Multi-Agent Reinforcement Learning with Graph Neural Networks for Online Multi-Hoist Scheduling

Lars Quaedvlieg
Department of Data Science and Knowledge Engineering
Maastricht University
Maastricht, The Netherlands

***Abstract*—This thesis explores an approach to solving the online multi-hoist scheduling problem by combining graph neural networks and multi-agent reinforcement learning. It approaches the problem by creating two sets of agents: source agents and hoists. When requested, a source agent selects a job from a queue of jobs in a source station to give to hoists. The hoists are responsible for picking up and dropping off jobs at stations, and coordinate with each other to avoid scenarios that would result in a deadlock. The devised algorithms are trained and benchmarked against other approaches, such as random and heuristic algorithms. Further insights into the methods are obtained from an analysis using dimensionality reduction on neural activations of the environment states. The results indicate that deadlocks are avoided in all experimental results. Furthermore, the approach in this thesis outperforms the other approaches in the benchmark by 7.50% to 10%. By analyzing the neural activations, it is shown that the hoist agents estimate that situations with many jobs being processed yield a higher job throughput than situations with less. Further research into the overall approach is recommended, but the results show potential to perform well against other approaches in existing literature.**

***Index Terms*—Online multi-hoist scheduling, graph neural network, reinforcement learning, multi-agent coordination, deadlock avoidance**

I. Introduction

Scheduling is an important problem setting for both academia and industry. Firstly, scheduling problems are transformed to benchmarks for research in NP-hard combinatorial optimization problems [1]. Some examples include the job-shop scheduling problem [2], the production scheduling problem [3], and the automatic guided vehicle problem [4]. Many of these problems are widely applicable to industrial settings, since many assignment problems can be formulated as scheduling problems [5]. Consequently, the algorithms devised from research contribute to optimizing countless industrial processes, due to similarities between specific scheduling problems.

In this thesis, we explore methods for a specific scheduling problem: the *online multi-hoist scheduling* problem (OMHSP); otherwise known as the dynamic multi-hoist scheduling problem [6]. For this problem, objects, which are placed on a rack, are moved to stations by overhead hoists. Within these stations, the parts receive (electro)chemical treatment. The objects follow pre-defined orders to visit stations. After visiting all stations in their order, the process for those objects is finished. The problem is online, since the scheduler receives jobs that arrive over time [7]. The scheduler schedules these jobs without knowledge about the future. In section IV, the problem is more formally introduced.

The field of mathematical optimization is concerned with the selection of a best solution, with regard to some objective, from some set of feasible solutions [8]. Most commonly, scheduling problems such as the multi-hoist scheduling problem (MHSP) are modelled and solved using techniques from mathematical optimization, an example being mixed-integer linear programming for modelling of the MHSP [9].

Common algorithms such as branch-and-bound techniques can solve these mathematical models to optimality [10] [11], but suffer from the curse of dimensionality [12]. The curse of dimensionality often prevents these algorithms from being applicable in real-world scenarios [13]. To devise more practically applicable algorithms, research in approximation algorithms increased. Instead of targeting an optimum solution of the problem, approximation algorithms aim to find an approximation to the optimum solution in at most polynomial time. Some examples of approaches are neighbourhood search [14], heuristics [15], and genetic algorithms [16]; the latter being limited to only single-hoist problems.

The approximation algorithms above mostly start with an initial job queue, but do not adapt well to online situations, as they need to re-solve from scratch every time a new job arrives [17]. The lack of adaptation is problematic for the OMHSP, since the configurations constantly change with new jobs coming in at a predefined rate. By formulating the OMHSP as a sequential decision-making problem, new jobs arriving over time does not impose an issue for the algorithm, since the state representation can be updated at every time step. An example of such a formulation is the Markov Decision Process (MDP).

In this thesis, we formulate the OMHSP as a Markov Decision Process, so it can be approached by using the reinforcement learning framework. This formulation is desirable, since the framework can optimize sequential decision-making problems by maximizing some reward signal. The goal of the research presented in this thesis is to maximize the job throughput for the online multi-hoist scheduling problem.

By utilizing techniques from reinforcement learning and

This thesis was prepared in partial fulfilment of the requirements for the Degree of Bachelor of Science in Data Science and Artificial Intelligence, Maastricht University, and was completed in cooperation with Aucos AG in Aachen, Germany. Supervisors: Dr. Möckel, Prof. Weiss; company supervisor: Florian Wimmenauer

graph representational learning, this problem is approached.

Several research questions have been formulated to contribute to discussing results surrounding the problem statement of this thesis. For the following paragraphs, "the approach" defines the approach that this thesis takes to solve the OMHSP.

Since deadlocks cause the entire process to halt, making the system unable to continue working on tasks, it is important to address the potential deadlocks, as also discussed in [18] [19]. One question to ask is "How can the approach be devised, such that deadlocks are guaranteed to be avoided?".

Due to the optimization context of the problem statement in literature [20] and this thesis, it is key to ask "How does the approach perform against the OMHSP?". Performance will be measured as the job throughput for multiple simulations of the OMHSP. Since it is infeasible for this thesis to assess the performance of the approach directly on the real setup of the problem, a simulation is created as an alternative tool of assessment. The hypothesis is that a higher throughput in the simulation will yield a better performance against the real-world problem.

Considering that multiple approaches to the OMHSP exist, it becomes necessary to benchmark different algorithms in order to compare them in terms of performance, which is measured in job throughput. The question becomes "How does the approach taken in this thesis perform on the OMHSP, as opposed to other methodologies in literature?". However, since there is no fixed benchmark for the OMHSP, it becomes difficult to perform this comparison. For that reason, the approach in this thesis is compared with other methods that are implemented for the purpose of comparison. Two examples are random and greedy heuristic algorithms, and their performance is also measured within the simulation.

Since the approach that this thesis is taking employs several black-box algorithms such as neural networks, it is difficult to explain how the system makes its decisions [21]. As poor decision-making can have significant negative impact on the throughput in the OMHSP, and thus a loss of resources in the industry, another question becomes "What insights can be observed from the approach?". Insights can be derived from information into the decision-making of the approach.

This thesis is organised as follows. Section II explains the contributions from this thesis in terms of current state-of-the-art. Section III describes background knowledge for the methodology. In section IV, the problem is presented. An explanation of the devised methods is given in section V. Experiments on the performance of these methods with respect to the research questions are found in VI. The experimental results are presented in section VII. Finally, the discussions and conclusions are drawn in section VIII and IX.

II. Related work

Previously, multi-agent reinforcement learning has been used with GNNs to approach the OMHSP [22]. However, the problem definition in that paper restricts the situation to a rail system where the overhead hoists can overtake each other. In this situation, deadlocks do not arise, which is a large problem in the one-way rail systems that are investigated in this thesis. Furthermore, regarding the methodology of that paper, the graph representation is distinct from the one used in this thesis.

The previous paper appears to be the only approach to the OMHSP using reinforcement learning and graph neural networks. More popular in literature are branch-and-bound approaches, which can also be used on a setup with a single rail [9]. In order to avoid deadlocks, a set of disjunctive inequalities is constructed. This thesis extends those rules to facilitate more efficient hoist movements, by utilizing a priority list of hoist behaviours.

There exist many papers using mathematical models of the MHSP [23] [24] [25], which all employ heuristics to speed up the search to an optimal solution within the mathematical model. This thesis makes use of a simulation with learning algorithms to prevent spending much time on search. Although this gives up the optimality guarantee, it benefits algorithmic runtime. Moreover, it allows the problem to be online, which mathematical models do not support.

III. Background

Reinforcement learning (RL) is the problem faced by an agent that must learn behavior through trial-and-error interactions with a dynamic environment [26]. Fully observable reinforcement learning environments can be formulated as a Markov Decision Process. Formally, this is a 5-tuple (S, A, R, P, γ) [27], where $R : S \times A \to \mathbb{R}$, $P : S \times A \times S \to [0, 1]$, and $\gamma \in \mathbb{R}$. Here, S and A are the state and action space of the agent, R is the reward function, P is the transition probability function and γ is the discount factor.

An RL agent is situated in a particular state $s \in S$, where it has to take an action $a \in A$. Action selection is done using the policy $\pi(a_t|s_t) \in [0, 1]$, which denotes the probability of taking action a_t given that the agent is in state s_t.

After taking action a in state s, the agent observes reward R^a_s and ends up in state s' with probability $P^a_{s,s'}$. This form of sequential decision-making is repeated until the agent reaches a terminal state, after which the episode ends. There also exist non-episodic problems, where there are no terminal states.

The return $G_t = \sum_{i=t}^{\infty} \gamma^{i-t} R_i$ is a discounted cumulative sum of rewards R_i. γ ensures that rewards observed later count exponentially less in the computation of the return from time step t. State value $v_\pi(s_t) = \mathbb{E}[G_t|S = s_t]$ is the expected return when in state s_t under policy π. $Q_\pi(s_t, a_t) = \mathbb{E}[G_t|S = s_t, A = a_t]$ is the expected return G_t when in state s_t and taking action a_t under policy π [28].

There are multiple approaches to reinforcement learning, but this thesis makes use of Policy Gradient methods [29], which are actor-critic algorithms. These methods use explicit policy- and value function estimator networks to make decisions.

Policy gradient methods compute an estimate of the gradient of the policy function, which is optimized with gradient ascent. One commonly used estimator is

$$L^{PG}(\theta) = \hat{\mathbb{E}}_t[\nabla_\theta \log \pi_\theta(a_t|s_t)\hat{A}_t] \tag{1}$$

The policy is approximated by a function with parameters θ. $\hat{A}_t$ is an estimator of the advantage at time t. Furthermore, $\hat{\mathbb{E}}_t[\dots]$ is an estimate of the expectation, estimated by mini-batch samples of experience.

However, empirically, performing multiple steps of optimization on this loss L^{PG} using the same trajectory often leads to destructively large policy updates [30]. In order to prevent these large policy updates, Trust Region Policy Optimization (TRPO) [31] was introduced. In this method, a constraint on the size of the policy update is imposed.

$$\begin{aligned} & r(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)} \\ & \max_\theta \hat{\mathbb{E}}_t \left[r(\theta)\hat{A}_t \right] \\ & \text{s.t. } \hat{\mathbb{E}}_t \left[KL\left[\pi_{\theta_{old}}(.|s_t), \pi_\theta(.|s_t)\right]\right] \leq \delta \end{aligned} \tag{2}$$

Here, θ_{old} are the parameters of the policy before the update. The constraint of KL-divergence [32] between the two policies prevents too large updates of the policy, since it may at most be $\delta \in \mathbb{R}$.

The problem TRPO aims to solve can also be framed as

$$\max_\theta \hat{\mathbb{E}}_t \left[\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}\hat{A}_t - \beta KL\left[\pi_{\theta_{old}}(.|s_t), \pi_\theta(.|s_t)\right]\right] \tag{3}$$

for some hyperparameter β, but selecting the value has been shown to be difficult [30].

IV. Online Multi-Hoist Scheduling Problem

A. Definition

The multi-hoist scheduling problem is an instance of a group of scheduling problems with a continuous temporal component. There exist multiple types of MHSPs [20], but this thesis limits itself to one definition of the OMHSP. The devised methods are tested on this specific definition, but are extendible to a wider range of problems, for example to online multi-hoist scheduling problems with multiple rails of hoists.

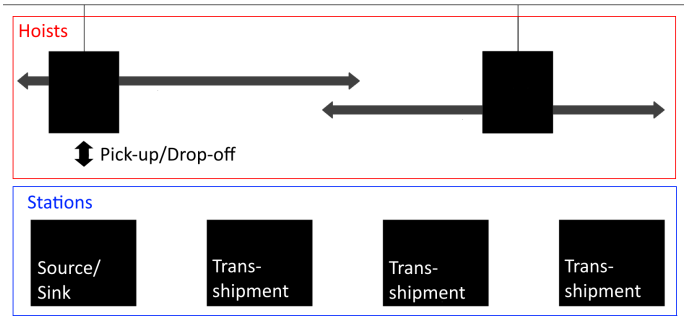

Fig. 1. Example setup of the online multi-hoist scheduling problem

In Figure 1, a setup with two hoists and four stations is depicted. The leftmost station is both a source and sink, where new jobs arrive, and completed jobs are discarded. Each job contains a tuple of tasks $(s_{source}, \cdots, s_{sink})$, indicating in which order the jobs need to go through the stations to be completed.

The goal of the OMHSP is to maximize the throughput of jobs in the line, which is defined as the number of completed jobs per second, with respect to the following conditions:

- Hoists share a single track and must not collide
- Transport time cannot be neglected
- Hoists can only move in their defined range.
- No storage between two stations is allowed.
- Hoists and inner stations can contain up to one job.

The definition above is supported by [20].

B. Simulation

In order to allow agents to learn a policy, a simulation of the problem is created. The simulation progresses with a constant interval Δt at every iteration.

A list of name references of objects, constants, and variables in the simulation are provided in appendix A. The environment is modelled as a stochastic process, where jobs arrive according to a Poisson Process [33] with a constant arrival rate λ^{job}. There is also a possibility to start the simulation with multiple jobs already in the queue, since many industrial applications prefer a backlog of jobs that are ready for processing.

Jobs are initialized with a task configuration $(s_{source}, \cdots, s_{sink})$, that needs to be completed in-order. Once all tasks have been completed, the job is finished by putting it into a sink station. The treatment time of each task is dependent on the station of the task, and when jobs are lifted from a station after being processed, they need to drip for a certain duration.

Hoists move according to a first-order displacement model, where the position $d_{t+1}^{hoist_i} = d_t^{hoist_i} \pm v_{avg}^{hoist_i} \cdot \Delta t$. Here, $v_{avg}^{hoist_i}$ is a constant velocity. When a hoist has reached its target position, a constant $t_{break}^{hoist_i}$ is accumulated to the action time to simulate breaking time. Identical behaviour exists for lifting and lowering time of the hoists. Finally, the simulation detects collisions, and prevents them from occurring by terminating the movement of the colliding hoist(s).

V. Methodologies

This section discusses the methods that have been developed to maximize the job throughput for the OMHSP. Two sets of agents make up the optimization pipeline: one set of agents are the hoists, which learn to control the movement within their ranges to pick-up and drop-off jobs between stations.

The source station can contain multiple jobs at a time. An additional agent is introduced to select the specific job from this station when a hoist requests it. Thus, this agent is responsible for selecting the jobs from the sources, which simplifies the action space of the other set of agents.

A. Job allocation from sources

Whenever a hoist requests a job from a source s, the job allocation agent returns a job for the requesting hoist.

The approach used for the source job allocation is an adapted version of policy learning with representational learning for job-shop problems [34]. The problem is translated to a graph representation, and reinforcement learning is applied to maximize the throughput when selecting jobs from a sink. For OMHSPs, the graph representation of the problem does not correspond to the one used for job-shop problems.

1) Problem formulation: To form the RL problem, the 5-tuple (S, A, R, P, γ) is defined.

The state space is composed of all possible permutations of the environment variables and constants. One permutation then represents information such as the allocation of jobs to stations, how much processing time is left for each job, and numerous other aspects of the simulation variables, constants, and objects. Figure 1 is a good indication of an example state.

The job selector agent uses an approximated representation of the state. For each job j, an embedding vector h_j is constructed, consisting of the following elements:

- A one-hot encoding of the operation status. $[1, 0, 0]$, $[0, 1, 0]$, and $[0, 0, 1]$ mean processing the job has not started, the job is being processed at a station, and the job is finished with processing respectively.
- The remaining processing time of the current task
- The proportion of tasks that have been completed
- The number of tasks remaining, including the current station task
- The processing time of the next task
- The accumulated time of all remaining task times

Each job serves as a node in a graph, with its feature vector being the node embedding. When a new job arrives at the source, the size of the graph increases. Between nodes, three types of edges are created:

- Precedent edges
  Exists from node i to j when the next task of the job corresponding to node i takes place in the station that is currently occupied by the job corresponding to node j.
- Succeeding edges
  Exists from node j to i when the next task of the job corresponding to node i takes place in the station that is currently occupied by the job corresponding to node j.
- Disjunctive edges
  Exists from node i to j and from node j to i when the next tasks of the jobs corresponding to node i and j take place in the same station.

This graph, denoted by $G = (V, H = (h_j, \cdots), E = E_{pre} \cup E_{suc} \cup E_{dis})$ is the approximation of the state space S of the agent. The node embedding in List V-A1 contains summarizing information regarding the future, since the edges in the graph only look at the next task in the tasks order, while there could be more tasks that still have to be completed. By summarizing this information, the agent has information about both the current situation and the future.

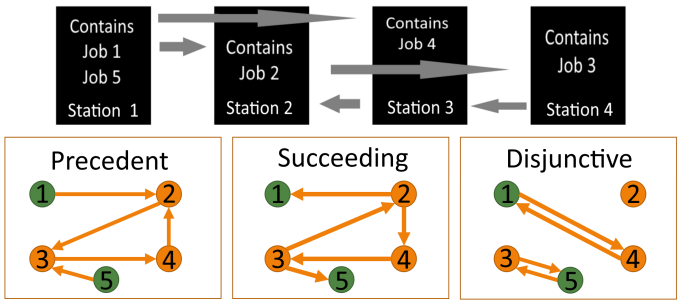

Fig. 2. Example of a graph structure for a problem configuration

The action space A is defined as the set of jobs that are currently inside of source station s, since the agent should choose one of these to dispense to a hoist. The action space changes at every iteration, since new actions might become available due to incoming jobs.

Figure 2 shows a visual representation of the graph that is constructed from a specific setup of jobs and task orders. However, the state representation is still just one graph with three different types of edges. Since job 1 will go to station 2, which is occupied by job 2, an edge is added from job 1 to job 2 in the precedent graph. A reverse edge is added in the succeeding graph. The addition of this edge is done for all preceding and succeeding constraints. Furthermore, since job 1 and job 4 are going to the same station after their current task, an undirected edge is added between both in the disjunctive graph. Each node has its corresponding embedding vector h_j associated to it. The green-coloured nodes are valid actions, since their jobs belong to the source station that the agent is selecting jobs from.

Since the goal is to maximize throughput, a reward signal of 1 is given whenever a job is completed successfully. However, at every iteration where no hoists request a job from s, the action space remains empty. Consequently, these would cause trivial state transitions, which contribute to the difficulty of the credit assignment problem [35]. To counteract the credit assignment problem, transitions are formulated in the form of a semi-MDP [36], which is still solvable with the RL framework.

In a semi-MDP, state transitions are only performed when actions can be selected by the agent. If a state transition occurs at time t and $t + n$, the reward given to the transition in the semi-MDP equals $\sum_{i=t}^{t+n} \gamma^{i-t} r_i$. Other transitions, such as simple progressions of the simulation timer without there being any possible action, are aggregated within one transition of the semi-MDP.

2) Agent structure: Recently, research in graph neural networks (GNNs) has been receiving attention because of the expressiveness of graphs. Due to their convincing performance, GNNs have become a widely applied graph analysis method [37]. Many NP-hard problems are formulated as sequential decision-making problems on graphs, which yields the combination of reinforcement- and graph representational learning. Examples such as routing optimization [38] or computer chip design [39] show promise in these techniques.

In this thesis, we utilize a framework called message-passing graph neural networks (MPNN) [40] to exploit the information from the state graphs $G = G^{(0)}$. These graph neural networks iteratively propagate information through an embedding graph. $h_j^{(k)}$ represents the embedding vector associated to node j. At $k = 0$, $h_j^{(0)}$ represents the initial embedding vector as defined in List V-A1.

$$\begin{aligned} h_j^{(k)} = f_n\Bigg(& ReLU\left(f_{pre}\left(\sum_{i\in N_{pre}(v_j)} h_i^{(k-1)};\theta_0\right)\right) \Big\| \\ & ReLU\left(f_{suc}\left(\sum_{i\in N_{suc}(v_j)} h_i^{(k-1)};\theta_1\right)\right) \Big\| \\ & ReLU\left(f_{dis}\left(\sum_{i\in N_{dis}(v_j)} h_i^{(k-1)};\theta_2\right)\right) \Big\| \\ & ReLU\left(\sum_{i\in V} h_i^{(k-1)}\right) \Big\| h_j^{(k-1)} \Big\| h_j^{(0)};\theta_3\Bigg) \end{aligned} \tag{4}$$

The computational process of this message-passing GNN is shown in Equation (4). The symbol $\|$ represents a vector concatenation operator, and $ReLU(x) = \max(0, x)$. The node embedding of node i at iteration k is the result of a multi-layered perceptron $f_n^{(k)}(.;\theta_0)$, computed from a vector concatenation of multi-layer perceptron outputs $f_{pre}^{(k)}(.;\theta_1), f_{suc}^{(k)}(.;\theta_2), f_{dis}^{(k)}(.;\theta_3)$, and the other feature vectors in the equation.

After k iterations of message passing, the embedding graph $G^{(k)} = (V, H = (h_j^{(k)}, \cdots), E)$ is obtained. At every iteration, the messages only reach the direct neighbours. Hence, multiple iterations are necessary to propagate information through a large part of the graph.

After calculating $G^{(k)}$, the policy and value of the graph embedding are computable.

$$\pi(a_t^i|G^{(k)}) = \frac{\exp(f_p(h_i^{(k)};\theta_4))}{\sum_{j\in A_t}\exp(f_p(h_j^{(k)};\theta_4))} \tag{5}$$

Equation (5) shows the policy, which uses a softmax operation over all allowed actions (A_t) at time step t. For optimization of the policy, PPO [30] is employed. TRPO, as seen in Section III, requires solving a constrained optimization to update the parameters, which is computationally demanding. PPO simplifies this concept by using a clipped surrogate objective. Furthermore, in practice, PPO is more stable than TRPO and algorithms such as DQN [41]. Since PPO uses a critic, the state value is estimated as:

$$V^{\pi}(G^{(k)}) \approx V(G^{(k)};\theta_5) = f_v(\sum_{i\in V} h_i^{(k)};\theta_5) \tag{6}$$

Given parameters set $\Theta = \{\theta_0, \cdots, \theta_5\}$, PPO aims to maximize $L_t^{total}(\Theta)$:

$$\begin{aligned} L_t^{CLP}(\Theta) =& \hat{\mathbb{E}}_t\left[\min(r_t(\Theta)\hat{A}_t, \text{clip}(r_t(\Theta), 1-\epsilon, 1+\epsilon)\hat{A}_t)\right] \\ L_t^{CRIT}(\Theta) =& \hat{\mathbb{E}}_t\left[\left(V(G_t^{(k)};\theta_5) - V_t^{target}\right)^2\right] \\ L_t^{ENT}(\Theta) =& \mathbb{E}\left[\log(\pi_\Theta)\right] \\ L_t^{total}(\Theta) =& \hat{\mathbb{E}}_t\left[L_t^{CLP}(\Theta) - \alpha L_t^{CRIT}(\Theta) + \beta L_t^{ENT}(\Theta)\right] \end{aligned}$$

Here, $min(x, y)$ returns the minimum value between x and y. The function $clip(x, a, b)$ clips the variable x to be within the range $[a, b]$. When $x < a$, $clip(x, a, b) = a$, and when $x > b$, $clip(x, a, b) = b$. The value estimator $V_t^{target} = \sum_{i=t}^{T} R_i$ is the realised sum of rewards. Finally, α and β are hyperparameters controlling the influence of each individual loss function on the total loss.

This loss function combines the actor loss $L_t^{CLP}(\Theta)$, critic loss $L_t^{CRIT}(\Theta)$, and an entropy term $L_t^{ENT}(\Theta)$. The entropy term has been added since the actor and critic share the layers from the GNN.

B. Multi-hoist behaviour learning

The second set of agents are created to optimize hoist movement on the rail. Since hoists cannot overtake each other in the one-rail setup, and have pre-defined ranges, there is a high likelihood for deadlocks to occur. These are situations where the process is not able to progress further.

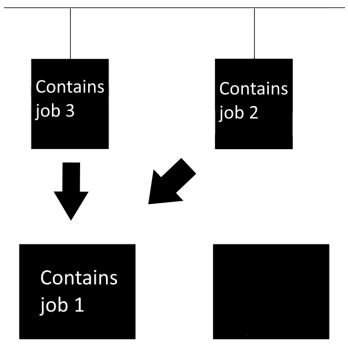

Fig. 3. Example of a deadlock situation

Figure 3 shows an example of a deadlock. Here, both hoists are carrying a job and want to bring it to a loaded station. Since there is no hoist to take job 1 out of the station, the process is stuck. Without multi-agent coordination, these deadlocks can occur from numerous behaviours.

When training a reinforcement learning agent without any guidance, the policy might end up causing deadlock situations. In order to ensure deadlock prevention, learning is done together with pre-defined behaviours and agent coordination, ensuring the decisions of agents cannot result in deadlocks.

When taking an action, a hoist selects a station that currently holds a job. Upon making this decision, a 4-stage process is started, which consists of the following stages:

S1. Going to the selected station
S2. Picking up the job from the selected station

S3. Moving the picked-up job to its next task's station
S4. Dropping off the picked-up job in its next task's station

This way, the agent only learns to select the right stations to maximize throughput, and does not control the movement of the hoist. By controlling these movements, deadlocks that would be caused by hoists running into each other are prevented.

1) Agent coordination: Since agents are placed on a one-rail line, there should be a certain order of movement priorities at which the agents behave. These priorities prevent them from running into a movement deadlock when going towards each other.

The priority queue below has been devised for the purpose of avoiding movement deadlocks.

Action stage (higher position in table is higher priority)
S2 and S4
S3, given that the targeted station is empty. Tie-breakers are resolved by choosing the hoist closest to the target station
S1. Tie-breakers are resolved by selecting the job with the least remaining time
(Resting)

Since an agent cannot stop when loading or unloading a job, these have highest priority. Then, since carrying a job for longer is worse than leaving a job in for treatment, S3 is given priority over stage S2, given that it can be unloaded in its target station at that time. Finally, resting agents have the lowest priority.

When agents collide, the priority queue decides on the behaviour of the collided agents. In this scenario, all agents participating in the collision will follow the behaviour of the highest-priority agent. For example, when the highest-priority agent is in S2, the other agents will wait for this agent to finish.

This procedure prevents deadlocks that would occur from hoists running into each other, but the process can still run into deadlocks such as the one shown in Figure 3. These are caused by permitting hoists to take actions that will result in getting stuck. The agents should coordinate their action selection to avoid these. Action pruning is used to prevent these situations from happening. Initially, the action space A of each hoist consists of all existing stations. Then, the following pruning rules are applied to this set:

- $A = \emptyset$ for non-resting hoists
- $A \leftarrow A \setminus \{a | a$ is an empty station$\}$
- $A \leftarrow A \setminus \{a | a$ is an out-of-range station$\}$
- $A \leftarrow A \setminus \{a | a$ contains a job to an out-of-range station$\}$
- $A \leftarrow A \setminus \{a | a$ is targeted by another hoist$\}$
- $A \leftarrow A \setminus \{a | a$ contains a job to a station being targeted by another hoist$\}$

Besides this initial pruning, the action lists of other agents are pruned again once another agent has made the decision to take an arbitrary action a_h. This pruning procedure consists of the following rules:

- $A \leftarrow A \setminus \{a_h\}$
- $A \leftarrow A \setminus \{a | a$ will result in a deadlock with a_h being taken$\}$

Whether taking an action will result in a deadlock is determined by the size of a maximum-size matching. Figure 4 shows two possible situations where matching is required. Here, the next station is occupied for each job that would be selected. If any of these station are selected, the matching algorithm will attempt matching other hoists to the other jobs in the chain. This matching is done by finding a maximum matching with the same size as the number of stations in the chain. The Hopcroft–Karp algorithm [42] is used to find this bi-partite matching, as it runs in polynomial time. This will naturally prevent these deadlocks from occurring, as actions are only taken if it is completely certain to be possible. If no matching is possible, the action cannot be taken.

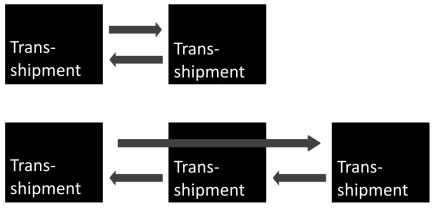

Fig. 4. Examples of station chains that require forced actions

Together with the prevention of deadlocks due to hoist movement, the deadlock prevention mechanism described in this subsection has been proved to be sufficient and necessary conditions for deadlock prevention [9].

2) Agent policy learning: The approach of learning to select which station to target is similar to the approach used for job allocation from sources, which is discussed in Section V-A. As with the job allocation problem, PPO is employed together with a GNN in the exact same way to form policy π for each agent. The difference between the methods comes down to the graph structure used to represent the problem.

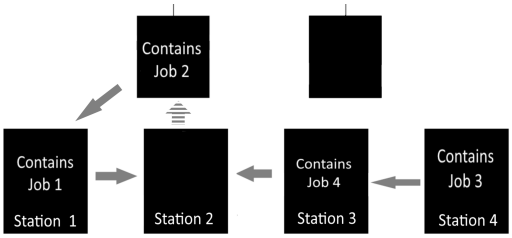

Fig. 5. Example of problem configuration for station selection

The dashed line in Figure 5 represents that the carried job by the left hoist used to be located in station 2. From the example configuration in the figure, the graph representation for the right hoist is created below.

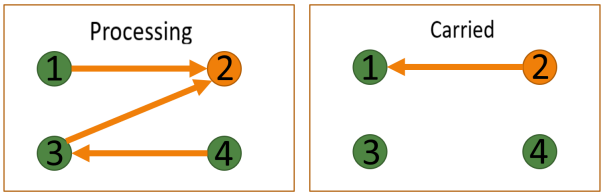

Fig. 6. Graph representation of example problem

There are two graphs: processing and carried graphs. This way, the GNN is able to separate jobs that are currently being carried by a hoist, and jobs that are in a station. The node feature vectors h_j are identical to the ones in List V-A1. The green nodes are valid actions after pruning, and station 2 cannot be selected as it is currently empty. The complete node embedding is shown in the equation below.

$$h_j^{(k)} = f_n\Bigg(ReLU\left(f_{proc}\left(\sum_{i\in N_{proc}(v_j)} h_i^{(k-1)};\theta_0\right)\right)\Big\| ReLU\left(f_{car}\left(\sum_{i\in N_{car}(v_j)} h_i^{(k-1)};\theta_1\right)\right)\Big\| ReLU\left(\sum_{i\in V} h_i^{(k-1)}\right)\Big\| h_j^{(k-1)}\Big\| h_j^{(0)};\theta_3\Bigg) \tag{7}$$

Optimization of the policy and value function estimates happens identically to section V-A, with a positive reward of 1 being awarded whenever a job is completed.

VI. Experiment Setup

The entire project is implemented in the Python programming language with a graphical user interface written using the Pygame library. All experiments are performed on a Windows 10 device with an Intel Core i7-10750H CPU and an NVIDIA Quadro P620 GPU. There was no use of any GPU acceleration.

Firstly, deadlock avoidance is investigated by running the coordination algorithm with a heuristic hoist agent and GNN job selector agent on 45 unique configurations for a time of $7,200$ seconds within each simulation. These configurations vary from containing two treatment stations and two hoists, to ten treatment stations with ten hoists. The order of tasks that jobs needs to complete are at most 100 random permutations of the set of stations.

Whether a configuration ends up in a deadlock is measured by the time difference between the last time step that any agent had the opportunity to take an action, and the final time of the simulation. When this number is large, one expects the configuration to be in a deadlock, while small values indicate that agents are still able to take actions, and are not in a deadlock scenario. Both the source job allocation agents and the hoist agents employ the Adam optimizer [44] to optimize their respective objective functions. The exact configuration files and raw data from the experiments can be found back in appendix A.

Secondly, the GNN hoist agent is trained on 200 simulations of $1,200$ seconds on a configuration of two hoists and five stations, with ten possible tasks orders, and its progression and training performance are analysed using plots of the loss functions and accumulated rewards. All specific hyperparameters used for the training of the algorithms, and the environment configurations, can be found back in appendix A.

In addition, 2-dimensional t-SNE [43] dimensionality reduction is performed on $100,000$ produced graph embedding $\sum_{i\in V} h_i^{(k)}$ of the final model. Paired together with the corresponding state-value estimates $V(G^{(k)};\theta_5)$, applying t-SNE aims to contribute to the explainability of the graph embeddings that are produced by the network. The scatterplot created by the two components of the t-SNE embedding will have $100,000$ data points with their associated state value estimates. If there are any patterns with similar colours in the plot, it means that t-SNE detected that those clusters of points are similar in some way. Then associating those data points to their original graph $G^{(0)}$ could reveal some properties about the clusters.

Finally, the methods presented in this thesis are compared to several other approaches, such as random and heuristic algorithms, in terms of their accumulated reward over different configurations, which are simulated for $3,600$ seconds each. Boxplots, histograms, and appropriate statistical tests are used to determine whether a significance difference in performance is present. Both heuristic agents chooses their target based on greedily selecting job with the lowest remaining processing time. The random agent chooses a random job to distribute in the case of the job selectors. The random hoist agent selects a random station to target.

VII. Results

A. Deadlock avoidance

Summary statistics	Time difference (seconds)
Count	45
Mean	0.695556
Standard deviation	1.908945
Minimum value	0.0
First quantile	0.0
Median	0.0
Third quantile	0.0
Maximum Value	10.1

Fig. 7. Summary statistics of the time differences of the deadlock experiment configuration files

The table above shows that for the 45 configurations, the largest difference in time between the end of the simulation and last possibility for any agent to take an action is 10.1 seconds. Since each simulation is executed for $7,200$ seconds, 10.1 seconds is about 0.14% of the entire simulation time. For all other utilized configuration, this proportion is lower.

B. Training progression of the hoist agent

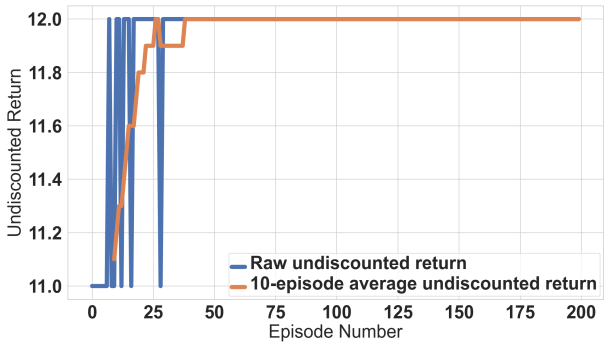

Fig. 8. Undiscounted return during hoist agent training

The figure above visualizes the progression of the cumulative reward over the training period of the hoist agent. At the start of training, the return fluctuates between 11 and 12. The agent consistently obtains a return of 12 after 29 episodes.

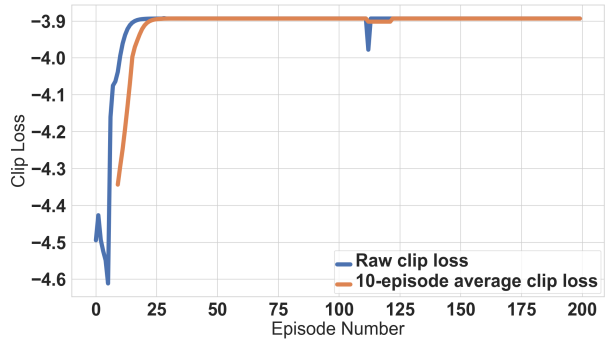

Fig. 9. Clip loss during hoist agent training

The clip loss, which should be maximized, starts out with a value of -4.49. After 113 episodes, the clip loss consistently obtains a value of around -3.893.

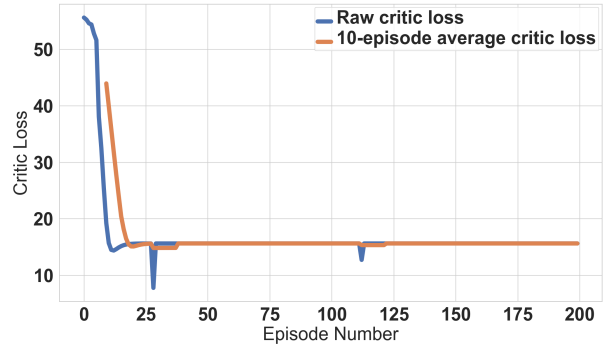

Fig. 10. Critic loss during hoist agent training

The loss function of the critic starts off with a value of 55.6. After 8 episodes, it has decreased to 25.5. After 29 episodes, the loss consistently obtains a value of around 15.64, with an outlier at episode 112 obtaining 12.7.

C. Performance benchmark

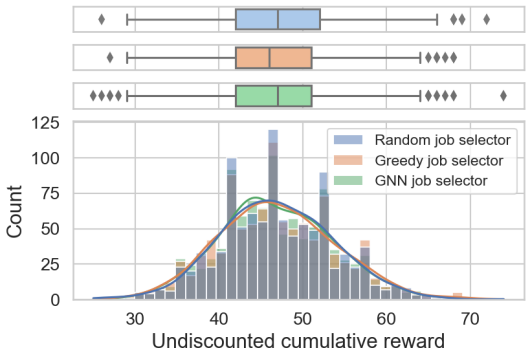

Fig. 11. Boxplots and histograms of the number of completed jobs on OMHSP configurations for different job selector algorithms

In Figure 11, three approaches for the job selector agent are compared. The boxplots, which are overlaid above the histograms, summarize the distribution of these three agents. The histogram shows the actual distribution of the return. From the boxplots, it can be seen that the three distributions look approximately normally distributed.

In order to compare the methods, a one-sided paired Wilcoxon signed-rank test [45] is performed between all pairs of agents. This is a non-parametric test, meaning it does not assume normality of distributions. Let X, Y, Z be random variables sampled from the distributions of the GNN, greedy, and random job selector agent respectively, and let the significance level to $\alpha = 0.01$.

- $H_0 : median(X - Y) = 0, H_a : median(X - Y) > 0$
  After performing the test, we obtain $\rho = 0.66$ with test statistic $220,110..0$. Since $\rho > \alpha$, we cannot reject H_0.
- $H_0 : median(X - Z) = 0, H_a : median(X - Y) > 0$
  After performing the test, we obtain $\rho = 0.53$ with test statistic $227,059.5$. Since $\rho > \alpha$, we cannot reject H_0.
- $H_0 : median(Y - Z) = 0, H_a : median(X - Y) > 0$
  After performing the test, we obtain $\rho = 0.61$ with test statistic $228,737.5$. Since $\rho > \alpha$, we cannot reject H_0.

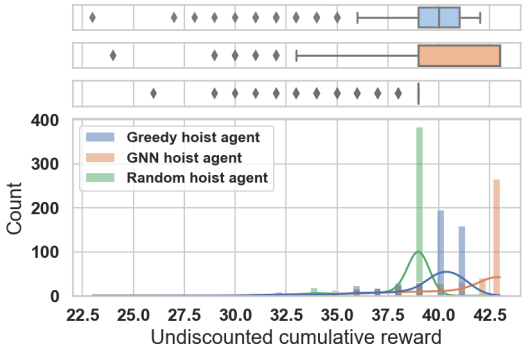

Fig. 12. Boxplots and histograms of the number of completed jobs on OMHSP configurations for different hoist algorithms

In Figure 12, three approaches for hoist agents are compared. The histograms and boxplots show the distribution of

the return. From the histograms, it can be seen that not all the three distributions are normally distributed.

Thus, the methods are again compared using the one-sided paired Wilcoxon signed-rank test. Let X, Y, Z be random variables sampled from the distributions of the GNN, greedy, and random hoist agent respectively, and let the significance level to $\alpha = 0.01$.

- $H_0 : median(X - Y) = 0, H_a : median(X - Y) > 0$ After performing the test, we obtain $\rho = 2.0 \cdot 10^{-18}$ with test statistic $81,722.5$. Since $\rho < \alpha$, we reject H_0.
- $H_0 : median(X - Z) = 0, H_a : median(X - Y) > 0$ After performing the test, we obtain $\rho = 4.8 \cdot 10^{-21}$ with test statistic $84,165.5$. Since $\rho < \alpha$, we reject H_0.
- $H_0 : median(Y - Z) = 0, H_a : median(X - Y) > 0$ After performing the test, we obtain $\rho = 1.6 \cdot 10^{-39}$ with test statistic $94,840.0$. Since $\rho < \alpha$, we reject H_0.

It is known that $median(X) = 43$, $median(Y) = 40$, and $median(Z) = 39$.

D. Explainability of the hoist agent

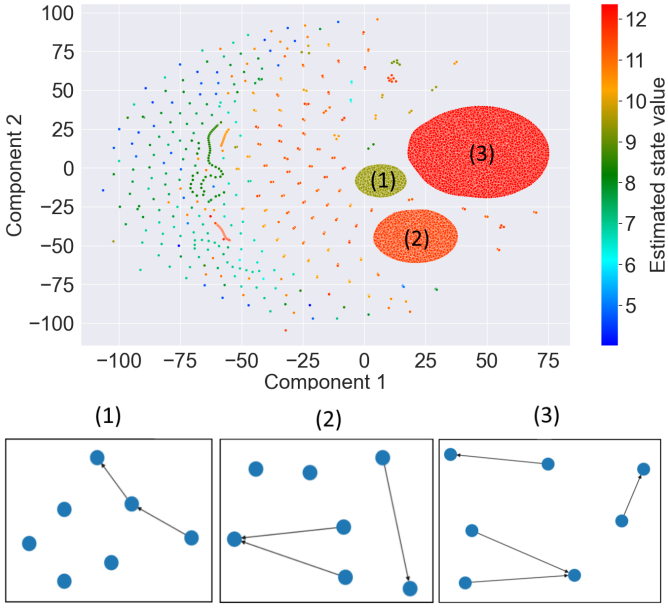

Fig. 13. 2-dimensional t-SNE embedding of graph embeddings $G^{(k)}$ paired with estimated state values

Figure 13 shows a scatterplot of the 2-dimensional t-SNE embedding of $100,000$ graph embeddings $G^{(k)}$. Each data point is paired with a value from a colour map, representing the estimated state value that the network produced for that $G^{(k)}$.

From the figure, it becomes clear that there are three prominent clusters of points in the plot. The state value estimates of these clusters are around 8, 11, and 12 respectively.

Looking at the initial graphs $G^{(0)}$ for the network, as shown below the scatterplot in te figure, the green cluster has one edge between two vertices. In the orange cluster, there is a larger cover of the vertices. Finally, in the red cluster, all vertices in the graph are covered by at least one edge between them.

VIII. Discussion

From the experimental results on the deadlock avoidance experiments, it becomes clear that none of the utilized configurations result in a deadlock. The maximum time difference is 10.1 seconds, which is only 0.14% in the context of a simulation of $7,200$ seconds. This proportion of time indicates that the agents are still taking actions near the end of the simulation, and have not stopped doing so due to deadlock. Hence, making any deadlock in these configurations very unlikely.

It is not possible to say if these results mean that deadlocks are guaranteed to be avoided. However, the configurations used in the experiment covered a large space of possible setups, making it more favourable that the method indeed guarantees to avoid deadlocks. For further research, the use of techniques from software and systems verification could provide a mathematical proof of these arguments.

Training the hoist agent uncovered interesting insights into the problem and the agent. Firstly, Figure 8 shows that the reward landscape of the problem is quite sparse. Even with random initialization, the agent's performance already alternated between scores of 11 and 12. After training for longer, the agent seems to converge to a consistent performance of 12. As visible in the loss plots, the agent is unable to optimize more after obtaining consistent returns of 12.

It is unclear whether 12 is the largest possible throughput, or the agent has gotten stuck in a local optimum. However, since some randomness would be expected in the later episodes, the first possibility seems more likely. It might also be possible that there are better solutions possible, but that by constraining the agents to make pre-defined movements, the variability of the agents gets constrained too much, leaving little room for improvement. However, a consistent improvement of 1 job per $1,200$ seconds still yields a rate of improvement of around 8.3%.

From the hypothesis testing done on the different job selector agents, no significance evidence was found against the hypothesis that the GNN job selector agent does not yield a median improvement over the other approaches. Since this is also true for the random job selector, the importance of selecting between jobs in a source station might not be significant in most general problem configurations. Yet, it cannot be ruled out that in some scenarios where certain task orders follow specific patterns, it is not useful to differentiate between jobs. Further analysis into specific situations might provide insights into this statement.

From the hypothesis testing done on the different hoist agents, significance evidence was found against the hypothesis that the GNN hoist agent does not yield a median improvement over the other approaches. Hence, there is clear evidence that the GNN hoist agent performs better on the testing configurations than the random and greedy heuristic approaches. These findings that the method proposed in this thesis generally performs better than the other two approaches. For the GNN hoist agent versus greedy hoist agent, the improvement is

7.50% per hour, and is a 10.3% increment for the GNN hoist agent versus the random hoist agent. However, there could still be certain configurations where the GNN hoist agent does not perform better; the hypothesis test was performed against the median performance.

The t-SNE embedding for the hoist agent gives an indication that the agent estimates the value of a state based on the number of jobs that are currently actively in the system, and how occupied the stations and hoists are. Figure 13 seems to reveal that more occupied situations have a higher estimated state value, and thus a higher potential for job throughput. Even so, this high occupancy may not actually hold for every situation in the cluster. It is merely an estimate of the network's beliefs, and those may not reflect the actual situation correctly. Furthermore, t-SNE is a stochastic dimensionality reduction algorithm that may produce different results every time. More model explainability techniques in future research could prove insightful to the methodologies presented in this thesis.

In terms of state-of-the-art research about the problem, the results from this thesis are an example that it is possible to utilize learning algorithms to approach the OMHSP. It is difficult to say where this work is placed within the literature in terms of performance, due to the lack of standardized benchmarks. However, its runtime complexity puts it ahead of other papers, such as the ones employing approaches from mathematical optimization.

There are certain decisions that were made in this thesis, that could be improved in future research. For example, both the hoist agents and job selector agents are forced to take an action whenever possible. This makes it impossible for certain situations to occur, such as one hoist waiting for the other to complete. As previously mentioned, constraining the variability of choices an algorithm can make, can limit its potential.

There are many possible directions of future research. One interesting direction is hierarchical reinforcement learning [46], where default behaviours can be used in a hierarchical manner to control hoist movement and coordination between agents.

While GNNs generalize well to new situations [39], one is expected to re-train the agent on a different configuration of the problem, since it is a completely different markov decision process. The field of meta reinforcement learning [47] is concerned with developing agents that learn how to learn in new situations where conditions are (slightly) tweaked, which could be applicable to the problem presented in this thesis.

This problem statement also looked into maximizing throughput, but another interesting problem setting could include job fairness, where no job should be neglected or stay in the system indefinitely. One possible approach to this problem includes a penalty for job queueing times in the reward function. Finally, it could be worthwhile researching algorithms that find policies that are guaranteed to avoid deadlocks, but constrain the problem less than the methods used in this thesis.

IX. Conclusions

This thesis has presented a novel approach to the online multi-hoist scheduling problem by combining graph neural networks with multi-agent reinforcement learning. In the introduction of the thesis, four research questions were presented, which are answered.

Due to the significant impact of deadlocks, the question "How can the approach be devised, such that deadlocks are guaranteed to be avoided?" was created. By using multi-agent coordination, deadlocks resulting from hoist behaviour, such as running into each other, are avoided. Furthermore, deadlocks stemming from hoist action selection are avoided by pruning the action space by removing stations that could result in a deadlock. The experimental results show there are no deadlocks in many of the configuration. However, it cannot be proved that this is the case for any configuration. Hence, it cannot be guaranteed solely by the experiments, but it does not occur for many practical configurations.

After training, the performance of the approach on the test configuration of the OMHSP yields a throughput of 12 jobs per 20 minutes; an 8.3% increment over the same approach before training. The performance varies between configurations, but the agent scores anywhere between 23 to 43 jobs for all configurations within the benchmark. The question "How does the approach perform against the OMHSP?" remains difficult to answer, but if the hypothesis regarding the simulation and real-world are correct, the results would be comparable.

To find out how the performance compares to other approaches, the agent was benchmarked against two other agents. The approach presented in this thesis scored an 7.5% to 10% improvement in throughput upon these agents, which was supported by statistical tests. The question "How does the approach taken in this thesis perform on the OMHSP, as opposed to other methodologies in literature?" is unable to be answered confidently due to there not being a standardized benchmark for the problem configurations. However, the experimental results show potential for the method to be investigated in more detail.

Finally, the question "What insights can be observed from the approach?" can be answered by using the experimental results from the explainability analysis. Using the t-SNE embedding, it was shown that the agent estimates that states where there are many jobs being treated have the potential for a high reward yield. Hence, it is more likely that the agent will attempt filling all stations in order to get a better estimate of the state value.

In conclusion, further research is necessary to determine the performance of the approach in this thesis against other algorithms in the literature. However, from the insights into the algorithm and performances against different configurations and other kinds of agents, the approach taken in this thesis has shown potential to perform well against approaches in literature. Furthermore, the approach has shown potential for usage within an industrial setting, especially due to the deadlock avoidance.

REFERENCES

[1] J. D. Ullman, "Np-complete scheduling problems," *Journal of Computer and System sciences*, vol. 10, no. 3, pp. 384–393, 1975.

[2] A. S. Manne, "On the job-shop scheduling problem," *Operations research*, vol. 8, no. 2, pp. 219–223, 1960.

[3] S. C. Graves, "A review of production scheduling," *Operations research*, vol. 29, no. 4, pp. 646–675, 1981.

[4] M. Gen and L. Lin, "Multiobjective evolutionary algorithm for manufacturing scheduling problems: State-of-the-art survey," *Journal of Intelligent Manufacturing*, vol. 25, no. 5, pp. 849–866, 2014.

[5] J. Kallrath, "Planning and scheduling in the process industry," *OR spectrum*, vol. 24, no. 3, pp. 219–250, 2002.

[6] M.-A. Manier and C. Bloch, "A classification for hoist scheduling problems," *International Journal of Flexible Manufacturing Systems*, vol. 15, no. 1, pp. 37–55, 2003.

[7] K. Pruhs, J. Sgall, and E. Torng, "Online scheduling.," 2004.

[8] J. A. Snyman, D. N. Wilke, *et al.*, *Practical mathematical optimization*. Springer, 2005.

[9] A. Che and C. Chu, "Single-track multi-hoist scheduling problem: a collision-free resolution based on a branch-and-bound approach," *International Journal of Production Research*, vol. 42, no. 12, pp. 2435–2456, 2004.

[10] E. L. Lawler and D. E. Wood, "Branch-and-bound methods: A survey," *Operations research*, vol. 14, no. 4, pp. 699–719, 1966.

[11] Z.-Q. Luo and W. Yu, "An introduction to convex optimization for communications and signal processing," *IEEE Journal on selected areas in communications*, vol. 24, no. 8, pp. 1426–1438, 2006.

[12] T. L. Morin and R. E. Marsten, "Branch-and-bound strategies for dynamic programming," *Operations Research*, vol. 24, no. 4, pp. 611–627, 1976.

[13] M. Caserta and S. Voß, "Metaheuristics: intelligent problem solving," in *Matheuristics*, pp. 1–38, Springer, 2009.

[14] E. Laajili, S. Lamrous, M.-A. Manier, and J.-M. Nicod, "An adapted variable neighborhood search based algorithm for the cyclic multi-hoist design and scheduling problem," *Computers & Industrial Engineering*, vol. 157, p. 107225, 2021.

[15] T. Sun, K. Lai, K. Lam, and K. So, "A study of heuristics for bidirectional multi-hoist production scheduling systems," *International Journal of Production Economics*, vol. 33, no. 1-3, pp. 207–214, 1994.

[16] J.-M. Lim, "A genetic algorithm for a single hoist scheduling in the printed-circuit-board electroplating line," *Computers & industrial engineering*, vol. 33, no. 3-4, pp. 789–792, 1997.

[17] L. A. Hall, A. S. Schulz, D. B. Shmoys, and J. Wein, "Scheduling to minimize average completion time: Off-line and on-line approximation algorithms," *Mathematics of operations research*, vol. 22, no. 3, pp. 513–544, 1997.

[18] T.-E. Lee, H.-Y. Lee, and S.-J. Lee, "Scheduling a wet station for wafer cleaning with multiple job flows and multiple wafer-handling robots," *International Journal of Production Research*, vol. 45, no. 3, pp. 487–507, 2007.

[19] E. Chové, P. Castagna, and R. Abbou, "Hoist scheduling problem: Coupling reactive and predictive approaches," *IFAC Proceedings Volumes*, vol. 42, no. 4, pp. 2077–2082, 2009.

[20] C. Bloch, A. Bachelu, C. Varnier, and P. Baptiste, "Hoist scheduling problem: state-of-the-art," *IFAC Proceedings Volumes*, vol. 30, no. 14, pp. 127–133, 1997.

[21] J. E. Dayhoff and J. M. DeLeo, "Artificial neural networks: opening the black box," *Cancer: Interdisciplinary International Journal of the American Cancer Society*, vol. 91, no. S8, pp. 1615–1635, 2001.

[22] K. Ahn and J. Park, "Cooperative zone-based rebalancing of idle overhead hoist transportations using multi-agent reinforcement learning with graph representation learning," *IISE Transactions*, vol. 53, no. 10, pp. 1140–1156, 2021.

[23] Y. Yih, "An algorithm for hoist scheduling problems," *The International Journal of Production Research*, vol. 32, no. 3, pp. 501–516, 1994.

[24] J. Lamothe, M. Correge, and J. Delmas, "A dynamic heuristic for the real time hoist scheduling problem," in *Proceedings 1995 INRIA/IEEE Symposium on Emerging Technologies and Factory Automation. ETFA'95*, vol. 2, pp. 161–168, IEEE, 1995.

[25] A. El Amraoui and M. Elhafsi, "An efficient new heuristic for the hoist scheduling problem," *Computers & Operations Research*, vol. 67, pp. 184–192, 2016.

[26] L. P. Kaelbling, M. L. Littman, and A. W. Moore, "Reinforcement learning: A survey," *Journal of artificial intelligence research*, vol. 4, pp. 237–285, 1996.

[27] M. L. Puterman, *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons, 2014.

[28] C. J. Watkins and P. Dayan, "Q-learning," *Machine learning*, vol. 8, no. 3, pp. 279–292, 1992.

[29] R. S. Sutton, D. McAllester, S. Singh, and Y. Mansour, "Policy gradient methods for reinforcement learning with function approximation," *Advances in neural information processing systems*, vol. 12, 1999.

[30] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

[31] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz, "Trust region policy optimization," in *International conference on machine learning*, pp. 1889–1897, PMLR, 2015.

[32] S. Kullback and R. A. Leibler, "On information and sufficiency," *The annals of mathematical statistics*, vol. 22, no. 1, pp. 79–86, 1951.

[33] J. F. C. Kingman, *Poisson processes*, vol. 3. Clarendon Press, 1992.

[34] J. Park, J. Chun, S. H. Kim, Y. Kim, and J. Park, "Learning to schedule job-shop problems: representation and policy learning using graph neural network and reinforcement learning," *International Journal of Production Research*, vol. 59, no. 11, pp. 3360–3377, 2021.

[35] R. S. Sutton, *Temporal credit assignment in reinforcement learning*. University of Massachusetts Amherst, 1984.

[36] R. S. Sutton, D. Precup, and S. Singh, "Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning," *Artificial intelligence*, vol. 112, no. 1-2, pp. 181–211, 1999.

[37] J. Zhou, G. Cui, S. Hu, Z. Zhang, C. Yang, Z. Liu, L. Wang, C. Li, and M. Sun, "Graph neural networks: A review of methods and applications," *AI Open*, vol. 1, pp. 57–81, 2020.

[38] P. Almasan, J. Suárez-Varela, A. Badia-Sampera, K. Rusek, P. Barlet-Ros, and A. Cabellos-Aparicio, "Deep reinforcement learning meets graph neural networks: Exploring a routing optimization use case," *arXiv preprint arXiv:1910.07421*, 2019.

[39] A. Mirhoseini, A. Goldie, M. Yazgan, J. W. Jiang, E. Songhori, S. Wang, Y.-J. Lee, E. Johnson, O. Pathak, A. Nazi, *et al.*, "A graph placement methodology for fast chip design," *Nature*, vol. 594, no. 7862, pp. 207–212, 2021.

[40] J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl, "Neural message passing for quantum chemistry," in *International conference on machine learning*, pp. 1263–1272, PMLR, 2017.

[41] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, *et al.*, "Human-level control through deep reinforcement learning," *nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[42] J. E. Hopcroft and R. M. Karp, "An n^5/2 algorithm for maximum matchings in bipartite graphs," *SIAM Journal on computing*, vol. 2, no. 4, pp. 225–231, 1973.

[43] L. Van der Maaten and G. Hinton, "Visualizing data using t-sne.," *Journal of machine learning research*, vol. 9, no. 11, 2008.

[44] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[45] R. F. Woolson, "Wilcoxon signed-rank test," *Wiley encyclopedia of clinical trials*, pp. 1–3, 2007.

[46] A. G. Barto and S. Mahadevan, "Recent advances in hierarchical reinforcement learning," *Discrete event dynamic systems*, vol. 13, no. 1, pp. 41–77, 2003.

[47] J. X. Wang, Z. Kurth-Nelson, D. Tirumala, H. Soyer, J. Z. Leibo, R. Munos, C. Blundell, D. Kumaran, and M. Botvinick, "Learning to reinforcement learn," *arXiv preprint arXiv:1611.05763*, 2016.

APPENDIX

SIMULATION OBJECTS, CONSTANTS AND VARIABLES

Objects

Name	Description
h_i	Hoist i, $h_i \in H$
j_i	Job i, $j_i \in J$
s_i	Station i, $s_i \in S$
$task_i$	Task i, $task_i \in T$

Constants

Name	Unit	Description
t_0^{sim}	$[s]$	Start time of the simulation (bootstraps when > 0)
Δt^{sim}	$[s]$	Interval per simulation time-step
d_{max}^{sim}	$[m]$	Maximum position in simulation
λ^{job}	$[s^{-1}]$	Job arrival rate, the number of jobs per second
p^{tasks_i}		Probability of job getting task order configuration i
$tc_i = (task_m, \cdots, task_n)$		Task order configuration i
$t_{rnd\_min}^{task_i}$	$[s]$	Minimum random time added to task i
$t_{rnd\_max}^{task_i}$	$[s]$	Maximum random time added to task i
$t_{rnd\_min}^{drip}$	$[s]$	Minimum drip time added to task i
$t_{rnd\_max}^{drip}$	$[s]$	Maximum drip time added to task i
$d^{station_i}$	$[m]$	Position of station i
$type^{station_i}$		Type of station, in $\{source, inner, sink\}$
$t_{drip}^{station_i}$	$[s]$	Drip time after job in station i
$t_{task}^{station_i}$	$[s]$	Task duration of job in station i
$d_{min}^{hoist_i}$	$[m]$	Minimum position of hoist i
$d_{max}^{hoist_i}$	$[m]$	Maximum position of hoist i
$d_{initial}^{hoist_i}$	$[m]$	Initial position of hoist i
w^{hoist_i}	$[m]$	Width of hoist i
$v_{avg}^{hoist_i}$	$[ms^{-1}]$	Average velocity of hoist i
$t_{break}^{hoist_i}$	$[s]$	Break time of hoist i
$t_{lift}^{hoist_i}$	$[s]$	Lifting time of hoist i
$t_{lower}^{hoist_i}$	$[s]$	Lowering time of hoist i

Variables

Name	Unit	Description
$task_{cur}^{job_i}$		Current task of job i
$t_{cur}^{job_i}$	$[s]$	Current time left for current task of job i
$task_t^{j,job_i}$	$[s]$	Process duration of task j for job i
$task_{drip}^{j,job_i}$	$[s]$	Dripping duration of task j for job i
$d_{cur}^{hoist_i}$	$[m]$	Current position of hoist i
$t_{action}^{hoist_i}$	$[s]$	Time left for current action of hoist i
$G = (V, E)$		Graph connecting hoists and stations to jobs

TRAINING HYPERPARAMETERS

GNN Hoist Agent and GNN Job Selector Agent

Hyperparameter name	Value
$learning\_rate$	0.0025
$discount\_factor$	1.0
gae_value	0.95
$clip\_value$	0.2
$critic\_coef$	0.5
$entropy\_coef$	0.01
$batch\_size$	200
num_epochs	5
$hid\_channels$	32
$num\_propagations$	3

The configurations that were used for training have been stored in JSON-format and attached to the resource files in a folder called ”res/experiments”. Please refer to this file for more information regarding the configurations that are used. All configuration files are equipped with a ”seed” variable, which ensures reproducibility of results.

DEADLOCK EXPERIMENT CONFIGURATIONS

The configurations that were used for measuring deadlocks have been stored in JSON-format and attached to the resource files in a folder called ”res/experiments”. Please refer to this file for more information regarding the configurations that are used. All configuration files are equipped with a ”seed” variable, which ensures reproducibility of results.