Multi-Agent Path Finding Using Provisionally Booking Nodes for Pickup and Delivery Problems

Daiki Shimada, Yuki Miyashita[a] and Toshiharu Sugawara[b]
Waseda University, Shinjuku, Tokyo 1698555, Japan

Keywords: Multi-Agent Pickup and Delivery, Cooperation, Path Finding.

Abstract: We propose an efficient method for determining subsequent movements based on temporarily generated shortest paths in the *multi-agent pickup and delivery* (MAPD) problem. The MAPD problem involves multiple agents (such as carrier robots) continuously performing transportation tasks in a vast environment with obstacles while avoiding collisions with other agents. Our method is an extension of the decentralized path-finding algorithms, *priority inheritance with backtracking* (PIBT), and can be efficient in environments with narrow one-way paths and few detours. Our method, *PIBT with provisional booking* (PIBT-PB), not only secures the next node as in PIBT but also provisionally books some nodes in advance based on dynamic priorities between agents to detect possible conflict earlier. Therefore, it reduces the number of "turning back" and wasted "waiting" actions in environments. Our experiments show that PIBT-PB is more efficient than the baselines, PIBT and *windowed* PIBT, and even in less restrictive environments, it performs as efficiently as PIBT.

1 INTRODUCTION

Significant interest has been growing in the field of multi-agent systems (MAS) designed for executing sophisticated collaborative and coordinated tasks. Examples of such applications include cooperative security monitoring using multiple patrolling robots, material transportation using carrier robots in warehouses (Ma et al., 2017), construction sites (Miyashita et al., 2023; Ankit et al., 2022), rescue environments (Jennings et al., 1997), and automated catering robots in restaurants (Wang, 2023). In these systems, agents must individually determine appropriate actions based on their recognition of the surroundings to fulfill their objectives while avoiding negative interference, such as collision and resource conflict. Particularly, we address the problem of the *multi-agent pickup and delivery* (MAPD) (Ma et al., 2017) in a construction site context. Here, numerous agents (e.g., carrier robots) continuously pick up heavy and large materials and deliver them to the required destinations during the night, ready for the next day's installation work by human builders. This process is carried out while avoiding collisions and deadlocks. Therefore, it can be viewed as a repeated *multi-agent path finding* (MAPF) problem, where multiple agents generate collision-free paths between their start and goal locations.

In a construction site, installation materials are stored in a few storage areas and carried to specific locations where the materials will be installed. Another example of an MAPD situation is where materials are carried to and from trucks in the loading bay. This suggests that agents in the MAPD problem tend to gather in a few areas, restricting the paths that they can select and easily leading to deadlock and congestion. These problems can be mitigated in automated warehouses because they are specially designed to accommodate the movement of carrier agents, with wide paths and many detours of similar lengths. However, this is not the case for construction sites; agents must move along relatively narrow paths that do not allow them to pass each other. The topological structure of the environment may change because of new walls and doors installed the previous day. Furthermore, agents may take detours to avoid head-on collisions, but some detours are relatively longer. Therefore, the agent has to find a possible collision as early as possible to generate collision-free paths.

Numerous studies on MAPF and MAPD problems (Ma et al., 2017; Okumura et al., 2019a; Sharon et al., 2015; Yamauchi et al., 2022; Li et al., 2020) have been conducted because of their many appli-

[a] https://orcid.org/0000-0002-1676-9346
[b] https://orcid.org/0000-0002-9271-4507

Shimada, D., Miyashita, Y. and Sugawara, T.
Multi-Agent Path Finding Using Provisionally Booking Nodes for Pickup and Delivery Problems.
DOI: 10.5220/0013158300003890
In *Proceedings of the 17th International Conference on Agents and Artificial Intelligence (ICAART 2025) - Volume 3*, pages 528-537
ISBN: 978-989-758-737-5; ISSN: 2184-433X

cations, and our study focuses on a decentralized path-finding algorithm, such as *priority inheritance with backtracking* (PIBT) (Okumura et al., 2019a) and *windowed priority inheritance with backtracking* (winPIBT) (Okumura et al., 2019b) algorithms, because they guarantee reachability in a decentralized manner with fewer constraints on the arrangement of destinations. Agents using these algorithms can determine paths for their tasks individually based on their priorities, thus reducing the planning time by localizing the computation. Although PIBT is usually efficient, it may delay the detection of possible collisions because the agent communicates with other agents two nodes away and can only secure the next position toward its destination. To address this issue, winPIBT exclusively secures some nodes with a fixed time window. A longer window enables agents to detect possible collisions earlier, but the overall efficiency may decrease because the low-priority agents are restricted from moving by the secured nodes, even if no collisions occur.

Thus, we propose a method called *PIBT with provisional booking* (PIBT-PB). In this method, agents secure their immediate next nodes as in PIBT, but refrain from securing subsequent nodes. Instead, they tentatively book several nodes in advance, which are referred to as provisional nodes. This strategy allows for earlier detection of potential head-on collisions and prevents the obstruction of other agents. The number of provisional nodes is flexible, and can be adjusted based on the topological structure of the environment. We show that PIBT-PB maintains the reachability of all agents in the relaxed bi-connected area, as in PIBT, and demonstrate that it performs as efficiently as PIBT. Our experimental results indicate that PIBT-PB reduces the makespan — the total time required to complete all tasks, including planning time — compared to the baseline methods, PIBT and winPIBT, in our test environment. In addition, we discuss the benefits and limitations of the proposed approach.

2 RELATED WORK

Many studies have been conducted on MAPF and MAPD problems (Ma et al., 2017; Okumura et al., 2019a; Sharon et al., 2015; Standley, 2010; Goldenberg et al., 2014; Wagner and Choset, 2015; Silver, 2005; Yamauchi et al., 2022; Li et al., 2020). They can be roughly classified into centralized and decentralized approaches. In centralized approaches, the information on the environment and all agents, are collected to a single agent/server and it calculates and distributes all reasonable collision-free paths to all agents (Sharon et al., 2015; Luna and Bekris, 2011). For example, in *conflict-based search* (CBS) (Sharon et al., 2015), the path planning process is divided into high- and low-level search subprocesses. In low-level search, agents independently generate the shortest paths to their destinations and send them to the centralized server. The server then modifies all paths to eliminate possible collisions and distribute them to individual agents. Although this approach is likely to control all agents optimally in terms of travel path length, the computational cost increases as the number of agents increases. Moreover, we must consider the system's flexibility issue in the sense that if one agent cannot move as planned for any reason, all agents will be affected by the replanning process for the entire control.

In a decentralized approach (Ma et al., 2017; Okumura et al., 2019a; Ma et al., 2019; Yamauchi et al., 2022; Li et al., 2020; Farinelli et al., 2020; Miyashita et al., 2023), agents autonomously decide their own paths. Although this approach is more flexible, as agents have only local information, it has some issues such as the optimality of generated paths and reachability. For example, in *token passing* (TP) (Ma et al., 2017), the agent accesses the token, a type of shared memory, to refer to its content, generates the shortest collision-free path, and stores the information on that path into the token. TP can guarantee reachability under a reasonable assumption for an automated warehouse but its performance degrades in our target environment because it has only a few endpoints. Li et. al (Li et al., 2020) proposed the *rolling-horizon collision resolution* (RHCR) in which agents replan their paths at regular intervals while checking possible collisions within a certain window size. However, in congested situations, path generation becomes costly and cannot guarantee reachability.

Meanwhile, PIBT can reach the destination with only local communication, and reachability is guaranteed in an environment whose topological structure is *relaxed bi-connected*. However, its short-sighted algorithm delays the detection of collisions, reducing the efficiency. To overcome this drawback, winPIBT (Okumura et al., 2019b) secures a fixed number of nodes in advance to detect possible collisions earlier. However, it is not always *safe*, meaning that agents may unnecessarily restrict the movement of other lower-priority agents. Our proposed method can be considered an extension of PIBT, where some nodes are provisionally booked in advance if the nodes are safe to book. Although we already reported the abstract of this extension (Shimada et al., 2025), we provide a detailed explanation of the algorithm

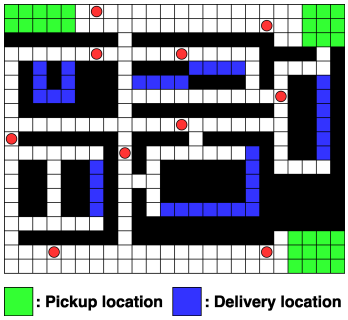
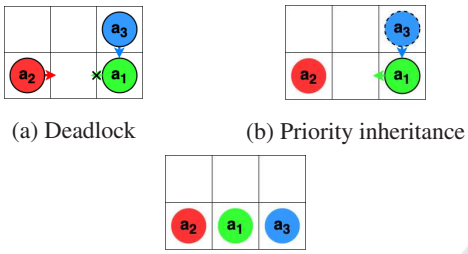


Figure 1: Example environment (Env. 3).



(a) Deadlock (b) Priority inheritance

(c) 1 step later

Figure 2: Procedure of Priority Inheritance.

and discusses the theoretical completeness of the algorithm and the results of experimental evaluation.

3 PROBLEM DESCRIPTION

3.1 MAPD

Let $A = \{a_1, a_2, ..., a_N\}$ be the set of N ($\in \mathbb{N}$) agents, where $\mathbb{N}$ is the set of non-negative integers. A MAPD environment can be expressed by a connected undirected graph $G = (V, E)$ embeddable in a two-dimensional Euclidean space, where V and E are the sets of nodes and edge connecting nodes, respectively. We introduce the discrete time $t \in \mathbb{N}$. An agent can move along edge $(u, v) \in E$ connecting two nodes $v, u \in V$. As G is undirected, if $(v, u) \in E$, $(u, v) \in E$. We assume that the length of all edges is 1, meaning that any agent can move to an adjacent node in one time step (by adding dummy nodes if necessary). An example of the grid-based environment is shown in Fig. 1, where red circles are agents that can move up, down, left, and right if there is space.

The location of agent a_i at time t is denoted by $v_i(t) \in V$. At each time t, a_i can move to $v_i(t+1) \in N_{v_i(t)}$ or stay at the same node $v_i(t)$, where $N_u = \{v \in V \mid (u, v) \in E\}$ is the set of the neighbor nodes of $u \in V$. A collision occurs when two agents stay at one node or cross the same edge simultaneously. Therefore, the following conditions must be satisfied for $\forall a_i, a_j \in A\ (i \neq j)$.

$$v_i(t) \neq v_j(t) \wedge (v_i(t) \neq v_j(t+1) \vee v_i(t+1) \neq v_j(t)) \quad (1)$$

Meanwhile, synchronized circular movements without collisions are assumed to be possible, that is,

$$\begin{aligned} v_i(t+1) = v_j(t) \wedge v_j(t+1) = v_k(t) \wedge \ldots \\ \wedge\, v_l(t+1) = v_i(t) \end{aligned} \quad (2)$$

is allowed for different agents $a_i, a_j, a_k, \ldots, a_l \in A$.

Let $\Gamma = \{\tau_1, \tau_2, ...\}$ be a finite set of tasks. In MAPD, task τ_j is represented by a pair of pickup node g_1 and delivery node g_2 ($g_1 \neq g_2$). When τ_j is assigned to a_i at t, τ_j is removed from Γ and a_i travels from $v_i(t)$ to g_1, g_2 by setting its destination in turn. After both destinations are visited, another task in Γ is assigned to a_i if $\Gamma \neq \varnothing$. This also means that all agents individually determine their next destinations when they have reached the current destinations. We denote the set of the current destinations of all agents as $D = \{d_1, \ldots, d_N\}$, where d_i is the destination of a_i and we define $d_i = nil$ if a_i has no destination.

3.2 Priority Inheritance with Backtracking (PIBT)

According to PIBT (Okumura et al., 2019a), each agent a_i has a priority that is updated at each time step. The agent interacts with nearby agents through a recursive process involving *priority inheritance* (PI) and *backtracking* (BT) to determine its next move based on the priority order.

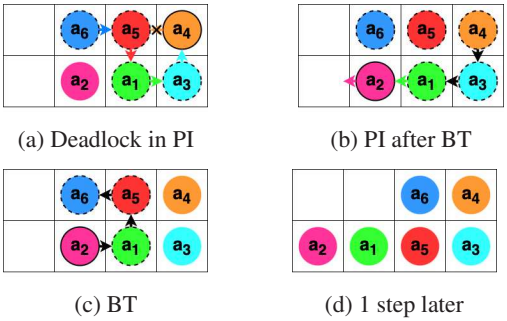
Definition 1. (Relaxed bi-connected graph) Graph $G = (V, E)$ is a relaxed bi-connected graph iff G is a connected graph, and there is a cyclic path of at least length 3 between any pair of neighbor nodes in G.

PIBT has been proven to ensure that all agents can reach their destinations without deadlock, livelock, or collision, if $G = (V, E)$ is a relaxed bi-connected graph and $|A| \leq |V|$. We briefly explain PIBT. Agent $a_i \in A$ has a priority

$$p_i(t) = \varepsilon_i + \eta_i(t) \quad (3)$$

for the movement along its planned path. Here, ε_i ($0 \leq \varepsilon < 1$) is the base value of the unique priority assigned to a_i initially, and $\eta_i(t)$ ($\in \mathbb{N}$) is the elapsed time after a_i updates its destination. Agent a_i sets $\eta_i(t)$ to 0 when arriving at its destination. Evidently, agents have different priorities.

First, agent a_i ranks the nodes in $N_{v_i(t)} \cup \{v_i(t)\}$, usually at time t, according to the estimated distance to its destination from $v \in N_{v_i(t)} \cup \{v_i(t)\}$ using some path-finding methods for MAS, such as *cooperative A* search* (CA*) (Silver, 2005). Then, a_i secures the



(a) Deadlock in PI (b) PI after BT

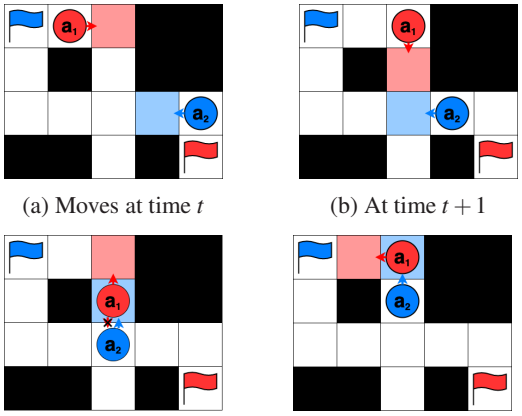
(c) BT (d) 1 step later

Figure 3: Example Procedure of Priority Inheritance and Backtracking.

next node in the order of priority as follows. Suppose that a_i has the highest priority in its local area. If agent a_j s.t. $p_j(t) < p_i(t)$ occupies node a_i and wants to move next, a_i pushes a_j and a_j must yield its current location by moving a certain direction. However, this may result in a potential collision with other agents. In this case, a_i inherits its priority to a_j to avoid it. An example is shown in Fig. 2, where the arrow indicates the potential moves and $p_1(t) < p_2(t) < p_3(t)$. In Fig. 2a, a_3 attempts to secure a_1's current node, and thus, a_1 has to move left, but as a_2 tries to move to the same node, a_1 cannot determine the next node. However, by PI, the priority of a_3 is inherited by a_1, thus, a_1 has a higher priority than a_2 (Fig. 2b) and can secure the left node. Meanwhile, a_2 does not move right but chooses to "stay" at its current node, which is the next ranked node toward its destination, as shown in Fig. 2c.

Backtracking (BT) is a process where the results of the priority inheritance (PI) process, which attempted to secure a node, are returned in reverse order. Particularly, an agent that fails to secure a node returns the result directly to the agent that inherited its priority. Upon receiving this information of failure, the priority inherited agent attempts to move to the next best node if possible. If there are no other agents at that node, it secures that node; otherwise, it recursively executes PI and attempts to secure that node. It then repeats this operation. If all relevant agents cannot determine the next location through the PI and BT processes, they decide that the PI is stuck and choose to remain at the current nodes. Note that in environments that meet the condition of the relaxed bi-connected graph, it is demonstrated that all agents can determine their next nodes by PI and BT (Okumura et al., 2019a).

An example procedure of BT is shown in Fig. 3, in which the priority of each agent is assumed to be greater in descending order of subscript numbers. First, the highest priority agent a_6 attempts to secure the next node where a_5 is located. Figure 3a shows



(a) Moves at time t (b) At time $t+1$

(c) Conflict and push (d) Push back again

Figure 4: Inefficient path planning in PIBT.

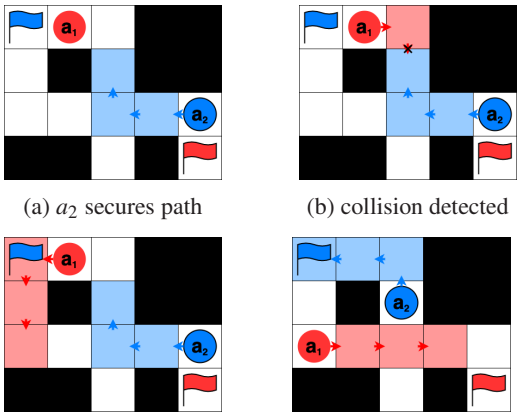
that the priority of a_6 is inherited along a_5, a_1, a_3, and a_4. However, because a_4 could not secure the next nodes, a_4 and a_3 backtrack by returning the failure results, as shown in Fig. 3b, and remaining in their current nodes. Subsequently, a_1 can select another node to move by pushing a_2 because a_1 has higher priority than a_2 owing to PI. This result of success is then delivered by backtracking along a_2, a_1, a_5, and a_6 (Fig. 3c). After that, all involved agents can decide their next nodes. Figure 3d shows their positions at the next time step, wherein a_3 and a_4 cannot move, but they can stay at the same positions.

3.3 winPIBT (windowed PIBT)

The winPIBT (Okumura et al., 2019b) algorithm enhances the PIBT by incorporating a *time window*, allowing agents to secure multiple nodes for consecutive time steps. This extension aims to mitigate the inefficiencies in PIBT, particularly those stemming from delayed collision detection.

An example of inefficiency in PIBT is illustrated in Fig. 4 at time t, where agents a_1 and a_2 move to their destinations indicated by flags of the same colors. Here, agent a_2 has a higher priority than a_1 ($p_2(t) > p_1(t)$), and the agents' secured nodes for the next moves are shown in the same colors of agents. In Figs. 4a and 4b, both agents secure and move according to the shortest paths that they individually generate. After that, they encounter and recognize the possible collision. Therefore, a_2 must push back the lower-priority agent a_1 a few times, as shown in Figs. 4c and 4d. These moves by a_1 are wasteful and inefficient.

For this problem, in winPIBT, the information of the secured nodes is shared among all agents (at least, close agents), and the highest priority agents that have



(a) a_2 secures path (b) collision detected

(c) a_1 replans a path (d) 3 steps later

Figure 5: Movements in winPIBT.

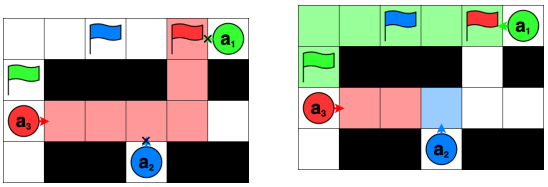
not yet determined the next node attempt to secure some future nodes whose number is specified by the fixed size of the time-window, W (> 0). We illustrate how winPIBT mitigates this wasted movement problem using Fig. 5, where $W = 3$. At t, a_2 secures the nodes for up to three time steps ahead (Fig. 5a). Then, a_1 attempts to secure three nodes but can secure only one neighbor node (Fig. 5b). If there were no other path, a_1 would stay at this node for three time steps, but in this situation, there is another path, and the node after three steps will be the same or closer to its destination. Thus, it secures three nodes along an alternative path at t (Fig. 5c). Figure 5d shows the next reservation at $t+3$, indicating that both agents can reach their destination without unnecessary movement. Note that it is obvious that when $W = 1$, winPIBT is equivalent to PIBT.

winPIBT introduces the *disentangled condition*, which intuitively requires that agents secure paths in advance while holding Condition (1). This condition includes that agent a_i cannot secure the number of nodes at t_0 that must be equal or less than the number of the nodes already secured by agent a_j if $p_j(t_0) > p_i(t_0)$ and cannot secure the node that has been secured by a_j to pass before a_j. Thus, they must satisfy the following condition,

$$v_i(t) \neq v_j(t+k) \text{ for } t_0 \leq \forall t \leq t_0 + w \text{ and } 0 \leq k \leq t_j^{\omega} \quad (4)$$

where $\pi_i = (v_i(t^0), ..., v_i(t_i^0 + w))$ is the nodes secured by a_i ($w \leq W$), $v_j(t+k)$ is the node at time $t+k$ secured by a_j, and t_j^{ω} is the last time when a_j already secured the node.

Although winPIBT has the reachability to destinations for all agents, the condition expressed by Eq. (4) indicates that the agents are likely to block the moves of other agents with the lower priorities even if no collision occurs, resulting in inefficiency. Figure 6a expresses an example situation in which $p_1 < p_2 < p_3$ for agents a_1, a_2, and a_3, and they move to their destinations with the same colors. Because a_3 first secures some nodes earlier, including a crossing node, a_2 and a_1 must wait for a_1 to pass, even if no collision occurs. If the fixed length of time window is short, agents may delay in detecting possible collisions. In contrast, a longer time window allows earlier collision detection but increases the likelihood of blocking other agents. Therefore, we aim to address this problem while effectively detecting possible collisions.



(a) A blocking situation. (b) Secured and provisional nodes.

Figure 6: Comparison between winPIBT and PIBT-PB.

4 PROPOSED METHOD

4.1 Maximal Bi-Connected Component and Node Classification

First, we analyze the characteristics of a bi-connected graph. Inefficient and wasteful behavior in PIBT occurs when a lower-priority agent a_i encounters a higher-priority agent a_j and a_i has no other way except turning back. This situation occurs in a narrow path, and the cost of going backward is significant if the distance is long.

First, let us consider the characteristics of the relaxed bi-connected graph.

Proposition 1. If $G = (V, E)$ is a relaxed bi-connected graph, $|N_v| \geq 2$ for $\forall v \in V$.

Proof. If $\exists u \in V$ s.t. N_u is the singleton set, i.e., $N_u = \{u'\}$. Clearly, the pair (u, u') does not have a cyclic path. □

We can classify nodes in V into *one-way* and *crossing* nodes from Proposition 1.

Definition 2. Node $v \in V$ is called a one-way node if $|N_v| = 2$; otherwise, v is called a crossing node.

We denote the sets of one-way and crossing nodes as $V_{ow} = \{v \mid |N_v| = 2\}$ and $V_{crs} = \{v \mid |N_v| \geq 3\}$, respectively. We also investigate the structure of a relaxed bi-connected graph.

Definition 3. Graph $G = (V,E)$ is a bi-connected graph iff any pair of nodes, $v_1, v_2 \in V$, has a cyclic path in G.

For graph $G = (V,E)$ and $V' \subset V$, we can naturally generate subgraph $G' = (V',E')$, where $E' = \{(v_1,v_2) \in E \mid v_1, v_2 \in V'\}$.

Definition 4. Subgraph $G' = (V',E')$ generated from graph $G = (V,E)$ is maximal bi-connected component iff G' is bi-connected and $\forall v \in V \setminus V'$, the subgraph naturally generated from $V' \cup \{v\}$ is not bi-connected.

Proposition 2. (Structure of a relaxed bi-connected graph) $G = (V,E)$ is a connected graph and $G_1 = (V_1,E_1), \ldots, G_K = (V_k,E_K)$ are all maximal bi-connected subgraphs of G. $G = (V,E)$ is a relaxed bi-connected graph iff G consists only of a number of subgraphs of maximal bi-connected components, i.e., $V = V_1 \cup \cdots \cup V_K$ and $E = E_1 \cup \cdots \cup E_K$. Furthermore, any pairwise intersections of maximal bi-connected subgraphs have at most one node (Miyashita et al., 2023), and this intersection node has at least four neighboring nodes, with two of them belonging to the same maximal bi-connected node.

Proof. The sufficient condition is straightforward. Suppose that G is a relaxed bi-connected graph. If there exists $\exists v \in V \setminus V_1 \cup \cdots \cup V_K$ such that $\exists v' \in N_v$ and $v' \in V_1 \cup \cdots \cup V_K$ (if not, G is not connected), then v and v' have a cyclic path. Thus, they belong to a maximal bi-connected subgraph, which also leads to a contradiction. The latter part is also evident because if their intersection has two distinct nodes, the union of these subgraphs forms a larger bi-connected graph (Miyashita et al., 2023), which contradicts that these subgraphs are maximal. Furthermore, suppose there is only one neighboring node v' of an intersection node v belonging to a maximal bi-connected subgraph G_k. Because $v', v \in V_k$, there is a cyclic path connecting these nodes, meaning that v has another neighboring node $v'' \in V_k$, which leads to a contradiction. This also means that any intersection node of maximal bi-connected graphs has at least four neighboring nodes. □

Proposition 2 shows that if a node u belongs to the intersection of two maximal bi-connected subgraphs, then $u \in V_{crs}$. Thus, a sequence of one-way nodes belongs to a maximal bi-connected component.

4.2 PIBT with Provisional Booking

The central idea of PIBT-PB for MAPD in a relaxed bi-connected area is that an agent first secures the next node. If it does not inherit the priority from other agents, it attempts to book further nodes provisionally if the nodes are safe.

Algorithm 1: PIBT-PB at current time t_0 for at least securing the node at t_0+1.

```
1  p_i(t_0) is defined by Eq. 3. Agent a_i at v_i(t_0) will
     secure a node for t_0 + 1.
2  π_i^S[t] ∈ V: Secured node by a_i for time t; a_i
     definitely visits to it at t.
3  π_i^P[t] ∈ V: Provisional nodes booked by a_i for t in
     advance.
4  procedure PIBT-PB(A, t_0):
5      // t_0 suggests the current time.
6      S ← A and is sorted by p_i(t_0) in descending
         order.
7      for a_i ∈ S do
8          if π_i^S[t_0 + 1] = nil then
9              // a_i has no secured node at t_0 + 1. if
                 (π_i^P[t_0 + 1] ≠ nil) then
10                 mPIBT(a_i, a_i, t_0)
11             else
12                 mPIBT(a_i, nil, t_0)
13             end
14         end
15     end
16 end
```

The algorithms are listed in Algs. 1, 2, and 3. We denote the array of nodes that agent a_i has secured at t as $\pi_i^S[t] \in V$, and the array of nodes that a_i provisionally booked at t as $\pi_i^P[t] \in V$. In the proposed method (function PIBT-PB in Alg. 1), agents' priorities are calculated using Eq. 3 every time. Suppose that at current time t_0 the highest priority agent $a_i \in A$ does not secure its next node yet ($\pi_i^S[t_0] = nil$). It first attempts to secure the next node using a PIBT-like method, mPIBT. Subsequently, it provisionally books additional nodes using addProvisionalNodes to detect a possible collision earlier. We describe how agents secure/provisionally book nodes in detail.

4.2.1 Securing next Node

In PIBT-PB (Alg. 1), if a_i has already provisionally booked the next node (Line 9 in Alg. 1)) it invokes $\mathsf{mPIBT}(a_i, a_i, t_0)$ to simply check the conflict. Otherwise, a_i invokes $\mathsf{mPIBT}(a_i, nil, t_0)$ (Line 12).

In Alg. 2, when $a_j \neq nil$, a_i's priority is inherited from a_j (including the case $a_i = a_j$) and a_i releases the provisional nodes because it is unintentionally pushed by another agent (Line 3). If a_i has no provisionally booked node at t_0+1, it calculates the shortest path to d_i starting from $\pi_i^S(t_0)$ ($= v_i(t_0)$) using CA* (or another similar algorithm) by setting the already secured node at t_0+1 and the provisional nodes booked by the higher priority agents thus far as obstacles (Line 7). This is temporarily stored to array Π. Note that agent

Algorithm 2: mPIBT at time t_0 to decide the next node to move.

```
1  procedure mPIBT(a_i, a_j, t_0):
2      if (a_j ≠ nil ∧ a_j ≠ a_i) then
3          π_i^P ← ∅, p_i(t_0) ← p_j(t_0)
4      end
5      if (π_i^P = ∅) then
6          // CA*(a_i) searches the shortest path to d_i
             by setting the secured and provisional
             nodes as obstacles.
7          Π ← CA*(a_i), π_i^P[t_0+1] ← Π[t_0+1]
8      end
9      while (π_i^P ≠ ∅) do
10         // If there is a potential conflict, PI
             occurs.
11         if (∃a_k ∈ A s.t. (p_k(t_0) <
             p_i(t_0)) ∧ (π_i^P[t_0+1] = π_k^S[t_0])) then
12             if mPIBT(a_k, a_i, t_0) is nil then
13                 // As a_k secured a node even if
                     mPIBT returns nil, the
                     following CA* returns another
                     path if exists.
14                 Π ← CA*(a_i)
15                 π_i^P[t_0+1] ← Π[t_0+1]
16                 continue
17             end
18         end
19         π_i^S[t_0+1] ← π_i^P[t_0+1]
20         if a_j = a_i then
21             return valid
22         end
23         if (∃a_k ∈ A s.t. π_k^P[t_0+1] = π_i^S[t_0+1])
             then
24             π_k^P ← ∅
25         end
26         if (a_j = nil) ∧ (π_i^S[t+1] ≠ d_i) then
27             addProvisionalNodes(a_i, Π, t_0)
28         end
29         return valid
30     end
31     π_i^S[t_0+1] ← π_i^S[t_0]
32     // stay if a_i finds no node to move next.
33     return nil
34 end
```

a_k s.t. $p_k(t_0) < p_i(t_0)$ initially by Eq. 3 may now have a higher priority by PI. In this case, a_k has already secured a node at t_0+1 but has no provisional nodes. If CA* cannot generate the path, it returns $\varnothing$ and jumps to Line 33. Otherwise, a_i continues with another generated path Π and $\pi_i^P[t_0+1]$ $(=\Pi[t_0+1])$.

If this is not the case, a_i secures the provisional node (Line 19) and returns "valid" if $a_i = a_j$. Subsequently, if $\exists a_k$, s.t. $\pi_k^P[t+1] = \pi_i^S[t+1]$, a_k release the provisional nodes because $p_k(t) < p_i(t)$. Furthermore, if a_i is not pushed by another agent, it invokes addProvisionalNodes to additionally book some provisional nodes if possible. If a_i cannot find any neighboring node, it remains at the current node (Line 33).

Algorithm 3: To find some provisional node after t_0+2.

```
1  procedure addProvisionalNodes(a_i, Π, t_0):
2      for (t ∈ {t_0+1, ...}) do
3          // Try to provisionally book π_i^P[t_0+2]
             and after that.
4          // If a_i reached d_i at t or the next node is
             crossing, exit from the for-loop.
5          if (π_i^P[t] = d_i) ∪ (Π[t+1] ∈ V_crs) then
6              return valid
7          end
8          // If a possible collision is detected,
9          if (∃a_k ∈ A s.t. (Π[t] = π_k^P[t]) ∨ (Π[t] =
             π_k^P[t+1] ∧ Π[t+1] = π_k^P[t])) then
10             // The lower-priority agent releases
                 all provisional nodes.
11             if p_k(t_0) < p_i(t_0) then
12                 π_k^P ← ∅, π_i^P[t+1] ← Π[t+1]
13             else
14                 π_i^P ← ∅
15                 return nil
16             end
17         end
18     end
19 end
```

4.2.2 Provisional Booking

In the function addProvisionalNodes (Alg. 3), agent a_i attempts to provisionally book some nodes until a_i has secured or provisionally booked its destination d_i, or before a crossing node (Line 5). If a_i detects a potential collision (Line 9), the provisional nodes of the lower-priority agent will be released. If a_i is the higher-priority agent, the next node in $\Pi[t+1]$ is provisionally booked and stored to $\pi_i^P[t+1]$.

Figure 6b illustrates the situation in which agents utilizing PIBT-PB avoid the unnecessary blocking of other agents shown in Fig. 6a. Initially, a_3 secures the right node and provisionally books another right node (depicted as the two red nodes in Fig. 6b) en route to its destination by using the mPIBT and addProvisionalNodes functions. Next, a_2 secures an upper node without booking any provisional nodes. Subsequently, a_1 secures the left node and provisionally books five left nodes, including its destination node, based on the shortest path to the destination.

4.3 Property of PIBT-PB

Each agent has unique priority. Hence, one agent a_i always has the highest priority. This agent contin-

ues to have top priority until it arrives at its destination. Furthermore, from mPIBT in PIBT-PB, a_i can always secure the highest ranked $v \in N_{v_i(t)}$ at t based on the shortest path to its destination because (1) pair $(v_i(t), v)$ has a cycle path, and (2) when a_i calculates the shortest path, other agents do not secure the next node, and the provisional nodes booked by other agents are not considered as obstacles. Therefore, the following proposition is evident from Prop. 2.

Proposition 3. An agent with the highest priority can reach its destination in a finite time if the environment G is the relaxed bi-connected graph.

When an agent reaches its destination, its priority becomes the lowest. Furthermore, if $p_i(t) > p_j(t)$ for $a_i, a_j \in A$, the same inequality holds at $t+1$ unless a_i reaches its destination at $t+1$. In addition, if there is an agent that cannot reach its designation, its priority will monotonically increase and become the highest, thus allowing it to finally reach the destination. Therefore, the following theorem holds.

Theorem 1. When environment G is a relaxed bi-connected graph, all agents can reach their destinations in a finite time.

When all nodes are crossing, PIBT-PB operates in the same manner as PIBT, as the process for determining agents' subsequent nodes is fundamentally equivalent to PIBT without the use of provisional node booking. This means that agents using PIBT-PB follow the same paths as those using PIBT. If the environment has some one-way nodes, agents can book provisional nodes and detect possible head-on collisions earlier than with PIBT.

Similar to winPIBT, PIBT-PB requires all agents to have access to shared memory containing information about locations, time, and agents of all secured/provisional nodes. However, in our proposed approach, agent a_i only provisionally books one-way nodes. Consequently, a_i utilizes the node information solely for the connected one-way path that terminates at a provisionally unbooked crossing node. This approach reduces the cost of accessing shared memory by organizing it according to the graph structure (Proposition 2) because the provisional nodes are contained within a specific maximal bi-connected component of G.

5 EXPERIMENTAL EVALUATION

5.1 Experimental Environment

We evaluate the proposed method by conducting experiments in three environments shown in Fig. 7 (Environment 1 — Env. 1), Fig. 8 (Environment 2 — Env. 2), and Fig. 1 (Env. 3) and compared the performance of agents using PIBT-PB with that of agents using the baselines, PIBT and winPIBT. Env. 1 is a model of a warehouse, which is often used for experiments in some papers. Pickup and delivery nodes are selected randomly when a task is generated. In this type of environment, we cannot identify the clear difference in performance between PIBT and winPIBT; however, we aim to demonstrate the superior performance of our method. Env. 2 is a simple model designed to illustrate the differences between PIBT and PIBT-PB. Although only four corners are one-way nodes, PIBT-PB outperforms PIBT. In Env. 2, green and blue nodes denote pickup and delivery points, respectively. Thus, the agent first moved to a green node and then proceeded to a blue node, located either on the same or opposite side. Env. 3 (Fig. 1) simulates a construction site where agents pick up materials at one of the storage areas (green nodes) and deliver them to a node in the installation areas (blue nodes). Their pickup and delivery locations are unbalanced, thus congestion may occur near pickup and delivery areas if the number of agents is large.

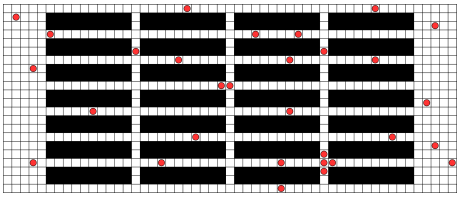


Figure 7: Environment 1 (randomly assigned destinations).

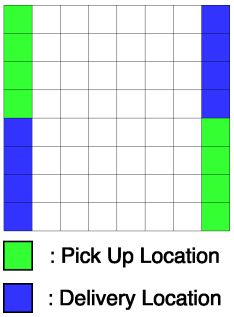


Figure 8: Environment 2.

The initial positions of the agents were randomly assigned to each environment, and 1000 (600) MAPD tasks were allocated to Envs. 1 and 3 (Env. 2). Each task τ comprises pickup and delivery points $\tau = (g_1, g_2)$. The assigned agent sets g_1 and then g_2 as its destinations. Upon task completion, another task is assigned from Γ, which is a set of MAPD tasks. The efficiency metric for MAPD is the *makespan*, which measures the number of steps from the start of the experiment to the completion of all tasks; a lower

makespan indicates a better performance. The window size of winPIBT W (> 1) was optimized for the shortest makespan and varied according to the environment. Experiments were conducted by varying the number of agents N to assess the performance impact on PIBT-PB and the baselines. The results presented below are the averages of 20 independent trials.

5.2 Experimental Results

The results are presented in Fig. 9. Across all three environments, the findings demonstrate that PIBT-PB outperformed the other methods, achieving shorter makespan time steps regardless of agent quantity. As shown in Figure 9a, PIBT and winPIBT displayed nearly identical performances in Env. 1, where the agent numbers ranged from 20 to 100, and the winPIBT window sizes were set at $W = 5$ and 10. The value $W = 10$ was larger in other environment. Thus, agents could find possible conflicts much earlier than those of agents using PIBT. However, they restricted other agents' movement. Because the advantages and disadvantages cancel each other out, the values of makespans for winPIBT were almost identical to those of PIBT. In contrast, PIBT-BP did not book any crossing nodes as provisional nodes in advance and needlessly restricted other agents' movements, while it could detect possible conflicts effectively. For example, the makespan of PIBT-PB (1040.8) was approximately 14.3% lower than that of PIBT (1214.7) when $N = 100$.

The makespans in Env. 2 are plotted in Fig. 9b, in which we varied the number of agents from 10 to 50. With 48 nodes in the moving area (white nodes), $N = 50$ means the environment is extremely crowded. Figure 9b shows that the agents using PIBT-PB exhibited shorter makespans than those using the baselines, even in such a simple environment. Because the pickup and delivery nodes could be set to four corner nodes, agents could reach them with the final step by provisional booking. Even in a grid-like environment, such as Env. 2, agents using winPIBT restricted other agents by securing some nodes on the opposite side, making the performance of winPIBT optimal when $W = 3$, a relatively small window size.

In Env. 3, PIBT-PB demonstrated superior performance, whereas PIBT and winPIBT ($W = 5$) showed nearly identical results, with winPIBT having a slight edge. In this setting, winPIBT agents frequently blocked other agents, causing them to wait at intersection points in the central areas. The benefits and drawbacks of winPIBT compared to PIBT balance each other, similar to Env. 1, resulting in a comparable efficiency. However, PIBT-PB agents successfully avoided unnecessary blockages at intersections where agents could potentially hinder each other's movements, thus enhancing the efficiency of detecting head-on collisions.

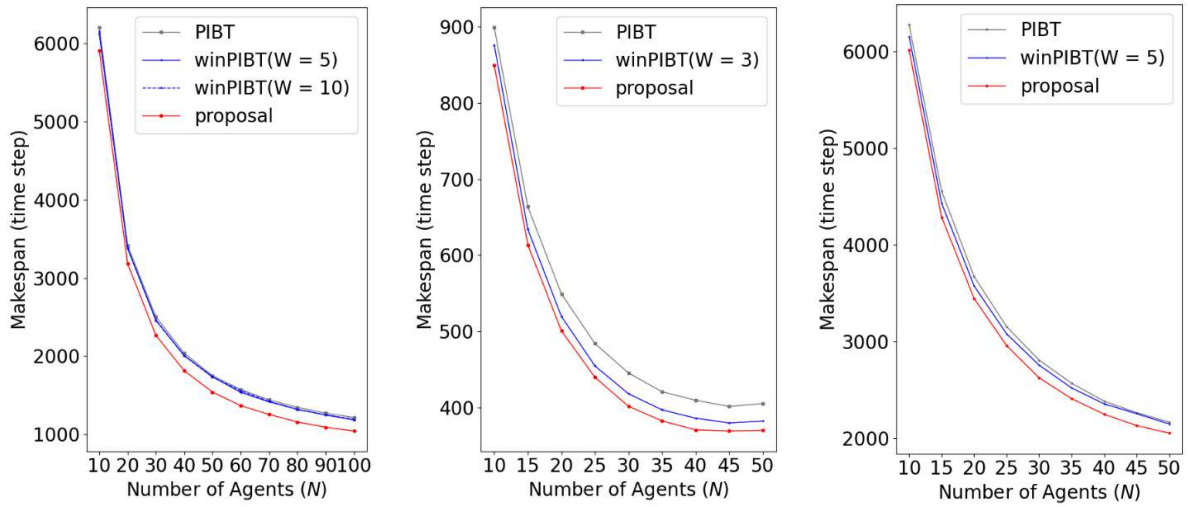
5.3 Discussion

Our experiments revealed that PIBT-BP improved MAPD problem-solving efficacy compared with traditional priority-based methods, including PIBT and winPIBT. WinPIBT agents increased path-planning efficiency by using time windows for early collision detection, unlike PIBT. However, winPIBT also imposed strict movement constraints on other agents, often hindering them at intersections by securing additional nodes. To mitigate this, we avoided securing extra nodes at intersections, where movement might be restricted. Critical scenarios to prevent potential collisions in narrow, one-way paths where agents cannot pass each other and must backtrack and waste time. Therefore, PIBT-PB agents books as many provisional nodes as possible on one-way routes to detect future collisions. PIBT-BP is expected to be particularly effective in maze-like environments, with limited or lengthy alternative routes and warehouse settings.

6 CONCLUSION

We proposed PIBT with provisional booking (PIBT-BP), which extends PIBT to detect possible collisions earlier. First, agents using PIBT-BP secure the next node similar to PIBT, and then attempt to provisionally book additional one-way nodes in advance, as booking a crossing node often blocks other agents' actions. Subsequently, if no collision is detected, the agent proceeds through the provisional nodes step-by-step. If a higher-priority agent approaches, it releases the booked nodes and generates an alternative path by treating the approaching agent as an additional obstacle. Nonetheless, agents always reach their destinations, because the first step follows a method that is essentially identical to PIBT.

One of the future works is to improve the dynamic prioritization used for controlling agents. In this study, the prioritization depends on the time elapsed since the destination was updated and restarted. Therefore, an agent with a nearby destination may have to take a detour owing to its priority level. To avoid this situation, we propose utilizing information, such as the distance between the agent and the destination, as well as the structural information of one-way paths in the environment. We also need to adapt



(a) Env. 1 (b) Env. 2 (c) Env. 3

Figure 9: Experimental Results of Makespan.

our method to accommodate flexible environments, such as construction sites. These environments undergo changes where paths are generated or eliminated owing to the construction of new walls, doors, polls, and changes in material storage areas.

REFERENCES

Ankit, K., Tony, L. A., Jana, S., and Ghose, D. (2022). Multi-agent Cooperative Framework for Autonomous Wall Construction. In Saraswat, M., Sharma, H., Balachandran, K., Kim, J. H., and Bansal, J. C., editors, *Congress on Intelligent Systems*, pages 877–894, Singapore. Springer Nature Singapore.

Farinelli, A., Contini, A., and Zorzi, D. (2020). Decentralized task assignment for multi-item pickup and delivery in logistic scenarios. In *AAMAS*.

Goldenberg, M., Felner, A., Stern, R., Sharon, G., Sturtevant, N., Holte, R., and Schaeffer, J. (2014). Enhanced partial expansion A*. *J.Artif.Intell.Res*, 50:141–187.

Jennings, J., Whelan, G., and Evans, W. (1997). Cooperative search and rescue with a team of mobile robots. In *1997 8th International Conference on Advanced Robotics. Proceedings. ICAR'97*, pages 193–200.

Li, J., Tinka, A., Kiesel, S., Durham, J. W., Kumar, T. K. S., and Koenig, S. (2020). Lifelong Multi-Agent Path Finding in Large-Scale Warehouses. In *AAMAS*.

Luna, R. and Bekris, K. E. (2011). Push and swap: Fast cooperative path-finding with completeness guarantees. In *IJCAI*, pages 294–300.

Ma, H., Honig, W., Kumar, T., Ayanian, N., and Koenig, S. (2019). Lifelong path planning with kinematic constraints for multi-agent pickup and delivery. In *AAAI*.

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

Miyashita, Y., Yamauchi, T., and Sugawara, T. (2023). Distributed Planning with Asynchronous Execution with Local Navigation for Multi-agent Pickup and Delivery Problem. In *AAMAS*.

Okumura, K., Machida, M., Défago, X., and Tamura, Y. (2019a). Priority Inheritance with Backtracking for Iterative Multi-agent Path Finding. In *Proc. of the Twenty-Eighth International Joint Conf. on Artificial Intelligence, IJCAI-19*.

Okumura, K., Tamura, Y., and Défago, X. (2019b). winPIBT: Expanded Prioritized Algorithm for Iterative Multi-agent Path Finding. *CoRR*, abs/1905.10149.

Sharon, G., Stern, R., Felner, A., and Sturtevant, N. R. (2015). Conflict-based search for optimal multi-agent pathfinding. *Artif.Intel*, 219:40–66.

Shimada, D., Miyashita, Y., and Sugawara, T. (2025). Path Finding with Flexible Provisional Booking in Multi-agent Pickup and Delivery Problems. In Arisaka, R., Sanchez-Anguix, V., Stein, S., Aydoğan, R., van der Torre, L., and Ito, T., editors, *PRIMA 2024: Principles and Practice of Multi-Agent Systems*, pages 74–80, Cham. Springer Nature Switzerland.

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

Standley, T. S. (2010). Finding optimal solutions to cooperative pathfinding problems. In *AAAI*, volume 1, pages 28–29.

Wagner, G. and Choset, H. (2015). Subdimensional expansion for multirobot path planning. *Artif.Intell*, 219:1–24.

Wang, Z. (2023). Research on optimal route planning and delivery strategy of multiple robots using HCA algorithm in a restaurant. In *2023 IEEE 2nd Int. Conf. on Electrical Engineering, Big Data and Algorithms (EEBDA)*, pages 1740–1746.

Yamauchi, T., Miyashita, Y., and Sugawara, T. (2022). Standby-Based Deadlock Avoidance Method for Multi-Agent Pickup and Delivery Tasks. In *AAMAS*.