Learning to Solve the Min-Max Mixed-Shelves Picker-Routing Problem via Hierarchical and Parallel Decoding

Laurin Luttmann[1] and Lin Xie[2]

[1] Leuphana University, Lüneburg, Germany
[2] Brandenburg University of Technology, Cottbus, Germany

Abstract. The Mixed-Shelves Picker Routing Problem (MSPRP) is a fundamental challenge in warehouse logistics, where pickers must navigate a mixed-shelves environment to retrieve SKUs efficiently. Traditional heuristics and optimization-based approaches struggle with scalability, while recent machine learning methods often rely on sequential decision-making, leading to high solution latency and suboptimal agent coordination. In this work, we propose a novel hierarchical and parallel decoding approach for solving the min-max variant of the MSPRP via multi-agent reinforcement learning. While our approach generates a joint distribution over agent actions, allowing for fast decoding and effective picker coordination, our method introduces a sequential action selection to avoid conflicts in the multi-dimensional action space. Experiments show state-of-the-art performance in both solution quality and inference speed, particularly for large-scale and out-of-distribution instances. Our code is publicly available at http://github.com/LTluttmann/marl4msprp

Keywords: Picker Routing · Mixed-Shelves Warehouses · Neural Combinatorial Optimization · Multi-Agent Reinforcement Learning

1 Introduction

Order picking, the process of retrieving items from a warehouse to fulfill customer orders, is a fundamental and often the most costly operation in warehouse logistics [8,4]. In conventional picker-to-parts warehouses, most of a picker's time is spent traveling between the shelves of the storage area [7,22]. To mitigate travel times, mixed-shelves storage strategies – where Stock-Keeping Units (SKUs) are distributed across multiple shelves of the storage area rather than assigned to a single location – have gained traction in recent years, demonstrating the potential to reduce travel distances and enhance overall efficiency [5,26,28,27,19]. This mixed-shelves approach gives rise to the Mixed-Shelves Picker Routing Problem (MSPRP), which focuses on determining optimal routes for pickers while considering the unique constraints of mixed-shelves warehouses and operations. Despite its practical significance, research on solving the MSPRP remains limited. While classical heuristics [26,27,7] and optimization-based methods often struggle to scale to large problem instances, emerging Neural Combinatorial

Optimization (NCO) approaches show promise in delivering high-quality solutions in near real-time [14,20,2]. However, existing NCO work on the MSPRP is largely limited to single-picker scenarios minimizing total distance [19]. In practical warehouse settings, multiple pickers operate concurrently, making the minimization of the maximum tour length among all pickers a more critical objective for maximizing overall throughput [21,18,2,12]. To bridge this gap, this work proposes the Multi-Agent Hierarchical Attention Model (MAHAM), designed to efficiently solve the min-max variant of the MSPRP via a hierarchical and parallel decoding strategy. Extending the Hierarchical Attention Model [19], MAHAM employs a multi-agent encoder for coordination and a decoder generating action probabilities for all pickers in parallel. This parallel approach enables fast decoding and learned workload balancing. To navigate the complex, multi-dimensional action space and avoid conflicts, MAHAM introduces a novel sequential action selection strategy, which leverages learned agent priorities and dynamic masking to iteratively determine feasible agent actions from their joint probability distribution. Experiments demonstrate MAHAM achieves state-of-the-art solution quality and inference speed, particularly generalizing well to large and out-of-distribution instances.

2 Related Work

Mixed-shelves Picker Routing. Various heuristics have been developed to address the MSPRP, including construction and improvement methods [25,26] and a variable neighborhood search approach [27]. However, these methods often require minutes of computation, which can be impractical for fast-paced operations. Only one neural learning approach has been proposed for the MSPRP, modeling it as a heterogeneous graph to optimize selection and routing for a single picker [19]. In practice, multiple pickers operate simultaneously, shifting the focus towards minimizing overall completion time instead of total travel distance. We thus explore a min-max variant of the MSPRP, aiming to balance travel distances among pickers.

Neural Combinatorial Optimization. While early work in the NCO field focus on problems involving a single agent like in the traveling salesman problem [24,14,16,17], recently more attention has been given to more complex, multi-agent variants of routing problems. Building on [14], the Equity Transformer [21], DPN [29], and 2d-Ptr [18] introduce attention-based policies for multi-agent min-max routing. However, these models can be seen as purely autoregressive approaches, constructing solutions for one agent at a time, thus neglecting potential agent coordination and exhibiting high generation latency for large problems with many agents. PARCO [2] aims to address these shortcomings by introducing parallel solution construction, using a Priority-based Conflict handler to avoid infeasible solutions when performing actions for multiple agents simultaneously.

In this work, we combine the hierarchical decoder of [19], designed for the integrated selection and routing in MSPRP, as well as parallel solution construction similar to [2], to learn high quality solutions for the min-max MSPRP. To

effectively avoid conflicts during hierarchical solution construction, we combine a Parallel Pointer Mechanism with a Sequential Action Selection algorithm.

3 Problem Formulation

3.1 Formal Definition of the MSPRP

This work focuses on a min-max variant of the MSPRP with split orders and split deliveries covered in [19]. The split orders assumption allows items of an order to be picked within different tours and split deliveries relaxes the assumption that the demand for an SKU must be satisfied by a single picker tour [28]. A tour is defined by the storage locations visited between two successive visits to a packing station $h \in \mathcal{V}^{\mathrm{D}}$, where picked items are unloaded and commissioned. During a tour, no more than κ units can be picked. Further, due to the mixed-shelves storage policy each shelf may consist of multiple storage locations or compartments storing units of different SKUs. Also, the mixed-shelves storage policy allows each SKU p to be retrieved from multiple storage locations $i \in \mathcal{V}_p^{\mathrm{S}}$.

The goal of the min-max MSPRP is to pick all d_p demanded units of all requested SKUs $p \in \mathcal{P}$ and returning them to a packing station $h \in \mathcal{V}^{\mathrm{D}}$ while minimizing the maximum travel distance among the individual pickers $m = 1, ..., M$, henceforth also called agents. Note that in order to compare our proposed method against baselines [19], [18], and [21], we assume that the number of agents is equal to the number of tours required to collect all demanded items given the picker capacity κ (i.e. $M = \left\lceil \frac{\sum_p d_p}{\kappa} \right\rceil$). The following mathematical model describes the min-max MSPRP covered in this work and Table 1 summarizes the notation used to define the model.

Table 1: Notation used in the MIP-Model

Symbol	Description
$\mathcal{P}$	Set of SKUs for picking
$\mathcal{V}$	Set of storage locations and packing stations ($\mathcal{V} = \mathcal{V}^{\mathrm{S}} \cup \mathcal{V}^{\mathrm{D}}$)
$\mathcal{E}$	Set of edges $\{(i,j) \mid i,j \in \mathcal{V},\, i \neq j\}$
$\mathcal{V}_p^{\mathrm{S}}$	Set of storage locations including picking item $p \in \mathcal{P}$
$\mathcal{B}$	Set of required tours $\{1, 2, ..., \lvert\mathcal{B}\rvert\}$
D_{ij}	Distance between two nodes $(i,j) \in \mathcal{E}$
κ	Maximum picking capacity per tour
d_p	Total demand for item $p \in \mathcal{P}$
n_i	Available supply at storage location $i \in \mathcal{V}^{\mathrm{S}}$
x_{ijb}	Binaray variable, indicating whether node $j \in \mathcal{V}$ has been visited after node $i \in \mathcal{V}$ in tour $b \in \mathcal{B}$
y_{ib}	Units picked up at location $i \in \mathcal{V}^{\mathrm{S}}$ in tour $b \in \mathcal{B}$

$$\textbf{min} \quad Z = \max_{b \in \mathcal{B}} \sum_{(i,j) \in \mathcal{E}} D_{ij} \cdot x_{ijb} \tag{1}$$

$$\textbf{s.t.} \quad \sum_{(i,j) \in \mathcal{E}} x_{ijb} = \sum_{(j,i) \in \mathcal{E}} x_{jib} \qquad \forall\, i \in \mathcal{V}, b \in \mathcal{B} \tag{2}$$

$$\sum_{(i,j) \in \mathcal{E}} x_{ijb} \leq 1 \qquad \forall\, i \in \mathcal{V},\, b \in \mathcal{B} \tag{3}$$

$$BigM \cdot \sum_{i \in \mathcal{V}} x_{ijb} \geq y_{jb} \qquad \forall\, j \in \mathcal{V}^{\mathrm{S}},\, b \in \mathcal{B} \tag{4}$$

$$\sum_{h \in \mathcal{V}^{\mathrm{D}}} \sum_{j \in \mathcal{V}^{\mathrm{S}}} x_{hjb} = 1 \qquad \forall\, b \in \mathcal{B} \tag{5}$$

$$\sum_{i \in S} \sum_{j \in S} x_{ijb} \leq |S| - 1 \qquad \forall\, b \in \mathcal{B},\, S \subset \mathcal{V}^{\mathrm{S}}, |S| \geq 2 \tag{6}$$

$$\sum_{i \in \mathcal{V}^{\mathrm{S}}} y_{ib} \leq \kappa \qquad \forall b \in \mathcal{B} \tag{7}$$

$$\sum_{i \in \mathcal{V}_p^{\mathrm{S}}} \sum_{b \in \mathcal{B}} y_{ib} = d_p \qquad \forall\, p \in \mathcal{P} \tag{8}$$

$$\sum_{b \in \mathcal{B}} y_{ib} \leq n_i \qquad \forall\, i \in \mathcal{V}^{\mathrm{S}} \tag{9}$$

$$x_{ijb} \in \{0, 1\} \qquad \forall\, (i,j) \in \mathcal{E},\, b \in \mathcal{B} \tag{10}$$

$$y_{ib} \geq 0 \qquad \forall\, i \in \mathcal{V}^{\mathrm{S}},\, b \in \mathcal{B} \tag{11}$$

The objective function (1) aims to minimize the maximum distance traveled by any picker. Constraints (2) ensure that every storage location visited during a picker's tour is also exited. Constraints (3) prevent storage locations from being visited multiple times within a single tour, though multiple visits are allowed across tours if the picker's capacity is insufficient to fulfill the demand in one trip. However, revisiting the same storage location within a single tour is inefficient and therefore disallowed. Using the Big-M formulation in (4), we ensure that items can only be picked from storage locations included in the respective tour. Since no more than κ items can be picked in one tour, setting $BigM = \kappa$ is sufficient. To guarantee that each tour begins and ends at a packing station, constraints (5) enforce that a packing station is exited exactly once per tour. Combined with the network flow constraints (2), this ensures that every tour returns to the packing station it initially departed from. Additionally, subtour elimination constraints (6) ensure that all visited storage locations are connected within a tour. Constraints (7) prevent the picker's capacity from being exceeded, while constraints (8) ensure that all customer orders are fulfilled. To avoid exceeding the available stock of items in any storage location during order picking, constraints (9) are enforced. Finally, constraints (10) and (11) define the domains of the decision variables x and y.

3.2 Markov Decision Process Formulation

The min-max MSPRP can be modeled as a cooperative Multi-Agent Markov Decision Process (MMDP) with M agents sharing a common reward. An MMDP is defined as $(\mathcal{S}, \mathcal{M}, \{\mathcal{A}_i\}, \Gamma, R)$, where $\mathcal{S}$ and $\mathcal{M}$ are finite sets of states and agents, respectively. Each agent m selects actions from $\mathcal{A}_m$, with the joint action space denoted as $\boldsymbol{\mathcal{A}}$. The transition function Γ determines state changes based on actions, and R is the shared reward function.

MMDPs involve sequential decision-making, where agents select and execute actions at each step until a terminal state s_T is reached. The min-max MSPRP is framed as an MMDP, with pickers (agents) visiting warehouse shelves to fulfill SKU demands. A shared θ-parameterized policy determines the next location and SKU to pick. This chapter formally defines the min-max MSPRP as an MMDP, specifying its state, action space, transition rule, and reward function.

State. The state s_t of the min-max MSPRP at step t can be represented as a heterogeneous graph $\mathcal{G} = (\mathcal{V}, \mathcal{P}, \mathcal{M}, E_t)$ with pickers, warehouse locations and SKUs posing different types of nodes in the graph. The set of warehouse locations $\mathcal{V}$ is the collection of all packing stations $\mathcal{V}^{\mathrm{D}}$ and shelves $\mathcal{V}^{\mathrm{R}}$. The state of an SKU $p \in \mathcal{P}$ is defined by its remaining demand d_{pt} at step t. Moreover, edges with weights E_t connect shelf and SKU nodes, specifying the storage quantity e_{vpt} of an item p in the respective shelf v at time t. Lastly, the state of the pickers $m \in \mathcal{M}$ is defined by their current location v_t^m, remaining capacity κ_t^m and the length of their current tour $\tau_{1:t}^m = (v_1^m, \ldots, v_t^m)$, denoted as $dist(\tau_{1:t}^m)$.

Action. A single agent action a_t^m is a tuple (v, p) specifying the next shelf to visit as well as the SKU to pick for agent m. Given s_t, visiting shelf v is a feasible action if it stores items of at least one SKU currently in demand. Furthermore, given the picking location v, the picker may only select an SKU for picking that is both still in demand and available in the current shelf. Note, that the quantity of picked items will be determined heuristically by the transition function Γ in order to decrease the complexity of the action space and facilitate policy learning.

The packing station can always be visited by a picker to unload picked items and thus to restore the capacity. When a picker's capacity is exhausted, visiting the packing station is the only possible action. Moreover, to facilitate agent coordination, a picker may always choose to stay at its current location in order to give other pickers precedence. This way, a hesitant picker may wait and evaluate what the other pickers are doing, before making the next move.

Transition. Given the joint actions $\boldsymbol{a}_t = (a_t^1, \ldots, a_t^m)$ of all agents, the transition function $\Gamma(s_t, \boldsymbol{a}_t)$ deterministically transits to s_{t+1}. The new state consists of the updated agent locations $\boldsymbol{v}_t = (v_t^1, \ldots, v_t^M)$ and agent tours $\tau_{1:t}^m = \tau_{1:t-1}^m \cup \{v_t^m\}$. To update the remaining demand, supply and picker capacities, the pick quantity y_t^m must be determined. Given the pick locations, SKUs and a permutation Ω over pickers, we iteratively determine the pick quantity as the minimum of the remaining demand of the selected SKU p, the storage quantity

at the agent's new location v, as well as the agent's remaining capacity:

$$y_t^{\Omega_k} = \min(\kappa_t^{\Omega_k},\, d_{pt} - \sum_{j=1}^{k-1} y_t^{\Omega_j} \cdot \mathbb{1}_{p_t^{\Omega_j}=p},\, e_{vpt}), \tag{12}$$

where no two agents may select the same shelf-SKU combination in the same decoding step, for which reason the supply e_{vpt} will not be altered by preceding agents. Given the pick quantities y_t^m, the transition function updates the demand $d_{pt+1} = d_{pt} - \sum_{m=1}^{M} y_t^m \cdot \mathbb{1}_{p_t^m=p}$, the supply $e_{vpt+1} = e_{vpt} - \sum_{m=1}^{M} y_t^m \cdot \mathbb{1}_{v_t^m=v;p_t^m=p}$ and the remaining picker capacity $\kappa_{t+1}^m = \kappa_t^m - y_t^m$. The problem instance x is solved once the demand for every SKU is met and all pickers have returned to the packing station they were starting from. A feasible solution to x, reaching the terminal state s_T in T construction steps, will be denoted as $\boldsymbol{a} \coloneqq (\boldsymbol{a}_1, \ldots, \boldsymbol{a}_T)$.

Reward. The MMDP formulation of the min-max MSPRP has a sparse reward function, which is only defined for a complete solution $\boldsymbol{a}$. We define the reward $R(\boldsymbol{a}, x)$ as the negative of the maximum travel distance of any picker, i.e. $R(\boldsymbol{a}, x) = -\max_{m \in \mathcal{M}} dist(\tau_{1:T}^m)$, and the goal of our approach is to maximize it.

4 Method

This section introduces our Multi-Agent Hierarchical Attention Model (MAHAM) – an extension of the Hierarchical Attention Model (HAM) architecture [19] – designed to address the multi-picker min-max variant of the MSPRP. In NCO, the sequential nature of the MDP underlying the CO problem often leads to the adoption of autoregressive (AR) models, which implement a sequential solution generation via an encoder-decoder network, formally represented as:

$$p_\theta(\boldsymbol{a}|x) = \prod_{t=1}^{T} g_\theta(a_t|s_t, H_t) \cdot f_\theta(H_t|s_t) \tag{13}$$

where s_t is the current state of the problem instance x in time step t reached after taking actions $\boldsymbol{a}_{1:t-1}$. The encoder f_θ and decoder g_θ together form the θ-parametrized policy network π_θ which in each step defines a probability distribution over actions conditioned on the current state. Specifically, the encoder network f_θ maps the current state s_t to a hidden representation H_t, which the decoder network g_θ then uses to select the next action a_t.

MAHAM follows the AR solution generation approach. However, the presence of multiple agents and a composite action space $\mathcal{A} \equiv \mathcal{V} \times \mathcal{P}$ introduce special needs which we carefully address with our architecture in Figure 1. While existing approaches tackle multi-agent problems by sequentially generating solutions for one agent after another [21] or using a separate decoder g_{θ^m} per agent [30], MAHAM poses a shared policy which constructs multiple picker routes in parallel through 1.) a separate agent encoder and 2.) a parallel decoding with sequential action selection scheme.

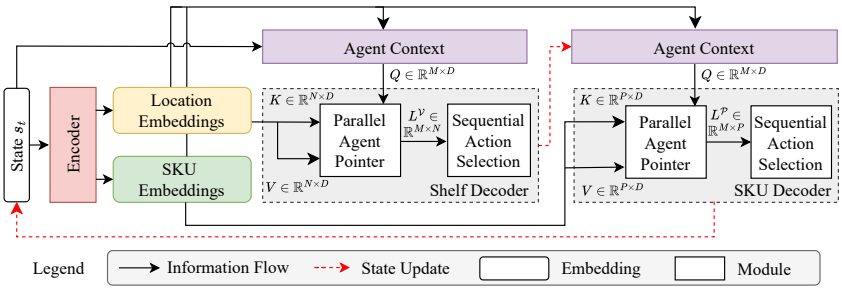


Fig. 1: Overview of the MAHAM Architecture

4.1 Encoder

Problem Encoder. As defined earlier, the min-max MSPRP can be represented as a heterogeneous graph with agents, packing stations, shelves and SKUs posing different node types. We follow [19] and first project these different node-types from their distinct feature spaces into a mutual embedding space of dimensionality D using type-specific transformations W_{ϕ_i} for node i of type ϕ_i. The features used to represent agents, stations, shelves and SKUs in the features space are listed in Table 5 in Appendix B.2.

Also similar to [19], we use several layers of self- and cross-attention between location and SKU nodes. To this end, we treat packing stations as shelves that store zero units for each SKU and concatenate their initial embeddings to those of the shelf nodes, yielding $H^0_{\mathcal{V}} = [H^0_{\mathcal{V}^{\mathrm{D}}} || H^0_{\mathcal{V}^{\mathrm{R}}}]$. Likewise, the initial SKU embeddings are denoted as $H^0_{\mathcal{P}}$. While self-attention is applied independently to shelf and SKU embeddings following the Transformer architecture [23], cross-attention allows shelves and SKUs to influence each other's embeddings. Consequently, shelf embeddings encode information about the SKUs they store, and SKU embeddings reflect their placement within the storage area – an essential property for hierarchical action selection. To perform cross-attention we compute a single matrix of attention scores A using shelf embeddings as queries Q and SKU embeddings as keys K. This contrasts with the MatNet [17] and HAM [19] architectures, which compute separate attention scores for each node type—once as queries and once as keys. Formally we perform:

$$A = \frac{QK^\top}{\sqrt{d_k}}, \qquad Q = W^Q H^{l-1}_{\mathcal{V}}, \qquad K = W^K H^{l-1}_{\mathcal{P}} \tag{14}$$

where W^Q and $W^K \in \mathbb{R}^{d_k \times D}$ are weight matrices learned per attention head[3] and d_k is the per-head embedding dimension. The resulting attention scores $A \in \mathbb{R}^{|\mathcal{V}| \times |\mathcal{P}|}$ can be interpreted as the (learned) influence of an SKU p on the embedding of location v. Similar to MatNet [17] we fuse these learned attention scores with the supply-matrix $E \in \mathbb{R}^{|\mathcal{V}| \times |\mathcal{P}|}$, which specifies how many units of SKU p are stored in location v. To this end, we concatenate the attention score

[3] For succinctness, we omit the layer and head enumeration

and the matrix of storage quantities and feed the resulting score vector through a multi-layer perceptron MLP : $\mathbb{R}^{|\mathcal{V}|\times|\mathcal{P}|\times 2} \to \mathbb{R}^{|\mathcal{V}|\times|\mathcal{P}|}$, with a single hidden layer comprising of D units and GELU activation function [10]. Further, we pass the transpose of the attention scores and of the supply matrix $A^\top$, $E^\top \in \mathbb{R}^{|\mathcal{P}|\times|\mathcal{V}|}$ through a second MLP to obtain the influence $A_{\mathcal{P}\to\mathcal{V}}$ of locations v on the SKU embeddings $H_\mathcal{P}$:

$$A_{\mathcal{V}\to\mathcal{P}} = \mathrm{MLP}_\mathcal{V}\big([A||E]\big), \quad A_{\mathcal{P}\to\mathcal{V}} = \mathrm{MLP}_\mathcal{P}\big([A^\top||E^\top]\big), \tag{15}$$

By avoiding to compute the (computationally expensive) attention scores twice, once to generate shelf embeddings and once for the SKU embeddings, our implementation of the cross-attention mechanism leverages parameter sharing, improving both efficiency and generalization performance, as demonstrated in Section 5. The resulting attention scores are then used to compute the embeddings for the nodes of the respective type:

$$H'_\mathcal{V} = \mathrm{softmax}(A_{\mathcal{V}\to\mathcal{P}})V_\mathcal{P}, \quad V_\mathcal{P} = W^V_\mathcal{P} H^{l-1}_\mathcal{P} \tag{16}$$

$$H'_\mathcal{P} = \mathrm{softmax}(A_{\mathcal{P}\to\mathcal{V}})V_\mathcal{V}, \quad V_\mathcal{V} = W^V_\mathcal{V} H^{l-1}_\mathcal{V} \tag{17}$$

As in [23], $H'_\mathcal{V}$ and $H'_\mathcal{P}$ are then augmented through skip connections, layer normalization, and a feed-forward network, yielding the location and SKU embeddings $H^l_\mathcal{V}$ and $H^l_\mathcal{P}$, respectively, of the current layer l.

Agent Encoder. To account for multiple agents, we introduce an Agent Context Encoder, as illustrated in Figure 2, into our MAHAM architecture. This encoder leverages the embeddings $H_\mathcal{V}$ and $H_\mathcal{P}$ from the problem encoder, along with the current state s_t, to generate embeddings for each picker. To facilitate informed decision-making at each decoding step, the agent embeddings incorporate three key types of information. First, spatial information of pickers is captured by using the embedding of a picker's current location. Further, the remaining capacity and the length of an agent's current tour are included in the agent encoder, helping the model to determine whether to continue the tour or send the picker to a packing station. Lastly, the total demand across all SKUs and the average-pooled SKU embeddings provide insights into the remaining workload. Each context feature is first projected into a shared embedding space of dimensionality D. The resulting embeddings are then concatenated and passed through

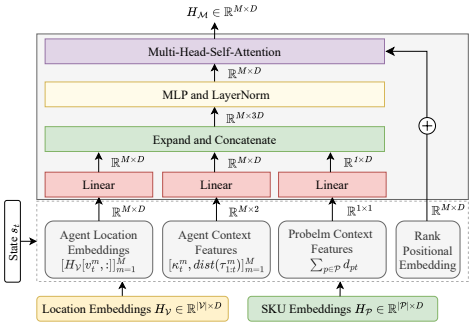


Fig. 2: Agent Context Encoder

an MLP, ensuring that the final representation is mapped back to the original embedding space.

Since coordination between pickers is critical in the min-max MSPRP, we add a Multi-Head-Self-Attention (MHSA) layer [23] at the end of the Agent Context Encoder, which enables message passing between agents. As in [23], we add a positional encoding to the agent embeddings before they enter the MHSA layer. However, given the absence of a natural ordering of pickers, we employ a Ranking-based Position Encoding, where pickers are ranked in descending order of their remaining capacity. This allows the encoder to better prioritize agents based on their current workload, which is crucial for the sequential action selection that will be described in Section 4.3. We denote the final agent embeddings as $H_{\mathcal{M}}$ and the set of all embeddings as $H = (H_{\mathcal{V}}, H_{\mathcal{P}}, H_{\mathcal{M}})$.

4.2 Parallel and Hierarchical Decoder

Given the embeddings for warehouse locations, SKUs, and agents from the encoder, the decoder determines the next location to visit by the pickers as well as the SKUs to be picked there. In contrast to other architectures like [21], our approach generates trajectories for all M simultaneously. This way, the agents can coordinate and balance the workload. If, for example, in step t the tour of picker $m = 1$ is much longer than that of picker $m = 2$, the agents can coordinate that $m = 2$ picks an SKU that is only available in far away shelves. This kind of coordination is not possible in purely autoregressive settings, where agent trajectories are constructed one after another.

For our parallel and hierarchical decoding scheme, we adopt the hierarchical decoder architecture from [19] to sample actions specifying the next locations v_t^m to visit and the SKU p_t^m to pick by agents $m = 1, \dots M$. To this end, we define two decoders $g_\theta^{\mathcal{V}} : \mathcal{S} \to \mathcal{V}$ and $g_\theta^{\mathcal{P}} : \mathcal{S} \to \mathcal{P}$ for action subspaces $\mathcal{V}$ and $\mathcal{P}$, respectively. Moreover, we define a partial transition function $s'_t = \Gamma_\circ(s_t, \boldsymbol{v}_t)$, generating an intermediate state s'_t with updated location information. The decoders can then be used in a hierarchical manner to generate the joint probability of a shelf-SKU pair according to the chain rule of probability [19]:

$$g_\theta(\boldsymbol{a}_t \mid s_t, H) = g_\theta^{\mathcal{V}}(\boldsymbol{v}_t \mid s_t, H) \cdot g_\theta^{\mathcal{P}}(\boldsymbol{p}_t \mid s'_t, H), \tag{18}$$

where $\boldsymbol{v}_t$ and $\boldsymbol{p}_t$ are the joint agent actions for the shelf and SKU sub-action spaces, respectively. MAHAM models the joint agent actions $\boldsymbol{a}_{dt}$ for each sub-action space d and the corresponding decoder g_θ^d as an autoregressive sequence generation process, similar to Equation (13):

$$p_\theta(\boldsymbol{a}_{dt} \mid s_t, H) = g_\theta^d(L_d \mid s_t, H) \cdot \prod_{i=1}^{M} \psi\left(a_{dt}^{\Omega(L)_i} \mid L_d, \boldsymbol{a}_{dt}^{\Omega(L)_{1:i-1}}\right) \tag{19}$$

where $L_d \in \mathbb{R}^{M \times |\mathcal{A}_d|}$ are the unnormalized log-probabilities (henceforth logits) over the joint action space generated by sub-policy g_θ^d and $\Omega(L)$ is a permutation over agents given L_d. Further, ψ is a stochastic function, which autoregressively

selects actions based on the logits L as well as the action sequence $\boldsymbol{a}_{dt}^{\Omega(L)_{1:i-1}}$ of preceding agents. Note, that while the policy p_θ itself acts autoregressively according to the sequential action selection strategy ψ, Equation (19) factors out the computationally expensive calculation of the log-probabilities, which is done in parallel for all agents, allowing an effective and efficient agent coordination and ranking. Therefore, both the shelf and SKU decoder are modifications of the AM decoder proposed by [14], which uses the cross-attention mechanism to generate unnormalized log-probabilities $\boldsymbol{l} \in \mathbb{R}^{|\mathcal{A}|}$. However, in contrast to [14], who use a single context vector as query, our architecture uses the agent embeddings $H_{\mathcal{M}} \in \mathbb{R}^{M \times D}$ in the cross-attention mechanism:

$$Q_d = \text{Attn}(H_{\mathcal{M}} W^Q,\, H_d W^K,\, H_d W^V) \tag{20}$$

$$L_d = C \cdot tanh\left(\frac{Q_d K_d^\top}{\sqrt{D}}\right) \tag{21}$$

where C is a scale parameter, K_d is a projection of the embeddings H_d belonging to the sub-action space d of the decoder. In the following we will show how we can use the logits of the joint action space $L_d \in \mathbb{R}^{M \times |\mathcal{A}_d|}$ to generate feasible actions $\boldsymbol{a}_{dt} = (a_{dt}^1, \ldots, a_{dt}^M)$ for all agents for the current sub-action space d.

4.3 Sequential Action Selection from Joint Logit-Space

Given the logits of the joint action space $L_d \in \mathbb{R}^{M \times |\mathcal{A}_d|}$ for subspace d, we iteratively select actions for each agent using common decoding strategies, such as greedy selection or sampling. To ensure feasibility, joint agent actions are initialized with a set of feasible default actions. For picker locations, the default action is to remain at the current position. Additionally, we introduce a dummy SKU that serves as the default SKU action and can be always selected.

After each agent received a default action, we iteratively refine the agent actions based on the logits L. Therefore, we first mask infeasible actions in L by setting their values to negative infinity. The masked logits are then converted into a single probability distribution over both agents and actions, rather than creating separate distributions per agent. This approach allows the policy to implicitly learn the ranking $\Omega(L)$ by assigning higher logits to agents that should act first. As a result, agents with greater confidence select actions earlier, while less confident agents act later. This step is crucial, as an agent's action can constrain the action space of others, and the order of selection directly affects the picking quantity y^m, as detailed in Section 3.2.

Given the probability distribution over the joint action space, a single action (i.e. agent-action combination) a^m is drawn. As a consequence, all actions of the chosen agent m are marked as infeasible. Further, more actions can be masked based on the actions taken so far using a sub-action specific masking function ξ_d. This way, we avoid that multiple agents select the same shelf- and SKU-combination as required per our MMDP formulation in Section 3.2. In the next iteration, the logits are computed with the updated action mask and the process repeats until no more actions can be selected (i.e., when all actions are marked infeasible). Algorithm 1 formally describes this process.

Algorithm 1 Sequential Action Selection from Joint Logit-Space

Require: Logits $L \in \mathbb{R}^{M \times |\mathcal{A}_d|}$, Binary Action Mask $\mathbf{M} \in \mathbb{R}^{M \times |\mathcal{A}_d|}$, Mask Update Function ξ_d, Temperature β, default action $\boldsymbol{a}'$ (e.g. $\boldsymbol{a}' \equiv \boldsymbol{a}_{t-1}$)
Ensure: Feasible agent actions $\boldsymbol{a} \in \mathbb{N}^M$ and agent permutation Ω

$\boldsymbol{a} \leftarrow \boldsymbol{a}'$ {Initialize agent actions}
$i \leftarrow 1$
while not all elements in $\mathbf{M}$ are 1 **do**
  $L' \leftarrow L - \mathbf{M} * \infty$ {Mask infeasible actions}
  $P_{ma} \leftarrow \frac{\exp(L'_{ma}/\beta)}{\sum_{i \in \mathcal{M}} \sum_{j \in \mathcal{A}_d} \exp(L'_{ij}/\beta)} \quad \forall\, m \in \mathcal{M}, a \in \mathcal{A}_d$ {Normalize}
  $a^m \sim \text{Categorical}(P)$ {Sample a single agent's action}
  $\Omega_i = m$ {Set m to be the ith agent in the permutation}
  $\boldsymbol{a}[m] = a^m$ {Update vector of actions}
  $\mathbf{M}[m, :] = 1$ {Mark all actions of agent m infeasible}
  $\mathbf{M} \leftarrow \xi_d(\mathbf{M}, \boldsymbol{a})$ {Update Mask according to subspace specific logic}
  $i \leftarrow i + 1$ {Increment step counter}
end while

4.4 Learning Method

During training, the objective is to adjust the parameters θ of the policy p_θ to maximize the reward $R(\boldsymbol{a}, x)$ for any given problem instance x. Formally, we can cast the optimization problem as follows:

$$\theta^* = \underset{\theta}{\operatorname{argmax}} \Big[\mathbb{E}_{x \sim P(x)} \big[\mathbb{E}_{\boldsymbol{a} \sim p_\theta(\boldsymbol{a}|x)} R(\boldsymbol{a}, x) \big] \Big]. \tag{22}$$

Due to the absence of large datasets containing the optimal solutions $\boldsymbol{a}$ for CO problem instances x, several Reinforcement Learning techniques have been developed to train neural CO solvers [15,16,13,11]. However, recently, self-supervised approaches have emerged in the realm of neural combinatorial optimization and already achieve state-of-the-art results on some CO problems [20,6].

A major advantage over REINFORCE-based learning is that single actions instead of entire trajectories can serve as training examples. While REINFORCE prohibits a re-encoding of the problem state after each decoding step due to the accumulation of gradient information during training, the use of single actions or sub-trajectories in self-supervised learning allows for stepwise encoding [20]. While this might impose unnecessary computational cost for static problems like the TSP, it is a strong benefit for a highly dynamic problem like the MSPRP, where after each step the demand, supply and capacity change. Therefore, in this work, we adopt the self-improvement approach described in [20]. This method samples $\alpha \gg 1$ candidate solutions for an instance x from the current best-known policy p_{θ^*} and selects the best one, $\boldsymbol{a}^* \coloneqq \operatorname{argmax}\{R(\boldsymbol{a}^1, x), \ldots, R(\boldsymbol{a}^\alpha, x)\}$, as a training example. Then, cross-entropy loss $\mathcal{L}_{\text{CE}} = -\sum_{t=1}^{T} \log p_\theta(a_t^* | s_t)$ is used to train the model on these pseudo-optimal solutions. The refined model is used

in the next iteration to generate new candidate solutions, leading to progressively better training examples as training advances.[4]

In order to apply self-improvement to learn the parameters of MAHAM, we first revise the autoregressive policy of Equation (13) and extend it with the components introduced by our MAHAM architecture:

$$p_\theta(\boldsymbol{a}|x) = \prod_{t=1}^{T} f_\theta(H|s_t) \cdot \prod_{d=\{\mathcal{V},\mathcal{P}\}} g_\theta^d(L_d|s_t, H) \cdot \prod_{m=1}^{M} \psi(a_{dt}^m | L_d, \boldsymbol{a}_{dt}^{1:m-1}), \tag{23}$$

where the encoder is factorized over the action sub-spaces $|\mathcal{V}|$ and $|\mathcal{P}|$, which both use the same encoder embeddings, and the decoder produces logits L_d only once for all M agents, allowing MAHAM to efficiently model dependencies in multi-agent decision-making. Resulting from this, we train the model with cross entropy loss via gradient descent using the following definition of the gradients:

$$\nabla_\theta \mathcal{L} = -\sum_{t=1}^{T} \sum_{d=\{\mathcal{V},\mathcal{P}\}} \sum_{m=1}^{M} \nabla_\theta \log\, p_\theta(a_{dt}^m | \boldsymbol{a}_{dt}^{1:m-1}, s_t) \tag{24}$$

5 Experiments

We study the effectiveness of MAHAM in solving the min-max MSPRP by comparing it with both traditional OR solvers as well as other multi-agent neural solvers. First, we use the exact solver Gurobi with two different time budgets (10 minutes and one hour) to solve a single instance from the test set. Further, due to the absence of (meta-)heuristics for the min-max variant of the MSPRP, we implement a greedy heuristic as a simple baseline. To compare MAHAM with other learning-based methods, we include HAM [19], 2d-Ptr [18], Equity Transformer [21], and PARCO [2] in the experiments. We describe all baseline solvers in Appendix A.

5.1 Comparison with Baselines

We present the main empirical results, comparing MAHAM against all baselines mentioned above, in Table 2, reporting the average objective function values (Obj.), gaps to the best-known solutions, and inference times for solving a single instance from the test set of the respective instance type. For training and evaluating MAHAM, we use the same instance types and instance generation method as described in [19]. Specifically, we use three different warehouse layouts, with 10, 25, and 40 shelves and vary the number of SKUs per layout type. We describe the generation of instances in detail in Appendix B.1. For neural baselines, we evaluate the performance using 1280 sampled solutions and reporting the objective value of the best one.

[4] A detailed description of the algorithm is given in Algorithm 2

Table 2: Comparison of MAHAM with baseline solvers.

MSPRP10 ($\|\mathcal{V}\| = 10$)									
$\|\mathcal{P}\|$	3			6			9		
Metric	Obj.	Gap	Time	Obj.	Gap	Time	Obj.	Gap	Time
Gurobi (10m)	**1.1675**	0.0%	28.2s	**1.6866**	0.0%	381.5s	**1.6187**	0.0%	249.5s
Gurobi (1h)	**1.1675**	0.0%	28.2s	**1.6866**	0.0%	381.5s	**1.6187**	0.0%	249.5s
Greedy	1.2536	7.37%	**0.10s**	1.7853	5.85%	**0.23s**	1.7311	6.94%	**0.24s**
HAM	1.1678	0.03%	0.32s	1.7089	1.32%	0.37s	1.6426	1.48%	0.44s
Equity Trans.	1.1678	0.03%	0,30s	1.6903	0.22%	0.35s	1.6351	1.01%	0.43s
2d-Ptr	**1.1675**	0.00%	0,32s	1.6967	0.60%	0.38s	1.6371	1.14%	0.46s
PARCO	**1.1675**	0.00%	0,25s	1.6888	0.13%	0.31s	1.6237	0.31%	0.30s
MAHAM	**1.1675**	0.0%	0,21s	1.6867	0.01%	0.25s	**1.6187**	0.0%	0.27s
MSPRP25 ($\|\mathcal{V}\| = 25$)									
$\|\mathcal{P}\|$	12			15			18		
Metric	Obj.	Gap	Time	Obj.	Gap	Time	Obj.	Gap	Time
Gurobi (10m)	1.7608	1.42%	600s	1.8402	2.87%	600s	1.8929	4.22%	600s
Gurobi (1h)	1.7395	0.19%	3512s	1.7915	0.15%	3600s	1.8301	0.77%	3600s
Greedy	3.3079	90.53%	0.50s	2.9636	65.68%	0.52s	3.4936	92.36%	0.57s
HAM	1.7813	2.60%	0.89s	1.8685	4.46%	1.13s	1.8954	4.36%	1.12s
Equity Trans.	1.7750	2.23%	0.79s	1.8328	2.46%	1.12s	1.8573	2.26%	1.11s
2d-Ptr	1.7508	0.84%	1.08s	1.8332	2.48%	1.13s	1.8681	2.86%	1.15s
PARCO	1.7447	0.49%	0.56s	1.8014	0.70%	0.49s	1.8282	0.66%	0.51s
MAHAM	**1.7362**	0.00%	**0.45s**	**1.7888**	0.00%	**0.46s**	**1.8162**	0.00%	**0.47s**
MSPRP40 ($\|\mathcal{V}\| = 40$)									
$\|\mathcal{P}\|$	15			20			30		
Metric	Obj.	Gap	Time	Obj.	Gap	Time	Obj.	Gap	Time
Gurobi (10m)	1.9163	17.26%	600s	2.1907	12.00%	600s	2.3398	32.99%	600s
Gurobi (1h)	1.7552	7.40%	3600s	2.0201	3.28%	3600s	1.8699	6.28%	3600s
Greedy	4.1010	150.93%	0.63s	5.1602	163.81%	1.02s	4.0420	129.74%	1.12s
HAM	1.7256	5.59%	1.31s	2.1334	9.07%	1.71s	1.9211	9.19%	3.66s
Equity Trans.	1.6985	3.93%	1.60s	2.0373	4.16%	2.16s	1.8355	4.33%	3.77s
2d-Ptr	1.6857	3.15%	1.65s	2.0245	3.50%	2.16s	1.8232	3.63%	2.90s
PARCO	1.6452	0.67%	0.72s	1.9760	1.02%	0.79s	1.7896	1.72%	1.16s
MAHAM	**1.6343**	0.00%	**0.54s**	**1.9560**	0.00%	**0.66s**	**1.7594**	0.00%	**0.92s**

Table 3: Large-scale generalization for unseen numbers of locations and SKUs

MSPRP50 ($\|\mathcal{V}\| = 50$)									
$\|\mathcal{P}\|$	100			250			500		
Method	Obj.	Gap	Time	Obj.	Gap	Time	Obj.	Gap	Time
Greedy	4.9638	122%	1.75s	5.7146	85%	6.22s	6.0632	51%	28.77s
2d-Ptr	3.9799	78%	5.56s	5.9040	91%	19.62s	6.6315	65%	61.85s
PARCO	3.9412	76%	3.98s	5.1629	67%	10.70s	5.2790	32%	27.20s
MAHAM w/o PS	2.3865	7%	3.68s	3.1643	2%	8.32s	4.1617	4%	17.55s
MAHAM	**2.2352**	0%	3.55s	**3.0916**	0%	7.79s	**4.0128**	0%	15.10s

MAHAM consistently outperforms other neural baselines in terms of solution quality and speed, with margins growing with the size of the problem instance. Also, MAHAM is on-par with the Gurobi solver on small instances and even outperforms it on larger instances, where no optimal solutions were found in the given time bounds.

5.2 Large Scale Generalization

We further evaluate the generalization performance of MAHAM on large-scale instances of the MSPRP that were not seen during training. The ability to generalize to larger instances is crucial for any NCO algorithm to make it applicable to dynamic real-world scenarios. We evaluate MAHAM against a purely autoregressive approach (2d-Ptr), PARCO as an alternative parallel decoding model, and the Greedy heuristic. Gurobi is not included in the evaluation as it can not

find solutions to any of the large instances with a time budget of one hour per instance. The results are shown in Table 3, where MAHAM consistently outperforms other methods while also being significantly faster. Most notable is the large performance gap compared to PARCO, which achieves competitive results in in-distribution testing, but seems to generalize worse to larger instances.

5.3 Ablation Studies

Picker Ranking and Coordination: A key aspect of MAHAM is the Sequential Action Selection. The quality of the solution generated by the autoregressive policy defined in Equation (19) and the action selection function ψ defined by Algorithm 1 strongly depends on the order $\Omega(L)$ in which pickers perform actions. In order to validate that the model is able to learn good agent rankings, i.e., by assigning higher logits to those agents that should have priority, we compare the (*learned*) agent priority of Equation (19) with a model that iterates over the set of agents in the order of their index m (*index*) as well as a model that determines the order Ω randomly (*random*).

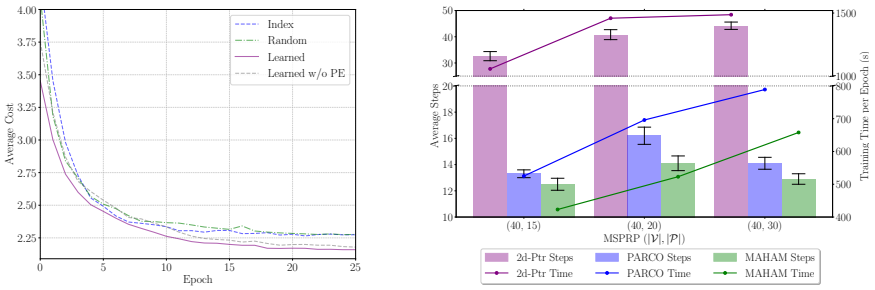
Moreover, MAHAM utilizes a separate agent encoder that enables effective agent coordination when computing the joint action logits in the decoders. The idea of utilizing an agent encoder for multi-agent problems itself is not novel, but has already been applied in the Equity Transformer [21], the 2d-Ptr [18], and PARCO [2]. However, in this work we fuse the agent encoder with a novel rank-dependent positional encoding followed by a multi-head self-attention layer. This enables effective communication between the agents based on their current utilization and ultimately enables the model to come up with optimal rankings.

Figure 3a summarizes the results of an ablation study testing the effectiveness of the proposed components in our MAHAM architecture. The full model with learned rankings and rank-dependent positional encodings (PE) performs significantly better than the models relying on an index-based or random order, and also achieves better solutions than MAHAM without the positional encoding.

Encoder Parameter Sharing: MAHAM introduces an efficient way to incorporate message-passing over different types of nodes in a heterogeneous graph. Through the parameter sharing (PS) approach described in Section 4.1, the MAHAM encoder saves roughly 20% in size, allowing it to process larger instances faster. In addition, parameter sharing acts as regularization, improving the generalization of the trained model. We compare MAHAM with and without parameter sharing in the cross-attention layer of the encoder on out-of-distribution instances in table 3. Parameter sharing consistently results in better solutions in less time.

5.4 Runtime Comparison

We study the efficiency of MAHAM by comparing it to the 2d-Ptr – acting as a purely autoregressive neural baseline – and PARCO, another parallel solution construction approach. The results are shown in Figure 3b. While the 2d-Ptr requires much more decoding steps to construct a solution, resulting in longer training times, MAHAM also needs less construction steps and is quicker



(a) (b)

Fig. 3: Solution quality of MAHAM on MSPRP instances for different ranking strategies (left) and MAHAM efficiency in comparison to the 2d-Ptr and PARCO (right)

to train than PARCO. This can be attributed to our Sequential Action Selection approach, which effectively avoids conflicts through adaptive masking. In PARCO on the other hand, agents may select the same shelf/SKU in the same decoding step, resulting in a conflict and consequently in one or more agents doing nothing in the respective stage.

6 Conclusion and Future Work

In this work, we introduced the first neural solver for the min-max Mixed-Shelves Picker Routing Problem. The core of our approach is the integration of a hierarchical and parallel decoding mechanism capable of efficiently constructing solutions over complex, multi-dimensional action spaces, such as those found in min-max MSPRP. While previous methods relied on sequential solution construction or parallel decision-making prone to conflicts, our approach achieves efficient and effective agent coordination, enabled by a novel Sequential Action Selection algorithm.

Our extensive experimental results, including traditional as well as neural solvers, demonstrate the superiority of MAHAM in both solution quality and inference speed, particularly for large-scale problem instances. These findings highlight the capabilities of neural solvers and prove them as a strong alternative to hand-crafted heuristics.

Future research directions include extending this approach to more dynamic warehouse environments with real-time demand fluctuations and exploring hybrid methods that integrate learning-based techniques with optimization heuristics for further performance improvements. Additionally, our framework could be adapted to other multi-agent combinatorial optimization problems beyond warehouse logistics, such as fleet routing and robotic task allocation.

References

1. Ba, J.L., Kiros, J.R., Hinton, G.E.: Layer normalization (2016), https://arxiv.org/abs/1607.06450
2. Berto, F., Hua, C., Luttmann, L., Son, J., Park, J., Ahn, K., Kwon, C., Xie, L., Park, J.: Parco: Learning parallel autoregressive policies for efficient multi-agent combinatorial optimization. arXiv preprint arXiv:2409.03811 (2024)
3. Berto, F., Hua, C., Park, J., Luttmann, L., Ma, Y., Bu, F., Wang, J., Ye, H., Kim, M., Choi, S., et al.: Rl4co: an extensive reinforcement learning for combinatorial optimization benchmark. arXiv preprint arXiv:2306.17100 (2023)
4. Boysen, N., Briskorn, D., Emde, S.: Parts-to-picker based order processing in a rack-moving mobile robots environment. European Journal of Operational Research **262**(2), 550–562 (2017). https://doi.org/10.1016/j.ejor.2017.03.053
5. Boysen, N., De Koster, R., Weidinger, F.: Warehousing in the e-commerce era: A survey. European Journal of Operational Research **277**(2), 396–411 (2019)
6. Corsini, A., Porrello, A., Calderara, S., Dell'Amico, M.: Self-labeling the job shop scheduling problem. arXiv preprint arXiv:2401.11849 (2024)
7. Daniels, R.L., Rummel, J.L., Schantz, R.: A model for warehouse order picking. European Journal of Operational Research **105**(1), 1–17 (Feb 1998). https://doi.org/10.1016/S0377-2217(97)00043-X
8. De Koster, R., Le-Duc, T., Roodbergen, K.J.: Design and control of warehouse order picking: A literature review. European Journal of Operational Research **182**(2), 481–501 (2007). https://doi.org/10.1016/j.ejor.2006.07.009
9. Gurobi Optimization, LLC: Gurobi Optimizer Reference Manual (2024), https://www.gurobi.com
10. Hendrycks, D., Gimpel, K.: Gaussian error linear units (gelus). arXiv preprint arXiv:1606.08415 (2016)
11. Hottung, A., Mahajan, M., Tierney, K.: Polynet: Learning diverse solution strategies for neural combinatorial optimization. arXiv preprint arXiv:2402.14048 (2024)
12. Hua, C., Berto, F., Son, J., Kang, S., Kwon, C., Park, J.: Camp: Collaborative attention model with profiles for vehicle routing problems. arXiv preprint arXiv:2501.02977 (2025)
13. Kim, M., Park, J., Park, J.: Sym-nco: Leveraging symmetricity for neural combinatorial optimization. In: Advances in Neural Information Processing Systems (2022)
14. Kool, W., van Hoof, H., Welling, M.: Attention, learn to solve routing problems! In: International Conference on Learning Representations (2019), https://openreview.net/forum?id=ByxBFsRqYm
15. Kool, W., van Hoof, H., Welling, M.: Buy 4 REINFORCE Samples, Get a Baseline for Free! ICLR Workshops (Apr 2019)
16. Kwon, Y.D., Choo, J., Kim, B., Yoon, I., Gwon, Y., Min, S.: Pomo: Policy optimization with multiple optima for reinforcement learning. Advances in Neural Information Processing Systems **33**, 21188–21198 (2020)
17. Kwon, Y.D., Choo, J., Yoon, I., Park, M., Park, D., Gwon, Y.: Matrix encoding networks for neural combinatorial optimization. In: Advances in Neural Information Processing Systems. vol. 34, pp. 5138–5149 (2021)
18. Liu, Q., Liu, C., Niu, S., Long, C., Zhang, J., Xu, M.: 2d-ptr: 2d array pointer network for solving the heterogeneous capacitated vehicle routing problem. In: Proceedings of the 23rd International Conference on Autonomous Agents and Multiagent Systems. pp. 1238–1246 (2024)

19. Luttmann, L., Xie, L.: Neural combinatorial optimization on heterogeneous graphs: An application to the picker routing problem in mixed-shelves warehouses. In: Proceedings of the International Conference on Automated Planning and Scheduling. vol. 34, pp. 351–359 (2024)
20. Pirnay, J., Grimm, D.G.: Self-improvement for neural combinatorial optimization: Sample without replacement, but improvement. Transactions on Machine Learning Research (2024)
21. Son, J., Kim, M., Choi, S., Kim, H., Park, J.: Equity-transformer: Solving np-hard min-max routing problems as sequential generation with equity context. In: Proceedings of the AAAI Conference on Artificial Intelligence. vol. 38, pp. 20265–20273 (2024)
22. Tompkins, J.A.: Facilities planning. John Wiley & Sons, 4th edn. (2010)
23. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, Ł., Polosukhin, I.: Attention is All you Need. In: Advances in Neural Information Processing Systems. vol. 30. Curran Associates, Inc. (2017)
24. Vinyals, O., Fortunato, M., Jaitly, N.: Pointer networks. Advances in neural information processing systems **28** (2015)
25. Weidinger, F.: Picker routing in rectangular mixed shelves warehouses. Computers & Operations Research **95**, 139–150 (Jul 2018). https://doi.org/10.1016/j.cor.2018.03.012
26. Weidinger, F., Boysen, N., Schneider, M.: Picker routing in the mixed-shelves warehouses of e-commerce retailers. European Journal of Operational Research **274**(2), 501–515 (Apr 2019). https://doi.org/10.1016/j.ejor.2018.10.021
27. Xie, L., Li, H., Luttmann, L.: Formulating and solving integrated order batching and routing in multi-depot agv-assisted mixed-shelves warehouses. European Journal of Operational Research **307**(2), 713–730 (2023)
28. Xie, L., Thieme, N., Krenzler, R., Li, H.: Introducing split orders and optimizing operational policies in robotic mobile fulfillment systems. European Journal of Operational Research **288**(1), 80–97 (2021)
29. Zheng, Z., Yao, S., Wang, Z., Tong, X., Yuan, M., Tang, K.: Dpn: Decoupling partition and navigation for neural solvers of min-max vehicle routing problems. arXiv preprint arXiv:2405.17272 (2024)
30. Zong, Z., Zheng, M., Li, Y., Jin, D.: Mapdp: Cooperative multi-agent reinforcement learning to solve pickup and delivery problems. In: Proceedings of the AAAI conference on artificial intelligence. vol. 36, pp. 9980–9988 (2022)

A Baselines

Gurobi [9]. We implement the mathematical model described in **??** in the exact solver Gurobi [9] and provide a time budget of 600 and 3600 seconds per test instance. We run the Gurobi solver with activated multi-threading on a machine equipped with two Intel Xeon E5-2690 v4 processors, totaling 28 physical cores and 56 logical threads.

Greedy. Due to the absence of heuristics developed for the min-max MSPRP, we develop a greedy heuristic as a simple baseline. The heuristic constructs solutions sequentially by assigning each agent logits for selecting a shelf, weighted inversely by its distance from the agent's current position. Similarly, SKUs are chosen with logits proportional to the number of units an agent could potentially pick. Given the logits, the same sequential action selection as described in Algorithm 1 is used to generate actions for all agent. Being a stochastic heuristic, we use it to generate 100 different solutions for each test instance and select the best one.

Hierarchical Attention Model [19]. The Hierarchical Attention Model (HAM) introduces the idea of a hierarchical decoder to generate actions over the decomposed action space of the MDP formulation of the MSPRP. Although HAM was developed to solve the min-sum MSPRP, creating $\mathcal{B}$ one after another, it can be used to solve the min-max MSPRP as well thanks to our assumption, that there are exactly as many pickers as there are tours. In this work, HAM is trained like all other models on the min-max-based reward defined in Section 3.2 using the learning method outlined in Section 4.4.

2d-Ptr [18]. The 2D Array Pointer network (2d-Ptr) addresses the heterogeneous capacitated vehicle routing problem (HCVRP) by using a dual-encoder setup to map vehicles and customer nodes effectively. This approach facilitates dynamic, real-time decision-making for route optimization. Its decoder employs a 2D array pointer for action selection, prioritizing actions over vehicles. The 2d-Ptr can be adapted to solve the min-max MSPRP by using the 2D pointer hierarchically to select shelves and SKUs and by using pickers instead of vehicles.

Equity Transformer (ET) [21]. The Equity-Transformer (ET) approach [21] addresses min-max routing problems by employing a sequential planning approach with sequence generators like the Transformer. It focuses on equitable workload distribution among multiple agents, applying this strategy to challenges like the min-max multi-agent traveling salesman and pickup and delivery problems. In our experiments, we modify the agent context in the decoder to the MSPRP setting

PARCO [2]. PARCO is a recent NCO framework for solving multi-agent CO problems. It uses a multi-pointer mechanism paired with a conflict handler to generate solutions for multiple agents in parallel. It is a versatile framework, which has been applied to different routing and scheduling problems.

B Model And Training Configuration

In the following, we detail the model and training parameters as well as the parameters for generating the training / test data. Besides that, to ensure proper reproducibility we provide all training details in our publicly available GitHub repository as configuration files.

B.1 Instance Generation

For training and evaluating MAHAM and the baselines described above, we use the same instance generation scheme described in [19], who generate instances for three warehouse types that differ in the number of available shelves. They generate instances with 10, 25 and 40 shelves referred to as MSPRP10, MSPRP25 and MSPRP40, respectively. While the number of shelves is fixed, the number of demanded SKUs is altered for each warehouse type.

We randomly select the $|\mathcal{V}^{\mathrm{S}}|$ storage locations from all $|\mathcal{P}| \times |\mathcal{V}^{\mathrm{R}}|$ possible SKU-shelf combinations and sample the supply from a discrete uniform distribution with mean $\bar{n}_i$. Likewise, the demand for each SKU is sampled from a discrete uniform distribution with mean $\bar{d}_p$. Lastly, we clip the demand of an SKU by the warehouse's total supply for it in order to ensure the feasibility of all generated instances. Table 4 summarizes the parameters of the different instances.

Table 4: Parameter values for instance generation

	MSPRP10			MSPRP25			MSPRP40			MSPRP50		
$\lvert\mathcal{V}^{\mathrm{R}}\rvert$	10	10	10	25	25	25	40	40	40	50	50	50
$\lvert\mathcal{V}^{\mathrm{S}}\rvert$	20	20	20	50	50	50	100	100	100	200	500	1000
$\lvert\mathcal{P}\rvert$	3	6	9	12	15	18	15	20	30	100	250	500
κ	6	9	9	12	12	15	12	15	15	15	15	15

B.2 Network Hyperparameters

To ensure valid and meaningful experiments, the hyperparameters are identical for all models. The size of the embeddings is set to 256 and the number of heads for multi-head attention mechanisms is set to 8. All models use $L = 4$ encoder layers, GELU activation functions [10], and Layer Normalization [1]. To map the different entities of the MSPRP into embedding space, all models use the same features outlined in Table 5.

Table 5: Features to describe the different entities in the min-max MSPRP

ϕ	Description
Station	Cartesian Coordinates of the packing station
	Amount of items to be commissioned at the station
	Number of agents belonging to the station
Shelf	Cartesian Coordinates of the shelf
	Number of different SKUs stored in shelf i: $n_i = \lvert\{p \in \mathcal{P} \mid E_{ip} > 0\}\rvert$
	The average supply for SKUs stored in shelf i: $\bar{e}_i = (\sum_{p \in \mathcal{P}} E_{ip})/n_i$
SKU	Demand d_p of SKU
	Number of shelves the SKU p is available in: $n_p = \lvert\{p \in \mathcal{P} \mid E_{ip} > 0\}\rvert$
	The average storage quantity of SKU p: $\bar{e}_p = (\sum_{i \in \mathcal{V}} E_{ip})/n_p$
Agent	Remaining capacity of picker κ_t^m
	Length of the picker's current tour $dist(\tau_{1:t}^m)$
	The total remaining demand of all SKUs $\sum_{p \in \mathcal{P}} d_p$
	The embedding of the agents current location $\boldsymbol{h}_{\mathcal{V}}^m = H_{\mathcal{V}}[v^m]$

B.3 Training Hyperparameters

We train all models using the self-improvement method described by [20]. To ensure consistency, we use identical hyperparameters and training environments for all neural baselines described in Appendix A. All models are trained on a single NVIDIA A100 GPU with 40GB of VRAM. Training spans 50 epochs, with each epoch generating $N = 5,000$ independent instances. For each instance, $\alpha = 100$ candidate solutions are sampled from the reference-policy π_{best}, and the best solution is added to the training dataset. After all instances are solved by the reference policy π_{best}, we draw training samples in mini-batches of $B = 2.000$ and determine the cross-entropy loss for the pseudo-optimal actions with respect to the target-policy π. Adam optimizer with a learning rate of 0.0001 is used to update the parameters of the target-policy, and the trainer class from the RL4CO [3] library is used to guide the learning process.

The validation dataset consists of 10,000 independently generated instances per epoch. If the target policy outperforms the reference policy on the validation set, the reference policy is updated, and the training dataset is reset. Algorithm 2 provides a detailed breakdown of these steps.

Algorithm 2 Self-improvement training for neural CO

Require: $\mathcal{X}$: distribution over problem instances; f_*: objective function
Require: N: number of instances to sample in each epoch
Require: α: number of sequences to sample for each instance
Require: $\textsc{Validation} \sim \mathcal{X}$: validation dataset

Randomly initialize policy π_θ
$\pi_{\text{best}} \leftarrow \pi_\theta$
$\textsc{Dataset} \leftarrow \emptyset$
for epoch **do**
  Sample set of n problem instances $\textsc{Instances} \sim \mathcal{X}$
  for each $x \in \textsc{Instances}$ **do**
    // Sample set of m feasible solutions
    $A := \{\boldsymbol{a}_{1:T}^{(1)}, \ldots, \boldsymbol{a}_{1:T}^{(m)}\} \sim \pi_{\text{best}}$
    // Add best solution to training dataset
    $\textsc{Dataset} \leftarrow \textsc{Dataset} \cup \{(x, \arg\max_{\boldsymbol{a}_{1:T} \in A} f_x(\boldsymbol{a}_{1:T}))\}$
  end for
  for batch **do**
    // Sample B instances and partial solutions from $\textsc{Dataset}$
    $\{(x_j, \boldsymbol{a}_{1:d_j}^{(j)})\}_{j=1}^{B} \sim \textsc{Dataset}, \qquad \{d_j\}_{j=1}^{B} \sim \mathcal{U}(1, T-1)$
    // Minimize batch-wise cross entropy loss
    $\mathcal{L}_\theta = -\frac{1}{B} \sum_{j=1}^{B} \log \pi_\theta \left(a_{d_{j+1}}^{(j)} | \boldsymbol{a}_{1:d_j}^{(j)} \right)$
  end for
  if greedy performance of π_θ on $\textsc{Validation}$ better than π_{best} **then**
    // update best policy
    $\pi_{\text{best}} \leftarrow \pi_\theta$
    // Empty Training Dataset
    $\textsc{Dataset} \leftarrow \emptyset$
  end if
end for