Kaven, Lea; Huke, Philipp; Göppert, Amon; Schmitt, Robert H.

Multi agent reinforcement learning for online layout planning and scheduling in flexible assembly systems

Journal of Intelligent Manufacturing

Journal of Intelligent Manufacturing (2024) 35:3917–3936
https://doi.org/10.1007/s10845-023-02309-8

Multi agent reinforcement learning for online layout planning and scheduling in flexible assembly systems

Lea Kaven[1] · Philipp Huke[1] · Amon Göppert[1] · Robert H. Schmitt[1,2]

Received: 31 March 2023 / Accepted: 14 December 2023 / Published online: 27 January 2024

Abstract
Manufacturing systems are undergoing systematic change facing the trade-off between the customer's needs and the economic and ecological pressure. Especially assembly systems must be more flexible due to many product generations or unpredictable material and demand fluctuations. As a solution line-less mobile assembly systems implement flexible job routes through movable multi-purpose resources and flexible transportation systems. Moreover, a completely reactive rearrangeable layout with mobile resources enables reconfigurations without interrupting production. A scheduling that can handle the complexity of dynamic events is necessary to plan job routes and control transportation in such an assembly system. Conventional approaches for this control task require exponentially rising computational capacities with increasing problem sizes. Therefore, the contribution of this work is an algorithm to dynamically solve the integrated problem of layout optimization and scheduling in line-less mobile assembly systems. The proposed multi agent deep reinforcement learning algorithm uses proximal policy optimization and consists of a decoder and encoder, allowing for various-sized system state descriptions. A simulation study shows that the proposed algorithm performs better in 78% of the scenarios compared to a random agent regarding the makespan optimization objective. This allows for adaptive optimization of line-less mobile assembly systems that can face global challenges.

Keywords Production control · Production scheduling · Layout optimization · Multi-agent deep reinforcement learning · Proximal policy optimization · Mobile resources · Flexible assembly

Introduction and motivation

Until now, production aimed at increased productivity in terms of an increased economy of sales, resulting in overproduction. In the light of a shift to more sustainable green production, product life cycles must be extended through the refurbishment of components and products. This refurbishment requires the disassembly and reassembly of many product variants and product generations within one assembly system. Conventional assembly systems with rigid structures are designed for cost-efficient mass production in stable market environments, thus not implementing the needed flexibility. (Hüttemann et al., 2017) More flexible assembly systems, like Lineless Mobile Assembly Systems (LMAS), offer to dissolve temporal and spatial restrictions by mobilizing all products and assembly relevant resources within the factory, using autonomous guided vehicles (AGV), mobile robotics, or worker guidance systems. The movement of assembly stations to new positions during production allows overall system reconfigurations. This high degree of flexibility increases the demand for planning and control. (Hüttemann et al., 2019) A control system must dynamically determine the factory configuration (layout planning) and calculate job routes for incoming products (scheduling) depending on the current system status. To exploit the full potential of LMAS, these decisions must be calculated online during production. (Qin & Lu, 2021) Existing solutions for this planning problem require exponentially rising computational capacities with increasing problem sizes and are very slow in practice. (Moslemipour et al., 2012) Machine learning approaches using neural networks (NN) can deliver

✉ Lea Kaven
l.kaven@wzl-mq.rwth-aachen.de

[1] Laboratory for Machine Tools and Production Engineering (WZL) of RWTH Aachen University, Aachen, Germany

[2] Fraunhofer Institute for Production Technology IPT, Aachen, Germany

fast decisions on complex problems and are robust against varying problem formulations. Solving the complex control tasks proves challenges when using preprogrammed agent behaviors. (Busoniu et al., 2008) Moreover, training data for these machine learning approaches, which is not available for control tasks in LMAS, is unnecessary when reinforcement learning (RL) is used. The planning and control of LMAS consists of two optimization problems (layout and scheduling). RL approaches are mainly limited to systems that focus on one a single task. Multi-agent reinforcement learning provides a promising approach to address this challenge by allowing multiple agents to learn and adapt to the changing environment. (Vithayathil Varghese and Mahmoud 2020; Johnson et al., 2022) Therefore, the contribution of this paper is a multi-agent RL algorithm for the online integrated layout planning and scheduling of LMAS.

Foundations

This section presents the foundations for developing an algorithm for integrated scheduling and layout planning in LMAS. First, the dynamic facility layout problem and the flexible job shop problem are defined. Then, the approach of proximal policy optimization is described. To integrate layout planning and scheduling, the concept of multi-agent reinforcement learning (MARL) is presented. Finally, pointer networks are explained as a RL method to solve combinatorial problems.

Integrated layout optimization and scheduling in flexible production systems

Researchers have differing views on a standardized and exact definition of layout problems. However, the most encountered formulation is the static facility layout problem. (Koopmans & Beckmann, 1957) They define the static facility layout problem as a common industrial problem that aims to configure facilities to minimize the cost of transporting materials between them. As an extension, the dynamic facility layout problem (DFLP) considers the change of material flows between facilities and production periods. (Rosenblatt, 1986) DFLP includes the selection of a static layout for each period and deciding on whether the layout will be changed to another layout in the next period. Therefore, the layout plan is determined to minimize the cost of rearranging facilities between periods and the objective function during periods. (Hosseini-Nasab et al., 2018) Periods are typically fixed and defined by weeks, months, or years. (Drira et al., 2006) In contrast to that, in flexible assembly systems, especially in the case of LMAS, reconfiguration is envisioned to happen both on a medium time scale (e.g., per shift) and on a short-term time scale (i.e., reacting to disturbances) to optimize the utilization of resources. (Hüttemann et al., 2019) This implies that the periods of the DFLP are of stochastic length in contrast to a priori-defined period lengths. The DFLP is a combinatorial optimization problem. (Moslemipour et al., 2012) It is considered an NP-hard problem. (Burggraf et al., 2021).

The job-shop scheduling problem (JSP) is a combinatorial optimization problem in which various manufacturing jobs consisting of operations are assigned to machines while trying to minimize a certain objective. (Zhang et al. 2019) The flexible job shop problem (FJSP) allows an operation to be processed by a given set of alternative machines. Therefore, the FJSP consists of sequencing but also of an assignment of operations to suitable machines (routing). Besides the routing flexibility, it is also possible for jobs to allow a different order of operations (operation flexibility). (Özgüven et al., 2010) The FJSP can be solved statically or dynamically, whereas the latter case corresponds to a scheduling system that can react to real-time events. (Ouelhadj & Petrovic, 2009) FJSP is also considered an NP-hard problem. (Brucker & Schlie, 1990).

In practice, DFLP and FJSP are dependent on each other. Following (Ripon and Torresen 2014), the decomposition of these integrated problems requires prior domain knowledge, and the final solution is sensitive to the solution of the previous stage. Compared to the traditional method, which solves the dynamic optimization problem and the scheduling problem separately, integrated methods can significantly improve the overall performance of the entire process. (Chu & You, 2014) Different solution approaches can be used to solve these combinatorial problems: exact, heuristic, and intelligent approaches. (Drira et al., 2006) Exact approaches require exponentially rising computational capacities with increasing problem sizes and are very slow in practice. (Moslemipour et al., 2012) Approximation methods provide valuable, yet not exact, solutions. They mostly rely on boundary conditions and objective functions that require analytical formulation. This requires assumptions and simplifications for complex environments like assembly systems, which is challenging since it requires high expert knowledge. (Klar et al., 2021) Considering the requirement of real-time solutions and varying problem formulations for a resolution approach in flexible assembly systems, intelligent approaches are potentially suited to solve DFLP and FJSP in LMAS. (Burggraf et al., 2021; Klar et al., 2021).

Deep reinforcement learning and proximal policy optimization

RL can be considered a computational approach to goal-directed learning from interaction. (Sutton & Barto, 2018) DRL methods utilize an NN to acquire and store experience over time. (Goldie & Mirhoseini, 2020) Regarding

the performance of RL methods, actor-critic and policy-based approaches, in contrast to value-based approaches, lend themselves well to high-dimensional and continuous action spaces. (Konda & Tsitsiklis, 1999; Samsonov et al., 2021) Proximal policy optimization (PPO) (Schulman et al., 2017) is one of the most commonly used policy gradient methods. (Petrazzini & Antonelo, 2021; Hsu et al., 2020; Wang et al. 2019) It applies to both, large discrete action spaces and continuous action spaces. (Wang et al. 2019) In contrast to standard policy gradient methods, the PPO objective function enables multiple epochs of minibatch updates. (Schulman et al., 2017) The surrogate objective L^{CLIP} is the key feature of PPO, as it regularizes excessively large policy updates and allows the algorithm to efficiently reuse available data. (Hsu et al., 2020).

The scheduling and layout planning of a flexible assembly can be formulated as a combinatorial problem (cf. Secion "Integrated layout optimization and scheduling in flexible production systems"). To solve combinatorial problems using RL, they can be modeled as sequence-to-sequence problems. (Sutskever et al., 2014) In sequence-to-sequence problems, not a single vector is passed as an input to a NN, but a sequence of vectors. The output, e.g., in a prediction task, is one of the inputs. Sutskever and Vinyals propose the application of RNNs to solve general sequence-to-sequence problems. The idea is to use one LSTM to read the input sequence, one timestep at a time, to obtain a large fixed dimensional vector representation, and then use another LSTM to extract the output sequence from that vector. The second LSTM is conditioned on the input sequence. (Sutskever et al., 2014) Sequences pose a challenge for NN because they require that the dimensionality of the inputs and outputs is known and fixed. Thus, it cannot be used for problems where the size of the output dictionary depends on the length of the input sequence. Vinyals proposed an extension of the sequence-to-sequence model by using a softmax probability distribution as a "pointer". (Vinyals et al., 2015).

DRL approaches are mainly limited to systems that adopted RL algorithms focused on learning a single task. (Vithayathil Varghese and Mahmoud 2020) To concurrently handle the two related tasks of layout-planning and scheduling, methods that tackle multiple tasks using multiple agents are needed. A multi-agent system describes multiple distributed agents who make decisions autonomously and interact within a shared environment. (Weiss, 1999) Each agent decides in each time step and works along with the other agents to achieve an individual predetermined goal. On the one hand, cooperative multi-agent systems are ones in which several agents attempt to solve tasks jointly or to maximize utility. (Panait & Luke, 2005) On the other hand, competitive agents have contradicting goals and compete against each other. Most work in MARL focuses on fully cooperative settings (Foerster et al. 2017). MARL algorithms can further be classified into two frameworks: centralized and decentralized learning (cf. Fig. 1). Centralized methods (Claus & Boutilier, 1998) assume a cooperative game and directly extend single-agent RL algorithms by learning a single policy to produce the joint actions of all agents simultaneously. In decentralized learning (Littman, 1994), each agent optimizes its reward independently. Cooperative MARL problems that are treated using a fully centralized approach can be characterized as centralized training centralized execution. (Gronauer & Diepold, 2021) This centralized approach might fail on relatively simple cooperative MARL problems in practice. (Sunehag et al. 2017) An alternative approach is to train independent learners to optimize for the team reward. This approach can be referred to as distributed training decentralized execution. The centralized training decentralized execution approach bridges the gap between fully centralized and fully decentralizes training and execution. Each agent holds an individual policy that maps local observations to a distribution over individual actions. During training, agents are endowed with additional information, which is then discarded at test time. (Gronauer & Diepold, 2021).

Most work in multi-agent RL has focused on homogeneous team compositions. (Oroojlooy & Hajinezhad, 2021) Team homogeneity allows for parameter sharing among agents and simpler network architectures, which leads to faster and more stable training. However, a multi-agent team will likely have a heterogeneous composition. Agents must leverage their unique abilities and rely on other agents' specializations to cooperate and find effective policies. (Wakilpoor et al., 2020).

Another distinction between RL agents is regarding the synchronicity of the decisions and updates of agents. A system is considered synchronous if there is a global clock and agents move in lockstep. A "step" in the system corresponds to a clock tick. In an asynchronous system, there is no global clock. The agents in the system can run at arbitrary rates relative to each other. One step for agent one can correspond to an arbitrary number of steps for agent two and vice versa. (Halpern, 2007) Only a few RL algorithms consider the problem of multiple agents acting asynchronously (e.g., (Wakilpoor et al., 2020; Calvo and Dusparic 2018)). To address asynchrony, a multi-agent decision process can be viewed as event-driven, where agents choose a new macro-action when prompted by an event in some set of events occurring in the environment. (Menda et al., 2019).

In MARL applications, PPO is significantly less utilized than off-policy learning algorithms. This is due to the belief that on-policy methods are less sample efficient than their off-policy counterparts in multi-agent problems. (Yu et al., 2021) Nonetheless, recent research shows promising results of PPO in multi-agent problems outperforming many off-policy approaches. Witt et al. (2020) demonstrate that independent

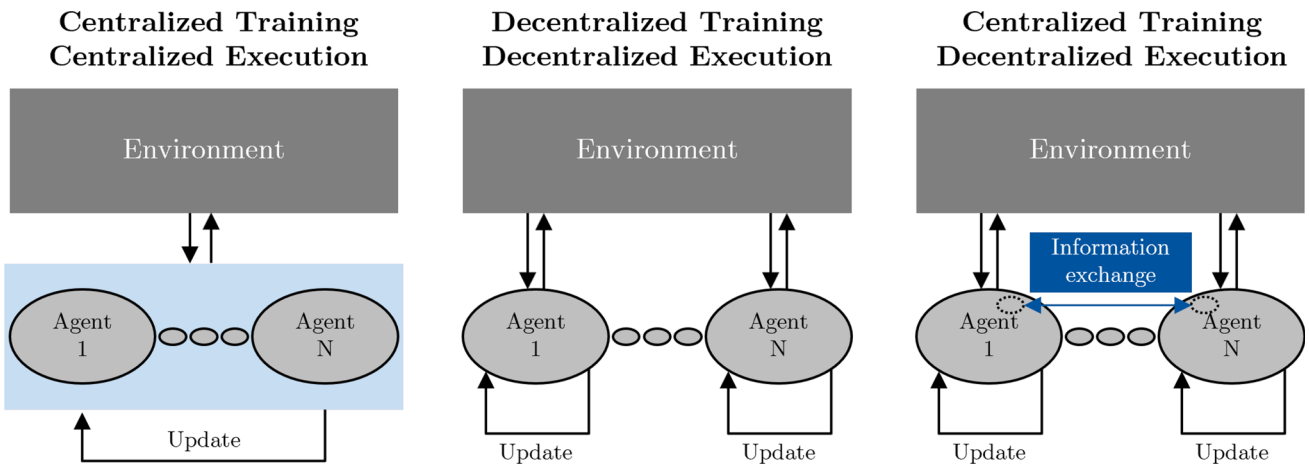

Fig. 1 Training schemes in the multi-agent setting. CTCE (left) holds a joint policy for all agents. Each agent updates its policy in DTDE (middle). CTDE (right) enables agents to exchange additional information during training, which is discarded at test time. (compare (Gronauer & Diepold, 2021))

PPO (IPPO) can outperform state-of-the-art joint learning approaches on popular multi-agent benchmarks. Yu et al. (2021) investigate Multi-Agent PPO, a variant of PPO specialized for multi-agent settings. They show that Multi-Agent PPO achieves surprisingly strong performance in multi-agent benchmarks, with minimal hyperparameter tuning and without any domain-specific algorithmic modifications or architectures. In most environments, they show that Multi-Agent PPO achieves strong results compared to off-policy baselines while exhibiting comparable sample efficiency.

State of the art on reinforcement learning based approaches for scheduling and layout optimization

To solve the layout and scheduling problem in LMAS, an algorithm tackling the integrated DFLP and FJSP is needed. The DFLP must include regular facility shapes with different sizes for different facilities and an open-field layout configuration with movable facilities. The FJSP should include routing flexibility and operation flexibility. Also, the algorithm must handle varying problem sizes and formulations without reformulating the algorithm/problem. Moreover, decisions must be fast to allow for completely reactive online scheduling. Finally, the algorithm must handle stochastic input data due to the system's specifications. As stated in Section "Integrated layout optimization and scheduling in flexible production systems", intelligent approaches such as RL are potentially suited to solve DFLP and FJSP in LMAS. Figure 2 shows potential RL solution approaches found in the literature.

Only Agrawal et al. (2021) tackle the integrated DFLP and FJSP problem. They solve a job scheduling and navigation control task in an autonomous mobile robot-driven shop floor using a MARL framework. The learning problem is modeled as a Markov decision process with communication amongst homogenous agents sharing a common policy. The policy is trained using PPO. The robots (used as AGVs) and humans can move freely on a discrete layout representation. Specific problems arise from this resolution approach, making it inapplicable for LMAS. First, besides the FJSP, this paper tackles a routing problem, not a layout problem. Machines cannot move, and no layout rearrangement is performed. Second, as the agents are exposed to only one machine cycle per episode, they do not learn to halt at a location near a machine. Third, the algorithm may be unable to handle problems of different sizes and formulations. Also, the real-time ability of the agents cannot be assessed correctly.

Several researchers focused on the isolated JSP using RL as a resolution technique. They do not offer complete robustness against varying problem sizes and formulations because the state representation and action space are linked to the size of the problem, i.e., the number of machines. Only a few approaches can reach good robustness (over 100% increase of problem size during inference), using graph representation learning as a technique for encoding the state of the problem. (cf. (Park et al., 2021; Hameed and Schwung 2023; Jing et al., 2022; Oren et al., 2021)) Park et al. (2021) propose a policy network to handle different-sized graph inputs. However, the approach is not stochastic, and routing flexibility is not given. Oren et al. (2021) use a Deep Q-Learning approach to predict an action for each state. The GNN computes the Q-values. Thus, the GNN is used both for encoding

		Hameed et al. (2023)	Jing et al. (2022)	Agrawal, Won et al. (2021)	Samsonov et al. (2021)	Park et al. (2021)	Joel et al. (2021)	Unger and Börner (2021)	Klar et al. (2021)	Ma et al. (2019)	Kim et al. (2020)	Xu et al. (2020)	Di and Yu (2021)	Mirhoseini and Goldie (2021)	This publication
DFLP	Regular facility shapes with different sizes for different facilities	○	○	●	○	○	●	●	●	●	●	●	●	●	●
DFLP	Open-field layout configuration with movable facilities	○	○	○	○	○	○	●	○	○	○	○	○	○	●
JSP	Stochastic input data	○	●	●	○	○	●	●	○	○	○	○	○	○	●
JSP	Uncertain model	○	●	●	○	●	●	●	○	○	○	○	○	○	●
JSP	Routing flexibility and operation flexibility	◑	●	●	○	○	○	○	○	○	○	○	○	○	●
JSP	Event-driven scheduling	●	●	●	●	●	●	○	○	○	○	○	○	○	●
Alg.	Online ability	●	●	●	◕	◑	◕	◔	◔	●	○	○	○	●	●
Alg.	Robust to different problem sizes	◕	◑	◑	◑	◕	◕	○	○	●	○	○	○	●	◕

Fig. 2 Reinforcement learning based approaches for scheduling and layout optimization

and decoding. The algorithm does not include operation flexibility. Hameed et al. (2023) also utilize a GNN for encoding state, enabling event-driven online scheduling decisions, but the algorithm lacks operational flexibility. Jing et al. propose a MARL based on a graph convolutional algorithm for FJSP, including routing and operational flexibility. Unger and Börner (2021) are the first to propose RL as a resolution technique for solving the isolated DFLP using a PPO RL algorithm and an actor-critic algorithm. The current state of the assembly system is encoded in colored pictures, allowing the representation of arbitrary many entities in one picture. The agent can place one element in a continuous space in x- and y- directions in each interaction step. A turn corresponds to the iteration of all system elements that need to be arranged. The approach is only able to handle problems of small size. Klar et al. (2021) propose a Double Deep Q-Learning algorithm to place four functional units next to a lane where AGVs transport the material. Like Unger and Börner, all units are placed one after another. It can be assumed that the problem size and formulation must not change for a trained algorithm. Since the state space shown in the paper is small, high decision time is assumed for large state spaces.

Since no feasible RL resolution approach solving the DFLP in LMAS exists, resolution approaches from other use cases have been investigated. Ma et al. (2019) propose a hierarchical RL approach for tackling the Traveling Salesman Problem (TSP). The TSP is a classical combinatorial optimization problem that is known to be NP-hard. (Gavish & Graves, 1978) By solving an NP-hard problem with restrictions, the problem is partially related to the integrated DFLP and FJSP. A graph pointer network (GPN) consisting of an encoder and a decoder approximately solves the TSP. The algorithm is generalizable for different problem sizes. Kim et al. (2020) propose an actor-critic RL approach to tackle a spatial rearrangement problem. The algorithm is dependent on the problem size and not suitable for large problem sizes. Xu et al. (2020) employ a GNN to tackle a tiling problem. The GNN solves the tiling problem by learning features to predict probabilities for tile placements. The approach is not applicable to LMAS, as uncertainties (e.g., statistical distributed machine breakdowns) cannot be considered. Di and Yu (2021) explore the interior scene design task, where a set of furniture must be arranged in a room by complying with certain criteria. A Deep RL approach is suggested, that is based on a simulation environment. The resolution approach is not independent of the problem size. Furthermore, ground truth data is used to train the agent. For the DFLP, this is hard to implement, because extensive training data is hard to provide in the case of facility layout planning. (Klar et al., 2021).

Mirhoseini and Goldie (2021) present a Deep RL approach to chip-floorplanning. A computer chip is divided into blocks,

each of which is an individual module. Chip floorplanning involves placing netlists onto chip canvases (two-dimensional grids) to optimize performance metrics while adhering to certain constraints. The chip-floor planning problem is modeled as a Markov decision process. States encode information about the partial placement, including the netlist (adjacency matrix), node features (width, height, type), edge features (number of connections), current node (macro) to be placed, and metadata of the netlist graph. Actions are all possible locations onto which the current macro can be placed without violating any constraints. Rewards are zero for all actions except the last action, where the reward is a negative weighted sum of proxy wire length, congestion, and density. To address the challenge of large state and action spaces, a GNN encodes the state space. The GNN is trained via regression to minimize the weighted sum of mean squared loss (negative reward). To incorporate the Edge-GNN into the RL policy network, the prediction layer is removed. Due to the graph representation and dedicated, supervised training of the encoder on various datasets, generalization to problems of different sizes and formulations is possible. Furthermore, placements generated by the pre-trained policy can be generated in subsecond times because it requires only a single forward pass through the pre-trained policy for each macro.

The state of the art shows, that current research approaches from scheduling and layout planning do not meet the requirements for controlling an LMAS. Notably, the following deficits were found:

1. No RL approach considers the integrated problem of FJSP and DFLP
2. Existing approaches for the subproblem FJSP do not consider the necessary flexibility in terms of operational flexibility
3. Existing approaches for the subproblem DFLP do not consider the necessary flexibility in terms of an open-field layout and movable facilities

Approaches from related subject areas offer the potential for application to the control of LMAS. Especially the approach for chip-floorplanning (Mirhoseini et al., 2021) provides a promising solution for large-scale layout problems.

To address this research gap, in this paper we propose the following contributions:

1. A MARL approach to solve the integrated problem of FJSP and DFLP in LMAS, including operational flexibility and an open field layout.
2. A training and simulation environment for LMAS control algorithms.

Multi agent reinforcement learning algorithm for the integrated online scheduling and layout planning in LMAS

The following section presents a MARL Algorithm for the integrated online scheduling and layout planning in LMAS. First, an overview and the general interaction cycle between the environment and the RL agents are described. Afterward, the decoder responsible for the scheduling and layout planning is shown in detail.

Overview and interaction cycle

To enable efficient training and validation of the algorithm, the real LMAS is modeled through a DES (cf. Fig. 1). During training, simulation scenarios are created using automatic scenario generation. Based on the generated scenarios, the simulation is performed. For initialization, a random agent is used. Following the paradigm of online scheduling, the algorithm for integrated scheduling and layout planning is called during the simulation whenever an assembly operation is finished. As a result, placement and scheduling decisions are fed back to the LMAS, respectively the DES (Fig. 3).

The requirement for real-time decision-making during inference favors using Deep RL for processing data. The various input data must be encoded to a fixed-size vector representation that NNs can read to handle varying problem sizes. The NNs work as a decoder in the described setting based on the encoded environment state. Therefore, the control agent consists of an encoder and a decoder. The encoder is implemented using a graph neural network (GNN). As a result, a fixed-size vector representing the current simulation state is passed to the encoder.

Environment description and system modelling

To prove the RL's applicability to the layout planning and scheduling of LMAS, the environment depicted in Fig. 4 was implemented using a DES. Assembly jobs enter the assembly system in a predefined order, considering a restricted number of work in progress (WIP). Whenever a job gets finished (all assembly operations of the job are finished), a new job enters the system. Each job belongs to a specific product type. Several product types are defined a priori. When a new job is created, a product type is assigned based on an equal probability distribution of all possible product types. Each product type demands assembly stations to perform a list of operations until a job is finished and can leave the assembly system. The implementation order of the operations is restricted to a predefined degree and specified through a precedence matrix. Each station can perform a set of assembly operations. During the initialization of the simulation, a station is assigned several operations sampled from a range of possible operations

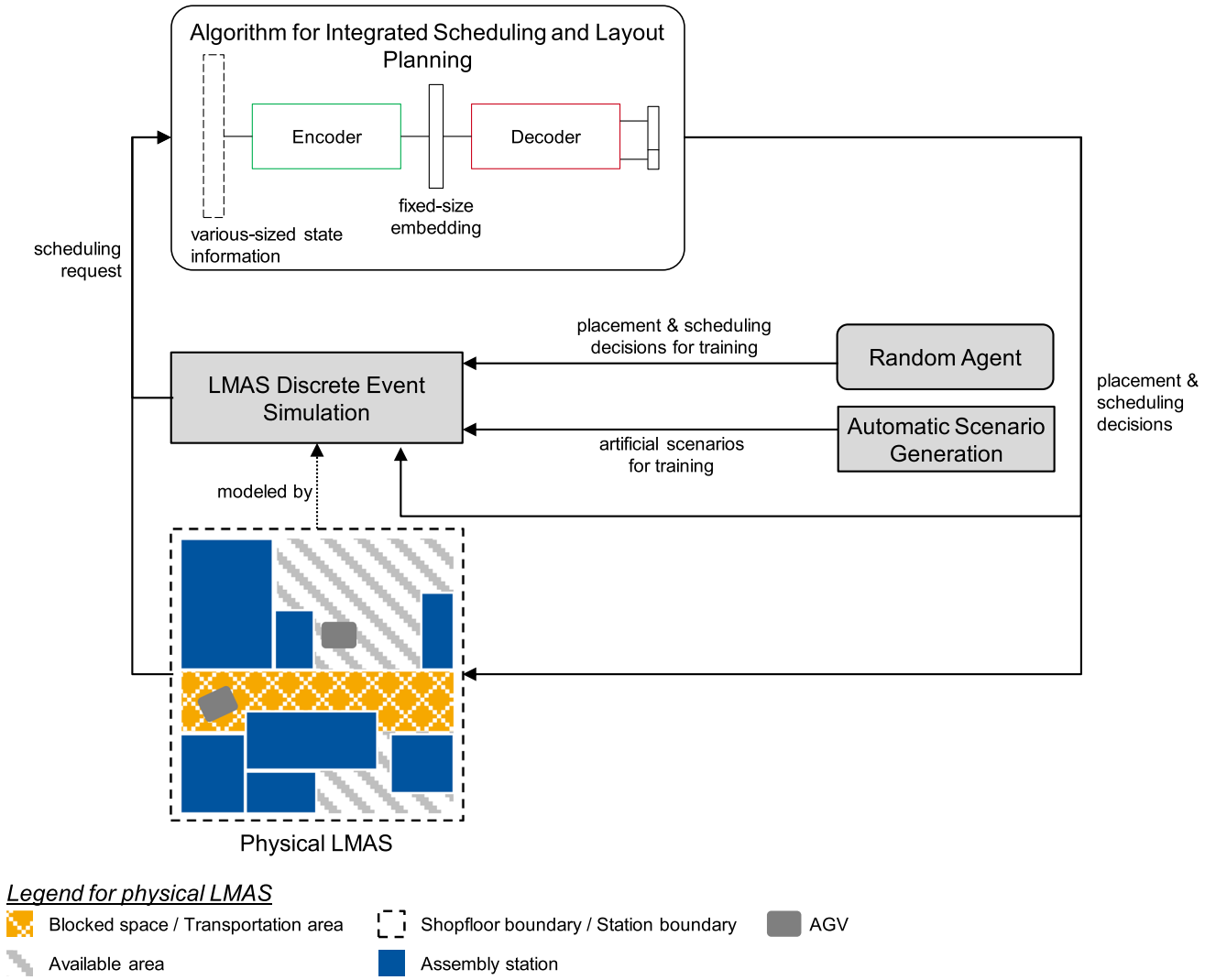

Fig. 3 Overview of elements for the training and application of the control algorithm. The physical LMAS is represented by a Discrete Event Simulation. The algorithm consists of an endcoder and a decoder

with equal probabilities. Thus, multiple stations may offer the same operation (operational flexibility). Jobs can only require operations that are offered by the assembly system. A new scheduling decision is requested each time an operation of a job finishes, and the job needs further operations to be implemented. If a job cannot be assigned to the next station, it blocks the current station until a new assignment can be found. The transportation time between the current and the following station is calculated on the airline distance using a deterministic transportation velocity.

Stations consist of an input buffer, an output buffer, and mobile resources. The mobile resources could be mobile robots but also human operators. Human operators cause uncertainties regarding the processing times of operations. (Lin et al., 2022) Therefore, the DES implements stochastic process times. The maximum length of the station's queue is fixed. Each station has a fixed rectangular size, but the size between different stations varies. The system is modeled as an open-field layout configuration. Thus, all stations can move freely on the shop floor. The orientation of the stations is fixed. Stations are not allowed to overlap. Stations are movable if they are idle, blocked, or in a breakdown. They cannot be moved if they are processing or in transport already. For ease of computing, the shop floor is discretized as a grid that restricts the final position of entities to intersections on the grid. The number of stations in the assembly system is given a priori to the simulation start. Despite that, a station can break down and be out of order for a particular time. The interval between breakdowns is denoted by the mean time between failures for every station. The breakdown duration is denoted by the mean time to repair. When entering a breakdown while being in transport, the transportation time is extended by the breakdown duration. No transportation costs are charged, but stations cannot process jobs while

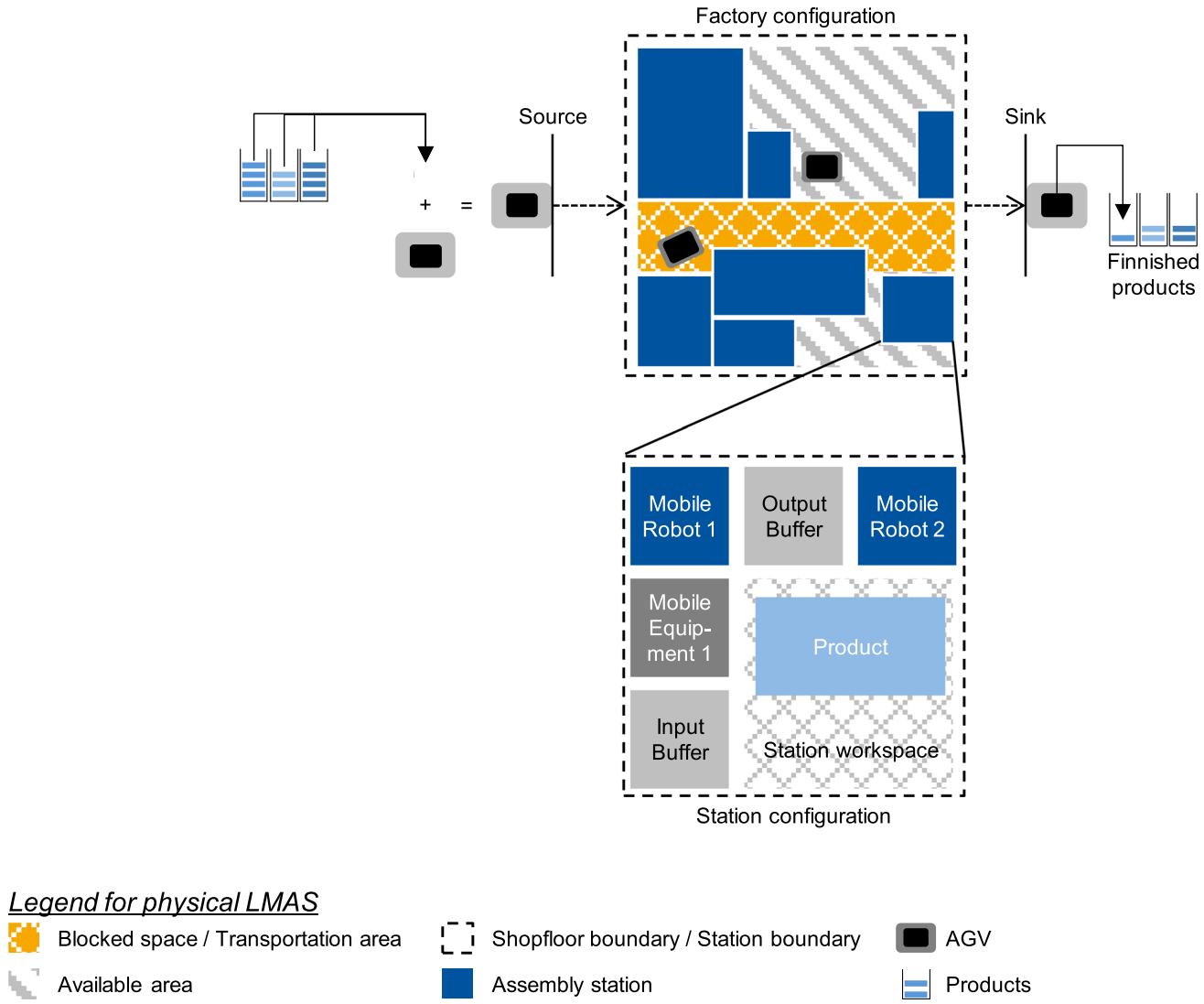

Fig. 4 Factory and station configuration in LMAS with assembly stations of different sizes and shapes. The decision is requested whenever a product enters the system or leaves a station

being in transport. Thus, transportation adds downtime to the system and negatively influences the makespan. The factory layout can be rearranged, whenever a new scheduling decision is requested. For the target position of a station, only coordinates that do not lie on blocked space, e.g., unmovable stations, are considered. A boundary is added to blocked spaces corresponding to the half the size of the station that must get moved to avoid overlapping stations. Hence, it is avoided that objects overlap when the centroid of the station gets moved to the available coordinate. A visualization of the reconfiguration procedure is shown in Fig. 5.

Cooperative, asynchronous, heterogeneous multi agent reinforcement learning algorithm

The task of the decoder is to compute decisions regarding the placement of stations and the scheduling of jobs. In the layout planning task, the control algorithm must determine new coordinates for a station. The action space corresponds to all possible coordinates on the shop floor. A 128×128 grid discretizes the layout. Independence of the action space and the shop floor size is thus given. The action space (for a single action) is 1282 and constant. Depending on the current state, some actions can be marked out (e.g., if space on the shop floor is blocked). The scheduling task is to find a possible station process pairing. Thus, the action space is dependent on the input space. The input space varies because the number of possible pairings varies with the number of stations,

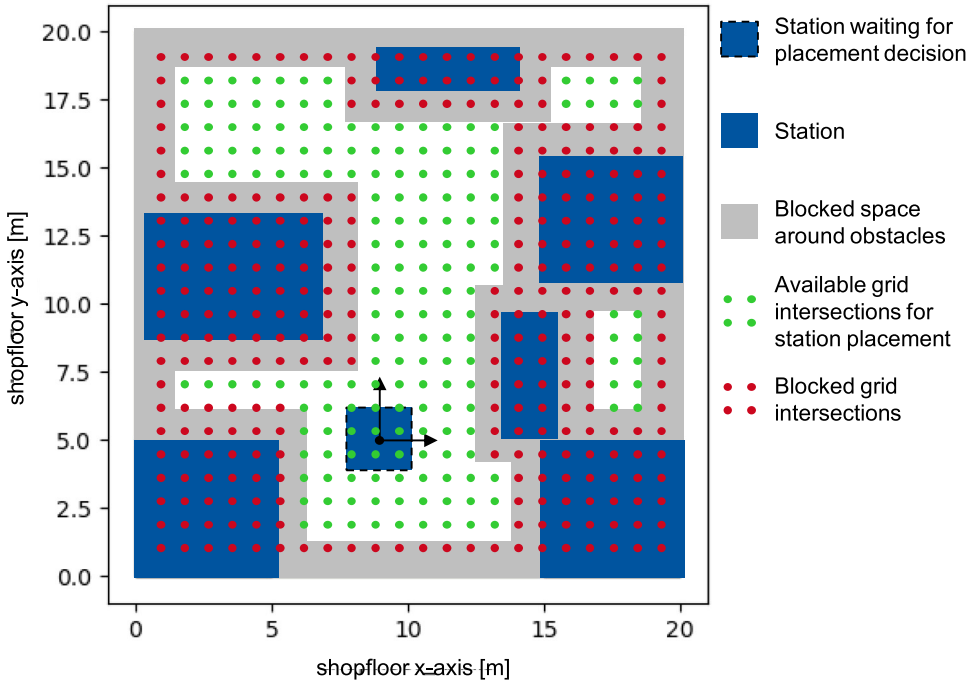

Fig. 5 Visualization of the shop floor during the layout planning process. Green and red dots correspond to grid intersections. The placement of the station with dashed green edges is shown. Light grey boundaries correspond to blocked space resulting from the station's size to move

processes, and process plan flexibility of the corresponding job. Accordingly, the integrated task of scheduling and layout planning consists of two tasks requiring different solution mechanisms.

The agents are called from the DES whenever an assembly operation is finished. When called from the DES, multiple station placements and one scheduling decision are performed. Several layout planning decisions are required because the layout planning decision corresponds to one station placement. An asynchronous MARL approach is implemented to address the challenge of autonomous decisions within a shared environment (assembly system). MARL allows agents to perform actions at different time steps and independent training parameters and policy structures. The shared goal of both agents is to minimize the makespan. The developed control algorithm is thus a *cooperative, asynchronous, heterogeneous* MARL algorithm (cf. Fig. 6).

The underlying problem is formulated as a Decentralized Partially Observable Markov Decision Process with separate rewards. The Decentralized Partially Observable Markov Decision Process generalizes the Partially Observable Markov Decision Process to multiple agents and thus can be used to model a team of cooperative agents situated in a stochastic, partially observable environment. (Oliehoek & Amato, 2016) It is defined by $(S, A, O, R, P, n, \gamma)$. S is the state space. A is the shared action space for each agent. $o_i = (s, i)$ is the local observation for agent i at state s. $(s_{t+1}|s_t, A)$ denotes the transition probability from s_t to s_{t+1} given the joint action A (a_L, a_S), where a_L corresponds to the layout planning decisions and a_S to the scheduling decision. The stochastic state transition is implicitly implemented in the DES (e.g., through random breakdowns of stations). Intermediate state transitions are deterministic and not performed in the DES. $r_{i,+1}(s_t, a_i)$ denote the agent specific reward function. γ is the discount factor (Yu et al., 2021).

The policies of the agents $\pi_{i,i}$ $(a_i \, |o_i)$ are parameterized with θ_i and based on the individual observations by each agent. Based on the individual rewards for each agent, an individual objective function can be derived:

$$J_i(\theta_i, G) = \frac{1}{K} \sum_{g \approx G} E_{g, a \approx \pi_i}[R_{i,a,g}]$$

Here $J_i(\theta_i, G)$ corresponds to the cost function of agent i. G denotes the dataset of simulations of size K. Each individual simulation is written as g. $R_{i,a,g}$ is the episode reward (episode = simulation run). $E_{g,a \approx \pi_i}\left[R_{i,a,g}\right]$ is the expected reward in the simulation of agent i using the collection of actions a sampled from the policy π_i.

Reward design

Information is passed to the decision algorithm whenever the DES requests a decision. The process flow of the simulation with an emphasis on decision-making and rewards is shown in Fig. 7. Steps are distinguished into intermediate and complete steps. Before the scheduling agent makes a scheduling decision, the layout planning agent decides for each movable station on a placement. This action is fed back to the

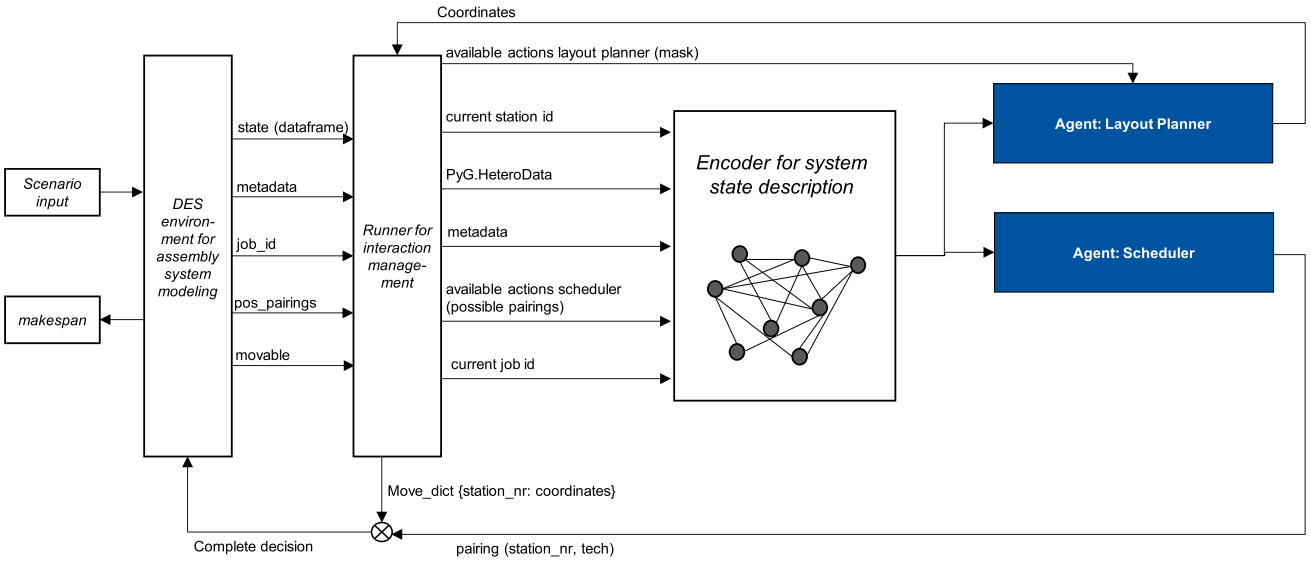

Fig. 6 Process flow of the control algorithm. The DES Runner manages the interaction with a Discrete Event Simulation. Based on the ouput of the encoder, the MARL algorithm returns a decision consisting of coordinates for all stations and a station and technology pairing for the next process step

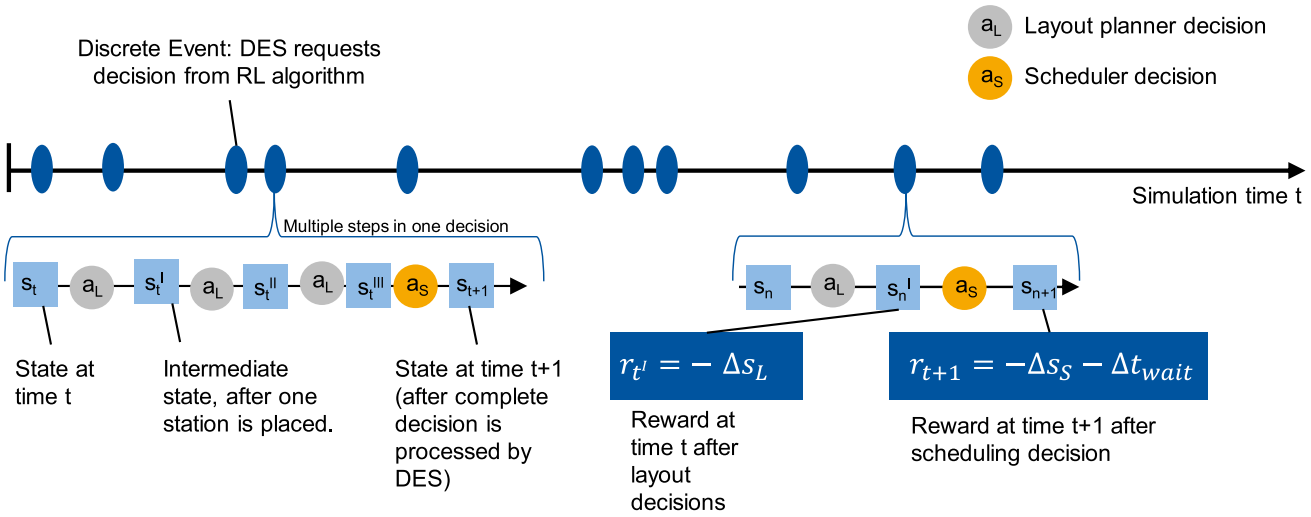

Fig. 7 Process flow with emphasis on decisions and rewards. Whenever a decision is requested, multiple layout decisions are fed back to calculate an intermediate state. The final state is calculated after the scheduling decision is added. It is then processed by the DES

DES. A new intermediate state is calculated, an intermediate reward is generated, and new observations are passed to the agent. This process is iterated until all movable stations are placed. Afterward, the action of the scheduler is, together with the accumulated actions of the layout planner, fed back to the DES. A reward is calculated, and the simulation is proceeded based on the entire scheduling decision. The process is iterated until the simulation is terminal. No station may be movable when the DES requests a decision. In that case, only the scheduler returns an action. This is possible due to the asynchronous MARL setting.

A reward for the layout planner is returned based on the negative distance between the station's initial and destination coordinates. The distance for a scheduled job to reach the scheduled station and the waiting time for the respective job until processing can start is returned as a negative reward.

The scheduler agent reward is the negative distance for the job to reach the scheduled station multiplied by parameter v and the negative waiting time for the respective job until processing can start multiplied by parameter τ.

$$r_{\text{scheduler}} = -v\Delta s_{\text{s}} - \text{ø1t}_{\text{wait}}$$

The layout agent reward is the recent transport distance of the moving station multiplied by η. Additionally, the waiting time of the following job scheduling decision is added retrospectively and multiplied with ϑ.

$$r_{\text{layout}} = -\eta \Delta s_L - \#1\text{t}_{\text{wait}}$$

Weights ν, η, ϑ are subject to change and can be optimized in a hyperparameter-optimization. The return $R_{i,a,g}$ of agent i for all steps (complete steps are treated equally to intermediate steps) in simulation g and chosen actions a with the discount factor γ is given in as:

$$R_{i,a,g} = \sum_{t=0}^{T_g} (\gamma^t * r_{i,t+1})$$

Layout planner

The layout planning agent solves the DFLP. Therefore, it computes the placement for one station per decision step, analogous to (Mirhoseini et al., 2021), placing one chip at a time. Since a quadratic grid discretizes the shop floor, it is straightforward to output a matrix with entries corresponding to grid intersections as the policy. The stochastic policy stores probabilities in the matrix, corresponding to the likelihood of moving the station to the corresponding grid intersection (which corresponds to coordinates on the shop floor).

The input to the policy network of the layout planner is the state vector encoded by the encoder (cf. Figure 8). The state vector is first processed by a fully connected NN with one layer. In order to process the vector to a 2D matrix, the vector is reshaped to a 3D matrix first (e.g., $\mathbb{R}^{512} \rightarrow \mathbb{R}^{4x4x32}$). The reshaping process keeps the number of elements constant. Only the position is changed. Then, deconvolution layers (Zeiler et al., 2010) are used to process the 3D matrix from shape $4 \times 4 \times 32$ to $128 \times 128 \times 1$. The deconvolution layers consecutively reduce the channel dimension and augment the first dimensions. Five deconvolutions are used, as shown in Fig. 8. Between the deconvolution layers, batch normalization and ReLU activation is performed. The output after the deconvolutions is the $128 \times 128 \times 1$ matrix. The policy is then masked using another matrix of the same shape. The mask, which describes the layout planner's available actions, consists of zero values corresponding to blocked space and non-zero values marking available space (cf. Fig. 9). The masked policy is then transformed into a categorical distribution, a discrete probability distribution over the actions. The stochastic policy of the layout planner is denoted by $\pi_{L,L}\ (a_L|o_L)$. In training, an action is sampled from the distribution. During inference, the most likely action is chosen greedily. The state vector forming the input of the layout planner policy network is also passed to the value network. A two-layer fully connected NN approximates the value function.

Scheduling agent

The scheduling agent solves the completely reactive online FJSP. Thus, the agent computes only the next station and process step pairing for each job. The scheduling agent must deal with varying input sizes and an input-dependent action space (number of possible pairings). Based on the popularity of pointer networks or attention mechanisms for combinatorial optimization problems like TSP and VRP, those techniques also seem promising for the FJSP. Moreover, two closely related approaches to pointer networks (Park et al., 2021) and (Zhang et al. 2020) have shown promising results for FJSP. Therefore, a pointer network using only the first output of the decoder is implemented. The inputs to the pointer network are d-dimensional vectors. u_j^i represents the pointer to input no. j. The encoder network produces a sequence of latent memory states e_j. The decoder also produces latent memory states, denoted by d_i. d_1 corresponds to the trainable vector g. The computation of the pointer is parameterized by the attention matrices $W_1, W_2 \in \mathbb{R}^{dxd}$ and an attention vector $v \in \mathbb{R}^d$. The stochastic policy of the scheduler is denoted by $\pi_{S,\theta_S}(a_S|o_S) = A(v, W_1, W_2, e, d_1)$.

$$u_j^i = v^T \tanh\left(W_1 e_j + W_2 d_i\right),\ j \in (1, \ldots, n)$$

$$A(v, W_1, W_2, e, d_i) = \text{softmax}\left(u^i\right)$$

The input vectors of the adapted pointer network are encoded information about possible job station pairings and system metadata. The architecture of the scheduling agent is shown in Fig. 10. The value network is of the same architecture as the layout planner value network. The input to the value network is different. Instead of the station information, the respective job information is passed along with the metadata embedding as input to the network.

The presented decoder consists of two agents in order to minimize the overall assembly makespan while solving a flexible job-shop problem and a dynamic facility layout problem. In order to apply the developed approach to a LMAS, the agents need to be trained using reinforcement learning.

Experiments and results

The developed MARL algorithm is trained using a PPO learning setup based on one base scenario (cf. Table 2). After a

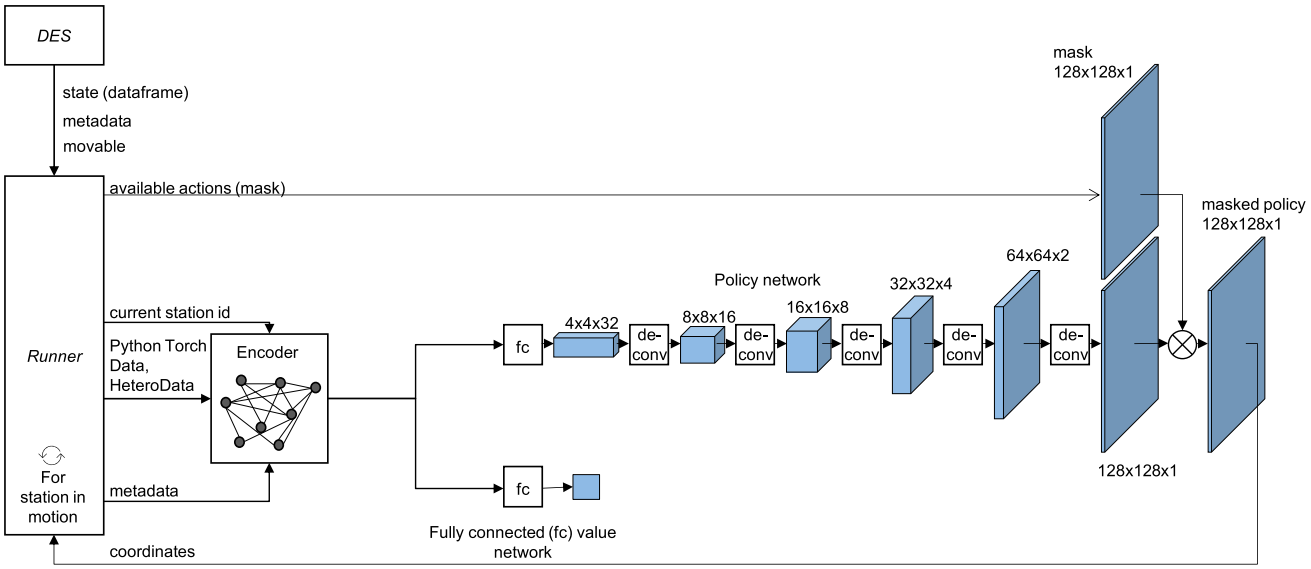

Fig. 8 Layout Planner architecture using policy de-convolutional and fully-connected (fc) value networks

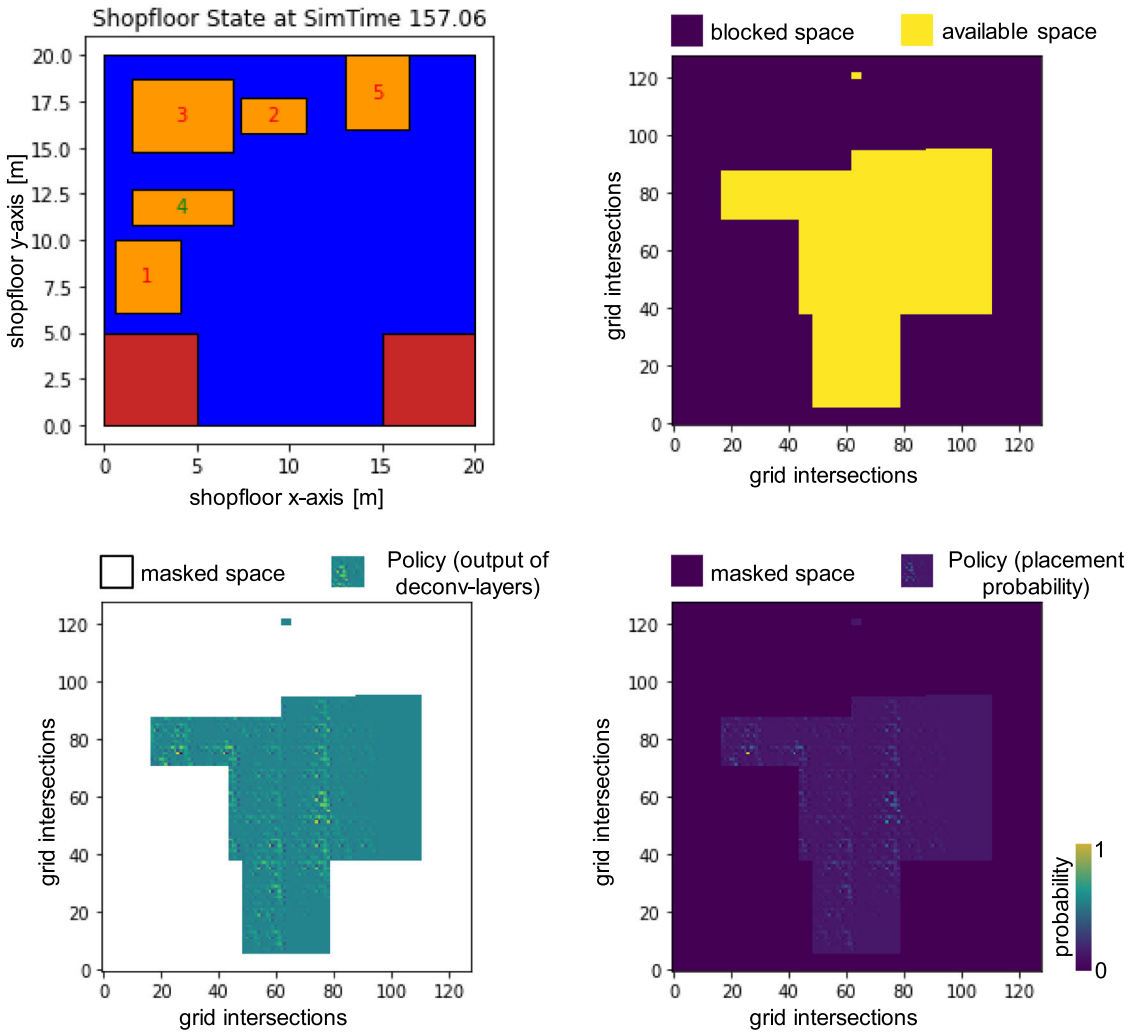

Fig. 9 Visualization of processing steps in the layout planner architecture. Station no.4 gets placed

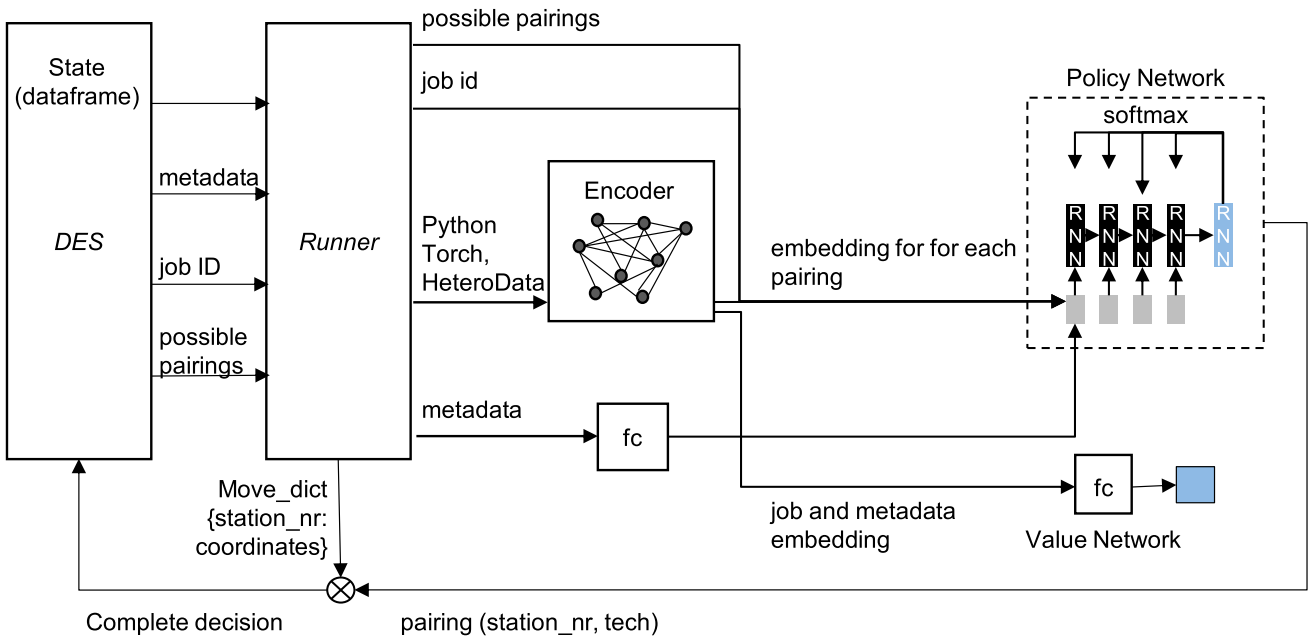

Fig. 10 Policy and value network architecture of the scheduler algorithm

hyperparameter optimization, the performance of the MARL algorithm is compared to a random agent and the untrained MARL. The results indicate good learning behavior. Finally, the generalization ability of the MARL algorithm is shown.

PPO learning set up

Like in single-agent RL settings, many different training approaches can be used in MARL. Based on the success of the layout optimization algorithm by Mirhoseini and Goldie (2021), the PPO method is also promising for the LMAS use case. This is especially true since policy-based approaches, in contrast to value-based approaches, work well on high-dimensional action spaces. A high-dimensional action space is given, especially for the layout planner, using a 128 × 128 grid.

The implemented PPO optimization includes Generalized Advantage Estimation (GAE) (Schulman et al. 2015), advantage normalization, observation normalization, and value normalization. (Schulman et al., 2017) GAE is a variance reduction scheme for policy gradients that adapts the advantage estimation function. With an adapted advantage estimation function, variance is significantly reduced while the introduced bias is kept as low as possible. (Schulman et al. 2015) Value normalization has been empirically proven to positively affect the performance of PPO training in MARL settings. (Yu et al., 2021) The benefit of advantage normalization for PG methods was shown by Gruslys et al. (2017) Moreover, input normalization, which was proven to improve convergence (LeCun et al., 2012), is used before passing data to the NNs. Active masks are used to denote inactive agents. Since an asynchronous MARL approach is used, agents are not active during every step. A mask is used to mark only the active steps of the agent and use only these steps for updating the policy.

Initial experiments on various sized datasets indicated a very high variance regarding the reward function. To reduce the variance, one scenario is used for decoder training and parameter optimization. A low variance is beneficial for the training process, since then changes in the reward function only relate to the agent's behavior. To date, there is no physical system that implements the LMAS. Therefore, discrete event simulation is used with standardized interfaces that could be used in a real-world application. The simulation scenarios are derived from existing industrial use cases and permutated to meet the LMAS requirements. The scenario comprises a 20 m × 20 m layout with five stations and three product types consisting of nine different assembly steps. In training mode, each simulation run starts with random positions of the stations on the shop floor.

The value networks for layout planner and scheduler use a two-layer fully connected NN with hidden layer sizes of 128 and 32. Both layout planner and scheduler are trained concurrently in the MARL setting. One training epoch corresponds to one simulation run of the respective training scenario. During each episode, each agent collects T timesteps of data. With the collected data, the PPO loss function is generated. Adam (Kingma & Ba, 2014), a gradient-based optimization algorithm for stochastic objective functions, is used for optimizing the network weights. Adam is considered the

Table 1 Hyperparameter variations used during training

PPO epochs n_{epochs}	PPO clipping value ϵ	Learning rate lr
1	0.05	1.00E−03
5	0.2	1.00E−04
10	0.5	1.00E−05

preferred optimizer in comparison to the classic stochastic gradient descent, as it has proven to perform better on PPO algorithms. (Schulman et al., 2017) No mini-batches of data are used to perform the weight updates, as large batches of data are preferred in PG methods due to a better gradient estimation. (Ilyas et al. 2018; McCandlish et al. 2018; Berner et al., 2019) Moreover, (Yu et al., 2021) empirically showed high sensitivity of the performance of multi-agent PPO regarding the batch size. MAPPO performed best with maximum batch size. Correspondingly, the whole episode data is used for each PPO update. Weight initialization is proven to significantly affecthe training process. (LeCun et al., 2012) Thus, the weights are initialized using Xavier Uniform Distribution. (Glorot & Bengio, 2010) Xavier Uniform Distribution is also used by (Mirhoseini et al., 2021) and (Yu et al., 2021) and has proven to deliver a good starting point for optimization in their use cases. The maximum episode length is set to 3000 time steps to avoid too large episodes. The training is performed for 3000 episodes.

Hyperparameter optimization

Several hyperparameters can be classified as highly influential on the success of NN algorithms. (Li et al., 2017) This makes hyperparameter optimization a critical part of the algorithm design. The learning rate is one of the most influential parameters on the convergence of the gradient descent in the training of NNs. In several use cases, it is observed that PPO implementation's performance is sensitive to clipping. Additionally, the number of PPO epochs per PPO update is a critical parameter for PPO performance. Therefore, a limited grid search is executed on the three hyperparameters learning rate l_r, PPO clipping value ϵ and PPO epochs n_{epochs}. The three variants of each variable hyperparameter are listed in Table 1.

Every parameter set tested is applied to ten simulation scenarios. Each scenario has the same parameters except for the random seed. The decoder is applied in inference mode. Thus, the action is not sampled from the policy but chosen greedily from the probability distribution over the possible actions. The respective makespan is returned after each simulation run. The results indicate high sensitivity to parameters, with the worst runs reaching a makespan of 3000 s and the best runs reaching a makespan below the 2300 s. The performance comparison (makespan) over all hyperparameter variations indicates three parameter combinations as promising (compare Fig. 11).

MARL learning capabilities for LMAS

To further check the training behavior of the three promising parameter sets, the episode rewards during training are visualized (cf. Fig. 11). The orange line is a linear trendline. The blue line is the 20-point moving average. The trendline for the top row charts shows a decreasing reward for the layout planner as the training continues. The scheduler rewards are rising. The layout planner rewards remain constant in the middle row charts, whereas the layout planner rewards are slightly rising. In the lower row, the trendline for the layout planner reveals an upward trend, whereas the scheduler rewards are falling. Thus, the bottom row charts show a mirroring trendline compared to the top row charts. Considering the overall makespan results, it can be stated that slight improvements in layout planner rewards or scheduler rewards can significantly impact the performance of the whole decoder.

In addition to the observed learning behavior shown in Fig. 11, the behavior of the layout planner is analyzed graphically. An animation of the shop floor after the simulation is depicted in Fig. 12. The left image shows the shop floor in the finished simulation run, which was controlled by the first parameter set from Fig. 11. The rewards of the layout planner indicate a trendline with a negative inclination. The shop floor shows stations that are spread across the shop floor. This is not beneficial since jobs must travel high distances to get to the respective station. The image in the middle shows the shop floor for the parameter set using *nepochs* 5. The layout planner rewards in Fig. 11 showed a constant trendline for the layout planner rewards. Accordingly, in the visualization of the shop floor, the stations are not as far spread as in the left image. The image on the right shows the most compact layout configuration (highest area efficiency). The respective parameter set also indicates the trendline with the highest positive inclination for the layout planner rewards.

Performance assessment of MARL for LMAS

To ensure the validity of the MARL algorithm, the output behavior of the algorithm must comply with the given requirements for the intended application of the algorithm. The intended application of the algorithm is the resolution of the DFLP and FJSP in LMAS. For assessment, two primary requirements are specified: (1) The need for fast decisions for completely reactive online scheduling and (2) the problem size robustness of the algorithm.

The decision time of the MARL algorithm was measured for different scenarios with up to 100 stations, 100 WIP, and

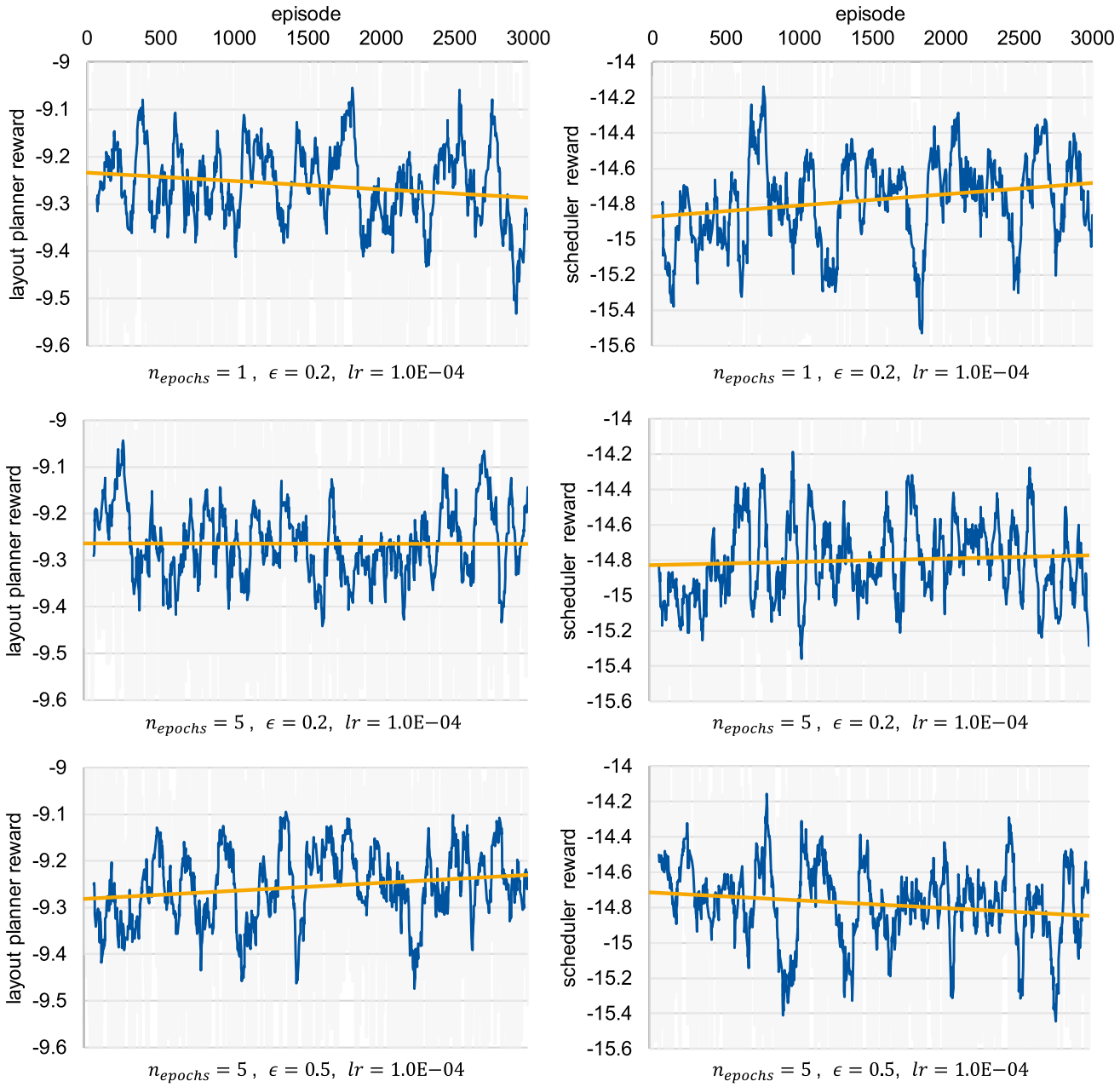

Fig. 11 Performance comparison of the agent rewards with varied hyperparameters. The grey graph denotes the unedited data. The blue graph is the 20-point moving average and the orange line is the trend line. A lower reward is better

100 processes. All tests indicate a decision time under 3.6 s and thus a complete agreement with requirement (1). The decision time is sensitive to several parameters. The decision time is almost linear depending on the number of stations. The WIP also indicates a strong influence on the decision time. The restriction and transport speed parameters indicate no significant impact on the decision time.

The makespan is a scalar to measure the performance of the control algorithm in LMAS. Thus, the makespan of the trained MARL algorithm is compared with a random agent and the untrained MARL algorithm. The simulation was performed ten times for each decision agent using a different random seed for each run. The MARL algorithm shows a far lower makespan for the trained algorithm than for the untrained version (cf. Figure 13). This indicates good learning behavior. The trained algorithm also reaches a better makespan than the random agent.

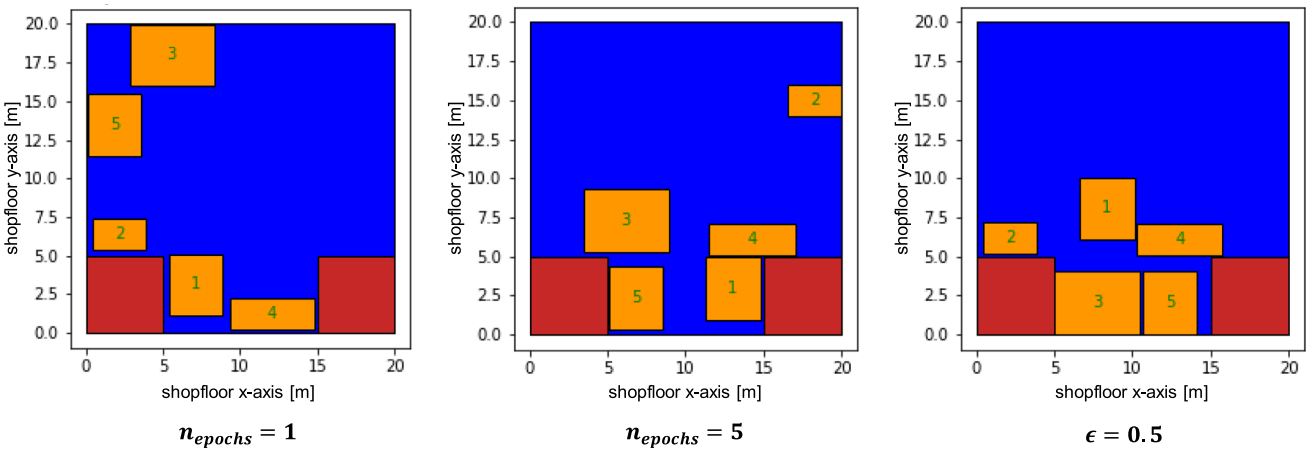

Fig. 12 Comparison of the shop floor layout after the simulation is finished. Each image shows the shop floor for a different parameter setting. The used scenario is the same as for the makespan comparison

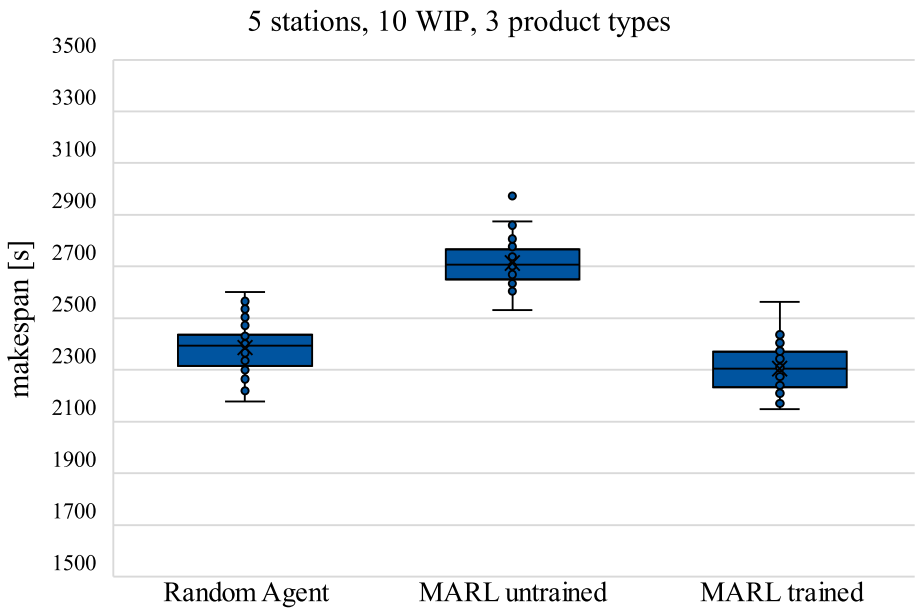

Fig. 13 Makespan comparison using the base scenario. The simulation was repeated ten times with different random seeds for each decision agent

PPO's generalization capabilities for LMAS

To analyze the generalization for the given environment, the parameter sets in Table 2 are varied. Parameter set 1 comprises the default settings for the system and product used in the assessment mentioned above. Parameter set 2–3 differ in the number of stations and the size of the shop floor layout. The rest of the parameters remains the same. Scenario 4–6 consists of assembly systems with varying transport speeds of the AGVs compared to scenario 1. Finally, scenarios 7–9 are systems with a varying number of product types.

The trained MARL and a random agent are applied to the parameter settings to minimize the system's makespan (cf. Fig. 14). It is observable that for scenarios that differ regarding the number of stations (scenarios 2 and 3) from the base scenario, the performance of the MARL algorithm drops significantly. When varying the transport speed in the scenarios (4–6), the MARL algorithm continues to deliver comparable results to the base scenario. This might be the case since the transport speed does not affect the encoder inputs. The transport speed is only encoded in the metadata embedding. Hence, the transport speed is a parameter that does not affect the algorithm's performance since it does not significantly alter the problem representation compared to the base scenario. An agent was trained using multiple scenarios and not just one base scenario to further assess the generalization capabilities of the developed MARL algorithm. This agent was then applied to scenarios 7–9. These scenarios include variations in the number of product types, significantly altering the problem representation. Nonetheless, the newly trained MARL algorithm shows good results

Table 2 Parameter settings for generalization experiments. Set 1 is the base scenario

Set	Layout	WIP	Number of stations	Number of product types	Transport speed	Median random [s]	Median MARL [s]
1	20 m × 20 m	40	5	3	0.5 m/s	2399,61	2378,929,242
2	40 m × 40 m	40	10	3	0.5 m/s	2324,67	2927,408,982
3	80 m × 80 m	40	20	3	0.5 m/s	3298,73	2905,11,758
4	20 m × 20 m	40	5	3	0.2 m/s	2938,9	2630,362,518
5	20 m × 20 m	40	5	3	2 m/s	2213,74	2189,911,056
6	20 m × 20 m	40	5	3	20 m/s	2181,3	2149,163,069
7	20 m × 20 m	40	5	4	0.5 m/s	2083,3	2134,29
8	20 m × 20 m	40	5	5	0.5 m/s	2947,55	2609,8
9	20 m × 20 m	40	5	6	0.5 m/s	2470,21	2345,18

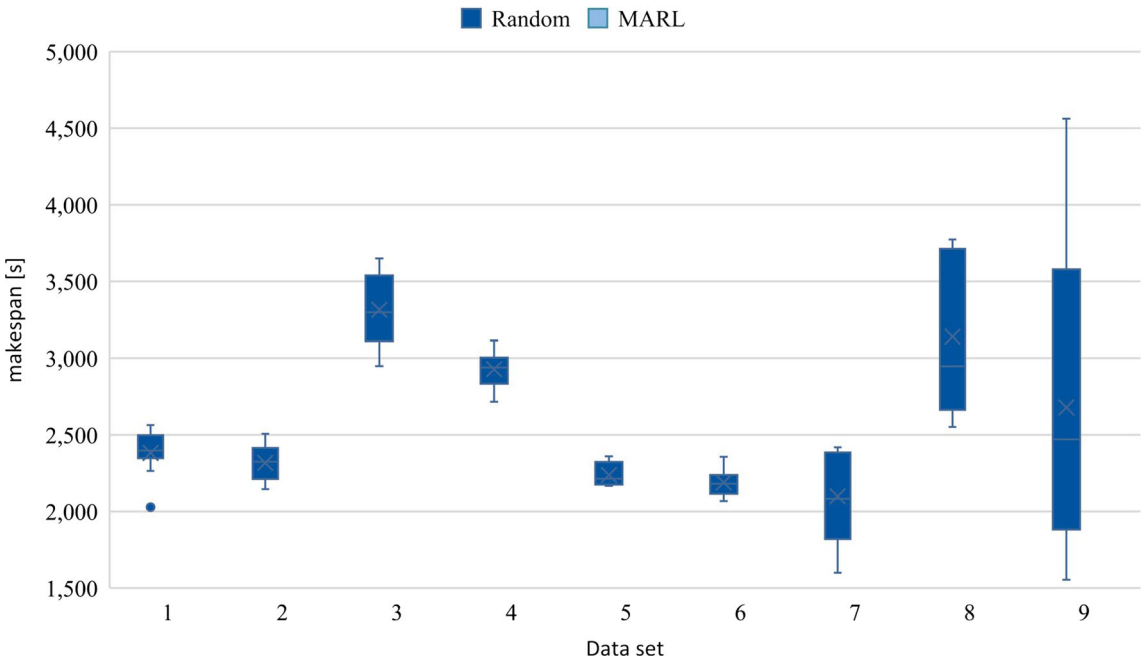

Fig. 14 Boxplot diagram displaying the application of the random agent and trained MARL agent on data sets with varying parameters. The middle line shows the median, the X denotes the average makespan

compared to the random agent, proving its ability of generalization. Lastly, when comparing the median over all scenarios, the MARL algorithm dominated in 78% of the cases.

Conclusion and future work

In this work, a multi-agent reinforcement learning approach for online layout planning and scheduling in line-less mobile assembly systems trained using a discrete event simulation was presented. The main contribution was a LMAS control

algorithm implementing operational flexibility and an open-field layout with movable facilities. The approach was able to adapt to changing conditions and effectively coordinate multiple jobs and stations within an assembly system to achieve the overall goal of minimizing the makespan. During simulation experiments, the developed MARL was proven to outperform a random approach in 78% of the tested scenarios. The results also showed a decrease in performance for scenarios of different sizes. Therefore, a training approach using different scenarios should be explored to improve the generalization capabilities of the algorithms.

Future research should focus on the incorporation of additional objectives, such as energy consumption, reconfiguration costs or costs for the necessary space. Moreover, the system should incorporate additional constraints e.g., real path movement, different resource setup capabilities and intralogistics. Human operators should be included within the simulation, considering walking speed and additional safety requirements regarding the layout. Another direction could be to extend the approach to handle more complex assembly tasks involving multiple pre-assembly steps. Additionally, it would be interesting to explore the use of additional hyper parameter combinations or of other reinforcement learning algorithms in addition to the applied PPO approach. Finally, it would be valuable to test the proposed approach on real-world assembly systems to assess its practicality and generalizability. Therefore, research should focus on embedding the MARL into a digital twin environment providing interfaces to MES, fleet-management software, or a decision support system.

Funding Open Access funding enabled and organized by Projekt DEAL. This work is part of the research project "AIMFREE" that is funded by the German Federal Ministry for Economic Affairs and Energy (BMWi) within the indirective on a joint funding initiative to fund research and development in the field of electromobility (funding number: 01MV19002A) and supported by the project management agency German Aerospace Center (DLR-PT). The authors are responsible for the content.

Data availability Data sets generated during the current study are available from the corresponding author on reasonable request. Restrictions apply to the availability of the used data as they are derived from an industrial use case, which were used under license for the current study, and so are not publicly available.

References

Agrawal, A., Won, S. J., Sharma, T., Deshpande, M., & McComb, C. A (2021). A multi-agent reinforcement learning framework for intelligent manufacturing with autonomous mobile robots. *Proceedings of the Royal Society A* 1, pp. 161–170. https://doi.org/10.1017/pds.2021.17

Berner, C., Brockman, G., Chan, B., Cheung, V., Dębiak, P., Dennison, C., Farhi, D., Fischer, Q., Hashme, S., Hesse, C., Józefowicz, R. et al. (2019). Dota 2 with Large Scale Deep Reinforcement Learning. Available http://arxiv.org/pdf/1912.06680v1.

Brucker, P., & Schlie, R. (1990). Job-shop scheduling with multipurpose machines. *Computing, 45*(4), 369–375. https://doi.org/10.1007/BF02238804

Burggraf, P., Wagner, J., & Heinbach, B. (2021). Bibliometric study on the use of machine learning as resolution technique for facility layout problems. *EEE Access, 9*, 22569–22586. https://doi.org/10.1109/ACCESS.2021.3054563

Busoniu, L., Babuska, R., & de Schutter, B. (2008). A comprehensive survey of multiagent reinforcement learning. *EEE Transactions on Systems Man and Cybernetics Part C, 38*(2), 156–172. https://doi.org/10.1109/TSMCC.2007.913919

Calvo, J., & Dusparic, I. (2019). Heterogeneous multi-agent deep reinforcement learning for traffic lights control.

Calvo, J. A., & Dusparic, I. (2018). Heterogeneous multi-agent deep reinforcement learning for traffic lights control. In: Irish Conference on Artificial Intelligence and Cognitive Science. Retrieved January 25, 2024, from https://api.semanticscholar.org/CorpusID:57661298.

Chu, Y., & You, F. (2014). Integrated scheduling and dynamic optimization by Stackelberg game: bilevel model formulation and efficient solution algorithm. *Industrial & Engineering Chemistry Research, 53*(13), 5564–5581. https://doi.org/10.1021/ie404272t

Claus, C., & Boutilier, C. (1998). The dynamics of reinforcement learning in cooperative multiagent systems. *AAAI/IAAI, 1998*(746–752), 2.

Di, X., & Yu, P. (2021). Deep reinforcement learning for producing furniture layout in indoor scenes. Retrieved January 25, 2024, from http://arxiv.org/pdf/2101.07462v1.

Drira, A., Pierreval, H., & Hajri-Gabouj, S. (2006). facility layout problems: A literature analysis. *IFAC Proceedings Volumes, 39*(3), 389–400. https://doi.org/10.3182/20060517-3-FR-2903.00208

Foerster, J., Farquhar, G., Afouras, T., Nardelli, N., & Whiteson, S. (2017). Counterfactual multi-agent policy gradients. Retrieved January 25, 2024, from http://arxiv.org/pdf/1705.08926v2.

Oliehoek, F. A., & Amato, C. (2016). *A concise introduction to decentralized POMDPs*. Springer.

Gavish, B., & Graves, S. C. (1978). The travelling salesman problem and related problems. In *Operations Research Center Working Paper; OR 078-78*.

Glorot, X., & Bengio, Y. (2010). Understanding the difficulty of training deep feedforward neural networks. In Proceedings of the thirteenth international conference on artificial intelligence and statistics. JMLR Workshop and Conference Proceedings, pp. 249–256.

Goldie, A., & Mirhoseini, A. (2020) Placement optimization with deep reinforcement learning, pp. 3–7. https://doi.org/10.1145/3372780.3378174.

Gronauer, S., & Diepold, K. (2021). Multi-agent deep reinforcement learning: A survey. *Artificial Intelligence Review*. https://doi.org/10.1007/s10462-021-09996-w

Gruslys, A., Dabney, W., Azar, M. G., & Piot, B. (2017). The reactor: A fast and sample-efficient actor-critic agent for reinforcement learning. Retrieved January 25, 2024, from http://arxiv.org/pdf/1704.04651v2.

Halpern, J. Y. (2007). Computer science and game theory: A brief survey. In *arXiv preprint cs/0703148*.

Hameed, M. S., & Schwung, A. (2023). Graph neural networks-based scheduler for production planning problems using reinforcement learning. *Journal of Manufacturing Systems, 69*, 91–102. https://doi.org/10.1016/j.jmsy.2023.06.005

Hosseini-Nasab, H., Fereidouni, S., Fatemi Ghomi, S. M., & Fakhrzad, M. B. (2018). Classification of facility layout problems: a review study. *The International Journal of Advanced Manufacturing Technology., 94*(1–4), 957–977. https://doi.org/10.1007/s00170-017-0895-8

Hsu, C. C., Mendler-Dünner, C., & Hardt, M. (2020). Revisiting design choices in proximal policy optimization. Retrieved January 25, 2024, from http://arxiv.org/pdf/2009.10897v1.

Hüttemann, G., Buckhorst, A. F., & Schmitt, R. H. (2019). Modelling and assessing line-less mobile assembly systems. *Procedia CIRP, 81*, 724–729. https://doi.org/10.1016/j.procir.2019.03.184

Hüttemann, G., Göppert, A., Lettmann, P., & Schmitt, R. H. (2017). Dynamically interconnected assembly systems. *WGP-Jahreskongress Aachen, 7*, 261–268.

Ilyas, A., Engstrom, L., Santurkar, S., Tsipras, D., Janoos, F., Rudolph, L., & Madry, A. (2018). A closer look at deep policy gradients. Retrieved January 25, 2024, from http://arxiv.org/pdf/1811.02553v4.

Jing, X., Yao, X., Liu, M., & Zhou, J. (2022). Multi-agent reinforcement learning based on graph convolutional network for flexible job shop scheduling. *Intelligence in Manufacturing.* https://doi.org/10.1007/s10845-022-02037-5

Johnson, D., Chen, G., & Lu, Y. (2022). Multi-agent reinforcement learning for real-time dynamic production scheduling in a robot assembly cell. *IEEE Robotics and Automation Letters., 7*(3), 7684–7691. https://doi.org/10.1109/LRA.2022.3184795

Kim, B., Jeong, Y., & Shin, J. G. (2020). Spatial arrangement using deep reinforcement learning to minimise rearrangement in ship block stockyards. *International Journal of Production Research, 58*(16), 5062–5076. https://doi.org/10.1080/00207543.2020.1748247

Kingma, D. P., & Ba, J. (2014). Adam: a method for stochastic optimization. Retrieved January 25, 2024, from http://arxiv.org/pdf/1412.6980v9.

Klar, M., Glatt, M., & Aurich, J. C. (2021). An implementation of a reinforcement learning based algorithm for factory layout planning. *Manufacturing Letters, 30*, 1–4. https://doi.org/10.1016/j.mfglet.2021.08.003

Konda, V., & Tsitsiklis, J. (1999). Actor-critic algorithms. In S. Solla, T. Leen, K. Müller (Eds.): Advances in neural information processing systems, vol. 12: MIT Press. Retrieved January 25, 2024, from https://proceedings.neurips.cc/paper_files/paper/1999/file/6449f44a102fde848669bdd9eb6b76fa-Paper.pdf.

Koopmans, T. C., & Beckmann, M. (1957). Assignment problems and the location of economic activities. *Econometrica, 25*(1), 53. https://doi.org/10.2307/1907742

Lecun, Y. A., Bottou, L., Orr, G. B., & Müller, K. R. (2012). Efficient BackProp. In G. Montavon, G. B. Orr, & K.-R. Müller (Eds.), *Neural networks: Tricks of the trade.* Springer.

Li, L., Jamieson, K., DeSalvo, G., Rostamizadeh, A., & Talwalkar, A. (2017). Hyperband: A novel bandit-based approach to hyperparameter optimization. *Journal of Machine Learning Research, 18*(1), 6765–6816.

Lin, C. H., Wang, K. J., Tadesse, A. A., & Woldegiorgis, B. H. (2022). Human-robot collaboration empowered by hidden semi-Markov model for operator behaviour prediction in a smart assembly system. *Journal of Manufacturing Systems, 62*, 317–333.

Littman, M. L. (1994). Markov games as a framework for multi-agent reinforcement learning. In Machine Learning Proceedings 1994: Elsevier, pp. 157–163.

Ma, Q., Ge, S., He, D., Thaker, D., & Drori, I. (2019). Combinatorial optimization by graph pointer networks and hierarchical reinforcement learning. Retrieved January 25, 2024, from http://arxiv.org/pdf/1911.04936v1.

McCandlish, S., Kaplan, J., Amodei, D., & Team, OpenAI Dota (2018). An empirical model of large-batch training. Retrieved January 25, 2024, from http://arxiv.org/pdf/1812.06162v1.

Menda, K., Chen, Y.-C., Grana, J., Bono, J. W., Tracey, B. D., Kochenderfer, M. J., & Wolpert, D. (2019). Deep reinforcement learning for event-driven multi-agent decision IEEE Transactions on Intelligent Transportation Systems. *20*(4), 1259–1268. https://doi.org/10.1109/TITS.2018.2848264

Mirhoseini, A., Goldie, A., Yazgan, M., Jiang, J. W., Songhori, E., Wang, S., Lee, Y. J., Johnson, E., Pathak, O., Nazi, A., Pak, J., et al. (2021). A graph placement methodology for fast chip design. *Nature, 594*(7862), 207–212. https://doi.org/10.1038/s41586-021-03544-w

Moslemipour, G., Lee, T. S., & Rilling, D. (2012). A review of intelligent approaches for designing dynamic and robust layouts in flexible manufacturing systems. *The International Journal of Advanced Manufacturing Technology, 60*, 11–27.

Oren, J., Ross, C., Lefarov, M., Richter, F., Taitler, A., Feldman, Z. et al. (2021). SOLO: search online, learn offline for combinatorial optimization problems. Retrieved January 25, 2024, from http://arxiv.org/pdf/2104.01646v3.

Oroojlooy, A., & Hajinezhad, D. (2021). A review of cooperative multi-agent deep reinforcement learning. Retrieved January 25, 2024, from http://arxiv.org/pdf/1908.03963v4.

Ouelhadj, D., & Petrovic, S. (2009). A survey of dynamic scheduling in manufacturing systems. *Journal Scheduling, 12*(4), 417–431. https://doi.org/10.1007/s10951-008-0090-8

Özgüven, C., Özbakır, L., & Yavuz, Y. (2010). Mathematical models for job-shop scheduling problems with routing and process plan flexibility. *Applied Mathematical Modelling, 34*(6), 1539–1548. https://doi.org/10.1016/j.apm.2009.09.002

Panait, L., & Luke, S. (2005). Cooperative multi-agent learning: The state of the art. *Autonomous Agents and Multi-Agent Systems, 11*(3), 387–434. https://doi.org/10.1007/s10458-005-2631-2

Park, J., Chun, J., Kim, S. H., Kim, Y., & Park, J. (2021). Learning to schedule job-shop problems: Representation and policy learning using graph neural network and reinforcement learning. *International Journal of Production Research., 59*(11), 3360–3377.

Petrazzini, I. G. B., & Antonelo, E. A. (2021). Proximal policy optimization with continuous bounded action space via the beta distribution. Retrieved January 25, 2024, from at http://arxiv.org/pdf/2111.02202v1.

Qin, Z., & Lu, Y. (2021). Self-organizing manufacturing network: A paradigm towards smart manufacturing in mass personalization. *Journal of Manufacturing Systems, 60*, 35–47. https://doi.org/10.1016/j.jmsy.2021.04.016

RiponNawaz, K. S., & Torresen, J. (2014). Integrated job shop scheduling and layout planning: A hybrid evolutionary method for optimizing multiple objectives. *Evolving Systems, 5*(2), 121–132. https://doi.org/10.1007/s12530-013-9092-7

Rosenblatt, M. J. (1986). The dynamics of plant layout. *Management Science, 32*(1), 76–86.

Samsonov, V., Kemmerling, M., Paegert, M., Lütticke, D., Sauermann, F., Gützlaff, A. et al. (2021). Manufacturing control in job shop environments with reinforcement learning, pp. 589–597. https://doi.org/10.5220/0010202405890597.

Schulman, J., Moritz, P., Levine, S., Jordan, M., & Abbeel, P. (2015). High-dimensional continuous control using generalized advantage estimation. Retrieved January 25, 2024, from http://arxiv.org/pdf/1506.02438v6.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., & Klimov, O. (2017). Proximal policy optimization algorithms. Retrieved January 25, 2024, from http://arxiv.org/pdf/1707.06347v2.

Sunehag, P., Lever, G., Gruslys, A., Czarnecki, W., Marian, Z. V., Jaderberg, M. et al. (2017). Value-decomposition networks for cooperative multi-agent learning. Retrieved January 25, 2024, from http://arxiv.org/pdf/1706.05296v1.

Sutskever, I., Vinyals, O., Le, V., & Quoc (2014). Sequence to sequence learning with neural networks. Retrieved January 25, 2024, from http://arxiv.org/pdf/1409.3215v3.

Sutton, R.S., Barto, A. (2018). Reinforcement learning. An introduction. 2nd edn. London: The MIT Press (Adaptive computation and machine learning, no 228).

Unger, H., Börner, F. (2021). Reinforcement Learning for layout planning–modelling the layout problem as MDP, pp. 471–479.

Vinyals, O., Fortunato, M., & Jaitly, N. (2015). Pointer networks. Retrieved January 25, 2024, from http://arxiv.org/pdf/1506.03134v2.

Vithayathil Varghese, N., & Mahmoud, Q. H. (2020). A survey of multi-task deep reinforcement learning. *Electronics, 9*(9), 1363.

Wakilpoor, C., Martin, P. J., Rebhuhn, C., & Vu, A. (2020). Heterogeneous multi-agent reinforcement learning for unknown environment mapping. Retrieved January 25, 2024, from http://arxiv.org/pdf/2010.02663v1.

Wang, Y., He, H., Wen, C., & Tan, X. (2019). Truly proximal policy optimization. Retrieved January 25, 2024, from http://arxiv.org/pdf/1903.07940v2.

Weiss, G. (1999). *Multiagent systems. A modern approach to distributed artificial intelligence*. MIT Press.

Witt, C. S. D., Gupta, T., Makoviichuk, D., Makoviychuk, V., Torr, P. H. S., Sun, M., & Whiteson, S. (2020). Is independent learning all you need in the starcraft multi-agent challenge? Retrieved January 25, 2024, from http://arxiv.org/pdf/2011.09533v1.

Xu, H., Hui, K.-H., Fu, C.-W., & Zhang, H. (2020). TilinGNN: Learning to tile with self-supervised graph neural network. *ACM Transactions on Graphics*. https://doi.org/10.1145/3386569.3392380

Yu, C., Velu, A., Vinitsky, E., Wang, Y., Bayen, A., & Wu, Y. (2021). The surprising effectiveness of PPO in cooperative, multi-agent games. Retrieved January 25, 2024, from http://arxiv.org/pdf/2103.01955v2.

Zeiler, M. D., Krishnan, D., Taylor, G. W., Fergus, R. (2010). Deconvolutional networks, pp. 2528–2535. https://doi.org/10.1109/CVPR.2010.5539957.

Zhang, C., Song, W., Cao, Z., Zhang, J., Tan, P. S., & Xu, C. (2020). Learning to dispatch for job shop scheduling via deep reinforcement learning. Retrieved January 25, 2024, from http://arxiv.org/pdf/2010.12367v1.

Zhang, J., Ding, G., Zou, Y., Qin, S., & Fu, J. (2019). Review of job shop scheduling research and its new perspectives under Industry 4.0. *Journal of Intelligent Manufacturing, 30*(4), 1809–1830. https://doi.org/10.1007/s10845-017-1350-2