2026 IEEE International Conference on Robotics and Automation (ICRA 2026)
June 1-5, 2026. Vienna, Austria

Dual-layer PIBT for Lifelong Multi-agent Pathfinding with Narrow Passages

Zhenyu Song, Ronghao Zheng, Senlin Zhang, Meiqin Liu

***Abstract*— Lifelong Multi-Agent Pathfinding (Lifelong MAPF) is an extension of the Multi-Agent Pathfinding (MAPF) problem. It has significant applications in scenarios such as warehouse logistics and delivery services. Narrow passages that restrict side-by-side traversal are common in such scenarios, posing a major challenge to lifelong MAPF problem. To address this issue, this paper proposes dual-layer PIBT, a lifelong MAPF method specifically designed for biconnected environments containing narrow passages. The method leverages loop decomposition of the biconnected graph to establish coordinated unidirectional constraints - all narrow passages belonging to the same loop are assigned consistent traversal directions, enabling rapid conflict-free navigation decisions. The experimental results demonstrate significant reductions in both makespan and task service time compared to the baseline method.**

I. INTRODUCTION

The MAPD [1] (Multi-Agent Pickup and Delivery) problem extends the MAPF problem by introducing a continuous stream of delivery tasks that agents must handle online. In MAPD problems, each task requires an agent to travel to a specified pickup location and then to a delivery location, all while avoiding collisions with other agents. This problem is particularly relevant in automated warehouse systems and other logistics applications where agents are persistently engaged in new tasks.

A. Related Works on Lifelong MAPF

The lifelong MAPF problem is a key aspect of the MAPD problem. It differs from classic MAPF by replacing the objective of a one-time, simultaneous goal achievement with a requirement for continuous, online task completion. This continuous operation demands significantly higher real-time performance and scalability from the planning algorithm. Several studies have focused on this problem, aiming to plan collision-free paths for all agents to complete their assigned tasks. The foundational work [2] first established completeness guarantees for "well-formed instances". In such instances, the endpoints (all pickup and delivery locations of tasks and initial locations of agents) are positioned in a way that no agent, when stationed at an endpoint, can block other agents from moving between two other endpoints. Building on this, subsequent studies such as [3], [4], [5], [6] have developed approaches for the "well-formed instance". For example, [3] introduced a framework that adapts classical MAPF algorithms, such as CBS, ECBS [7], to solve lifelong MAPF via fixed-horizon decomposition. [4] and [5], both based on the A* algorithm, propose enhancements for lifelong MAPF that incorporate conflict-resolution strategies and heuristic functions tailored to dynamic task settings. [6] adopts the large neighborhood search algorithm to assign a sequence of tasks to each agent and repeatedly applies a MAPF algorithm to compute collision-free paths. However, the well-formed MAPF instance imposes strict structural constraints, including the requirement to reserve some collision-free pathways for agents, which are often impractical in practical multi-agent systems, especially when maximizing space utilization is crucial.

Some other methods, such as [8], [9], [10], [11], impose no specific constraints on the distribution of the endpoints, allowing them to address more general lifelong MAPF problems compared to the aforementioned methods. [8] is a decentralized lifelong MAPF algorithm designed to ensure collision-free movement in environments without narrow passages. The method employs "figure-8" swaps to resolve deadlocks and uses a count-based protocol for deadlock detection. Priority inheritance with backtracking (PIBT) [9] is an algorithm capable of solving both MAPF and lifelong MAPF problems. PIBT operates by dynamically assigning a unique priority to each agent and employs a combination of priority inheritance and backtracking to resolve deadlocks, ensuring that all agents eventually reach their destinations on biconnected graphs (details on biconnected graphs are provided in SectionII-B). In each time step, all agents attempt to move in descending order of their priorities. If a lower-priority agent occupies the desired position of a higher-priority agent, it temporarily inherits the higher priority and searches for an alternative vertex, possibly pushing an even lower-priority agent. This process guarantees progress and prevents deadlocks, enabling collision-free planning on biconnected graphs. MAPF-GPT [10] adopts imitation learning to train decentralized policies directly from expert trajectories, whereas PRIMAL2 [11] integrates imitation and reinforcement learning to improve lifelong MAPF performance. Both methods are based on local observations, enabling distributed execution and scalability to large agent

This work was supported in part by the National Natural Science Foundation of China under Grants U25B6002 and U23B2060, and in part by the Zhejiang Provincial Natural Science Foundation of China under Grant LZ24F030001. (*Corresponding author: Ronghao Zheng.*)

Zhenyu Song, Ronghao Zheng, and Senlin Zhang are with the College of Electrical Engineering, Zhejiang University, Hangzhou 310027, China, and also with the State Key Laboratory of Industrial Control Technology, Zhejiang University, Hangzhou 310027, China. (e-mail: {zhenyusong, rzheng, slzhang}@zju.edu.cn)

Meiqin Liu is with the Institute of Artificial Intelligence and Robotics, Xi'an Jiaotong University, Xi'an 710049, China, and also with the State Key Laboratory of Industrial Control Technology, Zhejiang University, Hangzhou 310027, China. (e-mail: liumeiqin@zju.edu.cn)

populations.

Although the methods in [12], [8], [9], [10], [11] do not impose constraints on the endpoints of Lifelong MAPF, narrow passages pose significant challenges to their effectiveness. Specifically, [12], [8] cannot handle environments with narrow passages. Narrow passages also present substantial challenges for the PIBT algorithm, as resolving conflicts within them often requires some agents to fully withdraw, incurring significant time and travel costs. Moreover, since PIBT only resolves conflicts reactively rather than predicting and avoiding them, its efficiency is notably reduced in environments with narrow passages. MAPF-GPT [10] and PRIMAL2 [11], due to their reliance on purely local observations, are prone to head-to-head conflicts in long narrow passages and suffer from reduced efficiency in such scenarios. Unfortunately, narrow passages are not only prevalent but also unavoidable in many space-efficient environments, such as high-density warehouse systems and automated factories.

B. Related Works on MAPF in Environments with Narrow Passages

Given the challenges and significance of narrow passages in MAPF and lifelong MAPF as mentioned above, several studies, such as [13], [14], [15], [16], [17], have specifically investigated MAPF in environments with narrow passages. [13] extends [12] by abstracting certain narrow-passage scenarios into 5×2 subgraphs, which are solved offline and stored in a database for conflict resolution at runtime. However, the method is limited to a narrow class of cases and is not applicable to scenarios involving branched structures, such as T-junctions. [14] proposes ALPHA, a learning-based method that incorporates global environment information to handle narrow passages in highly structured, room-like maps. However, it is designed for the one-shot MAPF setting rather than lifelong MAPF. [15] proposes BIBOX, an algorithm that guarantees completeness for MAPF on bi-connected graphs, which typically encompass environments with narrow passages. However, it is designed for one-shot MAPF and cannot be applied to lifelong MAPF with streaming tasks. [16] tackles MAPF in environments with narrow passages by enforcing "one-way traffic constraints", which require all agents to move in the same direction within a passage. The method formulates the direction assignment as an integer programming (IP) problem, accelerated via heuristics for initial feasible solutions. [17] extends [16] by re-planning narrow passage directions dynamically. While [16] assigns one-way directions based on the initial distribution of the start and goal locations of the agents, [17] updates these directions as the agents complete their tasks.

However, both [16], [17] are limited to handling pathfinding within warehouse environments consisting solely of narrow passages and single-grid intersections. Furthermore, since the optimization models in [16], [17] do not incorporate the time dimension, neither [16] nor [17] can be applied to lifelong MAPF problems. Moreover, as both approaches rely on integer programming, their computation time increases rapidly in environments with a large number of agents (e.g., over 1 minute for 100 agents), making them unsuitable for meeting the real-time requirements of lifelong MAPF.

C. Contributions

Driven by the challenges discussed above, this paper introduces a dual-layer PIBT algorithm to tackle the lifelong MAPF problem in biconnected graphs with narrow passages. The proposed method operates in two layers: the upper layer constructs a topological graph where narrow passages are represented as edges and non-narrow regions as vertices; the lower layer employs PIBT to resolve conflicts among agents. In the upper layer, loop decomposition[18] is utilized to process the topological graph, and all narrow passages within each loop are uniformly directed based on agent priorities. This approach ensures the connectivity of the directed graph while avoiding the computational overhead of solving complex integer optimization models. Additionally, a dynamic re-planning method is designed based on the occupancy of each loop, effectively reducing the average time required for agents to reach their goals in lifelong MAPF scenarios. The main contributions of this work are summarized as follows:

- A dual-layer PIBT framework is proposed for lifelong MAPF in biconnected graphs with narrow passages. In contrast to the original PIBT algorithm [9], the upper layer proactively assigns traversal directions to narrow passages, thereby preventing head-on collisions and enabling more efficient conflict resolution.
- Unlike previous methods [16], [17] that are specifically designed for warehouse-like environments consisting solely of regions no wider than 2 grid cells, the proposed method can operate in more general environments that contain both narrow and non-narrow regions.
- Extensive experiments show that the proposed method significantly outperforms [9] in terms of task completion efficiency. Compared to [16], [17], it achieves comparable path lengths while requiring substantially less computation time.

II. PRELIMINARY

A. Lifelong Multi-Agent Pathfinding Problem

A Lifelong MAPF instance is defined on an undirected graph $G = (V, E)$ with a set of agents $\mathscr{A} = \{A_1, A_2, \ldots, A_n\}$, where $|\mathscr{A}| = n < |V|$. Each agent A_i has a unique start vertex $s_i \in V$, but unlike MAPF, it is not assigned a fixed goal vertex in advance. Instead, agents are continuously assigned goals from a stream $\Lambda = \{g_1, g_2, \ldots\}$. Let $\pi_i[t] \in V$ denote the location of agent A_i at timestep t. Agents must avoid two types of conflicts: (1) *vertex conflicts*, where two agents occupy the same vertex at the same time ($\pi_i[t] = \pi_j[t]$), and (2) *swap conflicts*, where two agents exchange positions simultaneously ($\pi_i[t] = \pi_j[t+1] \wedge \pi_i[t+1] = \pi_j[t]$). A complete lifelong MAPF algorithm guarantees that each goal in Λ will be reached in a finite number of timesteps after being assigned to an agent.

B. Biconnected Graph and Loop Decompose

An undirected graph $G = (V, E)$ is connected if, for every pair of distinct vertices $v_i, v_j \in V$, there exists a path between them, i.e., a sequence of edges that connects v_i to v_j. An undirected $G = (V, E)$ is biconnected if it is connected and remains connected after removal of any single vertex $v_i \in V$. In biconnected graphs, every pair of vertices is contained in at least one simple cycle. A simple cycle is a sequence of vertices $(u, \ldots, v_i, v_{i+1}, \ldots, u)$ where $u, v_i, \ldots$ in this cycle are distinct, and each consecutive pair of vertices in this cycle is connected by at least one edge in G.

Loop decomposition, also referred to as ear decomposition, provides a structural characterization of biconnected graphs. [19] proved that an undirected biconnected graph G can be decomposed into a sequence of subgraphs $\mathscr{L} = (L_0, L_1, \ldots, L_k)$, where L_0 is a simple cycle and each subsequent subgraph $L_i = (u, \ldots, v_i, \ldots, v)$ is a path such as $u, v \in \bigcup_{j=0}^{i-1} L_j$ and $v_i \notin \bigcup_{j=0}^{i-1} L_j$. All vertices in the path are distinct, except for u and v, which may be the same if L_i is a cycle. The union of all subgraphs $\bigcup_{i=0}^{k} L_i$ is equal to G. The two endpoints of each path L_i are denoted by $e_a(L_i)$ and $e_b(L_i)$, and the internal vertices of L_i are distinct from the endpoints. Let $\vec{G}$ denote the directed version of G, and $\vec{L}_i$ for $i \in \{0, \ldots, k\}$ denote the directed versions of L_i. The traversal direction $D(\vec{L}_i)$ of $\vec{L}_i$ (for $i \neq 0$) is defined as either $[e_a(L_i) \to e_b(L_i)]$ or $[e_b(L_i) \to e_a(L_i)]$, representing traversal from $e_a(L_i)$ to $e_b(L_i)$ or vice versa, respectively. For the loop $\vec{L}_0$, its direction $D(\vec{L}_0)$ is characterized as either clockwise or counterclockwise. Fig. 1 illustrates an example of loop decomposition for a biconnected graph.

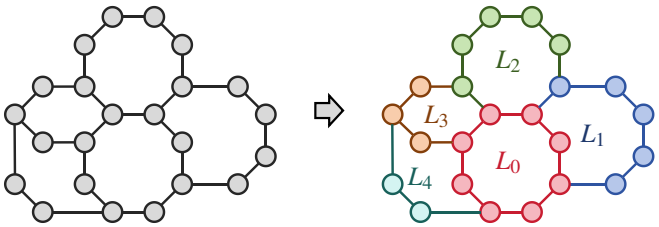


Fig. 1: Loop decomposition of a biconnected graph. This figure illustrates one possible decomposition of a biconnected graph $G = (V, E)$ into a sequence of subgraphs $\mathscr{L} = (L_0, L_1, L_2, L_3, L_4)$. The left part shows the original graph, and the right part displays one of its possible decompositions into loops. L_0 is a simple cycle, and each subsequent subgraph $L_1, \ldots, L_4$ is a path where the endpoints are part of the union of earlier subgraphs, while the internal vertices are distinct from those in the previous subgraphs.

III. DUAL-LAYER PIBT

This section introduces the dual-layer PIBT approach, which aims to address the lifelong MAPF problem in biconnected graphs with narrow passages. The approach is structured into two layers:

- **Upper Layer:** Focuses on assigning traversal directions to all narrow passages in the environment to prevent head-on conflicts between agents.
- **Lower Layer:** Utilizes the PIBT algorithm to plan conflict-free paths for each agent. In narrow passages, PIBT operates under the directional constraints provided by the upper layer. In non-narrow areas, it executes the standard PIBT algorithm.

The principles of the upper layer and how it integrates with the lower layer will be elaborated in the subsequent sections.

A. Upper Layer: Topological Graph Construction and Loop Decomposition

Let $\mathscr{N}(v)$ denote the set of neighboring vertices of v. In this work, a vertex v is considered part of a narrow passage if it has exactly two neighbors v_1 and v_2 and they do not share any common neighbors except v, i.e., $\mathscr{N}(v_1) \cap \mathscr{N}(v_2) = \{v\}$. A narrow passage is then defined as a sequence of such connected vertices, in which every vertex satisfies this condition. The geometry of these structures prevents side-by-side movement, often leading to congestion and potential deadlocks. Fig. 2 provides a simple example to illustrate narrow passages in a grid map. The grid cells within the red dashed box represent a narrow passage in the environment, while the grid cells within the blue and green dashed boxes represent non-narrow passage regions.

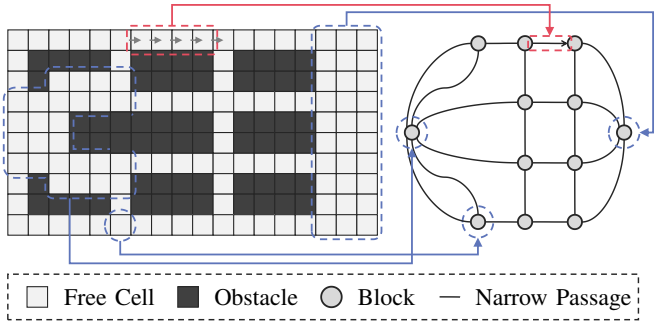


Fig. 2: An example of a 4-connected grid map with narrow passages. The figure illustrates a simplified warehouse environment, where black cells represent obstacles and white cells denote free spaces. The right part of the figure shows the corresponding topological graph, where vertices represent non-narrow regions (referred to *blocks*), and edges indicate narrow passages.

Resolving conflicts within narrow passages can lead to significant overhead in terms of makespan and service time, as agents may need to withdraw to avoid head-on conflicts. To address this issue, the proposed method assigns traversal directions to narrow passages, preventing agents from entering from opposite ends simultaneously.

To assign traversal directions to narrow passages, a topological graph $\mathscr{G}$ of the environment G is constructed, where narrow passages are represented as edges and non-narrow regions (referred to as *blocks* in the following sections) are represented as vertices. This abstraction enables a high-level analysis of the environment that highlights the interconnections between narrow passages and blocks. On this topological graph $\mathscr{G}$, the traversal directions for narrow passages are planned, which are then translated into directional constraints on G. Fig. 2 shows the corresponding topological graph on the right and the grid map on the left. The traversal directions of edges in the red dashed box (right) are mapped to the grid cells (left) accordingly.

After constructing the topological graph $\mathscr{G}$ of the environment, the traversal directions for narrow passages can be planned on it. It is important to note that narrow passages are not independent; their traversal directions are interrelated. Assigning directions independently may create *sources*, *sinks*, and other disconnected regions in the graph, as shown in Fig. 3, leading to the failure of the path planning task. This is particularly critical in MAPF and lifelong MAPF, as it can disrupt the connectivity of $\mathscr{G}$ and G. In lifelong MAPF, agents may be assigned tasks that require them to travel to any vertex in G after completing their current task, making it essential to ensure that G remains globally connected at all times for successful task completion.

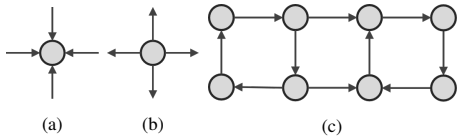


Fig. 3: Assigning directions independently can result in a disconnected graph. (a) shows a *sink*, where agents on this vertex cannot reach other vertices. If an agent is located on this vertex, the path planning task cannot be completed. (b) shows a *source*, where agents from other vertices cannot reach this vertex. If an agent's destination is on this vertex, the path planning task will also fail. In (c), there is neither a source nor a sink structure, but the four vertices on the right cannot reach the four vertices on the left, leading to a disconnected region within the graph.

Existing methods, such as [16], [17], which are based on integer programming, address this issue by adding connectivity constraints to their optimization models. However, these methods suffer from high computational complexity, especially in environments with many narrow passages or agents, as will be discussed in Section IV. To address the aforementioned issues, this paper leverages the properties of biconnected graphs and loop decomposition. Assigning traversal directions at the level of loops, rather than individual narrow passages, preserves the connectivity of the graph , as demonstrated in Theorem 1, thus preventing disconnections during path planning. This method avoids the construction of complex reachability constraints and the need to solve optimization models, ensuring the rationality of any assigned directions. Consequently, it provides a feasible solution for real-time narrow passage traversal direction planning.

Theorem 1: Let G be a biconnected graph, and $(L_0, L_1, \ldots, L_k)$ be one of its loop decompositions. For any direction $(D(\vec{L}_0), D(\vec{L}_1), \ldots, D(\vec{L}_k))$, $\vec{G} = \bigcup_{j=0}^{k} \vec{L}_i$ is strongly connected.

Proof: We demonstrate strong connectivity by induction in loop decomposition $(L_0, L_1, \ldots, L_k)$. For the base case, L_0 is a cycle and remains strongly connected under any direction. For the inductive step, assume that $\bigcup_{j=0}^{i-1} L_j$ is strongly connected. Consider the next loop L_i, which is a path with endpoints $e_a(L_i)$ and $e_b(L_i)$ in $\bigcup_{j=0}^{i-1} L_j$. Without loss of generality, assume that L_i is oriented as $e_a(L_i) \to e_b(L_i)$. For any $v_i \in L_i$ and $v_j \in \bigcup_{j=0}^{i-1} L_j$, there exists a path $v_i \to e_b(L_i) \to v_j$, and a path $v_j \to e_a(L_i) \to v_i$. For any $v_i, v_j \in L_i$, there exists a path $v_i \to v_j$ (if v_i precedes v_j in L_i) or $v_i \to e_b(L_i) \to e_a(L_i) \to v_j$. Thus, $\bigcup_{j=0}^{i} L_j$ remains strongly connected. By induction, the entire graph G remains strongly connected after assigning directions to all loops. ■

Remark 1: While global connectivity of the graph G is not strictly necessary for narrow passage traversal planning — since it only requires ensuring that agents can find paths to their current goals — it becomes crucial in the lifelong MAPF problem. In this context, agents are dynamically assigned new tasks, and their target nodes may be located anywhere in G. If the traversal directions of narrow passages do not guarantee global reachability, agents may fail to reach these dynamically assigned goals. To ensure robustness against such dynamic task assignments, the proposed method ensures that G maintains global connectivity at all times.

B. Upper Layer: Priority-Based Directional Assignment in Narrow Passages and Re-planning

As mentioned above, the assignment of the direction of narrow passages based on loop decomposition ensures the global connectivity of G. A critical challenge lies in determining the traversal direction for each loop in a real-time manner. Existing works, such as [16], [17], plan traversal directions for narrow passages by formulating optimization problems where the objective is to minimize the sum of agents' shortest-path lengths, but neglect multi-agent interactions. However, the computational time of these optimization-based methods increases rapidly with the number of agents and the size of the environment, making it difficult to meet the real-time requirements of lifelong MAPF problems. This limitation will be analyzed in detail in Section IV-C.

To address this issue, this work proposes a priority-based direction assignment scheme. In this scheme, each loop is assigned a traversal direction based on the priorities of the agents currently navigating through it. The priorities of the agents are determined by the rules of the PIBT algorithm. Specifically, let $p_i[t]$ denote the priority of agent A_i at time step t. Each agent is assigned an initial priority $p_i[0] = \varepsilon_i$ upon receiving a new task, where $\varepsilon_i \in (0,1)$ is a unique constant for each agent. In this work, the initial priorities are assigned randomly. Given that tasks are generated continuously with start and goal locations changing over time, a random assignment therefore serves as a simple yet effective default. Subsequently, the priority of each agent is updated at each time step as $p_i[t+1] = p_i[t] + 1$ until the task is completed. The agent with the highest priority first plans the shortest path from its current vertex to its current goal on the graph G. The traversal directions of the loops are then determined based on this path: loops that are part of the path are oriented in the direction of the path, while loops not involved in the path remain unchanged. Subsequently, agents with lower priority plan their paths on the updated graph, determining the directions of the loops accordingly. This process continues until either all loop traversal directions are determined or all agents have planned their optimal paths. Theorem 1 guarantees the global connectivity of the bicon-

nected graph G by assigning directions to loops, ensuring the feasibility of this priority-based direction assignment scheme. Fig. 4 provides an example to illustrate this process.

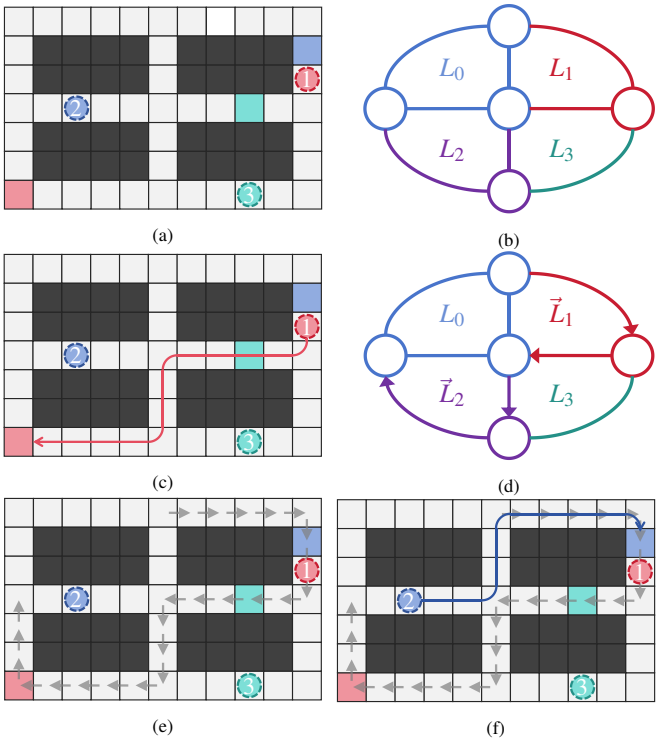


Fig. 4: Traversal direction assignment for loops. Circles with dashed borders represent agents, and grids with matching colors indicate their goals. Three agents, A_1, A_2, and A_3, are ordered by decreasing priorities. (a) displays their initial positions and goal locations. (b) illustrates the topological graph and loop decomposition of the environment in (a). Different loops are distinguished by distinct colors. In (c), the highest-priority agent A_1 plans an optimal path to its goal. This path traverses loops L_1 and L_2. Consequently, L_1 and L_2 are assigned traversal directions as shown in (d). The vertices within $\vec{L}_1$ and $\vec{L}_2$ are then constrained by the traversal directions, as depicted in (e). The remaining agents then iteratively follow this process until all agents have planned their paths or all loops have been assigned directions. Each agent must adhere to the traversal direction constraints imposed by higher-priority agents when planning its optimal path. For example, as shown in (f), agent A_2 plans its optimal path on a graph where certain vertices have already been assigned traversal directions.

Although this method does not guarantee optimality as optimization-based approaches do, it is capable of efficiently handling environments with a large number of agents. The time complexity of this priority-based direction assignment scheme is $\mathcal{O}(|\mathscr{A}|)\cdot T(|V|,|E|)$, where $T(|V|,|E|)$ denotes the time complexity required for an agent to find an optimal path on the graph. The runtime experiments with varying numbers of agents in Section IV-C further verifies the efficiency of the proposed method.

Although the initial assignment of directions ensures global connectivity, it does not account for dynamic changes in the environment as agents complete tasks and receive new ones. In such cases, the previously assigned traversal directions may no longer be suitable. Inspired by [17], this paper introduces a re-planning mechanism for loop traversal directions. Specifically, when a loop is occupied by only one agent or no agent, its traversal direction can be revised (re-planning state). Whenever any loop transitions into a re-planning state, the mechanism clears the traversal direction of any adjustable loop and re-executes the priority-based direction assignment algorithm while preserving the directions of other loops. Agents then reselect the traversal direction for the adjusted loop based on their current positions and task priorities. By allowing adjustments based on the system's real-time state, this re-planning mechanism dynamically optimizes loop traversal directions to improve overall path quality.

C. Lower Layer

When the traversal directions of the narrow passages are determined at the upper layer, these directions are applied to vertices that lie within these narrow passages in the graph G, and the lower layer utilizes the PIBT algorithm to plan paths for each agent. For vertices located in narrow passages, some of their adjacent vertices may become unreachable due to the imposed traversal directions. Let $\mathscr{N}^c(v)\subseteq\mathscr{N}(v)$ denote the set of reachable neighbors of v under the traversal constraints. The original PIBT algorithm iterates over all neighboring vertices $\mathscr{N}(v)$ of the agent's current location v to find feasible movement directions and selects the one closest to the agent's goal. To ensure that the agent's movement aligns with the traversal constraints imposed by the upper layer, this method modifies PIBT to iterate over the reachable neighbors $\mathscr{N}^c(v)$ instead. This adaptation guarantees that the agent's movement respects the directional constraints while maintaining conflict-free paths. Fig. 5 provides a concrete example demonstrating the relationship between $\mathscr{N}^c(v)$ and $\mathscr{N}(v)$.

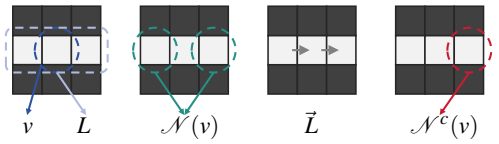


Fig. 5: Neighbor and reachable neighbor of a vertex. For directed $\vec{L}$, all grids belonging to L are subject to the traversal constraints imposed by the direction of $\vec{L}$. The reachable neighbors of a vertex represent the manifestation of these constraints at the grid level. Specifically, the reachable neighbors of a vertex form a subset of its neighbors, and an agent located at the vertex can only move to its reachable neighbors. This ensures compliance with the traversal direction of $\vec{L}$.

A critical question is whether PIBT can still plan conflict-free paths for agents on the directed graph $\vec{G}$. Let $\mathrm{diam}(\vec{G})=\max_{u,v\in V} d(u,v)$, where $d(u,v)$ is the length of the shortest directed path from u to v. Theorem 2 demonstrates that, for any loop decomposition of G and any direction of the loops, PIBT ensures that each agent reaches its goal within a finite number of timesteps. Theorem 2 guarantees the effectiveness of the modified PIBT algorithm to satisfy both the traversal constraints and the path planning requirements.

Theorem 2: For an undirected biconnected graph G and its loop decomposition $(L_0,\ldots,L_k)$, under any direction of $D(\vec{L}_0)$ to $D(\vec{L}_k)$, the PIBT algorithm generates a set of conflict-free paths $\{\pi_1,\ldots,\pi_n\}$ such that for any agent $A_i\in$

$\mathscr{A}$, $\pi_i[t'] = s_i$, and there exists a timestep $t \leq \mathrm{diam}(\vec{G}) \cdot |\mathscr{A}|$ such that $\pi_i[t'+t] = g_i$.

Proof: We prove that every pair of adjacent vertices $v_i, v_j \in V$ belongs to a simple cycle. If v_i and v_j are in the same loop P_m, then P_m itself forms a simple cycle. Otherwise, let P_m be the loop containing the edge $e = (v_i, v_j)$. Since G is strongly connected, there exists a shortest directed path $Q_1 : v_i \to v_j$ and a shortest directed path $Q_2 : v_j \to v_i$. Combining Q_1 and Q_2 forms a simple cycle C containing v_i and v_j. Thus, the theorem holds.

Under this condition, Theorem 1 of [9] proves that the PIBT algorithm generates a set of conflict-free paths $\{\pi_1, \ldots, \pi_n\}$ such that for any agent $a_i \in A$, $\pi_i[t'] = s_i$, and there exists a timestep $t \leq \mathrm{diam}(G) \cdot |A|$ such that $\pi_i[t'+t] = g_i$. Analogously for directed graph $\vec{G}$, the bound becomes $t \leq \mathrm{diam}(\vec{G}) \cdot |A|$. ■

IV. EXPERIMENTS

A. Setup

This paper evaluates the proposed method against baseline approaches under standard MAPD tasks. In an MAPD instance, agents are continuously assigned tasks from a stream $\Gamma = \{\tau_1, \tau_2, \ldots\}$, where each task $\tau_j = (v_{\text{pickup}}, v_{\text{delivery}})$ requires the assigned agent to visit a pickup location v_{pickup} followed by a delivery location v_{delivery}. Specifically, we assign tasks to the nearest idle agent. This task assignment strategy is widely employed in prior works [1], [3], [4], [18]. The experimental setup includes the following components:

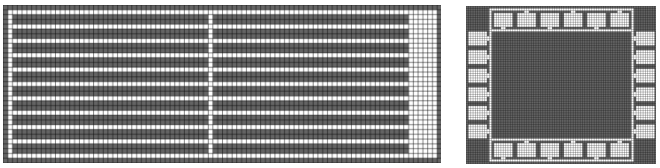
Environments: Since the proposed method is specifically designed for lifelong MAPF tasks in biconnected graphs with narrow passages, this work selects *warehouse* [3] and *office* environments for MAPD tasks, which are commonly used in MAPF and lifelong MAPF problems. These environments contain narrow passages and form biconnected structures.

Evaluation Metrics: The performance of MAPD solutions is typically evaluated using two key metrics: (1) *Makespan*, defined as $\max_{1 \leq i \leq n} t_i$, which is the time when the last agent finishes its final task; (2) *Service Time*, defined as $\frac{1}{|\Gamma|} \sum_{i=1}^{|\Gamma|} \Delta t_{\tau_i}$, where Δt_{τ_i} is the time elapsed from when task τ_i is assigned to when it is completed. Smaller values for these metrics indicate higher solution quality.

Baseline: The primary baseline for comparison is the PIBT [9] algorithm, chosen for its ability to guarantee solvability in biconnected graphs. Unlike other methods that may fail in environments with narrow passages, PIBT ensures conflict-free paths under the proposed direction assignment scheme, making it a suitable benchmark for evaluating our approach.

Task Configuration: All evaluations of *Makespan* and *Service Time* involve 500 pickup-and-delivery tasks executed by up to 500 agents. The simulation is terminated if the number of timesteps exceeds 10,000 or the total computation time exceeds 250 seconds. This implies that, on average, a feasible plan must be generated for each task within 0.5 seconds. For online planning algorithms, this latency is generally sufficient to satisfy the real-time requirements of MAPD in practical scenarios. The task allocation strategy follows the same approach as [9], where each task is assigned to the nearest available agent relative to the pickup location. Tasks are introduced at a fixed frequency, consistent with the settings in [9]. Specifically, four task frequencies are evaluated, corresponding to 1, 2, 5, and 10 new tasks added to the task set Γ per timestep.

Computer Platform: All experiments are conducted on a workstation equipped with an Intel® Core™ i5-13600KF CPU and 32 GB RAM.



(a) (b)

Fig. 6: (a) is a 92×33 warehouse-style environment with 10 rows and 2 columns of shelves, consisting of 1,150 free cells and 1,886 obstacles. (b) is an 82×68 office environment featuring 24 rooms, composed of 1,572 free cells and 4,004 obstacles.

B. Makespan and Service Time

Fig. 7 presents partial experimental results in *office* and *warehouse* with a *task frequency* of 10. The complete experimental data can be found in Table I. The experiments compared the original PIBT algorithm and the proposed Dual-layer PIBT (DL-PIBT) algorithm. In all experiments, all algorithms were able to complete the planning in 250 seconds, meeting real-time requirements. Overall, as the number of agents increased, DL-PIBT demonstrated increasingly superior performance over PIBT in terms of makespan and service time. However, in the *warehouse* environment with a lower number of agents (100 agents), PIBT outperformed both DL-PIBT. This is because, under low task loads, conflicts between agents are less frequent and the *warehouse* environment contains numerous intersections, which are advantageous for the yield behavior of agents in the PIBT algorithm. In contrast, DL-PIBT, which restricts the direction of movement in narrow passages, shows some disadvantages in these metrics, although the replanning mechanism can partially mitigate this issue. As the number of agents increases and conflicts become more frequent, the advantages of DL-PIBT become more pronounced.

C. Computation Time and Optimal Rate

In previous work such as [16], [17], optimization-based algorithms were employed to solve the narrow passage direction assignment problem by incorporating path connectivity constraints to ensure the reachability of all agents while minimizing the sum of their path lengths. These methods provide an optimal direction assignment for narrow passages under one-way traffic constraints, but suffer from significant computational overhead. In contrast, the method proposed in this work, which is based on loop decomposition and priority assignment, does not guarantee optimality. Therefore, this section compares the quality of the planned passage directions and the computation time required by both methods.

TABLE I: Experimental Results.

Agents	Method	Office								Warehouse							
		TF=1		TF=2		TF=5		TF=10		TF=1		TF=2		TF=5		TF=10	
		ST	MS	ST	MS	ST	MS	ST	MS	ST	MS	ST	MS	ST	MS	ST	MS
100	PIBT	282.30	1120.92	372.76	**905.84**	427.85	1072.72	448.82	1065.96	**148.29**	**829.28**	**217.65**	**777.48**	**270.38**	**747.16**	**291.97**	**738.16**
	DL-PIBT	**256.59**	**1061.32**	**337.89**	1027.00	**389.31**	**1004.72**	**398.93**	**991.04**	202.71	938.24	266.94	865.32	314.88	830.00	332.93	825.56
200	PIBT	232.33	980.48	292.37	905.84	338.98	858.88	347.79	858.56	142.24	818.44	190.38	712.16	224.83	660.36	238.73	660.92
	DL-PIBT	**159.31**	**784.24**	**200.37**	**692.40**	**229.96**	**647.36**	**243.87**	**632.84**	**140.69**	**769.80**	**161.30**	**607.04**	**190.97**	**557.96**	**201.65**	**537.92**
300	PIBT	246.66	967.12	291.00	869.64	322.91	815.04	243.88	811.16	167.71	867.56	203.81	742.88	228.54	682.80	233.25	656.72
	DL-PIBT	**158.62**	**775.84**	**181.38**	**631.60**	**207.54**	**576.80**	**215.95**	**559.68**	**144.24**	**767.92**	**149.82**	**571.56**	**167.91**	**513.40**	**169.84**	**482.12**
400	PIBT	265.67	978.36	303.79	872.96	328.82	817.52	340.86	794.56	195.37	911.56	224.74	787.28	245.53	720.76	254.05	712.04
	DL-PIBT	**170.97**	**787.08**	**191.15**	**642.12**	**212.78**	**576.84**	**218.89**	**561.76**	**160.33**	**806.88**	**162.44**	**603.24**	**165.43**	**496.20**	**171.94**	**489.00**
500	PIBT	286.18	1000.24	316.33	871.52	340.76	829.28	346.49	822.60	227.35	982.32	258.82	858.24	269.06	765.36	282.10	781.20
	DL-PIBT	**184.30**	**811.56**	**203.64**	**651.64**	**219.00**	**588.24**	**222.18**	**556.84**	**193.09**	**892.24**	**179.15**	**661.08**	**184.30**	**567.88**	**180.45**	**518.96**

Note: TF = Task Frequency, ST = Service Time, MS = Makespan.

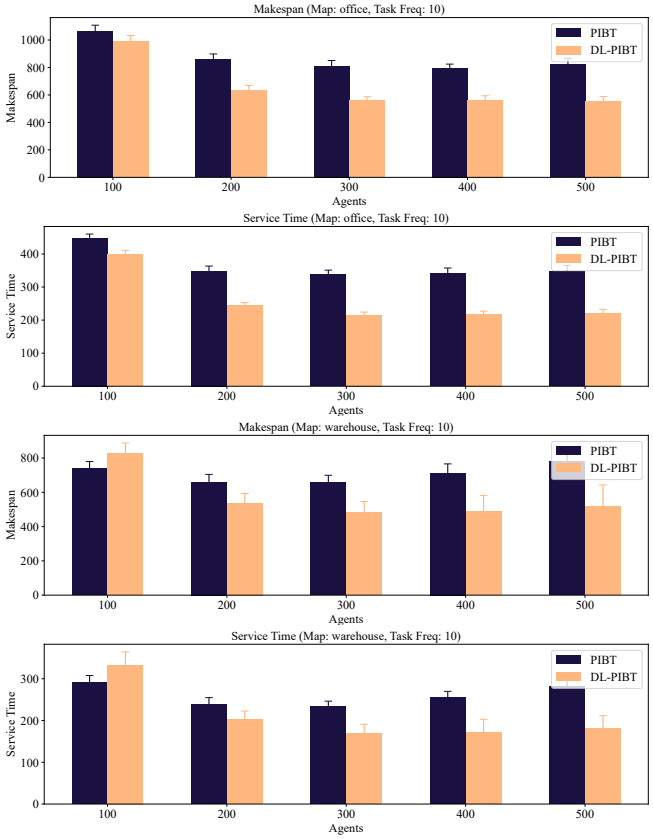


Fig. 7: Part of the experiment result. The top two figures depict the relationship between the number of agents and the metrics (*makespan* and *service time*), along with their standard deviations. These results are derived from for the PIBT and DL-PIBT algorithms in the "office" environment with a task frequency of 10. The bottom two figures present the corresponding outcomes in the "warehouse" environment.

Since optimization-based algorithms in previous works [16], [17] can only operate in environments consisting solely of narrow passages, the experimental setup is limited to a layout of 8×8 shelves (each with dimensions 10×2). Following the metric from [16], [17], this paper adopts the sum of all agents' shortest path lengths as the optimality criterion, matching the IP algorithm's optimization objective. Both Fig. 8 (computation time) and Fig. 9 (optimality rate) evaluate performance under varying numbers of agents.

The experimental results in Fig. 8 demonstrate that the proposed method achieves substantially faster computation

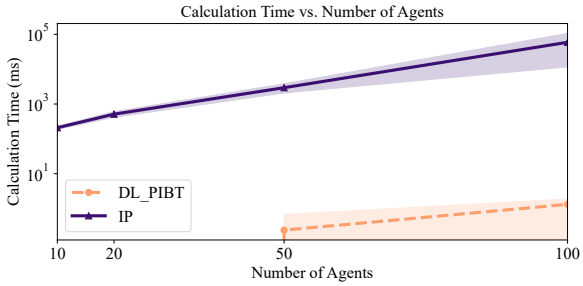


Fig. 8: Relationship between the number of agents and the planning time in narrow passage scenarios. The shaded regions represent the standard deviation of the data. Due to the significantly longer computation time required by the IP method compared to the proposed method, the y-axis is displayed in a logarithmic scale for better comparison. The IP method fails to solve problems with 200 or more agents within 10^6 ms. When the number of agents is fewer than 100, the proposed method can compute the passage direction in less than 1 ms.

TABLE II: Average Computation Time Required by the Proposed Method to Assign Traversal Directions with Different Numbers of Agents.

Number of Agents	100	200	300	400	500
Calculation Time (ms)	1.32	3.52	5.6	7.72	9.60

than the approach in [16]. In scenarios with 100 agents, the computation time is reduced by nearly five orders of magnitude. Table II further reports the time required by the proposed method to determine traversal directions in narrow passages for scenarios ranging from 100 to 500 agents. As shown, even with 500 agents, the proposed method completes the loop direction task in under 10 ms. This efficiency ensures real-time applicability in large-scale multi-agent systems involving more than 100 agents. Moreover, Fig. 9 illustrates that the optimality ratio between the proposed method and the IP-based optimal solution [16] remains consistently low across all test scenarios. Together, these results confirm that the proposed method can efficiently produce near-optimal narrow passage direction assignments while maintaining exceptional computational performance.

D. Real-World Experiments

To demonstrate the feasibility of the proposed method on physical robots, we conducted real-world experiments using *TurtleBot 3*, a two-wheeled differential-driven platform. Five robots were deployed in a warehouse-like environment to complete a total of 100 pickup and delivery tasks. The test

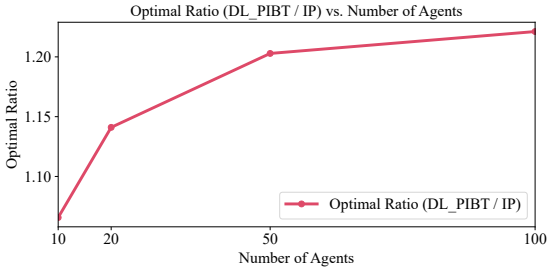
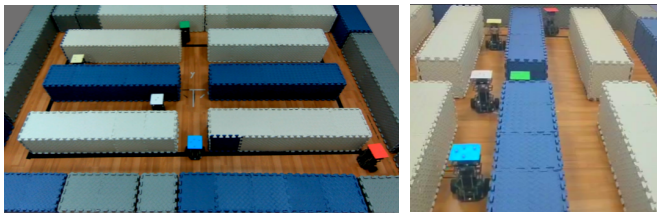


Fig. 9: Optimality rate of the proposed method under varying numbers of agents. The rate is calculated as the ratio between the total path length generated by the proposed method and that of the IP-based optimal solution. The proposed method maintains a consistently low optimality rate, indicating high solution quality with respect to the optimal IP-based benchmark.

arena measured 4.6 m $\times$ 2.76 m and contained 13 narrow passages. An overview of the test environment is shown in Fig. 10a. As illustrated in Fig. 10b, the passages are extremely narrow and do not allow two robots to pass side by side. The positions of the robots were tracked using the *FZMotion*® motion capture system. In the experiment, all robots operated at an average speed of approximately 0.3 m/s, and the five robots completed all 100 tasks with a makespan of 3 minutes and 47 seconds. A video of the experiment is provided in the supplementary material.



(a) (b)

Fig. 10: (a) Overview of the warehouse-like environment. (b) Execution snapshot showing that narrow passages prevent two robots from moving side by side.

V. CONCLUSIONS

This work proposes dual-layer PIBT, a scalable approach to lifelong MAPF in biconnected environments with narrow passages. By integrating loop decomposition with priority-based directional assignment, the proposed method effectively resolves head-on conflicts in narrow passages. Experimental results demonstrate that the proposed method consistently reduces both makespan and service time compared to baseline PIBT. Future research will explore extending the proposed method to more general factory layouts where the biconnectivity assumption may not hold. The design of adaptive direction assignment strategies will also be investigated to enhance responsiveness in highly dynamic settings. The approach will also be extended to heterogeneous multi-agent systems with diverse capabilities to assess its broader applicability.

REFERENCES

[1] H. Ma, J. Li, T. S. Kumar, and S. Koenig, "Lifelong multi-agent path finding for online pickup and delivery tasks," in *Proceedings of the 16th Conference on Autonomous Agents and MultiAgent Systems*, pp. 837–845, 2017.

[2] M. Čáp, J. Vokřínek, and A. Kleiner, "Complete decentralized method for on-line multi-robot trajectory planning in well-formed infrastructures," in *Proceedings of the International Conference on Automated Planning and Scheduling*, vol. 25, pp. 324–332, 2015.

[3] J. Li, A. Tinka, S. Kiesel, J. W. Durham, T. S. Kumar, and S. Koenig, "Lifelong multi-agent path finding in large-scale warehouses," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 35, pp. 11272–11281, 2021.

[4] F. Grenouilleau, W.-J. Van Hoeve, and J. N. Hooker, "A multi-label A* algorithm for multi-agent pathfinding," in *Proceedings of the International Conference on Automated Planning and Scheduling*, vol. 29, pp. 181–185, 2019.

[5] Z. Chen, J. Alonso-Mora, X. Bai, D. D. Harabor, and P. J. Stuckey, "Integrated task assignment and path planning for capacitated multi-agent pickup and delivery," *IEEE Robotics and Automation Letters*, vol. 6, no. 3, pp. 5816–5823, 2021.

[6] Q. Xu, J. Li, S. Koenig, and H. Ma, "Multi-goal multi-agent pickup and delivery," in *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 9964–9971, IEEE, 2022.

[7] S. Thomas, D. Deodhare, and M. N. Murty, "Extended conflict-based search for the convoy movement problem," *IEEE Intelligent Systems*, vol. 30, no. 6, pp. 60–70, 2015.

[8] H. Wang and M. Rubenstein, "Walk, stop, count, and swap: decentralized multi-agent path finding with theoretical guarantees," *IEEE Robotics and Automation Letters*, vol. 5, no. 2, pp. 1119–1126, 2020.

[9] K. Okumura, M. Machida, X. Défago, and Y. Tamura, "Priority inheritance with backtracking for iterative multi-agent path finding," *Artificial Intelligence*, vol. 310, p. 103752, 2022.

[10] A. Andreychuk, K. Yakovlev, A. Panov, and A. Skrynnik, "Mapf-gpt: Imitation learning for multi-agent pathfinding at scale," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 39, pp. 23126–23134, 2025.

[11] M. Damani, Z. Luo, E. Wenzel, and G. Sartoretti, "Primal _2: Pathfinding via reinforcement and imitation multi-agent learning-lifelong," *IEEE Robotics and Automation Letters*, vol. 6, no. 2, pp. 2666–2673, 2021.

[12] S. D. Han and J. Yu, "Ddm: Fast near-optimal multi-robot path planning using diversified-path and optimal sub-problem solution database heuristics," *IEEE Robotics and Automation Letters*, vol. 5, no. 2, pp. 1350–1357, 2020.

[13] R. A. Moan, C. McBeth, M. Morales, N. M. Amato, and K. Hauser, "Experience-based multi-agent path finding with narrow corridors," in *Robotics: Science and Systems*, 2024.

[14] C. He, T. Yang, T. Duhan, Y. Wang, and G. Sartoretti, "Alpha: Attention-based long-horizon pathfinding in highly-structured areas," in *2024 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 14576–14582, IEEE, 2024.

[15] P. Surynek, "A novel approach to path planning for multiple robots in bi-connected graphs," in *2009 IEEE international conference on robotics and automation*, pp. 3613–3619, IEEE, 2009.

[16] J. Huo, R. Zheng, S. Zhang, and M. Liu, "Dual-layer multi-robot path planning in narrow-lane environments under specific traffic policies," *Intelligent Service Robotics*, vol. 15, no. 4, pp. 537–555, 2022.

[17] Z. Song, R. Zheng, S. Zhang, and M. Liu, "Dynamic multi-robot path planning in narrow-lane environments with one-way constraint," in *2023 6th International Conference on Intelligent Autonomous Systems (ICoIAS)*, pp. 210–215, IEEE, 2023.

[18] D. J. Kavvadias, G. E. Pantziou, P. G. Spirakis, and C. D. Zaroliagis, "Hammock-on-ears decomposition: A technique for the efficient parallel solution of shortest paths and other problems," *Theoretical Computer Science*, vol. 168, no. 1, pp. 121–154, 1996.

[19] M. H. Carvalho and C. L. Lucchesi, "Ear decompositions of matching covered graphs," *Combinatorica*, vol. 19, no. 2, pp. 151–174, 1999.