IEEE Access
Multidisciplinary ⁝ Rapid Review ⁝ Open Access Journal

Received 14 February 2023, accepted 23 February 2023, date of publication 27 February 2023, date of current version 2 March 2023.

Digital Object Identifier 10.1109/ACCESS.2023.3249471

RESEARCH ARTICLE

Anytime Lifelong Multi-Agent Pathfinding in Topological Maps

SOOHWAN SONG, KI-IN NA, AND WONPIL YU
Intelligent Robotics Research Division, Electronics and Telecommunications Research Institute (ETRI), Daejeon 34129, South Korea

Corresponding author: Soohwan Song (soohwansong@etri.re.kr)

This work was supported by the Industrial Strategic Technology Development Program, Development of Heterogeneous Multi-Mobile Logistics Handling Robot Integrated Operation Simulator and Fleet Management System, through the Ministry of Trade, Industry and Energy (MOTIE), South Korea, under Grant 20015440.

ABSTRACT This study addresses a lifelong multi-agent path finding (lifelong MAPF) problem that continuously solves an MAPF instance online according to newly assigned goals. Specifically, we focus on lifelong MAPF in a topological map. This problem is challenging because the movement of the agent is restricted to narrow corridors in a topological map, rather than the entire map area. Bypasses may be limited or farther away in corridors, significantly complicating the computation of paths. Furthermore, low-quality solutions may cause traffic congestion or even deadlock in a corridor. Therefore, we propose a novel lifelong MAPF method that effectively resolves conflicts in corridors based on an anytime strategy. This method gradually improves the solution quality by updating sub-paths with heavy traffic congestion. Furthermore, we adopt several improvement steps to effectively resolve corridor conflicts in a conflict-based search (CBS). This method significantly reduces the search space and computation time of CBS. We conducted extensive experiments on various topological maps in warehouse and railway environments. The results show that the proposed method outperforms state-of-the-art methods in terms of throughput and success rate. In particular, the proposed method can resolve collisions with a longer time horizon than existing methods, considerably improving throughput on a topological map with long-range corridors and heavy traffic congestion.

INDEX TERMS Multi-agent pathfinding, mobile robots, logistics automation, path planning, multi-robot system, topological map.

I. INTRODUCTION

In automated warehouse systems, multiple mobile robots continuously process a sequence of incoming logistics tasks [1], [2]. Mobile robots move through warehouse spaces to deliver packages or inventory pods to each target location. For the autonomous navigation of mobile robots, the robot management system must continuously plan collision-free paths for all robots. This problem is known as *lifelong multi-agent path finding* (lifelong MAPF) and has recently been addressed in many studies [3], [4], [5], [6]. Lifelong MAPF is an online version of the MAPF problem [7], [8], [9] that determines the optimal collision-free paths from start locations to goal locations for agents. Here, an agent refers to a mobile unit such as a robot, vehicle, or train. A collision occurs when two agents simultaneously move to the same location at the same time step. Lifelong MAPF repeatedly solves an MAPF instance online according to newly assigned goal locations.

Most studies [3], [4], [5], [6] have defined the lifelong MAPF problem in a grid-like environment in the same way as Kiva systems [1], [2]. However, the Kiva system requires specialized setups to systematically operate many robots on a grid map, such as marker-based localization and a standardized space for Kiva robots. Therefore, it is difficult to apply this system to a general warehouse space or various types of mobile robots. To address this issue, we consider the lifelong MAPF problem on a topological map, where an environment is represented as a graph comprising nodes and lanes (Fig. 1a). Topological maps restrict a robot's movement to lanes, thereby improving the stability of navigation. In particular, large robots, such as automated guided forklifts,

The associate editor coordinating the review of this manuscript and approving it for publication was P. Venkata Krishna.

must move along lanes for safety purposes. Furthermore, topological maps can easily be applied to existing warehouse spaces with various structures.

However, lifelong MAPF on topological maps is challenging owing to several factors. First, corridor conflicts occur frequently when two agents simultaneously cross a corridor in opposite directions (Fig. 1b). MAPF algorithms unnecessarily search for many states to resolve this conflict, thereby significantly increasing computation time. For example, in a *conflict-based search* (CBS) [9], corridor conflict exponentially increases the search space according to corridor length [10]. Second, lifelong MAPF methods need to use an efficient algorithm, such as a windowed approach [6], [11] and suboptimal solvers [12], [13], for fast online computation. However, these algorithms generally produce low-quality solutions, which can cause traffic congestion in corridors, because detours are often limited to topological maps. Finally, a suboptimal solver sometimes causes deadlock in a dead-end corridor (Fig. 1c). To avoid this, the algorithm must exhaustively search for the optimal entry order for the dead-end corridor within a limited timeframe.

In this study, we propose a novel lifelong MAPF method that finds the best possible solution within a limited time by effectively resolving corridor conflicts in topological maps. This method consists of 1) Anytime lifelong MAPF algorithm and 2) Corridor-CBS algorithm. The anytime lifelong algorithm is based on the *rolling-horizon collision resolution* (RHCR) approach [6], which is one of the most efficient methods for lifelong MAPF. RHCR solves the windowed MAPF instances for fast computation. RHCR resolves collisions within a bounded time horizon while ignoring unnecessary long-term collisions. However, as shown in Fig. 3, this approach may cause traffic congestion in long-range corridors. Therefore, unlike the original RHCR [6], we applied an anytime approach with an adaptive time horizon to obtain improved solutions in a topological map. The anytime approach rapidly finds an approximate solution and then incrementally improves it while time is available. The proposed algorithm also quickly obtains an initial solution with a small time horizon and then iteratively refines a subset of the solution with an extended time horizon. This approach significantly improves the solution quality within a limited time, while providing local optimal solutions for congested or dead-end corridors.

Lifelong MAPF algorithms iteratively call a MAPF solver to solve a single MAPF instance. The proposed Corridor-CBS is a MAPF solver that plans collision-free paths for agents from their start to their goals. Corridor-CBS is based on the CBS algorithm [9] and has been extended to effectively resolve corridor conflicts in a topological map. A corridor conflict incurs a significant overhead cost for an agent to wait or bypass the corridor. Therefore, corridor conflicts are prioritized and resolved first. We also define a heuristic value based on corridor conflict relations and apply it to node selection in the best-first search of CBS. Existing methods for heuristic computation [14], [15] construct a multi-valued decision diagram (MDD) for cardinality computation; however, constructing MDDs for goal sequences in lifelong MAPF can result in significant computational overhead. In contrast, our method efficiently computes heuristics using corridor conflict information without constructing MDDs. These approaches greatly reduce the search space and number of node expansions while maintaining the solution quality. Therefore, Corridor-CBS proposed solver determines local optimal solutions faster than CBS.

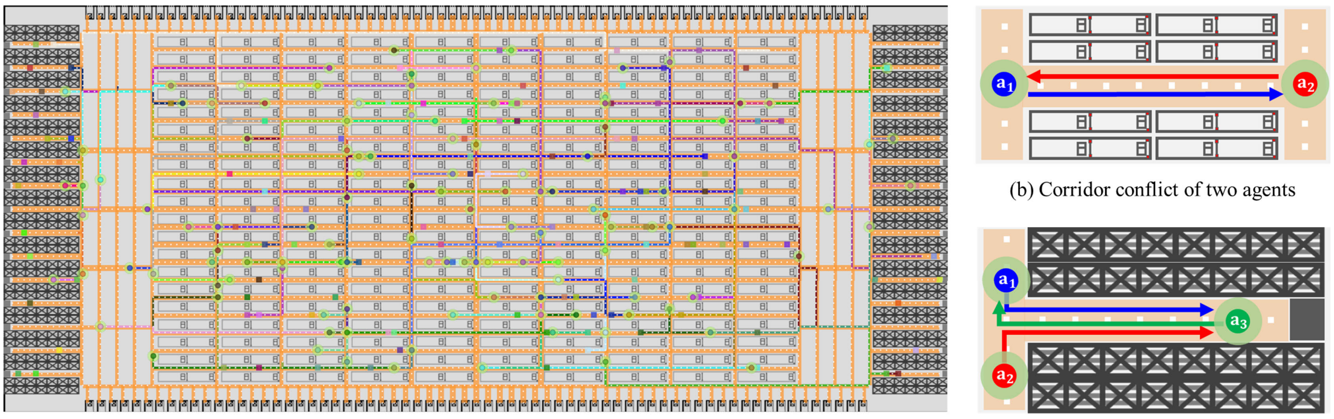
The following are the contributions of this study:

- We propose a novel method for lifelong MAPF in topological maps comprising only single-track corridors. The proposed method provides high-quality plans with high throughput while avoiding traffic congestion or deadlocks in corridors.
- We present an anytime RHCR algorithm that iteratively improves the solution quality using an adaptive time horizon. The original RHCR [6] only resolves collisions within a restricted time horizon using a suboptimal solver. However, the proposed RHCR can find a solution that avoids more distant collisions using an optimal solver.
- We present a MAPF solver called Corridor-CBS. Corridor-CBS significantly reduces the computation time of CBS in a topological map by applying several improvement steps related to corridor conflicts: corridor symmetry reasoning, prioritizing corridor conflict, and corridor heuristics.
- We evaluate the proposed algorithms through extensive experiments in warehouse and railway environments. The experiments show that the proposed algorithms outperform state-of-the-art methods [5], [6], [9], [11], particularly in a topological map with long-distance corridors and heavy traffic congestion.

II. RELATED WORK

This section discusses prior studies on lifelong MAPF (Section II-A) and one-shot MAPF (Section II-B) problems, which are summarized in Table 1. We also provide a literature review on MAPF studies applied to topological maps (Section II-C). The one-shot MAPF problem involves finding collision-free paths from the given start locations to the goal locations simultaneously. The lifelong MAPF problem is an online version of the one-shot MAPF problem, which solves a stream of single MAPF instances for continuously updated goals.

As described in Table 1, algorithms can be categorized according to *optimality* or *completeness* properties. Optimal algorithms always provide an optimal solution while their scalability for the number of agents is limited. In contrast, suboptimal algorithms focus on improving scalability by compromising solution quality. Completeness is the property that an algorithm eventually finds a feasible solution if one exists.



(a) Lifelong MAPF in the topological map

(b) Corridor conflict of two agents

(c) Conflict in a dead-end corridor

FIGURE 1. Automated warehousing scenario (Scenario 1). (a) Multiple agents automatically perform logistics operations on the topological map. Colored circles and rectangles represent agents and target locations, respectively. Each agent moves along white waypoints in orange lanes. This scenario is challenging because (b) corridor conflicts frequently occur when two agents simultaneously pass through a corridor from opposite directions. Furthermore, (c) conflicts in a dead-end corridor may lead to a deadlock situation.

TABLE 1. Summary of MAPF studies. Studies are classified into a "One-shot" MAPF solver and a "Lifelong" MAPF method based on the mid-dotted line. "Opt." and "Comp." represent the algorithm's optimality and completeness. In "Comp.", the triangle mark (△) means that the completeness is satisfied only in well-formed maps. "WF" is an algorithm that can only be used for well-formed maps. "Any." refers to an algorithm applied with the anytime strategy. "Topolo. Map" means an algorithm directly applied to topological maps.

Method	One-Shot/ Lifelong	Opt.	Comp.	WF	Any.	Topolo. Map
Push-and-Swap [16]	One-Shot	✗	✗	✗	✗	✗
MRRP [17]	One-shot	✗	✗	✗	✗	✓
CBS [9]	One-shot	✓	✓	✗	✗	✗
ICBS [18]	One-shot	✓	✓	✗	✗	✗
CBSH [14] [15]	One-shot	✓	✓	✗	✗	✗
CBSH-RTC [10] [19]	One-shot	✓	✓	✗	✗	✗
ECBS [12]	One-shot	✗	✓	✗	✗	✗
CCBS [20]	One-shot	✓	✓	✗	✗	✓
Anytime MAPF [21]	One-shot	✗	✓	✗	✓	✗
MAPF-LNS [22]	One-shot	✗	✓	✗	✓	✗
P-MAPF-LNS [23]	One-shot	✗	✓	✗	✓	✓
Token Passing [3]	Lifelong	✗	△	✓	✗	✗
RDP [5]	Lifelong	✗	△	✓	✗	✗
MLA* [4]	Lifelong	✗	△	✓	✗	✗
PRIMAL2 [24]	Lifelong	✗	✗	✗	✗	✗
RHCR [6]	Lifelong	✗	✗	✗	✗	✗
RHCR-RDP [11]	Lifelong	✗	✗	✓	✗	✗
exRHCR [25]	Lifelong	✗	✗	✗	✗	✗
Anytime-RHCR (Ours)	Lifelong	✗	✗	✗	✓	✓

A. LIFELONG MAPF

Lifelong methods must adopt an efficient suboptimal solver instead of an optimal solver for fast online computation. Therefore, as shown in Table 1, all lifelong methods do not guarantee the optimality of a solution. Most studies [3], [4], [5] have continuously replanned paths for a subset of agents whenever new goal locations are assigned. Ma et al. [3] proposed a token-passing method that continuously passes a token to an agent. The agent that receives the token has the authority to assign a task and plan a path. The method computes an agent's path by considering the paths of the other agents as dynamic obstacles. Liu et al. [5] planned paths only for the agents with new goals. They reserved dummy paths for isolated parking locations to avoid deadlocks. Grenouilleau et al. [4] proposed a multi-label A* algorithm for the low-level search of an MAPF solver. The algorithm extends the original A* to determine a path traversing two sequential goals.

These methods [3], [4], [5] generally provide low-quality solutions because they plan subpaths for several updated agents, rather than all paths. Furthermore, they are applicable only to well-formed maps [26], where each agent may rest at an isolated goal or parking location that is not blocked from other agents. Li et al. [6] proposed the RHCR method to address these issues. As discussed previously, RHCR continuously replans all paths at regular time steps by resolving collisions only within a bounded time horizon. RHCR can efficiently compute high-quality solutions for all agents because it disregards unnecessary long-term collisions. Furthermore, it can be used for both normal and well-formed maps. Madar et al. [25] also proposed an experienced RHCR (exRHCR) method that reuses a previous search effort to improve the efficiency of RHCR. Xu et al. [11] integrated RHCR with the idea of "reserving dummy paths" [5]. This method iteratively solves a windowed MAPF instance at regular time steps or when an agent's goal location is updated.

However, in topological maps, the performance of the RHCR-based methods [6], [11], [25] can be significantly degraded. Long-term collisions can affect future planning because bypasses are limited or farther away in topological maps. RHCR with a small bounded horizon may determine unnecessarily long detours or even lead to deadlock situations. Planning with a large bounded horizon may not be able to determine a solution within the specified time limit. To address this issue, we extend the RHCR based on an anytime strategy. The proposed method applies an adaptive bounded horizon to plan subpaths. It first obtains an initial solution with a small time horizon and then progressively improves a subset of the solution with an extended time horizon. This method can efficiently compute high-quality plans considering long-term collisions in topological maps.

B. MAPF SOLVER

Lifelong algorithms continuously call an MAPF solver to solve a single MAPF instance. The MAPF solver plans collision-free paths for agents from their starting locations to goal locations. Some solvers have applied a rule-based or prioritized method. The rule-based method [7], [16], [27] computes multi-agent paths based on simple movement rules such as push and swap. This guarantees the determination of a solution in polynomial time; however, the solution quality is relatively poor. The prioritized method [8], [17], [28] sequentially plans a single path for each agent according to a predetermined agent's priority. It does not guarantee the completeness and optimality of the solutions, but it is fast and scalable.

Sartoretti et al. [29] proposed a learning-based MAPF framework, namely PRIMAL. PRIMAL trains a single-agent policy on a partially observable map via reinforcement and imitation learning. They also proposed PRIMAL2 [24], an extension of PRIMAL, for application in lifelong MAPF. Their methods can quickly compute paths for a large number of agents; however, the paths do not guarantee completeness and optimality.

Sharon et al. [9] presented the CBS algorithm for obtaining complete and optimal solutions. CBS iteratively resolves a conflict between two agents by expanding a binary constraint tree until it determines a node without conflicts. Some studies have attempted to reduce the computation time of CBS while maintaining completeness and optimality by applying several techniques such as prioritizing conflicts [18], symmetry reasoning [10], and adding heuristics [14], [15]. We extended these techniques to be applicable to the lifelong MAPF problem in a topological map.

Some studies [12], [22], [30] have improved the scalability of an MAPF solver up to hundreds of agents while providing a near-optimal solution. Barer et al. [12] proposed a bounded-suboptimal variant of CBS called enhanced CBS (ECBS) to improve the scalability of CBS. The ECBS performs a focal search instead of a best-first search in the constraint tree of CBS to obtain a bounded suboptimal solution. Given a user-specified suboptimality factor α, the focal search guarantees that the solution cost is less than α times the optimal cost. ECBS can reduce the run time by increasing the suboptimality bound. Therefore, ECBS allows users to control the trade-off between scalability and solution quality through the suboptimality factor. Rahman et al. [30] further reduced the computation time of the ECBS by applying different suboptimal bounds to each agent.

Several studies [21], [22] have employed an anytime strategy for the MAPF problem. They quickly obtained an initial solution using a suboptimal solver and then iteratively improved a subset of the solution using an optimal or near-optimal solver. We also adopt the anytime strategy for path improvement. However, our method addresses a lifelong MAPF problem, unlike existing anytime methods [21], [22] that solve a one-shot MAPF problem. Our method iteratively improved a subset of the solution by resolving additional collisions within an extended time horizon in lifelong MAPF.

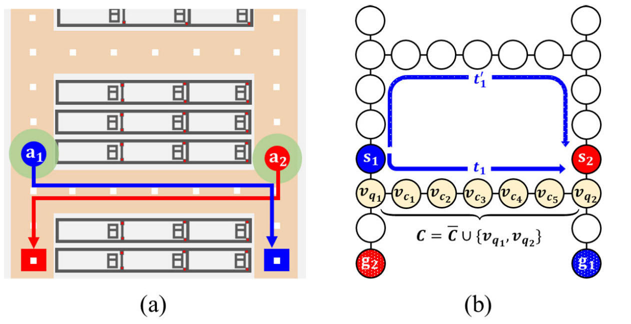
C. MAPF ON TOPOLOGICAL MAPS

Although many studies have solved the MAPF problem on grid maps, only a few studies have considered topological maps. Several studies have attempted to effectively solve the MAPF problem on grid maps using topological graphs [31] or corridor information [10], [32]. Liu et al. [31] constructed a sector-based topological graph and estimated traffic conditions for each sector. They planned multi-agent paths by considering the traffic conditions to avoid traffic congestion. Cohen et al. [32] extended CBS by considering additional heuristic information on highways and guiding agents to avoid collisions in corridors. Li et al. [10] also presented a corridor conflict resolution method for CBS, significantly reducing unnecessary expansions of constraint trees.

Several studies [17], [23] have directly solved the MAPF problem on topological maps. Binder et al. [17] defined four representative collision cases on topological maps (sequential, wait, avoid, and push) and presented a resolution method for each case. Li et al. [23] also considered the MAPF on a topological graph for the railway scheduling of a large number of trains. They solved four MAPF instances in parallel using an anytime strategy called large neighborhood search [22] to plan paths for thousands of agents efficiently. Andreychuk et al. [20] and Kasaura et al. [33] solved a continuous-time MAPF problem on a topological map. They used a variant of safe interval path planning (SIPP) [34] to handle continuous time conflicts on a graph. Although these methods focused on the MAPF in topological maps, currently, no appropriate algorithm exists for a lifelong MAPF problem. As mentioned previously, lifelong MAPF in topological maps is challenging because corridor conflicts and traffic congestion occur frequently. Therefore, this study presents a new method to effectively resolve corridor conflicts for the lifelong problem.

III. PROBLEM FORMULATION

The problem considered in this study is the lifelong MAPF in a topological map. The topological map is defined as an undirected graph $\mathcal{G}_{\mathcal{M}} = (V_{\mathcal{M}}, E_{\mathcal{M}})$, where each vertex $v \in V_{\mathcal{M}}$ represents a location in 2D, and each edge $e \in E_{\mathcal{M}}$ represents a lane connecting two vertices. As shown in Fig. 1, agents generally move in a warehouse environment



a_1
a_2
(a)
t_1'
t_1
s_1
s_2
v_{q_1} v_{c_1} v_{c_2} v_{c_3} v_{c_4} v_{c_5} v_{q_2}
$C = \bar{C} \cup \{v_{q_1}, v_{q_2}\}$
g_2
g_1
(b)

FIGURE 2. Illustration of the pathfinding of two agents, a_1 and a_2, with a corridor conflict. (a) Corridor conflict occurs when a_1 and a_2 cross corridor C in opposite directions. (b) The pathfinding problem is represented in $\mathcal{G_M}$. The corridor C is composed of a set of connected vertices with degree 2, $\bar{C} = \{v_{c_1}, v_{c_2}, v_{c_3}, v_{c_4}, v_{c_5}\}$, and two endpoints $\{v_{q_1}, v_{q_2}\}$. If a_2 moves though corridor $\bar{C}$ first, then a_1 must either wait to pass through C or use a bypass without traversing C. t_1 and t_1' are the earliest time steps when a_1 arrives the endpoint v_{q_2} through C or a bypass, respectively.

consisting of only narrow corridors. The vertices within a corridor represent waypoints where agents follow or wait. At each discrete time step t, an agent either moves to a connected vertex or remains at its current vertex. All the agents are assumed to precisely follow the planned paths on the vertices.

An MAPF instance includes a topological map $\mathcal{G_M}$ and a set of agents $\mathcal{A} = \{a_1, \ldots, a_N\}$ where each agent $a_i \in \mathcal{A}$ has a unique start $s_i \in V_\mathcal{M}$ and a unique goal $g_i \in V_\mathcal{M}$. A path for an agent a_i is defined as a sequence of vertices $p_i = \{v_i^{(0)}, \ldots, v_i^{(t_i)}\}$, starting at $s_i = v_i^{(0)}$ and ending at $g_i = v_i^{(t_i)}$, where t_i is the time step required to reach the goal. The cost or distance of a path p_i is defined as its travel time, equal to the time step t_i. A collision (or, synonymously, conflict) between two paths, p_i and p_j, occurs when a_i and a_j move to the same vertex at the same time step (called a vertex collision) or simultaneously cross the same edge in opposite directions (called an edge collision). The objective of a one-shot MAPF is to find collision-free paths $P = \{p_1, \ldots, p_N\}$ with minimum total cost, $cost(P) = \sum_{p_i \in P} t_i$, for all agents in $\mathcal{A}$.

The lifelong MAPF problem is an online version of the one-shot MAPF problem, continuously assigning a new task to an agent when the agent finishes its current task. Given a set of agents $\mathcal{A} = \{a_1, \ldots, a_N\}$ and a set of tasks $\mathcal{T} = \{\tau_1, \ldots, \tau_K\}$, the objective is for agents in $\mathcal{A}$ to complete all tasks in $\mathcal{T}$ by maximizing the *throughput* (average number of completed tasks per time step). *Cycle time* (average completion time per task) has a reverse relationship with throughput, so the objective is equivalent to minimizing the cycle time. Most studies [3], [4], [5] define a task $\tau_k \in \mathcal{T}$ as having two vertices, a pickup location, and delivery location. However, this study considered only one goal location g_k for a task τ_k. A task allocator iteratively assigns a task to an idle agent; however, this study does not consider the task allocation method. An agent sequentially visits its goal locations while avoiding vertex and edge collisions [9] with other agents.

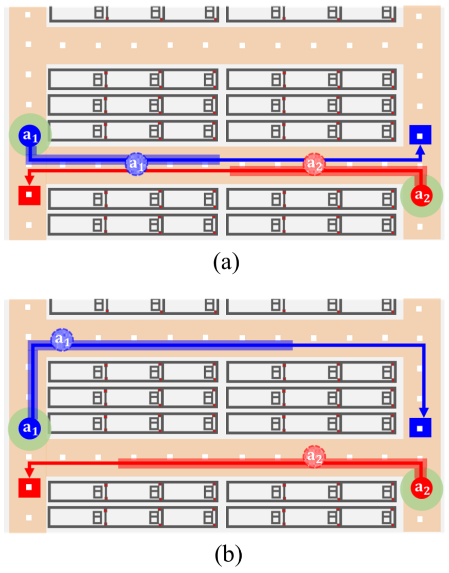


FIGURE 3. Examples of path planning of RHCR for different bounded time horizons. RHCR plans paths that resolve conflicts only up to a bounded time horizon ω (bold lines) and then re-plans paths from new starting points (dashed circles) after $\lambda = 4$ time steps. (a) If $\omega = 6$, RHCR cannot detect the corridor conflict, so it plans paths for two agents, a_1 and a_2, to enter the same corridor simultaneously. (b) If $\omega = 9$, RHCR plans paths, including a detour that avoids the corridor conflict.

We define a corridor conflict as a conflict that occurs when two agents simultaneously pass through a corridor from opposite directions. To resolve this conflict, an agent must wait until the other agent has completely passed the corridor; this significantly complicates the MAPF problem. Therefore, an effective detection and resolution of corridor conflicts is important. As shown in Fig. 2b, a corridor C comprises a set of connected vertices $\bar{C} = \{v_{c_1}, \ldots, v_{c_L}\} \subset V_\mathcal{M}$ with degree 2 and two endpoint vertices $\{v_{q_1}, v_{q_2}\} \in V_\mathcal{M}$ [10]. The corridor length $len(C)$ is defined as the distance between v_{q_1} and v_{q_2}, which is equal to the time steps it takes to move between the two vertices. It can be computed as $len(C) = |\bar{C}| + 1$ where $|\bar{C}|$ is the number of vertices in $\bar{C}$. This corridor information is stored in advance and can be directly accessed to detect corridor conflicts when planning paths.

IV. ANYTIME LIFELONG MAPF

To solve the lifelong MAPF problem, we employed the RHCR approach [6], one of the most efficient methods. RHCR solves a series of windowed MAPF instances instead of all MAPF instances [3], [4], [5]. Because multi-agent paths are constantly updated according to newly assigned tasks, RHCR disregards distant future plans. It iteratively plans multi-agent paths at regular λ time steps by resolving collisions only within a bounded time horizon of ω time steps. Furthermore, RHCR computes a path that visits a sequence of goals instead of a single goal. If the distance

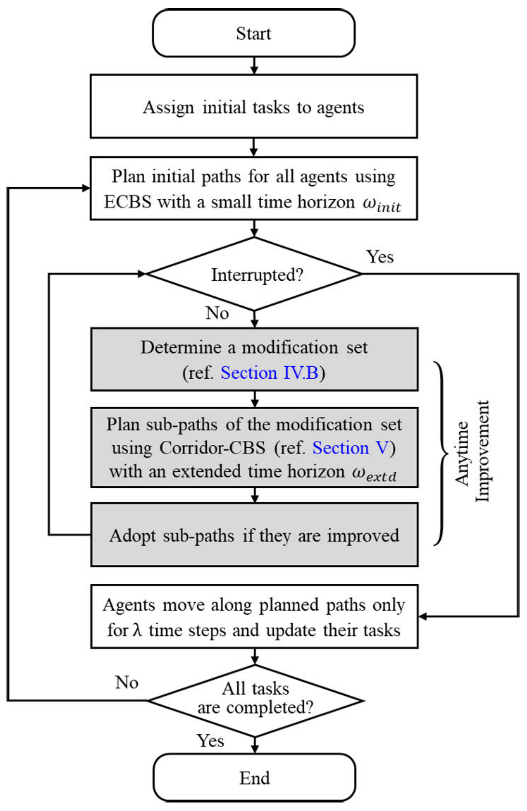

Start
Assign initial tasks to agents
Plan initial paths for all agents using ECBS with a small time horizon ω_{init}
Interrupted?
Yes
No
Determine a modification set (ref. Section IV.B)
Plan sub-paths of the modification set using Corridor-CBS (ref. Section V) with an extended time horizon ω_{extd}
Adopt sub-paths if they are improved
Anytime Improvement
Agents move along planned paths only for λ time steps and update their tasks
No
All tasks are completed?
Yes
End

FIGURE 4. Flowchart of Anytime-RHCR algorithm for lifelong MAPF. Gray boxes represent the process of anytime improvement step.

Algorithm 1 Anytime-RHCR Algorithm

Input: Agent set $\mathcal{A} = \{a_1, \ldots, a_N\}$, Task set $\mathcal{T} = \{\tau_1, \ldots, \tau_K\}$, Initial window ω_{init}, Extended window ω_{extd}, Replanning window λ

Output: None

```
1:  $\mathcal{T}' \leftarrow$InitTasks($\mathcal{T}, \mathcal{A}$)
2:  while $\mathcal{T}' \neq \emptyset$ do
        /* Initial Path Planning */
3:      $S \leftarrow$ GetStartLoc($\mathcal{A}$)
4:      $G \leftarrow$ GetGoalLoc($\mathcal{T}'$)
5:      $P \leftarrow$ SolveMAPF($S, G, w_{init}$)
        /* Path Improvement Step */
6:      while NotInterrupted() do
7:          $CF \leftarrow$ SearchConflicts($P, w_{extd}$)
8:          $\mathcal{A}_M \leftarrow$ GetModificationSet($\mathcal{A}, P, CF$)
9:          $[S_M, G_M, P_M] \leftarrow$ ActiveSet($\mathcal{A}, \mathcal{A}_M, S, G, P$)
10:         $P_{\bar{M}} \leftarrow P/P_M$
11:         $P^*_M \leftarrow$SolveMAPFSub($S_M, G_M, P_{\bar{M}}, w_{extd}$)
12:         if $CF \neq \emptyset$ or cost($P^*_M$) < cost($P_M$) then
13:             $P \leftarrow P^*_M \cup P_{\bar{M}}$
14:         end if
15:     end while
16:     $\mathcal{A} \leftarrow$ MoveAgents($\mathcal{A}, P, \lambda$)
17:     $\mathcal{T}' \leftarrow$ UpdateTasks($\mathcal{T}, \mathcal{T}', \mathcal{A}$)
18: end while
```

of the shortest path visiting all goals is less than ω, a goal of a new task is appended to the sequence. This approach reduces the computation time compared with methods that consider the entire time horizon while still producing high-quality solutions; therefore, RHCR can efficiently process the lifelong MAPF for a large number of agents. Furthermore, unlike existing methods [3], [4], [5] that can only be used in well-formed maps, RHCR can be used in all types of maps.

However, in topological maps, the performance of the RHCR is significantly affected by bounded time horizon ω and the type of MAPF solver. Topological maps consist of only corridors, causing many corridor conflicts and traffic congestions. RHCR with a small value of ω may cause a deadlock in a long-range corridor. Furthermore, to resolve a corridor conflict, an agent must wait until the other agent moving in the opposite direction has completely passed. This waiting time may be long and can accumulate unexpectedly owing to low-quality plans. As shown in Fig. 3a, RHCR sometimes cannot detect corridor conflict when it uses a bounded time horizon shorter than the corridor length. Eventually, the RHCR plans paths for two agents to enter a corridor from different directions simultaneously. The agent must then return the way it came in. If this occurs several times, traffic congestion is triggered, in which several agents are stuck in a corridor. To solve this problem, RHCR should use a larger time horizon, as shown in Fig. 3b. However, significant computation is required to plan all paths with a large time horizon. RHCR may not even find a solution within a limited period. Therefore, the bounded time horizon should be appropriately determined according to traffic congestion to obtain the best solutions for RHCR.

Therefore, we employed an anytime strategy to apply an adaptive time horizon to RHCR. We refer to the proposed lifelong MAPF algorithm as *Anytime-RHCR*. Fig. 4 provides a high-level overview of the Anytime-RHCR algorithm. In Fig. 4, the gray boxes represent the process of the proposed anytime step and the rest are the same as those of the original RHCR. The algorithm iteratively plans paths after agents move λ time steps and assigns a new task whenever an agent completes its current task. For path planning, Anytime-RHCR first obtains an initial solution using a fast suboptimal solver, ECBS, with a short time horizon ω_{init}. It then determines a subset of agents called the modification set. Anytime-RHCR iteratively improves the solution of the modification set using the optimal solver Corridor-CBS with an extended time horizon ω_{extd}. Therefore, the proposed method efficiently resolves conflicts over a longer time horizon within a limited computation time. The proposed method also effectively improves the solution quality from the optimal solver. This approach enables to determine high-quality plans for traversing complex corridors without deadlock. The following sections detail the overall process of Anytime-RHCR (Section IV-A), the modification set (Section IV-B), and windowed MAPF solvers for initial planning and path improvement (Section IV-C). Furthermore, in Section V, we detail the Corridor-CBS algorithm, which is a MAPF solver in the improvement step.

A. ANYTIME-RHCR

Algorithm 1 depicts the pseudocode of the Anytime-RHCR algorithm. The algorithm consistently assigned an incomplete task to an agent that had completed its current task. It also iteratively plans paths for all agents to execute assigned tasks at λ regular time steps. The algorithm quickly finds initial paths with a small window ω_{init} and then iteratively refines the subpaths of a modification set with an extended window ω_{extd}. The modification set is determined to be an agent related to additional conflicts in the ω_{extd}.

The algorithm initializes task subset $\mathcal{T}'$ from the set of all tasks $\mathcal{T}$ (Line 1). Each task set $\mathcal{T}_i' \subset \mathcal{T}'$ represents a set of sequential tasks assigned to an agent $a_i \in \mathcal{A}$. Each task set $\mathcal{T}_i'$ is continuously updated by appending a new task $\tau_k \in \mathcal{T}$ whenever a_i completes a task. (Line 17). The updated new task, τ_k, is removed from $\mathcal{T}$. The algorithm is terminated when all tasks in $\mathcal{T}$ are assigned and completed (Line 2).

The algorithm iteratively replans multi-agent paths at regular λ time steps. For each iteration, it sets start locations S to the current locations of agents $\mathcal{A}$ (Line 3) and goal locations G to the target locations of assigned tasks $\mathcal{T}'$ (Line 4). The algorithm applies the anytime MAPF approach to determine multi-agent paths where each agent $a_i \in \mathcal{A}$ moves from start location $s_i \in S$ to sequential goal locations $G_i \subset G$ without collision. The anytime approach first computes initial paths P for all agents using a suboptimal solver with a small time horizon ω_{init} (Line 5). It then gradually improves the quality of the subset of P using a near-optimal solver with adaptive ω_{extd} until interrupted; we refer to this stage as the path improvement step (Lines 6 – 15). We interrupt this step when the timeout expires or a limited iteration number is reached. Finally, all agents move along computed paths P only for λ time steps (Line 16) and update their tasks $\mathcal{T}'$ (Line 17).

In the improvement step, the algorithm iteratively determines a modification set of agents $\mathcal{A}_M \subset \mathcal{A}$ and replans their paths $P_M \subset P$ while fixing the other paths $P_{\bar{M}} = P/P_M$. It first finds conflicts CF among planned paths P that occur in extended time window ω_{extd} where ω_{extd} is larger than ω_{init} (line 7). It then selects N_M agents as a modification set $\mathcal{A}_M$ based on the conflicts (Line 8 and Section IV-B). Next, the algorithm extracts their start locations $S_M \subset S$, goal locations $G_M \subset G$, and paths $P_M \subset P$ (Line 9). It computes new paths P_M^* that do not collide with $P_{\bar{M}}$ using a near-optimal solver with extended window ω_{extd} (Line 11). Finally, the algorithm adopts new paths P_M^* (line 13) if the cost of P_M^* is lower than that of original paths P_M (Line 12). If conflict set CF is not empty, the MAPF solver resolves one or more conflicts in the CF; therefore, it directly adopts P_M^* irrespective of the cost. The algorithm repeats this procedure until interrupted.

B. SELECTION OF THE MODIFICATION SET

In Algorithm 1, determining agent set $\mathcal{A}_M$ is a critical component affecting the quality of a solution. The proposed method first attempts to solve conflicts in CF by selecting a set of agents related to the conflicts. The method then determines a set of agents to reduce the solution cost if all the conflicts are resolved. A different method is applied for selecting a modification set depending on the existence of conflicts in CF.

If conflicts exist, we select a set of agents that effectively resolve many conflicts in CF. Let $\mathcal{G}_{CF} = (V_{CF}, E_{CF})$ be the conflict graph where each vertex $v \in V_{CF}$ is an agent and each edge $e \in E_{CF}$ represents a conflicting relationship between the two agents [35]. The proposed method determines the largest connected component $V_{CF}' \subset V_{CF}$ in $\mathcal{G}_{CF}$. If $|V_{CF}'| > N_M$, the method determines an agent set $\mathcal{A}_M$ by randomly selecting the connected agents in V_{CF}'. If $|V_{CF}'| < N_M$, the method initializes $\mathcal{A}_M$ as the set of all agents in V_{CF}'. It then selects a random agent a_i in $\mathcal{A}_M$ and adds other agents that block path p_i of a_i. This process is repeated until $|\mathcal{A}_M| = N_M$.

If there are no conflicts, we apply an agent-based selection method [22]. This method determines an agent a_i with the largest delay and inserts it into $\mathcal{A}_M$. A delay is defined as the cost difference between the shortest and planned paths. The proposed method determines a set of agents that block the path of a_i and inserts them into $\mathcal{A}_M$. It then repeatedly adds agents that block the path of random agent $a_j \in \mathcal{A}_M$ until $|\mathcal{A}_M| = N_M$. The proposed method selects the new agent with the largest delay for each iterative step by maintaining the list of selected agents.

C. WINDOWED MAPF SOLVERS

This section describes the windowed MAPF solver for computing multi-agent paths P with bounded horizon collision checks (Lines 5 and 11 in Algorithm 1). Unlike the original solvers [9], [12], the windowed solver checks and resolves collisions only up to the first ω time steps on P and ignores the collisions of the remaining time steps. Therefore, after ω time steps, the solver determines the shortest paths to the agent's goals, irrespective of collisions. We employed the multi-label A* algorithm [4], [6] with the SIPP method [34] for the low-level search of the solver. This algorithm computes a single path visiting multiple goals of an agent in the location-time space.

For the initial planning (Line 5 in Algorithm 1), we employed the complete and bounded suboptimal solver, ECBS [12]. The ECBS can quickly determine a bounded suboptimal solution with a high suboptimality factor. For improvement planning (Line 12), we use a solver, a variant of CBS, namely *Corridor-CBS*. Corridor-CBS applies several corridor conflict resolution techniques to improve the computational efficiency of CBS in a topological map. Section V details the corridor conflict resolution of the Corridor-CBS.

Algorithm 2 High-Level Search of Corridor-CBS

```
Input: Start locations $S = \{s_1, \dots, s_N\}$, Goal locations $G = \{G_1, \dots, G_N\}$, Initial constraints $CT_{init}$
Output: Planned paths $P$, or "No solution"
1:  $\mathcal{R} \leftarrow$ GenerateRootNode($S, G, CT_{init}$)
2:  $OPEN$.Push($\mathcal{R}$)
3:  while $OPEN \neq \emptyset$ do
4:      $\mathcal{N} \leftarrow OPEN$.Pop()        // Node with lowest $f = c + h$
5:      if $\mathcal{N}.CF = \emptyset$ then
6:          return $\mathcal{N}.P$              // Solution found
7:      end if
8:      if NotComputed($\mathcal{N}.h$) then
            /* Corridor Heuristic Computation (Section V-C) */
9:          ClassifyCorridorConflict($\mathcal{N}.CF$)
10:         $\mathcal{N}.h \leftarrow$ ComputeCorridorHeuristic($\mathcal{N}$)
11:         $OPEN$.Push($\mathcal{N}$)
12:         continue
13:     end if
        /* Prioritizing Corridor Conflict (Section V-B) */
14:     $cf \leftarrow$ SelectConflict($\mathcal{N}.CF$)
15:     if IsCorridorConflict($cf$) then
            /* Corridor Symmetry Reasoning (Section V-A) */
16:         $[ct_1, ct_2] \leftarrow$ ResolveCorridorConfict($cf$)
17:     else
18:         $[ct_1, ct_2] \leftarrow$ ResolveNormalConfict($cf$)
19:     end if
20:     $\mathcal{N}_1 \leftarrow$ GenerateChild($\mathcal{N}, ct_1$)
21:     $\mathcal{N}_2 \leftarrow$ GenerateChild($\mathcal{N}, ct_2$)
22:     $OPEN$.Push($\mathcal{N}_1$)
23:     $OPEN$.Push($\mathcal{N}_2$)
24: end while
25: return "No solution"
```

V. CORRIDOR CONFLICT-BASED SEARCH

In this section, we describe a new MAPF solver called Corridor-CBS. The Corridor-CBS is used as the MAPF solver in the improvement step of Anytime-RHCR (Line 11 in Algorithm 1). Corridor-CBS solves the one-shot MAPF problem of generating collision-free paths from the starts to the goals. It only resolves conflicts within an extended time horizon ω_{extd}, as described in Section IV-C.

Algorithm 2 shows the pseudocode for the Corridor-CBS algorithm. Corridor-CBS extends the CBS algorithm [9] to resolve corridor conflicts in a topological map. In Algorithm 2, the parts added to the original CBS are highlighted in blue. CBS comprises a two-level search algorithm. High-level search resolves all conflicts on planned paths by expanding a binary constraint tree. Each node in the constraint tree includes a set of constraints and the shortest paths that satisfy its constraints. CBS iteratively assigns a new constraint to a node by expanding the constraint tree until it finds collision-free paths. The low-level search algorithm is a single-agent path planner that computes the shortest path for each agent that satisfies the constraints of the tree node. As mentioned before, we adopted the multi-label A* [4], [6] as a low-level search algorithm to find a path visiting multiple goals. We also used the SIPP [34] instead of a space-time A*. SIPP performs A* search in a safe time interval space, not a discrete time step, which greatly reduces the search space in single-agent path planning.

Algorithm 2 first generates a root node $\mathcal{R}$ by computing the shortest path with an initial constraint for each agent (Line 1) and inserts $\mathcal{R}$ into an *OPEN* list (Line 2). The algorithm then iteratively resolves a conflict between two agents, a_1 and a_2, by expanding the nodes of the binary tree until it detects a node with collision-free paths (Lines 3 – 24). Each node $\mathcal{N}$ contains a set of constraints $\mathcal{N}.CS$, a solution path satisfying the constraints $\mathcal{N}.P$, and conflicts that occur in the solution $\mathcal{N}.CF$. A conflict between two agents a_i and a_j is defined as a tuple $cf = \langle a_i, a_j, v, t\rangle$ or $cf = \langle a_i, a_j, e, t\rangle$ where a_i and a_j conflict on vertex $v \in V_{\mathcal{M}}$ or edge $e \in E_{\mathcal{M}}$ at time step t. A constraint is also defined as a tuple $ct = \langle a, v, t\rangle$ or $ct = \langle a, e, t\rangle$ where it prevents agent $a \in \mathcal{A}$ from moving vertex $v \in V_{\mathcal{M}}$ or edge $e \in E_{\mathcal{M}}$ at time step t. For each iteration, the algorithm pops a node $\mathcal{N}$ in the *OPEN* list based on the best first search strategy (Line 4). It selects a node with the minimum f-value that is the sum of a solution cost c and an admissible heuristic h [15]. If h has not been calculated, it is initialized to zero. The algorithm then selects a conflict cf in $\mathcal{N}.CF$ (Line 14) and resolves it by assigning different constraints ct_1 and ct_2 to each agent a_1 and a_2 (Lines 16 or 18), respectively. Let $cf = \langle a_1, a_2, v, t\rangle$ be a normal conflict (not a corridor conflict). The constraints for resolving cf are defined as: $ct_1 = \langle a_1, v, t\rangle$ and $ct_2 = \langle a_2, v, t\rangle$ where ct_1 and ct_2 prohibit agent a_1 and a_2 from moving vertex v at time step t, respectively. Finally, the algorithm generates two child nodes, $\mathcal{N}_1$ and $\mathcal{N}_2$, by appending each constraint, ct_1 and ct_2, and inserts $\mathcal{N}_1$ and $\mathcal{N}_2$ into the *OPEN* list (Lines 20 – 23). Please refer to [9] for more information on CBS.

The aforementioned algorithm operation is similar to that of the original CBS [9]. The CBS unnecessarily searches for many nodes to resolve a corridor conflict, exponentially increasing node expansion according to the corridor length [10]. Moreover, corridor conflicts frequently occur in topological maps because the movement of the agent is restricted to single-track corridors. Therefore, Corridor-CBS additionally considers several improvement steps that relate to corridor conflict resolution, including i) *corridor symmetry reasoning* (Line 16), ii) *prioritizing corridor conflict* (Line 14), and iii) *corridor heuristic* (Line 10). The following subsections describe each improvement step.

A. CORRIDOR SYMMETRY REASONING

A corridor conflict produces many unnecessary expansions of the constraint tree on the CBS, significantly increasing the search space [10]. To address this problem, we employed the corridor symmetry reasoning method [10]. This method appends a specialized range constraint for a corridor conflict instead of a simple vertex or an edge constraint. Li et al. [10]

introduced other reasoning methods for rectangular and target symmetries. However, we only considered the corridor symmetry because our problem did not involve rectangular and target symmetries.

Assume that a corridor conflict occurs where the shortest paths of agents a_1 and a_2 intersect in opposite directions inside corridor C. To resolve this conflict, an agent must either wait for another agent to exit the corridor or take another detour. Therefore, CBS must assign many constraints that prevent an agent from entering all nodes in C until the other agent exits, which increases the number of node expansions up to $2^{len(C)}$ [10]. To address this problem, the corridor symmetry resolution method [10] assigns a constraint to the entrance of C instead of the node where conflict occurs. Let t_1 and t_2 be the earliest time steps when a_1 and a_2 arrive at endpoints v_{q_2} and v_{q_1} of C, respectively (Fig. 2b). This corridor conflict can be effectively resolved by appending the following two constraints to the child nodes [10].

$$ct_1 = \langle a_1, v_{q_2}, [0, \min(t'_1 - 1, t_2 + len(C))]\rangle$$
$$ct_2 = \langle a_2, v_{q_1}, [0, \min(t'_2 - 1, t_1 + len(C))]\rangle$$

where a range constraint $\langle a, v, [t_{min}, t_{max}]\rangle$ prohibits agent $a \in \mathcal{A}$ from entering vertex $v \in V_{\mathcal{M}}$ from t_{min} to t_{max} time steps. t'_1 is the earliest time step when a_1 arrives at v_{q_2} through a bypass without traversing corridor C (see Fig. 2b). If a bypass does not exist, t'_1 is set to $+\infty$.

B. PRIORITIZING CORRIDOR CONFLICT

For each iteration, CBS selects an arbitrary conflict cf from a conflict set $\mathcal{N}.CF$. However, the CBS performance is significantly affected by the selection order. The selection of an inappropriate conflict causes unnecessary expansion of the constraint tree and increases the runtime. Boyarski et al. [18] proposed a method for prioritizing conflicts. They resolved conflicts in the order of cardinal, semi-cardinal, and non-cardinal conflicts. The cardinal or semi-cardinal conflict increases the cost of two child nodes or only one child node, respectively. On the other hand, the non-cardinal conflict does not increase the cost of either child node. This method speeds up high-level search by reducing the number of node expansions.

In prioritizing conflict [18], cardinality computation requires building an MDD [36] for each agent. An MDD includes all possible shortest paths of an agent. The cardinality of a conflict $\langle a_i, a_j, v, t\rangle$ can be determined by checking that all shortest paths of a_i and a_j pass through the vertex v at the time step t in the MDDs. Generating an MDD for multiple goals in our lifelong MAPF could result in a large computational overhead. Therefore, rather than calculating cardinality, we prioritized and resolved corridor conflicts first. To resolve a corridor conflict, an agent must wait until the other agent has completely passed the corridor. This waiting time significantly affects the cost of the planned paths. Therefore, a corridor conflict is more important than a pure cardinal conflict in topological maps. This prioritization improves the lower bound of the conflict tree and reduces the possibility of node expansion. Finally, it reduces the runtime of high-level search without the computation of the cardinality.

C. CORRIDOR HEURISTIC

In CBS, the best-first search in the binary constraint tree is performed, where the node with the lowest solution cost c (sum of path costs) is expanded first. Recently, several studies [14], [15] have considered an admissible heuristic h for cost evaluation that further improved the high-level search performance. Similar to the A* search algorithm, they explored the tree nodes according to the f value, where $f = c+h$. An admissible heuristic can be estimated by computing the minimum vertex cover of a conflict graph [15] or dependency graph [14]. The conflict and dependency graphs represent the cardinal conflict and dependency relationships, respectively, for all the agents. This heuristic value is the minimum cost that must be increased to resolve the conflicts in the current solution.

In this study, we computed an admissible heuristic based on corridor conflict relations between agents. A corridor conflict incurs an overhead cost for an agent to wait or bypass the corridor. Given a node $\mathcal{N}$ with a corridor conflict between two agents a_i and a_j, overhead cost Δc_{ij} is defined as [14]:

$$\Delta c_{ij} = \bar{c}_{ij} - c_{ij}$$

where $\bar{c}_{ij}$ is the minimum cost of the conflict-free paths of a_i and a_j in $\mathcal{N}$ and c_{ij} is their current cost. The overhead cost can be used for an admissible heuristic because a corridor conflict will increase the cost by at least Δc_{ij}. This method is more efficient than determining cardinal conflicts and dependency relationships because it does not require the construction of MDDs with multiple goals.

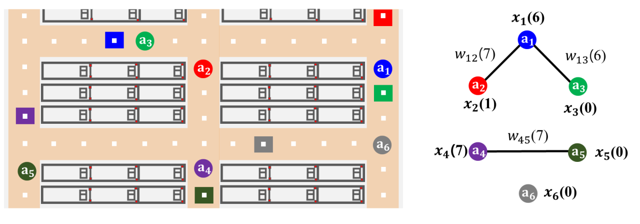
To compute the heuristic value of a node, we first construct corridor conflict graph $\mathcal{G}_{CCF} = (V_{CCF}, E_{CCF}, W_{CCF})$, representing the corridor conflict relations between every pair of agents. Similar to conflict graph G_{CF} (Section IV-B), each vertex $v_i \in V_{CCF}$ represents an agent. Each edge $e_{ij} \in E_{CCF}$ represents a corridor conflict relation between the two agents a_i and a_j, instead of a normal conflict. Each edge e_{ij} also has weight $w_{ij} \in W_{CCF}$, where w_{ij} is defined as overhead cost Δc_{ij}. We then solve an edge-weighted minimum vertex cover problem [14] for graph G_{CCF}. This involves assigning a non-negative integer value x_i to each vertex $v_i \in V_{CCF}$ to minimize $\sum_{v_i \in V_{CCF}} x_i$. For each edge $e_{ij} \in E_{CCF}$, the integer values should satisfy constraint of $x_i + x_j \geq w_{ij}$. Finally, the heuristic value h is determined as the sum of the integer values of the vertex cover solution. Fig. 5 shows an example of an admissible heuristic computation using the proposed method.

VI. EXPERIMENTAL RESULTS

We conducted four sets of experiments to evaluate the performance of the proposed method. First, we evaluated

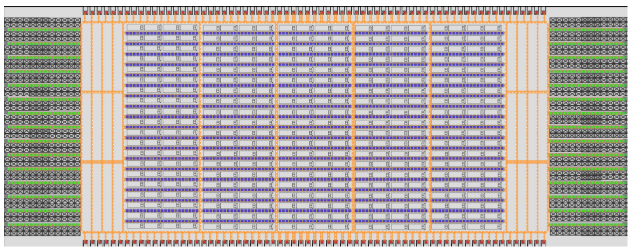
TABLE 2. **Average throughput of each method for different numbers agent N. The best and second-best throughputs are highlighted in bold font and underlined, respectively. For each scenario, we randomly generated tasks and determined a task sequence be assigned to each agent. We simulated 5000 time steps using the same task sequences for each method and counted the number of completed tasks to calculate throughput.**

Methods	Scenario 1					Scenario 2					Scenario 3				
	N:40	N:60	N:80	N:100	N:120	N:40	N:60	N:80	N:100	N:120	N:40	N:60	N:80	N:100	N:120
RDP	0.581	0.867	1.151	1.429	1.706	0.502	0.745	0.974	1.212	1.407	0.471	0.691	0.892	1.100	1.235
RHCR$_1$ (ω: 30)	0.581	0.864	1.144	1.429	1.708	0.504	0.749	0.979	1.225	1.413	0.466	0.695	0.883	1.082	1.223
RHCR$_2$ (ω: 40)	0.581	0.865	1.148	1.436	1.711	0.505	0.753	0.985	1.233	1.425	0.472	0.704	0.900	1.099	1.265
RHCR$_3$ (ω: 50)	0.583	0.869	1.151	1.438	1.719	0.504	0.754	0.988	1.239	1.430	0.476	0.706	0.904	1.121	1.283
RHCR-RDP$_1$ (ω: 40)	0.582	0.866	1.151	1.437	1.718	0.505	0.752	0.988	1.234	1.433	0.472	0.702	0.898	1.108	1.272
RHCR-RDP$_2$ (ω: 50)	0.583	0.868	1.153	1.441	1.722	0.505	0.755	0.991	1.240	1.436	0.476	0.705	0.909	1.120	1.288
Anytime-RHCR$_1$ (ω_{init}: 30, ω_{extd}: 50)	**0.588**	**0.878**	**1.160**	1.440	1.725	**0.514**	**0.766**	0.995	1.240	1.438	**0.488**	0.711	0.903	1.109	1.260
Anytime-RHCR$_2$ (ω_{init}: 40, ω_{extd}: 50)	0.587	0.874	1.157	**1.448**	**1.733**	0.514	0.764	**0.998**	**1.249**	**1.445**	0.486	**0.717**	**0.917**	**1.126**	**1.299**

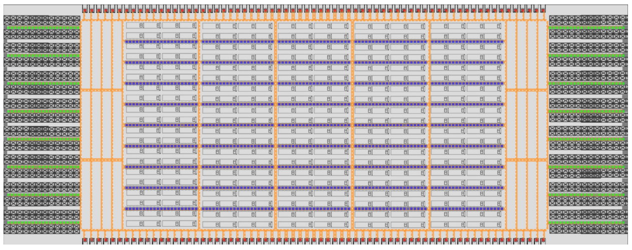


(a) (b)

FIGURE 5. **Example of an admissible heuristic computation based on corridor conflicts. Let $\mathcal{N}$ be a node with the shortest paths without considering collision for (a) the MAPF instance. Our method detects corridor conflicts in $\mathcal{N}$ and constructs (b) corridor conflict graph GCCF. In GCCF, the weight w_{ij} of an edge e_{ij} is defined as the overhead cost c_{ij}. The method then assigns a non-negative integer value x_i to each vertex v_i by solving the edge-weighted minimum vertex cover problem for GCCF. Finally, the admissible heuristic of $\mathcal{N}$ is determined as $\sum_{v_i \in V_{CCF}} x_i$: $h = 6 + 1 + 0 + 7 + 0 + 0 = 14$.**



(a) Scenario 2



(b) Scenario 3

FIGURE 6. **Simulated warehouse environments used in the experiments. The topological maps are represented with orange lanes and white nodes. The red, blue, and green nodes denote the parking station, rack station, and in/outbound station, respectively.**

the performance of the anytime lifelong MAPF algorithm compared with that of the baseline algorithms [5], [6], [11] (Section VI-A). Second, we analyzed the influence of the parameter, modification set size, in Anytime-RHCR (Section VI-B). Third, we evaluated the effectiveness of the Corridor-CBS using one-shot MAPF experiments (Section VI-C). Lastly, we verified the generalization capability of the proposed method using a railway map (Section VI-D). All algorithms were processed on a standard desktop PC with an Intel Core i7-6700K CPU and 64 GB of RAM, without a graphics processing unit.

A. LIFELONG MAPF EXPERIMENTS

The lifelong MAPF problem can be directly applied to the warehouse scenario of continuously transporting logistics. Warehouse environments are typically composed of narrow aisles (corridors), so they are suitable for applying topological maps. Therefore, we selected warehouse maps as our primary experimental environment. We considered three warehouse environments with topological maps (i.e., scenario1: Fig. 1, scenario2: Fig. 6a, and scenario3: Fig. 6b). As shown in Figs. 1 and 6a and 6b, the topological map consists of orange lanes and white nodes. Red nodes are isolated parking locations, which are the starting nodes of agents. Blue and green nodes represent a station connected to a rack (blue) or in/outbound (green) for loading and unloading tasks. All tasks in $\mathcal{T}$ are randomly generated at the blue and green nodes. They are located in corridors with a length of 11 (scenario 1) or 22 (scenarios 2 and 3). In scenario 3, we constructed the most challenging map with heavy traffic congestion by reducing the number of task corridors by half that of scenario 2.

The performance of the proposed method (**Anytime-RHCR**) was compared with that of the following lifelong MAPF methods:

- **Reserving dummy paths (RDP)** [5]: This method iteratively plans sub-paths for the agents with new goals while considering other paths as moving obstacles. It reserves dummy paths to isolated parking locations (red nodes in Fig. 6) to avoid deadlock situations.
- **RHCR** [6]: The original RHCR was adopted as the baseline method for the comparison. RHCR only resolves collisions on the bounded time horizon of ω at every planning time.
- **RHCR-RDP** [11]: An integrated method for RHCR and RDP. This method solves a windowed MAPF instance whenever agents move λ time-steps or update one or more goal locations. This method also plans paths by preserving dummy paths.

We used ECBS [12] for RDP, RHCR, RHCR-RDP, and the initial solver of Anytime-RHCR. We also used Corridor-CBS for improvement planning of Anytime-RHCR. We set the runtime limit of all algorithms to 10 s and the replanning period to $\lambda = 10$ time steps. We set the suboptimality factor of ECBS as $\alpha = 1.5$ for RDP. For RHCR, we considered three bounded time horizons, $\omega = 30$, $\omega = 40$, and $\omega = 50$ time steps, with different suboptimality factors, $\alpha = 1.5$, $\alpha = 2.0$, and $\alpha = 2.5$, respectively. We referred to the method for each parameter setting as RHCR$_1$, RHCR$_2$, and RHCR$_3$, respectively. Each suboptimality factor was determined to be the minimum value at which RHCR can always determine a solution within the runtime limit according to the size of ω in all scenarios. We also applied two parameter settings for RHCR-RDP: RHCR-RDP$_1$ ($\omega = 40$ and $\alpha = 2.0$) and RHCR-RDP$_2$ ($\omega = 50$ and $\alpha = 2.5$). For Anytime-RHCR, we also considered two methods with different time horizon settings: Anytime-RHCR$_1$ ($\omega_{init} = 30$, $\alpha = 1.5$, and $\omega_{extd} = 50$) and Anytime-RHCR$_2$ ($\omega_{init} = 40$, $\alpha = 2.0$, and $\omega_{extd} = 50$). We set the number of agents in the modification set $\mathcal{A}_M$ to $N_M = 5$ and the maximum iteration number of the improvement step to 20.

For each scenario, we randomly generated a set of sequential tasks for each agent and simulated 5,000 time steps to evaluate performance. Table 2 presents the throughput (average number of completed tasks per time step) of each method for different numbers of agents N. RDP generally had the worst performance in all three scenarios. RDP works similarly to prioritized path planning [28] but uses the ECBS solver. Therefore, RDP underperforms the RHCR-based method, which plans all paths simultaneously. RHCR showed better performance as the size of ω increased, even with different suboptimality factors of the ECBS. In particular, the performance gap between RHCR$_1$ and

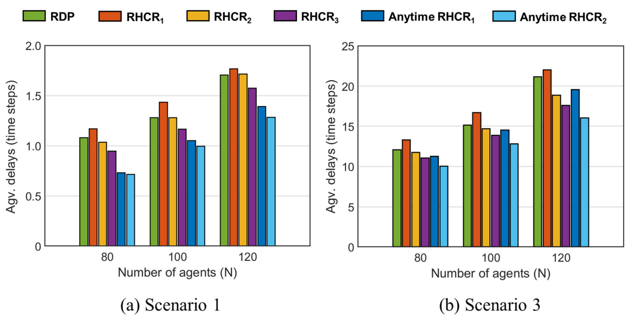


FIGURE 7. **Averaged delay of each method for different numbers of agents *N* in scenarios (a) 1 and (b) 3. The delays are averaged over all tasks processed by each method.**

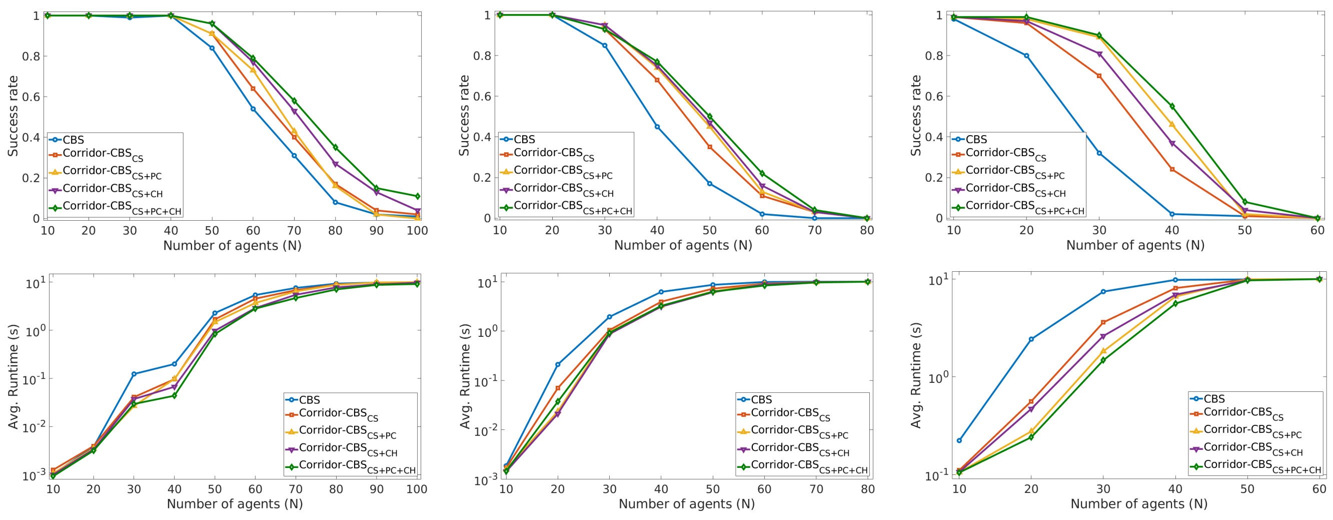
RHCR$_3$ increased in Scenario 3. Therefore, RHCR required a larger ω value in complex scenarios with considerable traffic congestion.

RHCR-RDP generally performed better than RHCR when ω was equal. RHCR-RDP replans paths much more frequently than RHCR, because RHCR-RDP performs replanning whenever an agent's goal is updated. In particular, when the throughput is very large, RHCR replans all paths in almost every time step, which is an inefficient approach that repeats unnecessary planning. Moreover, RHCR-RDP still performs worse than Anytime-RHCR.

Anytime-RHCR$_1$ showed relatively good performance when N was small, particularly when $N = 40$, and showed the best performance in all scenarios. Anytime-RHCR$_1$ performed better than RHCR methods in scenarios 1 and 2. However, Anytime-RHCR$_1$ showed significant performance drops when $N = 100$ or $N = 120$ in scenario 3; in particular, it had a lower performance than RHCR$_2$ and RHCR$_3$. RHCR$_1$ also had the worst performance. Therefore, the initial solution can greatly influence the final solution quality of our method. General corridor lengths are greater than 20 in scenario 3. Therefore, the MAPF solver does not sufficiently cover two corridors when ω_{init} is only 30. This suggested that the proposed method required the minimum specific value of ω_{init} to guarantee a reliable performance under heavy traffic conditions.

Anytime-RHCR$_2$ exhibited the best or second-best performance in all scenarios. In particular, Anytime-RHCR$_2$ showed much higher throughput than RHCR$_3$ in the most challenging condition ($N = 120$ in scenario 3). Our method can find an improved solution for a subset of agents with traffic congestion using Corridor-CBS, which can ultimately improve throughput. This implied that even if the same bounded time horizon was used for the search, the anytime strategy could significantly improve the solution quality in heavy traffic conditions.

In this experiment, we also analyzed the degree of traffic congestion for each method by evaluating the average delay. Fig. 7 shows the averaged delay of all tasks for each method and different N in scenarios 1 and 3. For each task, a *delay*



(a) Scenario 1 (b) Scenario 2 (c) Scenario 3

FIGURE 8. Success rates (upper) and average runtime (lower) of CBS [9] and Corridor-CBS under different numbers of agents (*N*) in three scenarios. We evaluated the performance of Corridor-CBS according to several combinations of corridor symmetry reasoning (CS), prioritizing corridor conflict (PC), and corridor heuristic (CH).

is defined as the difference between the agent's movement time steps and the shortest distance from a start location to a goal location without considering collision. As traffic gets congested, agents frequently wait or move to long detours, resulting in considerable delays; therefore, the averaged delay is strongly related to the degree of congestion. In scenario 1, the average delays are less than 2, whereas in scenario 3, the average delays are more than 10. This means that traffic congestion is much more considerable in scenario 3 than in scenario 1. Anytime-RHCR$_2$ had the best performance of averaged delay in all cases. Anytime-RHCR$_2$ always produces high-quality plans that do not cause congestion as much as possible.

B. INFLUENCE OF MODIFICATION SET SIZE

This section analyzes the influence of the size of the modification set N_M in Anytime-RHCR. As N_M increases, the solution quality of Anytime-RHCR is expected to improve in each iteration. However, a large N_M increases the computation time and reduces the number of iterations of the improvement step within a time limit. Therefore, an appropriate modification set size should be determined according to the complexity of MAPF instances. Table 3 shows the throughput and number of iterations of Anytime-RHCR according to N_M. We only considered a runtime limit of 10 s as the end condition of the improvement step, disregarding the maximum iteration number. Except for this end condition, we used the same settings as Anytime-RHCR$_2$ in Section VI-A.

As shown in Table 3, Anytime-RHCR performed best when N_M was large (7 or 10) in Scenario 1. Because Scenario 1 is relatively less congested, Anytime-RHCR can perform improvement steps more than 100 times, even when

TABLE 3. Throughput and the number of iterations of Anytime-RHCR according to modification set size N_M. "Iter" and "Thpt" represent the iteration number and throughput, respectively. For each case, the best throughput is highlighted in bold font.

		N_M:3		N_M:5		N_M:7		N_M:10	
		Iter	Thpt	Iter	Thpt	Iter	Thpt	Iter	Thpt
Scenario 1	N:100	2365	1.446	1046	1.454	520	1.457	228	**1.462**
	N:120	1930	1.733	824	1.743	356	**1.748**	115	1.747
Scenario 2	N:100	874	1.246	137	**1.257**	35	1.248	20	1.243
	N:120	751	1.448	80	**1.451**	21	1.446	13	1.435
Scenario 3	N:100	251	1.123	43	**1.128**	14	1.122	9	1.120
	N:120	122	1.295	22	**1.301**	12	1.294	7	1.288

$N_M = 10$. In Scenarios 2 and 3, the best performance was obtained when $N_M = 5$. In particular, Anytime-RHCR with $N_M = 7$ or $N_M = 10$ performed fewer than 15 iterations in Scenario 3, which was insufficient to improve the solution quality. These results indicate that Anytime-RHCR with a larger N_M performs better when performing the improvement step a sufficient number of times. However, if sufficient iterations are not guaranteed, Anytime-RHCR cannot significantly improve the performance.

C. EFFECTIVENESS OF CORRIDOR-CBS

In this experiment, we conducted an ablation study to verify the effectiveness of each improvement step of the Corridor-CBS. We evaluated the one-shot MAPF performance of

TABLE 4. Evaluation of the statistical significance of three test cases (Test1: CBS **vs** Corridor-CBS$_{\text{CS+PC+CH}}$**, Test2:** Corridor-CBS$_{\text{CS}}$ **vs** Corridor-CBS$_{\text{CS+PC}}$**, Test3:** Corridor-CBS$_{\text{CS}}$ **vs** Corridor-CBS$_{\text{CS+CH}}$**). The p-values are estimated by performing a paired sample t-test on the runtime of each case. The p-values with significant differences at the 1% significance level are marked in bold. '< 0.0001' represents a p-value smaller than 0.0001.**

	Scenario 1				
	N:20	N:40	N:60	N:80	N:100
Test1	0.0305	0.0215	**< 0.0001**	**< 0.0001**	**0.0017**
Test2	**< 0.0001**	0.9660	**0.0075**	0.5290	0.2197
Test3	**< 0.0001**	0.3256	**< 0.0001**	**0.0017**	0.1860
	Scenario 2				
	N:10	N:20	N:40	N:50	N:70
Test1	0.2411	0.0549	**< 0.0001**	**< 0.0001**	0.0455
Test2	0.3659	0.2070	**0.0053**	**0.0025**	0.3930
Test3	0.0725	0.1920	**0.0031**	**0.0003**	0.2720
	Scenario 3				
	N:10	N:20	N:30	N:40	N:50
Test1	0.2347	**< 0.0001**	**< 0.0001**	**< 0.0001**	0.0891
Test2	0.3221	0.0451	**< 0.0001**	**< 0.0001**	0.7189
Test3	0.2690	0.0421	**< 0.0001**	**< 0.0001**	0.0884

several combinations of corridor symmetry reasoning (**CS**) (Section V-A), prioritizing corridor conflict (**PC**) (Section V-B), and corridor heuristic (**CH**) (Section V-C), based on the original CBS [9]. For each method, we solved the windowed MAPF instances with a bounded time horizon of $\omega = 50$. To create the test datasets, we saved an MAPF instance at each planning time in the previous lifelong MAPF experiments (Section VI-A). We then evaluated the success rates and average runtimes of each method on the test datasets. The success rate refers to the percentage of solved instances within the time limit of 10 s. The average runtime was measured by setting the runtime of an unsolved instance as the time limit of 10 s.

Fig. 8 shows the experimental results for each method for the three scenarios. CBS had the worst performance in terms of success rate and runtime. CBS unnecessarily searches for a huge number of nodes to resolve corridor conflicts, increasing the runtime. In particular, CBS had a runtime of 0.223 s when $N = 10$ in scenario 3, while Corridor-CBS had a runtime of approximately 0.1 s. Corridor-CBS$_{\text{CS}}$ showed improved performance in comparison with CBS, indicating that the corridor symmetry reasoning [10] is effective in topological maps.

Corridor-CBS$_{\text{CS+PC}}$ and Corridor-CBS$_{\text{CS+CH}}$ generally performed better than Corridor-CBS$_{\text{CS}}$ in terms of the success rate and runtime. This indicated that prioritizing conflict and heuristic search methods based on corridor conflict information could improve the MAPF performance in a topological map. These methods were able to efficiently reduce the search space of CBS only using corridor conflict information, rather than creating complex MDDs [14], [18].

Corridor-CBS$_{\text{CS+PC+CH}}$ always had better success rates than CBS. The performance gaps in the success rates between Corridor-CBS$_{\text{CS+PC+CH}}$ and CBS became more significant as traffic congestion increased in a topological map. In particular, when $N = 30$ and $N = 40$ in scenario 3, the success rate of CBS was only 32% and 2%, whereas that of Corridor-CBS$_{\text{CS+PC+CH}}$ was 90% and 55%, respectively. The Corridor-CBS effectively resolved corridor conflicts through several improvement techniques. Therefore, the proposed method could significantly improve the MAPF performance of CBS in a topological map with high traffic congestion.

We also evaluated the statistical significance of each method using a parametric statistical test. We considered three test cases:1) CBS vs Corridor-CBS$_{\text{CS+PC+CH}}$, 2) Corridor-CBS$_{\text{CS}}$ vs Corridor-CBS$_{\text{CS+PC}}$, and 3) Corridor-CBS$_{\text{CS}}$ vs Corridor-CBS$_{\text{CS+CH}}$. We performed a paired-sample t-test on the runtime of each case to estimate the significance of the differences between the two methods. Table 5 presents the p-values of each test case for different numbers of agents N. The lower the p-value, the higher is the significance, whereas the higher the p-value, the lower is the significance. In Table 5, we marked the p-values with significant differences at the 1% significance level in bold. In all scenarios, each test case had a low p-value, except for cases in which N was too large or too small. This means that each pair of methods has a runtime result with a significant difference, except when the problem is simple or difficult to solve.

TABLE 5. Average throughput of each method for different numbers agent N in Scenario 4. We highlighted the best and second-best in bold and underlined fonts, respectively. An empty cell (-) represents a case where a solution was not found within a limited time.

Methods	**Scenario 4**			
	N:80	N:100	N:120	N:140
RHCR$_1$ (ω: 50)	0.461	0.547	0.649	0.747
RHCR$_2$ (ω: 60)	0.468	0.553	0.654	<u>0.758</u>
RHCR$_3$ (ω: 70)	0.469	0.554	-	-
Anytime-RHCR$_1$ (ω_{init}: 50, ω_{extd}: 70)	<u>0.471</u>	<u>0.558</u>	<u>0.658</u>	0.757
Anytime-RHCR$_2$ (ω_{init}: 60, ω_{extd}: 70)	**0.474**	**0.563**	**0.665**	**0.770**

D. LIFELONG MAPF ON RAILWAYS

In this section, we verify the generalization capability of Anytime-RHCR by evaluating its performance in a simplified railway system presented in the flatland challenge [37]. The railway environment is also generally used in MAPF experiments and can be represented as a topological map.

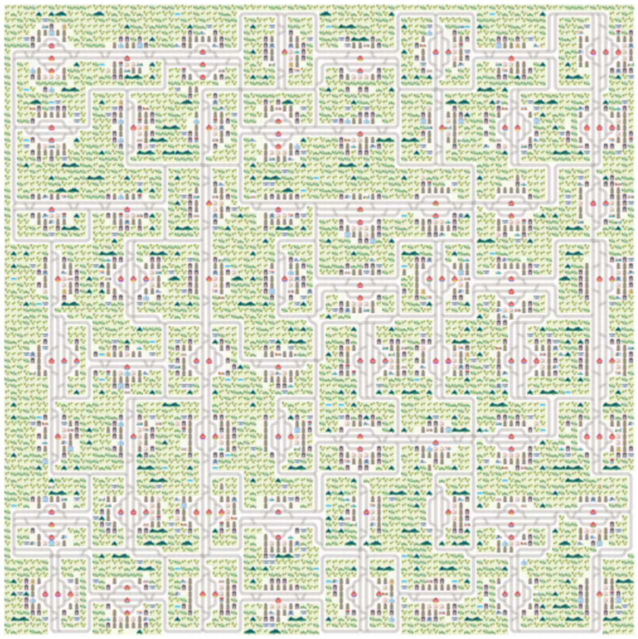


FIGURE 9. Railway environment used in Scenario 4. This environment contains a topological map composed of unstructured multiple tracks.

Fig. 9 shows the railway environment used in this experiment (Scenario 4). Unlike warehouse scenarios (Scenarios 1 - 3), in this railway environment, the topological map consists of multiple tracks that are unstructured in length and shape. Therefore, railway scenarios are more challenging than warehouse scenarios are.

Unlike the Flatland challenge of solving one-shot MAPF problems, we consider the lifelong MAPF problem in the railway environment. Each vertex in a topological map is defined as a railway cell. We generated a test set by randomly selecting the starting locations of the agents and the goal locations from the vertices. Agents can move both forward and backward. Because there is no isolated parking (dummy goal) location in this railway scenario, algorithms utilizing dummy path planning (RDP and RHCR-RDP) are unavailable. Therefore, we only evaluated the performance of Anytime-RHCR and RHCR. We used the same parameter settings for Anytime-RHCR and RHCR in Scenarios 1 - 3.

Table 5 presents the throughput of each method for different numbers of agents N. Anytime-RHCR$_2$ had the best performance in all cases. This indicates that the proposed method is effective, even for unstructured railways. Both the Anytime-RHCR and RHCR performed better as the bounded time horizon increased. However, when N is 120 or 150, RHCR$_3$ does not obtain a solution within the time limit. This means that RHCR requires considerable computation to search for collisions of all agents for up to 70 time steps. However, Anytime-RHCR finds an initial solution with a small time horizon and updates a subset of the solution with an extended time horizon. This approach allows the solver to successfully resolve collisions for up to 70 time steps, significantly improving the lifelong MAPF on topological maps.

VII. CONCLUSION AND FUTURE WORK

We present a novel algorithm based on the RHCR method for lifelong MAPF in topological maps. Generally, a conflict between two agents can be easily resolved by having one agent move sideways. However, in topological maps, sideways movement cannot be performed within a corridor; therefore, RHCR with a small bounded time horizon causes traffic congestion. To address this issue, we applied an anytime approach to the RHCR. The proposed algorithm quickly obtained an initial solution using a suboptimal solver with a small time horizon. It then iteratively refined a subset of the solution using a near-optimal solver with an extended time horizon. This approach allowed the solver to search over a longer time horizon within a limited time. Therefore, the proposed approach could obtain a high-quality solution in topological maps with limited bypasses and high traffic congestion. We also presented a MAPF solver called Corridor-CBS. Corridor-CBS significantly reduced the runtime of CBS by applying several improvements to effectively resolve corridor conflicts.

The experimental results showed that the proposed method provides a much higher throughput than state-of-the-art methods. In particular, in complex scenarios (Scenarios 3 and 4), RHCR caused heavy traffic congestion from a restricted time horizon or even failed to find a solution within a time limit. However, the proposed method produced a high-quality solution with low congestion because it searched for a longer time horizon using the optimal solver. This indicates that the proposed method is effective, even in complex topological maps with irregular and long-range corridors.

Although the proposed method achieves outstanding results in lifelong MAPF experiments, its performance is dependent on the quality of the initial solution (refer to the results of RHCR$_1$ and Anytime-RHCR$_1$ in scenario 3). Because of an inadequate suboptimality factor or initial time horizon, the proposed method fails to obtain an initial solution or returns a solution of very low quality. To address this issue, in future work, we can consider a method to adaptively determine the suboptimality factor and time horizon based on the degree of congestion. Evolutionary algorithms, such as membrane computing [38], [39], can be applied to estimate the best parameters. Orozco-Rosas et al. [38] employed a combination of membrane computing and a pseudo-bacterial genetic algorithm to estimate the parameters of an artificial potential field [40] for single-agent path planning. This approach may be applied to MAPF in the future. Similar to [41] and [42], a machine learning-based method also can be applied to determine the parameters.

In future work, we will study an integrated problem of lifelong MAPF and multi-agent task assignments. For this problem, there are two approaches: coupled [43] and decoupled [4], [11] approaches. The proposed Anytime-RHCR will be applicable to both approaches. We can also adopt a scheduling algorithm of automated storage and retrieval systems (AS/RS) [44], [45] for task assignments. Foumani et al. [45] determined the optimal sequence of orders and items inside each order for AS/RS based on the TSP algorithm with a cross-entropy method. This approach could be applied to plan task sequences of agents.

In addition, we will apply the proposed planning method to a robot management system that operates real robots. We can deliver the initial solution to the robots and then perform path improvement steps in parallel while the robots move. This design can secure more computational time for path planning and efficiently find better solutions. Similar to [25], the incremental search method that reuses previously planned paths can be applied to reduce the computational effort in the robot management system.

REFERENCES

[1] P. R. Wurman, R. D'Andrea, and M. Mountz, "Coordinating hundreds of cooperative, autonomous vehicles in warehouses," *AI Mag.*, vol. 29, no. 1, p. 9, 2008.

[2] A. Bolu and Ö. Korçak, "Adaptive task planning for multi-robot smart warehouse," *IEEE Access*, vol. 9, pp. 27346–27358, 2021.

[3] H. Ma, J. Li, T. K. S. Kumar, and S. Koenig, "Lifelong multi-agent path finding for online pickup and delivery tasks," 2017, *arXiv:1705.10868*.

[4] F. Grenouilleau, W.-J. van Hoeve, and J. N. Hooker, "A multi-label A* algorithm for multi-agent pathfinding," in *Proc. Int. Conf. Automated Planning Scheduling*, vol. 29, 2019, pp. 181–185.

[5] M. Liu, H. Ma, J. Li, and S. Koenig, "Task and path planning for multi-agent pickup and delivery," in *Proc. Int. Joint Conf. Auto. Agents Multiagent Syst. (AAMAS)*, 2019, pp. 1–9.

[6] 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 *Proc. AAAI Conf. Artif. Intell.*, 2021, vol. 35, no. 13, pp. 11272–11281.

[7] Q. Sajid, R. Luna, and K. Bekris, "Multi-agent pathfinding with simultaneous execution of single-agent primitives," in *Proc. Int. Symp. Combinat. Search*, 2012, vol. 3, no. 1, pp. 88–96.

[8] D. Silver, "Cooperative pathfinding," in *Proc. AAAI Conf. Artif. Intell. Interact. Digit. Entertainment*, 2005, vol. 1, no. 1, pp. 117–122.

[9] G. Sharon, R. Stern, A. Felner, and N. R. Sturtevant, "Conflict-based search for optimal multi-agent pathfinding," *Artif. Intell.*, vol. 219, pp. 40–66, Feb. 2015.

[10] J. Li, G. Gange, D. Harabor, P. J. Stuckey, H. Ma, and S. Koenig, "New techniques for pairwise symmetry breaking in multi-agent path finding," in *Proc. Int. Conf. Automated Planning Scheduling*, vol. 30, 2020, pp. 193–201.

[11] Q. Xu, J. Li, S. Koenig, and H. Ma, "Multi-goal multi-agent pickup and delivery," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst. (IROS)*, Oct. 2022, pp. 9964–9971.

[12] M. Barer, G. Sharon, R. Stern, and A. Felner, "Suboptimal variants of the conflict-based search algorithm for the multi-agent pathfinding problem," in *7th Annu. Symp. Combinat. Search*, 2014, pp. 19–27.

[13] H. Ma, D. Harabor, P. J. Stuckey, J. Li, and S. Koenig, "Searching with consistent prioritization for multi-agent path finding," in *Proc. AAAI Conf. Artif. Intell.*, 2019, vol. 33, no. 1, pp. 7643–7650.

[14] J. Li, A. Felner, E. Boyarski, H. Ma, and S. Koenig, "Improved heuristics for multi-agent path finding with conflict-based search," in *Proc. IJCAI*, Aug. 2019, pp. 442–449.

[15] A. Felner, J. Li, E. Boyarski, H. Ma, L. Cohen, T. S. Kumar, and S. Koenig, "Adding heuristics to conflict-based search for multi-agent path finding," in *Proc. Int. Conf. Automated Planning Scheduling*, vol. 28, 2018, pp. 83–87.

[16] R. J. Luna and K. E. Bekris, "Push and swap: Fast cooperative path-finding with completeness guarantees," in *Proc. 22nd Int. Joint Conf. Artif. Intell.*, 2011, pp. 294–300.

[17] B. Binder, F. Beck, F. König, and M. Bader, "Multi robot route planning (MRRP): Extended spatial–temporal prioritized planning," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst. (IROS)*, Nov. 2019, pp. 4133–4139.

[18] E. Boyarski, A. Felner, R. Stern, G. Sharon, D. Tolpin, O. Betzalel, and E. Shimony, "ICBS: The improved conflict-based search algorithm for multi-agent pathfinding," in *Proc. 24th Int. Joint Conf. Artif. Intell.*, 2015, pp. 223–225.

[19] J. Li, D. Harabor, P. J. Stuckey, H. Ma, G. Gange, and S. Koenig, "Pairwise symmetry reasoning for multi-agent path finding search," *Artif. Intell.*, vol. 301, Dec. 2021, Art. no. 103574.

[20] A. Andreychuk, K. Yakovlev, P. Surynek, D. Atzmon, and R. Stern, "Multi-agent pathfinding with continuous time," *Artif. Intell.*, vol. 305, Apr. 2022, Art. no. 103662.

[21] K. Okumura, Y. Tamura, and X. Défago, "Iterative refinement for real-time multi-robot path planning," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst. (IROS)*, Sep. 2021, pp. 9690–9697.

[22] J. Li, Z. Chen, D. Harabor, P. J. Stuckey, and S. Koenig, "Anytime multi-agent path finding via large neighborhood search," in *Proc. Int. Joint Conf. Artif. Intell. (IJCAI)*, Aug. 2021, pp. 1–9.

[23] J. Li, Z. Chen, Y. Zheng, S.-H. Chan, D. Harabor, P. J. Stuckey, H. Ma, and S. Koenig, "Scalable rail planning and replanning: Winning the 2020 flatland challenge," in *Proc. Int. Conf. Automated Planning Scheduling*, vol. 31, 2021, pp. 477–485.

[24] M. Damani, Z. Luo, E. Wenzel, and G. Sartoretti, "Primal_2: Pathfinding via reinforcement and imitation multi-agent learning-lifelong," *IEEE Robot. Autom. Lett.*, vol. 6, no. 2, pp. 2666–2673, Apr. 2021.

[25] N. Madar, K. Solovey, and O. Salzman, "Leveraging experience in lifelong multi-agent pathfinding," 2022, *arXiv:2202.04382*.

[26] M. Čáp, J. Vokřínek, and A. Kleiner, "Complete decentralized method for on-line multi-robot trajectory planning in well-formed infrastructures," in *Proc. Int. Conf. Automated Planning Scheduling*, vol. 25, 2015, pp. 324–332.

[27] B. De Wilde, A. W. T. Mors, and C. Witteveen, "Push and rotate: Cooperative multi-agent path planning," in *Proc. Int. Conf. Auto. Agents Multi-Agent Syst.*, 2013, pp. 87–94.

[28] M. Bennewitz, W. Burgard, and S. Thrun, "Optimizing schedules for prioritized path planning of multi-robot systems," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, vol. 1, May 2001, pp. 271–276.

[29] G. Sartoretti, J. Kerr, Y. Shi, G. Wagner, T. K. S. Kumar, S. Koenig, and H. Choset, "PRIMAL: Pathfinding via reinforcement and imitation multi-agent learning," *IEEE Robot. Autom. Lett.*, vol. 4, no. 3, pp. 2378–2385, Jul. 2019.

[30] M. Rahman, M. A. Alam, M. M. Islam, I. Rahman, M. M. Khan, and T. Iqbal, "An adaptive agent-specific sub-optimal bounding approach for multi-agent path finding," *IEEE Access*, vol. 10, pp. 22226–22237, 2022.

[31] Z. Liu, S. Zhou, H. Wang, Y. Shen, H. Li, and Y.-H. Liu, "A hierarchical framework for coordinating large-scale robot networks," in *Proc. Int. Conf. Robot. Autom. (ICRA)*, May 2019, pp. 6672–6677.

[32] L. Cohen, T. Uras, and S. Koenig, "Feasibility study: Using highways for bounded-suboptimal multi-agent path finding," in *Proc. Int. Symp. Combinat. Search*, 2015, vol. 6, no. 1, pp. 2–8.

[33] K. Kasaura, M. Nishimura, and R. Yonetani, "Prioritized safe interval path planning for multi-agent pathfinding with continuous time on 2D roadmaps," *IEEE Robot. Autom. Lett.*, vol. 7, no. 4, pp. 10494–10501, Oct. 2022.

[34] M. Phillips and M. Likhachev, "SIPP: Safe interval path planning for dynamic environments," in *Proc. IEEE Int. Conf. Robot. Autom.*, May 2011, pp. 5628–5635.

[35] J. Li, Z. Chen, D. Harabor, P. J. Stuckey, and S. Koenig, "MAPF-LNS2: Fast repairing for multi-agent path finding via large neighborhood search," in *Proc. AAAI Conf. Artif. Intell.*, 2022, pp. 10256–10265.

[36] G. Sharon, R. Stern, M. Goldenberg, and A. Felner, "The increasing cost tree search for optimal multi-agent pathfinding," *Artif. Intell.*, vol. 195, pp. 470–495, Feb. 2013.

[37] S. Mohanty, E. Nygren, F. Laurent, M. Schneider, C. Scheller, N. Bhattacharya, J. Watson, A. Egli, C. Eichenberger, C. Baumberger, G. Vienken, I. Sturm, G. Sartoretti, and G. Spigler, "Flatland-RL: Multi-agent reinforcement learning on trains," 2020, *arXiv:2012.05893*.

[38] U. Orozco-Rosas, K. Picos, and O. Montiel, "Hybrid path planning algorithm based on membrane pseudo-bacterial potential field for autonomous mobile robots," *IEEE Access*, vol. 7, pp. 156787–156803, 2019.

[39] U. Orozco-Rosas, O. Montiel, and R. Sepúlveda, "Mobile robot path planning using membrane evolutionary artificial potential field," *Appl. Soft Comput. J.*, vol. 77, pp. 236–251, Apr. 2019.

[40] O. Khatib, "Real-time obstacle avoidance for manipulators and mobile robots," in *Proc. IEEE Int. Conf. Robot. Autom.*, vol. 2, Mar. 1985, pp. 500–505.
[41] T. Huang, J. Li, S. Koenig, and B. Dilkina, "Anytime multi-agent path finding via machine learning-guided large neighborhood search," in *Proc. AAAI Conf. Artif. Intell.*, 2022, pp. 9368–9376.
[42] T. Huang, B. Dilkina, and S. Koenig, "Learning node-selection strategies in bounded suboptimal conflict-based search for multi-agent path finding," in *Proc. Int. Joint Conf. Auto. Agents Multiagent Syst. (AAMAS)*, 2021, pp. 1–9.
[43] 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 Robot. Autom. Lett.*, vol. 6, no. 3, pp. 5816–5823, Jul. 2021.
[44] S. Brezovnik, J. Gotlih, J. Balič, K. Gotlih, and M. Brezočnik, "Optimization of an automated storage and retrieval systems by swarm intelligence," *Proc. Eng.*, vol. 100, pp. 1309–1318, Jan. 2015.
[45] M. Foumani, A. Moeini, M. Haythorpe, and K. Smith-Miles, "A cross-entropy method for optimising robotic automated storage and retrieval systems," *Int. J. Prod. Res.*, vol. 56, no. 19, pp. 6450–6472, Oct. 2018.

SOOHWAN SONG received the B.S. degree in information and communication engineering from Dongguk University, Seoul, South Korea, in 2013, and the M.S. and Ph.D. degrees in computer science from the Korea Advanced Institute of Science and Technology (KAIST), Daejeon, South Korea, in 2015 and 2020, respectively. He was a Postdoctoral Researcher at the Neuro-Machine Augmented Intelligence Laboratory, KAIST. Since March 2021, he has been a Senior Researcher with the Intelligent Robotics Research Division, Electronics and Telecommunication Research Institute (ETRI). His research interests include robotics, motion planning, multi-robot systems, and computer vision.

KI-IN NA received the B.S. degree in mechanical engineering from the Pohang University of Science and Technology (POSTECH), Pohang, Republic of Korea, in 2009, and the M.S. and Ph.D. degrees in robotics program from the Korea Advanced Institute of Science and Technology (KAIST), Daejeon, Republic of Korea, in 2011 and 2022, respectively. Since 2011, he has been a Research Scientist with the Electronics and Telecommunication Research Institute (ETRI), Daejeon. His current research interests include detection and tracking of moving objects, socially-aware navigation, human–robot interaction, and artificial intelligence for real applications.

WONPIL YU received the B.S. degree in control and instrumental engineering from Seoul National University, Seoul, South Korea, in 1992, and the M.S. and Ph.D. degrees in electrical engineering from the Korea Advanced Institute of Science and Technology (KAIST), Daejeon, South Korea, in 1994 and 1999, respectively. He is with the Intelligent Robot Research Group, Electronics and Telecommunication Research Institute (ETRI), South Korea. He is also active in developing technology standards in mobile robotics, where he is currently working with the Map Data Representation (MDR) working group with a technical sponsorship from the IEEE Robotics and Automation Society. Prior to joining ETRI, in 2001, he worked for Agency for Defense Development (ADD), Daejeon, where he was involved in the development of a precision stabilizer for radar seeker system. His research interests include mobile robotics, agricultural robotics, and perception technology.