Standby-Based Deadlock Avoidance Method for Multi-Agent Pickup and Delivery Tasks

Tomoki Yamauchi
Waseda University
Tokyo, Japan
t.yamauchi@isl.cs.waseda.ac.jp

Yuki Miyashita
Waseda University
Tokyo, Japan
y.miyashita@isl.cs.waseda.ac.jp

Toshiharu Sugawara
Waseda University
Tokyo, Japan
sugawara@isl.cs.waseda.ac.jp

ABSTRACT

The *multi-agent pickup and delivery* (MAPD) problem, in which multiple agents iteratively carry materials without collisions, has received significant attention. However, many conventional MAPD algorithms assume a specifically designed grid-like environment, such as an automated warehouse. Therefore, they have many pickup and delivery locations where agents can stay for a lengthy period, as well as plentiful detours to avoid collisions owing to the freedom of movement in a grid. By contrast, because a maze-like environment such as a search-and-rescue or construction site has fewer pickup/delivery locations and their numbers may be unbalanced, many agents concentrate on such locations resulting in inefficient operations, often becoming stuck or deadlocked. Thus, to improve the transportation efficiency even in a maze-like restricted environment, we propose a deadlock avoidance method, called *standby-based deadlock avoidance* (SBDA). SBDA uses *standby nodes* determined in real-time using the *articulation-point-finding algorithm*, and the agent is guaranteed to stay there for a finite amount of time. We demonstrated that our proposed method outperforms a conventional approach. We also analyzed how the parameters used for selecting standby nodes affect the performance.

KEYWORDS

Multi-agent pickup and delivery tasks; Multi-agent path finding; Deadlock avoidance; Decentralized robot path planning

1 INTRODUCTION

The use of *multi-agent systems* (MAS) for complex and enormous tasks in real-world applications has recently attracted considerable attention. Examples include transport robots in automated warehouses [34], autonomous aircraft-towing vehicles [18], ride-sharing services [12, 37], office robots [31], and delivery systems with multiple drones [10]. However, simply increasing the number of agents may lead to inefficiency owing to redundant movements and resource conflicts such as collisions. Therefore, to improve the overall performance, it is essential to have coordinated actions that avoid negative effects among agents. In particular, collision avoidance is essential in our application, which is a pickup and delivery system applied in a restricted environment where the agents are large heavy-duty robots carrying heavy and large-sized materials.

This problem is formulated as a *multi-agent pickup and delivery* (MAPD) problem, where pickup and delivery tasks are assigned to individual agents simultaneously. The assigned agent then needs to move to the material storage area, load a specified material, deliver it to the required location, and unload it. Therefore, because there are numerous tasks to fulfill, the MAPD problem can be considered as an iteration of *multi-agent path-finding* (MAPF), in which multiple agents generate collision-free paths to their targets without collisions. Unfortunately, the MAPF problem is known to be an NP-hard problem to obtain the optimal solution [17], and thus the MAPD problem is more time-consuming.

There have been many studies focusing on MAPF/MAPD problems [11, 13, 14, 16, 19, 21, 28], and their results have been used in real-world applications. With these applications, generating plans without collisions and deadlocks becomes the central issue. For example, Okumura et al. [21] proposed *priority inheritance with backtracking* (PIBT) in which the agents decide the actions within a short time window to avoid deadlocks based on priorities and through local communication. Ma et al. [16] proposed *token passing* (TP) with using the *holding task endpoints* (HTE) [11, 13], which exclusively holds the *endpoints* including the pickup/delivery nodes of the tasks, to avoid collisions. Liu et al. [13] introduced *reserving dummy paths* (RDP), which consistently reserve *dummy paths* to a unique parking location of each agent, allowing the agents to fulfill their tasks with the same endpoint simultaneously.

However, these methods cannot be used in our target environments. For example, to guarantee completeness, PIBT requires the environment to be bi-connected; however, our environment cannot meet this requirement. For efficient movement, HTE and RDP assume a grid-like environment, which has many endpoints where agents can stay for any time length and/or many detours whose lengths are almost identical. Such requirements are possible in specially designed environments such as an automated warehouse. However, our environments are *ad hoc* and maze-like, such as a search-and-rescue or construction site, and usually have fewer pickup and/or delivery locations whose numbers may be unbalanced, resulting in congestion near a few of the endpoints. Moreover, detours are limited and often become much longer.

Therefore, we propose a deadlock avoidance method, i.e., *standby-based deadlock avoidance* (SBDA), for improving the transportation efficiency while allowing agents to conduct their tasks with the same endpoint even in a maze-like environment. We integrate this method with TP although the number of endpoints is quite

small in our target environments. The SBDA algorithm employs *standby nodes*, where an agent is guaranteed to wait for a finite amount of time near its destination (i.e., its endpoint). The set of standby nodes changes while the agents reserve such nodes to stay there; however, they can be determined efficiently in real-time using the *articulation-point-finding algorithm* (APF algorithm) in the graph theory [30]. When an agent moves toward the endpoint where other agents have already arrived and/or are traveling as their destination, SBDA enables the agent to temporarily wait at one of standby nodes remaining near the endpoint to avoid becoming stuck or entering a deadlock and head toward its destination in turn. Herein, we evaluate the performance of our proposed method and compare it with that using HTE as a baseline under various experimental conditions. We then demonstrate that our proposed method outperforms the baseline method in maze-like restricted environments. Finally, we analyze the features of the proposed method by conducting experiments under various parameter settings and applying an ablation study.

2 RELATED WORK

Studies on the MAPF/MAPD problem have been approached from a variety of perspectives [7, 15, 27]. One main approach is centralized planning and scheduling [1–3, 13, 28]. For example, Sharon et al. [28] proposed the *conflict-based search* (CBS) algorithm for the MAPF problem. CBS is a two-stage search approach, consisting of a *low-level search* in which agents generate their paths independently, and a *high-level search* in which a centralized planner generates optimal collision-free paths by receiving local plans from all agents. CBS and its extensions are used in many other centralized planners [1–3, 13]. Although we can expect optimal solutions/paths for the MAPD instances, the computational cost of a centralized planner rapidly increases with an increase in the number of agents [16].

Decentralized methods are more scalable and robust, and thus many studies have been conducted from this perspective [16, 21, 32, 33]. However, because the plans are generated individually, such methods for the MAPD problem must have completeness as well as the ability to detect and resolve conflicts between them under certain restrictions. For example, Okumura et al. [21] proposed PIBT for the MAPF/MAPD problem, in which agents determine the next nodes in accord with their priorities through communication with local agents. PIBT is effective and has been extended for use in more general cases [22, 23]; however, the completeness is guaranteed only in bi-connected environments. Ma et al. [16] proposed TP in which an agent selects tasks whose endpoints have not yet been reserved by other agents, and generates its collision-free plan by exclusively referring to the token, i.e., a synchronized shared block of memory. However, many of these studies also assume that the environments are grid-like such that, to avoid collisions, there are many escape nodes and the same-length of detours between two points. However, applying them to our maze-like environments may lead to a reduced efficiency of transportation and planning owing to the small number of endpoints and few detours, whose lengths are quite different.

By contrast, some studies [5, 8, 9, 24, 25, 38] have also considered an application to maze-like environments. For example, Damani et

al. [5] proposed *pathfinding via reinforcement and imitation multi-agent learning - lifelong* (PRIMAL$_2$), a distributed reinforcement learning framework for a *lifelong* MAPF (LMAPF), which is a variant of the MAPF in which agents are repeatedly assigned new destinations. However, they assumed that tasks are sparsely generated at random locations, and thus, unlike our environment, no local congestion occurs. Although other studies in the area of trajectory planning and robotics also assume maze-like environments [6, 26, 29], our study differs because we aim at completing the MAPD instance efficiently in a restrictive maze-like environment.

Previous studies have focused on deadlock avoidance in the MAPD problem [11, 13, 16, 19], similar to our approach. For example, HTE [11, 13] method assumes that the environment has many endpoints where agents can remain for a finite length of time; otherwise, the performance will decrease because fewer tasks can be executed in parallel. By contrast, RDP [13] always reserves dummy paths to the unique parking location of each agent to allow the agents to conduct their tasks with the same endpoint simultaneously; hence, numerous detours whose lengths are almost identical are necessary for achieving efficiency. However, our environment is maze-like and has fewer endpoints, and thus the use of endpoints is limited [19]. It also has a limited number of detours whose lengths may differ considerably.

3 PRELIMINARIES
3.1 Problem Formulation

The MAPD problem consists of an agent set $A = \{1, \ldots, M\}$, a task set $\mathcal{T} = \{\tau_1, \ldots, \tau_N\}$, and an undirected connected graph $G = (V, E)$ embeddable in a two-dimensional Euclidean space. Node $v \in V$ corresponds to a location, and an edge $(u, v) \in E$ $(u, v \in V)$ corresponds to a path along which an agent can move between locations u and v. We can naturally define the *length* of edge (u, v) by denoting $l(u, v)$. The *distance* between nodes v_1 and v_2 is defined as the sum of the lengths of edges appearing in the shortest path from v_1 to v_2. We assume that an endpoint is set to a dead-end in G. Our agent is a heavy-duty forklift-like autonomous robot with a picker in front, which carries a heavy material (500 kg to 1 ton) and can pick up (load) or put down (unload) this material using a picker in a specific direction at a specific node. We introduce discrete-time $t \in \mathbb{Z}^+$, where $\mathbb{Z}^+$ is the set of positive integers.

For agent $i \in A$, we define the *orientation* $o_i^t \in \mathbb{Z}^+$ and moving *direction* $d_i^t \in \mathbb{Z}^+$ of i at time t, where $0 \le o_i^t, d_i^t < 360$ in D increments, and $o_i^t = 0$ and $d_i^t = 0$ indicate the northward orientation and direction of G. In addition, the set of possible orientations is denoted as $\mathcal{D}$. For example, if $D = 90$, $\mathcal{D} = \{0, 90, 180, 270\} \ni o_i^t, d_i^t$. We assume $D = 90$ for simplicity, whereas D can have any number depending on the environmental structure.

Agents can conduct the following actions: *move*, *rotate*, *wait*, *load*, and *unload* on any node. Using the length $l = l(u, v)$, the rotation angle $\theta \in \mathbb{Z}^+$, and the waiting time t, the durations of actions *move*, *rotate*, *wait*, *load*, and *unload* are denoted by $T_{mo}(l)$, $T_{ro}(\theta)$, $T_{wa}(t)$, T_{ld}, and T_{ul}, respectively. Suppose that i is on v at time t. By action *move*, i moves forward or backward along edge (u, v) to u after appropriately changing its orientation o_i^t through a rotation. By *rotate*, i rotates D degrees clockwise (D) or counterclockwise ($-D$) from o_i^t, i.e., $o_i^{t+T_{ro}(D)} = o_i^t \pm D$, at v. Agent i has a

unique parking node $park_i \in V$ [13], which is the starting location at $t = 0$, and returns and remains there if i has no tasks to execute. Parking nodes are expressed by the red squares in Fig. 2.

The task τ_j is specified by the tuple $\tau_j = (\sigma_{\tau_j}^{ld}, \sigma_{\tau_j}^{ul}, \phi_{\tau_j})$, where $\sigma_{\tau_j}^{ld} = (v_{\tau_j}^{ld}, o_{\tau_j}^{ld})$ ($\in V \times \mathcal{D}$) are the location and orientation when loading a material ϕ_{τ_j}, and $\sigma_{\tau_j}^{ul} = (v_{\tau_j}^{ul}, o_{\tau_j}^{ul})$ ($\in V \times \mathcal{D}$) are the location and orientation when unloading a material ϕ_{τ_j}. When an agent loads and unloads a material, it needs to be oriented in a specific direction, considering the direction of the picker. Agents have to complete all tasks in $\mathcal{T}$ without collisions or deadlocks and then return to their own $park_i$.

3.2 Token Passing

Although the proposed SBDA can be adapted to some existing MAPD algorithms, in this paper, we show that it can be integrated into TP to efficiently solve an MAPD instance. TP [16] is a well-known MAPD algorithm, in which agents choose tasks themselves and generate paths using the information in the token. A token is a synchronized shared memory block containing the current paths of all agents, tasks currently assigned agents, and the remaining tasks that are not assigned agents.

The set of endpoints V_{ep} ($\subset V$) of an MAPD problem consists of all *task endpoints* that are possible pickup/delivery locations of the tasks and *non-task endpoints* including the initial location (i.e., $park_i$) of each agent. TP assumes that any agent can stay at an endpoint for any finite length of time without blocking the movement of other agents for their current tasks. We denote the set of task endpoints by V_{tsk} ($\subset V_{ep}$). It is assumed that the locations of all endpoints are given to the agents in advance. Hence, when an agent chooses one task whose pickup/delivery locations are *open* endpoints, meaning that they do not appear as the endpoints of the executing plans in the current token, a collision-free path is generated by looking at the content of the token. HTE in TP then holds the task endpoints to avoid collisions.

Although not all MAPD instances are solvable, *well-formed* MAPD instances are always solvable [4]. For TP, an MAPD instance is well-formed if and only if (a) the number of tasks is finite, (b) there are not fewer non-task endpoints than agents, and (c) a path exists between any two endpoints that does not traverse other endpoints [16]. In well-formed MAPD instances, agents can move to non-task endpoints (e.g., $park_i$) at any time and stay there as long as necessary to avoid collisions with other agents. This action might be able to reduce the number of agents to avoid an overly crowded environment. Obviously, the assumption regarding the endpoints makes these requirements hold. However, agents cannot conduct tasks simultaneously if their endpoints are overlapped; hence, this approach reduces the efficiency considerably in maze-like environments such as our considered environment.

4 STANDBY-BASED DEADLOCK AVOIDANCE

4.1 Status Management Token

We propose a novel deadlock avoidance method, i.e., SBDA, to improve the transportation efficiency while allowing agents to conduct tasks having the same endpoints even in a maze-like environment. We introduce the *status management token* (SMT), which is

an extension of a token [16] and a *synchronized block of information* [35], to manage the status of the planning, agents, and standby nodes and to detect conflicts with other agents, where we define a conflict as a situation in which multiple agents occupy the same node $v \in V$ or cross the same edge simultaneously. SMT contains the *reservation table* (RT), the *task execution status table* (TEST), and the *standby-node status table* (SST). RT is the set of RT tuples $(v, [s_v^i, e_v^i], i)$, which are valid reservation data used to generate collision-free plans, where $[s_v^i, e_v^i]$ is the occupancy intervals of agent i for node v. When $e_v^i < t_c$ (t_c is the current time), the tuple has been expired and deleted from RT. TEST is the set of TEST tuples (τ, v, i), where τ is the task currently being executed by i, and v is the associated destination, which is the load or unload node specified in τ. Therefore, two TEST tuples (τ, v_τ^{ld}, i) and (τ, v_τ^{ul}, i) are added when i selects task τ, and when i arrived at v_τ^{ld} or v_τ^{ul}, the corresponding entry is removed from TEST. SST is used to manage the status of all standby nodes, which are dynamically referred to and modified by SBDA. The structure of SST is described in Section 4.2.

SMT is a sharable memory area, and similar to a token in TP, only one agent can exclusively access it at a particular time. Although SMT may incur a slight performance bottleneck, the movement speeds of the robots are not fast and thus the time required for an overhead owing to a mutual exclusion is negligible for a realistic number of agents (e.g., less than 30 agents in the experiment environments as shown in Fig. 2).

4.2 Finding Potential Standby Nodes

Standby nodes are intuitively nodes by which, similar to non-task endpoints in TP, an agent is guaranteed to wait for a finite length of time on its way to the endpoint of the assigned task; however, they can be identified dynamically. We assume that, except for parking nodes, non-task endpoints are not given in advance. Let us define an *articulation point* (AP) and a *potential standby node*, which will be used as a standby node when needed:

Definition 4.1. A node in graph G is an *articulation point* iff removing it and the associated edges will split the connected area of G and increase the number of connected components.

Definition 4.2. A *potential standby node* in graph G is a node that is neither an AP, an endpoint, nor a dead-end in G.

Let us denote the set of all potential standby nodes in $G = (V, E)$ by $\mathcal{S}_{psn}(G)$ ($\subset V$). The next proposition is obvious from this definition.

PROPOSITION 4.3. *If $G = (V, E)$ is a connected graph and $\mathcal{S}_{psn}(G) \neq \varnothing$, then the subgraph generated by eliminating any node v_{psn} in $\mathcal{S}_{psn}(G)$ from G is connected.*

Therefore, even if an agent remains at a potential standby node, other agents can generate paths to reach their destinations without passing that node. We therefore have the following:

COROLLARY 4.4. *Any agent reserving a standby node can remain there for a finite length of time without blocking the movements of other agents.*

The set of all potential standby nodes $\mathcal{S}_{psn}(G)$ is efficiently identified using the APF algorithm such as Tarjan's algorithm [30],

Algorithm 1 Task selection by agent i

1: **function** SELECTTASK(i)
2:      $\mathcal{T}'$ = Set of tasks satisfying Cond. 1.
3:      // where $\tau = (\sigma_\tau^{ld}, \sigma_\tau^{ul}, \phi_\tau)$, and $\sigma_\tau^{ld} = (v_\tau^{ld}, o_\tau^{ld})$
4:      **if** $\mathcal{T}' \neq \varnothing$ **then**
5:          $\tau^* \leftarrow \arg \min_{\tau \in \mathcal{T}'} dist(v_c^i, v_\tau^{ld})$ // v_c^i: current location
6:          $\mathcal{T} \leftarrow \mathcal{T} \setminus \tau^*$; **return** τ^*
7:      **else return** false
8:      **end if**
9: **end function**

whose time complexity is O($|V|+|E|$) [20]. Now, we define a *standby node*, where G_t is the modified subgraph of G at time t, as below.

Definition 4.5. The potential standby node $v \in \mathcal{S}_{psn}(G_t)$ is a *standby node* when agent $i \in A$ reserves v to remain there.

When the agent leaves standby node v, v is no longer in standby node. We denote the set of all standby nodes with a reserving agent as $\mathcal{S}_t = \{(v,i)) \mid i \in A \text{ reserves } v \text{ as standby node}\}$.

When agents make a reservation to stay at a number of standby nodes (how such a reservation is created is explained in Section 4.4), other agents are prohibited to pass through them. This means that the structure of graph G is temporally modified by eliminating the standby nodes (and the associated edges). This modified graph at time t is then denoted by $G_t = (V_t, E_t)$. Thus, the set of potential standby nodes in G_t is denoted by $\mathcal{S}_{psn}(G_t)$, which is also efficiently identified using the APF algorithm. Note that $G = G_0$.

Before agents start the tasks in an MAPD problem, SBDA generates the set of *associated potential standby nodes* $s(v_{tsk})$ for every endpoint v_{tsk}. For endpoint $\forall v_{tsk} (\in V_{tsk})$, SBDA calculates $s(v_{tsk})$ ($\subset \mathcal{S}_{psn}(G)$) using

$$s(v_{tsk}) = \{v_{psn} \in \mathcal{S}_{psn}(G) \mid dist(v_{psn}, v_{tsk}) \leq \alpha\},$$

where $dist(v_1, v_2)$ is the distance between v_1 and v_2, i.e., the length of the shortest path between them. Parameter α (≥ 0) is the limit of the distance to a standby node from the endpoint v_{tsk}. Note that it is possible that $s(v_{tsk}) = \varnothing$ and $s(v_{tsk}) \cap s(v'_{tsk}) \neq \varnothing$ for $v_{tsk}, v'_{tsk} \in V_{tsk}$. Similarly, the set of potential standby nodes for v_{tsk} at t is denoted by $s_t(v_{tsk}) = s(v_{tsk}) \cap \mathcal{S}_{psn}(G_t)$. Therefore, $s(v_{tsk}) = s_0(v_{tsk})$ is the set of the initial potential standby nodes for G ($= G_0$). We also define the set of potential standby nodes that are not included in the associated potential standby nodes as

$$\mathcal{S}_{psn}^c(G_t) = \mathcal{S}_{psn}(G_t) \setminus \bigcup_{v \in V_{tsk}} s_t(v).$$

An element in $\mathcal{S}_{psn}^c(G_t)$ is called a *free potential standby node at t*. Note that it is possible that $\mathcal{S}_{psn}^c(G_t) = \varnothing$.

Information regarding the standby nodes is stored in SST. SST at t consists of (1) the initial set of potential standby nodes, $\mathcal{S}_{psn}(G)$, (2) the initial set of all pairs of endpoints and their potential standby nodes $\{(v_{tsk}, s(v_{tsk})) \mid v_{tsk} \in V_{tsk}\}$, (3) the set of pairs of standby nodes and agents i that reserve such nodes in the form of $(v_{sn}, i) \in \mathcal{S}_t$, and (4) the *crowded list*, CL ($\subset A$), which is the set of agents i that temporally remain at a free potential standby node in $\mathcal{S}_{psn}^c(G)$.
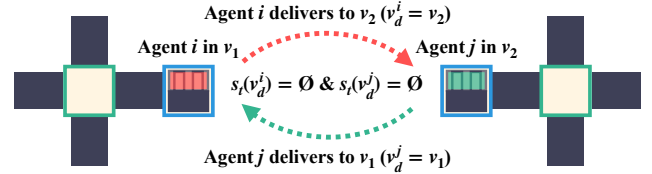

Figure 1: Deadlock state (i and j are on the same graph)

4.3 Task Selection Process

After the agents start the tasks in an MAPD problem, an agent with SBDA selects a task to conduct based on the potential standby nodes in SMT, decides its destination by selecting a standby node if necessary, and generates a path to it. The agent exclusively accesses the current SMT during this process.

The pseudo-code of the task selection process SELECTTASK(i) based on the potential standby nodes is shown in Algorithm 1. This is applied only when $\mathcal{T} \neq \varnothing$; otherwise, agent i returns to its parking node $park_i$. For $v_{psn} \in \mathcal{S}_{psn}(G_t)$, let $e_{v_{psn},t}^*$ be the last time that all agents will pass v_{psn} for all current plans. This can be calculated by looking at the elements $(v_{psn}, [s_{v_{psn}}^j, e_{v_{psn}}^j], j)$ in RT of SMT, and if such an element does not exist in RT, we set $e_{v_{psn},t}^* = t_c$, where t_c is the time when calculating $e_{v_{psn},t}^*$.

Agent i selects tasks $\tau = (\sigma_\tau^{ld}, \sigma_\tau^{ul}, \phi_\tau) \in \mathcal{T}$ that hold the following conditions (Cond. 1) at time t, where v_c^i is the current location of i, $\sigma_\tau^{ld} = (v_\tau^{ld}, o_\tau^{ld})$ and $\sigma_\tau^{ul} = (v_\tau^{ul}, o_\tau^{ul})$.

(1) If i remains at its parking node ($v_c^i = park_i$), the crowded list, CL, is empty.
(2) Its load node v_τ^{ld} is *open* (i.e., it does not appear as an endpoint in RT), or v_τ^{ld} has non-empty associated potential standby nodes (i.e., $s_t(v_\tau^{ld}) \neq \varnothing$) and $\exists v_{psn} \in s_t(v_\tau^{ld})$ such that $e_{v_{psn}}^* - t_c \leq \delta$.
(3) $|s_t(v_\tau^{ul})| + 1$ is greater than the number of tuples that appear in *TEST*, whose destination is v_τ^{ul}.

Here, because i cannot begin to stay at v_{psn} before $e_{v_{psn}}^*$, δ (≥ 0) is a threshold parameter of the margin time for reserving the standby node. The set of tasks that satisfies these conditions is denoted by $\mathcal{T}'$ [Line 2].

Then, i selects the closest task τ^*, i.e., $dist(v_c^i, v_{\tau^*}^{ld})$ in G_t is the smallest, where v_c^i is the current location of i [Lines 4–6]. If i cannot find such tasks, i returns to $park_i$. However, it is possible for i to occasionally check whether other agents have completed their tasks and for some endpoints/standby nodes to become open while i is going back and remains at $park_i$.

4.4 Use of Standby Nodes

After the previous process of task selection, agent i calls the path planning process to generate a path to move to v_τ^{ld} or v_τ^{ul} of the selected task τ. However, it is probable that these nodes are not open, and i may instead have to generate the path to the temporary destination. Thus, before generating a path, i calls function DECIDEDEST(i, v_d^i, v_c^i) in Algorithm 2 at current time t_c to decide the next actual destination, where v_d^i is the destination of i, which

---

Algorithm 2 Destination decision by agent i

1: **function** DECIDEDEST(i, v_d, v_c)
2:    // v_d: v_τ^{ld}, v_τ^{ul}, or $park_i$,
3:    // v_τ^{ld}: pickup, v_τ^{ul}: delivery, $park_i$: parking node
4:    **if** $i \in CL$ **then** remove i from CL
5:    **end if**
6:    **if** v_d satisfies one of Cond. 2, and v_d is open **then**
7:       **return** v_d
8:    **else if** $v_c \in s(v_d)$ **then return** v_c
9:    **else**
10:       $G_t^* = G_t \cup \{v_c\}$ // if $v_c \notin \mathcal{S}_{psn}$, $G_t = G_t^*$
11:       $V'_{psn}(G_t^*) = \{v \in \mathcal{S}_{psn}(G_t^*) \mid e_v^* - t_c \le \delta\}$
12:       **if** $S = V'_{psn}(G_t^*) \cap s(v_d) \ne \varnothing$ **then**
13:          $v_d \leftarrow \arg\min_{v \in S}(e_v^* - t_c)$
14:       **else if** $S' = V'_{psn}(G_t^*) \cap \mathcal{S}_{psn}^c(G_t^*) \ne \varnothing$ **then**
15:          $v_d \leftarrow \arg\min_{v \in S'} dist(v_d, v)$
16:          $CL = \{i\} \cup CL$
17:       **else**
18:          $v_d \leftarrow park_i$ // i returns to its parking nodes.
19:       **end if**
20:    **end if**
21:    **return** v_d
22: **end function**

---

is v_τ^{ld}, v_τ^{ul}, or $park_i$, and v_c^i is the current node of i, which is an endpoint, a standby node, or its parking node.

Before describing Algorithm 2, for $\forall i \in A$, we introduce the set of second conditions (Cond. 2) mainly to prevent a side entry for approaching the endpoint.

(1) $dist(v_c^i, v_d^i) \le \beta$, where β ($> \alpha$) is a threshold for directly heading toward the endpoint.
(2) No agent heads to an $s_t(v_d^i)$.
(3) $v_d^i == park_i$

If i can satisfy one of these conditions, it heads directly toward v_d^i if possible; otherwise, it heads toward a standby node of v_d^i. Note that β is also the threshold preventing other agents from being forced to wait too long at standby nodes owing to a cutting in line.

We briefly describe Algorithm 2. If v_d satisfies one of the conditions in Cond. 2, and v_d is *open*, i determines v_d as the destination [Lines 6–7]. Note that $park_i$ is always open. If v_c is a standby node of v_d, and v_d is not open, it returns v_c, remaining there for a while longer [Line 8]. If not, i tries to move a potential standby node of v_d as the temporary destination. Thus, it first calculates $V'_{psn}(G_t^*)$ [Line 11] by referring to RT and SST, where $G_t^* = G_t \cup \{v_c\}$ is a subgraph of G [Line 10]. If there is a potential standby node where other agents will pass through within a time of δ, the most appropriate node is selected [Lines 12–13] (see (2) in Cond. 1). If such a standby node does not exist, i instead tries to select another free potential standby node in $\mathcal{S}_{psn}^c(G_t^*)$. If it can select such a node, i is added to the crowded list, CL, demonstrating that it cannot select the element of $s(v_d)$ because the environment is crowded [Line 14–16]. Otherwise, it returns to its parking node [Line 18]. Note that when i selects and reserves the standby node as v_d [Lines 13

and 15], (v_d, i) is added to $\mathcal{S}_t$, whereas (v_c, i) is removed from $\mathcal{S}_t$ in SST if i leaves the current standby node [Line 7].

The temporal evacuation at a free potential standby node or parking node [Lines 14–18] allows an agent to avoid a deadlock, which occurs when agents i and j try to simultaneously exchange their unloading nodes and current locations and both nodes have no associated potential standby nodes. An example is shown in Fig. 1, in which two agents wait until their destinations become open or $s_t(v_d) \ne \varnothing$, and this situation cannot be solved if their destinations remain unchanged. However, when one agent sets another node as a temporary destination, one of the unloading nodes become open and another agent can start to move. After i arrives at standby node v_{sn} instead of the endpoint specified by τ, it invokes function DECIDEDEST to identify when i can move to the actual endpoint when i can obtain the right to access SMT. Then, agent i generates the path to the next destination by using a path finding algorithm, releases v_{sn}^i, removes the corresponding entry (v_{sn}^i, i) from $\mathcal{S}_t$, unlocks SMT, and finally, leaves v_{sn}.

Because only one agent can access SMT at a time, other agents that generate paths after agent i reserves standby node v_{sn} are prohibited to pass through v_{sn}. However, if j, which previously generates a path before agent i reserves v_{sn}, can pass through it, i must begin to wait at v_{sn} after j passes through v_{sn}; thus, i should reach v_{sn} with an appropriate delay by waiting somewhere, which may cause other delays to other agents. To reduce such a waiting time, the SBDA algorithm introduces Cond. 1 (2).

Finally, we prove that our proposed method is complete for our well-formed MAPD instances. In our SBDA, we modify the well-formed condition (c) of TP described in Section 3.2 as follows:

(c') When any agent generates a plan at time t, there exists a path between any two nodes of the endpoints or potential standby nodes at t that does not traverse other endpoints or standby nodes registered at t.

It is clear that the algorithm described thus far always satisfies condition (c'). In fact, the structure of graph G is temporarily modified when agent i selects a standby node to remain in from the potential standby nodes; however, another agent can always find a path to its destination without passing the standby nodes or endpoints in SBDA. This is because SBDA uses the APF algorithm to generate $\mathcal{S}_{psn}(G_t)$ when i selects the next task or a standby node instead of the destination in the task. Moreover, similar to TP, the agent exclusively accesses the current SMT and generates a path in turn. Therefore, we obtain the following theorem:

THEOREM 4.6. *SBDA solves all well-formed MAPD instances.*

PROOF. First, from functions SELECTTASK and DECIDEDEST, any agent will leave from any endpoint, $v_{ep} \in V_{ep}$, immediately after a load or unload task regardless of whether it has the next task. Suppose that agent i is now an endpoint or a standby node. If i has no task to conduct, i can generate a path to its parking node $park_i$. Otherwise, i heads to one of the endpoints, v_{ep}. If v_{ep} is open, i can generate a path to v_{ep}. If not, i selects a standby node or its parking node to head toward and generates a path to the selected node. Finally, i can remain at the reserved standby node or the parking node for any finite length of time (Corollary 4.4), and i can generate a path to v_{ep} if v_{ep} becomes open. □

Table 1: Parameter values used in the experiments

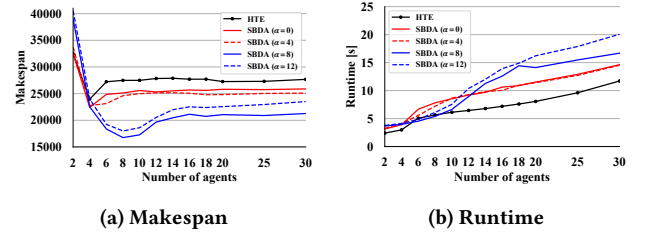| Description	Parameter	Value
No. of agents	M	2 to 30
No. of tasks	N	100
Orientation/direction increments	D	90
Duration of *move* per length 1	$T_{mo}(1)$	10
Duration of *rotate*	$T_{ro}(D)$	20
Durations of *load* and *unload*	T_{ld}, T_{ul}	20
Durations of *wait*	$T_{wa}(t)$	t
Margin time for reserving standby node	δ	100
Threshold for direct heading endpoint	β	20
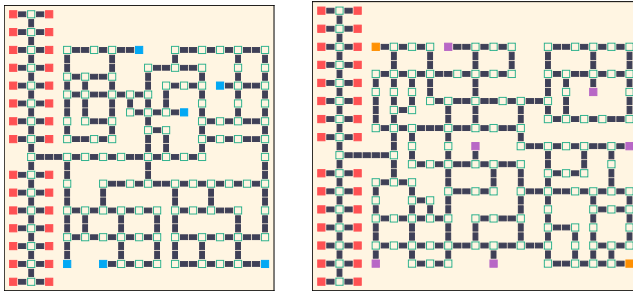
(a) Environment 1 **(b) Environment 2**

Figure 2: Experiment environments (red, parking nodes; blue, task endpoints; orange, pickup only locations; purple, delivery only locations; hollow green, nodes; black, edges)

(a) Makespan **(b) Runtime**

Figure 3: Comparison of SBDA and HTE (Env. 1)

(a) Makespan **(b) Runtime**

Figure 4: Comparison of SBDA and HTE (Env. 2)

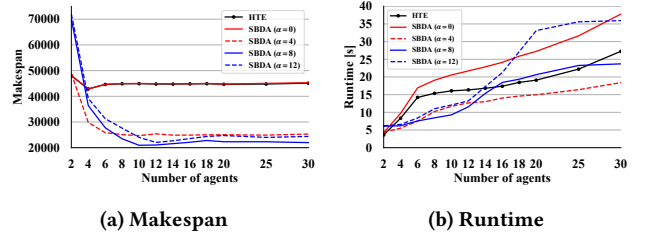# 5 EXPERIMENTS AND DISCUSSION

5.1 Experiment Setting

To evaluate the performance of our proposed method for executing the MAPD problem, we conducted the experiments under two different environments and compared the results with those using HTE as a baseline. In HTE, the agent selects the task whose loading and unloading nodes do not overlap with other endpoints of tasks that are currently being executed by other agents and whose loading node is the closest to the current node, v_c^i. Then, the agent generates the path to the loading node, followed by the path from the loading node to the unloading node of the task. If it cannot select such a task, it returns to the parking node. We used the space-time A^* as the path finding algorithm for both methods, which is also used in TP. The first environment (Env. 1) is a maze-like environment with few task endpoints ($|V_{tsk}| = 6$), assuming a construction site (Fig. 2a). Nodes are set at the intersections and ends of the edges. Agents can rotate, wait, and load/unload materials only at nodes. Task endpoints are expressed by the blue squares in Fig. 2a, where agents can both load and unload their materials. A break in the edge in this figure indicates the length of a block having a length of 1.

The second environment (Env. 2) is also a maze-like environment with fewer task endpoints ($|V_{tsk}| = 8$), and the number of pickup/delivery locations is skewed, as shown in Fig. 2b. In this figure, orange squares are the pickup locations in which agents can

only load their materials, and the purple squares are the delivery locations where agents can only unload their materials. The initial locations of the agents are randomly assigned to the parking nodes, which are expressed by the red squares in both environments. One hundred tasks are initially generated by randomly selecting pickup and/or delivery locations from the blue, orange, and purple squares according to the experimental setting, and added to the $\mathcal{T}$. Note that Envs. 1 and 2 are not clearly bi-connected.

To evaluate our proposed method, we measured the *makespan*, i.e., the time required to complete all tasks in $\mathcal{T}$, and the *runtime*, i.e., the total CPU time for task selection, destination decision, and path planning for all tasks by all agents. Makespan indicates the transportation efficiency, whereas the runtime indicates the planning efficiency. Other parameter values are shown in Table 1. We conducted our experiments on a 3.00-GHz Intel 8-Core Xeon E5 with 64 GB of RAM. The experiment results below are the average of 50 trials with different random seeds. Sample videos of our experiments can be found at https://youtube.com/playlist?list=PLKufA_6vumDU01_hYNWihEN4IGlTh_oZ8.

5.2 Exp. 1: Performance Comparison

In the first experiment (Exp. 1), we compared the performances of our proposed SBDA method and the baseline HTE under Envs. 1 and 2. We plotted the results of Env. 1 in Fig. 3. We focus particularly on the results when $\alpha = 8$, which exhibited the highest transportation efficiency (a detailed discussion of the effect of the parameter α of SBDA is given in Section 5.3).

Figure 3a indicates that SBDA significantly reduces the makespan compared to the HTE method. In particular, when the number of agents is $M = 8$, SBDA reduces the makespan of HTE by approximately 39%. In HTE, agents cannot simultaneously execute tasks that have the same endpoint. Hence, only three or four agents can execute tasks in parallel even if M is larger because $|V_{tsk}| = 6$

under Env. 1. By contrast, because multiple agents can simultaneously execute tasks whose loading and unloading nodes overlapping with other executing tasks by using the standby nodes effectively, SBDA considerably improves the makespan through the higher parallelism of the task executions. As an exception, when $M = 2$, the makespan of SBDA is higher than that of HTE because when M is extremely small compared to the number of task endpoints $|V_{tsk}|$, agents can easily find tasks whose endpoints do not overlap and can always execute pickup-and-delivery tasks simultaneously without considering the standby nodes.

Figure 3b shows that the runtime for SBDA slightly increases compared to HTE. The only CPU usage in HTE comes from the task selection and path planning that generate a collision-free path in the less crowded environment. Hence, even if the number of agents increases, the runtime does not significantly increase, and is only used for a confirmation of the task selection. Meanwhile, in SBDA, many agents can select the tasks to execute and generate many paths mainly to endpoints and standby nodes, thus increasing the runtime. However, considering the parallel behavior, the CPU time spent per agent is not large.

The experiment results under Env. 2, which is plotted in Fig. 4, indicate that SBDA can improve the performance much more than HTE, and the length of the makespan, i.e., the average time required to complete the MAPD instances, is less than half that of HTE. For example, when $M = 10$, SBDA reduces the makespan of HTE by approximately 53%, which is a larger improvement than that under Env. 1. Because there are only two loading nodes, the number of agents working simultaneously is limited in HTE, and the makespan remains high. For SBDA, instead of many agents moving toward a small number of loading locations without any control, the agents can automatically wait at the standby nodes close to the loading nodes and move there in turn, which can increase the number of agents working at the same time. Furthermore, because Cond. 1 prevents too many agents from entering the work area, and Cond. 2 prevents the agents from ignoring the agents waiting at the standby nodes and cutting in to move toward the endpoint first, such conditions prevent excessive congestion, and thus contribute to the high efficiency.

5.3 Exp. 2: Features of SBDA

We conducted a number of experiments to confirm the impact of parameter α, which limits the distance between standby nodes and the endpoint v_{tsk} under Exp. 1, and then, Cond. 1 through an ablation study in the second experiment (Exp. 2). We also report the effect of Cond. 2 elsewhere [36].

5.3.1 Impact of parameter α. First, to analyze the features of SBDA in Exp. 1, we examined the impact of the parameter α on the performance. We conducted the same experiments under Envs. 1 and 2 with values of $\alpha = 0, 4, 8$, and 12. The results are also shown in Figs. 3 and 4. From Figs. 3a and 4a, the makespan of both environments decreases as the value of α increases from zero to 8, and conversely, increases when $\alpha = 12$. The larger the value of α, the larger the number of associated potential standby nodes for each task endpoint. This means that (2) and (3) in Cond. 1 of the task selection (Section 4.3) are more easily satisfied. Hence, more agents left their parking nodes as long as the environment was not too

crowded (Cond. 1 (1)), resulting in an increase in the parallelism during the task executions and a decrease of the makespan.

However, when α was too large, such as $\alpha = 12$, the makespan worsened. There seem to be three possible reasons for this. First, too much parallelism in a task execution causes congestion in the environment, leading to larger waiting times and longer detours. In SBDA, agents need to move toward their destinations without passing through the standby nodes. Although the connectivity of the graph G_t is maintained because of the property of the standby nodes, if more nodes in G_t are eliminated, the paths to the endpoints will be lengthened and the agents will be forced to make significant detours, resulting in long wait times at the standby nodes. As the second reason, the agents have to remain in standby nodes far from their task endpoints. Hence, when an agent starts toward an endpoint, the endpoint is exclusively held for a slightly longer period of time until the agent arrives, resulting in a chain reaction of reduced transportation efficiency. Finally, because more agents can fulfill their tasks, local congestion is likely to occur, particularly biasing the destinations.

We can also see from Figs. 3a and 4a that the number of agents for achieving the highest efficiency depends on the value of α as well as the structure of the environment. However, it seems that in all cases, except for $\alpha = 0$ in Env. 2, SBDA exhibited a better performance than HTE for $M \geq 4$. Note that under Env. 2, the performance was the highest when $M = 10 - 12$, which is slightly larger than that of Env. 1, and did not decrease significantly when $M > 12$, unlike the performance under Env. 1 when $M > 8$. This is because Env. 2 has only two pickup locations, which are thus more prone to localized congestion than Env. 1. Nevertheless, as the number of agents increases, Cond. 1 (2) of task selection can steer agents to tasks whose pickup nodes are less crowded and ultimately eliminate the congestion bias, thus reducing the makespan. In particular, this effect appeared more strongly under Env. 2 than under Env. 1, and thus, Cond. 1 (2) can prevent the agents from entering too much of the work area, averting a loss in efficiency.

If we look at Figs. 3b and 4b, the runtime required for planning is quite large when $\alpha = 0$ and M is between 6 and 10 under Env. 1 and when $M \geq 4$ under Env. 2, although the number of agents moving in the work area seemed small. This may be because agents in the parking nodes frequently probe for a possible task selection, whereas agents with tasks do not do so.

5.3.2 Ablation study. Focusing on Cond. 1 of the task selection (Section 4.3), we compared SBDA with SBDA without Cond. 1 (1) (SBDA-w/o-1), SBDA without Cond. 1 (2) (SBDA-w/o-2), and SBDA without Cond. 1 (3) (SBDA-w/o-3) as an ablation study to verify the effect of each condition. Figs. 5 and 6 show the comparative results of SBDA and ablation methods under Env. 1 and Env. 2, respectively. First, from Fig. 5a, the makespan of SBDA-w/o-1 was almost the same as that of SBDA under $M \leq 10$, whereas the makespan increased as M increased after $M = 12$, particularly under Env. 1. This indicates that Cond. 1 (1) contributed significantly to preventing many agents from entering the work area and crowding it. Under Env. 2, there are only two pickup nodes, and Cond. 1 (2) also prevents agents from entering the crowded work area; thus, the makespan did not increase significantly. This analysis also supports the fact that the runtime of SBDA-w/o-1 rapidly increases
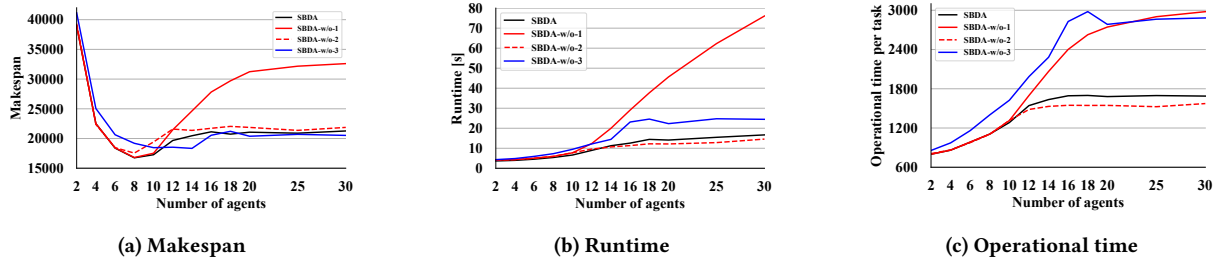
(a) Makespan

(b) Runtime

(c) Operational time

Figure 5: Ablation study with task selection conditions (Env. 1)

(a) Makespan

(b) Runtime
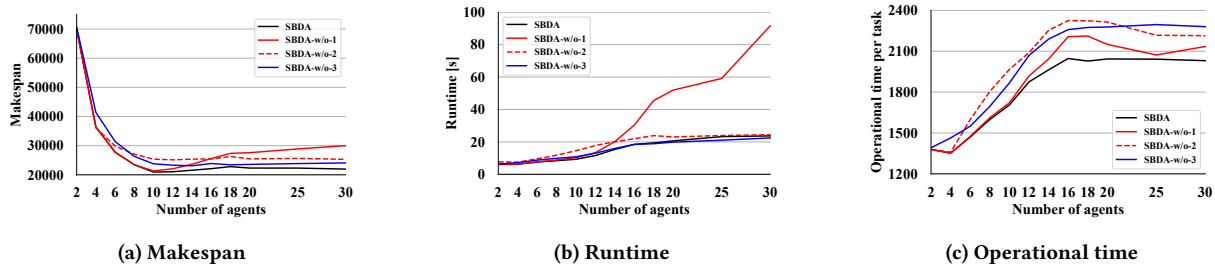
(c) Operational time

Figure 6: Ablation study with task selection conditions (Env. 2)

(Figs. 5b and 6b). Second, the minimum values of the makespan of SBDA-w/o-2 and SBDA-w/o-3 were larger than those of SBDA. If Cond. 1 (2) or (3) was removed, the agent selected a task without considering the status of the load or unload node or the associated potential standby nodes, resulting in a temporary evacuation when heading toward the load or unload node. However, this did not frequently occur in SBDA.

In Figs. 5c and 6c, we also plot the *operational time per task* or simply the *operational time*, which is the average time required to complete one task. In general, as the parallelism increases, the environment becomes more crowded and the operational time increases. However, SBDA-w/o-2 has the smallest value under Env. 1 (Fig. 5c). Because Cond. 1 (2) was ineffective, many agents evacuated the free potential standby nodes and CL was likely to be nonempty. As the results indicate, Cond. 1 (1) limited the number of agents entering the work area, reducing the parallelism. However, in Env. 2, the operational time of SBDA-w/o-2 increased because the agents had to stay at the free potential standby nodes much longer because of the limited number of loading nodes. Moreover, this also occurred because, by removing Cond. 1 (2), the agents always selected the tasks with the closest loading nodes without considering congestion; in particular, the agents in the parking nodes were biased toward to the upper-left loading node under Env. 2. In other cases, i.e., in SBDA-w/o-1 and SBDA-w/o-3, the operational time was large because too many agents entered the work area.

Finally, Figs. 5c and 6c show that the operational time of SBDA-w/o-3 tended to be large under both environments. Through the removal of Cond. 1 (3), the agents were likely to select tasks without considering the status of the unload nodes. Hence, they could not move to the unload nodes or their associated potential standby nodes after the load was completed and were repeatedly forced

to temporarily remain at free potential standby nodes or parking nodes with carrying the materials. Such interruptions in task execution lengthened the operation time.

6 CONCLUSION

We presented a deadlock avoidance method, called SBDA, for the MAPD problem to improve the transportation efficiency even in a maze-like environment. The central idea of SBDA is the use of *standby nodes* in which, even if an agent stays there, the connectivity of the environment remains. Furthermore, standby nodes can be identified with a low computational cost using the articulation-point-finding algorithm used in graph theory and are thereby found in real-time. SBDA guarantees completeness for well-formed MAPD instances by effectively using standby nodes that are guaranteed to wait for any finite amount of time. We evaluated the proposed method in comparison with a well-known conventional method, HTE, in restricted maze-like environments based on our envisioned applications. Our experiment results demonstrated that SBDA considerably outperforms the conventional method. We also analyzed the features of our proposed method and observed that the parallelism of the task execution can be controlled by changing the value of α, the parameter used for selecting standby nodes, and that the transportation efficiency can be significantly improved by appropriately setting the value of α.

To improve the flexibility and usability of SBDA for real-world applications, we plan to study a method for determining the appropriate value of α from a graph structure.

ACKNOWLEDGMENTS

REFERENCES

[1] Matteo Bellusci, Nicola Basilico, and Francesco Amigoni. 2020. Multi-Agent Path Finding in Configurable Environments. In *Proceedings of the 19th International Conference on Autonomous Agents and MultiAgent Systems*. 159–167.

[2] Eli Boyarski, Ariel Felner, Roni Stern, Guni Sharon, David Tolpin, Oded Betzalel, and Eyal Shimony. 2015. ICBS: improved conflict-based search algorithm for multi-agent pathfinding. In *Twenty-Fourth International Joint Conference on Artificial Intelligence*.

[3] Eli Boyrasky, Ariel Felner, Guni Sharon, and Roni Stern. 2015. Don't split, try to work it out: bypassing conflicts in multi-agent pathfinding. In *Twenty-Fifth International Conference on Automated Planning and Scheduling*.

[4] Michal Čáp, Jiří Vokřínek, and Alexander Kleiner. 2015. Complete decentralized method for on-line multi-robot trajectory planning in well-formed infrastructures. In *Twenty-Fifth International Conference on Automated Planning and Scheduling*. 324–332.

[5] Mehul Damani, Zhiyao Luo, Emerson Wenzel, and Guillaume Sartoretti. 2021. PRIMAL$_2$: Pathfinding Via Reinforcement and Imitation Multi-Agent Learning-Lifelong. *IEEE Robotics and Automation Letters* 6, 2 (2021), 2666–2673.

[6] Harshal S Dewang, Prases K Mohanty, and Shubhasri Kundu. 2018. A robust path planning for mobile robot using smart particle swarm optimization. *Procedia computer science* 133 (2018), 290–297.

[7] Ariel Felner, Roni Stern, Solomon Eyal Shimony, Eli Boyarski, Meir Goldenberg, Guni Sharon, Nathan Sturtevant, Glenn Wagner, and Pavel Surynek. 2017. Search-based optimal solvers for the multi-agent pathfinding problem: Summary and challenges. In *Tenth Annual Symposium on Combinatorial Search*.

[8] Taoan Huang, Bistra Dilkina, and Sven Koenig. 2021. Learning Node-Selection Strategies in Bounded-Suboptimal Conflict-Based Search for Multi-Agent Path Finding. In *Proceedings of the 20th International Conference on Autonomous Agents and MultiAgent Systems*. 611–619.

[9] Rahul Kala. 2012. Multi-robot path planning using co-evolutionary genetic programming. *Expert Systems with Applications* 39, 3 (2012), 3817–3831.

[10] Daniel Krakowczyk, Jannik Wolff, Alexandru Ciobanu, Dennis Julian Meyer, and Christopher-Eyk Hrabia. 2018. Developing a Distributed Drone Delivery System with a Hybrid Behavior Planning System. In *Joint German/Austrian Conference on Artificial Intelligence (Künstliche Intelligenz)*. Springer, 107–114.

[11] Jiaoyang Li, Andrew Tinka, Scott Kiesel, Joseph W Durham, TK Satish Kumar, and Sven Koenig. 2021. Lifelong Multi-Agent Path Finding in Large-Scale Warehouses. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 35. 11272–11281.

[12] Minne Li, Zhiwei Qin, Yan Jiao, Yaodong Yang, Jun Wang, Chenxi Wang, Guobin Wu, and Jieping Ye. 2019. Efficient Ridesharing Order Dispatching with Mean Field Multi-Agent Reinforcement Learning. In *The World Wide Web Conference*. ACM, 983–994. https://doi.org/10.1145/3308558.3313433

[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 18th International Conference on Autonomous Agents and MultiAgent Systems*. International Foundation for Autonomous Agents and Multiagent Systems, 1152–1160.

[14] 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*, Vol. 33. 7651–7658. https://doi.org/10.1609/aaai.v33i01.33017651

[15] Hang Ma, Sven Koenig, Nora Ayanian, Liron Cohen, Wolfgang Hönig, TK Kumar, Tansel Uras, Hong Xu, Craig Tovey, and Guni Sharon. 2017. Overview: Generalizations of multi-agent path finding to real-world scenarios. *arXiv preprint arXiv:1702.05515* (2017).

[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 16th Conference on Autonomous Agents and MultiAgent Systems*. International Foundation for Autonomous Agents and Multiagent Systems, 837–845.

[17] Hang Ma, Craig Tovey, Guni Sharon, TK Satish Kumar, and Sven Koenig. 2016. Multi-agent path finding with payload transfers and the package-exchange robot-routing problem. In *Thirtieth AAAI Conference on Artificial Intelligence*.

[18] Robert Morris, Corina S Pasareanu, Kasper Luckow, Waqar Malik, Hang Ma, TK Satish Kumar, and Sven Koenig. 2016. Planning, scheduling and monitoring for airport surface operations. In *Workshops at the Thirtieth AAAI Conference on Artificial Intelligence*.

[19] Zhenbang Nie, Peng Zeng, and Haibin Yu. 2020. Effective Decoupled Planning for Continuous Multi-Agent Pickup and Delivery. In *2020 Chinese Control And Decision Conference (CCDC)*. IEEE, 2667–2672.

[20] Esko Nuutila and Eljas Soisalon-Soininen. 1994. On finding the strongly connected components in a directed graph. *Inform. Process. Lett.* 49, 1 (1994), 9–14. https://doi.org/10.1016/0020-0190(94)90047-7

[21] Keisuke Okumura, Manao Machida, Xavier Défago, and Yasumasa Tamura. 2019. Priority Inheritance with Backtracking for Iterative Multi-agent Path Finding. In *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI-19*. International Joint Conferences on Artificial Intelligence Organization, 535–542. https://doi.org/10.24963/ijcai.2019/76

[22] Keisuke Okumura, Yasumasa Tamura, and Xavier Défago. 2019. winPIBT: Extended Prioritized Algorithm for Iterative Multi-agent Path Finding. *arXiv preprint arXiv:1905.10149* (2019).

[23] Keisuke Okumura, Yasumasa Tamura, and Xavier Défago. 2021. Time-Independent Planning for Multiple Moving Agents. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 35. 11299–11307.

[24] Zhongqiang Ren, Sivakumar Rathinam, and Howie Choset. 2021. MS*: A New Exact Algorithm for Multi-agent Simultaneous Multi-goal Sequencing and Path Finding. *arXiv preprint arXiv:2103.09979* (2021).

[25] Zhongqiang Ren, Sivakumar Rathinam, and Howie Choset. 2021. Multi-objective Conflict-based Search Using Safe-interval Path Planning. *arXiv preprint arXiv:2108.00745* (2021).

[26] D Reed Robinson, Robert T Mar, Katia Estabridis, and Gary Hewer. 2018. An efficient algorithm for optimal trajectory generation for heterogeneous multi-agent systems in non-convex environments. *IEEE Robotics and Automation Letters* 3, 2 (2018), 1215–1222.

[27] Oren Salzman and Roni Stern. 2020. Research Challenges and Opportunities in Multi-Agent Path Finding and Multi-Agent Pickup and Delivery Problems. In *Proceedings of the 19th International Conference on Autonomous Agents and MultiAgent Systems*. 1711–1715.

[28] Guni Sharon, Roni Stern, Ariel Felner, and Nathan R. Sturtevant. 2015. Conflict-based search for optimal multi-agent pathfinding. *Artificial Intelligence* 219 (2015), 40–66. https://doi.org/10.1016/j.artint.2014.11.006

[29] Haseeb Tahir, Mujahid N Syed, and Uthman Baroudi. 2019. Heuristic approach for real-time multi-agent trajectory planning under uncertainty. *IEEE Access* 8 (2019), 3812–3826.

[30] Robert Tarjan. 1972. Depth-first search and linear graph algorithms. *SIAM journal on computing* 1, 2 (1972), 146–160. https://doi.org/10.1137/0201010

[31] Manuela Veloso, Joydeep Biswas, Brian Coltin, and Stephanie Rosenthal. 2015. CoBots: Robust Symbiotic Autonomous Mobile Service Robots. In *Proceedings of the 24th International Conference on Artificial Intelligence* (Buenos Aires, Argentina) (*IJCAI'15*). AAAI Press, 4423–4429.

[32] Hanlin Wang and Michael Rubenstein. 2020. Walk, Stop, Count, and Swap: Decentralized Multi-Agent Path Finding With Theoretical Guarantees. *IEEE Robotics and Automation Letters* 5, 2 (2020), 1119–1126. https://doi.org/10.1109/LRA.2020.2967317

[33] Ko-Hsin Cindy Wang and Adi Botea. 2011. MAPP: a scalable multi-agent path planning algorithm with tractability and completeness guarantees. *Journal of Artificial Intelligence Research* 42 (2011), 55–90.

[34] Peter R Wurman, Raffaello D'Andrea, and Mick Mountz. 2008. Coordinating hundreds of cooperative, autonomous vehicles in warehouses. *AI magazine* 29, 1 (2008), 9–9. https://doi.org/10.1609/aimag.v29i1.2082

[35] Tomoki Yamauchi, Yuki Miyashita, and Toshiharu Sugawara. 2021. Path and Action Planning in Non-uniform Environments for Multi-agent Pickup and Delivery Tasks. In *European Conference on Multi-Agent Systems*. Springer, 37–54. https://doi.org/10.1007/978-3-030-82254-5_3

[36] Tomoki Yamauchi, Yuki Miyashita, and Toshiharu Sugawara. 2022. Standby-Based Deadlock Avoidance Method for Multi-Agent Pickup and Delivery Tasks. *arXiv preprint arXiv:2201.06014* (2022).

[37] Naoki Yoshida, Itsuki Noda, and Toshiharu Sugawara. 2020. Multi-agent Service Area Adaptation for Ride-Sharing Using Deep Reinforcement Learning. In *International Conference on Practical Applications of Agents and Multi-Agent Systems*. Springer, 363–375. https://doi.org/10.1007/978-3-030-49778-1_29

[38] Han Zhang, Jiaoyang Li, Pavel Surynek, Sven Koenig, and TK Satish Kumar. 2020. Multi-Agent Path Finding with Mutex Propagation. In *Proceedings of the International Conference on Automated Planning and Scheduling*, Vol. 30. 323–332.