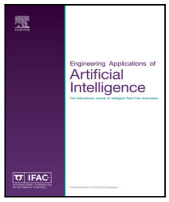
Engineering Applications of Artificial Intelligence 101 (2021) 104226

Contents lists available at ScienceDirect

Engineering Applications of Artificial Intelligence

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



Automation of load balancing for Gantt planning using reinforcement learning

Jong Hun Woo, Byeongseop Kim, SuHeon Ju, Young In Cho *

Department of Naval Architecture and Ocean Engineering, Seoul National University, Seoul, Republic of Korea

ARTICLE INFO

Keywords:
Shipbuilding
Production planning
Workload balancing
Reinforcement learning
Deep neural networks

ABSTRACT

Typically, in the shipbuilding industry, several vessels are built concurrently, and a production plan is established through a hierarchical planning process. This process largely comprises strategic planning (long-term) and master planning (mid-term) aspects. The portion that requires the most manual work of the planner is the load balancing in the master planning stage. The load balancing of master planning is an area where optimization studies using mixed integer programming, genetic algorithms, tabu search algorithms, and others have been actively conducted in the field of operational research. However, its practical application has not been successful due to the complexity and the curse of dimensionality, which is dependent on the manual work of the planner. Therefore, a new method that can facilitate the efficient action of optimal decisions is required, replacing conventional production planning methods based on the manual work of the planner. With the advent of the 4th industrial revolution in recent years, machine learning technology based on deep neural networks has been rapidly developing and applied to a wide range of engineering problems. This study introduces a methodology that can quickly improve the load balancing problem in shipyard master planning by using a deep neural network-based reinforcement learning algorithm among various machine learning techniques. Furthermore, we aim to verify the feasibility of the developed methodology using the ship block production data of an actual shipyard.

1. Introduction

The shipbuilding industry is a typical order-driven industry, and unlike the general mass production industry, where the emphasis is placed on demand forecasting, compliance with the contract of a ship owner is a key production constraint. Therefore, it is important to establish production plan with workload while complying with the delivery date determined at the time of contract. Shipyard production planning is characterized by backward planning of the ship block production, such as assembly, outfitting, and painting, based on the erection network plan at the dry dock or skid birth (Fig. 1).

In the case of large-scale shipyards, several ships are built concurrently. Consequently, in this type of shipyard a production plan is established through a hierarchical planning process based on the planning period and target. The hierarchical planning process is largely divided into strategic planning and master planning—the portion requiring the most manual work of the planner being the load balancing in the master planning stage. In addition, master planning requires several iterative updates due to changes in strategy or resources, taking up a considerable number of man-hours and makespan. The load balancing of master planning is an area where optimization studies using mixed integer programming (MIP), genetic algorithms (GAs), tabu search (TS) algorithms, and others have been actively conducted in the field of operational research (OR). However, its practical application has not been successful due to the complexity and the curse of dimensionality, which is dependent on the manual work of the planner. Therefore, a new method that can facilitate the efficient action of optimal decisions is required, replacing the conventional production planning method based on manual work in situations where changes in master planning are required. With the advent of the 4th industrial revolution in recent years, machine learning technology based on deep neural networks has been rapidly developing and applied to a wide range of engineering problems.

In the master planning phase of the shipyard, iterative jobs are conducted for workload balancing while satisfying various constraints for ship block and ship zone schedules, as shown in Fig. 2 (Lee et al., 2020). Starting with the erection planning process, the process proceeds to the pre-erection (PE) block and assembly block, the process of adjusting production activities to achieve load balancing in each step being performed iteratively. In particular, for the PE block and assembly block, the planning is conducted repetitively by a human planner until load balancing reaches a satisfactory level, resulting in the segment becoming a bottleneck task for production planning. To be

* Corresponding author.
E-mail addresses: j.woo@snu.ac.kr (J.H. Woo), jjolla93@snu.ac.kr (B. Kim), wntngjs@snu.ac.kr (S. Ju), whduddlsi@snu.ac.kr (Y.I. Cho).

https://doi.org/10.1016/j.engappai.2021.104226
Received 13 November 2020; Received in revised form 25 February 2021; Accepted 4 March 2021
Available online 22 March 2021

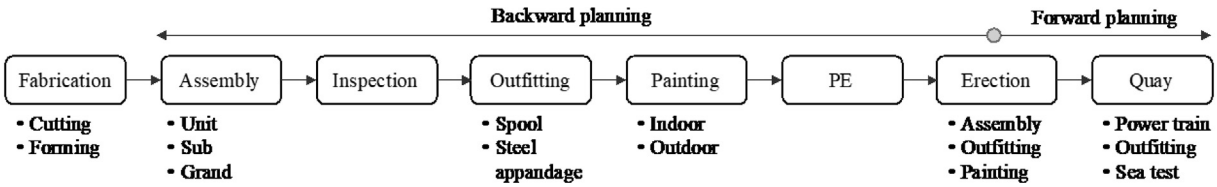


Fig. 1. Shipbuilding process and planning direction.

Abbreviations

A2C	Advantage actor–critic
A3C	Asynchronous advantage actor–critic
CNN	Convolutional neural network
CSP	Constraint satisfaction problem
DES	Discrete event simulation
DKQRL	Q-learning reinforcement with a knowledge-driven ϵ-greedy algorithm
DQN	Deep Q-network
ES	Earliest start
FCN	Fully convolutional network
FEA	Finite element analysis
FF	Finish-to-finish
FS	Finish-to-start
GA	Genetic algorithm
LF	Latest finish
MIP	Mixed integer programming
MLCS	Modified lowest cost search
NAN	Not a number
OR	Operation research
PE	Pre-erection
RNN	Recurrent neural network
SA	Simulated annealing
SARSA	State–action–reward–state–action
T/O	Turn-over
TS	Tabu search

more specific, the computation of a large amount of data needs to be conducted to determine load balancing after activity adjustments, and the adjustment work cannot be completed immediately but needs to be performed iteratively over several cycles, which requires a significant amount of time.

Indeed, for problems with precedence relationships between jobs (for the jobshop) or activities (for activity network planning) while each activity has a respective duration, a variety of research has been conducted using constraint satisfaction problem (CSP) methods. These approaches have been successfully applied to problems such as planning airport flight scheduling or establishing an optimal plan for a jobshop-type production system. In the shipbuilding sector, there have been numerous attempts to address such an activity scheduling using optimization techniques, such as the CSP method (Basán et al., 2017; Hu et al., 2019; König et al., 2007; Lee et al., 1994; Rose and Coenen, 2015). However, this methodology is currently applied only on limited occasions.[1] For this reason, in many shipyard situations—in a planning environment (dedicated software) in which the constraints on precedence relationships are reflected in the initial plan made using a Gantt chart (Clark, 1922)—the human planner manually adjusts the activities and continues with repetitive jobs until load balancing reaches a satisfactory level, as shown in Fig. 5.

In this study, we propose an automated method of load balancing for a Gantt-planning type of practice such as the planning of a block assembly process. This study introduces a methodology that can quickly improve the load balancing problem of Gantt planning by using a deep neural network-based reinforcement learning algorithm among various machine learning techniques. Furthermore, we aim to verify the feasibility of the developed methodology using the ship block production data of an actual shipyard.

Reinforcement learning is a research field that derives an optimal policy by formulating the Bellman equation for problems that can be modeled as a Markov process and has been actively researched since the 1990s. In addition, in the study of Hinton et al. (2006), the development of a deep belief network algorithm effective for deep learning triggered innovation in the field of artificial intelligence and new opportunities with reinforcement learning emerged.

The following studies on reinforcement learning have been actively conducted in the fields of product allocation, inventory management, and production planning in relation to production system optimization in the manufacturing sector. Lee et al. (2019) conducted a study on scheduling problems based on reinforcement learning for semiconductor manufacturing processes. In this study, the state–action–reward–state–action (SARSA) algorithm was applied to train the model to determine the work to be put into the process among the jobs waiting in the buffer based on the scores for various dispatching rules. Lee et al. (2019) conducted a study on reinforced learning-based scheduling with the aim of learning a planning model applicable to various production environments for semiconductor packaging lines. To this end, this study focused on the robustness of the learning model and studied the scheduling problem using the deep Q-network-based (DQN) reinforcement learning algorithm to which normalized learning was applied. Shin and Ru (2010) conducted a study to learn the scheduling method that adaptively changed the dispatching rule as the probability of rework changed with respect to manufacturing systems that had an unstable rework probability. Lee et al. (2020) conducted a study to apply the DQN algorithm to the scheduling problem of injection mold production with the objective of minimizing the weighted tardiness. The multi-agent reinforcement learning algorithm in which agents are allocated to different machines and each agent perceives only the state of the relevant machine was proposed. Shi et al. (2020) conducted a study on applying a deep reinforcement learning algorithm to the scheduling problems for transferring units operating on automated production lines with stochastic processing time. Hameed and Schwung (2020) conducted a study to represent production environments as graphs and use a deep reinforcement learning algorithm based on graph neural network to solve the job-shop scheduling problems.

In the shipbuilding sector, the following studies that applied optimization techniques in OR for the block assembly process and block erection process, in which problem formulation was relatively easy, have been published. In the study of Cho et al. (1998), an improvement method for block assembly planning was investigated in consideration of the bottleneck process. Bae et al. (2007), investigated minimizing

[1] During the period from the late 1990s to the mid-2000s, large and medium-sized shipyards in Korea invested heavily in planning optimization for activities with precedence relationships using CSP-based algorithms that were popular at the time. However, as of 2020, there are no cases in which optimization algorithms such as CSP, GA, and TS are applied to activity planning. There is no official publication stating this information, so we ask for your understanding that there is no related reference.

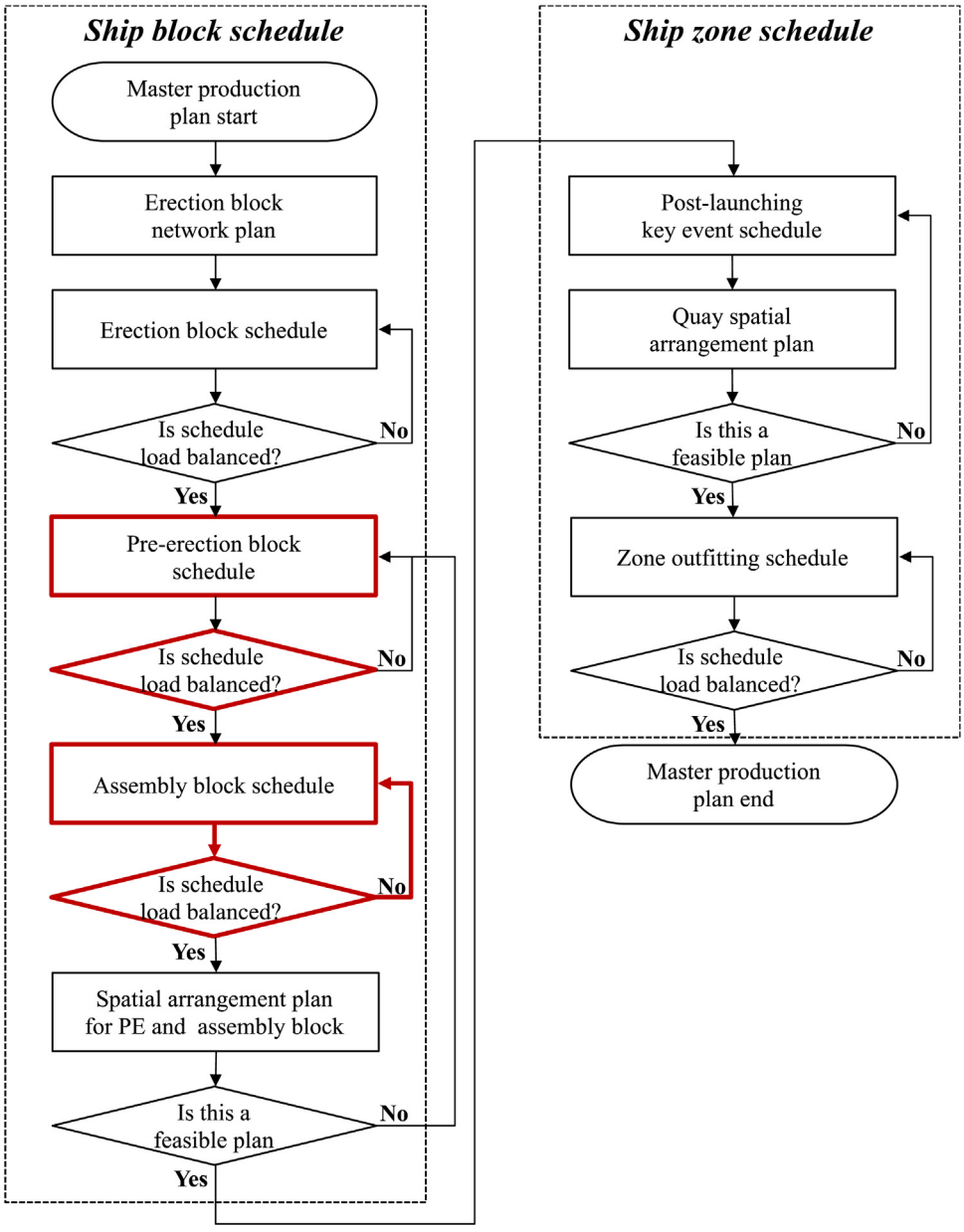


Fig. 2. Process of master planning (Lee et al., 2020).

makespan by optimizing the block process sequence input to the assembly process using a GA. In the work of Woo et al. (2003), optimization was performed for block erection planning and pre-erection planning using a heuristic algorithm. In the study of Hwang et al. (2010), the optimization of production planning of a unit assembly line was performed by applying the simulated annealing (SA) algorithm in the execution planning stage. Liu et al. (2011) proposed a production planning optimization method that considered the inventory level of blocks and pieces using a GA. In the study of Basán et al. (2017), to minimize the ship block assembly makespan, a heuristic study using the discrete event simulation (DES) simulation method considering various constraints was simulated.

In addition, numerous studies have been conducted on spatial optimization, an important constraint in shipbuilding production planning. In the study by Zhuo et al. (2012), a hybrid planning method was proposed for block assembly process planning in which an initial plan was established through rule-based dispatching by simulation, and spatial planning was performed using an enumeration-based search algorithm. Kwon and Lee (2015) proposed a two-step heuristic planning technique for the spatial scheduling of large ship blocks, in which simple priority rules and dispatching rules were used for the quick grouping of assembly blocks and spatial scheduling was performed for the grouped blocks using a diagonal fill placement method. Shang et al. (2017) applied the best contact algorithm and genetic algorithm to an actual shipyard block assembly process for spatial scheduling optimization, thereby minimizing the makespan and supporting production planning.

However, although these planning optimization studies achieved effective results under restricted conditions, most of them have not been successful in actual shipyard situations due to the complexity and curse of dimensionality. In the case of large shipyards in South Korea, a heuristic algorithm has been applied in a limited way for initial plan establishment, but the load balancing planning is still performed manually at every step by a human planner.

In contrast, in the shipbuilding sector in recent years, as part of production advancement research using machine learning technology, a study on lead-time prediction model development (Jeong et al., 2020) and a study on prognostic maintenance for production resources

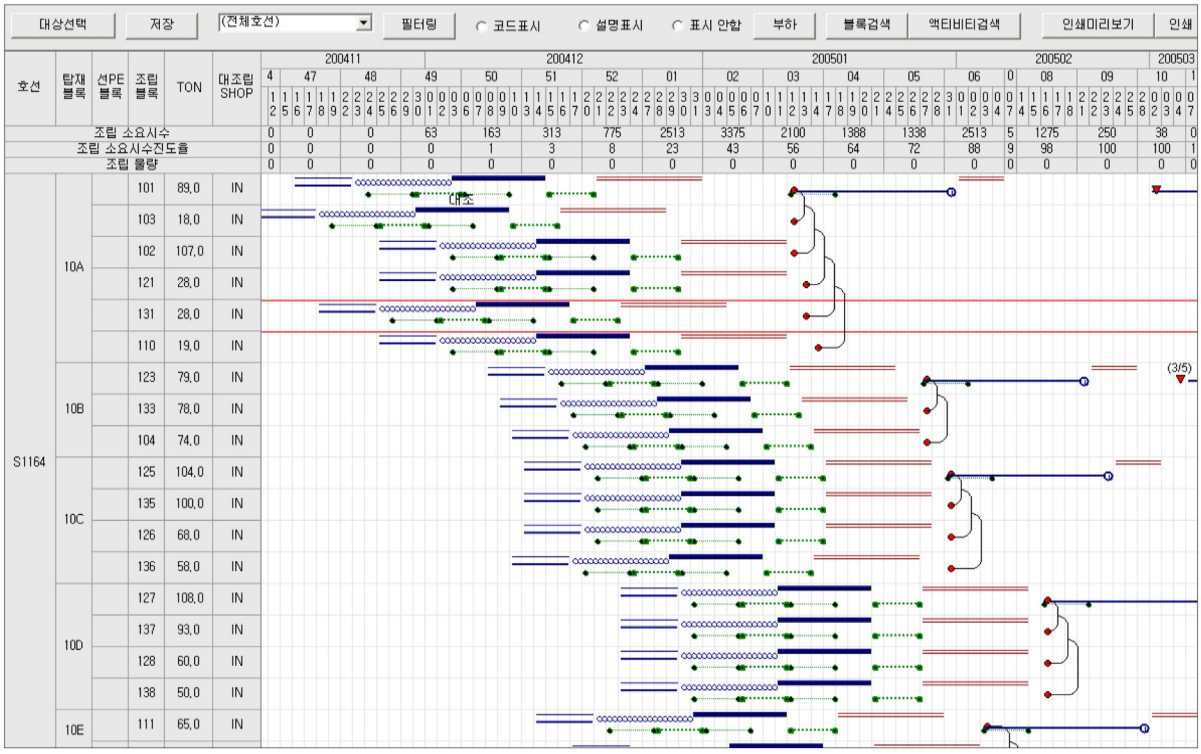


Fig. 3. Example of block assembly planning with a Gantt chart type planning environment.

(Lee and Kim, 2014) using supervised learning based on deep neural networks have been conducted. In the field of reinforcement learning, for complex pipe routing design in the early stages of ship design, a reinforcement learning technique was implemented to minimize the connection length and bending of pipes in the hull environment modeled in unity (Shin et al., 2020). In addition, in the shipbuilding production management area, the reinforcement learning algorithm was used to determine the location of the stockyard, demonstrating superior results to the existing heuristic algorithm (bottom-left fill algorithm) (Kim et al., 2020). In the study by Romero-Hdz et al. (2020), a Q-learning reinforcement technique with a knowledge-driven ϵ-greedy algorithm (DKQRL) study was conducted in a thermo-mechanical finite element analysis (FEA) environment for optimization of the welding sequence for block assembly in the shipyard. As a result of the research, compared to the existing techniques, such as the modified lowest cost search (MLCS) method, GA, and exhaustive search, welding deformation was reduced by 71%. As the feasibility of reinforcement learning methods using deep neural networks has been verified – as can be seen from these previous studies – artificial-intelligence-based research into production processes that has not yet been put into practical use (with algorithms such as GA, TS, and MIP) is increasingly being conducted.

The objective of this work was to achieve the load balancing of production planning using reinforcement learning based on a deep neural network under the assumption that an initial plan was established for the production plan (Fig. 3) in the form of a Gantt chart. It should be noted in advance that our aim was not to find an optimal solution through mathematical formulation of the planning problem, such as with the existing CSP, GA, and TS techniques. The objective of this study was to replace the Gantt-based activity planning currently being performed manually in the majority of shipyards due to the limitations of the abovementioned optimization methodologies and to develop artificial intelligence that could derive improved load balancing results compared to those obtained by manual work.

For this purpose, a grid-type environment was developed for the simulation of the Gantt-planning environment. Reinforcement learning was performed in the grid environment using duration and man-hour information obtained from the shipyard and activity data, including activity relationships. For the reinforcement learning algorithm, the asynchronous advantage actor–critic (A3C) algorithm was used in this study. In addition, we aimed to analyze the feasibility of the developed environment and learning algorithm through the application of the reinforcement learning algorithm to the activity planning of real-world ship blocks.

2. Problem definition

In this study, a reinforcement learning study was conducted for the block assembly process during the shipyard production planning. This process involves assembling ship blocks by dividing them into several steps to supply blocks erected from a dry dock or skid berth. In a general assembly process, parts are manufactured through a working (cutting and forming) process, and assembly of parts starts through a subassembly process. After the subassembly process, blocks go through unit assembly and grand assembly processes before being transferred to the erection process. If the assembly is performed with the top and bottom turned over for manufacturing convenience, the turn-over process can be added after the grand assembly process. In this work, among the above processes, a load balancing study with reinforcement learning techniques was conducted for the unit assembly and grand assembly of flat unit block/curved unit blocks and the turn-over process.

The target assembly process in this study is shown in Fig. 4, and the production planning for this assembly process was performed using the Gantt planning method, as shown in Fig. 5. The unit assembly and grand assembly have a finish-to-start (FS) relationship as the grand assembly can only start once the unit assembly has been completed. Because the turn-over process starts by installing the wires in the blocks, the process is performed in parallel during the grand assembly process, and the completion of the turn-over process indicates the completion of the grand assembly process itself. Thus, the grand assembly and turn-over processes have a finish-to-finish (FF) relationship.

A diagram of the environment and learning is shown in Fig. 6. The actual planning target at the bottom is modeled in the grid with

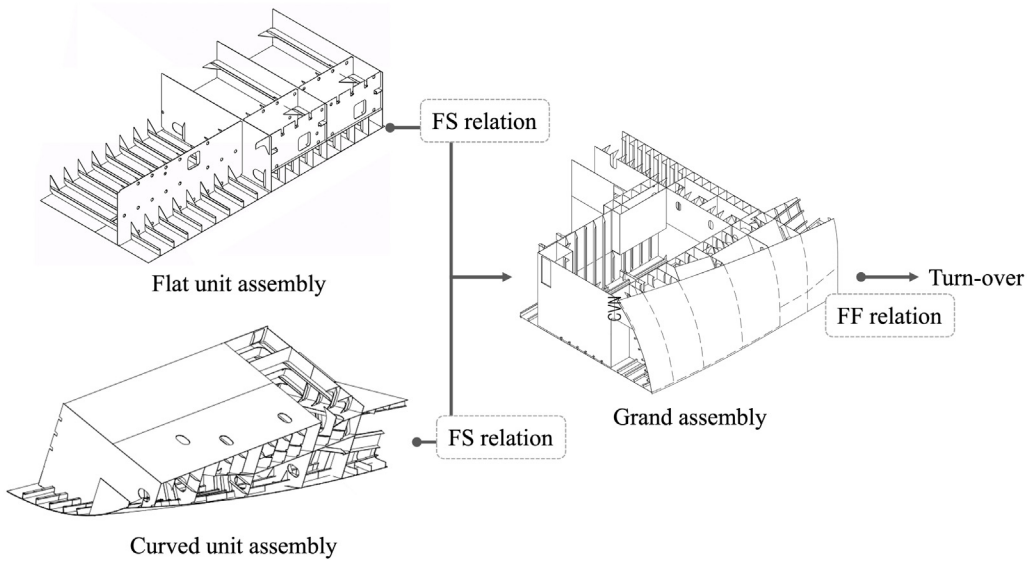


Fig. 4. Target assembly process.

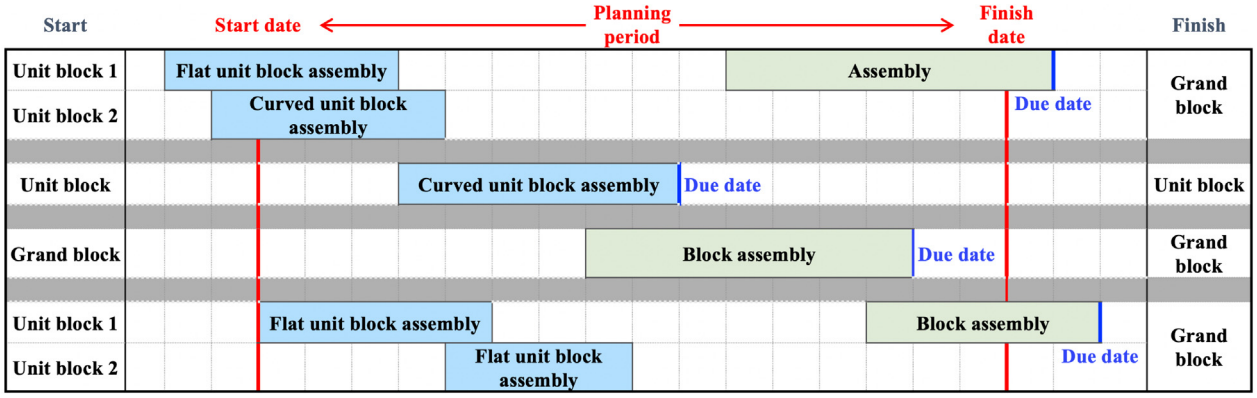


Fig. 5. Problem domain (partially selected).

the environment in the middle. For the learning algorithm, the A3C algorithm with a single agent was used. Each step is performed until positions are determined for input activities for the learning, and when the start date (or end date) for all activities is determined, one episode is completed. Furthermore, for each step, the agent instructs an action on the environment, and an immediate reward for the action is given, the next state being fed back to the agent. The episode proceeds until the reward converges, and when there is no further increase in the reward, the learning is stopped. At this stage, as the learning status (i.e. has the reward converged) and the learning quality are determined according to the definition of state, action, and reward, the optimal combination of conditions is searched for various scenarios.

3. Reinforcement algorithm

In this study, the A3C algorithm was used. In the initial phase of the study, the DQN algorithm (Mnih et al., 2013) was applied, but no convergence was achieved in any case. The A3C algorithm applied in this work is a policy-based reinforcement learning algorithm and was also introduced by Mnih et al. (2016). Several agents perform learning in the respective independent environment, and by introducing agents that learn independently, the correlations between data are reduced. The advantage actor–critic (A2C) algorithm is applied to the learning of individual agents, and the learning results of each agent through the A2C algorithm are updated asynchronously in the global network. In the A2C algorithm, two deep neural networks are used. One network called critic is for evaluating the value function of states. States are input to critic-network, and then the corresponding value functions are calculated. The other network called actor is for approximating the policy of the agent. States are input to the actor-network, and then probabilities of selection for each action are calculated. The agent selects the action based on these probabilities. A2C algorithm is about how to update the weights of these two networks. Based on the samples of states, actions, and rewards which the agent obtains while proceeding with the episodes, the critic network uses the squared error function as a loss function to adjust its weights, and the actor network uses the cross-entropy function. The detailed expressions are given in Table 1. As shown in Fig. 7, an individual agent marked as a worker was operated by the A2C algorithm, and the A3C algorithm was a learning method in which the learning results of each agent were asynchronously updated in the global network.

The process of the A3C algorithm is summarized in Table 1. The A3C learning algorithm starts by initializing the entire step counter T to 0 (line 1). T is a variable shared by all agents, and the sum of the number of steps performed by each agent is recorded—in the code, it is set as a variable called global episodes. Next, the step counter t of the threads is initialized to 1 (line 2). Here, the time step in which each agent is located is recorded—in the code, it is set as a variable called the total number of steps.

Next, the gradient for the weight of the global policy neural network is initialized to *0*, and the weight of the corresponding local policy

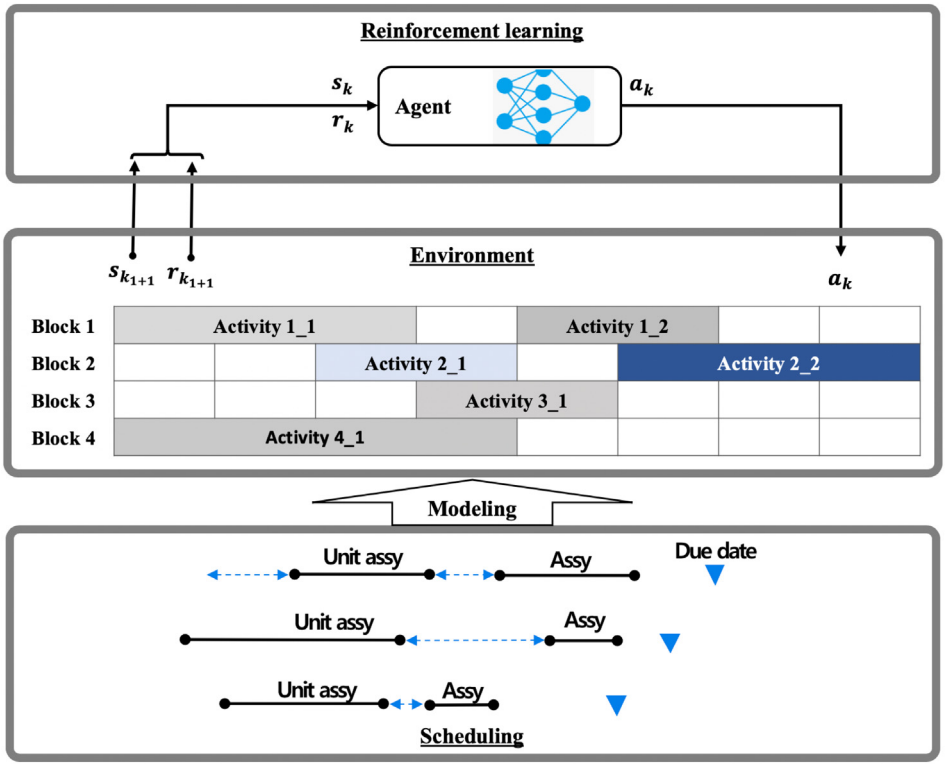


Fig. 6. Diagram of reinforcement learning for Gantt-planning-based decisions.

Table 1
Pseudocode of A3C learning algorithm.

1	Asynchronous advantage actor–critic algorithm pseudocode
2	Initialize global shared counter $T = 0$
3	Initialize thread step counter $t \leftarrow 1$
4	**repeat**
5	Reset gradients: $d\theta \leftarrow 0$ and $d\theta_v \leftarrow 0$
6	Synchronize thread-specific parameters $\theta' = \theta$ and $\theta'_v = \theta_v$
7	$t_{start} = t$
8	Get state s_t
9	**repeat**
10	Perform a_t according to policy $\pi(a_t \mid s_t ; \theta')$
11	Receive reward r_t and new state s_{t+1}
12	$t \leftarrow t + 1$
13	$T \leftarrow T + 1$
14	**until** terminal s_t or $t - t_{start} == t_{max}$ where $t_{max} = Nstep$ of bootstrapping
15	$R = \begin{cases} 0 \\ V(s_t, \theta'_v) \quad for\ non-terminal\ s_t \end{cases}$
16	**for** $i \in \{t-1, \cdots, t_{start}\}$ do
17	$R \leftarrow r_i + \gamma R$ (γ : $discount factor$)
18	Accumulate gradients wrt. θ': $d\theta \leftarrow d\theta + \nabla_{\theta'} log\pi(a_i \mid s_i ; \theta')(R - V(s_i, \theta'_v))$
19	Accumulate gradients wrt. θ'_v: $d\theta_v \leftarrow d\theta_v + \partial(R - V(s_i, \theta'_v))^2 / \partial\theta'_v$
20	**end for**
21	Perform asynchronous update of θ using $d\theta$ and of θ_v using $d\theta_v$ using learning rate α
22	**until** $T > T_{max}$

where
s_t: state of environment at time t
a: action triggered by policy π
R: reward returned by environment with input of action a
θ: Weight variable of global policy network
θ_v: Weight variable of global value network
θ': Weight variable of local policy network
θ'_v: Weight variable of local value network.

network is initialized as the global policy neural network weight (line 5, 6). Then, t_{start}, which is the starting time step of *N-step* bootstrapping, is initialized to the time step of the current agent (line 7). Here, *n-step* bootstrapping is a method of updating weights for every *n* defined steps—unlike the temporal difference method that updates weights at each step and the Monte Carlo method that updates weights at the end of an episode.

Next, with the input of state s_t from the environment (line 8), the algorithm proceeds to the iterative step that collects samples. Action a_t is selected from the policy network $\pi(a_t \mid s_t ; \theta')$, and the reward r_t and the next state s_{t+1} are received from the environment (line 10, 11).

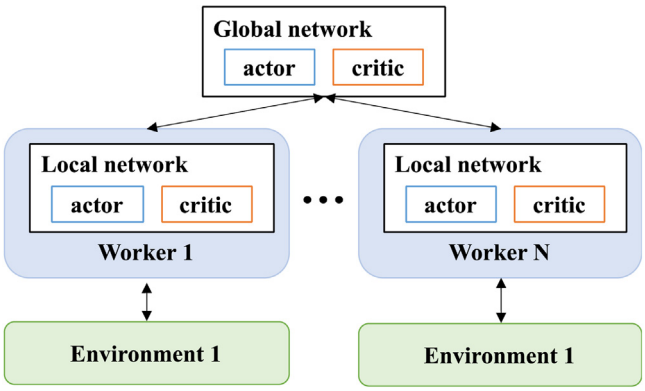


Fig. 7. Schematic diagram of A3C reinforcement algorithm.

Table 2
Learning parameters.

Learning parameters	Value
Learning rate	1e−5
Discount factor	0.99
Number of step	30

Then, the global step counter T and thread step counter t are increased by *1* (line 12, 13). This process is repeated until the terminal state is reached or the preset t_{max} time step is reached (line 14).

Next, the return value R is calculated from the reward received from the environment (line 15). At this point, the return value for the terminal state is *0*, and that for states other than the terminal state is calculated based on the discounting factor. Subsequently, the gradients for the policy network and value network are updated using the return values. In this way, gradients are iteratively calculated and cumulated for all states in the sample, and the weight of the global network is finally updated using the cumulated values (line 18, 19, 21). The flow diagram about the A3C mechanism is added in Appendix A.

Parameters related with learning include learning rate, discount factor, number of step with N-step bootstrapping. In this study, the appropriate values for each parameter are found using a trial and error method. The values used in this study for the parameters are shown in Table 2.

Two convolutional neural networks (CNNs) and two fully convolutional networks (FCNs) were used as the neural networks in this study. In the initial code, one recurrent neural network (RNN) was used at the end of the algorithm, but in the process of updates from Case 1 to Case 4 (described in Section 5)—when there was a RNN—the learning was discontinued with the increasing frequency of NAN[2] occurrence. Consequently, the RNN was excluded from the final case (see Table 3).

4. Environment modeling

4.1. Input

Before constructing a learning environment, we describe the format and content of the input data in this section. Table 4 shows the block list and corresponding information for the E62 block group among the blocks (Appendix B: Table B.1.[3]) constituting the project number 1962.

[2] NaN stands for Not a Number and is a member of a numeric data type that can be interpreted as a value that is undefined or unrepresentable, especially in floating-point arithmetic.

[3] In this paper, from the entire data, the unit assembly of flat unit block (06)/curved unit block (07) and grand assembly (04) and turn-over (08) processes are the subject of learning. The numbers in parentheses represent the process number in Appendix B: Table B.1

Table 3
Configuration of the neural network of each agent.

Layer	Hyper-parameter	
CNN 1	Filter: 32, kernel: 8 × 8, stride: 4 × 4 activation function: ELU	
CNN 2	Filter: 64, kernel: 4 × 4, stride: 2 × 2 activation function: ELU	
FCN 1	Neuron: 256 activation function: ELU	
FCN 2 (Output layer)	Critic	Neuron: 1 Activation function: Linear
	Actor	Neuron: 2 Activation function: SoftMax

In the block data, the last digit (in the case of the unit assembly) or the second to last digit (in the case of the block assembly) indicates the port and starboard symmetric, respectively. Then, as can be seen from the start and finish dates, the port and starboard blocks always undergo processes simultaneously. Therefore, in this study, the same block number consisting of port and starboard was preprocessed as one block (Table 5).

One block group (E62) is a block corresponding to a grand block in Fig. 4 and is assembled through the work of the block assembly stage. In addition, the four blocks (E620, E623, E624, and E627) constituting the block group correspond to a flat unit assembly when the process number is 06 and a curved unit assembly when the process number is 07.

4.2. State

The state of the environment for learning was configured as a grid-type environment in consideration of the input data in 4.1. For the target problem, a plan was established based on the ship block, and the target of the plan at reinforcement learning was the production activity for each block. The environment for the reinforcement learning was configured as a grid type, as shown in Fig. 8.

As shown in Fig. 8, one block group was assigned to one row, and the block group was formed by assembling unit blocks first, as shown in Fig. 4 and then by assembling them into a block of a higher level—that is, several unit blocks could be included in one row. In addition, if there was a turn-over (T/O) process after the assembly, the activity was added. In other words, one block group could have four types of activities as sub-activities: flat unit block assembly, curved unit block assembly, block assembly, and T/O, and there was no limit on the number of activities.

Next, the constraints on the activity relationship and the due date were considered.

For the activity relationship, the first constraint, there is no activity relationship between flat unit block assembly and curved unit block assembly—the FS relationship with block assembly for flat unit block assembly and curved unit block assembly and block assembly has an FF relationship with T/O activity. Activities with an FS relationship cannot overlap on the Gantt chart, and the activities must be scheduled so that the start date of the succeeding activity is set after the finish date of the preceding activity. Activities with an FF relationship can overlap, but the activities must be scheduled so that the finish date of the succeeding activity is placed before the finish date of the preceding activity. In addition, activities that do not have any relationship can be arbitrarily overlapped.

The second constraint is that of the due date condition for each assembly block. As shown in Fig. 8, for block-specific activities – earliest start (ES) and latest finish (LF) – were added. However, in actual learning, only the LF condition was set as a constraint. In other words, in the case of the forward direction process, the constraint that it does not proceed beyond the LF when moving in the forward direction,

Table 4
Input data (partially selected).

Project	Block	Block group	Process	Start date	Finish date	Duration	Mh	Weight	Stage	Due date
1962	E620P	E62	07	20190122	20190129	6	223	0	Unit assembly	20190311
1962	E620S	E62	07	20190122	20190129	6	212	0	Unit assembly	20190311
1962	E623P	E62	06	20190124	20190129	4	143	0	Unit assembly	20190311
1962	E623S	E62	06	20190124	20190129	4	109	0	Unit assembly	20190311
1962	E624P	E62	06	20190118	20190123	4	53	0	Unit assembly	20190311
1962	E624S	E62	06	20190118	20190123	4	21	0	Unit assembly	20190311
1962	E627P	E62	06	20190115	20190118	4	108	0	Unit assembly	20190311
1962	E627S	E62	06	20190115	20190118	4	105	0	Unit assembly	20190311
1962	E62P0	E62	04	20190130	20190215	9	276	212	Block assembly	20190311
1962	E62S0	E62	04	20190130	20190215	9	222	203	Block assembly	20190311

Table 5
Input data after pre-processing (port and starboard blocks merging).

Project	Block	Block group	Process	Start date	Finish date	Duration	Mh	Weight	Stage	Due date
1962	E620	E62	07	20190122	20190129	6	223	0	Unit assembly	20190311
1962	E623	E62	06	20190124	20190129	4	143	0	Unit assembly	20190311
1962	E624	E62	06	20190118	20190123	4	53	0	Unit assembly	20190311
1962	E627	E62	06	20190115	20190118	4	108	0	Unit assembly	20190311
1962	E62	E62	04	20190130	20190215	9	276	212	Block assembly	20190311

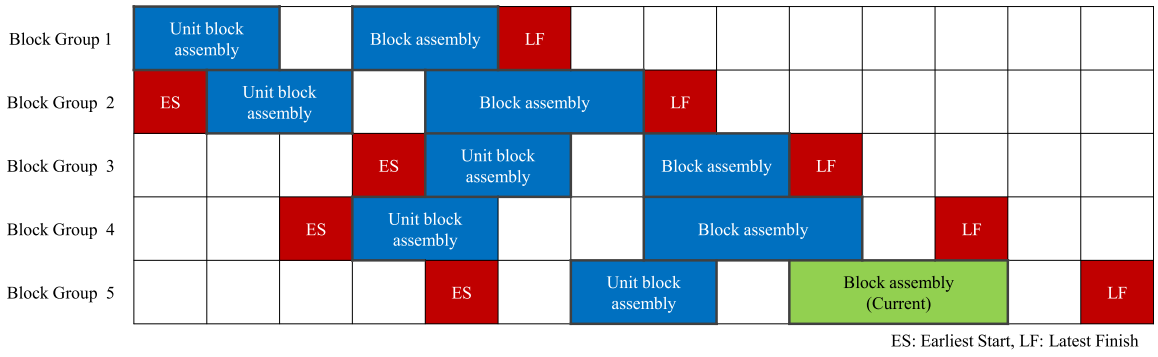


Fig. 8. Learning environment of grid-type configuration.

starting from 0 of the time horizon, was applied. In the case of the backward direction process, because it starts from the LF and proceeds backward, there is no activity that proceeds beyond the LF.

The forward method was tested only in the first learning trial of this study, but no satisfactory results were obtained. For all subsequent learning cases, backward direction learning was performed. For blocks that have completed the block assembly process, outfitting and painting are performed. Because the shipyard production planning is conducted in the backward direction from the erection process, the due date of the assembly process taking into account the start date of the outfitting process is given for each block. Therefore, on the Gantt chart, learning started with the location of the activities initialized so that the finish date of the final activity of the assembly process for each block was scheduled before the given due date.

4.3. Action

There are two actions performed by an agent. The first action is to move the location of the activity – which is the current planning target – one space to the left in the grid environment (advancing the start date of the activity by one day), and the second action is to fix the location of the activity (confirming the plan). In other words, master planning is established in such a way that the agent moves the activity in the backward direction to adjust the plan and then confirms the plan in a suitable location that achieves load balancing. This process is iteratively performed for other activities. At this point, for the value of each state cell, *1* is assigned to the location of the activity currently on the move, *2* is assigned to the location of the fixed activity, and *3* is assigned to the location of the constraint condition (due date). In addition, *0* is assigned to the empty place (place with no workload), as shown in Fig. 9.

4.4. Reward

The objective of this study was to increase the performance of workload balancing in production planning. As the learning outcome differs depending on the reward, the method of considering workload varied for reflection in the learning.

The first method tried corresponds to Case 1 in Table 6, and the reward was *0* when there were three or more activities that overlapped each day for the time horizon for learning, *1* when two activities overlapped, and *2* when the activities did not overlap. Subsequently, the reward was determined by summing the respective values for all planning days.

However, because the extent of overlapping of activities is different according to the learning target in this method, in Case 2 of Table 6, the mean of the number of overlapping activities for the total number of target days was used as the reference value, and the number of overlapping activities per day was compared with the reference value to determine the reward. In addition, for the section of summing up the reward, only the section with learning in progress was considered. If the number of daily overlapping activities was smaller than the reference value, the reward was set to *2*. If the number of activities was the same, it is set to *1*, and if it was greater, the reward was set to *-1*, and these were added for all planning days to calculate the final reward.

Next, for Case 3, though it was similar to Case 2, the actual man-hours were considered instead of the presence or absence of workload. That is, in Cases 1 and 2, the daily workload was set to *1* regardless

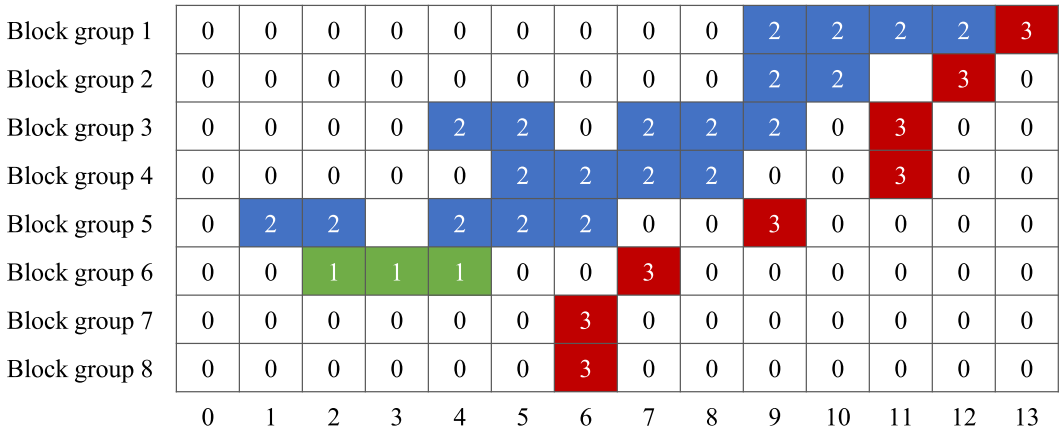


Fig. 9. Learning environment with values.

Table 6
Learning cases.

	Direction	State				Action	Reward	Time window
		Value	Lag	Relation	Scope			
Case 1–1	Forward		Not considered	No	Include fixed activity	Right/Fix	*2*: Activities do not overlap *1*: 2 activity overlap *0*: 3 or more activity overlap	No
Case 1–2	Backward		Not considered	No	Include fixed activity	Left/Fix		No
Case 2	Backward		Not considered	No	Include fixed activity	Left/Fix	*2*: Less than reference value[a] *1*: Same with reference value *-1*: Greater than reference value	Yes
Case 3	Backward	MH	Not considered	Yes	Include fixed activity	Left/Fix	*1*: Same of less than reference value[b] *-1*: Greater than reference value	Yes
Case 4	Backward	MH	Finite	Yes	Include all activity	Left/Fix	100/Deviation	Yes

[a]Reference value: total man-hours/planning range (daily man-hours assumed to be *1* for all activities in operation).
[b]Reference value: total man-hours/planning range (daily man-hours set as actual value of assigned workload).

of the man-hours of the activity, but in Case 3, the man-hours of the activity divided by the duration of the activity was set as the workload. In other words, it was assumed[4] that the man-hours and duration for the activity were planned for each block, and the man-hours were evenly distributed for the duration of the activity. Additional descriptions for Cases 1 to 3 can be found in Appendix C.

Furthermore, for Case 4, in addition to the consideration of man-hours, the reward was determined using the following method. In the previous case, if there was a workload, a value of *1* was assigned to the day, but to consider the man-hours in the determination of the reward, the total man-hours of the activity was divided by the planned duration, and this value was set as the daily workload. For example, an activity with a planned duration of 10 days and 800 man-hours was assumed to have a daily workload of $800/10 = 80$ man-hours/day. In addition, balancing the load implies summing the daily workloads of all the defined activities for each day and then minimizing the standard deviation (σ_L) of these values. This can be expressed by the following equation: In this case, $\frac{100}{\sigma_L}$, with the inverse of the standard deviation, was defined as the reward. Fig. 10 shows an example of the reward calculation using this definition.

$$\sigma_L = \sqrt{\sum_{d=1}^{D} \frac{(L_d - \mu)}{D}}$$

D : Planning horizon

L_d : Workload at day d

μ : Average workload.

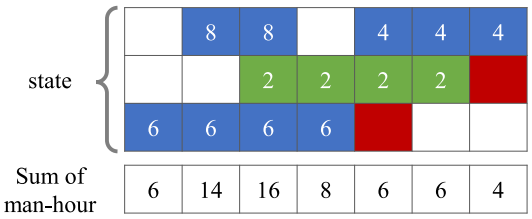
5. Learning cases

Using the developed environment and A3C algorithm, the cases were analyzed, as shown in Table 6. The cases in Table 6 were executed in a manner to overcome the problems of the previous case through sequential learning and analysis. The contents of the learning and analysis for each case are shown below.

5.1. Forward and backward

Using the developed environment and algorithm, learning on Case 1–1 was performed for project number 1962. In Case 1–1, the forward direction planning method was implemented so that the activity moved from the start date of the planning range toward the due date. However, the learning results showed that the agent could not search the state

[4] In actual shipbuilding or architectural planning, the man-hours of unit activities show the distribution in s-curve form. This is because most types of the work have relatively little man-hour input at the start and end of the work, and relatively large amounts of man-hour input in the middle of the work. However, in this paper, it is assumed that the man-hours assigned to the activity are evenly distributed during the start–finish period of the activity.



$$\mu = \frac{(6 + 14 + 16 + 8 + 6 + 6 + 4)}{7} = 8.57$$

$$\sigma_L = \sqrt{\frac{(6 - 8.57)^2 + (14 - 8.57)^2 + (16 - 8.57)^2 + (86 - 8.57)^2 + (6 - 8.57)^2 + (6 - 8.57)^2 + (4 - 8.57)^2}{7}} = 4.24$$

$$reward = \frac{100}{\sigma_L} = 23.6$$

Fig. 10. Example of reward calculation.

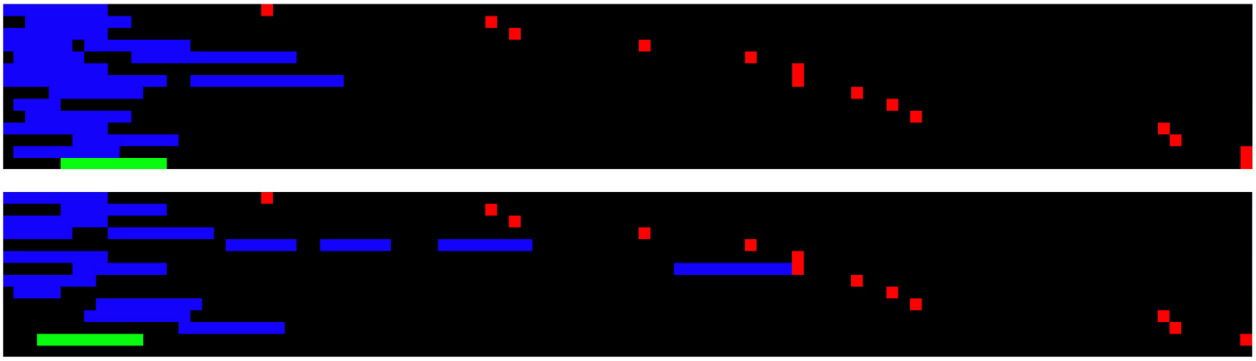


Fig. 11. Learning results of Case 1–1.

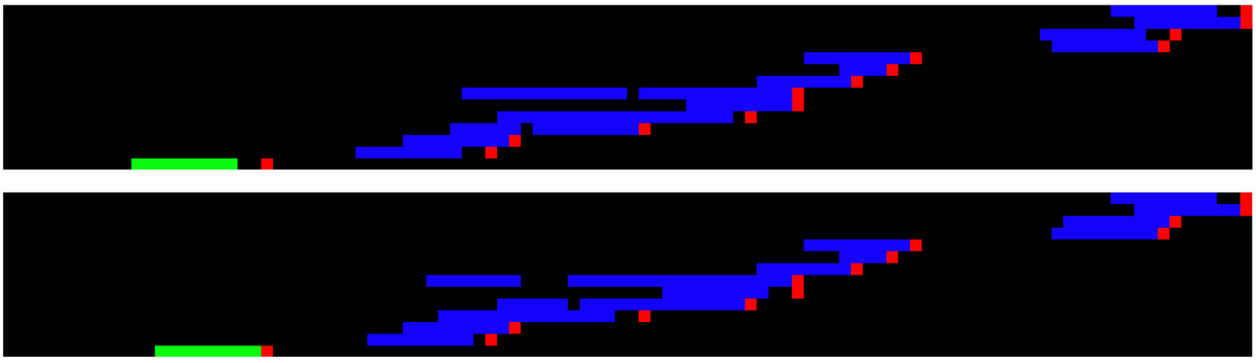


Fig. 12. Learning results of Case 1–2.

space sufficiently and fix the location of the activity near the start date of the planning range, as can be seen in Fig. 11.

Consequently, in Case 1–2, the backward direction planning method was implemented for project number 1962 so that the environment was changed to move the activity from the due date for each block group. In this case, as shown in Fig. 12, more feasible results for production activity load balancing were obtained compared to Case 1–1 (with the forward direction planning method).

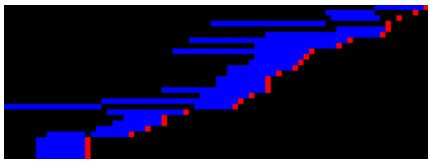
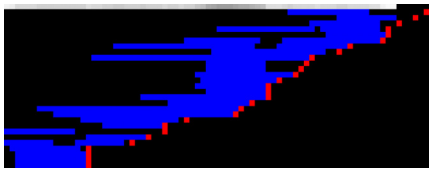
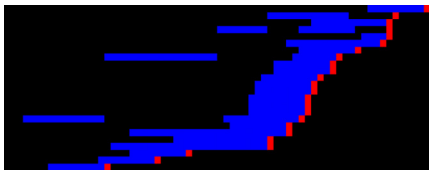
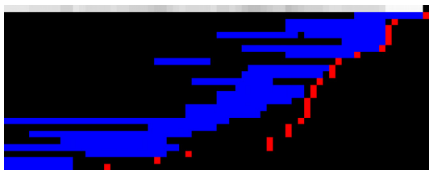
When the method of Case 1–2 was applied to a few projects (2086 and 2095), it was confirmed that the load balancing level improved compared to the initial plan. In other words, it can be seen that the A3C reinforcement algorithm determined a significant move policy for activity in the backward direction.

However because the learning target of Case 1 was for the entire input data, when new data was input, learning had to be performed again, which required considerable computation time. Thus, its application to numerous activities, as shown in the existing optimization methods, such as CSP, GA, and TS, is difficult in practice.

5.2. Time window

In Case 2, the concept of a time window was introduced into the states to allow the trained neural network model to be applied to new input data. As shown in Fig. 13, when states were riven in grid form for the entire planning range and the block groups for planning, a time window was applied to reduce the dimension of the states to the

Table 7
Learning results of Case 1–2.

Ship	Manual	Test
2086	 (Deviation of workload: 3.63)	 (Deviation of workload: 2.99)
2095	 (Deviation of workload: 4.59)	 (Deviation of workload: 2.12)

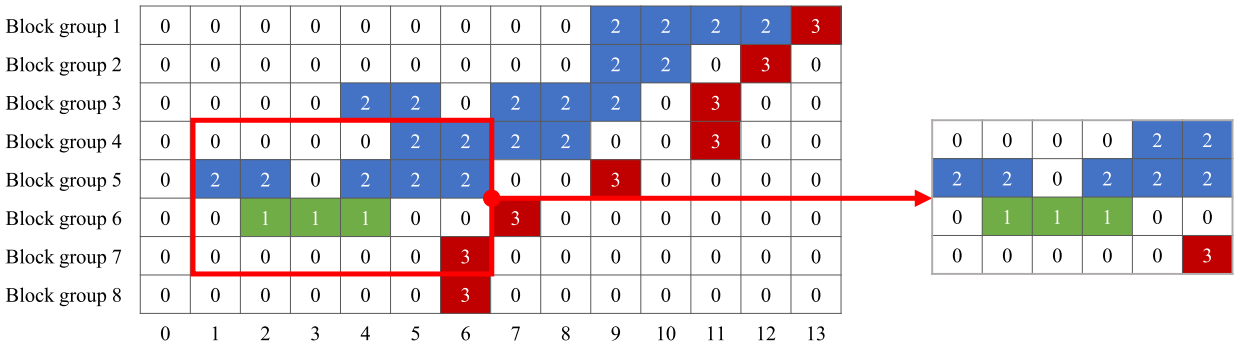


Fig. 13. Concept of time window.

vicinity of the current activity for the planning. At this point, the size of the time window was set as a hyperparameter to enable adjustment of its size. For the test of the time window, the trained neural network for project number 2095 was applied collectively to block groups (101 in activity unit) of project numbers 1962, 2086, and 2095. As a result of the application, it can be seen that the balancing level of the workload improved (from 5.00 to 3.91), as shown in Table 8.

The limitation of Case 2 is that it cannot reflect the actual planned man-hours of the activity. In Case 2, the value input to each cell of the state indicates only the presence or absence of an activity plan on the applicable date. Alternatively, in calculating the reward, the daily workload of each activity is simply set to be *1*. Therefore, in Case 2, load balancing represents balancing for the number of overlapped activities per day, not for the man-hours put into each planning day. However, in actual master planning, the objective is to achieve load balancing considering the planned man-hours of each activity, and thus there is a limitation in applying the learning model of Case 2 without modification.

5.3. Man-hours

In Case 3, to facilitate the application of the learning model in accordance with the objective of master planning, which is the load balancing in consideration of planned man-hours, planned man-hours were reflected as workload and states were configured with the workload for each planning day. As previously described in Section 4.4, the daily workload was calculated by dividing the planned man-hours of each activity by the planned duration, and this was assigned to each cell of the state where the activity was located. In Table 9, the results of the test using the learned neural network showed that the workload deviation decreased from 150.7 to 142.71. It can be seen that the unit of deviation in Cases 1 and 2 changed with the consideration of man-hours. In addition, in Tables 7 and 8, the activities with fixed locations are shown in the same color, but in Table 9, with the consideration of man-hours, values assigned to cells constituting all activities change, and this variation is shown by differentiating the values with brightness. That is, if the man-hours assigned were relatively large, the brightness was darker, and if the man-hours were relatively small, it was lighter.

However, in Case 3 – because there was no separate constraint on the lag between activities with an FS relationship, as shown in the learning result on the right side of Table 9 – an activity with excessive movements occurred. In other words, because there was no constraint between the preceding and succeeding activities for one block group, the interval was excessively widened. In addition, in Case 2, in defining the state, the activities with confirmed plans were assigned a value of *2*, and activities with an ongoing plan were assigned a value of *1*, which enabled differentiation between these two types of activities. However, in Case 3, with the consideration of man-hours, the value of a state was defined as a daily load, unable to differentiate the activities with learning underway, leading to a negative impact on the learning process.

5.4. Lag and minor changes of state

In Case 4, lag constraints were added between related activities. The lag was set at 10 days as a default value and could be adjusted as a hyper-parameter. In addition, the activity currently in the process of learning was added as a separate row to the state, and for the improvement of the state, a separate row displaying the sum of daily workload was also added. Assuming Fig. 14(a) to be the entire problem domain, in the states of Case 4, the time window displayed with bold edges in (a) are the states for learning. Furthermore, as shown in (b), a row for activity with currently ongoing learning was

Table 8
Learning results of Case 2.

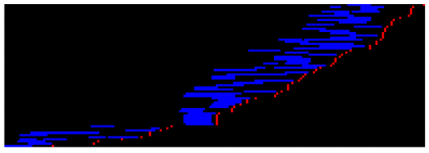
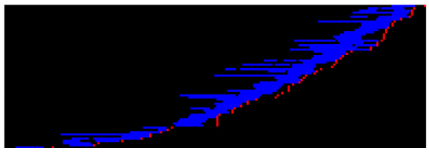
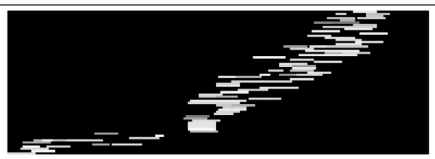
Ship	Manual	Test
1962 2086 2095		
	(Deviation of workload: 5.00)	(Deviation of workload: 3.91)

Table 9
Learning results of Case 3.

Ship	Manual	Test
2086		
	(Deviation of workload: 150.7)	(Deviation of workload: 142.71)

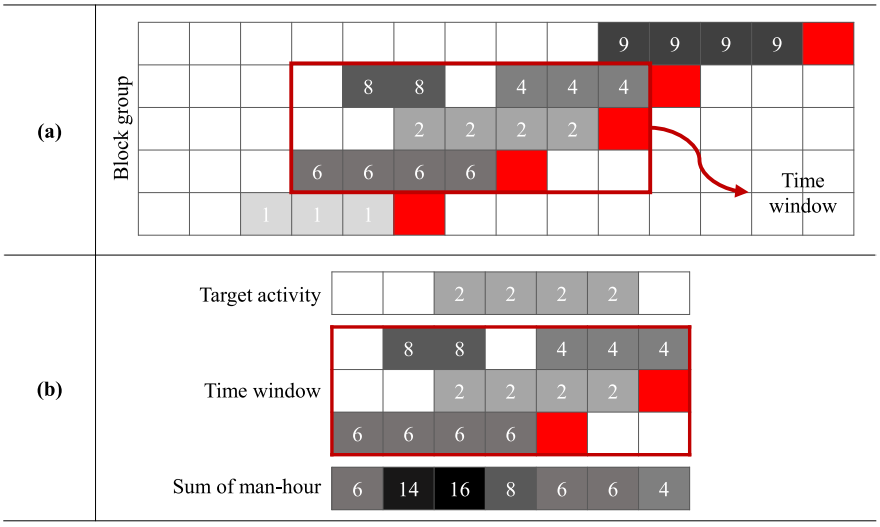


Fig. 14. State of Case 4.

added on top of the time window, enabling the differentiation of the activities with learning underway, and beneath the time window, a row consisting of the summation of daily man-hours was added, enabling the consideration of man-hour distribution for the learning process.

6. Learning results

The Case 4 model determined through various simulations in Section 5 was applied to approximately 100-activity planning. The test cases are outlined in Table 10.

A time window of 15 × 50 and lag constraint of 10 days were set for each test, and the learned neural network with target ships 2028 (41 block groups, 59 activities), 2095 (28 block groups, 42 activities), and P873 (19 block groups, 24 activities) was applied from Tests 1–3. Table 10 shows that each case of Tests 1–3 was designed as a group of block groups corresponding to the three target ships. In addition, as each block group was composed of unit assembly, block assembly, and T/O processes for one or more blocks, the number of activities is also shown in the Table. The merged number is shown together because port assembly and starboard assembly are always performed concurrently, as described in Section 4.1, and these assemblies were merged accordingly—that is, the number of block groups represents the number of rows of the target for which the network, the result of the learning, performs the load balancing planning, and the number of merged activities represents the number of each bar on which learning is performed.

Table 11 outlines the test results of applying the networks learned from target ships 2028, 2095, and P873 for Tests 1–3. The test results are shown in terms of the man-hour deviation for the manually processed work in the shipyard in (a), the man-hour deviation for the initial condition listed in the due date of each block group for all activities before the learning starts in (b), and the man-hour deviation for the results applied with the learned neural networks in (c), and the ratio in each case is shown.

As indicated in Table 11, in six of the nine tests, the result of (c) with the learned network applied demonstrated a higher level of load balancing than those of (a). However, for the results of Test 1–1, Test 1–2, and Test 2–1, the results with the learned network applied exhibited a lower level of load balancing than manual processing.

For all test cases, target ship P873 showed an improved level of load balancing. As can be seen in Table 11, the results of Test 1–3, Test 2–3, and Test 3–3 (which were the tests applied with the neural network learned from the target ship P873 information) showed consistently improved load balancing compared to the manual method. Table 12 shows the test results for Tests 1–3 using the neural network learned

Table 10
Target ships and related number of blocks/activities for each test cases.

Test case	Ship	Number of block group		Number of activities	Number of merged activities	
Test 1	2021	23	78	73	36	110
	2022	35		106	53	
	2027	20		43	21	
Test 2	2028	41	85	118	59	136
	2029	22		76	38	
	2062	22		78	39	
Test 3	1962	24	66	58	29	91
	2095	28		84	42	
	2086	14		40	20	

Table 11
Test results with developed learning algorithm.

Test number	Ship (learning)	Test case	Deviation				Improvement ratio w/ manual [(a)–(d)]/(a)	Improvement ratio w/ initial state [(b)–(d)]/(b)	Improvement ratio w/ initial state [(c)–(d)]/(c)
			(a) Manual	(b) Initial	(c) CSP	(d) Test			
Test 1-1	2028	Test 1	212.78	316.97		233.03	–10%	26%	
Test 1-2	2095					258.36	–21%	18%	
Test 1-3	P873					203.45	4%	36%	
Test 2-1	2028	Test 2	357.82	393.68		365.31	–2%	7%	
Test 2-2	2095					316.97	11%	19%	
Test 2-3	P873					313.23	12%	20%	
Test 3-1	2028	Test 3	185.16	172.94		147.45	20%	15%	
Test 3-2	2095					143.11	23%	17%	
Test 3-3	P873					134.71	27%	22%	

from P873 in comparison with the results of the manual method. (The other results of learning with 2028 and 2095 are added as Appendix D.

Next, to examine the possibility of actual application in industry, it was applied to about 781 activities over a certain period of time (3.5 month), not to a activities of specific ship. The learned neural network is a network that has been learned from ship 2095. As shown in Fig. 15, the learning application result was derived. For the same scheduling target, compared to the workload deviation of 717.32 in the case of manual work, the training application result was 576.91, indicating that about 19.6% of the load leveling improvement was achieved.

7. Discussion

As in Section 6, the effectiveness of the reinforcement learning algorithm presented in this paper can be verified through the application about various test cases. Firstly, the results of the application with several test cases, those are in Table 10, showed improved results in 6 out of 9 cases, and 2 out of 3 slightly poorer test cases showed a difference of less than 10% compared with manual scheduling. In addition, Test 3, which applied the learning result from ship P873, consistently showed better results than the manual scheduling, and it was confirmed that the difference in the application results appeared depending on what data was used for learning.

Next, the result of test with 781 activity scheduling for 781 activities showed an improved output not only in terms of improvement in the level of workload balancing but also in the calculation speed, 5 s, that was not highlighted in the small number of activity scheduling of 100 or less.

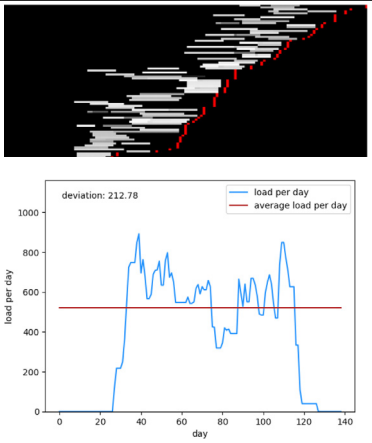
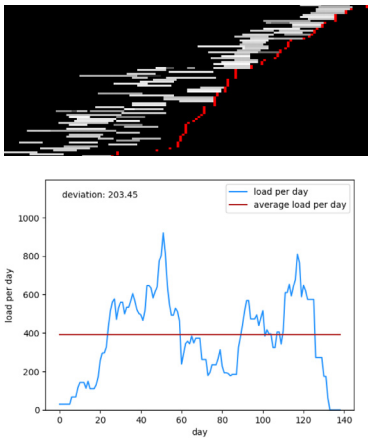
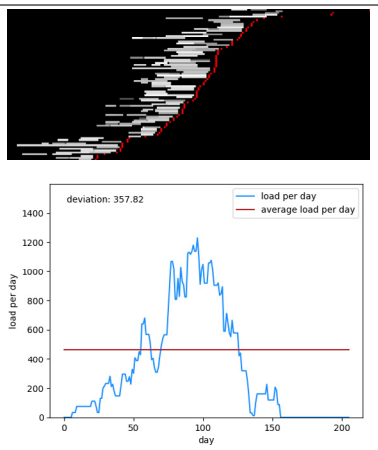
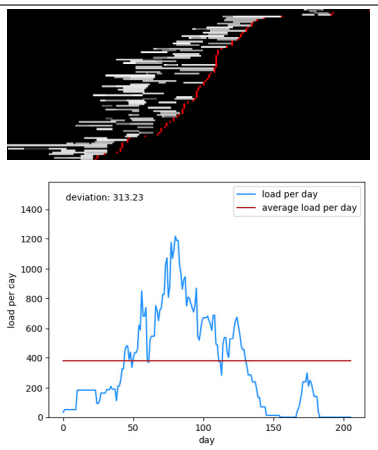
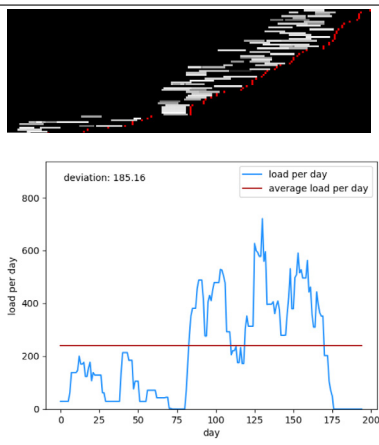
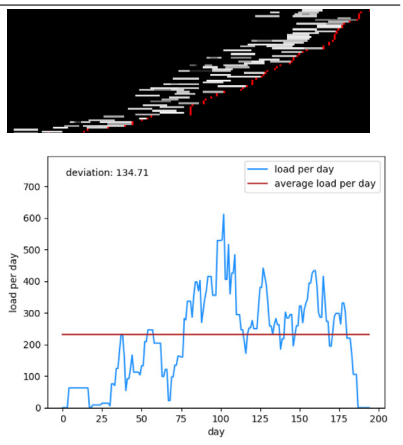
The advantage of the algorithm developed in this paper is that it can automate the manual Gantt scheduling workload balancing. In addition, since the artificial neural network that has been learned can be applied almost in real time, it is possible to establish a plan at a much faster speed compared to manual scheduling. In the case of the test cases in Table 10, the calculation is completed within 1 s even when there are more than 100 activities. In the case of manual work, the variance will vary from person to person, but it takes a lot more time to adjust the activity while observing the change in the workload distribution.[5] In addition, optimization using MIP (mixed integer programming) or CSP (constraint satisfaction problem) method can be applied theoretically, but as the number of activities increases, the computation time increases exponentially, targeting tens of thousands of activities. The scheduling problem with this cannot be solved practically. Of course, more tests and learning are required to apply the neural network developed in this study to a real problem, but it can be said that it is differentiated from the existing method in that the application of the artificial neural network that has been learned is done in near real time. In addition, although it is difficult to quantify, in the case of manual work, even if the same activity scheduling is performed, different results may be shown depending on the person in charge, resulting in poor consistency. However, in the case of learned AI, it is differentiated in that it can ensure consistency of planning work because it enables consistent activity scheduling.

However, the methodology presented in this paper is different from optimization. In the case of meta heuristic (genetic algorithm, simulated annealing, etc.) or CSP (constraint satisfaction problem) methods introduced in Section 1, if only computation time is not a problem, an optimal solution can be derived. Of course, in the actual case of a large project consisting of hundreds of activities, such as a shipyard production scheduling or a scheduling for civil engineering and construction work consisting of hundreds to thousands of activities (in addition to this, it requires repetitive scheduling due to frequent plan changes), meta heuristic and CSP method are not applicable.

Nevertheless, since the methodology presented in this paper does not derive the optimal solution, further research is needed in the

[5] In fact, in the case of shipbuilding production planning, it is known that it takes about a week to establish a 6-month production plan consisting of about 10,000 activities.

Table 12
Test result with a neural network learned from P873.

Ship	Manual	Test with neural network
Test 1–3		
Test 2–3		
Test 3–3		

future. For example, in a continuing study on how to apply a learning algorithm in a various aspect, for example, when the activity is adjusted with the same artificial neural network once again in the learning result shown in Fig. 15, the workload deviation is decreased by about 2% (19% → 21%). So, it is expected that some degree of improvement in scheduling level can be expected through repeated application of artificial neural networks to the target activity case. In addition, in the case of network learning with reinforcement algorithm, it is expected that the shortcomings can be compensated by learning about more various activity patterns through learning from the data of multiple ships, rather than learning only with the activities of single ship.

8. Conclusions

In this study, we developed a planning automation system using the A3C reinforcement learning algorithm for automation of shipyard production planning that has previously been dependent on the manual work of a human planner. The target planning process was the assembly process of the ship blocks constituting the ship, and the study was conducted by obtaining the ship block assembly production planning data given in the form of Gantt planning.

For the reinforcement learning environment, the Gantt-planning environment was simulated in grid form, and the precedent–subsequent relationship between blocks, the man-hours of each assembly activity, and the lag between the related activities were reflected in the simulation. The objective of learning was set to be workload balancing,

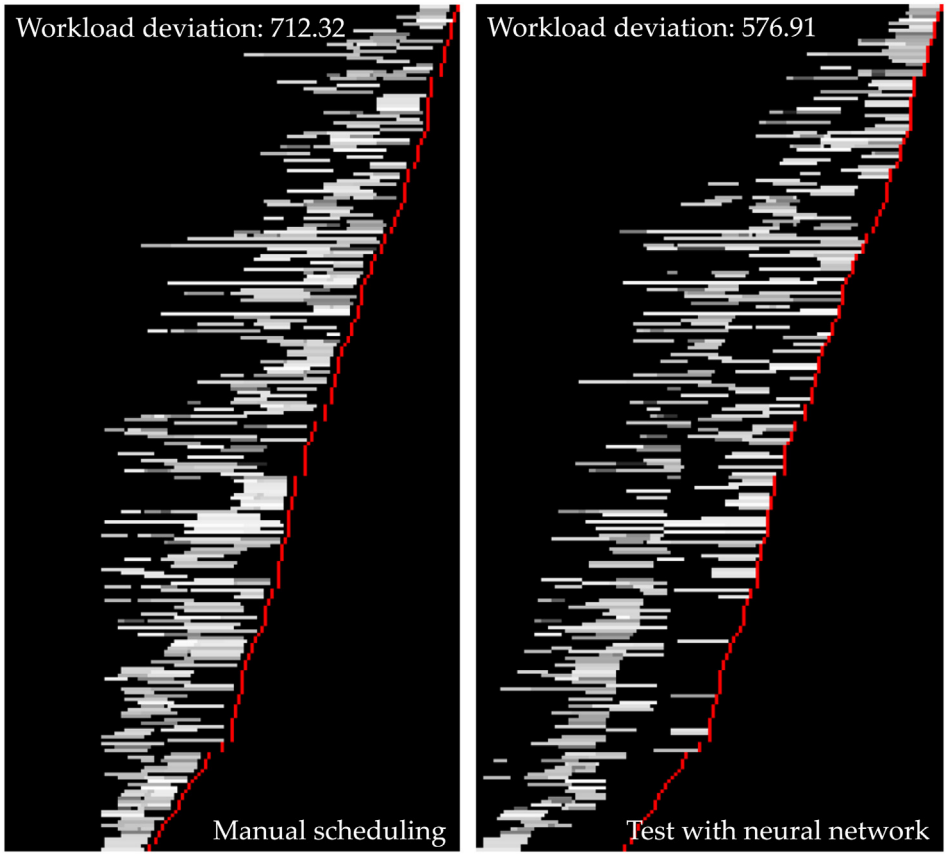


Fig. 15. Test result with a neural network learned from ship 2095 (left: result of manual, right: result of neural network).

and the learning was executed such that the deviation of the daily cumulative man-hours of the activities was minimized.

To reduce the search space dimension, the movement of the activity was performed in the backward direction with reference to the due date of each block, and the concept of a time window was introduced so that re-learning for different input information (block and corresponding activity) was not required, enabling the application of the learning results to various targets. In addition, as the learning result was sensitive to the definition of state and reward, using trial-and-error methods, a case that could achieve successful workload balancing results for various blocks was determined.

The neural network with the A3C reinforcement algorithm learning developed in this study showed a higher level of workload balancing results than human planning with a high probability, demonstrating the possibility that the proposed method could be applied to production planning automation. However, in some cases, it was observed that the workload balancing result was inferior to those obtained by human planning.

In future research, we aim to investigate a method for analyzing the pattern of the planned quantity and performing the learning for a separate agent for each pattern and also to study a method that iteratively applies the time window based on the initially confirmed plan, to gradually modify the plan.

CRediT authorship contribution statement

Jong Hun Woo: Supervision, Writing - original draft. **Byeongseop Kim:** Conceptualization, Methodology, Software. **SuHeon Ju:** Data curation, Investigation. **Young In Cho:** Visualization, Software, Validation, Writing - review & editing.

Declaration of competing interest

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

Acknowledgments

This research was supported by following research projects:
(1) IoT and AI-based development of Digital Twin for Block Assembly Process (20006978) of the Korean Ministry of Trade, Industry and Energy.

Appendix A. A3C algorithm

The flow diagram for the reinforcement algorithm described in Section 3 is explained in detail here. Fig. 16 shows the data flow of the environment and neural network connected around the A3C pseudo code in Table 1. Following the numbers indicated in Fig. 16, the flow of the A3C algorithm is explained as follows.

First, in ① where action selection is made, the actor network calculates the probability for each action in the current state from (policy), and selects an action based on the calculated probability. And sending the selected action to the environment, the action is performed in the environment. In the environment, the simulation is performed according to the selected action, and the next state (s_{t+1}) corresponding to ②and the reward (r_t) for it are returned.

Next, in ③, R corresponding to the target value for the update of neural network is calculated. In detail, R is calculated, which is the target value for updating the critical network, using the rewards obtained from each environment of the state visiting the environment while proceeding N steps for each worker corresponding to the local

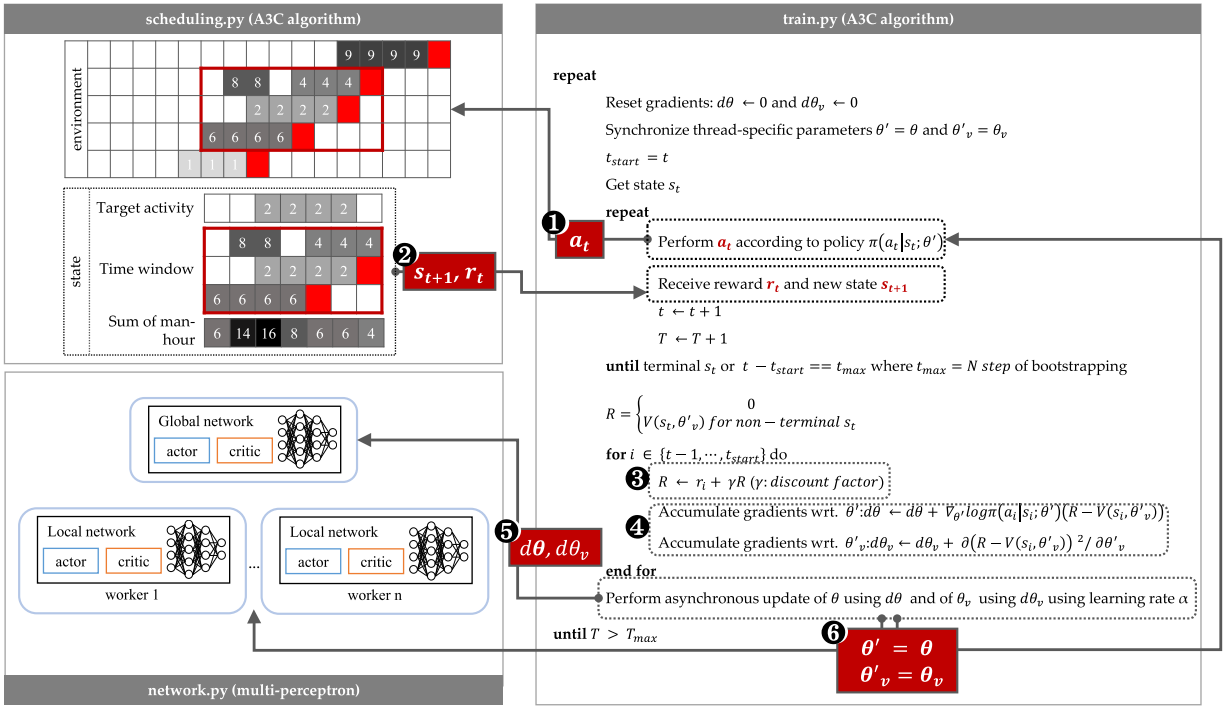


Fig. 16. Detail flowchart of A3C reinforcement algorithm with environment.

Table B.1
Input data of 1962 project (target vessel).

Project	Block	Block group	Process	Start date	Finish date	Duration	Mh	Weight	Stage	Due date
1962	B23P0	B23	04	20190328	20190409	9	392	89	Block assembly	20190411
1962	B23S0	B23	04	20190328	20190409	9	366	88	Block assembly	20190411
1962	B24P0	B24	04	20190329	20190410	9	192	52	Block assembly	20190412
1962	B24S0	B24	04	20190329	20190410	9	177	51	Block assembly	20190412
1962	B338P	B33	05	20190108	20190108	1	2	0	Unit assembly	20190215
1962	B338S	B33	05	20190108	20190108	1	2	0	Unit assembly	20190215
1962	B33P0	B33	04	20190109	20190121	9	214	45	Block assembly	20190215
1962	B33S0	B33	04	20190109	20190121	9	213	45	Block assembly	20190215
1962	B42P0	B42	04	20190228	20190313	8	296	62	Block assembly	20190316
1962	B42S0	B42	04	20190228	20190313	8	296	62	Block assembly	20190316
1962	B510P	B51	07	20190211	20190218	6	290	0	Unit assembly	20190307
1962	B510S	B51	07	20190211	20190218	6	269	0	Unit assembly	20190307
1962	B512P	B51	05	20190219	20190219	1	32	0	Unit assembly	20190307
1962	B512S	B51	05	20190219	20190219	1	32	0	Unit assembly	20190307
1962	B516P	B51	07	20190211	20190218	6	152	0	Unit assembly	20190307
1962	B516S	B51	07	20190211	20190218	6	150	0	Unit assembly	20190307
1962	B51P0	B51	04	20190220	20190305	8	203	163	Block assembly	20190307
1962	B51S0	B51	04	20190220	20190305	8	203	163	Block assembly	20190307
1962	D11P0	D11	04	20190110	20190122	9	410	191	Block assembly	20190311
1962	D11S0	D11	04	20190110	20190122	9	492	194	Block assembly	20190311
1962	E22P0	E22	04	20190103	20190115	9	190	31	Block assembly	20190125
1962	E22S0	E22	04	20190103	20190115	9	194	31	Block assembly	20190125
1962	E512P	E51	05	20190118	20190118	1	25	0	Unit assembly	20190226
1962	E512S	E51	05	20190118	20190118	1	24	0	Unit assembly	20190226
1962	E513P	E51	05	20190108	20190109	2	12	0	Unit assembly	20190226
1962	E513P	E51	06	20190110	20190117	6	44	0	Unit assembly	20190226
1962	E51P0	E51	04	20190121	20190131	9	195	97	Block assembly	20190226
1962	E51S0	E51	04	20190121	20190131	9	159	72	Block assembly	20190226
1962	E620P	E62	07	20190122	20190129	6	223	0	Unit assembly	20190311
1962	E620S	E62	07	20190122	20190129	6	212	0	Unit assembly	20190311
1962	E623P	E62	05	20190123	20190123	1	19	0	Unit assembly	20190311
1962	E623P	E62	06	20190124	20190129	4	143	0	Unit assembly	20190311
1962	E623S	E62	05	20190123	20190123	1	21	0	Unit assembly	20190311
1962	E623S	E62	06	20190124	20190129	4	109	0	Unit assembly	20190311
1962	E624P	E62	05	20190117	20190117	1	3	0	Unit assembly	20190311
1962	E624P	E62	06	20190118	20190123	4	53	0	Unit assembly	20190311
1962	E624S	E62	05	20190117	20190117	1	3	0	Unit assembly	20190311
1962	E624S	E62	06	20190118	20190123	4	21	0	Unit assembly	20190311

(continued on next page)

Table B.1 (*continued*).

Project	Block	Block group	Process	Start date	Finish date	Duration	Mh	Weight	Stage	Due date
1962	E627P	E62	05	20190114	20190114	1	17	0	Unit assembly	20190311
1962	E627P	E62	06	20190115	20190118	4	108	0	Unit assembly	20190311
1962	E627S	E62	05	20190114	20190114	1	17	0	Unit assembly	20190311
1962	E627S	E62	06	20190115	20190118	4	105	0	Unit assembly	20190311
1962	E62P0	E62	04	20190130	20190215	9	276	212	Block assembly	20190311
1962	E62S0	E62	04	20190130	20190215	9	222	203	Block assembly	20190311
1962	E63P0	E63	04	20190312	20190315	4	5	2	Block assembly	20190319
1962	E63S0	E63	04	20190312	20190315	4	5	2	Block assembly	20190319
1962	E912A	E91	05	20190111	20190111	1	9	0	Unit assembly	20190213
1962	E91A0	E91	04	20190114	20190124	9	94	51	Block assembly	20190213
1962	S138P	S13	05	20190208	20190208	1	19	0	Unit assembly	20190321
1962	S138S	S13	05	20190208	20190208	1	19	0	Unit assembly	20190321
1962	S13P0	S13	04	20190212	20190222	9	419	79	Block assembly	20190321
1962	S13S0	S13	04	20190212	20190222	9	419	79	Block assembly	20190321
1962	S21P0	S21	04	20190325	20190404	9	397	100	Block assembly	20190418
1962	S21S0	S21	04	20190325	20190404	9	397	100	Block assembly	20190418
1962	S22P0	S22	04	20190326	20190405	9	588	115	Block assembly	20190418
1962	S22S0	S22	04	20190326	20190405	9	588	115	Block assembly	20190418

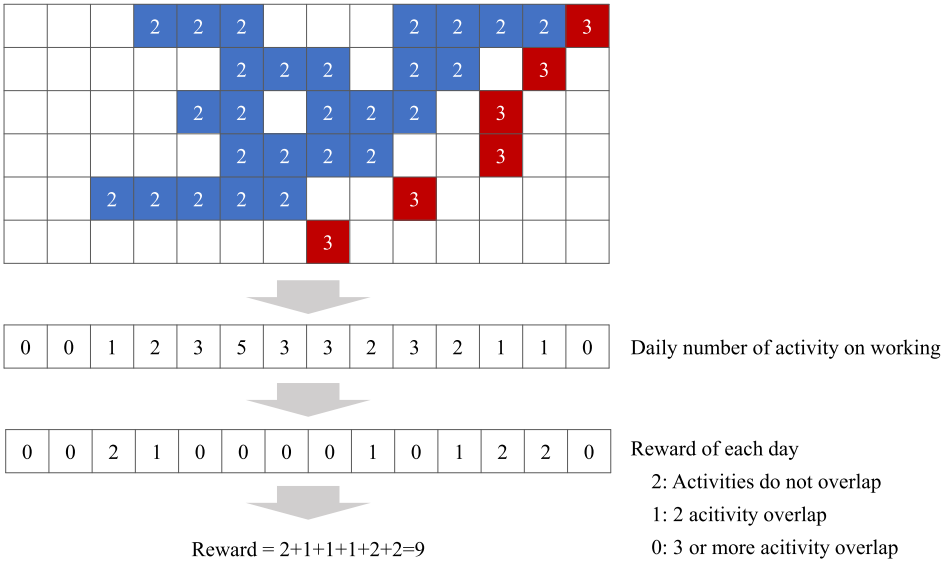


Fig. C.1. Reward calculation of case 1.

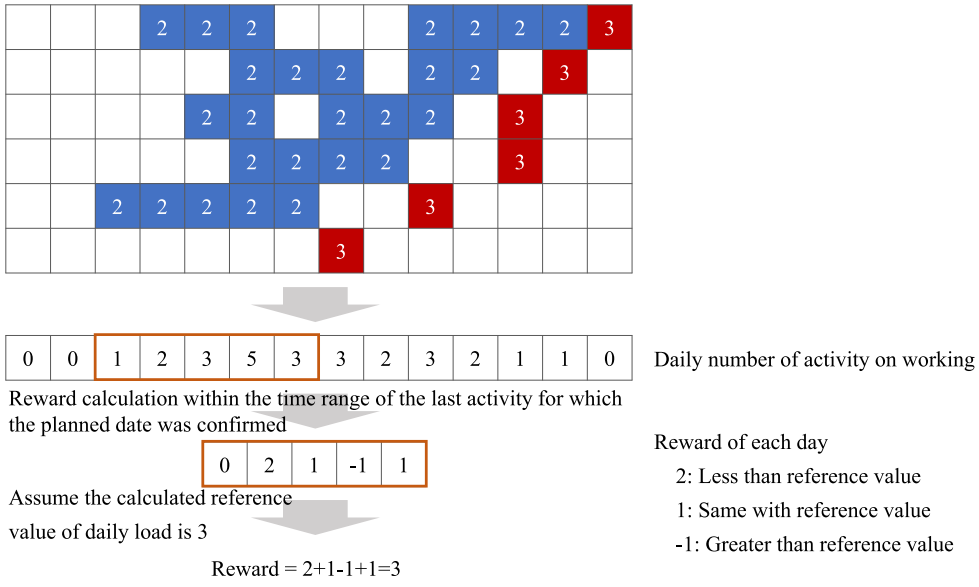


Fig. C.2. Reward calculation of case 2.

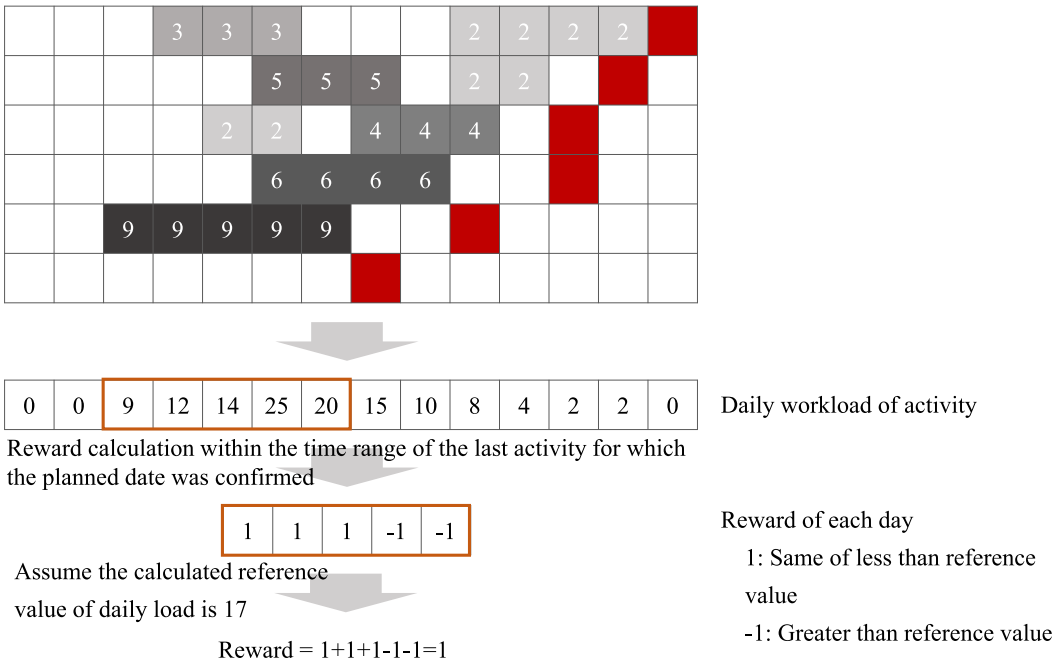
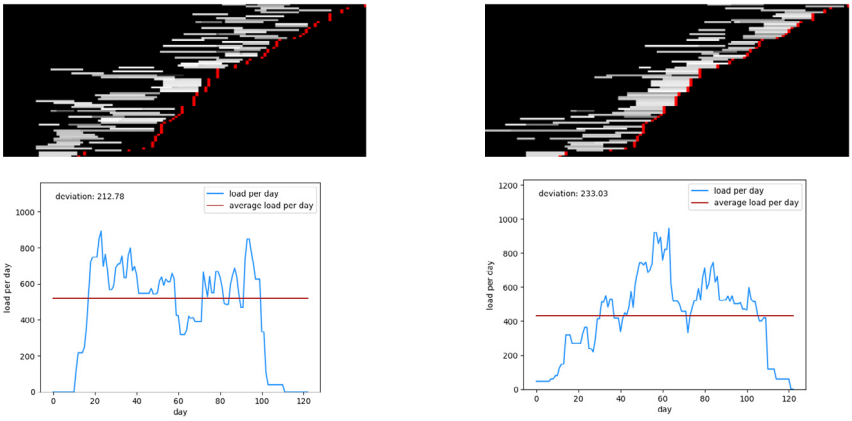
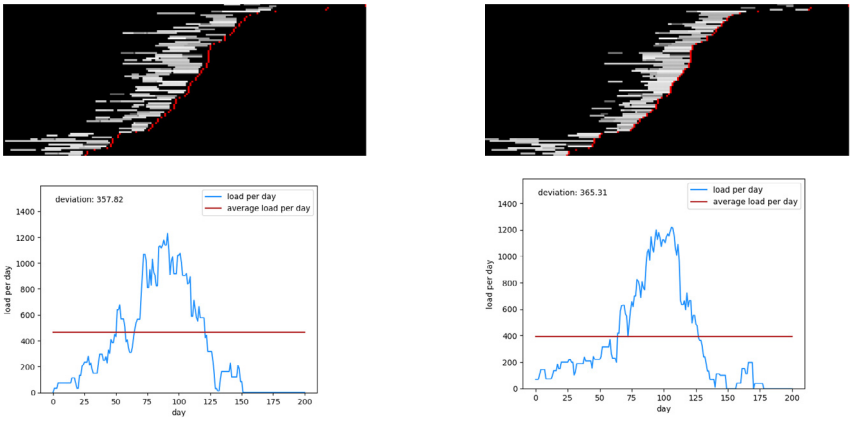


Fig. C.3. Reward calculation of case 3.

Table D.1
Test result with a neural network learned from 2028.

Ship	Manual	Test
Test 1–1		
Test 2–1		

(*continued on next page*)

network. At this time, the target value of each state is set as the sum of the reward received in the next step and the value of the value function of the last state, starting from the state.

When the R corresponding to the correct answer is obtained, ④ calculates the partial derivative value for the weight of the network. More specifically, the slope values ($d\theta$ and $d\theta_v$, respectively) for the

Table D.1 (*continued*).

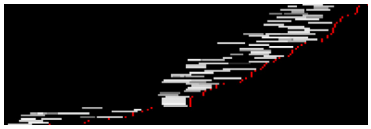
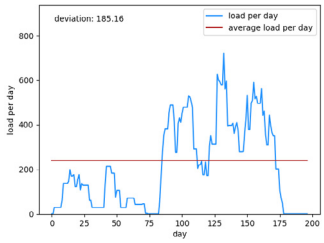
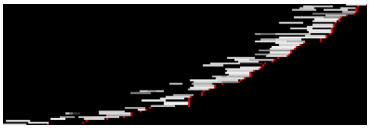
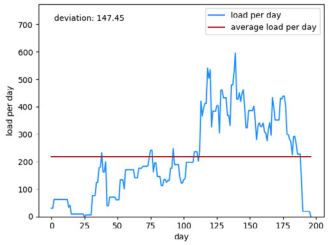
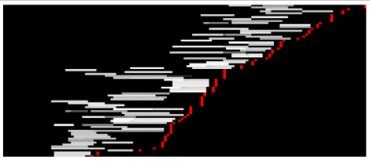
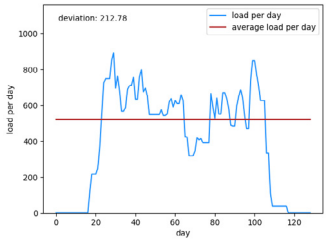
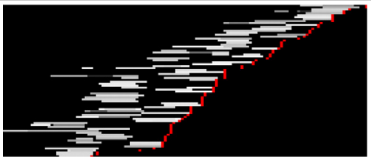
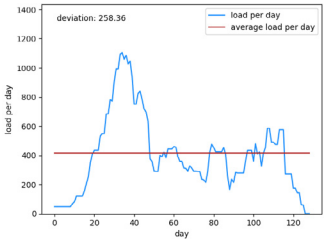
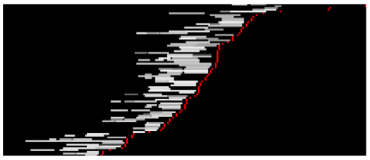
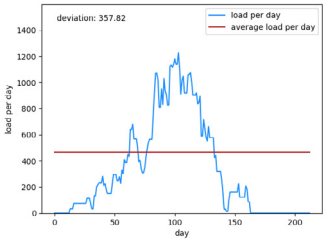
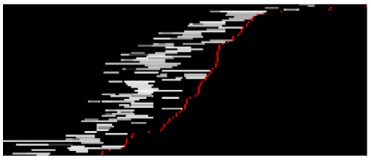
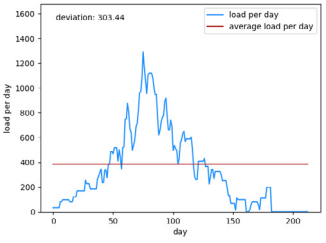
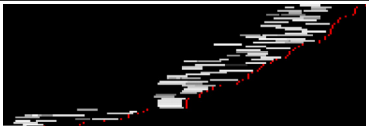
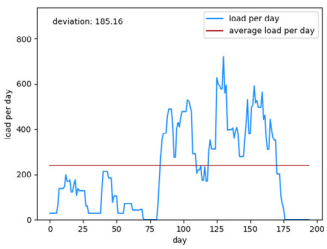
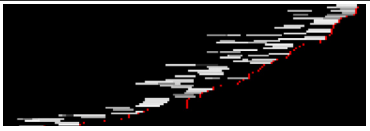
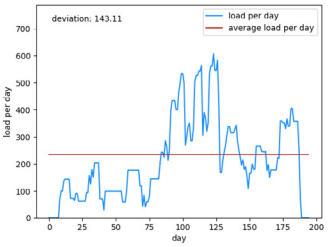
Ship	Manual	Test
Test 3-1	 	 

Table D.2
Test result with a neural network learned from 2095.

Ship	Manual	Test
Test 1–2	 	 
Test 2–2	 	 
Test 3–2	 	 

weights of the actor network and the critical network are calculated. And, using the weight value calculated here, the weight value of the global network is updated as shown in ⑤. That is, the weight of the global network is updated with the slope value calculated in ④.

Finally, the weights of the local network are updated using $d\theta$ and $d\theta_v$. Here, the meaning of update means copying the weight of the global network updated in ⑤ to the local network.

The A3C reinforcement algorithm performs this procedure repeatedly, and in this case, independent threads are executed as much as the number of configurable local networks (or the number of workers). And, local network update is performed asynchronously, as in the meaning of asynchronous, which is the first letter of the A3C name. Not surprisingly, the greater the number of local networks, the faster the convergence speed of learning process, since a wider problem area can be explored at the same time. However, since one worker uses one CPU core, the number of workers that can be set is limited depending on the hardware CPU setting.

Appendix B. Sample input data

Table B.1 is an example of the data used for learning in this study. Project means the ship number, and block and block group are the intermediate product units that make up the ship. Process is a code that expresses the type of detailed process that produces a block. Each block is allocated a pre-planned duration and man-hour required to perform the process for the block. This man-hour is a variable used for load leveling, which is an object function of learning. Stage is a description of a process. Lastly, due date is the completion date for each block process to be completed, and is applied as a constraint upon learning. The start date and finish date were rule-based planning dates performed in advance by humans and were used for comparison with the learning results in this study.

Appendix C. Reward calculation

Case 1

In Case 1, the reward depends on how many activities overlap each day. First, for each day, the number of overlapping activities is counted. Second, for each day, the number of planned activities is converted to a score. Finally, all scores are summed up to calculate the reward (see Fig. C.1).

Case 2

In Case 2, the reward depends on the difference between the daily number of activities and the target value. First, for each day, the number of overlapping activities is counted. Second, the values within the time range of the activity confirmed lastly by the reinforcement learning agent are selected. Third, for each day, the number of planned activities within the selected range is converted to a score. Finally, all scores are summed up to calculate the reward (see Fig. C.2).

Case 3

In Case 3, the reward depends on the difference between the daily workload and the target value. First, for each day, the total daily workload is calculated by summing the workloads of all planned activities. Second, the values within the time range of the activity confirmed lastly by the reinforcement learning agent are selected. Third, the daily workload within the selected range is converted to a score. Finally, all scores are summed up to calculate the reward (see Fig. C.3).

Appendix D. Test results

See Tables D.1 and D.2.

References

Bae, H.C., Park, K.C., Cha, B.C., Moon, I.K., 2007. Scheduling of shipyard sub-assembly process using genetic algorithms. IE Interfaces 20, 33–40.

Basán, N.P., Achkar, V.G., Méndez, C.A., Garcia-del Valle, A., 2017. A heuristic simulation-based framework to improve the scheduling of blocks assembly and the production process in shipbuilding. In: 2017 Winter Simulation Conference (WSC). IEEE, pp. 3218–3229.

Cho, K., Oh, J., Ryu, K., Choi, H., 1998. An integrated process planning and scheduling system for block assembly in shipbuilding. CIRP Annal. 47, 419–422.

Clark, W., 1922. The Gantt Chart: A Working Tool of Management. The Ronald Press Company, New York.

Hameed, Mohammed Sharafath Abdul, Schwung, Andreas, 2020. Reinforcement learning on job shop scheduling problems using graph networks. eprint arXiv:2009.03836.

Hinton, G.E., Osindero, S., Teh, Y.-W., 2006. A fast learning algorithm for deep belief nets. Neural Comput. 18, 1527–1554, https://doi.org/10.1162/neco.2006.18.7.1527.

Hu, S.C., Zhang, Z.Z., Wang, S., Kao, Y.G., Ito, T., 2019. A project scheduling problem with spatial resource constraints and a corresponding guided local search algorithm. J. Oper. Res. Soc. 70, 1349–1361, https://doi.org/10.1080/01605682.2018.1489340.

Hwang, I.-H., Noh, J.-K., Lee, K.-K., Shin, J.-G., 2010. Short-term scheduling optimization for subassembly line in ship production using simulated annealing. J. Korea Soc. Simul. 19, 73–82.

Jeong, J.H., Woo, J.H., Park, J., 2020. Machine learning methodology for management of shipbuilding master data. Int. J. Naval Archit. Ocean Eng. 12, 428–439, https://doi.org/10.1016/j.ijnaoe.2020.03.005.

Kim, B., Jeong, Y., Shin, J.G., 2020. Spatial arrangement using deep reinforcement learning to minimise rearrangement in ship block stockyards. Int. J. Prod. Res. 1–15, https://doi.org/10.1080/00207543.2020.1748247.

König, M., Beißert, U., Steinhauer, D., Bargstädt, H.-J., 2007. Constraint-based simulation of outfitting processes in shipbuilding and civil engineering. In: Proceedings of the 6th EUROSIM Congress on Modeling and Simulation. Citeseer.

Kwon, B., Lee, G.M., 2015. Spatial scheduling for large assembly blocks in shipbuilding. Comput. Ind. Eng. 89, 203–212, https://doi.org/10.1016/j.cie.2015.04.036.

Lee, J.K., Choi, H.R., Yang, O.R., Kim, H.D., 1994. Scheduling shipbuilding using a constraint directed graph search: DAS-ERECT. Intell. Syst. Account. Finance Manag. 3, 111–125, https://doi.org/10.1002/j.1099-1174.1994.tb00060.x.

Lee, Y.G., Ju, S., Woo, J.H., 2020. Simulation-based planning system for shipbuilding. Int. J. Comput. Integr. Manuf. 1–16, https://doi.org/10.1080/0951192X.2020.1775304.

Lee, S.S., Kim, J.W., 2014. Case study for development of maintenance system for equipment of LNG-FPSO topside. J. Ocean Eng. Technol. 28, 533–539, https://doi.org/10.5574/KSOE.2014.28.6.533.

Lee, W.J., Kim, B.H., Ko, K., Shin, H., 2019. Simulation based multi-objective fab scheduling by using reinforcement learning. In: 2019 Winter Simulation Conference (WSC). IEEE, pp. 2236–2247.

Liu, Z., Chua, D.K.H., Yeoh, K.W., 2011. Aggregate production planning for shipbuilding with variation-inventory trade-offs. Int. J. Prod. Res. 49, 6249–6272, https://doi.org/10.1080/00207543.2010.527388.

Mnih, V., Badia, A.P., Mirza, M., Graves, A., Lillicrap, T., Harley, T., Silver, D., Kavukcuoglu, K., 2016. Asynchronous methods for deep reinforcement learning. In: International Conference on Machine Learning, pp. 1928–1937.

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., Riedmiller, M., 2013. Playing atari with deep reinforcement learning. arXiv preprint arXiv:1312.5602.

Romero-Hdz, J., Saha, B.N., Tstutsumi, S., Fincato, R., 2020. Incorporating domain knowledge into reinforcement learning to expedite welding sequence optimization. Eng. Appl. Artif. Intell. 91, 103612, https://doi.org/10.1016/j.engappai.2020.103612.

Rose, C.D., Coenen, J.M., 2015. Comparing four metaheuristics for solving a constraint satisfaction problem for ship outfitting scheduling. Int. J. Prod. Res. 53, 5782–5796, https://doi.org/10.1080/00207543.2014.998786.

Shang, Z.Y., Gu, J.A., Ding, W., Duodu, E.A., 2017. Spatial scheduling optimization algorithm for block assembly in shipbuilding. Math Probl. Eng. 2017, 1923646, https://doi.org/10.1155/2017/1923646.

Shi, Daming, Fan, Wenhui, Xiao, Yingying, Lin, Tingyu, Xing, Chi, 2020. Intelligent scheduling of discrete automated production line via deep reinforcement learning. Int. J. Prod. Res. 58 (11).

Shin, D.S., Park, B.C., Lim, C.O., Oh, S.J., Kim, G.Y., Shin, S.C., 2020. Pipe routing using reinforcement learning on initial design stage. J. Soc. Nav. Archit. Korea 57, 191–197, https://doi.org/10.3744/SNAK.2020.57.4.191.

Shin, H.J., Ru, J.P., 2010. An adaptive scheduling algorithm for manufacturing process with non-stationary rework probabilities. J. Korea Acad.-Indust. Cooper. Soc. 11, 4174–4181, https://doi.org/10.5762/KAIS.2010.11.11.4174.

Woo, S.B., Ryu, H.G., Hahn, H.S., 2003. Heuristic algorithms for resource leveling in pre-erection scheduling and erection scheduling of shipbuilding. IE Interfaces 16, 332–343.

Zhuo, L., Chua Kim Huat, D., Wee, K.H., 2012. Scheduling dynamic block assembly in shipbuilding through hybrid simulation and spatial optimisation. Int. J. Prod. Res. 50, 5986–6004, https://doi.org/10.1080/00207543.2011.639816.