%	447.20	371.80	0.869	98.79%	30.93	13.99	0.502
	2077	6	93.67%	197.45	121.67	0.888	98.90%	36.08	11.42	0.518
	2100	6	92.64%	205.35	157.83	0.891	98.90%	36.08	11.42	0.523
	2266	6	85.85%	459.64	395.98	0.907	98.90%	36.08	11.42	0.542
	2300	6	84.58%	516.95	456.03	0.912	98.90%	36.08	11.42	0.549
	2454	5	95.13%	219.16	63.68	0.920	99.08%	34.69	11.02	0.558
	2500	5	93.38%	251.86	112.88	0.944	99.08%	34.69	11.02	0.565
	2643	5	88.33%	419.17	277.99	0.952	99.08%	34.69	11.02	0.577
	2800	5	83.38%	683.82	659.15	0.981	99.08%	34.69	11.02	0.582
	2832	5	82.43%	735.43	711.63	0.992	99.08%	34.69	11.02	0.589

Table 5. Comparison of the offline training times of DPPO–CNN and PPO–CNN.

Instance	Training Time (h)	
	DPPO–CNN	PPO-CNN
P9	0.15	0.10
P12	0.20	0.12
P16	0.35	0.20
P24	0.50	0.35
P65	1.00	0.68
P148	1.50	0.95
P205	4.00	2.34

6. Conclusions and Future Work Avenues

In this article, the load balancing-oriented TALBP has been studied and a deep reinforcement learning algorithm combining distributed proximal policy optimization (DPPO) and convolutional neural network (CNN) has been presented. To the best of our knowledge, this is the first attempt to solve the load balancing-oriented TALBP based on deep reinforcement learning. A mathematical model with objectives of optimizing line efficiency (*LE*), smoothness index (*SI*), and completion time smoothness index (*CSI*) is provided. In the proposed deep reinforcement learning algorithm, distributed multiple processes have been proposed to improve the search speed and capability of solutions. A total of 18 task assignment state features for the load balancing-oriented TALBP environment have been considered, which ensure that the agent can obtain more useful information from the environment and perform optimal selection as completely as possible, and different reward functions according to objectives and their implicit information have been proposed to guide good solution direction. Fianlly, the performance of the proposed algorithm has been verified on all scales of benchmark instances via a comparison with the single-process deep reinforcement learning algorithm in terms of model training and solution results.

Although the proposed algorithm performs better in this study, there is still the possibility of improvement; for example, we could directly obtainin the Pareto-optimal solution set using the deep reinforcement learning algorithm and/or extend the application of the proposed algorithm to other versions of the TALBP or other types of assembly lines, such as linear and parallel assembly lines, as well as similar production planning problems, such as the two-sided disassembly line balancing problem [43], the assembly sequence problem [44], and so on.

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

Funding: This research is funded by China National Heavy Duty Truck Group Co., Ltd., (Project No. 22H010100926), and New young teachers research start-up Fund of Shanghai Jiao Tong University (Project No. 22X010503668).

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: The data presented in this study are available on request from the corresponding author.

Acknowledgments: The authors are grateful to the anonymous reviewers, whose valuable comments and suggestions helped a lot to improve this paper, and to the editors for their kind and sincere support.

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

Appendix A

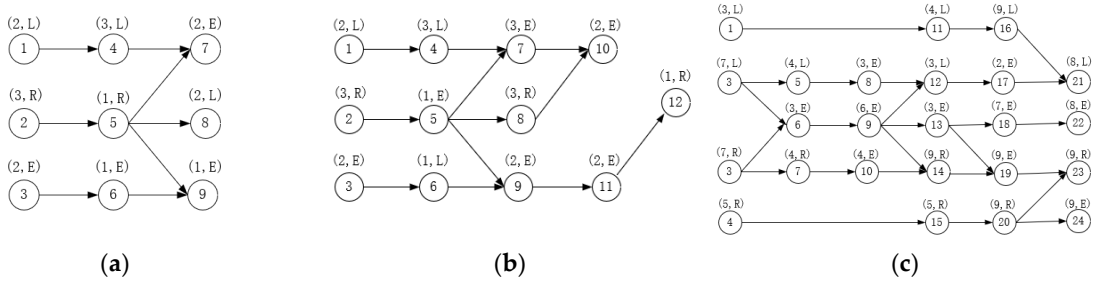


Figure A1. Small-scale cases: (**a**) P9; (**b**) P12; (**c**) P24.

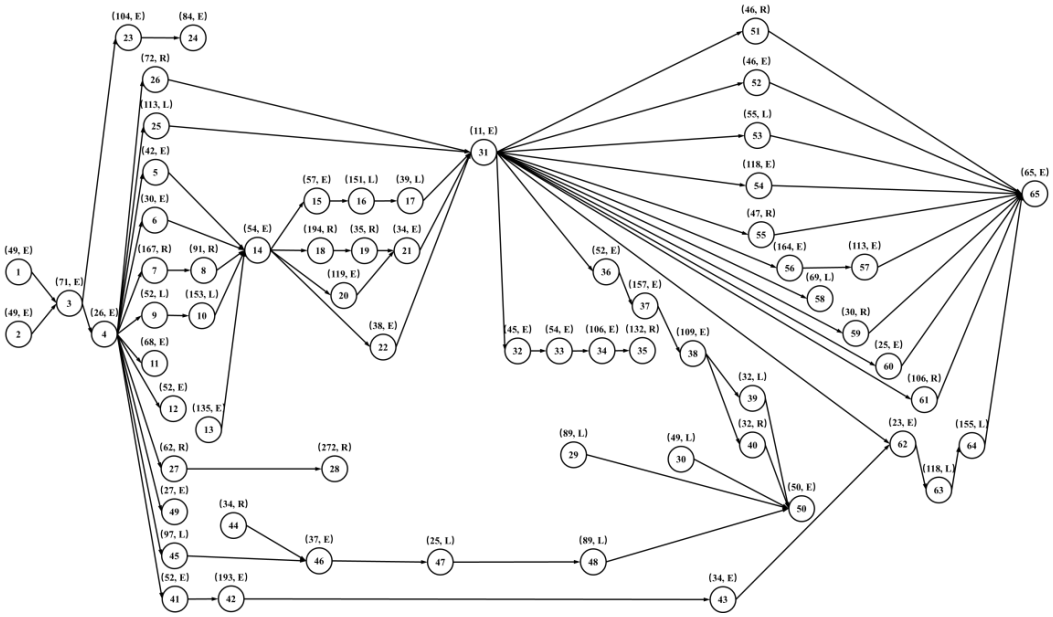


Figure A2. P65.

Table A1. P148.

NO.	Time	Side	Immediate Sucessors	NO.	Time	Side	Immediate Sucessors
1	16	E	5, 6, 7, 8	75	101	E	88, 97
2	30	E	3	76	5	E	77
3	7	E	4, 5, 6, 7	77	28	E	78
4	47	E	8	78	8	E	79
5	29	E	14	79	111	E	80
6	8	E	9	80	7	E	81
7	39	E	14	81	26	E	106
8	37	E	10	82	10	E	83, 89, 143, 146
9	32	E	14	83	21	E	
10	29	E	14	84	26	E	85
11	17	E	12	85	20	E	
12	11	E	13	86	21	E	
13	32	E		87	47	E	
14	15	E	15, 16	88	23	E	111
15	53	L	17	89	13	E	90
16	53	R	17	90	19	E	79
17	8	E	18, 19	91	115	E	105

Table A1. *Cont.*

NO.	Time	Side	Immediate Sucessors	NO.	Time	Side	Immediate Sucessors
18	24	L	20	92	35	E	135
19	24	R	20	93	26	L	
20	8	E	21, 22, 23, 24	94	46	E	
21	7	R	25, 26, 27, 28	95	20	E	101
22	8	L	25, 26, 27, 28	96	31	E	104
23	14	L	25, 26, 27, 28	97	19	E	
24	13	R	25, 26, 27, 28	98	34	E	101
25	10	R	29	99	51	E	100
26	25	R	29	100	39	E	101
27	11	L	29	101	30	E	102, 103
28	25	L	29	102	26	E	127
29	11	E	31	103	13	E	127
30	29	R		104	45	E	
31	25	E	36	105	58	E	119
32	10	L	34	106	28	E	107
33	14	R	35	107	8	E	108
34	41	L	36	108	43	E	109
35	42	R	36	109	40	E	110
36	47	R	37	110	34	E	
37	7	R	38, 45	111	23	E	112
38	80	R	39	112	162	L	113
39	7	R	40	113	11	L	114, 116, 120, 123, 128
40	41	R	41, 48, 55	114	19	E	115
41	47	R		115	14	E	125
42	16	L	43	116	31	E	117
43	32	L	44	117	32	E	118
44	66	L		118	26	E	126
45	80	L	46	119	55	E	
46	7	L	47	120	31	E	121
47	41	L	48, 49, 55	121	32	E	122
48	13	E		122	26	E	126
49	47	L		123	19	E	124
50	33	E	51	124	14	E	125
51	34	L	53,69	125	19	E	
52	11	L	53	126	48	E	
53	118	L		127	55	E	
54	25	L	133	128	8	L	129
55	7	R	54, 72, 76, 87, 88	129	11	L	130
56	28	E	73	130	27	L	131, 137
57	12	L	79	131	18	L	
58	52	L	84, 86	132	36	E	135
59	14	E	75, 87	133	23	L	135
60	3	E		134	20	R	135
61	3	E	62	135	46	E	136
62	8	E	63	136	64	E	
63	16	E	67	137	22	L	
64	33	R	65, 71, 72	138	15	E	139
65	8	E	66,99	139	34	E	140
66	18	E	67	140	22	E	
67	10	E	68	141	151	L	142
68	14	E	95,98	142	148	R	143, 146, 147, 148
69	28	R	82	143	64	L	
70	11	R	71	144	170	L	145
71	118	R		145	137	R	147, 148
72	25	R	134	146	64	R	
73	40	E	84, 86, 87, 88, 96	147	78	L	
74	40	E	75	148	78	R	

Table A2. P205.

NO.	Time	Side	Immediate Sucessors	NO.	Time	Side	Immediate Sucessors
1	692	E	36	104	68	R	113
2	42	E	3, 4	105	232	L	106, 107
3	261	R	5	106	122	L	108
4	261	L	5	107	151	E	108
5	157	E	7, 13	108	31	L	113
6	90	E	36	109	97	E	113
7	54	R	8	110	308	R	113
8	67	R	9	111	116	L	113
9	30	R	10	112	312	R	113
10	106	R	11	113	34	E	114, 115, 116, 117, 118, 119, 120, 121, 122, 123, 124, 161, 162, 163, 169, 171, 174, 203, 204, 205
11	32	R	12	114	128	L	160
12	62	R	36	115	54	E	160
13	54	L	14	116	175	R	160
14	67	L	15	117	55	E	160
15	30	L	16	118	306	E	126
16	106	L	17	119	59	E	126
17	32	L	18	120	59	E	126
18	62	L	36	121	66	E	126
19	56	E	36	122	66	E	126
20	67	E	22	123	23	E	126
21	86	E	22	124	244	E	125
22	37	E	23	125	54	E	126
23	41	E	24, 34	126	294	R	127, 128, 129
24	72	E	26, 27, 28	127	84	E	135
25	86	R	28	128	61	E	135
26	16	L	35	129	57	E	135
27	51	R	35	130	38	R	136
28	66	R	29	131	944	E	132
29	41	R	30, 33	132	511	R	133
30	72	R	31, 32	133	625	R	189
31	51	R	35	134	445	R	189
32	16	R	35	135	68	L	136, 137, 138, 139, 140, 141, 142, 144, 145, 147, 148, 149, 150, 151, 152, 153, 158
33	15	R	35	136	53	L	189
34	15	L	35	137	49	E	160
35	85	E	36	138	92	E	160
36	59	E	37, 40, 41, 42, 62, 69, 72, 75, 83, 110, 111, 112	139	236	E	160
37	23	L	38	140	116	L	143
38	13	L	39	141	265	L	143
39	19	L	45	142	149	L	143
40	108	E	43, 54	143	74	L	160
41	214	E	92	144	332	E	160
42	80	E	43, 54	145	324	E	146
43	37	L	44	146	104	L	160
44	84	L	45	147	51	L	160
45	18	L	46, 48, 51, 53	148	58	R	160
46	12	L	47	149	67	R	160
47	29	L	92	150	49	R	160
48	37	L	49	151	107	E	160
49	13	L	50	152	38	L	160
50	70	L	92	153	27	L	154
51	217	L	52	154	68	E	155
52	72	L	92	155	207	E	156
53	85	L	92	156	202	E	157

Table A2. *Cont.*

NO.	Time	Side	Immediate Sucessors	NO.	Time	Side	Immediate Sucessors
54	43	R	55	157	83	E	189
55	97	R	56, 59, 61	158	35	R	159
56	37	R	57	159	58	R	189
57	13	R	58	160	42	E	164, 170, 178, 179, 184
58	35	R	92	161	68	R	167
59	217	R	60	162	68	R	165
60	72	R	92	163	68	R	164
61	85	R	92	164	103	R	165
62	25	E	63	165	103	R	166
63	37	E	64	166	103	R	167
64	37	E	65, 68	167	103	R	168
65	103	E	66	168	103	R	177
66	140	E	67	169	68	L	170
67	49	E	80	170	103	L	172
68	35	E	80	171	68	L	172
69	51	E	70	172	103	L	173
70	88	E	71	173	103	L	175
71	53	E	73	174	68	L	175
72	144	E	73	175	103	L	176
73	337	E	74	176	103	L	177
74	107	E	76	177	10	E	185, 186, 187, 188, 194, 195
75	371	E	92	178	187	E	180
76	97	E	77, 78, 79	179	134	L	180
77	166	E	80, 82	180	89	L	181, 183
78	92	L	80	181	58	L	182
79	92	R	80	182	49	L	
80	106	E	81	183	134	L	
81	49	E	84	184	53	L	
82	92	E	92	185	334	E	189
83	371	E	92	186	24	R	189
84	87	E	85	187	76	R	189
85	162	E	86, 88, 90	188	76	L	189
86	96	E	87	189	192	E	190, 191, 193
87	79	E	92	190	98	E	
88	96	E	89	191	258	R	192
89	42	E	92	192	165	E	
90	88	R	91	193	38	R	
91	90	R	92	194	115	E	197
92	97	R	93, 94, 95, 96, 97, 98, 99	195	83	L	196
93	270	R	135	196	56	R	197
94	452	E	135	197	29	R	198, 199, 201
95	48	R	113	198	303	R	
96	338	E	113	199	18	R	200
97	34	E	100	200	29	R	
98	65	E	100	201	154	L	202
99	50	E	100	202	90	L	
100	112	E	101, 103, 105, 109, 130, 131, 134	203	93	L	
101	48	E	102	204	94	E	
102	117	E	113	205	165	E	
103	50	E	104				

References

1. Zhang, Y.; Hu, X.; Wu, C. A modified multi-objective genetic algorithm for two-sided assembly line re-balancing problem of a shovel loader. *Int. J. Prod. Res.* **2018**, *56*, 3043–3063. [CrossRef]
2. Zhang, Y.; Hu, X.; Wu, C. Improved imperialist competitive algorithms for rebalancing multi objective two-sided assembly lines with space and resource constraints. *Int. J. Prod. Res.* **2020**, *58*, 3589–3617. [CrossRef]
3. Hu, X.; Wu, E.; Bao, J.; Jin, Y. A Branch-and-bound Algorithm to Minimize the Line Length of a Two-sided Assembly Line. *Eur. J. Oper. Res.* **2010**, *206*, 703–707.

4. Hu, X. *Two-Sided Assembly Line Balancing Algorithm and Its Application*, 1st ed.; Science Press: Beijing, China, 2015; pp. 3–10.
5. Bartholdi, J.J. Balancing Two-sided Assembly Lines: A Case Study. *Int. J. Prod. Res.* **1993**, *31*, 2447–2461. [CrossRef]
6. Li, J.; Pang, D.; Zheng, Y.; Le, X. Digital Twin Enhanced Assembly Based on Deep Reinforcement Learning. In Proceedings of the 11th International Conference on Information Science and Technology (ICIST), Chengdu, China, 21–23 May 2021; pp. 432–437.
7. Lv, Y.; Tan, Y.; Zhong, R.; Zhang, P.; Wang, J.; Zhang, J. Deep reinforcement learning-based balancing and sequencing approach for mixed model assembly lines. *IET Coll. Intell. Manuf.* **2022**, *4*, 181–193. [CrossRef]
8. Lee, T.O.; Kim, Y.; Kim, Y.K. Two-sided assembly line balancing to maximize work relatedness and slackness. *Comput. Ind. Eng.* **2001**, *40*, 273–292. [CrossRef]
9. Hu, X.; Wu, E.; Jin, Y. A station-oriented enumerative algorithm for two-sided assembly line balancing. *Eur. J. Oper. Res.* **2008**, *186*, 435–440. [CrossRef]
10. Wei, N.; Liu, S.; Chen, C.; Xu, Y.; Shih, Y. An Integrated Method for Solving the Two-Sided Assembly Line Balancing Problems. *J. Adv. Manuf. Syst.* **2023**, *22*, 181–203. [CrossRef]
11. Li, Z.; Tang, Q.; Zhang, L. Two-sided assembly line balancing problem of type I: Improvements, a simple algorithm and a comprehensive study. *Comput. Oper. Res.* **2016**, *79*, 78–93. [CrossRef]
12. Álvarez-Miranda, E.; Pereira, J.; Vargas, C.; Vilà, M. Variable-depth local search heuristic for assembly line balancing problems. *Int. J. Prod. Res.* **2023**, *61*, 3102–3120. [CrossRef]
13. Yılmaz, D.; Emel, K.A.; Uğur, Ö.; Mehmet, S.İ. A modified particle swarm optimization algorithm to mixed-model two-sided assembly line balancing. *J. Intell. Manuf.* **2017**, *28*, 23–36.
14. Huang, D.; Mao, Z.; Fang, K.; Yuan, B. Combinatorial Benders decomposition for mixed-model two-sided assembly line balancing problem. *Int. J. Prod. Res.* **2022**, *60*, 2598–2624. [CrossRef]
15. Kim, Y.K.; Song, W.S.; Kim, J.H. A Mathematical Model and a Genetic Algorithm for Two-sided Assembly Line Balancing. *Comput. Oper. Res.* **2009**, *36*, 853–865. [CrossRef]
16. Lei, D.; Guo, X. Variable neighborhood search for the second type of two-sided assembly line balancing problem. *Comput. Oper. Res.* **2016**, *72*, 183–188. [CrossRef]
17. Kang, H.; Lee, A.H.I. An evolutionary genetic algorithm for a multi-objective two-sided assembly line balancing problem: A case study of automotive manufacturing operations. *Qual. Technol. Quant. Manag.* **2023**, *20*, 66–88. [CrossRef]
18. Azizoglu, M.; Imat, S. Workload smoothing in simple assembly line balancing. *Comput. Oper. Res.* **2018**, *89*, 51–57. [CrossRef]
19. Walter, R.; Schulze, P. On the performance of task-oriented branch-and-bound algorithms for workload smoothing in simple assembly line balancing. *Int. J. Prod. Res.* **2022**, *60*, 4654–4667. [CrossRef]
20. Uğur, Ö.; Bilal, T. A tabu search algorithm for two-sided assembly line balancing. *Int. J. Adv. Manuf. Technol.* **2009**, *43*, 822–829.
21. Lan, C.H.; Yee, M.S. Construct an INLP Mathematical Model to solve the Two-sided Assembly Line Balancing problem of Type-3. *Adv. Mater. Res.* **2012**, *383–390*, 4302–4305. [CrossRef]
22. Purnomo, H.D.; Wee, H.M. Maximizing production rate and workload balancing in a two-sided assembly line using Harmony Search. *Comput. Ind. Eng.* **2014**, *76*, 222–230. [CrossRef]
23. Li, D.; Zhang, C.; Shao, X.; Lin, W. A multi-objective TLBO algorithm for balancing two-sided assembly line with multiple constraints. *J. Intell. Manuf.* **2016**, *7*, 725–739. [CrossRef]
24. Li, Z.; Tang, Q.; Zhang, L.; Hu, J. Balancing two-sided assembly line with a simple and effective iterated local search algorithm. *ICIC Express Lett.* **2015**, *9*, 2695–2701.
25. Wu, Y.; Tang, Q.; Zhang, L. A hybrid gravitational search algorithm for two-sided assembly line balancing problem with zoning constraints. *ICIC Express Lett. Part B Appl.* **2016**, *7*, 2633–2640.
26. Buyukozkan, K.; Kucukkoc, I.; Satoglu, S.I.; Zhang, D.Z. Lexicographic bottleneck mixed-model assembly line balancing problem-Artificial bee colony and tabu search approaches with optimised parameters. *Expert Syst. Appl.* **2016**, *50*, 151–166. [CrossRef]
27. Yadav, A.; Agrawal, S. Minimize idle time in two sided assembly line balancing using exact search approach. In Proceedings of the 2019 International Conference on Management Science and Industrial Engineering, Phuket, Thailand, 24–26 May 2019; pp. 220–227.
28. Yadav, A.; Verma, P.; Agrawal, S. Mixed model two sided assembly line balancing problem: An exact solution approach. *Int. J. Syst. Assur. Eng.* **2020**, *11*, 335–348. [CrossRef]
29. Abdullah Make, M.R.; Ab Rashid, M.F.F. Optimization of two-sided assembly line balancing with resource constraints using modified particle swarm optimisation. *Sci. Iran.* **2022**, *29*, 2084–2098.
30. Chen, X. Research on Scheduling and Navigation Strategies Based on Reinforcement Learning. Master's Thesis, Zhejiang University, Hangzhou, China, 2021.
31. Tercan, H.; Meisen, T. Machine learning and deep learning based predictive quality in manufacturing: A systematic review. *J. Intell. Manuf.* **2022**, *33*, 1879–1905. [CrossRef]
32. Li, C.; Zheng, P.; Yin, Y.; Wang, B.; Wang, L. Deep reinforcement learning in smart manufacturing: A review and prospects. *CIRP J. Manuf. Sci. Technol.* **2023**, *40*, 75–101. [CrossRef]
33. Mnih, V.; Kavukcuoglu, K.; Silver, D.; Rusu, A.A.; Veness, J.; Bellemare, M.G.; Graves, A.; Riedmiller, M.; Fidjeland, A.K.; Ostrovski, G.; et al. Human-level control through deep reinforcement learning. *Nature* **2015**, *518*, 529–533. [CrossRef]
34. Chen, X.; Yao, L.; McAuley, J.; Zhou, G.; Wang, X. Deep reinforcement learning in recommender systems: A survey and new perspectives. *Knowl.-Based Syst.* **2023**, *264*, 110335. [CrossRef]

35. Goby, N.; Brandt, T.; Neumann, D. Deep reinforcement learning with combinatorial actions spaces: An application to prescriptive maintenance. *Comput. Ind. Eng.* **2023**, *179*, 109165. [CrossRef]
36. Kallestad, J.; Hasibi, R.; Hemmati, A.; Sorensen, K. A general deep reinforcement learning hyper heuristic framework for solving combinatorial optimization problems. *Eur. J. Oper. Res.* **2023**, *309*, 446–468. [CrossRef]
37. Zhang, Z.; Liu, H.; Zhou, M.; Wang, J. Solving Dynamic Traveling Salesman Problems With Deep Reinforcement Learning. *IEEE Trans. Neural Netw. Learn.* **2023**, *34*, 2119–2132. [CrossRef] [PubMed]
38. De Sirisuriya, S.C.M.S.; Fernando, T.G.I.; Ariyaratne, M.K.A. Algorithms for path optimizations: A short survey. *Computing* **2023**, *105*, 293–319. [CrossRef]
39. Jiang, Y.; Cao, Z.; Zhang, J. Learning to Solve 3-D Bin Packing Problem via Deep Reinforcement Learning and Constraint Programming. *IEEE Trans. Cybern.* **2023**, *53*, 2864–2875. [CrossRef]
40. Dai, H.; Khalil, E.B.; Zhang, Y.; Dilkina, B.; Song, L. Learning combinatorial optimization algorithms over graphs. *Adv. Neural Inf. Process. Syst.* **2017**, *30*, 6349–6359.
41. Zhang, C.; Wu, Y.; Ma, Y.; Song, W.; Le, Z.; Cao, Z.; Zhang, J. A review on learning to solve combinatorial optimisation problems in manufacturing. *IET Collab. Intell. Manuf.* **2023**, *5*, e12072. [CrossRef]
42. Kim, Y.K.; Kim, Y.; Kim, Y.J. Two-sided assembly line balancing: A genetic algorithm approach. *Prod. Plan. Control* **2000**, *11*, 44–53. [CrossRef]
43. Wang, K.; Li, X.; Gao, L. A multi-objective discrete flower pollination algorithm for stochastic two-sided partial disassembly line balancing problem. *Comput. Ind. Eng.* **2019**, *130*, 634–649. [CrossRef]
44. Raju Bahubalendruni, M.V.A.; Biswal, B.B. An efficient stable subassembly identification method towards assembly sequence generation. *Natl. Acad. Sci. Lett.* **2018**, *41*, 375–378. [CrossRef]