Deep Reinforcement Learning for Two-sided Online Bipartite Matching in Collaborative Order Picking

Luca Begnardi L.BEGNARDI@TUE.NL
Hendrik Baier H.J.S.BAIER@TUE.NL
Willem van Jaarsveld W.L.V.JAARSVELD@TUE.NL
Yingqian Zhang YQZHANG@TUE.NL
Eindhoven University of Technology, The Netherlands

Editors: Berrin Yanıkoğlu and Wray Buntine

Abstract

As a growing number of warehouse operators are moving from human-only to Collaborative human-robot Order Picking solutions, more efficient picker routing policies are needed, since the complexity of coordinating multiple actors in the system increases significantly. The objective of these policies is to match human pickers and robot carriers to fulfill picking tasks, optimizing pick-rate and total tardiness of the orders. In this paper, we propose to formulate the order picking routing problem as a more general combinatorial optimization problem known as Two-sided Online Bipartite Matching. We present an end-to-end Deep Reinforcement Learning approach to optimize a combination of pick-rate and order tardiness, and to deal with the uncertainty of real-world warehouse environments. To extract and exploit spatial information from the environment, we devise three different Graph Neural Network architectures and empirically evaluate them on several scenarios of growing complexity in a simulation environment we developed. We show that all proposed methods significantly outperform greedy and more sophisticated heuristics, as well as non-GNN-based DRL approaches. Moreover, our methods exhibit good transferability properties, even when scaling up test problem instances to more than forty times the size of the ones the models were trained on. Code is available at: `https://github.com/ai-for-decision-making-tue/DRL-TOBM-CPR`.

Keywords: Collaborative order picking; online combinatorial optimization; online bipartite matching; deep reinforcement learning; graph neural networks.

1. Introduction

Order picking is a crucial component of warehouse operations and it is estimated to account for up to 65% of total operating costs (Ho et al., 2008). Thanks to recent developments of robotics and self-driving vehicles, more and more warehouse operators are shifting from the traditional picker-to-part order picking system (OPS), where humans handle all of the work, to a collaborative human-robot order picking system (CHR-OPS). In this new system, humans work along a fleet of automated mobile robots (AMR), each of which is assigned a set of items to be collected before returning to depot. The humans' task is then to pick items from shelves and place them on the robots, which handle transportation across the warehouse. This collaboration potentially allows to considerably reduce the inefficiency of traditional OPS due to the human pickers not having to walk long distances between pick locations and depot anymore. However, the possible performance increase in CHR-OPS depends on several optimization problems, including order batching, batch sequencing, and picker routing (Srinivas and Yu, 2022). While the first two have been extensively investigated in the literature with both classical and learning-based methods (Henn, 2015; Cals et al., 2021; Beeks et al., 2022), fewer studies focus exclusively on picker routing. The objective of the picker routing problem is to decide which retrieving tasks should be assigned to each human picker in order to optimize a certain metric, such as total order tardiness. However, most of the existing approaches for CHR-OPS are not specific for the picker routing problem, but try to address it together with the batching and sequencing strategy. These approaches mostly adopt handcrafted heuristics (Azadeh et al., 2020) or meta-heuristics (Srinivas and Yu, 2022) and assume the environment to be fully deterministic, resulting in highly suboptimal decisions when applied in real warehouses.

To take into account the stochastic component of real-world scenarios, we propose to frame the CHR-OPS picker routing problem (henceforth, we will refer to it as CPR, for "Collaborative Picking Routing") as an instance of Online Bipartite Matching (OBM), a well-known problem in combinatorial optimization (CO) (Karp et al., 1990). In OBM, a set of known entities need to be dynamically assigned to entities from a different set (unknown a priori) which are disclosed sequentially, with the objective to maximize some metric depending on the final set of assignments. In CPR, orders are associated with AMRs beforehand, but due to the highly dynamic nature of warehouse environments it is very difficult to predict exactly when they will be available for picking. Therefore, orders can be modeled as incoming entities appearing randomly in the system. The task is then to assign, or match, orders to human pickers. Morever, while the number of pickers is usually fixed, their location is constantly changing as they move to retrieve items. In doing this, they are also affected by the stochasticity of the whole environment, and can also be considered as dynamically appearing in the system. Because of this, our scenario is more similar to the two-sided online bipartite matching (TOBM) in spatial data (Li et al., 2020).

Several learning-based methods have been applied to the OBM problem. Alomrani et al. (2023) address the one-sided OBM with deep reinforcement learning (DRL), showing good optimality ratios on the test instances. However, their approach considers one of the two sets of entities fixed and does not take into account the spatial component inherent in the order-picking routing problem. In the study from Wang et al. (2019), the authors adopt an approach combining DRL and traditional combinatorial optimization algorithms. Despite being able to handle online nodes from both sets to be matched, this approach still does not fully take spatial information into account.

To address these gaps, we develop and apply several end-to-end DRL approaches using three different graph neural network (GNN) architectures to extract information from the spatial data to guide decisions. We show that all methods achieve significantly better performances compared to greedy and more sophisticated heuristics in a simulated environment. Moreover, our methods show good transferability properties, even when scaling up test problem instances to more than forty times the size of the training ones.

To the best of our knowledge, this is the first work to frame CPR as an instance of TOBM, to deal with the high level of stochasticity characterizing warehouse environments. We develop and apply a set of deep reinforcement learning approaches for CPR which are able to scale to large problem instances, significantly beating benchmark heuristics. Moreover, it is also the first to work apply end-to-end DRL to TOBM.

2. Related work

Collaborative human-robot order picking The most common approach to CPR is the use of heuristics or rule-based systems. For instance, Azadeh et al. (2020) study the impact of different zoning strategies and propose a dynamic programming approach to dynamically switch between these strategies depending on the state of the warehouse. Many existing works on collaborative human-robot order picking focus on simultaneously optimizing all the decisions involved in the process, including order batching, batch releasing, and routing of pickers and robots. Srinivas and Yu (2022) develop a mixed integer linear programming (MILP) model taking into account all the mentioned decisions, which can only be solved exactly for very small problem instances. They then propose an approach combining two metaheuristics (simulated annealing and adaptive neighborhood search) to deal with larger instances. Given the complexity of the problem, other works try to isolate sub-problems and tackle them individually: Löffler et al. (2022) study the case of a single picker assisted by robots in a single-block warehouse and propose a dynamic programming approach to solve it. Recently, Krnjaic et al. (2023) approached CPR using Hierarchical Multi-Agent (deep) Reinforcement Learning (MARL). They treat both pickers and AMRs as individual agents and learn shared policies to decide the next item to serve for both classes of agents, outperforming simple heuristics like "Follow Me" and "Pick, Don't Move". All these works have one major assumption in common, which we believe to be the strongest and most problematic: they consider the whole environment as deterministic, while real-world warehouses are extremely dynamic and stochastic environments. To deal with this, we propose to tackle the problem with a fully reactive approach guided by Deep Reinforcement Learning.

Deep learning for combinatorial optimization In the last years, interest has grown among researchers for applying deep learning to combinatorial optimization problems (COP). Since many problems can be represented as graphs, graph neural networks are among the most promising architectures, as discussed by Cappart et al. (2022). Deep reinforcement learning has also quickly emerged as a powerful framework for COP, due to its inherent focus on optimization. GNNs and DRL can, therefore, be combined to achieve high quality solutions in a range of COPs. Kool et al. (2019) apply them together with an attention mechanism to solve routing problems, such as the Travelling Salesman Problem (TSP) and the Vehicle Routing Problem (VRP), which are naturally represented as graphs in 2D space.

However, all these problems are offline, meaning that there is no uncertainty involved. In fact, not many works have investigated the use of deep learning techniques for online combinatorial optimization. Among these, many combined deep learning with traditional optimization techniques: Zhao et al. (2022), for instance, achieved very high performances on the online 3D bin packing problem combining DRL with GNNs and heuristics. Wang et al. (2019) approaches the Two-sided Online Bipartite Matching problem using DRL to learn how to batch decisions in order to improve the matchings generated by the offline Hungarian optimization algorithm. Even fewer studies adopted an end-to-end learning approach. Peer et al. (2018) used an end-to-end Deep Q-Network (DQN) to solve the train shunting problem with stochastic arrivals and departures. Finally, Alomrani et al.

(2023) developed a DRL framework for the one-sided Online Bipartite Matching problem. They experiment with several neural network architectures, finding that working in a graph environment lets models achieve better optimality gaps than the greedy heuristic.

Despite being very powerful, this last framework only deals with the one-sided version of OBM, where the assumption of fixed entities is too strong to correctly model many real-world scenarios. On the other hand, the approach of Wang et al. (2019) tackles the two-sided problem, but does not work end-to-end, and is unable to exploit the information coming from the spatial data available in many real-world applications such as order picking.

3. Problem setting

In this section, we first introduce the real-world CPR and discuss its similarities to the online bipartite matching problem. Based on these similarities, we describe our environment model. Finally, we provide the formulations of the Markov Decision Processes used for the deep reinforcement learning tasks.

3.1. Collaborative picking as OBM

In the collaborative human-robot order picking routing problem, a set of pickers P work in a warehouse, assisted by a set R of AMRs. In the existing approaches, each AMR is assigned with a pre-generated pickrun, i.e. a sequence of locations to visit inside the warehouse and orders that must be picked at those locations. Since the pickrun is known in advance, the problem is usually treated as an instance of VRP. The objective of these strategies is typically to minimize the cumulative distance walked by the pickers or to minimize the cumulative tardiness of the picks, if orders have due dates. However, these solutions usually assume a deterministic behavior of the system, while real-world warehouses are very dynamic environments. For instance, AMRs could fail, pickers could take a break or some other kind of delay might occur, blocking an entire aisle for several minutes. To deal with this inherent stochasticity of the environment, we propose a framework where decisions of which picker assign to which AMR are taken only when required, instead of computed entirely beforehand.

One well-known setting that allows this is the Online Bipartite Matching (OBM) problem (Karp et al., 1990). Here we have two disjoint sets of entities U and V. U is fixed and known a priori, while elements $v \in V$ are disclosed sequentially together with potential matches $\{(u, v) : u \in U\}$ with elements of the other set. The task is to make instantaneous and irrevocable decisions on which match to select upon arrival for the element v. A common constraint is that elements of U cannot be matched more than once, and often the time horizon is fixed at $T = |V|$, so it is finite and known in advance. There are, however, a few differences between this formulation and CPR: first, neither the set of pickers nor that of AMRs are fixed. While the total number of elements in each set does not change, their locations and availability constantly change, effectively making them appear as different elements every time we look at the state of the system. Second, in the original OBM framework it is possible to skip a match and the incoming element simply disappears. This is not the case in CPR, as neither AMRs nor pickers will leave the system if they are not matched in a certain decision. For these reason, the task is more similar to the Two-Sided

Online Bipartite Matching Problem (TOBM) in spatial data introduced in Li et al. (2020), where entities of both sets U and V dynamically appear in the system.

3.2. Dynamic collaborative order picking environment

Based on the formulation of the CPR as TOBM, we introduce an online decision making environment settled in a grid-world, with grid points representing pick locations. Since the real-world problem does not have a real-time decision constraint, we model time as discrete, with each timestep potentially representing a real world time window of a few seconds.

Similar to the real-world CPR scenario, we have a set P of pickers moving in a square grid of size $n \times n$ to pick orders. We will use the term "order" instead of "AMR", because if an AMR is waiting in a picking location, it means that an order is available for picking there. Each order can be in three possible states: announced, ongoing, or tardy. The arrival of every order is first announced t_a timesteps before it is effectively available for picking. This represents the fact that, in real-world CPR, AMRs usually follow pre-assigned pickruns, so it is possible to know their future locations, but the exact time in which they will become available is only known with little anticipation due to the environment stochasticity. After t_a timesteps, an order's state switches to ongoing, and it is now pickable. From this moment, orders start to incur a holding cost C_H for each timestep in which they are not picked. Finally, every order is also associated with a number t_o of timesteps within which it should be picked after switching to ongoing, to represent the fact that orders have due dates. If an order is not picked within this timeframe, its state switches to tardy, and the cost it incurs per timestep increases to C_T. The number of timesteps since an order became tardy is tracked in t_T. Once an order is picked, it disappears from the system and immediately stops incurring any costs. At each timestep, up to a certain number N_o of new orders can be announced on the grid with a probability P_o. This probability is applied independently to the appearance of each new order. The locations in which new orders can appear are sampled randomly from a uniform distribution across all locations. In this way, we avoid modeling the behavior of AMRs in details, while still capturing the uncertainty it would introduce in the environment. If an order is set to appear in an occupied location, it is skipped, in line with the fact that orders represent AMRs, and two AMRs can not share a physical location in the warehouse. We therefore represent each order o as a tuple $(l^o, a^o, s^o, t^o_a, t^o_o, t^o_T)$, where l^o is its location on the grid and a^o is an integer value set to -1 if the order is unassigned, and equal to the index of the picker assigned to it otherwise.

During a timestep, each picker p can move, following Manhattan geometry, from its location l_p to a neighboring location. Pickers have two possible states: idle and assigned. Whenever a picker is idle, it[1] requires a decision: it can either be assigned to an unassigned order o or stay where it is. After being assigned to order o, the picker starts to move towards location l^o following the shortest path. Once order o is reached, if the order is in state ongoing or tardy the picker immediately picks it and becomes idle again, otherwise (if o is in state announced) the picker waits in the same location until the order switches to ongoing before performing the pick and returning to idle. Note that a picker may sometimes enter the location of an order, but can not pick that order unless instructed to. Each picker is

1. In real-world warehouses pickers are humans, but since here we consider simulation entities we will refer to them with the pronoun "it".

represented as a tuple (l^p, s^p, d^p), where l^p is its location, s^p its state and d^p its destination, i.e. the location of the order to pick. If the picker is idle, its destination is its current location l^p.

The uniformly distributed random arrival location of the orders mimics the difficulty of predicting much in advance where picks will be available due to the uncertainty associated with the behavior of the AMRs, while the announcement mechanism mimics that picks can be predicted shortly before the AMR arrives. Moreover, having discrete timesteps could be seen as applying a batching strategy to the decision times. This allows us to assess how different architectures are able to exploit these batches to coordinate decisions.

3.3. MDP formulation

We now provide two slightly different formulations of the Markov Decision Process upon which we build our different Deep Reinforcement Learning approaches. The only difference among these formulations is how the actions are shaped.

State At each timestep t, the state s_t is the set of all the nodes in the grid-graph as well as its distance matrix, and the list of nodes containing pickers and orders. In these formulations we choose not to include a terminal state.

Action Since in each timestep it is possible that more than one picker is idle and, therefore, requires a decision, such decisions will be taken sequentially to considerably reduce the number of possible actions per step. However, since the order in which the decisions are taken is very important, we propose two possible ways to deal with this problem.

- **Destination only action**: the order of the decisions is fixed. Once the picker is selected, the action represents the node of the grid-graph containing either the order assigned to the picker (i.e., its next destination) or the picker itself (skip assignment). In this setting, the maximum number of possible actions only depends on the size of the graph as it is equal to the number of nodes $|N|$.

- **Picker and destination action**: the picker who is going to act is part of the decision. Here, the actions are tuples (picker, destination). Clearly, in this setting the maximum number of actions at each timestep depends also on the number of pickers in the system: $|N||P|$.

Reward The reward function depends on the costs generated by non-picked ongoing and tardy orders. This means that we have both shared and delayed rewards, as these costs depend on the actions taken by multiple pickers and can only be seen some time after each decision has been taken. First, the environment generates a cost at each timestep, equal to $c_t = C_H \cdot N_t^O + C_T \cdot N_t^T$, where N_t^O, N_t^T, C_H and C_T are, in order, the number of orders in state ongoing and tardy at time t, holding and tardiness costs. To correctly assign the right credit to each action, we keep track of these costs for each picker: the share of reward generated by each picker applying action a at timestep t_i and accomplishing the task at time t_f is: $r_p = \sum_{t=ti}^{t_f-1} c_t/|P|$. Here we are dividing the costs by $|P|$, since all pickers contribute equally to the cumulative reward. Finally, we need to aggregate all the rewards generated by different pickers in one single reward signal. The reward at time t will be the sum of the rewards generated by the pickers $P_t \subset P$ finishing a task in that timestep: $r_t = -\sum_{p \in P_t} r_p$. Since the objective of DRL is to maximize a reward signal, we take the negative of the costs.

Objective In both formulations we want to minimize the cumulative costs received by the pickers. The objective function is simply the maximization of the cumulative reward: $R = \sum_t r_t$. Combining the two signals for holding and tardiness costs we are trying to find the trade-off between avoiding tardiness as much as possible and, at the same time, picking all the orders as quickly as possible.

4. DRL methods

4.1. Input representation

Based on the two MDP formulations, we propose three different approaches based on different neural network architectures. Our goal is to learn a centralized policy to match pickers to orders dynamically at each timestep. All three architectures work with data represented as a graph $G(N, E)$, where N is the set of nodes and E the set of edges connecting nodes of G. Since our environment is settled in a grid-world, we consider cells as nodes, and edges are available between nodes only if the respective cells are adjacent in the grid. At every timestep t, information about pickers and orders in current state is provided to the DRL agent as features of the graph's node, as can be seen in table 1.

This input representation is crucial for two reasons: first, considering pickers, orders and their respective characteristics as node features allows to learn policies which are independent from the number of such entities. This is because the information is treated per node instead of globally, and it is possible to apply the same computation to any given number of nodes. Second, representing information as a graph allows us to capture spatial patterns and exploit them to take better decisions.

Feature type	**Description**	**Variable type**	**Architectures**
Global features	Distance matrix containing the distances between each node of the grid-graph	Matrix (Int)	EA, MH
	List of locations of the pickers as integers in range $[0, \|N\|-1]$	List(Int)	EA
Node features	Node contains the controlled picker	Bool	NA
	Number of (other) idle pickers in the node	Int	NA, EA, MH
	Number of (other) assigned pickers in the node	Int	NA, EA, MH
	If node contains the destination of an(other) picker, distance between node and current location of the approaching picker (0 otherwise)[2]	Int	NA, EA, MH
	If node contains an order in state announced, t_a	Int	NA, EA, MH
	If node contains an order in state ongoing, t_o	Int	NA, EA, MH
	If node contains an order in state tardy, t_T	Int	NA, EA, MH
	Distance between the node and the location of the controlled picker	Int	NA

Table 1: Input features of the different network architectures

2. Pickers waiting in the same location of an order are considered one step away.

4.2. Sequential actions and masking

To deal with the issue of multiple decisions required at the same time, we propose to take them sequentially, within the same timestep. This allows to internally modify the state when an action is taken (without letting time advance nor any external event happen) in order to increase the information available when taking the next decision in the same timestep. In particular, this allows to mask orders that have already been assigned to other pickers, so that they cannot be chosen as matching.

The action masking system is a crucial component of our framework, since it is used to switch from the grid-world space to the bipartite matching setting. As we showed before, using a graph it is possible to create architectures invariant to the number of pickers and AMRs in the system. To do so, the output of the neural network must be proportional to the number of nodes in the original graph, many of which are empty, contain other pickers or contain orders already assigned to other pickers, and should not be part of the bipartite graph since they would not be feasible matchings. Masking these nodes results in allowing only actions that are available in the MDP formulations based on the TOBM problem.

4.3. Architectures

The first layers are the same (with differences in the input layer depending on the features) for all the proposed architectures. Three layers of GNN are applied to aggregate information and create embeddings for each node. Note that for this process we use the grid-graph representing the state of the whole environment and not the bipartite graph for which we want to find the best matching. These embeddings are then processed in different ways to obtain different outputs, based on the type of action required by the underlying MDP. In particular, the only actions available will be related to nodes containing unassigned orders and the current location of the picker, equivalent to the 'skip assignment' action. Clearly, if the action selected for a picker is to remain in the same location it will remain idle, triggering a new decision request in the next timestep.

Node Actions (NA) The first architecture is based on the *destination only action* MDP formulation. Given that the picker for which the decision needs to be taken has already been selected by the environment, the action represents the destination of the selected picker. The node embeddings are then individually processed with an MLP, resulting in an output of one single value per node, which, after the application of a softmax activation function, represent the probability of choosing each node as the destination n_d for the controlled picker. Finally, to avoid choosing empty or non-allowed destinations, those are masked before applying the softmax function.

In this setting the number of actions is equal to the number N of nodes in the graph. However, while during the training phase it is important that the MLP is applied to all node embeddings in order to correctly propagate the errors, at inference time it is possible to apply the action mask before the application of the MLP, saving a considerable amount of computation.

Edge Actions (EA) As mentioned in Subsection 3.3, the order in which multiple decisions are taken in the same timestep is important. Hence, letting the environment choose it may lead to highly suboptimal policies, with a low level of coordination between the pickers.

To solve this issue we embed the choice of the acting picker in the decision and, therefore, in the learning process. Instead of processing all node embeddings individually, we use the locations of the pickers provided in the input to select the "origin" nodes.

We then take the cartesian product of this "origin" nodes and all the nodes in the graph (the possible destinations), to which we concatenate the distance between them, obtained through the distance matrix, to generate embeddings for all the edges connecting a picker with a possible location (destination) in the grid. These edge embeddings are then processed with MLP and softmax to obtain a probability distribution over the edges.

The number of actions in this setting is $|P||N|$, which leads to a very fast growth of the action space. Due to this, the application of the mask is even more important, as it allows to discard actions on two dimensions: for instance, if a picker is not idle it should not be selected, which leads to an effective masking of $|N|$ possible actions at the same time.

Multi-headed Actions (MH) The previous architecture solves the issue of coordination between pickers, but at a very high cost in terms of required computation, especially for large problem instances. This problem is not new in combinatorial optimization. One possible way to deal with it is to decouple the decision, tackling different subproblems sequentially (Zhao et al., 2021), exactly as we are already doing for the main matching problem. This means first deciding which picker should act, and then picking its destination. Since we still have a graph input, instead of selecting a picker directly we select the node containing it, then the simulation environment takes care of understanding which idle picker is in that location. If multiple pickers are idle in the same selected location, then one of them will be selected randomly, as there are no additional constraints to take into account. The actions learned by the DRL agent now take the form (n_p, n_d), where $n_p \in N$ is the location of the selected picker, and $n_d \in N$ its destination.

To do this, we first process the node embeddings with an MLP and softmax as in the NA approach, except for the fact that instead of masking all nodes not containing orders we mask those not containing pickers. We then take the value having the highest probability from the output distribution and use it to index the distance matrix to obtain the distance of the selected node (the one containing the acting picker) from all other nodes in the grid. Finally we concatenate the node embeddings each node embedding with the corresponding distance and process the result with an MLP and softmax to have the probability distribution over possible destination nodes.

Since the output size of each head of the neural network is equal to the number of nodes $|N|$, the total number of actions in this setting is only $2|N|$. This approach should then be able to considerably outperform **NA**, learning which picker should act first in each situation, while scaling much better than **EA**, since its output grows linearly with the number of nodes $|N|$ and is unaffected by the number of pickers $|P|$.

4.4. Algorithms and hyperparameters

We trained our DRL agents using Proximal Policy Optimization (PPO) (Schulman et al., 2017) a popular model-free RL baseline due to its effectiveness and sample efficiency. We already described the architecture for the policy heads, the critic heads simply processes all the node embeddings through an MLP with one hidden layer of size 64, and then applies a global mean pooling to obtain one single value for the whole graph input.

In our architectures we used multiple Graph Isomorphism Network (GIN) (Xu et al., 2019) layers to create node embeddings. We chose GIN as it is at the same time very powerful and quite simple. The message passing formula is $\mathbf{x}'_i = h_{\mathbf{\Theta}} \left((1+\epsilon) \cdot \mathbf{x}_i + \sum_{j \in \mathcal{N}(i)} \mathbf{x}_j \right)$, where $h_{\mathbf{\Theta}}$ represents a MLP of which we set the size to 64. The hyperparameters used to train the final agents are the default ones from Tianshou's PPO implementation[3], including a discount factor of 0.99. We apply a learning rate scheduler, with an exponential decay of 0.9 starting from a learning rate of 0.005 for the Adam optimizer.

5. Experiments

5.1. Experimental setup

We use the simulation environment described earlier to train our DRL agents. In particular, since we are interested in assessing the transferability of the proposed methods, we train on a single, small environment (Env 1) configuration and evaluate on 4 additional ones with different characteristics (Env 2-5). The details of these environments are in table 2.

Despite having different sizes, Env 1, 2, 4, 5 and 6 share the same incoming order-to-picker ratio and present similar picker-to-node and order-to-node ratios. Env 3, on the other hand, has a much smaller order-to-picker ratio, which increases considerably the level of coordination between pickers required from the controller agent to achieve good performances.

Environment name	**Grid size**	**|P|**	$\mathbf{N_o}$	**Training**	**Testing**
Env 1	6 × 6	2	1	True	True
Env 2	10 × 10	6	3	False	True
Env 3	10 × 10	6	1	False	True
Env 4	20 × 20	24	12	False	True
Env 5	40 × 40	96	48	False	True
Env 6[4]	80 × 80	384	192	False	True

Table 2: Environment configurations. P is the set of pickers and N_o the maximum number of new orders announced per timestep.

Larger grid sizes have an impact on the expected rewards, as the pickers need more timesteps to reach orders because of increased distances, which is reflected in additional holding costs. Other parameters are kept the same across all configurations: (1) Holding and tardiness costs C_H and C_T are set, respectively to 1 and 10. (2) Probability of new order per timestep P_o: 0.8. This probability influences every 'new order' event independently, meaning that in an environment with 1 order per timestep, an average of 0.8 orders per timestep will actually appear on the grid, while 2.4 orders will appear on average in an environment with 3 maximum orders per timesteps. (3) New order location distribution: uniform across grid locations. (4) Time windows range: [5, 10]. All newly generated t_a

3. Default parameters of PPO policy in Tianshou.
4. Because of the very large size of the instances, only used for assessing inference time.

and t_o (timesteps before orders' states switch, respectively, from announced to ongoing and from ongoing to tardy) are in this range. All instances are randomly initialized, except for three fixed parameters: (1) All pickers are idle in the initial state; (2) A number of orders between 0 and *grid size* - 1 is initialized in the environment; (3) Initial orders can be in announced or ongoing state, but not tardy.

The dynamic CPR environment was implemented in Python 3.9 with PettingZoo (Terry et al., 2021). The DRL methods were implemented using Pytorch and Pytorch Geometric for the neural networks and Tianshou for PPO. Training and testing experiments were performed on an Intel(R) Core(TM) i7-7700HQ CPU @ 2.80GHz and 16GB of RAM, without the use of GPU. All policies were trained for 3 million timesteps and each training instance required less than 4 hours.

5.2. Baselines

Since we work on an online problem, we cannot apply classical TSP or VRP based solutions common in the order picking literature. At the same time, having shared and delayed rewards makes approaches for different OBM (Alomrani et al., 2023; Wang et al., 2019) also not suitable for our problem. Therefore, following the findings of Li et al. (2020), according to whom greedy heuristics work better for TOBM problems in spatial data than permutation search and online algorithms, we devised four heuristics, following the same main algorithm, referred to as **GD** (Greedy Distance).
Given an idle picker $p \in P$:

1. All unassigned orders in state *tardy* or *ongoing* are sorted based on their distances $dist(l^o, l^p)$ from the controlled picker p.
   (a) The closest order is chosen as the destination of picker p.
2. If there are no orders in state *tardy* or *ongoing*, unassigned *announced* orders are sorted, again based on their distances from p.
   (a) The closest order is chosen as destination for p.
3. Finally, if no *announced* are in the system, p stays in the same location.

As this procedure leads to a lack of coordination between the pickers, we extended 1a and 2a so that the first order o in the sorted set is chosen as destination only if there are no other idle pickers closer to it than p ($dist(l^o, l^p) < dist(l^o, l^{p'})\ \forall p' \in P^i \setminus p$, where P^i is the set of idle pickers). We refer to this *coordinated* heuristic as **CD** (Coordinate Distance).

We improve further by changing the sorting criteria in points 1 and 2 from the distance between order o and the controlled picker p to a more sophisticated *cost-based* metric, which considers for each pair of orders (o, o'), the incurred cost if p picked o before *o'* and vice-versa, trying to foresee the change of the environment to a limited degree. Combining this *cost-based* metric with the above two algorithms, we define two additional heuristics, referred to as **CC** (Coordinated Cost-based) and **GC** (Greedy Cost-based) depending on whether they adopt the *coordinated* strategy for points *1a* and *2a* or not.

Finally, to assess the importance of capturing spatial information in node embeddings through GNNs, we apply the same structure run the three proposed methods with a simpler architecture, based on Inv-FF, as described in Alomrani et al. (2023), which do not involve any GNN layer and simply process raw input features instead of node embeddings. We refer to these architecture as **NA-InvFF**, **EA-InvFF** and **MH-InvFF**, in contrast with **NA-GNN**, **EA-GNN** and **MH-GNN** representing our proposed approaches using GNNs.

We evaluate our methods directly on the average reward of a set number of episodes of fixed length. In addition to pure performances, we also assess inference time required for each decision of the different methods, to evaluate efficiency and scalability.

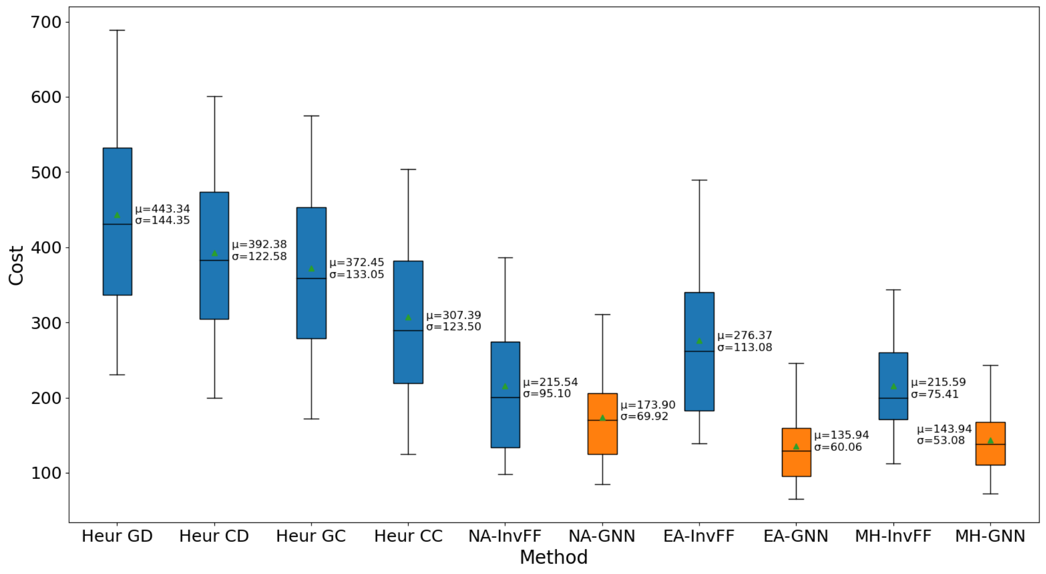

Figure 1: Performances of the various methods on the environment Env 1.

5.3. Results

First, we assess the performances of the different methods on Env 1 - the same used to train the DRL agents. Figure 1 shows the distributions of cumulative costs over 100 episodes of 100 timesteps each. We notice that all DRL approaches clearly outperform the heuristics by a considerable margin. Among these, the methods using GNN architectures, in orange, achieve the best results, with **EA-GNN** improving over 55% and almost 37% on, respectively, the best performing heuristic and Inv-FF approach. This shows how exploiting spatial information through graph neural networks can contribute to making better informed decisions in a highly stochastic environment like the one at hand.

One of our goals is to train agents which can be used effectively on different environments, without any additional retraining effort. Figure 2 reports the performances of the different methods on the remaining environments presented in table 2 (excluding Env 6). It can be seen that, for Env 2, Env 4 and Env 5, the GNN-based approaches (in orange) still significantly outperform the other methods. This shows that the use of GNNs really allows to capture spatial patterns that the agent can exploit to act better in different environments, with the largest (Env 5) being more than forty times as large as the one used for training (Env 1). For Env 2 and 4 **EA-GNN** still provides the best results, with a 10%

improvement over **MH-GNN**. However, results are missing for Env 5 for both **EA-GNN** and **EA-InvFF**, meaning that, despite the improved masking system, these method do not scale efficiently and would probably not be suitable for real-world sized problem instances. On Env 5, **MH-GNN** is therefore the best approach, achieving an improvement of 37% and 75% on, respectively, **NA-InvFF** and **CC**.

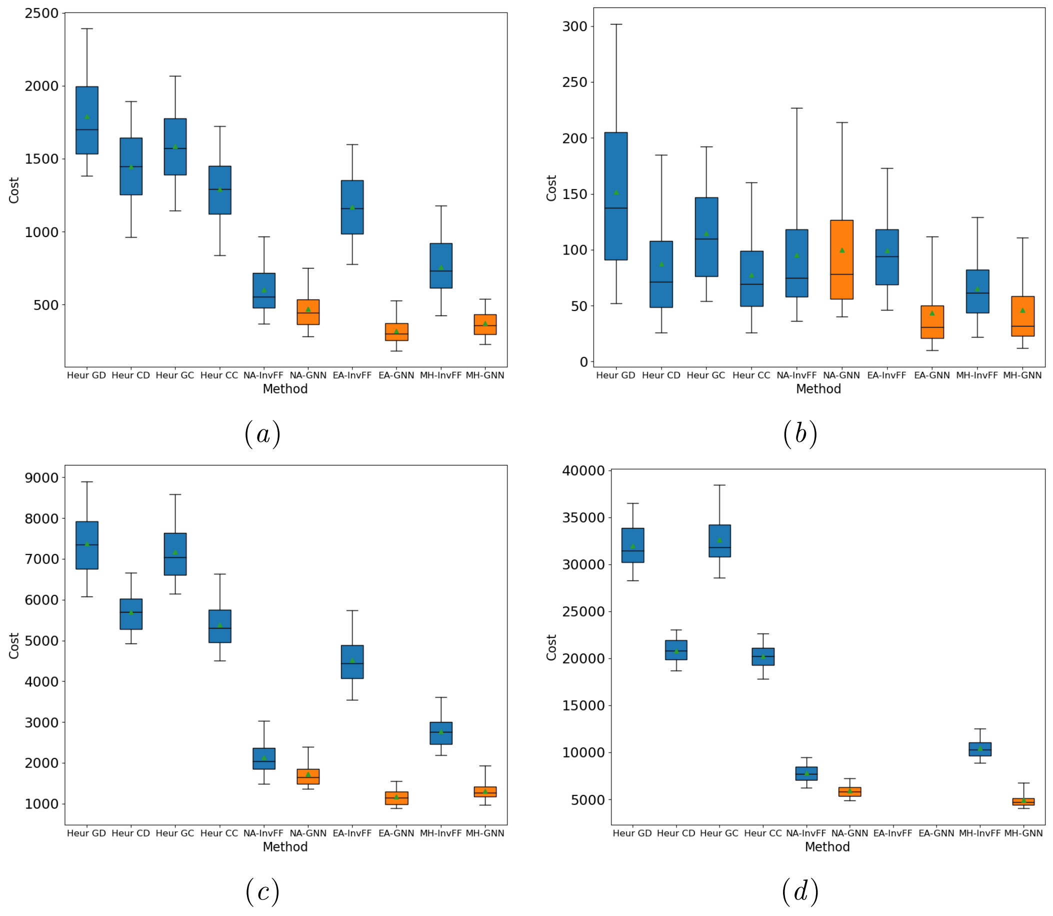

Figure 2: Comparison of the performances of various methods on Env 2 (a), Env 3 (b), Env 4 (c), Env 5 (d). All DRL-based methods are trained on Env 1. In d, results are missing for EA-InvFF and EA-GNN since neigher of the two methods managed to finish the test runs within the maximum dedicated time of four hours.

An interesting point is raised by the performances on Env 3: since in this configuration there are, on average, more idle pickers than pickable orders, coordination between pickers becomes extremely important. Because of this, **NA-GNN** is not able to outperform the heuristics that exploit picker coordination (**CD** and **CC**), while **EA-GNN** and **MH-GNN**, which select the acting picker as part of the decision, achieve better results.

Finally, we are interested in estimating how our methods compare with the baselines and with each other in terms of inference time. As already mentioned, in real-world CPR there

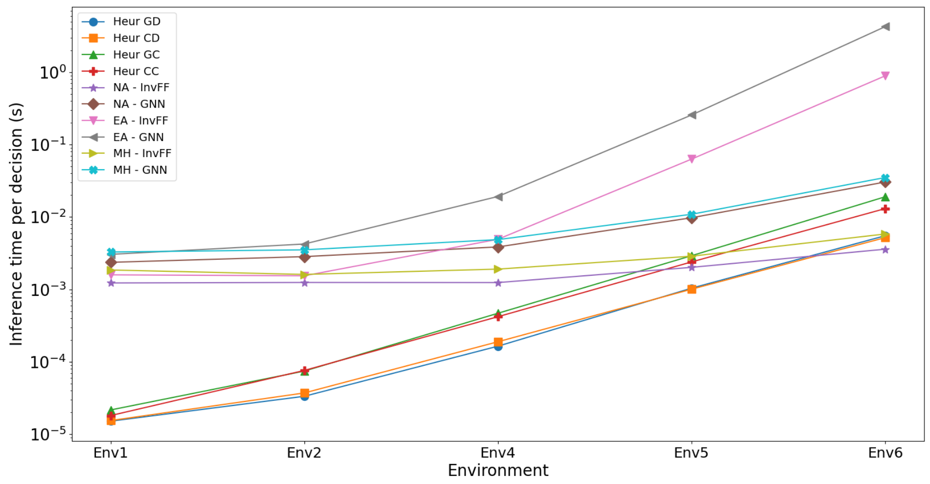

Figure 3: Decision time required for each method. Since, especially for the heuristics in small environment configurations, these times are close to zero, the logarithmic scale allows to better evaluate the trends.

is no hard constraint on real-time decisions. However, understanding how the parameters of the environment influence the time needed for computing actions is crucial to assess which methods can actually be applied to problem instances larger than those considered so far. To do so, we kept track of the time required to compute each decision during the experiments discussed earlier, and in additional tests run with a limited number of episodes on the even larger Env 6 as well.

Figure 3 shows the average inference time required by the different methods for each decision across the environment configurations (leaving out Env 3, which presents a different order-to-picker ratio). As the size of the problem instances increase, the time required to compute actions also grows for every method. It appears clear, though, that this growth follows very different trends. While the heuristics are much faster than DRL approaches on small environment configurations, this gap constantly shrinks, until they are already slower than NA-InvFF and MH-InvFF in the largest problem instances. The same would most likely happen with **NA-GNN** and **MH-GNN** in even larger environment configurations. Finally, the plot makes clear how, despite its excellent performance, the **EA-GNN** architecture is not scalable. Its action space corresponds to the size of the grid-graph multiplied with the number of pickers in the system, as described in Subsection 4.3. Overall, **MH-GNN** appears to be the best trade-off between performance and scalability, with its action space being linear in the size of the graph, and independent of the number of pickers.

6. Conclusion

We propose to frame the Collaborative Human-Robot Order Picking problem as Two-sided Online Bipartite Matching with spatial structure. This allows us to better deal with the inherent stochasticity of warehouse environments compared to existing approaches, which plan long sequences of decisions in advance. We adopt DRL to solve the OBM problem, and devise three different Graph Neural Network architectures to allow the DRL agent to capture the relevant spatial information on which to base its decisions. We show that our approaches achieve significantly better performances compared to heuristic strategies, while also exhibiting good and efficient scalability to larger problem instances. Among these, MH-GNN shows the best trade-off between performance and efficiency. Possible future research directions include the extension of the proposed methods to even larger and more realistic warehouse scenarios and the use of model-based DRL techniques to improve decisions planning ahead, exploiting the knowledge about future pick locations.

Acknowledgments

This research was made possible by TKI Dinalog and the Topsector Logistics and has received funding from the Ministry of Economic Affairs and Climate Policy (EZK) of the Netherlands.

References

Mohammad Ali Alomrani, Reza Moravej, and Elias B Khalil. Deep Policies for Online Bipartite Matching: A Reinforcement Learning Approach. *Transactions on Machine Learning Research*, 2023.

Kaveh Azadeh, Debjit Roy, and M.B.M. René de Koster. Dynamic Human-Robot Collaborative Picking Strategies. *SSRN Electronic Journal*, 2020. ISSN 1556-5068.

Martijn Beeks, Reza Refaei Afshar, Yingqian Zhang, Remco Dijkman, Claudy Van Dorst, and Stijn De Looijer. Deep Reinforcement Learning for a Multi-Objective Online Order Batching Problem. *Proceedings of the International Conference on Automated Planning and Scheduling*, 32:435–443, June 2022. ISSN 2334-0843, 2334-0835.

Bram Cals, Yingqian Zhang, Remco Dijkman, and Claudy van Dorst. Solving the online batching problem using deep reinforcement learning. *Computers & Industrial Engineering*, 156:107221, June 2021. ISSN 03608352.

Quentin Cappart, Didier Chételat, Elias Khalil, Andrea Lodi, Christopher Morris, and Petar Veličković. Combinatorial optimization and reasoning with graph neural networks, September 2022.

Sebastian Henn. Order batching and sequencing for the minimization of the total tardiness in picker-to-part warehouses. *Flexible Services and Manufacturing Journal*, 27(1):86–114, March 2015. ISSN 1936-6582, 1936-6590.

Ying-Chin Ho, Teng-Sheng Su, and Zhi-Bin Shi. Order-batching methods for an order-picking warehouse with two cross aisles. *Computers & Industrial Engineering*, 55(2): 321–347, September 2008. ISSN 03608352.

R. M. Karp, U. V. Vazirani, and V. V. Vazirani. An optimal algorithm for on-line bipartite matching. In *Proceedings of the twenty-second annual ACM symposium on Theory of* *computing - STOC '90*, pages 352–358, Baltimore, Maryland, United States, 1990. ACM Press. ISBN 978-0-89791-361-4.

Wouter Kool, Herke van Hoof, and Max Welling. Attention, Learn to Solve Routing Problems!, February 2019.

Aleksandar Krnjaic, Raul D. Steleac, Jonathan D. Thomas, Georgios Papoudakis, Lukas Schäfer, Andrew Wing Keung To, Kuan-Ho Lao, Murat Cubuktepe, Matthew Haley, Peter Börsting, and Stefano V. Albrecht. Scalable multi-agent reinforcement learning for warehouse logistics with robotic and human co-workers, 2023.

Yiming Li, Jingzhi Fang, Yuxiang Zeng, Balz Maag, Yongxin Tong, and Lingyu Zhang. Two-sided online bipartite matching in spatial data: experiments and analysis. *GeoInformatica*, 24(1):175–198, January 2020. ISSN 1384-6175, 1573-7624.

Maximilian Löffler, Nils Boysen, and Michael Schneider. Picker Routing in AGV-Assisted Order Picking Systems. *INFORMS Journal on Computing*, 34(1):440–462, January 2022. ISSN 1091-9856, 1526-5528.

Evertjan Peer, Vlado Menkovski, Yingqian Zhang, and Wan-Jui Lee. Shunting Trains with Deep Reinforcement Learning. In *2018 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, pages 3063–3068. IEEE, October 2018. ISBN 978-1-5386-6650-0.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal Policy Optimization Algorithms, August 2017. arXiv:1707.06347 [cs].

Sharan Srinivas and Shitao Yu. Collaborative order picking with multiple pickers and robots: Integrated approach for order batching, sequencing and picker-robot routing. *International Journal of Production Economics*, 254:108634, December 2022. ISSN 09255273.

J. K. Terry, Benjamin Black, Nathaniel Grammel, Mario Jayakumar, Ananth Hari, Ryan Sullivan, Luis Santos, Rodrigo Perez, Caroline Horsch, Clemens Dieffendahl, Niall L. Williams, Yashas Lokesh, and Praveen Ravi. PettingZoo: Gym for Multi-Agent Reinforcement Learning, October 2021.

Yansheng Wang, Yongxin Tong, Cheng Long, Pan Xu, Ke Xu, and Weifeng Lv. Adaptive Dynamic Bipartite Graph Matching: A Reinforcement Learning Approach. In *2019 IEEE 35th International Conference on Data Engineering (ICDE)*, pages 1478–1489, Macao, Macao, April 2019. IEEE. ISBN 978-1-5386-7474-1.

Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How Powerful are Graph Neural Networks?, February 2019.

Hang Zhao, Chenyang Zhu, Xin Xu, Hui Huang, and Kai Xu. Learning practically feasible policies for online 3D bin packing. *Science China Information Sciences*, 65(1):112105, December 2021. ISSN 1869-1919.

Hang Zhao, Yang Yu, and Kai Xu. Learning Efficient Online 3D Bin Packing on Packing Configuration Trees. *International Conference on Learning Representations*, 2022.