Journal of Manufacturing Systems 60 (2021) 487–499

Contents lists available at ScienceDirect

Journal of Manufacturing Systems

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

Optimizing task scheduling in human-robot collaboration with deep multi-agent reinforcement learning

Tian Yu, Jing Huang, Qing Chang *

Department of Mechanical and Aerospace Engineering, University of Virginia, Charlottesville, VA, 22904, USA

ARTICLE INFO

Keywords:
Human-Robot Collaboration
Real-time task scheduling
Multi-agent reinforcement learning

ABSTRACT

Human-Robot Collaboration (HRC) presents an opportunity to improve the efficiency of manufacturing processes. However, the existing task planning approaches for HRC are still limited in many ways, e.g., co-robot encoding must rely on experts' knowledge and the real-time task scheduling is applicable within small state-action spaces or simplified problem settings. In this paper, the HRC assembly working process is formatted into a novel chessboard setting, in which the selection of chess piece move is used to analogize to the decision making by both humans and robots in the HRC assembly working process. To optimize the completion time, a Markov game model is considered, which takes the task structure and the agent status as the state input and the overall completion time as the reward. Without experts' knowledge, this game model is capable of seeking for correlated equilibrium policy among agents with convergency in making real-time decisions facing a dynamic environment. To improve the efficiency in finding an optimal policy of the task scheduling, a deep-Q-network (DQN) based multi-agent reinforcement learning (MARL) method is applied and compared with the Nash-Q learning, dynamic programming and the DQN-based single-agent reinforcement learning method. A height-adjustable desk assembly is used as a case study to demonstrate the effectiveness of the proposed algorithm with different number of tasks and agents.

1. Introduction

With the development of Industry 4.0, the Human-Robot Collaboration (HRC) has been raising increasing attention from the industry in recent years. A considerable amount of researches and applications have been done on task scheduling and planning issues in human-robot collaborative manufacturing systems. For instance, to resolve the integrated process planning and scheduling problems in a job shop type of flexible manufacturing systems, an object-coding genetic algorithm (OCGA) is proposed [1]. However, most of these studies focus on single-human-single-robot collaboration [2–5]. Various challenges, especially the optimal task scheduling for multiple agents and multiple conflicting objectives in HRC are yet to be addressed properly. Therefore, researches on multi-agent and multi-level task scheduling have become the new trend of studies on HRC in manufacturing systems. For example, Mehta et al. [6] propose a multi-level programming model for multiple robot agents and human agents to maximize the effectiveness of the entire system with limited resources. Chen et al. [7] present an automatic subtask allocation strategy for humans and robots, in which a logic mathematic method is designed to describe this discrete-event system quantitatively. In order to solve the multi-agent task scheduling problem, Gombolay et al. [8] present a centralized task assignment algorithm in conjunction with a mixed-integer program solver. As the HRC systems have been increasingly well studied, a task scheduling policy would be preferable if all necessary human-level and robot-level information are fully incorporated into the policy. However, it inevitably leads to a problem with an enormous state space, which is intractable with traditional searching-based optimization methods. Therefore, it is significant to embrace new tools and methodologies emerging in Artificial Intelligence (AI) and machine learning (ML) areas to develop intelligent decision-making support systems for task scheduling in HRC systems.

In recent years, game-theoretic and reinforcement learning (RL) models and methodologies are widely applied to the multi-agent task scheduling problems [9,10]. It is believed that the equilibrium concept in game theories and multi-agent training methods are highly potent in dealing with multi-constraint and multi-agent optimization problems. For example, Hu et al. [11] introduce a Q-learning based approach for

* Corresponding author.
E-mail address: qc9nq@virginia.edu (Q. Chang).

https://doi.org/10.1016/j.jmsy.2021.07.015
Received 11 March 2021; Received in revised form 3 June 2021; Accepted 9 July 2021
Available online 15 July 2021

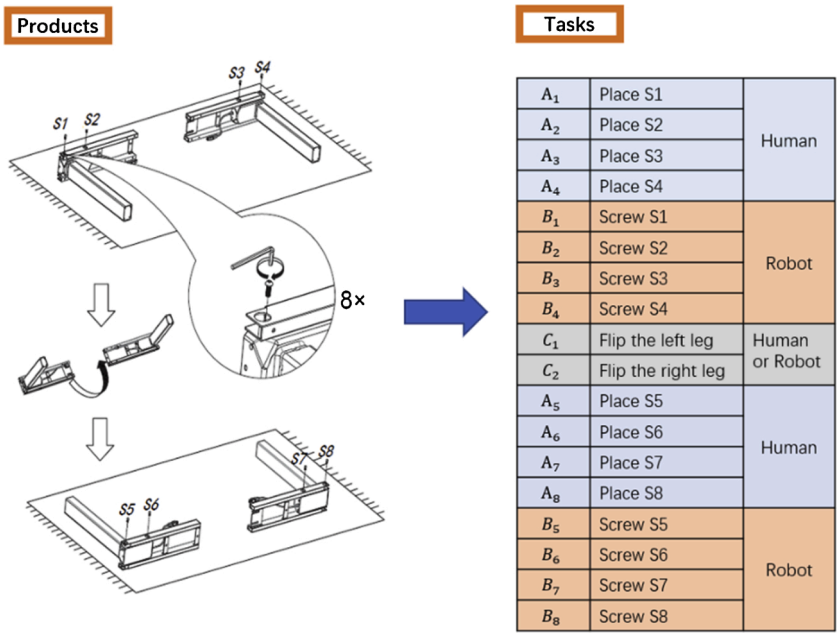


Fig. 1. Diagram of decomposing the product in the human-robot collaborative assembly into tasks for humans, robots or both with a rough assembly sequence.

obtaining Nash equilibria in general-sum stochastic games. Although the proof of convergence of the algorithm is provided for games with finite game and action spaces, their approach is computationally infeasible for all but the simplest examples. Above all, existing approaches are usually applicable within a very small task space or restricted by prior expert's knowledge from static global point of view, which cannot appropriately describe the dynamic process of task scheduling in HRC. It is well noted that the multi-agent task scheduling in HRC is an NP-hard problem [12], therefore, it will be extremely time-consuming to yield an optimal working sequence through traversal-based algorithms for a large task space and complex task structures.

Over the past few years, the development of deep neural networks (DNNs) [13] and significant advances of deep reinforcement learning (DRL) have been witnessed in lots of outstanding large-scale sequential decision-making problems [14–21]. Notably, lots of successful DRL applications model the systems as MARL problems. For example, Duan et al. [22] propose a sequential cooperative game algorithm for cost and completion time optimization while fulfilling storage constraints for large-scale work-flow scheduling. Iranpour et al. [23] proposed a distributed load-balancing and admission-control algorithm based on a fuzzy game-theoretic model for large-scale SaaS clouds.

Inspired by above researches, in this paper, an HRC assembly process is conceived as a chessboard game with specific rules determined by the constraints in assembly tasks. Based on this, the task scheduling problem is formulated into a Markov game model and develop a Deep-Q-network based MARL (DQN-MARL) algorithm aiming at optimizing the completion time of an assembly task. Each DQN agent is trained in an MARL environment by observing all the other agents' actions and rewards to learn the joint distribution action along with environment updates. The resulting task scheduling plans are generated through a self-learning and self-optimizing manner. The proposed approach enables the HRC assembly to work with a large number of state features and a large action space.

The main contributions of this paper are: (1) formatting a typical HRC assembly process as a multi-agent assembly chessboard game with specific mapping rules and playing rules, which will simplify the task scheduling problem with the constraints embedding in the game rules. Such formatting not only clearly reflects the system state but also allow the leverage of the powerful MARL and convolution neural networks (CNN) in cope with complicated task scheduling in the HRC system with large state and action spaces; (2) developing a DQN-MARL algorithm to obtain an optimal scheduling policy without human intervention or expert knowledge, which can be used for efficient online scheduling; (3) generalizing the proposed assembly chessboard game and the DQN-MARL algorithm for a broader task/product family with similar task structures.

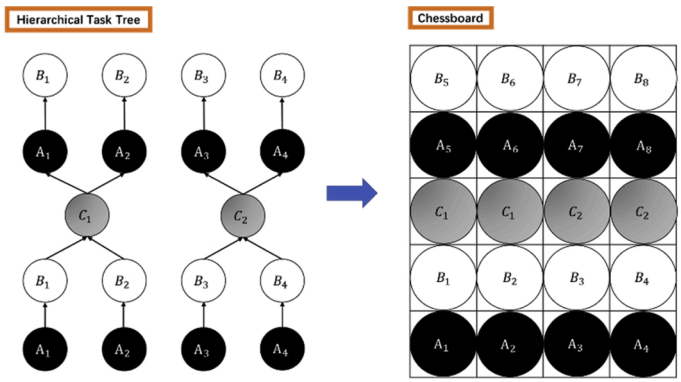
The remainder of the paper is organized as follows: Section 2 describes the HRC assembly problem and the formation of the assembly chessboard game. In Section 3, the task scheduling problem is formulated as a Markov Game by defining the states, actions, and reward function. The problem is solved with DQN-MARL algorithm in Section 4. Section 5 is a case study of implementing the DQN-MARL algorithm in a HRC assembly task scheduling problem. Finally, conclusions and future work are discussed in Section 6.

2. Problem description

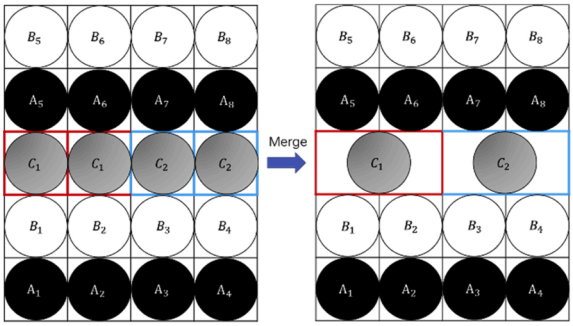
2.1. HRC assembly problem description

In the human-robot collaborative assembly system, the assembly tasks can be distinguished into different categories based on the evaluation of their physical properties and assembly characteristics [24]. To take the advantages of both human and robots, tasks in the human-robot collaborative assembly system can be categorized into three types: type I representing the tasks can be done by humans only, type II representing the tasks can be done by robots only, and type III representing the tasks can be done by either humans or robots. In this research, we do not focus on the task categorization and assume that the task type is given for specific assembly processes. For example, in a desk assembly process as shown in Fig. 1, tasks such as placing screws can be treated as type I tasks because humans are more flexible and faster to place screws into assembly holes. As robots can fasten screws much faster than humans, screwing tasks can be classified as type II tasks. Tasks like flipping and rotating are regarded as type III tasks as they can either be done by humans or robots.

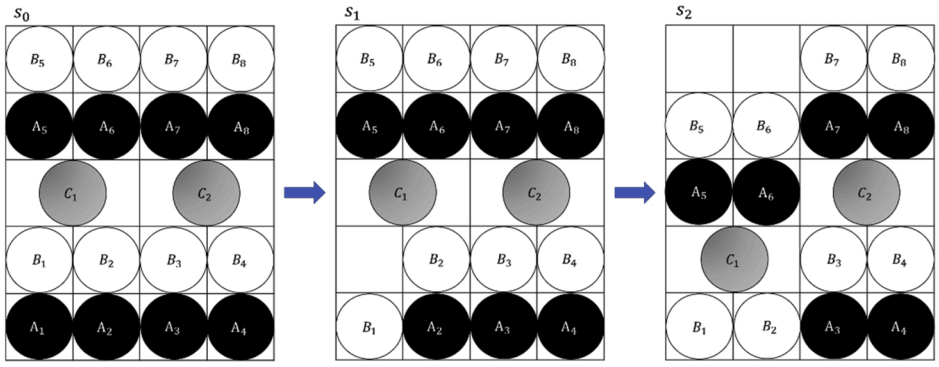
Typically, given a product, the plan of assembly is generated with a very rough assembly sequence based on physical constraints due to the product design, experts' experience or just by the preference of designers. However, there are still many possibilities to improve the assembly efficiency such as when to assign the tasks to the proper agents (i.



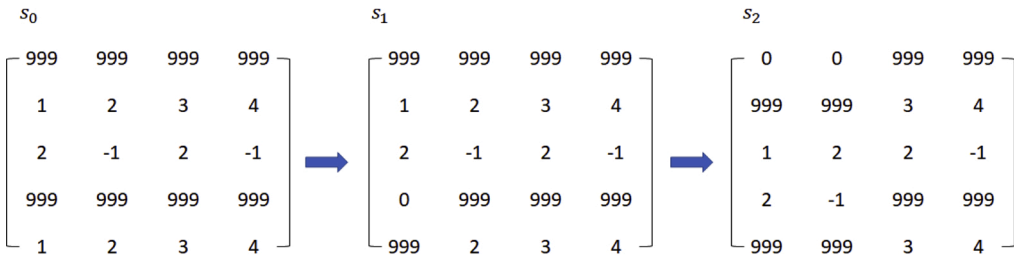
(a)



(b)



(c)



(d)

Fig. 2. Diagrams of the format and transitions of the assembly chessboard. (a) Diagram of mapping the assembly process in Fig. 1 into the assembly chessboard based on the corresponding hierarchical task tree. (b) Diagram of merging the adjacent stones representing the same task in the same row into a united cell. (c) The state of the assembly process and scheme of state transitions. (d) Task matrices of the human agent and the corresponding matrix transformation during the game playing.

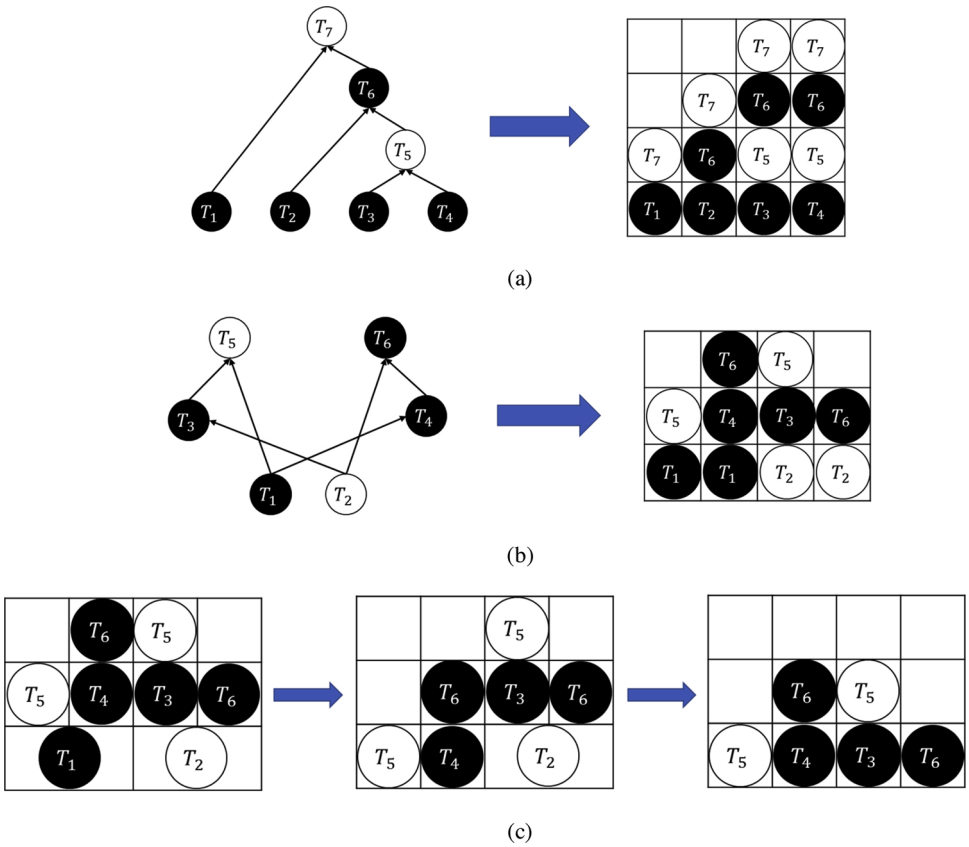


Fig. 3. Mapping the tasks in the hierarchical task tree into the chessboard with (a) task dependency relations and (b) mixed logic relations across tasks of different hierarchical tasks trees. (c) Working flow of assembly tasks.

e., humans or robots) and whether a human or robot needs to take on a type III task. Subsequently, for HRC assembly, it is significant to develop an adaptive method that determines the optimal task scheduling policies according to real-time system states.

In this paper, the problem to solve can be described as: *under a multi-agent HRC assembly environment, develop a method to find the optimal real-time task scheduling policy, such that the overall completion time for the entire assembly is minimized.*

2.2. Assembly chessboard game

First of all, to solve the HRC task scheduling problem described above, the task information and constraints are intended to be formatted into a matrix. Using this matrix as an input, the advanced deep learning method which will be discussed in Section 3 can be used to determine the optimal task sequence and task assignment, especially for type III tasks. Inspired by Alpha Go, the task structure of HRC assembly is formatted into a chessboard with three types of stones representing three types of tasks as shown in Fig. 2 (a). The relations of dependent tasks and/or concurrent tasks are embedded in the assembly chessboard using the chessboard mapping rules along with specific game playing rules, which can perfectly reflect the assembly process.

To take advantage of the assembly chessboard, the following assumptions are made in this paper:

1 Each stone represents a minimum task or a subtask that can be done by only one agent at a time.
2 There are no conflicts among tasks or subtasks.

Chessboard mapping rules:

1 The chessboard has $w \times h$ grids, where the width w is determined by the number of parallel tasks at an assembly step that has the maximum parallel tasks among all assembly steps, and the height h is determined by the number of sequential steps for the entire assembly.
2 Black stones represent type I tasks.
3 White stones represent type II tasks.
4 Grey stones represent type III tasks.
5 By going through each branch of the hierarchical task tree, the stones representing corresponding tasks are mapped into the chessboard with one stone occupying one grid in the chessboard.
6 Tasks with no sequential constraints, are placed in the same row of the chessboard.
7 Tasks with sequential constraints, i.e., tasks on the same branch of the task tree, are placed in the same column with prior tasks being set in the lower row of the chessboard.
8 After all tasks in the hierarchical task tree are mapped into the chessboard, adjacent stones representing the same task can be merged into an untied one.

Chessboard game playing rules:

1 Players can only pick corresponding stones from the bottom row of the chessboard when they are available.
2 Each player can only pick one stone at a time.
3 The stone in the bottom row can only be picked when there are no stones representing the same task in upper rows.
4 Taking the stone away from the chessboard will cost the player the same associated time needed for the task that the stone represents.
5 When a stone in the bottom row is taken away, all upper stones within the same column will fall down one grid if the stones occupy the same size of the grid cells.

6 The game starts after mapping all assembly tasks into the chessboard with corresponding stones and ends until no stones left in the chessboard.

To illustrate the chessboard mapping rules and game playing rules, the desk assembly shown in Fig. 1 is taken as an example. In the first part of assembly, human workers are supposed to place the screw S1, S2, S3 and S4 into assembly holes so that robots can fasten them afterwards. Such placing-and-fastening tasks have sequential constraints. However, which screw to be assembled first has no specific sequential constraints. Therefore, according to the mapping rules, A_1 to A_4 representing task 1 to task 4 are parallel tasks and should be placed into the first row of the chessboard. As sequential tasks, B_1 to B_4 representing task 5 to task 8 are placed into the corresponding columns of A_1 to A_4 but in the second row on the chessboard. After screw S1 and S2 are fastened into assembly holes, the left leg of the desk needs to be flipped for screw S5 and S6 on the other side of the leg to be assembled. As the flipping work C_1 has sequential constraints with both B_1 and B_2, based on the mapping rules, it will be placed into the same column of both B_1 and B_2. In this way, C_1 occupies two grids in the third row. Same rules apply to C_2, the parallel task of C_1. The whole pattern of assembly chessboard is initialized as shown in the first figure of Fig. 2 (c).

After tasks are initialized in the chessboard, the HRC assembly process can be analogized by playing the chessboard game under the playing rules. For example, if the first task A_1 is picked and finished by a human operator, the corresponding stone A_1 will be removed. Consequently, the upper row stone B_1 will move down to the bottom row as shown in the second figure of Fig. 2 (c). It means that at this moment stone B_1 can be done at the same time or at the same level of other black stone tasks at the bottom row. Note that stone C_1 and all upper stones cannot move down at this time, since stone C_1 occupies two grid cells but only one stone below it falls down. Next, if stone A_2 is picked, then stone B_2 will move down one row to fill the bottom grid cell and consequently stone C_1 and all upper stones will also move down as shown in the third figure of Fig. 2 (c).

Applying the mapping rules and game playing rules above, the task structure at any time moment t can be formulated into a $w \times h$ state matrix s_t as shown in Fig. 2 (d). The assembly will start from the bottom row, i.e., the 5th row in Fig. 2 (d). Take one human agent as an example, the positive numbers in matrices in Fig. 2 (d) represent the completion time for corresponding tasks. To bias tasks that might fit better for human operator's capability (type I tasks) or for robot's advantages (type II tasks), the completion time of human agents for type II tasks are set to be a large number such as 999 min and the completion time of robot agents for type I tasks are also set to be 999 min. 0 in the matrix denotes that there is no task in corresponding position. −1 in the matrix denotes a bounding relation with its previous cell. For example, in Fig. 2 (c), stone C_1 takes two grids. Correspondingly, in matrix s_0, $s_0(3,1) = 2$ represents that it will take the human agent 2 min to finish task C_1. $s_0(3,2)$ is set as -1 since no task is needed in this cell but it is bounded with the task in $s_0(3,1)$ as a precondition for the tasks to be performed in the next row, i.e., row 2 in the example of Fig. 2 (d).

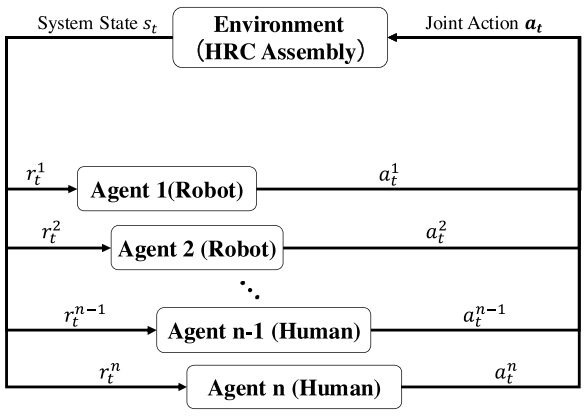


Fig. 4. Overview of MARL framework for task scheduling in HRC assembly.

In some special cases, there exist different task dependency relations and mixed-logic relations for tasks across different task trees. Following the mapping rules, these special structures can also be mapped to chessboards, as shown in Fig. 3 (a) and (b). It is noticed that, in these scenarios, stones representing the same task may not be put into the same row, e.g., T_7 and T_6 in Fig. 3 (a), because of the special task structure. In these cases, one of stones representing the same task may fall to the bottom row during the game process while there are still stones representing the same task left in upper rows. In this situation, the game playing rule 3 works to prevent the stone to be taken away. For example, suppose that the working sequence for tasks in Fig. 3 (b) is $T_1 \rightarrow T_2 \rightarrow \cdots$. As shown in Fig. 3 (c), after T_1 is finished and the stone representing T_1 is taken away, the stone representing T_5 in the same column falls down to the bottom row. However, from the hierarchical task tree we know that T_5 can only be done after T_1 and T_3 are both finished. At this time, based on the game playing rule 3, the stone T_5 in the bottom row cannot be picked because there is still one stone representing T_5 in Row 3, Column 3. The same rule will be applied to T_6 when the stone representing T_2 is taken away.

Remark 1. Using chessboard mapping rules and game playing rules, most common assembly processes can be modelled using this assembly chessboard. Although there might be some limitations in representing the task conflicts and other complicated task relations, the approach proposed in this paper is sufficient to address most assembly problems in the real world. More importantly, the proposed chessboard provides a systematic way to represent largely erratic assembly processes in an intuitive and concise manner, which lays the foundation for the HRC assembly task planning based on state-of-the-art deep learning techniques.

3. Problem formulation

3.1. Notations and assumptions

The mathematical notations and assumptions used in this paper are defined as the following:

1 $TK_u, u = 1, 2, \ldots, U$, represents the u^{th} task with a total of U tasks.
2 Ag_i, $i = 1, 2, \ldots, n$, represents the i^{th} agent. $Ag_i = \begin{cases} 0, & \text{the } i^{\text{th}} \text{ agent is a robot} \\ 1, & \text{the } i^{th} \text{ agent is a human} \end{cases}$
3 T_u^i, represents the i^{th} agent's average completion time on task TK_u.
4 $d_{TK_u}^i(t)$, represents the remaining time for an agent Ag_i to finish a task TK_u at time. For example, if a task A_i has $T_{Ai} = 30\ min$, at time t it has already been done by the operator h for 10 min, then $d_{TK_u}^i(t) = 20\ min$.
5 $H_i(t) \in \{0, 1\}$, represents the availability of the agent Ag_i at time t. If Ag_i is working on task TK_u, then H_i equals to 1. Otherwise, it is set to be 0, i.e.
6 $$H_i(t) = \begin{cases} 0, & d_{TK_u}^i(t) = 0 \\ 1, & d_{TK_u}^i(t) > 0 \end{cases}.$$
7 Humans have the priority of selecting actions over robots when they are available simultaneously.
8 Each agent can be assigned to only one task at a time.
9 Each task should be finished by only one agent.
10 At any time t, the information of each agent is transparent to all other agents.

11 The time each agent takes to complete a given task is random. Agent g_i' s completion time for u^{th} task $T_u^i \sim \Pr(T_u^i = x)$ follows a probability distribution, e.g., normal distribution.

3.2. HRC assembly problem formulation

Based on the chessboard structure and rules described in previous sections, we can have a clear insight of the task options at any moment by observing stones in the bottom row of the chessboard and track the task scheduling process by recording the sequence of stones picked by agents. Therefore, the HRC task scheduling problem can be simplified as: at any moment t, given the format of the task structure as the chessboard and the availability of agents, find the optimal policy to schedule tasks in the bottom row of the chessboard to agents, such that the overall completion time of all assembly tasks is minimized. In this way, the HRC task scheduling problem can be fitted into the Markov game paradigm as shown in Fig. 4, in which humans and robots are abstracted as agents. The state, action and reward function of the MARL framework are defined as following.

Given the information and pattern of the tasks on the chessboard along with the agents' availability at any time t, the system state s_t can be defined as:

$$s_t = \left[p_1(t), \ldots, p_Q(t), d_1(t), \ldots, d_Q(t), \;\; H_1(t), .., H_n(t)\right] \tag{1}$$

where Q is the total number of unfinished tasks; $p_u(t) = (x_u, y_u)$ is the position of the u^{th} task on the chessboard; $d_u(t) \in (0, T_u]$ is the remaining time of the u^{th} task; $H_i(t)$ is the availability of the i^{th} agent.

For each state s_t, each available agent can either pick a task to finish or just wait and take no actions. Thus, the actions of agent Ag_i at time t can be defined as:

$$a_t^i = \begin{cases} 0, & if Ag_i takes no actions \\ TK_u, & if Ag_i picks TK_u \end{cases} \tag{2}$$

The joint action of all available agents at time t forms the decision-making as $\boldsymbol{a}_t = \left(a_t^1, \ldots, a_t^n\right)$

In this problem, the final goal for all agents is to work cooperatively to minimize the overall completion time of all tasks. Therefore, the reward function r_t^i for each agent Ag_i should be directly related to the completion time CT. The reward function is defined as:

$$r_t^i = \begin{cases} 0, & t < CT \\ -CT, & t = CT \end{cases} \tag{3}$$

As mentioned above, at each time moment t, each agent has two actions and the options of which task to be picked also depends on the number of tasks in the bottom row. Therefore, the total assignments of agents to tasks could be a huge number just like selecting moves in the chess game. For complicated assembly jobs, such as automotive assembly, this option space can be exponentially increased. Therefore, the assembly-chessboard game problem may have an ultra-high dimension. A proper algorithm has to be developed to solve this problem both effectively and efficiently.

4. Obtain optimal task scheduling policy through MARL

4.1. MARL framework in HRC assembly

Based on the problem formulations, in the MARL framework, for every state s_t, each agent will select their own actions $a_t^1, \ldots, a_t^n$ and receive corresponding rewards $r_t^i(s_t, a_t^1, \ldots, a_t^n)$. These actions form a joint action $\boldsymbol{a}_t = (a_t^1, \ldots, a_t^n)$, and consequently the state will transit to the next state s_{t+1} according to the transition probability $p(s_{t+1}|s_t, \boldsymbol{a}_t)$ satisfying $\sum_{s_{t+1} \in S} p(s_{t+1}|s_t, \boldsymbol{a}_t) = 1$. With the goal of maximizing their own discounted accumulated rewards, each agent Ag_i select the action a_t^i under the policy π_t^i, which is the decision-making rule corresponding to the probability distribution of the agent's actions. Let $\pi^i = (\pi_0^i, \pi_1^i, \ldots, \pi_t^i, \ldots)$ be the policy of agent Ag_i for the whole game. Given a initial state s, the accumulated reward with discount factor $\gamma \in [0, 1)$ can be represented as:

$$v^i\left(s, \pi^1, \ldots, \pi^n\right) = \sum_{t=0}^{\infty} \gamma^t \mathbb{E}\left(r_t^i \middle| \pi^1, \ldots, \pi^n, s_0 = s\right) \tag{4}$$

4.2. Equilibrium strategies and multi-agent Q-learning

According to [11], in a Markov game, the Nash equilibrium is a joint policy in which each agent's policy is optimal considering the policies of other agents. Based on [11], in a Markov game, a Nash equilibrium point is defined as a tuple of n policies $(\pi_*^1, \ldots, \pi_*^n)$ such that for all $s \in S$ and $i = 1, \ldots, n$

$$v^i\left(s, \pi_*^1, \ldots, \pi_*^n\right) \geq v^i\left(s, \pi_*^1, \ldots, \pi_*^{i-1}, \pi^i, \pi_*^{i+1}, \ldots, \pi_*^n\right) \text{ for all } \pi^i \in \Pi^i \tag{5}$$

where Π^i represents all available policies for agent Ag_i.

To obtain the equilibrium policies in our HRC assembly problem, the multi-agent Q-learning method used in [11] is applied. According to [11], a Nash Q-value is defined as the expected accumulated discounted rewards when all agents follow specific Nash equilibrium policies. For agent Ag_i, the corresponding Nash Q-function Q_{t*}^i at state s_t can be expressed as:

$$Q_{t*}^i(s_t, \boldsymbol{a}_t) = r_t^i(s_t, \boldsymbol{a}_t) + \gamma \sum_{s_{t+1} \in S} p(s_{t+1}|s_t, \boldsymbol{a}_t) s_{t+1} v_t^i\left(s_t, \pi_*^1, \ldots, \pi_*^n\right) \tag{6}$$

where $(\pi_*^1, \ldots, \pi_*^n)$ is the joint equilibrium policy and $r_t^i(s_t, \boldsymbol{a}_t)$ is agent Ag_i' s one step reward in state s_t under joint action $\boldsymbol{a}_t$.

Such extended Q-learning algorithm differs from single-agent Q-learning method in using next state's Q-values to updated current state's Q-values. In the multi-agent Q-learning, agents update their Q-values based on future Nash equilibrium payoffs, while in single-agent Q-learning, agents' Q-values are updated with their own payoffs. In other words, the agent has to observe its own reward as well as all other agents' rewards. At $t = 0$, our learning agent Ag_i learns its Q-value by forming an arbitrary guess, e.g., $Q_0^i(s, \boldsymbol{a}) = 0$ for all $s \in S$. Then, at each time t, agent Ag_i updates its Q-value and calculates a Nash equilibrium $\pi_*^1(s_{t+1}) \cdots \pi_*^n(s_{t+1})$ for the state s_{t+1} according to

$$Q_{t+1}^i(s_{t+1}, \boldsymbol{a}_{t+1}) = (1 - \alpha_t) Q_t^i(s_t, \boldsymbol{a}_t) + \alpha_t \left[r_t^i + \gamma Nash Q_t^i(s_{t+1}, \boldsymbol{a}_{t+1})\right] \tag{7}$$

where

$$Nash Q_t^i(s_{t+1}, \boldsymbol{a}_{t+1}) = \pi_*^1(s_{t+1}) \cdots \pi_*^n(s_{t+1}) \bullet \mathrm{Q_t^i}(\mathrm{s_{t+1}}, \boldsymbol{a}_{t+1}) \tag{8}$$

is the payoff of agent Ag_i in state s_{t+1} for the selected equilibrium.

According to the definition of the Nash equilibrium in (5), applying the equilibrium policy $\pi_*^1(s_{t+1}) \cdots \pi_*^n(s_{t+1})$, each agent Ag_i' s payoff will be maximized. Therefore, to obtain the equilibrium policy, each agent Ag_i need to know all other agents' $Q_t^j(s_{t+1}, \boldsymbol{a}_{t+1})$. However, at time t, other agents' Q-values are not given. Therefore, agent Ag_i has to build its own belief about other agents' Q-values. Same as updating its own Q-value, agent Ag_i guesses $Q_0^j(s, \boldsymbol{a}) = 0$ for all other agents at the beginning. With the propagation of the game, other agents' actions and rewards will be observed and used to updates its belief of all other agents g_j' s Q-value as:

$$Q_{t+1}^j(s_{t+1}, \boldsymbol{a}_{t+1}) = (1 - \alpha_t) Q_t^j(s_t, \boldsymbol{a}_t) + \alpha_t \left[r_t^j + \gamma Nash Q_t^j(s_{t+1}, \boldsymbol{a}_{t+1})\right] \tag{9}$$

In addition, the proof of convergence of the optimality, i.e., the convergence of $(Q_t^1, \ldots, Q_t^n)$ to $(Q_*^1, \ldots, Q_*^n)$ has been provided in [11], which will not be elaborated in this paper.

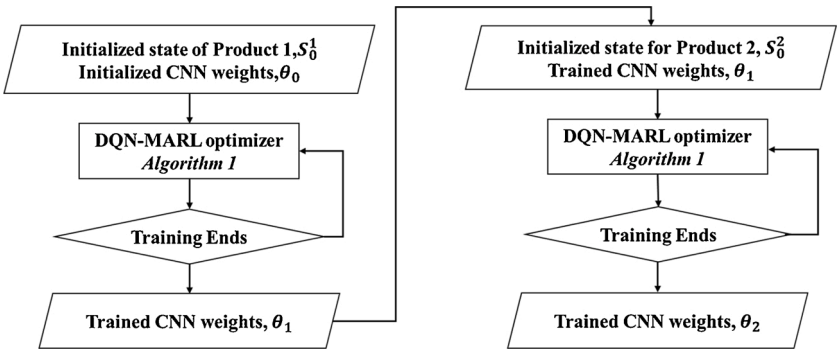


Fig. 5. Flow chart for applying a trained CNN weights from one assembly product to the training process of a new assembly product.

4.3. Applying DQN to obtain task scheduling policy

According to the discussion above, to obtain the equilibrium policy $(\pi_*^1, \ldots, \pi_*^n)$, the agent Ag_i has to maintain and update n Q-values $(Q_t^1, \ldots, Q_t^n)$. Let $|S_t|$ represent the number of all applicable states. Suppose that each agents' action space $|A_t^i|$ are the same, i.e., $|A_t^1| = \cdots = |A_t^n| = |A|$. The total number of entries in $Q_t^j(s_t, \boldsymbol{a}_t)$ is $|S_t| \bullet |A|^n$. If all agents' Q-values are saved in a Q-table, the total memory space requirement is $n|S| \bullet |A|^n$. Therefore, the naive Nash-q method proposed in [11] cannot be put into practice when the state space is very large. The problem with large state spaces is not just the memory needed for large tables, but the time needed to fill and search them accurately.

In recent years, the emergence of DQN algorithms have well addressed the scalability issue in Q-learning. It is a stable and scalable approach to complex and ultra-high-dimensional RL problems, such as Atari video games. In DQN, the Q-values are approximated with a neural network $\boldsymbol{\theta}$, i.e., $Q(s, a^1, \ldots, a^n; \boldsymbol{\theta}) \approx Q(s, a^1, \ldots, a^n)$. Instead of filling a large table, the DQN seeks to iteratively update the neural network parameters, which would well approximate the Q-values until the optimal policy $(\pi_*^1, \ldots, \pi_*^n)$ is obtained eventually.

Remark 2. A critical question raised here is that *can decentralized algorithms be faster than its centralized counterpart* [25]? In this paper, we combine DQN with MARL in Section 4.2 to generate a decentralized algorithm as shown in Algorithm 1. In the proposed DQN-MARL method, to determine the optimal joint actions $\boldsymbol{a}_t$ for each state s_t, agents are trained parallelly to learn the correlated policy $(\pi_*^1, \ldots, \pi_*^n)$ based on the knowledge of other agents' rewards and actions. Suppose there are n available agents and m tasks at time t. Since all agents are trained parallelly, each agent only needs to consider its own possible allocations to tasks, which needs $O(nm)$ computational steps to go through all possible actions for all the agents. In the contrast, in our previous study [26], all allocations of agents' actions are considered in each step and controlled in a centralized fashion, which can be treated as a single-agent reinforcement learning (SARL) method. In the DQN-SARL method, it needs $O(m^n)$ computational steps to go through all possible actions. Therefore, with large number of m and n, the DQN-MARL method can outperform the DQN-based SARL (DQN-SARL) method in terms of computational efficiency, even if the same DQN are applied to the SARL method.

After the offline training using Algorithm 1, the online playing is carried out to implement the DQN-MARL algorithm in practice. Given all the inputs and parameters of Algorithm 1 along with the chessboard simulator, the parameters $\boldsymbol{\theta}$ of the neural network will be obtained through the offline training. Using the trained neural network, the final decision-making process of each agents' task scheduling will be launched online based on Procedure 1. Since Procedure 1 only contains one forward propagation, the time for one step decision-making in the real-time HRC assembly is significantly short (less than 1 s).

4.4. Transferring trained CNN to broader product assembly with different task scenarios

The convolutional neural network (CNN), a type of neural network that is specialized in processing images or tensors, is adopted in the DQN-MARL algorithm. Since tasks in the task scheduling problem are

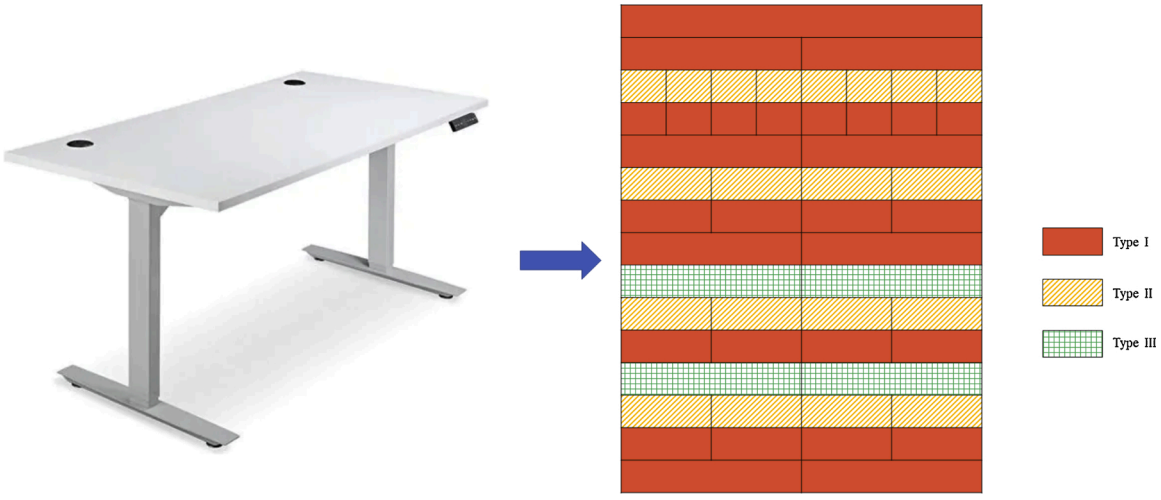


Fig. 6. Diagram of assembly tasks mapped into the chessboard. The red cells denote type I tasks. The yellow striped cells denote type II tasks. The green gridded cells denote type III tasks. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article).

Table 1
Working time for 53 assembly tasks.

Task Name	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
Human (min)	1	2	1	2	3	4	999	999	999	999	2	2	1	2	3	4	999	999
Robot (min)	999	999	999	999	999	999	1	2	3	4	2	2	999	999	999	999	1	2
Task Name	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36
Human (min)	999	999	2	2	1	2	1	2	3	4	999	999	999	999	1	2	1	2
Robot (min)	3	4	2	2	999	999	999	999	999	999	1	2	3	4	999	999	999	999
Task Name	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52	53	
Human (min)	3	4	1	2	3	4	999	999	999	999	999	999	999	999	1	2	3	
Robot (min)	999	999	999	999	999	999	1	2	3	4	1	2	3	4	999	999	999	

mapped into the assembly chessboard, the usage of CNN is motivated due to its capability in capturing characteristic features (referred to as task features) from an image at different levels similar to a human brain. These features can also be reserved by CNN even if they are rotated, flipped or shifted [27]. This attribute of CNN is very significant in practice since structures of assembly tasks for various products in a workshop could be rather similar. These products are referred to as a product family in this paper. Since the Q-value for each state and action pair from a trained CNN will obtain a more reasonable pre-weighted value than a random one, adopting the trained CNN can speed up the re-training of CNN for various products assembly scheduling with similar task features.

Therefore, random patterned chessboards can be generated to represent more general task scenarios accommodating various products assembly tasks, which can then be used to train DQNs to obtain more general task scheduling policies to deal with the change of task structures for various products assembly. However, this training process could be very time consuming. To release the burden, a trained CNN from one product assembly can be re-used to different types of products' assembly through adopting the selected trained weights based on the understanding of the task features. Fig. 5 describes this procedure. This method can largely speed up the re-training process for new product assembly with similar task structures, which will be demonstrated through a case study in Section 5.3.

5. Case study

In order to validate the effectiveness of the proposed method, the assembly of a height-adjustable desk shown in Fig. 6 is used for numerical experiments. Without loss of generality, we assume the assembly tasks include 29 type I tasks, 20 type II tasks and 4 type III tasks. The assembly chessboard is generated based on the installation guide of the desk with each agent Ag_i' s working time for task TK_u follows a normal distribution $\mathcal{N}(T_u^i, \sigma^2)$ with T_u^i shown in Table 1 and $\sigma = 1$ min. For comparison, three other methods, the DQN-SARL method, naive Nash-Q learning method, and the dynamic programming (DP) method (a commonly used value iteration algorithm in dealing with MDP problems [28]), are used as alternative ways to solve the same task scheduling problem. For DQN-SARL method, all allocations of human and robot workers are controlled by one agent, e.g., at each state s_t, the action a_t includes all the combinations and allocations of the available agents to the tasks in the bottom row of the chessboard. The corresponding reward $Q_t(s_t, a_t)$ is generated by a CNN with the same parameters as that for DQN-MARL method. For the naive Nash-Q leaning method, as discussed in Section 4.4, the Q-values of each agent are saved in a Q table. For the DP method, all possible actions of agents for each state are considered and evaluated in a centralized manner. The actions are optimized by backing up estimates of the optimally evaluated state values, which are always saved by updating the evaluation function. In addition, the generalization of the proposed methods for a broader product family assembly with similar task instances is also studied. Two performance metrices are considered: (1) The training time needed for the algorithms to converge to obtain a task scheduling policy (2) The completion time to finish the entire assembly tasks using the trained policy for online scheduling. From the case study, three significant results intend to be concluded: (1) The proposed DQN-MARL method is effective in optimizing the task scheduling; (2) The proposed DQN-MARL method outperforms the naive Nash-Q, the DP and the DQN-SARL methods when the task space and agent number increases in terms of the aforementioned two performance metrices; (3) The well-trained CNN from the proposed method can be beneficial to obtaining optimal task scheduling policies for a broader product family assembly with new task instances when task structures are similar.

5.1. Experiment parameter setting

For demonstration purpose, it is assumed that the average completion time of all human workers are identical and the same assumption works for all robots. The average working time of both human and robot agents are shown in Table 1. To distinguish type I tasks from type II and type III tasks, the completion time of human agent for type II tasks is set as 999 min and robot agent for type I tasks as 999 min.

Given the task parameters, all 53 assembly tasks are mapped into an 8×15 chessboard as shown in Fig. 6. Using this task structure, the completion time of each agent can be easily formatted into a completion time matrix and stack all matrices to form a n^{th} order tensor as the input layer of the CNN in the proposed method. The CNN architecture used in the case study is defined as followings:

- $15 \times 8 \times$ n input layer;
- Convolutional layer with 10 filters of kernel size 2×2;
- 2×2 Max-pooling layer;
- Convolutional layer with 10 filters of kernel size 2×2;
- 2×2 Max-pooling layer;
- Flattening layer;
- Dense layer with 128 units and ReLu activation;
- First output from the dense layer;
- Second output from the dense layer;

The capacity of the replay memory N_{mem} is $5,000$. The training batch size is $b = 32$. The target network replacement frequency C is 2000 episodes. The reward discount rate $\gamma = 0.95$. The parameter for ε -greedy is at first set to be 0.8 and linearly diminished to 0.1 when it reaches 30,000 episodes and fixed to 0.1 for all subsequent episodes. The proposed algorithm is trained using TensorFlow with 4 GPUs and 4 CPUs. The training time t_{train} is limited to $T_{bound} = 1200$ min.

5.2. Effectiveness evaluation for the proposed method

To evaluate the effectiveness of the proposed method in obtaining optimal HRC task planning policy, 100 chessboards with the same task structure but random working time of each task following normal distribution $\mathcal{N}(T_u^i, \sigma^2)$ are generated. After the training time t_{train} reaches its limit T_{bound}, the policy is considered to be obtained for each algo-

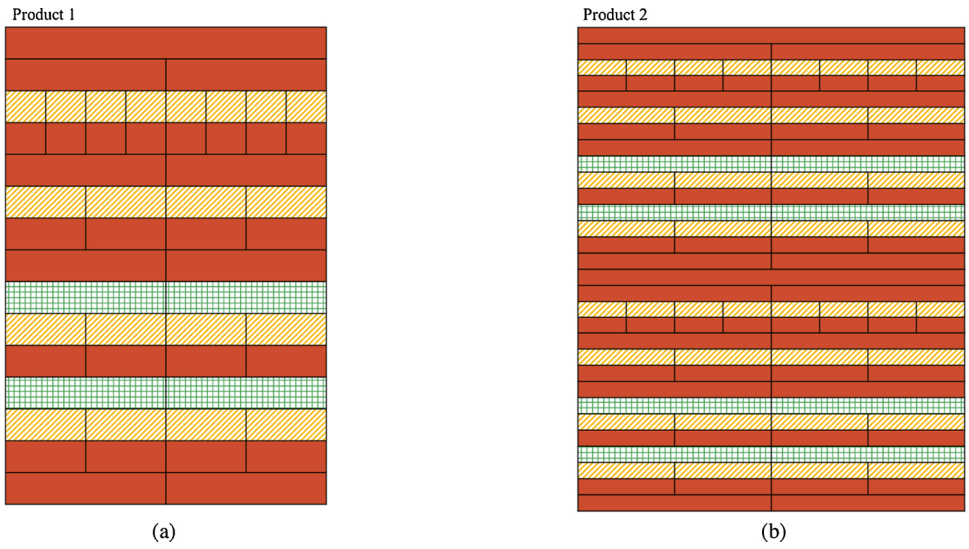


Fig. 7. Assembly chessboard structure of (a) Product 1 (53 tasks) and (b) Product 2 (106 tasks).

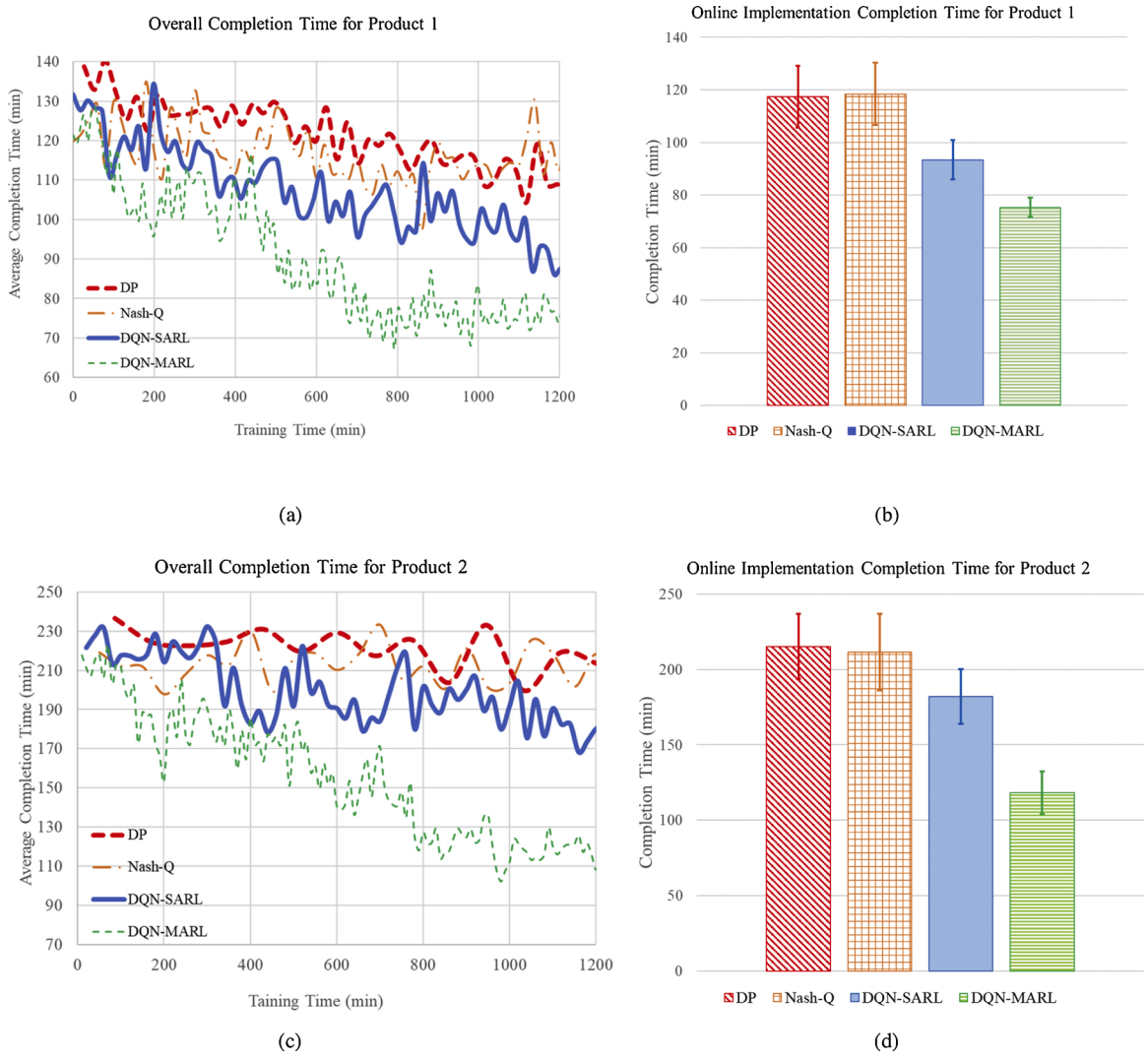


Fig. 8. Diagrams of the offline training (a), (c) and corresponding online implementation process (b), (d) with 3 humans and 3robots for Product 1 and Product 2 using dynamic programing (square dot red line) naive Nash-Q (long dashed orange line), DQN-based SARL (solid blue line) and DQN-based MARL (dashed green line) method. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article).

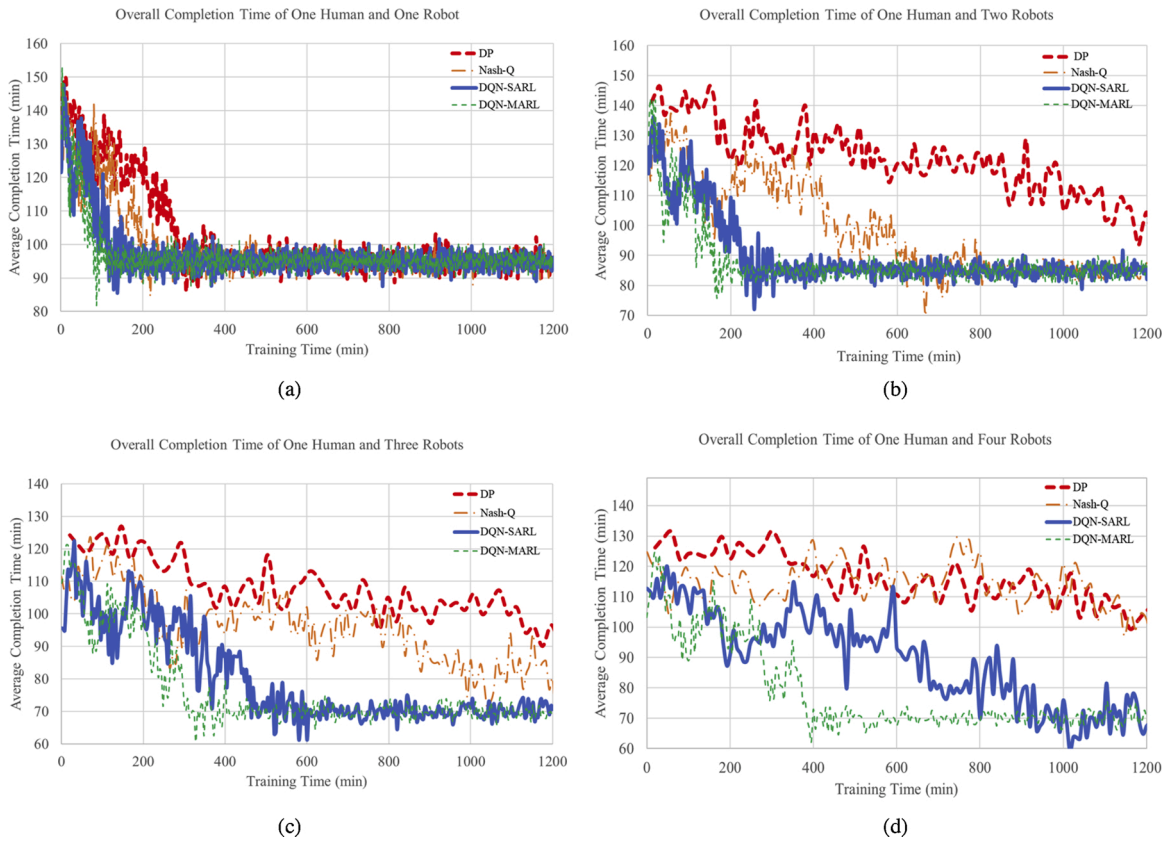


Fig. 9. Training process of four methods with (a) one-human-one-robot, (b) one-human-two-robots, (c) one-human-three-robots and (d) one-human-four-robots.

rithm, and will be used for online task scheduling on additional constructed 100 chessboards with random working time for each task.

First, four aforementioned algorithms are compared in the task scheduling for products with different number of tasks given 3 humans and 3 robots. As shown in Fig. 7 (a) and (b), Product 1 contains 53 tasks while Product 2 contains 106 tasks. The completion time during the training process for the four algorithms are shown in Fig. 8 (a) and (c). After the training processes end, the trained policies obtained by the four algorithms are used for online implementations to assemble Product 1 and Product 2, as shown in Fig. 8 (b) and (d). It can be found that within the given limited training time, only the proposed DQN-MARL method can reach a stable policy for both products although it takes longer to barely obtain one for product 2. It is also noted that the obtained policy based on DQN-MARL clearly outperforms the ones based on the other three methods for online implementation in completing the entire assembly. Therefore, the proposed DQN-MARL method outperforms the other three methods, and its advantage becomes more notable with the increasing number of tasks, as indicated in Remark 2, since the other three methods are centralized schemes. In addition, it is clear that DQN-based algorithms (i.e., DQN-SARL and DQN-MARL methods) outperform

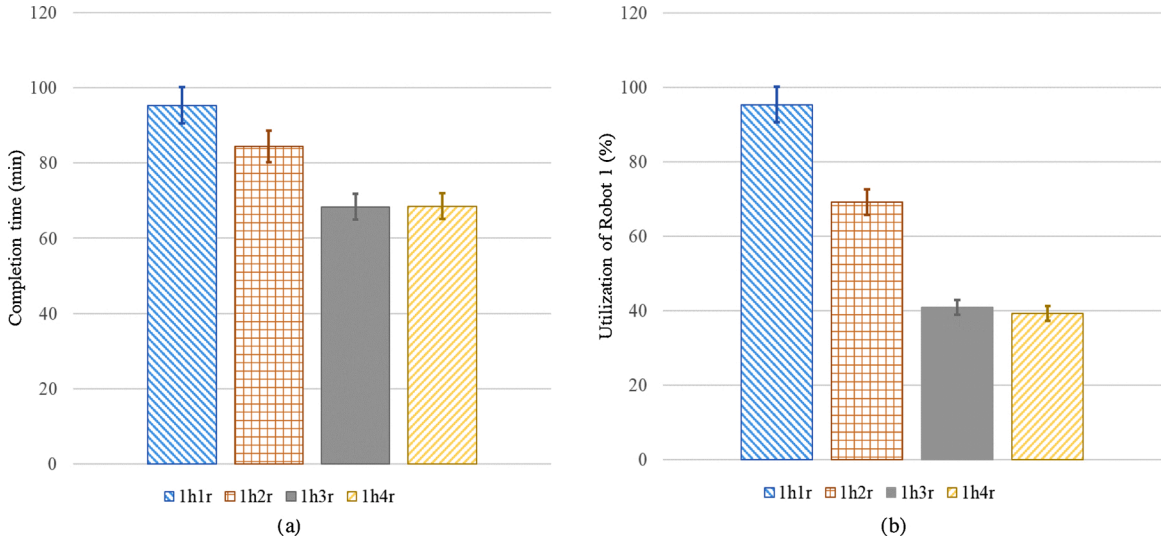
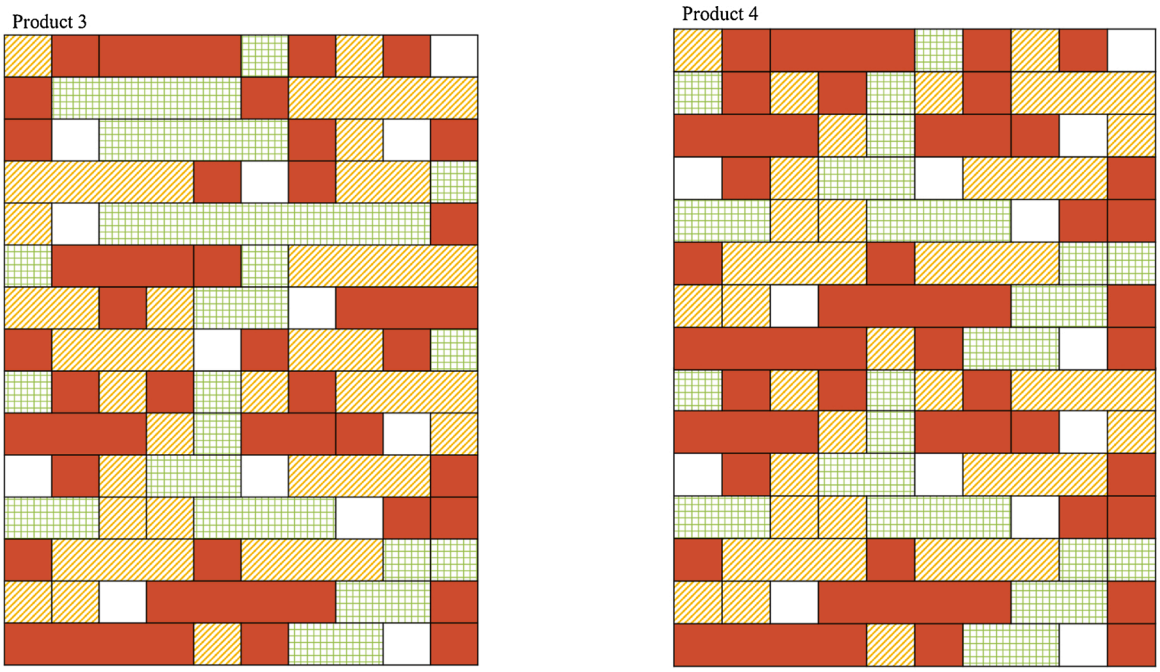
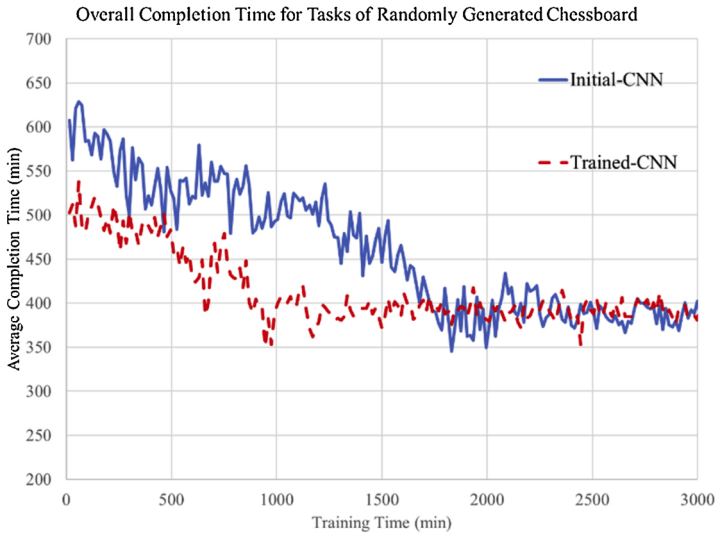


Fig. 10. (a) Completion time of 100 online implementations under one-human-one-robot (1h1r), one-human-two-robot (1h2), one-human-three-robot (1h3r) and one-human-four-robot (1h4r) scenarios using DQN-MARL method. (b) utilization of the 100 online implementations using DQN-MARL method.





(c)

Fig. 11. (a) Assembly chessboard of Product 3. (b) Assembly chessboards of Product 4. (c) Training process of Product 4 with the trained CNN from Product 3 (red dash line) and without the trained CNN from Product 3 (blue solid line). (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article).

the naive Nash-Q method and the DP method in terms of both the computational efficiency during the training period and the optimal completion time during the online task scheduling.

Next, we compare the four algorithms with different number of robot agents. Fig. 9 illustrates the training period of the entire 53 tasks with 1 human and an increasing number of robots from 1 to 4. It can be seen that more time is needed to find an optimal task scheduling policy with increasing number of agents. The proposed DQN-MARL method can still find an optimal policy in a shorter time comparing with the other three methods. Since an optimal task scheduling policy for any algorithm will take full advantage of agents' working time, fewer agents may lead to a scenario close to that of single agent due to the full utilization of agents. When the number of agents increases, the advantage of the DQN-MARL becomes more significant, as shown in Fig. 9 (a)–(d). It is noted that when number of agents reaches to 4 and 5, the non-DQN-based methods can hardly find an optimal scheduling policy. This verifies the analysis in Section 4.3. Moreover, the proposed DQN-MARL method is more effective and computationally efficient than the DQN-SARL method with the increasing number of agents, as indicated in Remark 2.

In addition, we investigate the agent's utilization by implementing the trained DQN-MARL policy for increasing number of robot agents from 1 to 4. As shown in Fig. 10 (a) and (b), with the increasing number of robot agents, both the overall completion time and the Robot 1′s utilization decrease as expected. It is also noticed that the completion

time and the Robot 1′s utilization of the one-human-three-robot scenario are nearly the same as that for the one-human-four-robot scenario. Therefore, for this product assembly, one-human-three-robot is the most efficient setting to complete the entire assembly tasks with minimum time. Additional robots are not necessary and costly.

5.3. Transferring trained CNN to broader products assembly

Additional studies are performed to investigate how to extend the proposed method to assembling various products involving different task scenarios such as the ones shown in Fig. 3. For a general case, a large number of chessboards can be randomly generated (or initiated) to represent various products with various task structures. One can train a general task scheduling policy for a large combination of chessboards representing a broad product family. Indeed, such process will be super time consuming when training process for each type of chessboard (representing a type of product) starts from random scratch. One possible solution is to re-use a trained CNN built from a similar product, as discussed in Section 4.4. To demonstrate this idea, a hypothetic product, product 3 that contains 81 tasks is used, whose task structure is shown in Fig. 11 (a). Applying the DQN-MARL, a CNN and an assembly policy can be obtained for product 3. Now a new product 4 (see Fig. 11 (b)) that has similar task structures to that of product 3 needs to be assembled. The training process for product 4 is shown in Fig. 11 (c), which compares the training with and without adopting the trained CNN from product 3. It is clear that adopting the trained CNN from one product can largely speed up the training process in obtaining a stable task scheduling policy for products with similar task structures.

To summarize, the proposed DQN-MARL method outperforms DQN-SARL, naive Nash-Q and DP method not only in terms of the computational efficiency of obtaining a stable task scheduling policies during the training period but also in the effectiveness of online optimizing the task scheduling, especially in the scenarios with large number of tasks and agents. In addition, the proposed DQN-MARL method can also be generalized for HRC assembly of a broad product family with similar task structures by adopting or re-using a trained CNN.

6. Conclusion

This paper aims to solve a multi-agent task scheduling problem in the HRC environment. The HRC product assembly is formulated to an assembly chessboard game by defining the task mapping rules and game playing rules. A decentralized DQN-MARL method that is capable of obtaining correlated equilibrium solutions of task scheduling is developed. The proposed DQN-MARL algorithm is featured by the combination of the traditional DQN algorithm for RL and the cooperative and correlated equilibrium model. The benefit of the DQN-MARL against DQN-SARL in solving HRC assembly scheduling problem is discussed. Comprehensive case studies based on an assembly of height-adjustable desks are performed. It is demonstrated that the proposed DQN-MARL method is effective in obtaining optimal task scheduling policies for complicated task structures and the method can be generalized to broader similar products assembly. It can be shown that the proposed DQN-MARL method outperforms other machine learning algorithms such as DQN-SARL, naive Nash-Q learning method and DP method in optimizing task scheduling for HRC assembly.

For the future work, more complicated task structures will be considered. Uuncertainties involved in human-robot interactions and multi-agent coordination due to such uncertainties will be further explored and incorporated in the task scheduling. In addition, the proposed method relies on the knowledge of all agents' actions and rewards within the system. We intend to extend the research to scenarios where agents have limited connections and non-perfect information from other agents.

Declaration of Competing Interest

The authors report no declarations of interest.

Acknowledgments

This work was supported by the U.S. National Science Foundation (NSF) Grant. CMMI1853454.

References

[1] Zhang L, Wong TN. An object-coding genetic algorithm for integrated process planning and scheduling. Eur J Oper Res 2015;244(July (2)):434–44. https://doi.org/10.1016/j.ejor.2015.01.032.
[2] Chen F, Sekiyama K, Cannella F, Fukuda T. Optimal subtask allocation for human and robot collaboration within hybrid assembly system. IEEE Trans Autom Sci Eng 2014;11(October (4)):1065–75. https://doi.org/10.1109/TASE.2013.2274099.
[3] Pellegrinelli S, Orlandini A, Pedrocchi N, Umbrico A, Tolio T. Motion planning and scheduling for human and industrial-robot collaboration. CIRP Ann Manuf Technol 2017;66(1):1–4. https://doi.org/10.1016/j.cirp.2017.04.095.
[4] Liu Hongyi, Wang Lihui. Remote human–robot collaboration: a cyber–physical system application for hazard manufacturing environment. J Manuf Syst 2020;54:24–34.
[5] Müller R, Vette M, Mailahn O. Process-oriented task assignment for assembly processes with human-robot interaction. Procedia CIRP 2016;44:210–5. https://doi.org/10.1016/j.procir.2016.02.080.
[6] Malvankar-Mehta MS, Mehta SS. Optimal task allocation in multi-human multi-robot interaction. Optim Lett 2015;9(December (8)):1787–803. https://doi.org/10.1007/s11590-015-0890-7.
[7] Chen F, Sekiyama K, Cannella F, Fukuda T. Optimal subtask allocation for human and robot collaboration within hybrid assembly system. IEEE Trans Autom Sci Eng 2014;11(October (4)):1065–75. https://doi.org/10.1109/TASE.2013.2274099.
[8] Gombolay MC, Wilcox RJ, Shah JA. Fast scheduling of robot teams performing tasks with temporospatial constraints. IEEE Trans Robot 2018;34(February (1)):220–39. https://doi.org/10.1109/TRO.2018.2795034.
[9] Zheng L, Yang J, Cai H, Zhang W, Wang J, Yu Y. MAgent: a many-agent reinforcement learning platform for arti_cial collective intelligence. 2017. p. 1–2.
[10] Lowe R, Wu Y, Tamar A, Harb J, Abbeel OP, Mordatch I, et al. Multi-agent actor-critic for mixed cooperative-competitive environments. In: Guyon I, editor. Advances in neural information processing systems 30. Red Hook, NY, USA: Curran Associates; 2017. p. 6379–90.
[11] Hu Junling, Wellman Michael P. Nash Q-learning for general-sum stochastic games. J Mach Learn Res 2003;4(November):1039–69.
[12] Casgrain P, Ning B, Jaimungal S. Deep Q-learning for Nash equilibria: Nash-DQN. arXiv preprint 2019. arXiv:1904.10554.
[13] Mnih V, Kavukcuoglu K, Silver D, Rusu AA, Veness J, Bellemare MG, Graves A, Riedmiller M, Fidjeland AK, Ostrovski G, Petersen S. Human-level control through deep reinforcement learning. Nature 2015;518(7540):529–33.
[14] Silver D, Huang A, Maddison CJ, Guez A, Sifre L, Van Den Driessche G, et al. Mastering the game of go with deep neural networks and tree search. Nature 2016;529(7587):484–9.
[15] Silver D, Schrittwieser J, Simonyan K, Antonoglou I, Huang A, Guez A, Hubert T, Baker L, Lai M, Bolton A, Chen Y. Mastering the game of go without human knowledge. Nature 2017;550(7676):354–9.
[16] Oliff Harley, Liu Ying, Kumar Maneesh, Williams Michael, Ryan Michael. Reinforcement learning for facilitating human-robot-interaction in manufacturing. J Manuf Syst 2020;56:326–40.
[17] Hu Liang, Liu Zhenyu, Hu Weifei, Wang Yueyang, Tan Jianrong, Wu Fei. Petri-net-based dynamic scheduling of flexible manufacturing system via deep reinforcement learning with graph convolutional network. J Manuf Syst 2020;55:1–14.
[18] Su J, Adams S, Beling PA. Value-decomposition multi-agent actor-critics. arXiv preprint 2020. arXiv:2007.12306.
[19] Kim Yun Geon, Lee Seokgi, Son Jiyeon, Bae Heechul, Do Chung Byung. Multi-agent system and reinforcement learning approach for distributed intelligence in a flexible smart manufacturing system. J Manuf Syst 2020;57:440–50.
[20] Leng Jinling, Jin Chun, Vogl Alexander, Liu Huiyu. Deep reinforcement learning for a color-batching resequencing problem. J Manuf Syst 2020;56:175–87.
[21] Chen Shengkai, Fang Shuiliang, Tang Renzhong. A reinforcement learning based approach for multi-projects scheduling in cloud manufacturing. Int J Prod Res 2019;57(10):3080–98.
[22] Duan R, Prodan R, Li X. Multi-objective game theoretic schedulingof bag-of-tasks workflows on hybrid clouds. Ieee Trans Cloud Comput 2014;2(1):29–42.
[23] Iranpour E, Sharifian S. A distributed load balancing and admission control algorithm based on Fuzzy type-2 and Game theory for large-scale SaaS cloud architectures. Future Gener Comput Syst 2018;86:81–98.
[24] Bilberg A, Malik AA. Digital twin driven human–robot collaborative assembly. CIRP Ann 2019;68(1):499–502.
[25] Lian X, Zhang C, Zhang H, Hsieh CJ, Zhang W, Liu J. Can decentralized algorithms outperform centralized algorithms? a case study for decentralized parallel stochastic gradient descent. arXiv preprint 2017. arXiv:1705.09056.

[26] Yu T, Huang J, Chang Q. Mastering the working sequence in human-robot collaborative assembly based on reinforcement learning. IEEE Access 2020;8:163868–77.
[27] Maddison CJ, Huang A, Sutskever I, Silver D. Move evaluation in Go using deep convolutional neural networks. arXiv preprint 2014. arXiv:1412.6564.
[28] Barto AG, Bradtke SJ, Singh SP. Learning to act using real-time dynamic programming. Artif Intell 1995;72(1–2):81–138. https://doi.org/10.1016/0004-3702(94)00011-O.