2023 62nd IEEE Conference on Decision and Control (CDC)
December 13-15, 2023. Marina Bay Sands, Singapore

Conflict-Free Node-to-Robot Scheduling for Lifelong Operation in a Warehouse with Narrow-Corridor Environment

Sharad Kumar Singh*, Hemantharaj M*, Sayantani Bhattacharya, Manish Jha

***Abstract*— This paper presents a solution to lifelong Multi-Agent Path Finding (MAPF) problems for long and narrow-corridor environments. In this setting, robots need to navigate conflict-free paths while adapting to new goals. We propose an algorithm called Conflict-Free Node-To-Robot Scheduling (CFNRS), which effectively coordinates the paths of robots on a given graph in a constrained environment. The algorithm assigns nodes of the graph, ensuring no conflicts with other robots. In particular, we introduce a Deadlock-Detection and Resolution mechanism to find and resolve conflicts and ensure conflict-free paths. We have introduced a problem-reduction technique for improved efficiency. The proposed algorithms are evaluated through simulations in narrow-corridor environments and compared to existing state-of-the-art MAPF solvers, demonstrating their validity and effectiveness in ensuring that robots can navigate conflict-free paths.**

***Index Terms*— Multi-agent Systems, Logistics, Multi-Agent Path Finding.**

I. INTRODUCTION

Multi-Agent Path Finding (MAPF) is a well-researched problem in the field of Artificial Intelligence (AI) and robotics, with numerous real-world applications, including automated warehouses, service robots, air traffic control, etc. [1], [2], [3]. The problem involves finding a conflict-free path for multiple agents from their starting locations to their desired goal locations amidst known static obstacles in the environment [4]. MAPF is studied in two variants: classic MAPF and lifelong MAPF. In classic MAPF, the robots are assigned with single-source and single-destination; however, in a lifelong variant of MAPF, robots receive new goals once they reach the original goal location.

An exhaustive literature exists on MAPF problems, with various methods proposed to solve them. Some of the most popular ones are Conflict-Based Search (CBS) [5], Explicit Estimation CBS (EECBS) [6], Priority-Based Search (PBS) [7] and Pairwise Symmetry Reasoning [8] for the classic problem. An offline version of the lifelong MAPF solver is introduced in [9]. In the online setting, lifelong MAPF solvers are introduced in [2], [3], [10], and [11], where re-planning is performed to complete multiple assigned tasks. Since MAPF is an NP-hard problem, finding optimal solutions for large-scale MAPF problems always requires a trade-off between computation time and optimality [12].

The rise of e-commerce and online retail has led to high demand for warehouse storage, prompting automated warehouses with narrow corridors to optimize storage density and improve efficiency. These environments require robots to store and retrieve items from shelves in the narrow corridors, as shown in Fig. 1, but the obstacle clusters formed by the shelves pose challenges in finding conflict-free paths for the fleet of robots [13]. MAPF algorithms face two main challenges in long and narrow corridors: congestion and deadlocks. Congestion occurs when agents are stuck in a particular area, unable to move because other agents are blocking their path. Deadlocks occur when agents are blocked in a way that no agent can move, leading to a frozen state. In [14], a multi-phase planning algorithm is demonstrated for finding collision-free paths in a similar environment for the narrow corridor. An efficient dual-layer algorithm is proposed in [15] to find the collision-free path in a narrow-lane environment for multi-agent path planning.

This paper addresses the MAPF challenge in a narrow and long corridor setting, focusing on collision-free navigation for agents. The major contributions of this paper are as follows:

- We propose a novel Conflict-Free Node-To-Robot Scheduling (CFNRS) algorithm that strategically assigns nodes to robots, ensuring conflict avoidance and unobstructed movement.
- We introduce a Deadlock-Detection and Resolution Algorithm that identifies and resolves deadlocks through wait-spot strategies.
- We demonstrate a problem-reduction technique that streamlines deadlock detection and resolution by generating a more manageable problem set.

The rest of the paper is organized as follows. Section II introduces the MAPF problem formulation, while section III presents the conflict-free node-to-robot scheduling solution approach. Section IV delves into an in-depth comparative study and simulation results that validate the proposed methods. Conclusions and future prospects

All the authors are with Addverb Technologies Ltd., Noida, India, sharad.singh@addverb.com, hemantharaj.m@addverb.com, sayantani.bhattacharya@addverb.com, manish.jha@addverb.com .

*Both authors contributed equally to this work.

are outlined in section V.

II. Problem Formulation

We model the warehouse environment as an undirected graph $\mathcal{G}(N, E)$, with nodes N denoting locations and edges E indicating connections. The problem involves a fleet of n robots, denoted as $\mathcal{R} = \{R_1, \cdots, R_n\}$. In a *classic* MAPF problem, our goal is to find conflict-free paths for each robot, with source node s_i and goal node g_i. However, in the *lifelong* MAPF scenario, each robot continuously receives new goals after reaching the current goal. Using Dijkstra's algorithm, we determine the shortest path P_i for each robot. The complete set of robot paths is denoted as $\mathcal{P} = \{P_1, \cdots, P_n\}$. We use $\mathcal{P}[\text{start}]$ and $\mathcal{P}[\text{goal}]$ to represent the set of robots' source and goal nodes, respectively. The warehouse layout is described as **warehouse-C-R-L-W-S.map**, with C and R as columns and rows, L and W as corridor length and width, and S as shelve spacing. For instance, in Fig. 1, **warehouse-2-4-7-2-1.map** has $C = 2$ (c_1 and c_2), $R = 4$ (r_1 to r_4), $L = 7$, $W = 2$, and $S = 1$. Narrow corridors created by storage shelves (pink cells in Fig. 1) pose challenges for collision-free paths, requiring strategic coordination among robots due to one-robot-at-a-time passages.

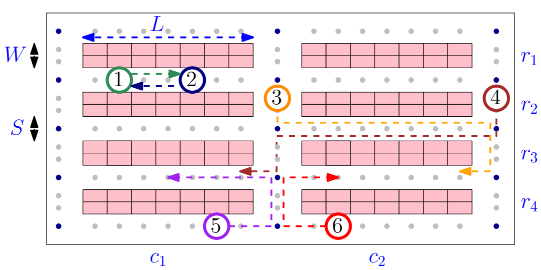


Fig. 1. A **warehouse-C-R-L-W-S.map** with corridors length L, storage shelves (in pink cells) and robots' initial location (circles).

This paper addresses the lifelong MAPF problem in a similar environment by acquiring a conflict-free node-to-robot schedule for a given path-set $\mathcal{P}$. This goal can be divided into the following sub-problems:

Problem 1: Obtain a reduced set from the initial path set $\mathcal{P}$ to minimize conflicts with other robots.

Problem 2: Given the reduced problem set obtained from Problem 1, identify the robot pairs in deadlock and obtain an updated path set using deadlock resolution.

Problem 3: For the updated path set obtained from Problem 2, find a collision-free node-to-robot schedule for all robots at each node in path P_i, considering the robots receive a new goal after reaching the current one.

In this paper, we assume that all robots have unique start and goal nodes on graph $\mathcal{G}$, uninterrupted power supply for continuous operation, and receive new goals upon reaching the current one, nullifying their previous behaviour (as in [1], [7]), which facilitates formulating the lifelong MAPF problem.

III. Conflict-Free Node-To-Robot Scheduling

In this section, we present our solutions to the problems discussed in section II.

A. Definitions:

To begin with, we define three key concepts: node dependency, safe spot for a robot, and low-priority robot.

1) Node Dependency Between Robots: In the context of robot paths, the dependence of path P_i of robot R_i on path P_j of robot R_j arises when the starting location of R_j is positioned on P_i, meaning $P_j[\text{start}] \in P_i$. If P_i and P_j share common nodes ($P_i \cap P_j[\text{start}] \neq \phi$), P_i relies on P_j for nodes $P_i \cap P_j$, termed as $\mathcal{N}_{dij}$, indicating P_i depends on P_j. Mathematically,

$$\mathcal{N}_{dij} := \begin{cases} \phi & \text{if } P_i \cap P_j[\text{start}] = \phi \\ P_i \cap P_j & \text{if } P_i \cap P_j[\text{start}] = P_j[\text{start}] \end{cases}$$

Example 1: To illustrate, consider a scenario with six robots navigating a narrow-corridor environment (Fig. 2). The robots follow the following shortest paths:

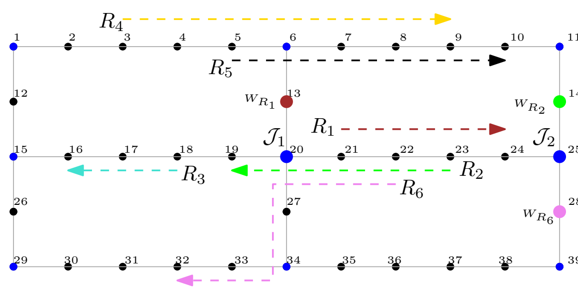


Fig. 2. Paths of 6 robots in the narrow-corridor environment

$P_1 = \{21, 22, 23, 24\}, P_2 = \{23, 22, 21, 20, 19\}, P_3 = \{18, 17, 16\}, P_4 = \{3, 4, 5, 6, 7, 8, 9\}, P_5 = \{5, 6, 7, 8, 9, 10\}, P_6 = \{22, 21, 20, 27, 34, 33, 32\}$. Here, P_3 is not dependent on any other robot, as starting node $P_j[\text{start}]$, $j \neq \{3\}$ does not appear in the path of P_3. Thus, $\mathcal{N}_{dj3} = \phi$ for all robots $j = \{1, 2, 4, 5, 6\}$. Similarly, $\mathcal{N}_{dj4} = \phi \;\forall\; j = \{1, 2, 3, 5, 6\}$. For R_4 and R_5, where $P_4 \cap P_5 = \{5, 6, 7, 8, 9\}$, $\mathcal{N}_{d45} = \{5, 6, 7, 8, 9\}$ implies R_4 awaits these nodes for reaching its goal. Also, $\mathcal{N}_{d12} = \mathcal{N}_{d21} = \{21, 22, 23\}$ and $\mathcal{N}_{d16} = \mathcal{N}_{d61} = \{21, 22\}$.

2) Safe spot for a robot: A node $\mathcal{N}_i \in P_i$ becomes a safe spot for robot R_i if R_i can access $\mathcal{N}_i$ collision-free, and $\mathcal{N}_i$ does not reside on any other robot's path (as $\mathcal{N}_i \notin P_j$, $\forall j \neq i$). For instance, in Fig. 2, $\mathcal{N}_3 = P_3[\text{start}] \notin P_j \;\forall\; j = \{1, 2, 4, 5, 6\}$ is the safe spot for robot R_3. Similarly, $\mathcal{N}_4 = P_4[\text{start}] \notin P_j \;\forall\; j = \{1, 2, 3, 5, 6\}$ is the safe spot of R_4, while $\mathcal{N}_5 = \{10\} \in P_5$ serves as the safe spot for R_5 (as $\mathcal{N}_5 \notin P_j \forall\; j = \{1, 2, 3, 4, 6\}$).

3) Low-Priority Robot: A robot R_i is defined as a low-priority robot if it can reach a safe spot $\mathcal{N}_i$ where $\mathcal{N}_{dji} = \phi \ \forall j \neq i$. We define the set of low-priority robots as $\mathcal{L}_R$ and their corresponding paths as $\mathcal{L}_P$. For instance, in Fig. 2, as $\mathcal{N}_3 = P_3[\text{start}]$, $\mathcal{N}_4 = P_4[\text{start}]$ and $\mathcal{N}_5 = \{10\}$, we have $\mathcal{L}_R = \{R_3, R_4, R_5\}$ and $\mathcal{L}_P = \{P_3, P_4, P_5\}$ respectively.

Next, we present the solution to Problem 1 based on the definitions introduced above.

B. Problem Reduction Algorithm

This section introduces a technique for simplifying multi-robot motion planning problems by eliminating non-conflicting low-priority robots. When a robot R_i can access a safe spot, i.e., $\mathcal{N} \notin P_j$, $\forall j \neq i$, its path P_i is added to $\mathcal{L}_P$. Algorithm 1 describes the proposed problem-reduction technique. Given a graph $\mathcal{G}$ and the set of paths of all robots $\mathcal{P}$, steps 3-5 ensure collision-free movement along all robots' paths. Steps 6-9 identify low-priority robots with safe spots and include their paths in $\mathcal{L}_P$. The resulting reduced problem set $\mathcal{P}'$ contains only paths of robots requiring coordination and mutual dependence. For example, in Fig. 2, robots R_3, R_4, and R_5 are low-priority robots as they can each access a safe spot away from other robots' paths, hence $\mathcal{L}_P = \{P_3, P_4, P_5\}$. In contrast, robots R_1, R_2, and R_6 cannot access such safe spots due to $\mathcal{N}_{d12} = \mathcal{N}_{d21} \neq \phi$ and $\mathcal{N}_{d16} = \mathcal{N}_{d61} \neq \phi$.

Algorithm 1: Problem Reduction Algorithm

Input : Paths: $\mathcal{P} = \{P_1, \cdots, P_n\}$, Graph: $\mathcal{G}$
Output : Reduced Path Set: $\mathcal{P} \backslash \mathcal{L}_P$
Initialize: $\mathcal{L}_P = [\ ]$
1 **for** $R_i \leftarrow R_1$ **to** R_n **do**
2   **for** $P_i[j] \leftarrow P_i[start]$ **to** $P_i[goal]$ **do**
3   **if** $P_i[j] = \mathcal{P}[start] \backslash P_i[start]$ **then**
4   break; ▷ Collision check
5   **end if**
6   **if** $\mathcal{N}_i = P_i[j] \notin \mathcal{P} \backslash P_i$ **then**
7       $\mathcal{L}_P$.append(P_i);
8   break;
9   **end if**
10   **end for**
11 **end for**
12 **return** $\mathcal{P} \backslash \mathcal{L}_P$

Next, we evaluate the outcome of Algorithm 1 to detect robot deadlock pairs, as outlined in Problem 2.

C. Deadlock Detection Algorithm

A set of robots is said to be in a deadlock if multiple agents are blocked and unable to reach their destinations. In this case, there exists a chain of dependencies between the robots, and no robot can move without colliding with another robot. Algorithm 2 provides a deadlock detection approach for the input $\mathcal{P}' = \mathcal{P} \setminus \mathcal{L}_P$ obtained from Algorithm 1. The algorithm starts by creating a dependency graph DG for all robots in steps 1-7, where an edge between two robots, R_i and R_j, is formed if their next move depends on each other, i.e., $\mathcal{N}_{dij} \neq \phi$. Next, in step 8, we obtain all the cycles in the graph and store them in $\mathcal{C}$. In steps 9-15, we check if the given cycle is in deadlock. Using cycle $\mathcal{C}_i \in \mathcal{C}$, we get the set of all the robots $\mathcal{R}_{\mathcal{C}i}$ forming the cycle. If all the robots in the cycle satisfy the condition that the robot's next location depends on the movement of some other robot, i.e., all the robots in $R_k \in \mathcal{R}_{\mathcal{C}i}$ satisfy the condition of $P_k[\text{next\_step}] \in \cup \mathcal{N}_{dij} \forall R_i, R_j \in \mathcal{R}_{\mathcal{C}i}$, we include the cycle as Deadlock cycle $\mathcal{D}_c$, as indicated in step 13. Further, the deadlock pairs are obtained using step 12.

Algorithm 2: Deadlock Detection Algorithm

Input : Paths: $\mathcal{P} \backslash \{\mathcal{L}_P\}$, Graph: $\mathcal{G}$
Output : Deadlock pairs: $\mathcal{D}_p$, Deadlock cycle: $\mathcal{D}_c$
Initialize: $\mathcal{C} = [\ ]$, $\mathcal{D}_c = [\ ]$, Dependency Graph: DG
1 **for** $R_i \leftarrow R_1$ **to** R_n **do**
2   **for** $R_j \leftarrow R_1$ **to** R_n, $R_j \neq R_i$ **do**
3   **if** $\mathcal{N}_{dij} \neq \phi$ **then**
4       $DG.addedge(E_{ij})$
5   **end if**
6   **end for**
7 **end for**
8 $\mathcal{C} \leftarrow$ Get all the Cycles in DG
9 **for** $\mathcal{C}_i \in \mathcal{C}$ **do**
10   $\mathcal{R}_{\mathcal{C}i} \leftarrow$ Get all the Robots in $\mathcal{C}_i$
11   **if** $P_k[next\_step] \in \cup_{R_i, R_j \in \mathcal{R}_{\mathcal{C}i}} \mathcal{N}_{dij}, \forall R_k \in \mathcal{R}_{\mathcal{C}i}$ **then**
12     $\mathcal{D}_p$.append($\mathcal{R}_{\mathcal{C}i}$) ▷ Add deadlock pair
13     $\mathcal{D}_c$.append($\mathcal{C}_i$) ▷ Add deadlock cycle
14   **end if**
15 **end for**
16 **return** $\mathcal{D}_p$, $\mathcal{D}_c$

Theorem 3.1: Let $\mathcal{P}$ be paths for all robots, and $\mathcal{P}' = \mathcal{P} \backslash \mathcal{L}_P$ is the reduced path set. Then, deadlock detection in $\mathcal{P}'$ is equivalent to detection in $\mathcal{P}$.

Proof: Consider dependency graph $\mathcal{DG}$ formed with $\mathcal{P}$. Removing $\mathcal{L}_P$ from $\mathcal{P}$ corresponds to removing a subset of nodes and edges from this graph. Specifically, we remove all the nodes with no incoming edges as they are low-priority robots. Since $\mathcal{L}_P$ does not intersect with any other robot's path, removing $\mathcal{L}_P$ from $\mathcal{P}$ does not affect any cycles in the remaining dependency graph. Thus, deadlock detection in $\mathcal{P}'$ is equivalent to detection in $\mathcal{P}$. ■

To illustrate, in Fig. 2, the reduced problem $\mathcal{P} \backslash \{P_3, P_4, P_5\}$ is given as an input to the deadlock detection Algorithm 2. The dependency graph $DG = \{E_{12}, E_{21}, E_{16}, E_{61}, E_{26}\}$ where three cycles $\mathcal{C}_1$, $\mathcal{C}_2$, $\mathcal{C}_3$ are formed with $\mathcal{R}_{\mathcal{C}1} = \{R_1, R_2\}$ and $\mathcal{R}_{\mathcal{C}2} = \{R_1, R_6\}$ and $\mathcal{R}_{\mathcal{C}3} = \{R_1, R_2, R_6\}$. As $\mathcal{N}_{d12}, \mathcal{N}_{d21} = \{21, 22, 23\}$, $\mathcal{N}_{d16}, \mathcal{N}_{d61} = \{21, 22\}$, all the robots in

$R_k \in \mathcal{R}_{\mathcal{C}i}$ satisfy the condition of P_k[next_step] $\in \cup \mathcal{N}_{dij} \forall R_i, R_j \in \mathcal{R}_{\mathcal{C}i}$. Hence, the deadlock cycle $\mathcal{D}_c = \{\mathcal{C}_1, \mathcal{C}_2, \mathcal{C}_3\}$ is an output of the Algorithm 2.

In the next section, we discuss the approach for resolving deadlocks obtained from the detection algorithm.

D. Deadlock Resolution Algorithm

This section presents a technique for resolving deadlocks for a given deadlock cycle $\mathcal{D}_c$. First, we identify the robots in deadlock and create a directed cyclic graph $\mathcal{D}_G(R_D, \mathcal{D}_p)$, where nodes represent a set of robots in deadlock R_D, and edges represent deadlock pairs $\mathcal{D}_p$. Next, we obtain the Feedback Node Set $\mathcal{F}$, which is the set of nodes (robots) that, when removed from $\mathcal{D}_G$, transforms it into a directed acyclic graph. Alternatively, $\mathcal{F}$ is the set of all subsets F of $\mathcal{D}_G$ such that $R_D \backslash F$ in the graph $\mathcal{D}_G$ forms an acyclic graph.

To resolve deadlocks, we search for wait spots (deviating from the original path) W_F for all robots in a selected node-set $F \in \mathcal{F}$ using junctions, which differs from safe-spot (Section III-A.2). These wait spots provide temporary locations for robots to move and free up space for other agents in the deadlock cycle and should not be in the path of any robot in the deadlock cycle ($\mathcal{P}_D$) to avoid collisions, i.e., $W_F \notin \mathcal{P}_D$ $\forall R_i \in F$. Narrow corridor layouts with limited detours, possibly occupied by other robots, make wait spot-based resolution a viable and rapid option, preventing the system from getting stuck in an unproductive state. If wait spots can be found for all robots in *any* node set $F \in \mathcal{F}$, we update the paths for all the robots in F (denoted by P_F) by adding a subpath from the start to the wait-spot, and from the wait-spot to the goal, i.e., $P_F \leftarrow P_F[\text{start} : W_F] \cup P_F[W_F : \text{goal}]$. The efficiency of this technique depends on the system's size, complexity, and the number of deadlock cycles requiring resolution.

Remark 1: To reduce computation time, we can limit the solution to cases where any node-set $F \in \mathcal{F}$ can provide the updated path rather than comparing the costs of all node sets, which is an NP-Hard problem.

Algorithm 3 proposes steps to resolve the deadlock cycle obtained using Algorithm 2. Step 1 creates $\mathcal{D}_G(R_D, \mathcal{D}_p)$, and step 2 provides $\mathcal{F}$, which contains the set of nodes (robots) that can be removed from $\mathcal{D}_G$ to make it a directed acyclic graph. In steps 3-8, for every $F \in \mathcal{F}$, if W_F is obtained for all robots in F, paths are updated by adding the wait spots to the path.

In Fig. 2, deadlock cycles, $\mathcal{C}_1$: $(R_1 - R_2)$, $\mathcal{C}_2$: $(R_1 - R_6)$, and $\mathcal{C}_3$: $(R_1 - R_2 - R_6)$ are provided as input to Algorithm 3. The algorithm identifies the feasible node sets, $F_1 = \{R_1\}$ and $F_2 = \{R_2, R_6\}$, which remove cycles from $\mathcal{D}_G$. It then finds wait spots, $W_{F_1} = \{13\}$ for $F_1 = \{R_1\}$; $W_{F_2} = \{14, 28\}$ for $F_2 = \{R_2, R_6\}$, via junctions $\mathcal{J}_1$ and $\mathcal{J}_2$ (Fig. 2). Among the feasible sets, F_1 is selected and wait spot $W_{F_1} = \{13\}$ is assigned to R_1 and returns the updated path $P_1 = \{21, 20, \mathbf{13, 20, 21, 22}, 23, 24\}$ (Fig. 3).

Algorithm 3: Deadlock Resolution Algorithm

Input : Paths: $\mathcal{P}$, Graph: $\mathcal{G}$, Deadlock cycle: $\mathcal{D}_c$, Deadlock pair: $\mathcal{D}_p$
Output : Updated Paths
1 $\mathcal{D}_G(R_D, \mathcal{D}_p) \leftarrow$ *Get the robots in* $\mathcal{D}_c$
2 Get $\mathcal{F} = \{F \subset \mathcal{D}_G$ such that $R_D \backslash F$ in $\mathcal{D}_G$ is Directed Acyclic Graph$\}$
3 **for** *every* $F \in \mathcal{F}$ **do**
4   Get wait spot for all robots in F, W_F $\forall$ $R_i \in F$, $W_F \notin \mathcal{P}_D$ via Junction
5   **if** $W_F \neq \phi$ **then**
6   **return** $P_F \leftarrow P_F[\text{start} : W_F] \cup P_F[W_F : \text{goal}]$
7   **end if**
8 **end for**

In the next section, we discuss the move-one-step scheduler to get a collision-free path for the robots.

E. Node-to-Robot Scheduling Algorithm

In the Move-one-step scheduler algorithm, multiple robots traverse a graph to reach their respective goals without colliding with each other. Algorithm 4 works iteratively by attempting to move each robot one step at a time, checking for collisions and deadlocks (in steps 2-8), and updating the robot's position if no conflicts are detected. If a conflict or deadlock is detected, the algorithm rejects the updated path and keeps the robot at its current node; otherwise, if there is no collision or deadlock, the robot gets its schedule for that node and updates its start position. The algorithm terminates when the final node-to-robot schedule for all robots is obtained.

Algorithm 4: Move-One-Step Scheduler

Input : Paths: $\mathcal{P} = \{P_1, \cdots, P_n\}$, Graph: $\mathcal{G}$
Output : Node-to-robot schedule
1 **for** $R_i \leftarrow R_1$ **to** R_n **do**
2   **if** P_i*[next_step]* $\neq \mathcal{P}$*[start]*$\backslash P_i$*[start]* **then**
3         $P'_i \leftarrow P_i[\text{next\_step} : \text{goal}]$ $\mathcal{P}' = \{\mathcal{P} \backslash P_i\} \bigcup P'_i$
4   **if** *Deadlock Detection Algorithm($\mathcal{P}'$, $\mathcal{G}$)*$= \phi$ **then**
5             $R_i : P_i[\text{start}] \leftarrow P'_i[\text{start}]$
6             $\mathcal{P} \leftarrow \mathcal{P}'$
7   **end if**
8   **end if**
9 **end for**
10 **return** Node_to_robot_schedule

The complete algorithmic framework is proposed in CFNRS Algorithm 5. Here, Updated_Robot_Path function (in steps 1-5) takes the current paths of all robots as input, calculates the paths that do not intersect with each other using Algorithm 1, checks for deadlocks using Algorithm 2, and resolves them using Algorithm 3. The updated paths are then returned and used for generating a node-to-robot schedule using Algorithm 4. Updated_Robot_Path is repeatedly called whenever a robot is assigned a new goal, as it can potentially lead to new deadlocks. Overall, Algorithm 5 ensures conflict-free traversal of robots to their respective goals.

Algorithm 5: CFNRS Algorithm

Input : Paths: $\mathcal{P} = \{P_1, \cdots, P_n\}$, Graph: $\mathcal{G}$
Output : Updated Paths $\mathcal{P}'$, Node-to-robot schedule
1 **Function** *Updated_Robot_Path($\mathcal{P}'$, $\mathcal{G}$)*:
2 Get $\mathcal{P}' \setminus \mathcal{L}_P$ using Algorithm 1 for input $(\mathcal{P}', \mathcal{G})$
3 Get $\mathcal{D}_p, \mathcal{D}_c$ using Algorithm 2 for input $\mathcal{P}' \setminus \mathcal{L}_P$
4 Find updated paths for robots in deadlock using Algorithm 3 for input $\mathcal{D}_p, \mathcal{D}_c$
5 **return** Updated paths $\mathcal{P}'$
6 $\mathcal{P}' \leftarrow \mathcal{P}$
7 Updated_Robot_Path($\mathcal{P}'$, $\mathcal{G}$)
8 **while** *all the robots not at the final goal* **do**
9 **for** $R_i \leftarrow R_1$ **to** R_n **do**
10 **if** *R_i reached its current goal* **then**
11 Assign next goal to R_i and get P_i'
12 $\mathcal{P}' = \{\mathcal{P} \setminus P_i\} \bigcup P_i'$
13 Updated_Robot_Path($\mathcal{P}'$, $\mathcal{G}$)
14 **end if**
15 **end for**
16 Get Node_to_robot_schedule using Algorithm 4
17 **end while**

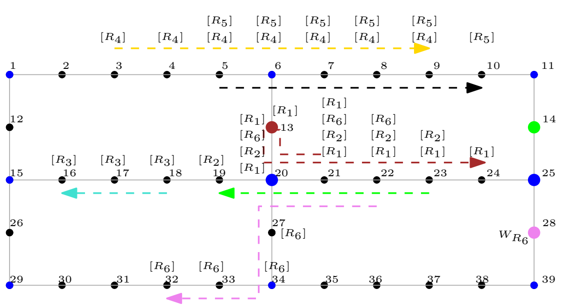


Fig. 3. Schedule for all the robots at each node in the form of stack

Illustrated in Fig. 3, Algorithms 5 unfold as follows: Steps 1-5 ascertain conflict-free paths, yielding an updated path for R_1 with $W_{R_1} = \{13\}$, via Algorithm 3. No path revisions occur for singular task assignments (steps 9-15). In step 16, each robot sequentially probes node-to-robot schedules within graph node $\mathcal{G}(N, E)$. Employing Algorithm 4, R_1 engages node $P_1[\text{next\_step}] = \{20\}$, gauging collisions (step 3) and deadlock potential (steps 5-8) against peers. Unimpeded by deadlock, R_1 adopts the $\{20\}$ schedule. As R_1 arrives at $\{13\}$ and proceeds towards $\{20\}$, it evaluates collision risks with other robots. In this assessment, a deadlock involving R_2 surfaces, resulting in the non-issuance of a schedule for this node. This iterative procedure extends to all robots, ultimately generating the schedule.

IV. Experimental Results

In this section, we compare the performance of the proposed Algorithm with state-of-the-art MAPF solvers in C++. All the experiments are performed on a machine with an 11th Gen Intel® Core™ i7-11850H processor with 16 cores, running at 2.50 GHz, and equipped with 32 GB of RAM.

A. Comparison results for Classic MAPF

In this section, we compare our CFNRS Algorithm with EECBS (sub-optimality factor $\alpha = 1.1$), EECBS ($\alpha = 1.2$), EECBS ($\alpha = 2$) [6], optimized PBS [7], and CBSH [8] for a long-corridor environment.

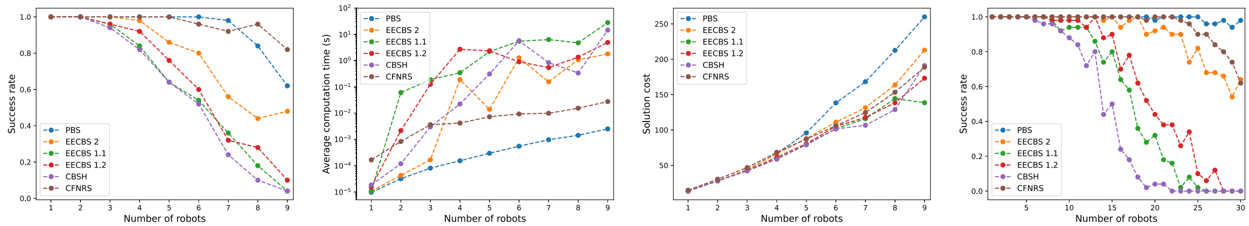
1) Scenario 1 (S1): We analyze the **warehouse-1-2-18-2-1.map** with an 18-grid corridor in a 7×20 grid warehouse, where 52% of total grids form obstacle clusters. In Fig. 4(a), we report the success rate out of 50 instances for each number of agents (with random starting and target vertices and the mean values reported), given the 60 s timeout. For each number of agents, the average computation time for all the successful cases (excluding the timeout cases) is depicted in Fig. 4(b). Due to the small layout, increasing the number of robots impacts the success rate and computation time for any MAPF solver. The total solution cost (sum of time steps required for a robot to reach its goal) for each number of agents is shown in Fig. 4(c). Notably, the optimized PBS and CFNRS solvers provide better performance than the other solvers for the narrow-corridor environment.

2) Scenario 2 (S2): We examined the **warehouse-2-4-18-2-1.map** with an 18-grid corridor in a 39×13 grid warehouse and 56% (of total grids) obstacles. In Figs. 4(d), 4(e), and 4(f), we report the success rate, average computation time, and solution cost for all the successful cases (excluding the timeout cases), respectively.

Overall, the results reported in scenarios 1 and 2 show that the CFNRS-based solver is comparable or better in computation time, success rate, and solution cost than the MAPF solvers for the narrow-corridor environment.

B. Lifelong operations using CFNRS Algorithm

As in lifelong MAPF, robots receive new goal locations after reaching their current destinations; we perform 50 random initializations for each number of agents and report the average number of goals visited (tasks performed) in 900 s. The maximum time limit for recomputing the deadlock-free path using the CFNRS Algorithm is 60 s. The success rate (no timeout) is reported in Fig. 4(g), and the average total number of goals visited for successful cases is reported in Fig. 4(h). Clearly, the success rate in the classical problem does

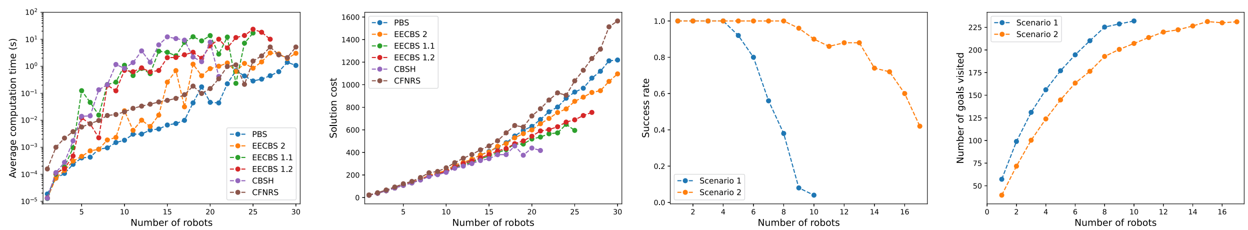


(a) S1: Success rate for 1 to 9 Agents

(b) S1: Computation time for successful cases

(c) S1: Solution cost for successful cases

(d) S2: Success rate for 1 to 30 Agents



(e) S2: Computation time for successful cases

(f) S2: Solution cost for successful cases

(g) Success rate

(h) Number of goals visited

Fig. 4. Scenario S1 and S2: Panels (a)-(f) used to compare various solvers in **warehouse-1-2-18-2-1.map** and **warehouse-2-4-18-2-1.map**. Comparison of goal visited and success rate in S1 and S2 for Lifelong operations using CFNRS Algorithm in Panels (g) and (h).

not ensure a high success rate for lifelong operations with random goal assignment.

V. Conclusions and Future Work

In this paper, we studied lifelong MAPF problems in a long and narrow corridor environment. We proposed a conflict-free Node-to-Robot Scheduling (CFNRS) algorithm for coordinating a robot fleet, integrating problem reduction, deadlock detection, and resolution techniques to ensure collision-free routes. We demonstrate the algorithm's efficacy through various examples and evaluate it against existing methods using simulated narrow-corridor warehouse scenarios. Our future work involves incorporating robot footprint constraints for industrial relevance and practical validation using real robot fleets in warehouse environments.

References

[1] R. Stern, N. R. Sturtevant, A. Felner, S. Koenig, H. Ma, T. T. Walker, J. Li, D. Atzmon, L. Cohen, T. S. Kumar, *et al.*, "Multi-agent pathfinding: Definitions, variants, and benchmarks," in *Twelfth Annual Symposium on Combinatorial Search*, 2019.

[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] S. Ardizzoni, I. Saccani, L. Consolini, and M. Locatelli, "Multi-agent path finding on strongly connected digraphs," in *2022 IEEE 61st Conference on Decision and Control (CDC)*, pp. 7194–7199, IEEE, 2022.

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

[6] J. Li, W. Ruml, and S. Koenig, "Eecbs: A bounded-suboptimal search for multi-agent path finding," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 35, pp. 12353–12362, 2021.

[7] H. Ma, D. Harabor, P. J. Stuckey, J. Li, and S. Koenig, "Searching with consistent prioritization for multi-agent path finding," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, pp. 7643–7650, 2019.

[8] J. Li, D. Harabor, P. J. Stuckey, H. Ma, G. Gange, and S. Koenig, "Pairwise symmetry reasoning for multi-agent path finding search," *Artificial Intelligence*, vol. 301, p. 103574, 2021.

[9] V. Nguyen, P. Obermeier, T. C. Son, T. Schaub, and W. Yeoh, "Generalized target assignment and path finding using answer set programming," in *Twelfth annual symposium on combinatorial search*, 2019.

[10] Q. Wan, C. Gu, S. Sun, M. Chen, H. Huang, and X. Jia, "Lifelong multi-agent path finding in a dynamic environment," in *2018 15th International Conference on Control, Automation, Robotics and Vision (ICARCV)*, pp. 875–882, IEEE, 2018.

[11] 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.

[12] J. Yu and S. M. LaValle, "Structure and intractability of optimal multi-robot path planning on graphs," in *Twenty-Seventh AAAI Conference on Artificial Intelligence*, 2013.

[13] L. Cohen, G. Wagner, D. Chan, H. Choset, N. Sturtevant, S. Koenig, and T. S. Kumar, "Rapid randomized restarts for multi-agent path finding solvers," in *Eleventh Annual Symposium on Combinatorial Search*, 2018.

[14] M. Peasgood, J. McPhee, and C. Clark, "Complete and scalable multi-robot planning in tunnel environments," *IFAC Proceedings Volumes*, vol. 39, no. 20, pp. 26–31, 2006.

[15] 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.