Multi-Agent Pickup and Delivery with Heterogeneous Agents

Benedetta Flammini
Politecnico di Milano
Milan, Italy
benedetta.flammini@polimi.it

Francesco Amigoni
Politecnico di Milano
Milan, Italy
francesco.amigoni@polimi.it

Bruno Lacerda
Stateful Robotics
Oxford, United Kingdom
bruno@statefulrobotics.com

ABSTRACT

The Multi-Agent Pickup and Delivery (MAPD) problem involves a team of agents that plan collision-free paths to perform tasks, which appear over time, consisting of picking up and delivering items. Usually, agents are considered homogeneous. For example, an agent can move everywhere in the environment and can complete any task. In this paper, we study the MAPD problem in a setting with heterogeneous agents, where different classes of agents have distinct capabilities and can operate in different areas of the environment. Unlike classical MAPD formulations, our setting requires cooperation across agent classes to complete tasks, possibly spanning multiple operable zones, which requires agents of different classes to exchange items through handovers at frontier locations between their operable zones. To address this challenge, we propose a two-level planning framework that first computes high-level paths for items across zones, determining where exchanges should occur, and then determines which agents will transport items along these high-level paths while generating collision-free low-level paths for each transporting agent. This approach coordinates item transfers and agent movements while respecting their heterogeneous capabilities. We evaluate our approach in simulated environments inspired by practical real-world scenarios.

KEYWORDS

Multi-Agent Pickup and Delivery; Heterogeneous Agents; Planning and Coordination

ACM Reference Format:
Benedetta Flammini, Francesco Amigoni, and Bruno Lacerda. 2026. Multi-Agent Pickup and Delivery with Heterogeneous Agents. In *Proc. of the 25th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2026), Paphos, Cyprus, May 25 – 29, 2026*, IFAAMAS, 9 pages. https://doi.org/10.65109/GKGU6726

1 INTRODUCTION

Algorithms for coordinating teams of heterogeneous robotic agents are fundamental in many real-world situations where diverse capabilities are required to complete complex delivery tasks. Typical applicative domains include logistics and construction sites, where robots with different payloads and mobility ranges cooperate to move items between storage, packing, and shipping areas, and healthcare environments, where robotic teams can assist in the timely delivery of medical supplies and meals to patients. The Multi-Agent Pickup and Delivery (MAPD) [16] problem involves coordinating multiple agents that must transport items between pickup and delivery locations while avoiding collisions and optimizing efficiency in settings in which tasks appear dynamically.

In this paper, we study a version of the MAPD problem in which agents belong to different classes, characterized by distinct item-handling capabilities. Each agent is restricted to an operable zone, i.e., a region of the environment where it can move and perform tasks (Figure 1). Since these operable zones may be disjoint or only partially overlapping, a single agent cannot always complete an entire pickup and delivery task on its own. As a result, items often need to traverse multiple zones, requiring transfers between agents at the boundaries of their zones. Agent heterogeneity and the need for inter-zone handovers introduce temporal and coordination challenges that are not present in classical homogeneous MAPD.

To address these challenges, we propose a two-level planning framework, similar to those presented in [3] and [11] to address different problems. At the high level, when a task appears (Figure 1a), the framework determines high-level paths to transfer items across operable zones, specifying the sequence of zones and frontier points through which each item must pass (Figure 1b). At the low level, our framework assigns the resulting pickup and delivery subtasks to agents within each zone and generates collision-free spatio-temporal paths for the assigned agents, while ensuring that inter-zone handovers are feasible (Figure 1c). We extend the Token Passing (TP) algorithm [13], a standard approach for MAPD, to handle both inter-zone transfers and intra-zone deliveries.

We make three main contributions: (i) we formalize MAPD with heterogeneous agents and operable zones; (ii) we present a hierarchical algorithm that computes high-level paths for the items and coordinates intra-zone subtask assignment and collision-free low-level paths; and (iii) we empirically validate the approach in simulated environments inspired by realistic scenarios, demonstrating the effective coordination of heterogeneous agents under dynamic task arrivals.

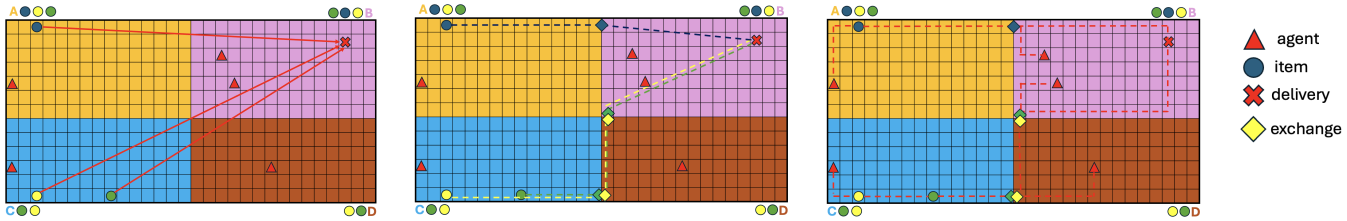
2 BACKGROUND

2.1 MAPD

The Multi-Agent Pickup and Delivery (MAPD) problem [16] involves a set of n agents operating in an environment represented by an undirected connected graph $G = (V, E)$, where vertices V denote locations and edges E their connections. Time is discrete, and, at each time step, every agent executes an action. Two types of actions are allowed: if an agent is at vertex $v \in V$ at time t, at time $t + 1$ it can either wait in v or move to an adjacent vertex $v' \in V$ such that $(v, v') \in E$. Each action requires one time step.

Tasks are represented by a set $\mathcal{T}$, which contains all unassigned tasks. Due to the dynamic nature of MAPD, new tasks may be added to $\mathcal{T}$ over time. Each task $\tau_j = (s_j, d_j) \in \mathcal{T}$ consists of a

Proc. of the 25th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2026), C. Amato, L. Dennis, V. Mascardi, J. Thangarajah (eds.), May 25 – 29, 2026, Paphos, Cyprus. © 2026 International Foundation for Autonomous Agents and Multiagent Systems (www.ifaamas.org). https://doi.org/10.65109/GKGU6726



(a) A new task appears that requires a blue, a yellow, and a green item to be delivered to the cross location.

(b) High-level items' paths (dashed lines) composed of sequences of pickup, delivery, and exchange locations.

(c) Agents' paths (dashed red lines) for the sequences of pickup and delivery subtasks in which the high-level items' paths can be divided.

Figure 1: Example of multi-agent pickup and delivery with heterogeneous agents. The agents operating in the environment are divided into 4 classes (A, B, C, and D). Each class of agents can only operate in a zone (orange for A, pink for B, light blue for C, and brown for D) and handle some types of items (represented next to the class name).

pickup location $s_j \in V$ and a delivery location $d_j \in V$. An agent is considered occupied if it has an assigned task, and free otherwise. Free agents can be assigned any task in $\mathcal{T}$, subject to the constraint that a task can only be assigned to one agent. An occupied agent becomes free upon completing its assigned task.

To complete a task $\tau_j = (s_j, d_j)$, an agent moves from its current location to the pickup location s_j, and then proceeds to the delivery location d_j. Thus, accomplishing a task requires an agent a_i to plan and execute a sequence of actions (a path) π_i that brings it from its current position to s_j and subsequently to d_j. Collision avoidance is required: two agents cannot occupy the same vertex at the same time step (vertex conflict), nor traverse the same edge in opposite directions at the same time step (edge or swapping conflict).

The objective in MAPD is to plan paths that complete all tasks in minimal time. Solution quality is typically measured using either service time, the average number of time steps needed to complete a task from its arrival in $\mathcal{T}$, or makespan, the total number of time steps required to complete all tasks in $\mathcal{T}$ (assumed finite).

Not all MAPD instances are solvable. A sufficient condition for solvability is the instance being well-formed [16]. To formalize this, we introduce non-task endpoints, which are locations where agents can remain indefinitely without obstructing others. In contrast, task endpoints correspond to all pickup and delivery locations. A MAPD instance is well-formed if: (1) the total number of tasks is finite, (2) the number of non-task endpoints is at least equal to the number of agents, and (3) for every pair of endpoints, there exists a path connecting them that does not pass through any other endpoint.

2.2 Token Passing

Ma et al. [16] propose both centralized and decentralized algorithms for solving well-formed MAPD instances. Centralized methods yield lower service time and makespan but require more computation, while decentralized ones are less effective yet more efficient.

One notable decentralized approach is Token Passing (TP) [13], in which each agent autonomously assigns itself to tasks and plans collision-free paths while leveraging some global knowledge of the environment and other agents. This global knowledge is encapsulated in a token, a synchronized memory block shared among agents that contains the current task set $\mathcal{T}$, task assignments, and agents' currently planned paths π_i.

In TP, an agent a_i assigned to a task τ_j employs a path planner (e.g., A* or Dijkstra) to compute minimum-cost collision-free paths in a state space where states are pairs of locations and time steps. An edge exists between state (v, t) and state $(v', t+1)$, with $v, v' \in V$, if either $(v, v') \in E$ or $v = v'$. States edges that would result in a vertex edge conflict with other agents according to their current paths in the token are removed from the state space set of edges.

Initially, each agent is assumed to remain at its starting location. At each time step, new tasks may be added to $\mathcal{T}$. Each free agent requests access to the token once per time step; the system grants it to requesting agents sequentially. An agent with the token assigns itself to the task with the pickup location closest to its current position according to a heuristic h (e.g., Manhattan distance in the case of grid environments), provided no other agent is currently assigned to a task with the same pickup or delivery locations. Once a task is assigned, the agent plans a minimum-cost path from its current location to the delivery location via the pickup location, ensuring that the path is collision-free with respect to the paths stored in the token. The agent returns the token and starts following the planned path. If no suitable task exists or no collision-free path can be found, the agent computes a path to a non-task endpoint.

3 RELATED WORK

3.1 MAPD variants

MAPD extends Multi-Agent Path Finding (MAPF) [20] (which, in a sense, is a one-shot problem) by incorporating dynamically appearing tasks and temporal considerations that go beyond static path planning. Unlike MAPF, which focuses on computing collision-free paths for agents moving from their start locations to pre-assigned goals, MAPD requires agents to not only plan collision-free paths but also handle dynamically generated pickup and delivery tasks, effectively combining path planning with task assignment.

Several MAPD variants have been proposed: for example, Ma et al. [15] present a variant that takes into account kinematic constraints and computes continuous paths for agents with given velocities. In [8], TP is modified to consider task appearance probabilities,

while in [14] TP is extended to take into account possible agents delays. In [10], MAPD is extended to allow agents to be allocated multiple tasks simultaneously according to their payload capacity, as well as to consider pickup and drop-off costs. Similarly, in [4, 5, 7, 12], robots can perform more than one task at the same time, meaning that they can pick up multiple items simultaneously. Another formulation of MAPD is that of Multi-Goal Multi-Agent Pickup and Delivery (MG-MAPD) [22], in which each task is composed of an ordered sequence of delivery locations.

To the best of our knowledge, we are the first to address MAPD with heterogeneous agents. The closest related work is [6], which studies the Online Pickup and Delivery Problem (PDP) with transfers. In that setting, the goal is to deliver items as quickly as possible under agent capacity constraints, with the added assumption that agents can transfer objects between them. Unlike our scenario, however, agents in [6] can move freely throughout the environment, and exchanges can occur at any location.

3.2 Collaborating Heterogeneous Agents

In heterogeneous multi-robot systems, agents with different capabilities often need to cooperate to accomplish tasks [21]. A widely studied scenario involves teams of UAVs and UGVs, where UAVs follow task schedules and coordinate with UGVs for recharging [2, 19], or where one class supports the other in missions such as search and rescue [18] or parcel delivery [1]. Another related concept is that of marsupial robot systems [9, 17], in which carrier robots transport and deploy passenger robots to overcome terrain limitations or perform tasks requiring different capabilities.

Our formulation shares some similarities with these settings, such as the presence of multiple robot classes that must coordinate meeting times and locations. However, it differs in a key aspect: agents are not strictly dependent on each other to complete tasks. Unlike UAV-UGV or marsupial systems, where one robot class either supports or enables the actions of the other, MAPD with heterogeneous agents requires all agents to actively decide when and where to meet in order to complete their assigned tasks.

4 PROBLEM FORMULATION

We consider an environment modeled as a four-connected grid, more generally represented by an undirected graph $G = (V, E)$. Time is considered discrete, with time steps t. Going beyond classical MAPD, where agents are assumed to be homogeneous, we consider the *MAPD problem with heterogeneous agents*.

We assume the existence of a finite set Λ of item *categories*, where each $\lambda \in \Lambda$ represents a specific type of object that requires distinct handling capabilities. $\lambda(i)$ is the category of item i. Agents differ in their capacity and eligibility to transport items of different categories. Formally, we consider a set $\mathcal{A}$ of heterogeneous agents. Each agent $a \in \mathcal{A}$ belongs to a *class* $C(a)$, which defines its capabilities and constraints. Multiple agents can belong to the same class, meaning that all agents in a given class share the same properties. Each class C is characterized by:

- An *operable zone* $G_C = (V_C, E_C)$, which defines the portion of the environment that agents of class C can navigate. Operable zones of different classes may partially or fully overlap.
- An *item constraint* $I_C : \Lambda \to \mathbb{N}$, which assigns to each item category the maximum quantity that agents in the class are allowed to transport. For instance, $I_C(\lambda) = 2$ means that an agent from class C can transport 2 items of category λ. This number could also be zero, indicating that the specific item category cannot be transported by the agents in that class.

Each agent $a \in \mathcal{A}$ belongs to exactly one class $C(a)$, namely its movement and item constraints are predefined and do not change over time. Agents can perform five possible actions: *wait* at their current location; *move* to an adjacent vertex; *pick up* an item; *deliver* an item; or *handover* an item to another agent located in an adjacent vertex. Note that item exchanges require agents to remain at adjacent locations for at least one time step. All actions cost (last) one time step.

Each item category $\lambda \in \Lambda$ is mapped to a set of vertices $V_\lambda \subset V$ through the function Λ_L, which describes the locations from which items of that category can be picked up. For example, $\Lambda_L(\lambda) = \{v, v'\}$ means that items of category λ can be equivalently picked up from v or v'. We assume that all vertices in $\Lambda_L(\lambda)$ have an indefinitely large storage of items of category λ.

We consider an online finite stream of tasks, meaning tasks are dynamically added to the task set $\mathcal{T}$ and their locations and appearance times are not known in advance. Each task $\tau \in \mathcal{T}$ requires the transportation of a set of items, which can belong to different categories, to a single designated delivery location (not necessarily at the same time step). Formally, a task is defined as a tuple $\tau = (t_\tau, d_\tau, I_\tau)$ where t_τ is the appearance time, d_τ is the drop-off location, and I_τ is the set of items that must be transported.

Since items in I_τ may be located in different operable zones, completing a task generally requires the cooperation of multiple heterogeneous agents. Thus, for each task $\tau = (t_\tau, d_\tau, I_\tau)$, every item $i \in I_\tau$ must be transported from one of its pickup locations $p_\lambda \in \Lambda_L(\lambda(i))$ to the task's designated drop-off location d_τ. Due to the heterogeneity of agents and the potential disjointedness of their operable zones, the transportation of an item may require a sequence of handovers between agents belonging to different classes and operating in different zones. These inter-zone transfers are necessary to ensure that each item reaches its destination, particularly when no single agent class has full operability across any path between the item's pickup(s) and delivery locations.

Tasks are not assigned to individual agents as atomic units. Instead, a single task may be fulfilled collaboratively by multiple agents, potentially belonging to different classes. Moreover, the items associated with a task may be delivered at different times, i.e., partial deliveries are allowed. Note that rather than modeling each item category as a separate task, we group related items of different categories under a common request. This reflects realistic scenarios, such as grocery delivery (see Example 4.1.2), where multiple different items must be delivered jointly to fulfill a single customer order.

The objective of this problem is to compute a set of collision-free paths $\{\pi_i\}$ (one for each agent a_i) that allow agents to complete all the tasks. A collision occurs when multiple agents attempt to occupy the same location or traverse the same edge simultaneously. Specifically, a vertex collision occurs if two agents a_i and a_j occupy the same location v at the same time step t, i.e., $\pi_i(t) = \pi_j(t) = v$.

An edge collision arises when two agents traverse the same edge $e = (v, v')$ in opposite directions at the same time step t, formally expressed as $\pi_i(t) = v, \pi_i(t+1) = v'$ and $\pi_j(t) = v', \pi_j(t+1) = v$. Collisions have to be prevented for both agents in the same class and in different classes, if their operable zones overlap.

The quality of a solution can be evaluated using different metrics, such as (i) the makespan, which quantifies the number of time steps required to complete all tasks (assumed to be finite), and (ii) the service time, which refers to the average number of time steps to complete a task from its appearance.

4.1 Examples

4.1.1 Medical Supply Distribution in Hospitals. Consider a scenario where a hospital requires the delivery of medical supplies to various drop-off locations. Items are initially stored in a dedicated area. Two agent classes handle the delivery process:

- Suppliers (C_S): these agents operate within designated supply zones and are responsible for picking up items from storage locations and bring them to intermediate transfer locations. Their bigger size allows them to transport multiple items simultaneously.
- Deliverers (C_D): these agents are responsible for transporting items from intermediate transfer locations to their final drop-off locations. Their smaller size enables them to maneuver through tighter spaces, making them better suited for final deliveries in constrained environments, but they can transport only one item at a time.

All item categories can be handled by both suppliers and deliverers. Suppliers retrieve items from storage and transfer them to deliverers, that perform the final delivery to the destinations.

4.1.2 Online Grocery Shopping. In an online grocery shopping scenario, a team of heterogeneous agents is responsible for collecting and assembling customer orders within a supermarket. Each order (task) consists of a set of grocery items that must be gathered from different aisles and delivered to a designated packing station.

We distinguish different classes of agents, which depend on the specific aisle in which they operate (e.g., fruit, beverages, meat) and are responsible for managing items in their assigned section. Agents in different aisles belong to distinct classes, meaning they differ in the categories and quantities of items they can transport.

4.1.3 Construction Sites. In a construction site scenario, a team of heterogeneous mobile robots delivers materials and equipment between storage areas, work zones, and assembly points. Robots differ in payload capacity and operational range, and each is assigned to specific material types (i.e., item categories) and can traverse different terrains. Tasks require coordination between multiple transport robots to ensure materials are delivered efficiently.

5 PROPOSED ALGORITHM

5.1 Overview

Figure 2 reports a high-level overview of our algorithm for solving MAPD with heterogeneous agents. The core idea of the approach is to decompose the MAPD problem with heterogeneous agents into a set of classical MAPD subproblems that can be solved in parallel.

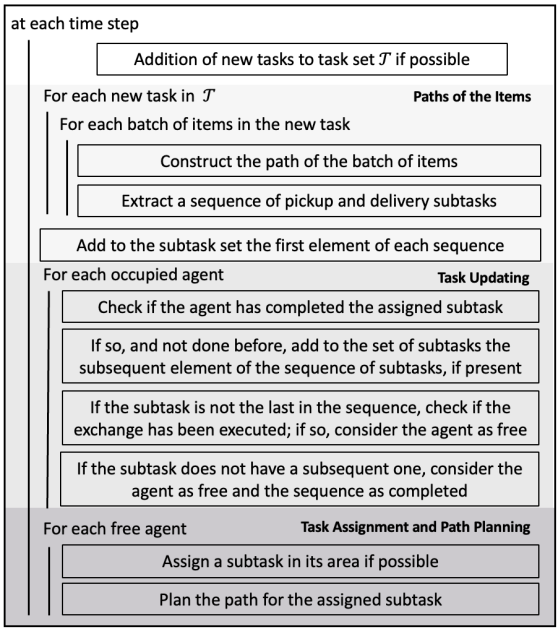


Figure 2: Algorithm schema.

To achieve this, we need to reshape the tasks by transforming them into multiple sequences of classical pickup and delivery subtasks (Section 5.2). A task τ, as defined in Section 4, consists of a collection of items belonging to different categories that must all be delivered to a common destination. For each task $\tau = (t_\tau, d_\tau, I_\tau)$, we consider each involved item category separately, constructing a high-level path that transports the items of that category from their pickup locations (chosen from $\Lambda_L(\lambda)$ for the items of category λ) to the delivery location d_τ. This process involves identifying all the operable zones that such items must traverse and determining the exchange points between these zones. Traversing an operable zone means that an agent operating within that zone must pick up the items from one exchange location (or from a pickup location) and deliver them to another (or to the delivery location). Therefore, building the high-level paths for the items and identifying the corresponding exchange locations leads to defining a sequence of pickup and delivery subtasks, one for each operable zone to be visited. From the perspective of the agents within each operable zone, this setup resembles that of a classical MAPD problem, with the additional constraint that items are not simply delivered, but are handed off to another agent for pickup. Once the available pickup and delivery subtasks in each operable zone are updated, agents are assigned subtasks if possible (Section 5.3.1) and then plan paths to complete their assigned subtasks (Section 5.3.2).

5.2 Paths of the Items

We now detail how to find a high-level path for the items of a generic task $\tau = (t_\tau, d_\tau, I_\tau) \in \mathcal{T}$.

5.2.1 Task Feasibility. To consider a task τ as solved, all the items I_τ have to be delivered to the delivery location d_τ. However, not all of the items of a task will follow the same path, since they have

different pickup locations, and they may not be able to traverse the same operable zones due to the different capabilities of the agents. Thus, given a task τ, we aim to find a path for the items of category λ involved in the task. We do it sequentially, by considering the different item categories involved in a task.

The objective is to determine a path for the items in I_τ to reach the delivery location d_τ. The set I_τ is represented as a vector of length $|\Lambda|$ such that its entries specify the number of items to be delivered for each item category λ. For instance, $I_\tau(\lambda) = 3$ means that the task τ requires to deliver 3 items of category λ.

Each item category λ is associated with a set of possible pickup locations $\Lambda_L(\lambda)$. We randomly choose one to be considered as a pickup location p_λ. This implies that we need to find a path for the items of category λ that goes from p_λ to d_τ. The first step is to assess whether the required batch of items of category λ, which we denote B^λ can be transported by a single agent within the operable zone corresponding to the pickup location p_λ. If this condition is satisfied, we proceed to compute the path. Otherwise, B^λ must be partitioned into multiple batches, each involving a feasible number of items of category λ, and a separate path must be determined for each batch. (In the case no agent can pickup items of category λ from p_λ, another pickup location is selected from $\Lambda_L(\lambda)$.)

Specifically, we aim to create the minimum possible number of batches, since more batches would imply more exchanges between the agents of different zones. Thus, assuming that agents that can perform the pickup at p_λ can carry a maximum of $max_\lambda = \max_C I_C(\lambda)$ items for category λ (where the maximum is taken over the classes C whose agents can operate in the zone of the pickup location p_λ), and the task requires the transportation of n items ($n = I_\tau(\lambda)$), if $n \leq max_\lambda$, then a single batch B^λ of items is formed. Otherwise, if $n > max_\lambda$, we define the number of batches as $g = \left\lceil \frac{n}{max_\lambda} \right\rceil$. Let $q = \left\lfloor \frac{n}{g} \right\rfloor$ and $r = n \bmod g$. Here, q represents the base batch size, while r is the number of items that remain after evenly distributing q items per group. We define the size of batch ℓ, denoted B^λ_ℓ, as $|B^\lambda_\ell| = \begin{cases} q+1 & \text{if } \ell \leq r, \\ q & \text{if } \ell > r, \end{cases} \quad \ell = 1, \ldots, g$. In this way, we obtain the minimal number of batches with a balanced partition of the items while respecting the constraints of the agents on the maximum number of items.

5.2.2 High-Level Path of an Item Category. The previous step guarantees that a batch of items B^λ can be transported by an agent within the corresponding pickup operable zone. Now, we aim to find a high-level plan that describes the operable zones that have to be visited to reach the delivery location for items in the batch B^λ. The high-level planning graph $G^{hl}_{B^\lambda}$ is constructed as follows, starting from the environment graph G.

The nodes in $V^{hl}_{B^\lambda}$ represent the operable zones in which the items of category λ of B^λ can be transported. Thus, each operable zone G_C is represented by a single node v_C, and the node is included in $V^{hl}_{B^\lambda}$ if the item capacity for the agents in G_C for item category λ is greater than 0 (i.e., $I_C(\lambda) > 0$). An edge between two nodes v_A and v_B is present if the corresponding operable zones are adjacent, i.e., if there exists at least an edge (u, v) in the environment graph G where $u \in V_A$ and $v \in V_B$.

Our aim is to find the high-level path with the minimum number of exchanges among agents, which is the path that visits the minimum number of operable zones. Thus, we simply find the shortest path between the operable zone where the pickup location is contained and the operable zone containing the delivery location. In this way, we obtain the ordered sequence of operable zones that the items in the batch have to visit to reach the delivery location.

5.2.3 Planning Graph of an Item Category. Once we have the sequence of operable zones representing the shortest path found on $G^{hl}_{B^\lambda}$ for a batch B^λ, we need to define the exchange locations, i.e., those points where the agents exchange the items. To do so, we build a tailored planning graph for the items of the batch B^λ.

Let us suppose that the high-level shortest path for the batch B^λ is $[v_X, v_Y, v_Z]$, corresponding to the operable zones G_X, G_Y, G_Z. The node set V_{B^λ} of G_{B^λ} consists of two types of nodes:

- Pickup and delivery locations: they represent the location where the items of the batch B^λ have to be picked up (p_λ) and their final delivery location d_τ.
- Frontier nodes: for each operable zone G_C included in the high-level path of the items, we consider all those cells $v \in V_C$ that are adjacent to at least one cell $u \in V_A$ belonging to a different operable zone G_A such that v_A belongs to the high-level shortest path and v_A and v_C are adjacent. In our example, it implies considering all the nodes of v_X adjacent to v_Y, all the nodes of v_Y adjacent to v_X, all the nodes of v_Y adjacent to v_Z, and all the nodes of v_Z adjacent to v_Y.

The edge set E_{B^λ} of G_{B^λ} is defined as follows:

- Task–frontier edges: the pickup p_λ and the delivery d_τ locations are connected to the frontier nodes of the operable zones to which they belong. In our example, since p_λ is contained in v_X, p_λ will be linked to all the frontier nodes belonging to v_X. Similarly, for d_τ and v_Z.
- Same frontier–frontier edges: each frontier node is connected to the other frontier nodes within the same operable zone, provided they are not adjacent to the same neighboring zone. This implies that the frontier nodes of v_Y adjacent to v_X are connected to the frontier node of v_Y adjacent to v_Z.
- Different frontier–frontier edges: each frontier node is connected to all the frontier nodes belonging to the next operable zone along the high-level shortest path, provided they are adjacent in G.

Each frontier node is associated with a weight vector whose length equals the number of item categories. The λ-th entry of this vector specifies the number of items of category λ that have to be delivered and/or picked up at that frontier node.

5.2.4 Planning the Items' Path. When planning paths for items of a category, the objective is to exploit shared deliveries across different item categories, allowing a single agent to deliver multiple items at the same location. However, once a subtask to deliver items of some category is assigned and the agent has planned its path, that path becomes fixed. Therefore, it is not convenient to select exchange locations that are part of already assigned subtasks, as these locations are already occupied by other agents, and it is not known when such locations will become free, since it depends on the time of the exchange. In addition, if the quantity of a given

item category assigned to a location exceeds the agents' capacity, this strategy becomes inefficient. Therefore, it is necessary to account for these constraints by leveraging the weights of the frontier nodes: priority is given to nodes where no deliveries or pickups are currently scheduled, but where available subtasks have a delivery or pickup location, while ensuring feasibility with respect to the agent's capacity. In this way, when assigning subtasks, the same agent can consider multiple subtasks with the same pickup or delivery simultaneously. Note that we prefer to predefine fixed exchange locations as dynamically optimizing them would require prior knowledge of agent–task assignments and availability, effectively necessitating a complete task schedule for all agents and substantially increasing coordination complexity.

In determining the high-level path of items of a category, which is a sequence of frontier nodes, we would like to find the shortest path in terms of number of time steps and, at the same time, favor nodes where not assigned subtasks have pickup or delivery locations, while penalizing those of already assigned subtasks and respecting item capabilities constraints. To combine these objectives, we define the weight w_{kj} of the edge that connects nodes k and j in V_{B^λ}:

$$w_{kj} = dist(k, j) + penalty(j), \tag{1}$$

where $dist(k, j)$ is the Manhattan distance between node k and node j in G, and $penalty(j)$ is the factor that accounts for the occupancy of node j. If node j has already assigned subtasks, or node j is free but the inclusion of the number of items of category λ leads to not respecting the constraint capabilities for the agents of that class, $penalty(j)$ is a positive integer number, otherwise, if node j is free and does not belongs to already assigned subtasks, is equal to -1. In all the other cases, $penalty(j)$ is equal to 1. Note that for different frontier–frontier edges, the distance is always 1.

Given the graph G_{B^λ} with the weighted edges, we use the Dijkstra algorithm to find the shortest path. After computing the path for the first batch of items, the weight function associated with the frontier nodes is updated to incorporate the newly assigned deliveries. This procedure is then repeated for all remaining batches and item categories within the task, and subsequently extended to all available tasks. The output of this process for a single task is a collection of sequences $\Sigma = \{\sigma_1, \sigma_2, \ldots, \sigma_T\}$ (one for each batch of items) with $\sigma_k = \big((p_1, d_1), \ldots, (p_{m_t}, d_\tau)\big)$, where each pair (p_j, d_j) corresponds to a pickup–delivery operation in a distinct operable zone. These sequences are subject to precedence constraints: a new pickup location becomes available to the agents only after the items have reached the preceding delivery location. Thus, pickup–delivery pairs must be executed in order and are revealed to the agents incrementally at runtime.

Note that, when building the items' path, we also need to consider if the quantity of items in B^λ is feasible across all traversable operable zones. If not, the items are again divided into feasible batches, and the pickup and delivery subtasks are replicated for each batch.

5.3 Task Assignment and Path Planning

5.3.1 Task Updating and Assignment. Once paths have been determined for all tasks newly added to $\mathcal{T}$, the next step is to update completed tasks and subtasks and assign those that are still available. A task $\tau \in \mathcal{T}$ is considered completed only when all items in I_τ have been delivered to d_τ, meaning all subtask sequences associated with these items are finished. Completion is verified by checking whether an agent has finished its assigned subtask. If it is the final subtask in its sequence, we consider the agent as free, and we then check whether all other sequences for items in I_τ are also complete; if so, τ is marked as completed. If the subtask is not the last in its sequence and the next subtask is not yet in the MAPD subtask set, it is added and becomes available to agents in the corresponding operable zone. If the next subtask is already in the set, we verify whether the handover has occurred; if so, the agent is free.

After the available tasks, subtasks, and agents' status update, we can proceed to assign the subtasks. We assume that each agent can handle multiple subtasks simultaneously, subject to the item constraints of its class (i.e., $I_C()$). From the agents' perspective, operations within each operable zone can thus be modeled as a Multi-Goal Multi-Agent Pickup and Delivery (MG-MAPD) problem [22]. When assigning subtasks, all subtasks with the same delivery location are considered together. The algorithm then selects the most efficient combination of items, i.e., the one that maximizes the number of completed subtasks, and assigns it to the corresponding agent, respecting its capacity constraints.

Note that the duration of each delivery cannot be determined in advance due to the need for item transfers between agents in adjacent operable zones. For efficiency, the agent in the subsequent zone does not wait for the preceding agent to arrive; instead, it begins other subtasks and only proceeds to pick up the item once it becomes available. This issue does not arise for deliveries at the end of a sequence, where the item is dropped off at its final destination.

5.3.2 Path Planning. After task assignment, TP is combined with a spatio-temporal version of Dijkstra's algorithm to plan agents' paths. Agents share a single token that stores the paths of all other agents and use it to compute collision-free paths.

5.4 Algorithm Analysis

5.4.1 Agents Communication. For agent communication, we distinguish between agents inside the same operable zone and agents in different operable zones. In the former case, agents use TP to plan paths; thus, they share information through the token. Agents in different operable zones do not know each others' tasks and paths; however, if two zones are adjacent, agents may have to exchange items. Consider two zones, A and B, and an item that must be transported from A to B. An agent in A is responsible for carrying the item to a designated exchange location at the boundary between the two zones. Once the agent in A reaches this exchange location, a new task is generated in B, representing the continuation of the transportation from the exchange location to the final destination within B. Each operable zone has its own task manager, and new tasks are created through inter-zone communication between task managers (see [3] for a similar solution to improve the efficiency of TP for plain MAPD).

5.4.2 Computational Complexity.

High-Level Path for an Item Category. Given a batch of items B^λ for category λ, the first step is to construct a high-level path across operable zones. Let N denote the number of operable zones. Constructing the high-level planning graph requires checking adjacency

between zones, with at most $O(N^2)$ edges in the worst case. Computing the shortest path from the pickup to the delivery zone using the Dijkstra's algorithm has complexity $O(E + N \log N)$, where E is the number of edges in the high-level graph.

Planning Graph of an Item Category. After determining the sequence of operable zones, we build a planning graph G_{B^λ} consisting of the pickup and delivery locations, and the frontier nodes representing cells adjacent to other zones along the high-level path. Let F denote the total number of frontier nodes along the path. The edges of G_{B^λ} include task–frontier, same frontier–frontier, and different frontier–frontier edges, resulting in at most $O(F^2)$ edges. Finding the shortest path on this weighted graph using Dijkstra's algorithm has complexity $O(F^2 + F \log F)$.

Batch Handling. If the number of items exceeds the agent's capacity, the task is split into $g = \lceil n/max_\lambda \rceil$ batches. Each batch requires building its own high-level path and planning graph. Therefore, the complexity of processing all batches scales linearly: $O\big(g \cdot (E + N \log N + F^2)\big)$.

Task Assignment. For each agent, all subtasks sharing the same delivery location are considered together. The algorithm selects the combination of items that maximizes the number of completed subtasks while respecting agent capacities. If all subsets of subtasks are considered, the worst-case complexity is exponential in the number of subtasks T per delivery location $O(2^T)$.

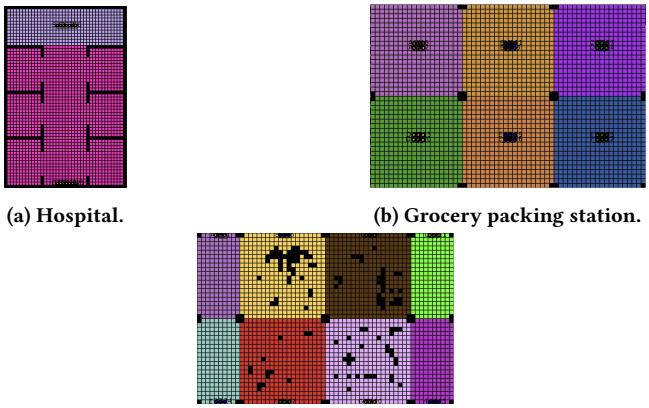
Agent Path Planning. After subtask assignment, agents compute collision-free paths using Token Passing combined with the spatio-temporal Dijkstra's algorithm. Let L denote the number of locations in the environment and $T_{\max}$ the planning horizon. The spatio-temporal graph has $O(L \cdot T_{\max})$ nodes, leading to a complexity per agent of $O(L \cdot T_{\max} \log(L \cdot T_{\max}))$.

Overall Complexity. Let A be the number of agents. Denoting by g the number of batches per task, F the total number of frontier nodes, T the tasks per delivery location, L the total number of locations, and $T_{\max}$ the planning horizon, the overall complexity can be expressed as $O\Big(g \cdot (E + N \log N + F^2) + 2^T + A \cdot L \cdot T_{\max} \log(L \cdot T_{\max})\Big)$. The dominant factors are the batch processing and frontier graph planning (F^2), the exponential search for task combinations (2^T, in our implementation, we set a 20 seconds timeout), and agent spatio-temporal path planning.

6 EXPERIMENTS

6.1 Experimental Setting

We test our approach on three different maps inspired to the examples of Section 4.1: a hospital corridor (Figure 3a, Example 4.1.1), a grocery packing station (Figure 3b, Example 4.1.2), and a construction site (Figure 3c, Example 4.1.3). Table 1 reports the experimental parameters for such environments. In all the environments, item pickups and deliveries are usually located at the border of the map. Task arrival times are randomly generated using a Poisson exponential inter-arrival distribution. Starting from $t = 0$, we sample the inter-arrival times Δt_i from an exponential distribution with rate μ ($\mu = 0.4$ in our experiments) and compute the absolute arrival times cumulatively: $t_i = t_{i-1} + \Delta t_i$. This ensures that task arrivals



(a) Hospital. **(b) Grocery packing station.**

(c) Construction site.

Figure 3: Maps of the environments used for the experiments.

	Hospital	Grocery	Construction
size	40 × 60	60×40	60×40
#classes	2	6	8
#agents	9 per class	6 per class	3 per class
#tasks	20	20	20
#item categ.	9	10	8

Table 1: Experimental parameters.

are random but follow a Poisson-like process, which is commonly used to model tasks appearing over time in dynamic environments. The item categories, item quantities, and delivery locations of each task are chosen randomly from predefined sets.

The considered evaluation metrics are: (i) makespan, (ii) service time (which could be seen as a proxy for path quality), and (iii) number of subtasks, representing how many MAPD subtasks (for the baseline) or MG-MAPD subtasks (for our approach) need to be serviced. To the best of our knowledge, there are no algorithms that deal with MAPD problems with heterogeneous agents. For this reason, we introduce a baseline based on TP, which is similar to our proposed approach, but instead of treating each subproblem as a MG-MAPD, it considers a classical MAPD, where agents can only execute a pickup and delivery subtask at a time. Results are averaged over 30 runs, each differing in task locations, item quantities, and item categories. Experiments were conducted on a machine equipped with an Intel(R) Xeon(R) Silver 4114 CPU running at 2.20 GHz and with 64 GB of RAM.

6.2 Experimental Results

Figure 4 reports the makespan, service time, and number of MAPD subtasks for the hospital environment (left column), grocery packing station (central column), and construction site (right column) for both our approach and the baseline. In all the environments and for all the metrics, our approach outperforms the baseline, which means that allowing agents to tackle multiple subtasks simultaneously leads to an increase in efficiency (smaller service time and makespan). In fact, the last row of the plot reports the number of

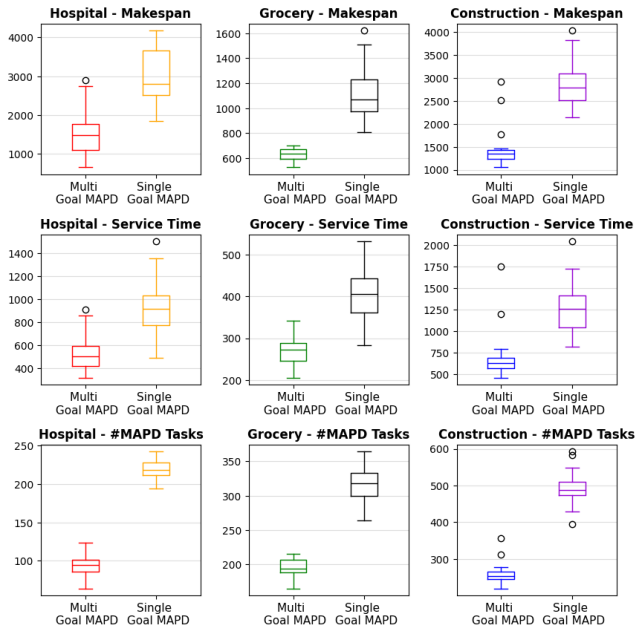


Figure 4: Makespan, service time, and number of subtasks in the tested maps.

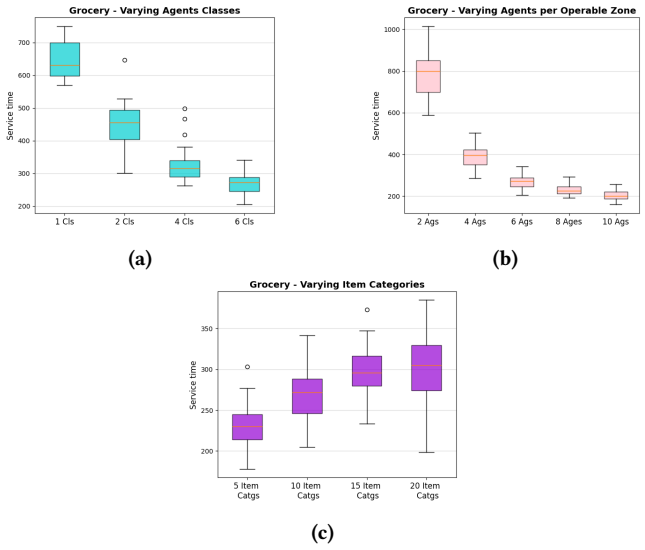


Figure 5: Service time in the grocery packing station map varying number of agents per class (Figure 5a), of agents per operable zone (Figure 5b), and of item categories (Figure 5c).

completed subtasks, considered as pickup and delivery tasks for the baseline and as multi-goal pickup and delivery tasks for our approach. The reduction of the serviced subtasks is remarkable: our approach approximately halves the number of subtasks, which results in halving the makespan and the service time.

In terms of computational time, our approach and the baseline show comparable performance in the grocery packing station (around 120 seconds per run) and construction site environments (around 500 seconds per run). In the hospital scenario, however, our approach is noticeably slower (about 500 seconds per run versus 200 seconds per run for the baseline). This difference arises from the environment's configuration: suppliers (see Example 4.1.1) are assumed to have high capacity across all item categories, requiring more frequent computation of feasible subtask combinations and thus increasing overall computational time.

To better assess the performance of our approach, we conducted additional experiments in the grocery packing station map, varying three key parameters: the number of agent classes (and corresponding operable zones), the number of agents per zone, and the number of item categories. Each parameter was varied independently, while the others were fixed as in Table 1. Figure 5 reports the service time obtained under these different conditions (similar trends are observed for the other environments). Figure 5a shows that increasing the number of agent classes reduces service time. This improvement reflects the benefits of specialization and greater parallelism among agents operating in different zones. Since the number of agents per class is fixed, the total number of agents also increases, meaning part of the gain is due to higher agent availability rather than heterogeneity alone. Figure 5b displays how the number of agents per class (and, consequently, per operable zone) affects service time. As more agents are added, service time decreases due to increased delivery capacity. However, the rate of improvement slows down with increasing agent numbers, reflecting a saturation effect: beyond a certain point, additional agents offer only marginal gains because most subtasks are already handled efficiently. Figure 5c shows that service time rises as the number of item categories increases. More item types add task complexity, leading to frequent transfers across zones, higher coordination overhead, and slower overall completion.

7 CONCLUSION

We presented a novel formulation of the Multi-Agent Pickup and Delivery (MAPD) problem involving heterogeneous agents with distinct capabilities and operable zones. Unlike classical MAPD, our setting requires agents to cooperate across classes and exchange items through handover operations at frontiers between zones. To solve this problem, we proposed a two-level planning framework that first computes high-level paths to transfer items across zones and then coordinates task allocation and collision-free path planning for individual agents. Experiments show that our method effectively manages inter-class cooperation and dynamic tasks.

Future work will explore scalability improvements and learning-based heuristics for selecting exchange locations and, more generally, for shortening task completion times, such as choosing a tailored pickup location when assigning tasks. Moreover, we will include other aspects that characterize heterogeneity of agents, like their velocity, and test our approach on more complex maps, like those with more obstacles at the boundaries of the operable zones.

ACKNOWLEDGMENTS

This paper is supported by PNRR-PE-AI FAIR project funded by the NextGeneration EU program.

REFERENCES

[1] Barbara Arbanas, Antun Ivanovic, Marko Car, Matko Orsag, Tamara Petrovic, and Stjepan Bogdan. 2018. Decentralized planning and control for UAV–UGV cooperative teams. *Autonomous Robots* 42, 8 (2018), 1601–1618.

[2] Ahmad Bilal Asghar, Guangyao Shi, Nare Karapetyan, James Humann, Jean-Paul Reddinger, James Dotterweich, and Pratap Tokekar. 2023. Risk-aware recharging rendezvous for a collaborative team of UAVs and UGVs. In *Proceedings of the International Conference on Robotics and Automation (ICRA)*. 5544–5550.

[3] Marcello Bavaro and Francesco Amigoni. 2025. Hierarchical Token Passing Algorithm for Multi-Agent Pickup and Delivery. In *Proceedings of the European Conference on Mobile Robots (ECMR)*.

[4] Zhe Chen, Javier Alonso-Mora, Xiaoshan Bai, Daniel D Harabor, and Peter J Stuckey. 2021. Integrated task assignment and path planning for capacitated multi-agent pickup and delivery. *IEEE Robotics and Automation Letters (R-AL)* 6 (2021), 5816–5823.

[5] Evren Çilden and Faruk Polat. 2022. Multiagent pickup and delivery for capacitated agents. In *International Conference on Practical Applications of Agents and Multi-Agent Systems (PAAMS)*. 76–87.

[6] Brian Coltin and Manuela Veloso. 2014. Online pickup and delivery planning with transfers for mobile robots. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*. 5786–5791.

[7] Antonello Contini and Alessandro Farinelli. 2021. Coordination approaches for multi-item pickup and delivery in logistic scenarios. *Robotics and Autonomous Systems* 146 (2021), 103871.

[8] Andrea Di Pietro, Nicola Basilico, and Francesco Amigoni. 2023. Multi-Agent Pickup and Delivery with Task Probability Distribution. In *Proceedings of the 2023 International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*. 2580–2582.

[9] Michail Kalaitzakis, Brennan Cain, Nikolaos Vitzilaios, Ioannis Rekleitis, and Jason Moulton. 2021. A marsupial robotic system for surveying and inspection of freshwater ecosystems. *Journal of Field Robotics* 38, 1 (2021), 121–138.

[10] Fumiya Kudo and Kai Cai. 2023. A TSP-Based Online Algorithm for Multi-Task Multi-Agent Pickup and Delivery. *Robotics and Automation Letters (RA-L)* 8, 6 (2023), 5910–5917.

[11] Kevin Leahy, Zachary Serlin, Cristian-Ioan Vasile, Andrew Schoer, Austin M Jones, Roberto Tron, and Calin Belta. 2021. Scalable and robust algorithms for task-based coordination from high-level specifications (scratches). *IEEE Transactions on Robotics (T-RO)* 38, 4 (2021), 2516–2535.

[12] Yifei Li, Hao Ye, Ruixi Huang, Hejiao Huang, and Hongwei Du. 2023. Multi-Load Agent Path Finding for Online Pickup and Delivery Problem. In *Proceedings of the International Computing and Combinatorics Conference (COCOON)*. 285–296.

[13] Minghua Liu, Hang Ma, Jiaoyang Li, and Sven Koenig. 2019. Task and path planning for multi-agent pickup and delivery. In *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*. 1152–1160.

[14] Giacomo Lodigiani, Nicola Basilico, and Francesco Amigoni. 2023. Robust Multi-Agent Pickup and Delivery with Delays. In *2023 European Conference on Mobile Robots (ECMR)*. 1–8.

[15] Hang Ma, Wolfgang Hönig, TK Satish Kumar, Nora Ayanian, and Sven Koenig. 2019. Lifelong path planning with kinematic constraints for multi-agent pickup and delivery. In *Proceedings of the AAAI Conference on Artificial Intelligence*. 7651–7658.

[16] Hang Ma, Jiaoyang Li, TK Kumar, and Sven Koenig. 2017. Lifelong multi-agent path finding for online pickup and delivery tasks. In *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*. 837–845.

[17] Robin R Murphy, Michelle Ausmus, Magda Bugajska, Tanya Ellis, Tonia Johnson, Nia Kelley, Jodi Kiefer, and Lisa Pollock. 1999. Marsupial-like mobile robot societies. In *Proceedings of the Annual Conference on Autonomous Agents (AGENTS)*. 364–365.

[18] Roberto G Ribeiro, Luciano P Cota, Thiago AM Euzébio, Jaime A Ramírez, and Frederico G Guimarães. 2021. Unmanned-aerial-vehicle routing problem with mobile charging stations for assisting search and rescue missions in postdisaster scenarios. *Transactions on Systems, Man, and Cybernetics: Systems* 52, 11 (2021), 6682–6696.

[19] Guangyao Shi, Nare Karapetyan, Ahmad Bilal Asghar, Jean-Paul Reddinger, James Dotterweich, James Humann, and Pratap Tokekar. 2022. Risk-aware UAV–UGV rendezvous with chance-constrained Markov decision process. In *Proceedings of the Conference on Decision and Control (CDC)*. 180–187.

[20] Roni Stern, Nathan Sturtevant, Ariel Felner, Sven Koenig, Hang Ma, Thayne Walker, Jiaoyang Li, Dor Atzmon, Liron Cohen, TK Kumar, et al. 2019. Multi-agent pathfinding: Definitions, variants, and benchmarks. In *Proceedings of the International Symposium on Combinatorial Search (SoCS)*, Vol. 10. 151–158.

[21] Kai M Wurm, Christian Dornhege, Bernhard Nebel, Wolfram Burgard, and Cyrill Stachniss. 2013. Coordinating heterogeneous teams of robots using temporal symbolic planning. *Autonomous Robots* 34, 4 (2013), 277–294.

[22] Qinghong Xu, Jiaoyang Li, Sven Koenig, and Hang Ma. 2022. Multi-goal multi-agent pickup and delivery. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 9964–9971.