# Dynamic Shelf Arrangement and Task Assignment with Stable Matching for Multi-Agent Pickup and Delivery with Multi-Item Packing Problem

**Yosuke Fujisawa[1], Yusaku Wakasugi[2], Kazuya Nakazawa[2], Ryo Matsubara[2], Toshiharu Sugawara[1]**

[1]Department of Computer Science and Communications Engineering
Waseda University, Tokyo 1698555, Japan
[2]R&D Division, Panasonic Connect Co., Ltd. Osaka 5408553, Japan
y.fujisawa@isl.cs.waseda.ac.jp, sugawara@waseda.jp
{wakasugi.yusaku, nakazawa.kazuya, matsubara.ryo}@jp.panasonic.com

## Abstract

The *multi-agent pickup and delivery* (MAPD) problem requires not only planning collision-free paths for carrier agents to carry items while avoiding obstacles but also optimizing task allocation among agents, and it has become a key problem in automated warehouse environments. Most existing studies rely on simplified models that fail to capture the constraints of real-world warehousing. We investigate a more realistic online variant, *MAPD with multi-item packing problem* (MAPD-MP), where agents process *picking lists* of multiple tasks, each requiring an agent to carry a shelf containing items to a picking station and return it to the original location. We incorporate the fact that each shelf has a distinct probability of being requested. To solve this problem, we integrated conventional planning methods with three components: *shelf arrangement optimization*, which optimizes shelf arrangement based on probabilistic demand; *task assignment using the Gale–Shapley algorithm*, which uses stable matching for task allocation; and *holding-node integration*, in which shelves are temporarily placed at holding nodes for subsequent tasks. Our experiments demonstrate that our method improves cooperative efficiency in large-scale agent teams and increases the picking-list processing throughput.

## Introduction

In the *multi-agent path finding* (MAPF) problem, multiple agents in an environment plan collision-free paths to their own destinations while avoiding obstacles and other agents (Stern et al. 2019; Atzmon et al. 2020). Agents must coordinate their path planning with other agents to efficiently complete the overall task. The *multi-agent pickup and delivery* (MAPD) problem extends MAPF to a *lifelong* setting with task allocation (Ma et al. 2017; Matsui 2024), where agents carry items from pickup to delivery locations. MAPF and MAPD have gained attention in applications such as automated warehouses (Li et al. 2021), construction material transport (Miyashita, Yamauchi, and Sugawara 2023), and autonomous driving (Yan, Zheng, and Wu 2024). This study focused on MAPD for automated warehouse operations.



Although substantial progress has been made in MAPD in warehouse domains, most research relies on assumptions that ignore key operational details of the problem. One such assumption is that agents directly carry goods, whereas in some modern warehouses, goods are stored on mobile shelves, and agents carry the shelves to their destinations (Wurman, D'Andrea, and Mountz 2008). The available paths differ depending on whether agents carry shelves/racks. In particular, when an agent is carrying a shelf, it cannot share the same node with other shelves, which introduces additional collision avoidance constraints and requires priority setting based on path freedom to optimize efficiency. Recent studies have addressed variants of shelf-transporting MAPD (Li and Ma 2023; Makino, Ohama, and Ito 2024; Fujisawa et al. 2025) to capture scenarios requiring the concurrent delivery of multiple shelves in a picking list of customer-ordered multiple items to a designated station for packing them, such as the *multi-agent pick-and-delivery with multi-item packing* (MAPD-MP) problem, where agents carry shelves to picking stations where e-commerce goods are packed and return them to their original positions (Fujisawa et al. 2025).

To maximize picking list *throughput*, which is the number of completed tasks in picking lists per time step, under this realistic assumption, we propose an enhanced MAPD-MP solver that integrates three complementary components by utilizing the existing planning methods such as *prioritized path planning with carrying states* (3PCS) (Fujisawa et al. 2025): *shelf arrangement optimization* (SAO) dynamically rearranges shelf positions based on selection probabilities to reduce agents' travel distances; *task assignment via the Gale–Shapley algorithm* (TAGS) casts agent–task allocation as a stable matching problem, where the Gale–Shapley algorithm is used to mitigate imbalance in task-processing time across agents; and *holding node integration* (HNI) proactively moves shelves that are likely to appear in upcoming tasks to dedicated holding nodes, smoothing the arrival time variability among picking lists. These components integrate seamlessly with existing MAPD and MAPD-MP solvers.

Finally, we experimentally show that our approach facilitates effective multi-agent cooperation and significantly improves picking list throughput in warehouse-inspired environments, using tasks generated by Zipf distribu-

tions (Øverby and Audestad 2021), which are often observed in real-world automated warehouses, especially those serving e-commerce, expressing a highly skewed demand as a long-tail phenomenon (Brynjolfsson, Hu, and Smith 2010; Rahman and Kirby 2024).

## Related Work

Many approaches have been proposed for the MAPF/MAPD. For MAPF, centralized search methods, such as *conflict-based search* (CBS) (Sharon et al. 2015) and *lazy-CBS* (Gange, Harabor, and Stuckey 2019), obtain high-quality collision-free paths; however, their computational cost increases exponentially with the number of agents. Decoupled approaches, such as *cooperative A** (Silver 2005) and its extensions (Wu, Bhattacharya, and Prorok 2020), offer better scalability by allowing agents to plan independently, although they may produce longer paths. To address this trade-off between scalability and quality, meta-algorithmic approaches for MAPF have gained attention, with methods such as LNS2 (Li et al. 2022) and LaCAM* (Okumura 2024) being proposed.

Several efficient decentralized planning methods have been proposed for the lifelong MAPF and MAPD problems. The *token passing* (TP) algorithm (Ma et al. 2017) enables agents to plan via cooperative A* using a shared token and guarantees completeness for certain MAPD instances. *Priority inheritance with backtracking* (PIBT) improves scalability by allowing agents to move based on local interactions without collisions (Okumura et al. 2022), and its extensions achieve a balance between efficiency and scalability (Shimada, Miyashita, and Sugawara 2025).

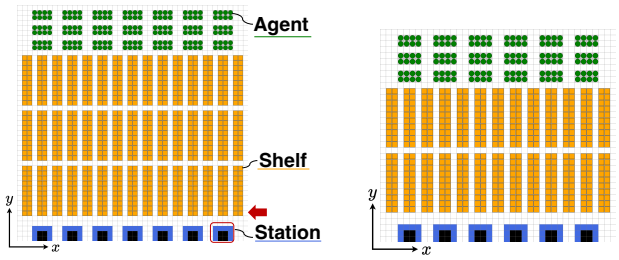
Learning-based methods, including PRIMAL$_2$ (Damani et al. 2021) and SACHA (Lin and Ma 2023), have gained attention; however, compared to non-learning-based approaches, it is generally more difficult to provide theoretical guarantees. SCRIMP (Wang et al. 2023) augments local observations with inter-agent communication for coordination.

Some studies have modeled warehouse-specific MAPD variants where agents carry shelves, as in DD-MAPD (Li and Ma 2023) and MARPF (Makino, Ohama, and Ito 2024). (Fujisawa et al. 2025) formalized the MAPD-MP problem, where agents carry shelves to picking stations per picking lists and return them to pickup positions, and proposed 3PCS for planning efficient paths. In contrast, our study aimed to enhance planning effectiveness by building on 3PCS by including task preemption using holding nodes, shelf rearrangement, and effective agent assignment.

## Preliminaries and Problem Definition

### Model of Environment and Agent

We introduce discrete time steps $t$, where $t \geq 0$ is a non-negative integer. The set of $N$ carrier agents is denoted by $A = \{a_1, \ldots, a_N\}$, and the set of $M$ shelves is denoted by $S = \{s_1, \ldots, s_M\}$. Let the undirected graph $G = (V, E)$ be the environment, where $V$ denotes the nodes that carrier agents can stay in and $E \subset V \times V$ denotes the edges. Agent $a_i$ on $v \in V$ at time $t$ can move to node $v' \in V$ at $t+1$ if $e_{v,v'} = (v, v') \in E$. A carrier agent can load a shelf $s_j \in S$, carry it along edges, and unload it. At $t$, nodes where $a_i \in A$ and shelf $s_j$ are located are denoted as $v_i(t) \in V$ and $w_j(t) \in V$, respectively. The initial position of shelf $s_j$ is called the *home node* of $s_j$ and is denoted by $h(s_j)$ $(= w_j(0))$. Hereafter, the carrier agent is referred to as an agent.




(a) Large-scale (Env. 1) (b) Medium-scale (Env. 2)

Figure 1: Two Vertical Environments (Env. 1 and Env. 2).

For a warehouse environment, we define the set of $K > 0$ stations $\mathcal{P} = \{\rho_1, \rho_2, \ldots, \rho_K\}$, where $\rho_k \in \mathcal{P}$ is represented by nodes $\rho_k = \{v_1^{\rho_k}, \ldots, v_{u_k}^{\rho_k}\} \subset V$, where $u_k$ denotes station $\rho_k$ capacity. We assume $u = u_1 = \cdots = u_K$ for simplicity. Figure 1a shows this environment, where green circles represent agents, blocks at the bottom are seven stations ($K = 7$) with eight blue nodes ($u = 8$), each called a *station node*, and orange nodes are shelf areas where shelves can be placed (they cannot be placed on other nodes).

We introduce the parameter $c_i(t) \in S \cup \{\bot\}$ for agent $a_i$ to represent the shelf $s_j$ that $a_i$ carries at $t$; otherwise, $c_i(t) = \bot$. Clearly, the following condition holds:

$$c_i(t) = s_j \Rightarrow v_i(t) = w_j(t) \tag{1}$$

for any agent $a_i$ and shelf $s_j$, but its converse does not hold. When $c_i(t) = \bot$ and $a_i$ and $s_j$ are on the same node (i.e., $v_i(t) = w_j(t)$), the agent can load $s_j$, resulting in $c_i(t) = s_j$, and carry it. When $a_i$ carries shelf $s_j$ (i.e., $c_i(t) = s_j$), $a_i$ can unload the shelf at the current node at $t$, leading to $c_i(t) = \bot$, after which the shelf remains at node $w_j(t)$ $(= v_i(t))$ until it is carried again, as a shelf cannot move by itself. We introduce parameter $d_j(t)$ to represent the state of shelf $s_j \in A \cup \{\bot\}$; $d_j(t) = a_i$ indicates that $s_j$ is carried by $a_i$ at $t$, and $d_j(t) = \bot$ indicates that $s_j$ is not carried at $t$, thus remaining at the same node at $t+1$. Therefore, $a_i$ can load shelf $s_j$ at $t$ when $c_i(t) = d_j(t) = \bot$ and $v_i(t) = w_j(t)$, after loading $c_i(t) = s_j$ and $d_j(t) = a_i$, assuming loading and unloading are instantaneous.

Two agents cannot occupy a single node or traverse an edge in opposite directions simultaneously, as this may cause a collision. No two shelves can also simultaneously occupy a node. This indicates that $a_i$ can pass under shelves when $c_i(t) = \bot$, whereas it cannot when carrying a shelf. Formally, these constraints can be represented by

$$v_i(t) \neq v_{i'}(t) \wedge w_j(t) \neq w_{j'}(t) \tag{2}$$

$$v_i(t) \neq v_{i'}(t+1) \vee v_{i'}(t) \neq v_i(t+1) \tag{3}$$

for $\forall a_i, a_{i'} \in A$ s.t. $i \neq i'$ and $\forall s_j, s_{j'} \in S$ s.t. $j \neq j'$. Note that Eq. 3 prohibits head-on collision at an edge.

## Picking List and MAPD-MP Problem

We introduce a *picking list*, consisting of multiple *tasks*: carrying different shelves to a station where items are packed from those shelves after all required shelves arrive, and returning shelves to their home nodes, where shelves are loaded. A task is defined as a tuple $\tau_{(i,j,k)} = \langle s_i, v_k^{\rho_j} \rangle$, indicating loading shelf $s_i \in S$, carrying it to node $v_k^{\rho_j}$ of picking station $\rho_j$, returning to the home node, and unloading it. The $n$-th picking list for station $\rho_j$, denoted by $L_n^{\rho_j} = \{\tau_{(i_1,j,k_1)}, \tau_{(i_2,j,k_2)}, \dots\}$ s.t. $|L_n^{\rho_j}| \leq u$, is defined as a set of tasks, where $\{s_{i_1}, s_{i_2}, \dots\}$ are different shelves and $\{v_{k_1}^{\rho_j}, v_{k_2}^{\rho_j} \dots\}$ ($\subset \rho_j$) are different nodes in $\rho_j$. As agents and shelves cannot occupy the same node, $v_{k_n}^{\rho_j} \neq v_{k_{n'}}^{\rho_j}$.

Once shelves in $L_n^{\rho_j}$ arrive at their nodes in $\rho_j$ and remain for $W$ time steps, the *first half* of the picking list is completed, where $W$ represents the time for packing items. All shelves must then return to the shelf nodes for unloading in the *second half* of the task. An *MAPD-MP instance* is defined as the set of picking lists; this may be infinite because of tasks required over time. We define the *primitive picking list* $L_n^{pr} = \{\tau^p{}_{i_1}, \tau^p{}_{i_2}, \dots\}$, where $\tau^p{}_i = \langle s_i \rangle$ is a primitive carrying task that is not assigned any station to carry shelf $s_i$ for packing. A primitive picking list may be generated from customer requests and allocated to a station for packing, which becomes the picking list.

One extension from the previous MAPD-MP is the *shelf-selection probability* $p_j$ for shelf $s_j$, which estimates that shelf $s_j$ is likely to be included in tasks to streamline the MAPD-MP. This models warehouse requirements, where certain products are ordered more frequently than others, with probabilities determined from the latest customer order data. These probabilities can be shared with all agents.

## Well-formed MAPD-MP Problem

We address a *well-formed MAPD-MP* problem, a subset of the MAPD-MP problem extended from the well-formed MAPD problem defined in (Ma et al. 2017). The set of *non-task endpoints* $V_{pk}$ in MAPD-MP includes the initial positions $v_i(0)$ of each agent $a_i$ and allows agents to wait without restrictions. The set of *task endpoints* $V_{tsk}$ in MAPD-MP is the union of picking station nodes and home nodes of shelves, that is, $V_{tsk} = \bigcup_{\rho_k \in \mathcal{P}} \rho_k \cup \{h(s_j) \mid s_j \in S\}$. The complete set of *endpoints* is $V_{ep} = V_{pk} \cup V_{tsk}$.

We define a well-formed MAPD-MP problem as one that satisfies the following three conditions:

a) The number of tasks (and picking lists) is finite.

b) The number of non-task endpoints $|V_{pk}|$ is not smaller than the number of agents $N$.

c) For any endpoints $v, v' \in V_{ep}$, there exists a path from $v$ to $v'$ that does not pass through other endpoints.

The initial positions of the agents are non-task endpoints (i.e., parking nodes); therefore, Condition b) naturally holds.

## 3PCS Algorithm

Here, we describe 3PCS, an existing method for solving well-formed MAPD-MP problems, as our proposed method extends it, although it can adapt to another multi-agent path planning algorithm. 3PCS is a decentralized path-planning algorithm based on TP (Ma et al. 2017), where agents coordinate paths and task assignments via a shared data structure called a *token*. In 3PCS, agent $a_i$ has a priority $q_i(t)$ based on its carrying state:

$$q_i(t) = \begin{cases} 1 & \text{if } \exists k \text{ s.t. } c_i(t) = s_k \\ 0 & \text{if } c_i(t) = \perp \end{cases} \tag{4}$$

Under this priority scheme, a high-priority agent $a_i$ plans its path while ignoring the paths of lower-priority agents. If a conflict occurs, the lower-priority agent re-plans to avoid the paths of high-priority agents. However, a replanning agent $a_j$ may not be at an endpoint, and sometimes, no conflict-free path exists. In such cases, the low-priority agent $a_j$ unable to find a path is promoted to the highest priority ($q_j(t) = 2$), and $a_i$'s path planning is repeated. Through this process, all agents obtain conflict-free paths. This indicates that agents without shelves can pass through endpoints other than the endpoints of their assigned tasks, shortening path length and improving performance as they can pass under shelves at task endpoints in general.

In 3PCS, agents use a simple task assignment strategy. When $a_i$ has no path in the token, it proceeds as follows:

- If $a_i$ is already assigned a task, it plans a path to the next destination for the second-half of the task.
- Otherwise, $a_i$ seeks task $\tau_{(l,j,k)} = \langle s_l, v_k^{\rho_j} \rangle$ where $s_l$ and $v_k^{\rho_j}$ are not performing other tasks. It generates a path to $v_k^{\rho_j}$ that avoids collisions with higher-priority agents for the first half of task $\tau_{(l,j,k)}$.
- If no task exists and the agent's current location $v_i(t)$ is not on any unassigned task's destination, the agent stays there; otherwise, it plans a path to the nearest non-task endpoint.

Please refer to (Fujisawa et al. 2025) for details of 3PCS.

3PCS enables agents to pass under shelves while prioritizing agents carrying shelves in path planning, generating paths that are impossible with TP. These features contribute to efficient completion of the picking list. Furthermore, 3PCS is complete for well-formed MAPD-MP problems, guaranteeing a solution.

# Proposed Method

The proposed method consists of three components and integrates them into existing path planners, such as 3PCS, to maximize the *picking list throughput* (i.e., the number of completed picking lists per time step). The first component is *shelf arrangement optimization*, which rearranges shelf home nodes using the shelf-selection probability $p_j$. Second, we introduce the task assignment procedure, formulated as a *stable matching problem* (Gale and Shapley 1962). Third, *holding nodes* are added, where agents temporarily hold shelves near stations for subsequent picking lists, allowing them to move to the next tasks in advance, and other agents later take these shelves to target stations. The agent must determine the appropriate nodes for the holding shelves. These

components are independent of path planning, allowing integration by replacing task assignment in existing MAPD-MP solvers. Furthermore, these components preserve endpoint-to-endpoint planning and rely on a complete planner such as 3PCS; consequently, completeness can be established under the same assumptions. To efficiently execute the MAPD-MP, a queue at each station enables agents to view future picking lists, allowing effective utilization of the approach. Detailed descriptions of each component are provided in the subsequent sections.

## Picking List Prefetch

We introduce a queue of picking lists for each station $\rho_j$, denoted by $\mathcal{L}^{\rho_j} = \{L_1^{\rho_j}, \dots, L_\delta^{\rho_j}\}$, whose elements are picking lists assigned to $\rho_j$, where $\delta$ is a small positive integer. In our experiments, $\delta = 2$. The queue elements indicate the picking lists assigned to $\rho_j$ for subsequent execution, allowing agents to prefetch upcoming tasks in the picking list in the queue.

When the first half of $L_i^{\rho_j}$ is completed, it is removed from queue $\mathcal{L}^{\rho_j}$ while agents proceed to the second half. Then, the first picking list $L_{i+1}^{\rho_j}$ in queue $\mathcal{L}^{\mathcal{P}_j}$ is performed, and a primitive picking list $L_n^{pr} = \{\tau^p{}_{n_1}, \tau^p{}_{n_2}, \dots\}$ satisfying this condition is selected: any shelf $s_{n_k}$ in $L_n^{pr}$ is not in tasks in queues of all stations or $d(s_{n_k}) = \perp$. The picking list $L_n^{\rho_j}$ is generated from $L_n^{pr}$ by adding station $\rho_j$ and allocating station nodes to the primitive tasks. The resulting $L_n^{\rho_j}$ is added to its queue $\mathcal{L}^{\rho_j}$. The first element is an *active picking list* as it is ready to be performed. The set of tasks that have not yet been assigned to agents in active picking lists is denoted by $\mathcal{T}$. Our method works without queues when $\delta = 1$, as all the picking lists in the queues are active.

## Shelf Arrangement Optimization (SAO)

The agent $a_m$ assigned $\tau_{(i,j,k)}$ at time $t$ plans the collision-free path to station node $v_k^{\rho_j}$ via $w_i(t)$ for the first-half task. Consider an environment similar to that shown in Fig. 1a. In this environment, the farther the distance from the station, the longer the carrying time. While performing tasks, rearranging shelves with higher shelf-selection probabilities $p_i$ closer to stations and placing those with lower probabilities farther away can increase the picking list throughput.

To achieve this, immediately before the agent plans a path to return shelf $s_j$ to its home node $h(s_j) \in V$, it attempts to swap the home node with another working agent with the lowest shelf-selection probability, whose home node is closer to the stations. We define the distance from the home node of $s_j$ to the stations $D(s_j)$. For example, assuming the environment shown in Fig. 1a, we can define

$$D(s_j) = h(s_j).y \tag{5}$$

where for $v \in V$, $v.y$ denotes the $Y$-coordinate value with origin $(0, 0)$ at the lower left corner, although distance $D(s_j)$ can be defined by environmental structures. The details are described by Algorithm 1.

First, let $S_{cand}$ denote the set of shelves $s_k$ being carried to the station, where distance $D(s_k)$ is smaller than $D(s_j)$

Algorithm 1: Shelf Arrangement Optimization

**Input**: Shelf $s_j$ being returned to its home $h(s_j)$
**Output**: Optimized home nodes of shelves

1: $S_{cand} \leftarrow \emptyset$
2: **for** each shelf $s_k$ in $S$ **do**
3: **if** $s_k$ is being carried to a station and $D(s_j) > D(s_k)$ **then**
4: $S_{cand} \leftarrow S_{cand} \cup \{s_k\}$
5: **end if**
6: **end for**
7: $s_{tgt} \leftarrow \arg\min_{s_k \in S_{cand}} p_k$
8: **if** $p_{tgt} < p_j$ **then**
9: swap the node of $h(s_{tgt})$ and $h(s_j)$
10: **end if**

Algorithm 2: Task Set Selection and Assignment for Agents

1: **Input**: Stations $\mathcal{P}$, Agents with no assigned tasks $A^*$
2: $\mathcal{P}_{yet} \leftarrow \{\rho_j \in \mathcal{P} \mid n_{free}^{\rho_j} > 0\}$
3: **while** $\mathcal{P}_{yet} \neq \emptyset$ and $A^* \neq \emptyset$ **do**
4: $n_{min} \leftarrow \min_{\rho_j \in \mathcal{P}_{yet}} n_{free}^{\rho_j}$
5: $\mathcal{P}^* \leftarrow \{\rho_j \mid n_{free}^{\rho_j} = n_{min}\}$
6: $\mathcal{P}_{yet} \leftarrow \mathcal{P}_{yet} \setminus \mathcal{P}^*$
7: $\mathcal{T}^* \leftarrow$ {unassigned tasks in active picking lists of stations in $\mathcal{P}^*$}
8: $A^*_{asgnd} \leftarrow$ **TaskAssignmentWithGS**$(A^*, \mathcal{T}^*)$
9: $A^* \leftarrow A^* \setminus A^*_{asgnd}$
10: **end while**

(Lines 1-6). We set

$$s_{tgt} = \arg\min_{s_k \in S_{cand}} p_k \tag{6}$$

(Line 7). If $p_{tgt} < p_j$, swap the home positions of $s_{tgt}$ and $s_j$ (Lines 8-10). This rearrangement allows shelves with higher selection probabilities to migrate towards positions closer to the stations. Because $s_{tgt}$ is still heading towards the station, it does not affect the current plan.

## Task Assignment using Stable Matching

TAGS is a task assignment procedure using the Gale–Shapley (GS) algorithm (Gale and Shapley 1962). The GS algorithm finds a *stable matching* between bipartite sets, assuming each element of one set has a rank for all elements in the other set. We consider two groups of agents and unassigned tasks in the picking lists of the first elements of station queues, which should be performed subsequently. The GS algorithm preferentially selects picking lists with fewer unassigned tasks to enhance completion (randomly selected for tie breaks). To apply the GS algorithm, agents rank unassigned tasks by distance to shelf home positions and tasks rank free agents by distance to shelves.

Algorithm 2 shows our procedure for selecting task sets, where $n_{free}^{\rho_j}$ denotes the number of unassigned tasks in the first picking list in station $\rho_j$'s queue, and $A^*$ is the set of

Algorithm 3: Task Assignment using GS Algorithm

1: **function** TASKASSIGNMENTWITHGS($A^*, \mathcal{T}^*$)
2: $\mathcal{T}_{free} \leftarrow \mathcal{T}^*$
3: $\forall \tau \in \mathcal{T}^*, A^{\tau}_{yet} \leftarrow A^*$ // set of agents not yet proposed to
4: $M \leftarrow \emptyset$ // set of assigned pairs $\langle \tau, a \rangle$
5: **while** $\mathcal{T}_{free} \neq \emptyset$ **do**
6: Pop a free task $\tau \in \mathcal{T}_{free}$
7: $a \leftarrow \arg\min_{a' \in A^{\tau}_{yet}} \textbf{cost}(a', \tau)$
8: $A^{\tau}_{yet} \leftarrow A^{\tau}_{yet} \setminus \{a\}$
9: **if** $\forall \langle \tau', a' \rangle \in M, a' \neq a$ **then**
10: $M \leftarrow M \cup \{\langle \tau, a \rangle\}$
11: **else**
12: Let $\tau'$ be the task currently assigned to $a$.
13: **if** $\textbf{cost}(a, \tau) < \textbf{cost}(a, \tau')$ **then**
14: $M \leftarrow (M \setminus \{\langle \tau', a \rangle\}) \cup \{\langle \tau, a \rangle\}$
15: **if** $A^{\tau'}_{yet} \neq \emptyset$ **then**
16: $\mathcal{T}_{free} \leftarrow \mathcal{T}_{free} \cup \{\tau'\}$
17: **end if**
18: **else if** $A^{\tau}_{yet} \neq \emptyset$ **then**
19: $\mathcal{T}_{free} \leftarrow \mathcal{T}_{free} \cup \{\tau\}$
20: **end if**
21: **end if**
22: **end while**
23: **return** $\{a \mid \langle \tau, a \rangle \in M\}$
24: **end function**

agents with no assigned tasks. First, we generate $\mathcal{P}_{yet} = \{\rho \in \mathcal{P} \mid n^{\rho}_{free} > 0\}$ and define

$$n_{min} = \min_{\rho \in \mathcal{P}_{yet}} n^{\rho}_{free} \tag{7}$$

(Lines 2-4). $\mathcal{T}^*$ is defined as the set of unassigned tasks in active picking lists of stations $\forall \rho_{j'} \in \mathcal{P}_{yet}$, where $n^{\rho_{j'}}_{free} = n_{min}$. The selected stations are removed from $\mathcal{P}_{yet}$ (Lines 5-7). For task set $\mathcal{T}^*$ and agents $A^*$ that have not yet been assigned a task, we call the **TaskAssignmentWithGS** procedure, and the set of assigned agents $A_{asgnd}$ is returned. Then, $A^*$ is recalculated (Lines 8 and 9). This process is repeated until $\mathcal{P}_{yet}$ or $A_{yet}$ becomes empty.

Algorithm 3 presents **TaskAssignmentWithGS**($A^*, \mathcal{T}^*$) for matching agents to tasks. We initialize the set $\mathcal{T}_{free}$ of unpaired tasks as $\mathcal{T}^*$, and for each task $\tau$, we initialize $A^{\tau}_{yet}$ of agents to which $\tau$ has not proposed as $A^*$. Variable $M$ is defined as the set of established pairs (Lines 2-4). We select an unassigned task $\tau$ and choose agent $a$ with minimum cost from $A^{\tau}_{yet}$ (Lines 5-8). The cost function is defined for agent $a_n$ and task $\tau_{(i,j,k)} = \langle s_i, v^{\rho_j}_k \rangle$ as the path length from position $v_n(t)$ to shelf position $w_i(t)$ using an A*-search.

If $a$ is not paired with any task, we assign $(\tau, a)$ and add it to $M$ (Lines 9-10). If $a$ is already paired with task $\tau'$, we compare the costs of $(\tau, a)$ and $(\tau', a)$ and store only the lower-cost pair in $M$. The removed task $\tau$ or $\tau'$ returns to $\mathcal{T}_{free}$ (Lines 11-20). By repeating this process, we obtain a task-optimal stable matching between tasks and agents. Finally, this function returns the assigned agents.

Algorithm 4: Subtask Assignment for Agent

**Input**: Agent $a^*$; Next picking lists $\mathcal{L}^{next}(t)$ in queues;
Estimated completion time of first-half tasks of active picking lists $\{t^{\rho_1}_{comp}, t^{\rho_2}_{comp}, \ldots\}$
**Output**: Selected subtask for $a^*$
// $\mathcal{T} = \emptyset$ must be true at the beginning
**if** all nodes in $V_{hld}$ are reserved **then**
**return** // no subtask is assigned
**end if**
$\mathcal{S}_{cand} \leftarrow \emptyset$ // set of possible subtasks
**for** each next picking list $L^{\rho_j}_m \in \mathcal{L}^{next}(t)$ **do**
**for** each task $\tau_{(i,j,k)} \in L^{\rho_j}_m$ **do**
**if** there is no subtask of $\tau_{(i,j,k)}$ assigned **then**
$v^n_{hld} \leftarrow \arg\min_{v^{n'}_{hld} \in V_{hld}} \textbf{cost}(v^{n'}_{hld}, v^{\rho_j}_k)$
$\mathcal{S}_{cand} \leftarrow \mathcal{S}_{cand} \cup \{\langle \tau_{(i,j,k)}, v^n_{hld} \rangle\}$
**end if**
**end for**
**end for**
$\sigma^n \leftarrow \arg\min_{\sigma^{n'}_{(i,j,k)} \in \mathcal{S}_{cand}} \textbf{GetScore}(a^*, \sigma^{n'}_{(i,j,k)}, t^{\rho_j}_{comp})$
**if** $\sigma^n \neq \perp$ **then**
// assign $\sigma^n$ to $a^*$
// make $v^n_{hld}$ busy until $\sigma^n$ is completed
**return** $\sigma^n$
**end if**

## Holding Node Integration (HNI) for Path Planning

We introduce *holding nodes* that allow agents to temporarily hold shelf $s_i$ near stations (e.g., the bottom row indicated by an arrow in Fig. 1a) for the next picking list $L^{\rho_j}_{n+1}$ after the current list $L^{\rho_j}_n$. Shelves are later picked up by another agent, carried to the station for the first half task, and then returned to their original position for the second half. Note that, initially, no shelves are placed on the holding nodes. We denote the set of holding nodes as $V_{hld}$. Assuming that holding nodes are non-task endpoints ($V_{hld} \subset V_{pk}$), agents can stay at these nodes indefinitely.

We denote the subtask carrying shelf $s_i$ to holding node $v^n_{hld} \in V_{hld}$ prior to executing $\tau_{(i,j,k)}$ as $\sigma^n_{(i,j,k)} = \langle \tau_{(i,j,k)}, v^n_{hld} \rangle$. When $\sigma^n_{(i,j,k)}$ is assigned to an agent, it must be completed before $\tau_{(i,j,k)}$ starts. However, an agent can be assigned only a portion of $\tau_{(i,j,k)}$ in the picking list without executing $\sigma^n_{(i,j,k)}$. The agent assigned to $\sigma^n_{(i,j,k)}$ need not perform $\tau_{(i,j,k)}$. Once an agent is assigned $\tau_{(i,j,k)}$ after $\sigma^n_{(i,j,k)}$ completion, it must plan a path to the holding node $v^n_{hld}$, load shelf $s_i$, carry it to station node $v^{\rho_j}_k$, and return to $h(s_i)$.

Algorithm 4 describes a *subtask assignment procedure* to assign a subtask to agent $a_m$. This procedure selects and assigns a subtask from candidates that are expected to most effectively improve the picking list throughput. The assignment occurs only when $\mathcal{T} = \emptyset$, meaning that all tasks in the active picking lists are already assigned, as tasks in $\mathcal{T}$ must be executed first. We assume that the queue lengths at the stations exceed one ($\delta \geq 2$).

If all nodes in $V_{hld}$ are occupied or reserved, this procedure terminates (Lines 5-7); otherwise, such a free holding node is called *open*. Next, we initialize the set $\mathcal{S}_{cand}$ of candidate subtasks as being empty. For task $\forall\tau_{(i,j,k)} \in L^{\rho_j}_{n+1}$ in $\mathcal{L}^{next}(t)$, if its associated subtask has not been assigned, it selects the open holding node with the smallest cost relative to station node $v^{\rho_j}_k$, where $\mathcal{L}^{next}(t)$ is the set of second picking lists in all queues. The subtask $\langle\tau_{(i,j,k)}, v^n_{hld}\rangle$ is then added to $\mathcal{S}_{cand}$ (Lines 8-16). Finally, from $\mathcal{S}_{cand}$, the subtask most likely to improve the throughput for agent $a^*$ is selected by **GetScore**. This subtask is assigned to agent $a^*$, and the selected storage node $v^n_{hld}$ enters a busy state until completion (Lines 17-22).

Function **GetScore**$(a_m, \sigma^n_{(i,j,k)}, t^{\rho_j}_{comp})$ given in Alg. 5 calculates four costs for agent $a_m$: (1) cost $C_1$ for moving from position $v_m(t)$ to home node $h(s_i)$ of shelf $s_i$; (2) cost $C_2$ for moving from $h(s_i)$ to node $v^{\rho_j}_k$ in station $\rho_j$; (3) cost $C_3$ for moving from $h(s_i)$ to holding node $v^n_{hld}$ in $\sigma^n_{(i,j,k)}$; and (4) cost $C_4$ for moving from $v^n_{hld}$ to $v^{\rho_j}_k$ (Lines 2-5), where cost is the path length between two nodes.

As shown in Fig. 2, if no subtask is assigned, the time to carry shelf $s_i$ to station $\rho_j$ after the active picking list is completed is estimated by $t^{comp}_{\rho_j} + C_1 + C_2$, where $t^{comp}_{\rho_j}$ is the estimated completion time of the active picking list for station $\rho_j$. If a subtask is assigned to agent $a_m$, it can start immediately, and the cost of carrying $s_i$ to $\rho_j$ after the holding node is $C_1+C_3+C_4$. Thus, the benefit score gained by assigning the subtask is

$$\begin{aligned}&(t^{comp}_{\rho_j} + C_1 + C_2) - (C_1 + C_3 + C_4)\\ &\quad = C_1 + C_2 - C_4 + \big(t^{comp}_{\rho_j} - (C_1 + C_3)\big).\end{aligned} \tag{8}$$

The shelf $s_i$ should be held at node $v^n_{hld}$ when $s$ is expected to arrive before the current active picking list is completed, that is, when $t^{comp}_{\rho_j} > (C_1 + C_3)$. Because a longer holding time does not improve efficiency, the benefit score considering holding time can be calculated as

$$Q = C_1 + C_2 - C_4 + \min\big(t^{comp}_{\rho_j} - (C_1 + C_3), 0\big) \tag{9}$$

(Line 6).

When the cost function uses the distance from the A* search, the score $Q$ represents the expected time reduction to carry the shelf to the station versus no subtask assignment. The completion time $t^{comp}_{\rho_j}$ is estimated from the path of the agent assigned to the task in an active picking list. More specifically, $t^{comp}_{\rho_j}$ is defined as the maximum estimated arrival time of all agents involved in tasks within the active picking list, calculated from the current paths, plus $W$. If no agents have reached their assigned shelves, making estimation impossible, $t^{comp}_{\rho_j}$ is set to infinity. Thus, $\min\big(t^{comp}_{\rho_j} - (C_1 + C_3),\, 0\big) = 0$.

If $Q < 0$, agent $a_m$ cannot obtain benefit from executing subtask $\sigma^n_{(i,j,k)}$ and **GetScore** returns $\perp$. Otherwise, to encourage the selection of shelves closer to the agent's position $v_m(t)$, it returns $Q - C_1$, subtracting the cost from the current position to the shelf's home node (Lines 7-11).

Algorithm 5: GetScore Function

```
1: function GetScore(a_m, σ^n_(i,j,k), t^{ρ_j}_comp)
2:     C_1 ← cost(v_m(t), h(s_i))
3:     C_2 ← cost(h(s_i), v^{ρ_j}_k)
4:     C_3 ← cost(h(s_i), v^n_hld)
5:     C_4 ← cost(v^n_hld, v^{ρ_j}_k)
6:     Q ← C_1 + C_2 − C_4 + min(t^{ρ_j}_comp − (C_1 + C_3), 0)
7:     if Q < 0 then
8:         return ⊥
9:     else
10:        return Q − C_1
11:    end if
12: end function
```

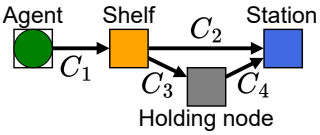



Figure 2: Distance structure used in **GetScore** function

## Experimental Evaluation

### Experimental Settings

We conducted experimental evaluations in MAPD-MP warehouse environments to demonstrate the effectiveness of our method, comparing it with baselines. We present the results of an ablation study by omitting each component of our method. The experimental environments are shown in Figs. 1a (Env. 1) and 1b (Env. 2), consisting of two-dimensional grid spaces, where green circles denote the initial positions of the agents, orange squares represent shelf home nodes, and blue square blocks indicate stations. The numbers of nodes $|V|$, agents $|A|$, shelves $|S|$, and stations $|\mathcal{P}|$ in these environments are listed in Table 1. We set $W = 0$ to evaluate only the performance of carrying tasks. We made Env. 2 smaller for the evaluation of congested situations. The number of agents varied within the specified range in increments of eight. Each experiment was run for $10,000$ timesteps, and we report the average values over 100 runs. The implementation was written in Java using an Apple M4 Pro CPU with 64 GB of RAM.[1]

We fixed the picking list size to $u = 8$, which is equal to the number of station nodes at all stations. This configuration represents the most time-consuming scenario for processing the picking list. The shelf-selection probability $p_i$ of shelf $s_i$ was set using the Zipf distribution

$$p_i = \frac{i^{-\alpha}}{\sum_{j=1}^{|S|} j^{-\alpha}} \quad (i = 1, 2, \ldots, |S|), \tag{10}$$

where $\alpha$ is a Zipf exponent controlling skewness when $\alpha \geq 0$; a higher $\alpha$ creates a more biased distribution. We set $\alpha = 1.0$ and $2.0$ for Env. 1 and $\alpha = 2.0$ for Env. 2.

[1]The program code for the experiment is available at https://github.com/SunnyWist/SAO-TAGS-HNI.

| | Env. 1 (Fig. 1a) | Env. 2 (Fig. 1b) |
|---|---|---|
| No. of nodes $\|V\|$ | $46 \times 47$ | $40 \times 36$ |
| No. of agents $\|A\|$ | $56 \sim 168$ | $48 \sim 144$ |
| No. of shelves $\|S\|$ | 900 | 520 |
| No. of stations $\|\mathcal{P}\|$ | 7 | 6 |

Table 1: Environmental parameters used in the experiments

| Type | $\|A\|$ | Throughput | CPU time [s] | Task op. rate |
|---|---|---|---|---|
| TP | 56 | 0.04915 | 1.581 | 0.986 |
| | 112 | 0.07580 | 4.578 | 0.686 |
| | 168 | 0.07560 | 6.747 | 0.457 |
| 3PCS | 56 | 0.05076 | 1.986 | 0.985 |
| | 112 | 0.08309 | 6.045 | 0.704 |
| | 168 | 0.08274 | 7.560 | 0.467 |
| 3PCS +All | 56 | 0.06386 | 2.305 | 0.986 |
| | 112 | 0.12157 | 9.074 | 0.809 |
| | 168 | 0.14266 | 13.998 | 0.626 |

Table 2: Results in Env. 1 (Fig. 1a) at $\alpha = 2.0$

The Zipf distribution and its parameters reflect empirical patterns in automated warehouse systems (Brynjolfsson, Hu, and Smith 2010; Rahman and Kirby 2024; Øverby and Audestad 2021).

We selected two baseline methods for comparison. First, TP (Ma et al. 2017), a decentralized MAPD solver, was modified for MAPD-MP to generate paths for station and home nodes. The agents were prohibited from traversing the endpoints. Second, we used the naive 3PCS (Fujisawa et al. 2025), a solver for the general MAPD-MP problem that ignores shelf-selection probabilities. For ablation, we tested 3PCS with SAO (denoted as 3PCS+SAO), 3PCS with SAO and TAGS (denoted as 3PCS+SAO+TAGS), and 3PCS with SAO with HNI (denoted as 3PCS+SAO+HNI). Our proposed method integrates SAO, TAGS, and HNI, denoted as 3PCS+All. We evaluated these methods using throughput (number of completed picking lists per time step), CPU planning time, task operating rate (ratio of time agents move for tasks, excluding waiting and holding time), and holding node usage rate (when HNI is applied).

## Experimental Results and Discussion

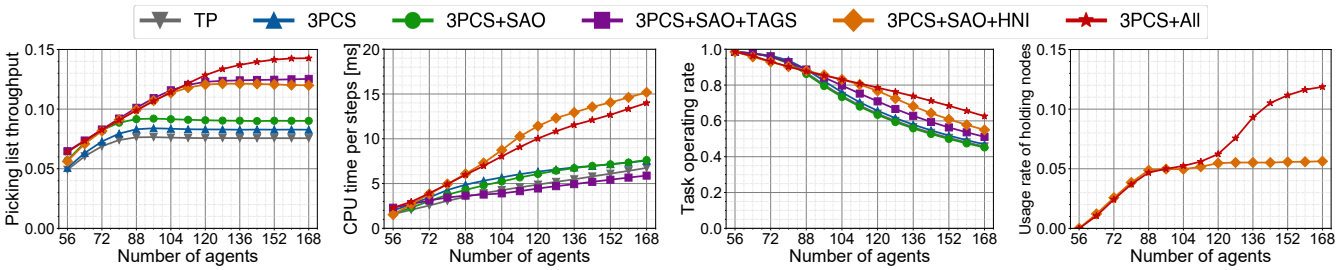
**Performance Analysis** Figure 3a presents the throughput in Env. 1 with Zipf exponent $\alpha = 2.0$. Our proposed method, 3PCS+All, outperformed all other methods in all cases. The baselines, TP and 3PCS, performed well when $|A| \leq 80$ but could not increase the throughput as $|A|$ increased. 3PCS+SAO showed increased throughput for smaller agent numbers (up to 88) compared to the baseline but showed little improvement beyond that point. Both 3PCS+SAO+TAGS and 3PCS+SAO+HNI achieved a throughput comparable to that of 3PCS+SAO when the number of agents was 80 or fewer, but their throughput continued to increase up to approximately 120 agents. 3PCS+All increased in throughput when $|A| \geq 120$, achieving the highest overall throughput. These results show that the proposed method effectively utilizes agents, even with large numbers of agents.

Table 2 summarizes the throughput values of TP, 3PCS, and 3PCS+All for $\alpha = 2.0$ in Env. 1. When $|A| = 168$, 3PCS+All's throughput increased by approximately $88.7\%$ compared to TP and $72.4\%$ compared to 3PCS. As the number of agents grew from 56 to 168, TP's throughput increased by approximately 1.54 times, and that of 3PCS increased by 1.63 times. In contrast, 3PCS+All achieved an increase of 2.23 times. Although frequent pauses owing to congestion are expected with more agents, 3PCS+All can effectively utilize a large number of agents.

We also present the CPU time per time step (Fig. 3b), the rate of steps during which agents were assigned tasks (i.e., the task operating rate) (Fig. 3c), and the usage rate of holding nodes for the methods that incorporate holding nodes (Fig. 3d) in Env. 1 with $\alpha = 2.0$. Table 2 also lists the values of these metrics for TP, 3PCS, and 3PCS+All when $|A| = 56, 112, 168$. As shown in Fig. 3b, the CPU time per time step increased with the number of agents for all methods, maintaining linear growth. The maximum CPU time was approximately 15 ms, which is sufficiently small for real-time performance in the operational time scales of carrier robots (agents) in warehouse systems. Because TAGS assigned tasks to better agents, the lengths of paths were likely to be shorter, resulting in lower CPU time usage.

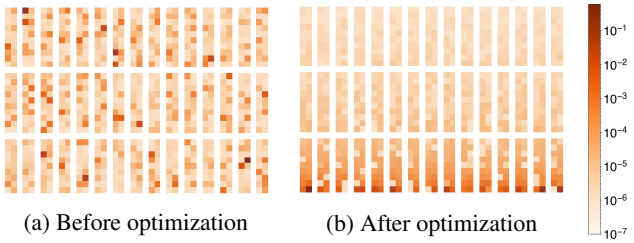
Figure 3c shows that the task operating rate decreased as the number of agents increased for all methods because concurrent picking list processing is limited; increasing the number of agents could not increase the number of assignable tasks. However, 3PCS+SAO+TAGS, 3PCS+SAO+HNI, and 3PCS+All maintained higher operating rates than the baseline methods, particularly with more agents. Task assignment using the GS algorithm helped balance task-processing times across agents, reducing instances in which agents completed tasks much earlier than others and maintaining a high operating rate. The introduction of holding nodes simultaneously increased the number of active agents through subtask allocation, thereby improving the overall operating rate. Through these combined components, 3PCS+All maintained an operating rate approximately $20\%$ higher than that of TP, even with 168 agents.

An interesting phenomenon can be observed in Fig. 3d. The usage rate of holding nodes increased with the number of agents for methods employing holding nodes. In 3PCS+SAO+HNI, the rate reached about $5\%$ (88 agents), with minimal growth thereafter. In 3PCS+All, the rate continued to increase to $12\%$ at 168 agents, indicating more effective holding-node utilization. A key feature of 3PCS+All is that the growth slowed at 88 agents but increased again beyond 120 agents. This occurs because many agents in 3PCS become idle beyond 88 agents. In 3PCS+All, agents performed subtasks to retrieve shelves from holding nodes and deliver them to stations, explaining the resumed increase. Beyond 120 agents, shelf transport subtasks to holding nodes increased, raising the usage rate further.



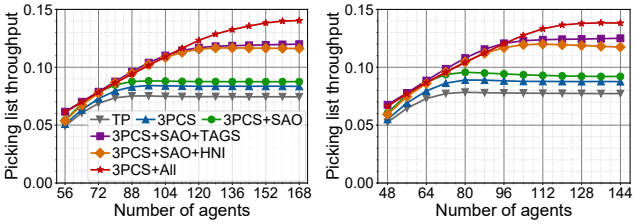

(a) Throughput (b) CPU time (c) Task operating rate (d) Usage rate of holding nodes

Figure 3: Results in Env. 1 (Fig. 1a) at $\alpha = 2.0$



(a) Before optimization (b) After optimization

Figure 4: Shelf-Selection Probability Distribution ($\alpha = 2.0$).




(a) Env. 1 at $\alpha = 1.0$ (b) Env. 2 at $\alpha = 2.0$

Figure 5: Comparison of throughput in each environment

**Distribution of Shelves** We investigated the effect of the SAO by comparing the initial and final states of shelf locations with shelf-selection probabilities. Figure 4 shows Env. 1 with $\alpha = 2.0$; the heatmaps of shelf-selection probabilities before execution (at $t = 0$) are shown in Fig. 4a and after execution in (at $t = 10,000$) in Fig. 4b when using 3PCS+SAO. Each heatmap shows the shelf-selection probabilities through color intensity, with darker colors indicating higher probabilities. Because probabilities follow the Zipf distribution, the color bar uses a logarithmic scale.

The shelves were initially distributed uniformly (Fig. 4a), and shelves with high probabilities in the upper regions may be selected frequently, causing longer agent travel times and reduced throughput. Through execution, shelves with higher shelf-selection probabilities were positioned closer to the stations. This shows that dynamic shelf arrangement optimization relocates frequently selected shelves closer to station-proximal areas, creating an arrangement aligned with task demand.

**Difference in Zipf Exponent and Congested Environment** Figure 5a presents the throughput in Env. 1 with Zipf exponent $\alpha = 1.0$. Although the shelf-selection probabilities differed from those in the $\alpha = 2.0$ case, their overall trend was similar (Fig. 3a). In particular, 3PCS+All again achieved the highest throughput in this scenario. This suggests that our proposed method is not dependent on a specific selection distribution and can be effective across a wide range of distributions.

Figure 5b shows the throughput for each number of agents in Env. 2 with $\alpha = 2.0$. As Env. 2 has fewer nodes and shelves than Env. 1, agents faced congestion more frequently, making throughput improvement harder although their paths are shorter. The overall trend was nearly identical to that of Env. 1, with 3PCS+All achieving nearly the highest throughput. This suggests that our method remains effective even in environments with frequent conflicts. When the number of agents was low, 3PCS+SAO+TAGS was slightly more efficient, likely because it used holding nodes even without congestion, which resulted in longer paths.

Overall, our proposed methods effectively exploit both the structural characteristics of MAPD-MP environments and the shelf-selection probabilities induced by Zipf's law. As a result, 3PCS+All achieved substantial improvements in throughput, even when the number of agents increased.

## Conclusion

This study addressed the MAPD-MP problem, where shelves are carried to stations and returned to their original positions, by introducing shelf selection skew, which is common in automated warehouses, and augmenting stations with short picking list queues. Leveraging these characteristics, we proposed an approach with three components: (i) SAO, which arranges shelves by shelf-selection probabilities; (ii) TAGS, which uses stable matching to improve task allocation; and (iii) HNI, which enables efficient task processing by relocating shelves to holding nodes. We experimentally demonstrated that our approach, particularly the full combination of all components, effectively utilized agents and improved throughput at high agent densities, outperforming the TP and 3PCS baselines. We plan to extend our method to support asynchronous agent execution and apply it to more complex environmental configurations in real-world automated warehousing.

## Acknowledgments

This work is partly supported by JSPS KAKENHI Grant Number 25K03188.

## References

Atzmon, D.; Stern, R.; Felner, A.; Sturtevant, N. R.; and Koenig, S. 2020. Probabilistic Robust Multi-Agent Path Finding. *Proc. of the Int. Conf. on Automated Planning and Scheduling*, 30: 29–37.

Brynjolfsson, E.; Hu, Y. J.; and Smith, M. D. 2010. The Longer Tail: The Changing Shape of Amazon's Sales Distribution Curve. *SSRN Electronic Journal*.

Damani, M.; Luo, Z.; Wenzel, E.; and Sartoretti, G. 2021. $\text{PRIMAL}_2$: Pathfinding Via Reinforcement and Imitation Multi-Agent Learning - Lifelong. *IEEE Robotics and Automation Letters*, 6(2): 2666–2673. Conf. Name: IEEE Robotics and Automation Letters.

Fujisawa, Y.; Wakasugi, Y.; Nakazawa, K.; Matsubara, R.; and Sugawara, T. 2025. Prioritized Path Planning for Multi-agent Pickup and Delivery with Multi-item Packing Problem. In *Advances in Practical Applications of Agents, Multi-Agent Systems, and Computational Social Science: The PAAMS Collection*, 92–104. Springer Nature Switzerland. ISBN 978-3-032-07638-0.

Gale, D.; and Shapley, L. S. 1962. College admissions and the stability of marriage. *The American mathematical monthly*, 69(1): 9–15.

Gange, G.; Harabor, D.; and Stuckey, P. J. 2019. Lazy CBS: Implicit Conflict-Based Search Using Lazy Clause Generation. *Proc. of the Int. Conf. on Automated Planning and Scheduling*, 29: 155–162.

Li, B.; and Ma, H. 2023. Double-Deck Multi-Agent Pickup and Delivery: Multi-Robot Rearrangement in Large-Scale Warehouses. *IEEE Robotics and Automation Letters*, 8(6): 3701–3708.

Li, J.; Chen, Z.; Harabor, D.; Stuckey, P. J.; and Koenig, S. 2022. MAPF-LNS2: Fast Repairing for Multi-Agent Path Finding via Large Neighborhood Search. *Proc. of the AAAI Conf. on Artificial Intelligence*, 36(9): 10256–10265.

Li, J.; Tinka, A.; Kiesel, S.; Durham, J. W.; Kumar, T. S.; and Koenig, S. 2021. Lifelong multi-agent path finding in large-scale warehouses. In *Proc. of the AAAI Conf. on Artificial Intelligence*, volume 35, 11272–11281. Issue: 13.

Lin, Q.; and Ma, H. 2023. SACHA: Soft Actor-Critic With Heuristic-Based Attention for Partially Observable Multi-Agent Path Finding. *IEEE Robotics and Automation Letters*, 8(8): 5100–5107.

Ma, H.; Li, J.; Kumar, T. S.; and Koenig, S. 2017. Lifelong Multi-Agent Path Finding for Online Pickup and Delivery Tasks. In *Proc. of the 16th Conf. on Autonomous Agents and MultiAgent Systems*, AAMAS '17, 837–845. Richland, SC: IFAAMAS.

Makino, H.; Ohama, Y.; and Ito, S. 2024. MARPF: Multi-Agent and Multi-Rack Path Finding. In *2024 IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, 8435–8441.

Matsui, T. 2024. Integration of Efficient Techniques Based on Endpoints in Solution Method for Lifelong Multiagent Pickup and Delivery Problem. *Systems*, 12(4): 112.

Miyashita, Y.; Yamauchi, T.; and Sugawara, T. 2023. Distributed Planning with Asynchronous Execution with Local Navigation for Multi-agent Pickup and Delivery Problem. In *Proc. of the 2023 Int. Conf. on Autonomous Agents and Multiagent Systems*, AAMAS '23, 914–922. Richland, SC: IFAAMAS ISBN 978-1-4503-9432-1.

Okumura, K. 2024. Engineering LaCAM*: Towards Real-time, Large-scale, and Near-optimal Multi-agent Pathfinding. In *Proc. of the 23rd Int. Conf. on Autonomous Agents and Multiagent Systems*, AAMAS '24, 1501–1509. Richland, SC: IFAAMAS. ISBN 979-8-4007-0486-4.

Okumura, K.; Machida, M.; Défago, X.; and Tamura, Y. 2022. Priority inheritance with backtracking for iterative multi-agent path finding. *Artificial Intelligence*, 310: 103752.

Rahman, M. A.; and Kirby, E. D. 2024. The Lean Advantage: Transforming E-Commerce Warehouse Operations for Competitive Success. *Logistics*, 8(4): 129. Publisher: Multidisciplinary Digital Publishing Institute.

Sharon, G.; Stern, R.; Felner, A.; and Sturtevant, N. R. 2015. Conflict-based search for optimal multi-agent pathfinding. *Artificial Intelligence*, 219: 40–66.

Shimada, D.; Miyashita, Y.; and Sugawara, T. 2025. Multi-Agent Path Finding Using Provisionally Booking Nodes for Pickup and Delivery Problems:. In *Proc. of the 17th Int. Conf. on Agents and Artificial Intelligence*, 528–537. Porto, Portugal: SCITEPRESS. ISBN 978-989-758-737-5.

Silver, D. 2005. Cooperative Pathfinding. *Proc. of the AAAI Conf. on Artificial Intelligence and Interactive Digital Entertainment*, 1(1): 117–122.

Stern, R.; Sturtevant, N.; Felner, A.; Koenig, S.; Ma, H.; Walker, T.; Li, J.; Atzmon, D.; Cohen, L.; Kumar, T. K.; Barták, R.; and Boyarski, E. 2019. Multi-Agent Pathfinding: Definitions, Variants, and Benchmarks. *Proc. of the Int. Symposium on Combinatorial Search*, 10(1): 151–158.

Wang, Y.; Xiang, B.; Huang, S.; and Sartoretti, G. 2023. SCRIMP: Scalable Communication for Reinforcement- and Imitation-Learning-Based Multi-Agent Pathfinding. In *2023 IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, 9301–9308. ISSN: 2153-0866.

Wu, W.; Bhattacharya, S.; and Prorok, A. 2020. Multi-Robot Path Deconfliction through Prioritization by Path Prospects. In *2020 IEEE Int. Conf. on Robotics and Automation (ICRA)*, 9809–9815. ISSN: 2577-087X.

Wurman, P. R.; D'Andrea, R.; and Mountz, M. 2008. Coordinating Hundreds of Cooperative, Autonomous Vehicles in Warehouses. *AI Magazine*, 29(1): 9–9.

Yan, Z.; Zheng, H.; and Wu, C. 2024. Multi-agent Path Finding for Cooperative Autonomous Driving. In *2024 IEEE Int. Conf. on Robotics and Automation (ICRA)*, 12361–12367.

Øverby, H.; and Audestad, J. A. 2021. The Long Tail. In *Introduction to Digital Economics: Foundations, Business Models and Case Studies*, 231–241. Cham: Springer International Publishing. ISBN 978-3-030-78237-5.