Scalable Online Coverage Path Planning for Multi-Robot Systems

Ratijit Mitra[1] and Indranil Saha[2]

Abstract— **Online coverage path planning to explore an unknown workspace with multiple homogeneous robots could be either *centralized* or *distributed*. While distributed planners are computationally faster, centralized planners can produce more efficient paths, reducing the duration of completing a coverage mission significantly. To exploit the power of a centralized framework, we propose a *receding horizon* centralized online multi-robot planner. In each *planning horizon*, it generates collision-free paths that guide the robots to visit some obstacle-free locations (aka *goals*) not visited so far, which in turn help them explore some new regions with their laser rangefinders. We formally prove that, under reasonable conditions, it enables the robots to cover a workspace *completely* and subsequently analyze its time complexity. We evaluate our planner for ground and aerial robots by performing experiments with up to 128 robots on six 2D grid-based benchmark obstacle maps, establishing scalability. We also perform Gazebo simulations with 10 quadcopters and real experiments with 2 four-wheel ground robots, demonstrating its practical feasibility. Furthermore, a comparison with a state-of-the-art distributed planner establishes its superiority in coverage completion time.**

I. Introduction

Multi-robot coverage path planning (CPP) deals with obtaining *safe* navigational paths for the robots so that they can visit obstacle-free workspace regions of interest exhaustively to carry out some given tasks. It has numerous applications that include precision agriculture [1], demining [2], search and rescue [3], surveying [4], to name a few. A CPP algorithm, which is often called Coverage Planner (CP), is said to be *complete* if it ensures that the entire obstacle-free region gets visited whenever feasible. A major objective while designing a CP is to minimize the *mission completion time*, where the mission is to achieve *complete coverage*. Often, multiple robots [5], [6], [7], [8], [9], [10], [11], [12], [13], [14] are used to improve the mission completion time, but at the cost of the additional complexity in algorithm design to deal with the robots effectively.

In the vast literature on CPP (see [15], [16], [17] for recent surveys), the workspace has been assumed to be either *known* [18], [4], [19], where the robots are fully aware of obstacle locations, or *unknown* [20], [9], [21], [2], [22], [14], [3], where they are not. In a known workspace, the paths can be obtained in one go (*offline* approach). In contrast, in an unknown workspace, the subpaths are obtained in successive *iterations* as more and more workspace regions get explored by the robots (*online* approach).

Based on the placement, a CP can be classified as either *centralized* or *distributed*. In a centralized framework (e.g. [9], [23], [14], [24], [25], [3], [26], [27], [28], [29], [30]), an instance of CP, solely responsible for obtaining the paths, resides in a server. However, in a distributed framework (e.g. [31], [32], [33], [34], [35], [10], [12], [36]), the robots are delegated to obtain their paths *collaboratively* as each robot runs an instance of the same CP locally. In this paper, we address the online multi-robot CPP problem. Thus, let us focus on the strengths and weaknesses of the existing online solutions that use any of the frameworks mentioned above. Tree-based centralized CPs for offline multi-robot coverage [28], [29] are easily extendable to online scenarios. Frontier-based centralized CPs [30], [27], [26], [25], however, are particularly designed for online multi-robot coverage. Although most centralized CPs ensure complete coverage, very few have addressed the scalability with many robots. Coming to the distributed CPs, several of them, like [31], [32], [34], [35], [36], ensure complete coverage with any number of robots, whereas [33] needs at least 4 robots to ensure the same. As distributed CPs involve local decision-making and are free from dealing with the global state space, they are often much more scalable than centralized CPs. However, the local decisions made by the distributed CPs hurt the overall efficiency of the multi-robot system. Unlike the centralized CPs, the local decisions lack global insights that can help distribute the robots' tasks optimally. Thus, an outstanding problem in online multi-robot CPP is designing a solution that retains the efficiency of the centralized CPs and scales like the distributed CPs.

This paper aims to design a scalable centralized solution for the online multi-robot CPP problem, which ensures complete coverage. We base our CP on the receding horizon strategy [3] as it promises scalability. Existing works on receding horizon online multi-robot CPP generate the paths either *synchronously* [14] or *asynchronously* [9]. The drawback of synchronous planning with a user-defined fixed horizon length, as done in [14], is that the robots that reach their goals earlier than others have to wait till the end of the horizon, leading to suboptimal usage of resources. On the other hand, asynchronous planning may reserve longer paths for some robots when other nearby robots to the reserved goals become available over time. Moreover, none of these papers guarantees complete coverage. We present a synchronous receding horizon online multi-robot CP that *dynamically* sets the paths' length for all the robots in each horizon. The CP assigns a goal to each robot and finds their collision-free paths based on the available information. However, each robot executes its plan till the robot with the shortest plan reaches its goal. In this way, we ensure that on each horizon, at least one goal is visited, and all the robots advance towards some goal. As replanning happens in the next horizon for all the robots, the goal assignment may change for some robots if some other robots can visit those goals sooner. Thus, our CP is highly dynamic and efficient in resource utilization.

For evaluation, we consider 6 different 2D grid-based benchmark workspaces and two different types of robots for their coverage. We compare our algorithm with the state-of-the-art distributed coverage path planning algorithm BoB [35], which outperforms the best known centralized coverage path planning algorithm MSTC [28]. The comparison reveals that our algorithm provides a significantly improved mission completion time for multi-robot systems with a large number

[1] Ratijit Mitra is with Department of Computer Science and Engineering, Indian Institute of Technology Kanpur `ratijit@cse.iitk.ac.in`
[2] Indranil Saha is with Department of Computer Science and Engineering, Indian Institute of Technology Kanpur `isaha@cse.iitk.ac.in`

of robots due to its capability of finding highly efficient plans for each robot.

II. Problem

A. Preliminaries

Let $\mathbb{R}$ and $\mathbb{N}$ denote the set of real numbers and natural numbers, respectively, and $\mathbb{N}_0$ denotes the set $\mathbb{N} \cup \{0\}$. Also, for $m \in \mathbb{N}$, we write $[m]$ to denote the set $\{n \in \mathbb{N} \mid n \leq m\}$, and $[m]_0$ to denote $[m] \cup \{0\}$. The size of the countable set $\mathcal{S}$ is denoted by $|\mathcal{S}| \in \mathbb{N}_0$. Furthermore, we denote the set of Boolean values $\{0, 1\}$ by $\mathbb{B}$.

1) Workspace Representation: We consider an unknown 2D workspace ($W = \{(x, y) \mid x \in [X] \wedge y \in [Y]\}$) with *stationary* obstacles, where $X,\ Y \in \mathbb{N}$ are its size along the x and the y axes, respectively. A workspace is decomposed into *non-overlapping* square grid cells, some of which are obstacle-free (W_{free}), traversable by robots, and the rest are fully obstacle-occupied (W_{obs}), which are not. In other words, $W_{free} \cap W_{obs} = \emptyset$ and $W_{free} \cup W_{obs} = W$.

2) Robots and Motion Primitives: We employ a team of $R \in \mathbb{N}$ *location-aware* and *homogeneous* robots where we assume that they have unlimited battery capacity. Each robot is fitted with 4 laser rangefinders on all four sides to detect obstacles in W as per the field of view and can communicate *reliably* over a network. Moreover, each robot fits entirely within a grid cell. Let the i ($\in [R]$)-th robot be denoted by r^i. The state of r^i at the $j (\in \mathbb{N}_0)$-th *discrete* time step is denoted by s^i_j which is a tuple of its location and possibly orientation. The robots have a set of *motion primitives* (M) to change their current state at every time step. We assume that M contains a special motion primitive $\mathtt{H}$ (*Halt*) to keep any current state unchanged in the next time step. Each motion primitive $\mu \in M$ has some cost $\mathtt{cost}(\mu) \in \mathbb{R}$ (e.g., distance traversed, energy consumed, etc.) associated with it. All the motion primitives take the same $\tau \in \mathbb{N}$ unit time for execution. Let $\mathcal{L}(s^i_j)$ denote a tuple containing only the location component of s^i_j. Initially, the robots are *randomly* deployed in different cells of W_{free}, comprising their start states S, where $S = \{s^i_0 \mid i \in [R] \wedge \mathcal{L}(s^i_0) \in W_{free} \wedge \forall j \in [R] \setminus \{i\}\ (\mathcal{L}(s^i_0) \neq \mathcal{L}(s^j_0))\}$.

Example 1: To illustrate our algorithm, we consider the following two different sets of motion primitives of robots - one for the aerial robots and the other for the ground robots.

(i) *Quadcopter 2D.* We consider the 2D movements of a quadcopter with state $s^i_j = (x^i_j, y^i_j) \in \mathbb{N}^2$ which is its location in W. The set of motion primitives is given by $M = \{\mathtt{H}, \mathtt{MT}, \mathtt{MB}, \mathtt{MR}, \mathtt{ML}\}$ where $\mathtt{H}$ keeps a quadcopter in its current cell, and $\mathtt{MT}$, $\mathtt{MB}$, $\mathtt{MR}$, $\mathtt{ML}$ *move* it to its immediate *top, bottom, right*, and *left* cells, respectively, in the next time step.

(ii) *Turtlebot.* A Turtlebot [37], a standard differential drive robot widely used for academic research, can move in the forward direction and can rotate around its axis. In this case, keeping track of the orientation of the robot is essential. Thus, the state of the robot is denoted by $s^i_j = (x^i_j, y^i_j, \theta^i_j)$, where $(x^i_j, y^i_j) \in \mathbb{N}^2$ is its location in W and $\theta^i_j \in \{0\ (\mathtt{East}), 1\ (\mathtt{North}), 2\ (\mathtt{West}), 3\ (\mathtt{South})\}$ denotes its direction at the j-th time step w.r.t. the positive x axis. The set of motion primitives for this robot is given by $M = \{\mathtt{H}, \mathtt{TL}, \mathtt{TR}, \mathtt{MN}\}$, where $\mathtt{H}$ keeps its current state unchanged. Motion primitives $\mathtt{TL}$ and $\mathtt{TR}$ *turn* it 90° to its *left* and *right*, respectively, while keeping it in its current cell. Motion primitive $\mathtt{MN}$ *moves* it to the *next* cell in the direction pointed by its current orientation θ^i_j.

3) Path of a Robot: The path π^i of r^i is a *finite* sequence of states of r^i, denoted by $(s^i_j)_{j \in [\Lambda]_0}$, where the length of π^i is denoted by $\Lambda \in \mathbb{N}_0$, which is equal for all the robots. This is not an unrealistic assumption, as the paths of the robots can be made equal by applying $\mathtt{H}$ at the end appropriately. The path π^i gets generated when a *finite* sequence of motion primitives $(\mu^i_j \in M)_{j \in [\Lambda]}$ is applied on s^i_0 s.t.

$$s^i_j \xrightarrow{\mu^i_{j+1}} s^i_{j+1},\ \forall j \in [\Lambda - 1]_0.$$

B. Problem definition

Before defining the Coverage Path Planning (CPP) problem, we formally define *collision-free paths* of the robots and *complete coverage* of W as follows:

Definition 2.1 (Collision-free Paths of the Robots): A set of robot paths ($\Pi = \{\pi^i \mid i \in [R]\}$) is *collision-free* if the following three conditions hold for each π^i:

1) $\forall j \in [\Lambda]_0\ \mathcal{L}(s^i_j) \in W_{free}$ [*Avoids obstacles*]
2) $\forall j \in [\Lambda]_0\ \forall k \in [R] \setminus \{i\}$ $(\mathcal{L}(s^i_j) \neq \mathcal{L}(s^k_j))$ [*Averts same cell collisions*]
3) $\forall j \in [\Lambda]\ \forall k \in [R] \setminus \{i\}$ $\neg((\mathcal{L}(s^i_{j-1}) = \mathcal{L}(s^k_j)) \wedge (\mathcal{L}(s^i_j) = \mathcal{L}(s^k_{j-1})))$ [*Averts head-on collisions*]

Definition 2.2 (Complete Coverage of W): A set of robot paths Π *completely covers* a workspace $W = \langle W_{free}, W_{obs} \rangle$ if every obstacle-free cell gets visited by at least one robot, i.e., $\bigcup_{i \in [R]} \bigcup_{j \in [\Lambda]_0} \{\mathcal{L}(s^i_j)\} = W_{free}$.

Finally, we define the CPP problem formally as follows:

Definition 2.3 (Coverage Path Planning Problem): Given a workspace $W = \langle W_{free}, W_{obs} \rangle$, a set of R robots, their initial states S, and the set of motion primitives M, find the collision-free paths Π of the robots that ensure complete coverage of W. Mathematically, the CPP problem can be defined as a function $\mathcal{P}_{CPP}$ s.t.

$$\Pi = \mathcal{P}_{CPP}(W, R, S, M). \tag{II.1}$$

To evaluate the performance of a CP, we choose the *mission time* (T_m) as a metric, which is the duration from the beginning till the CP attains complete coverage. Any CP aims to minimize T_m. Observe the trade-off between the *total computation time* (T_c) and the *total path execution time* (T_p), spent at the planner's end and the robots' end, respectively. A CP can spend a large T_c to generate highly efficient paths, which reduces T_p, or spend a small T_c to generate relatively inefficient paths, which increases T_p.

III. Coverage Algorithm

In this section, we present our coverage path planning algorithm. We deal with an unknown workspace for which only the boundary and the size are known to the coverage planner, but the locations of the obstacles are not known a priori. The planner divides the exploration into multiple horizons. In each horizon, due to *partial* visibility of W, the workspace cells are classified as follows: (i) *Unexplored* (W_u): yet to get explored by the robots, (ii) *Obstacle* (W_o): explored and found occupied with obstacles, (iii) *Goal* (W_g): explored, found obstacle free, and yet to get visited by the robots, and (iv) *Covered* (W_c): already visited by the robots. In other words, W_u, W_o, W_g, and W_c are *pairwise disjoint*, and $W = W_u \cup W_o \cup W_g \cup W_c$.

Our overall coverage methodology, named $\mathtt{GAMRCPP}$ (stands for **G**oal **A**ssignment-based **M**ulti-**R**obot **CPP**), is shown in Algorithm 1. In each horizon, the

Algorithm 1: GAMRCPP ($W = (W_u, W_o, W_g, W_c), R, S, M$)

```
   Result: Paths of the robots $(\Pi)$
1  $\Pi \leftarrow \emptyset$
2  while $W_g \neq \emptyset$ do
3      $\Sigma \leftarrow$ GAMRCPP_Horizon$(W, R, S, M)$        // Algo. 2
4      parallel for $i \in [R]$ do
5          $W^i \leftarrow$ send_path_and_get_local_view$(\sigma^i)$
6          $\pi^i \leftarrow \pi^i : \sigma^i$                  // Concatenate
7      $W \leftarrow$ update_workspace$(\{W^i \mid i \in [R]\})$
8      $S \leftarrow$ update_init_states$(\Sigma)$
```

GAMRCPP_Horizon function in Algorithm 2 is invoked (line 3) to compute the paths for the robots based on the current view of the workspace W and the current states of the robots S. The generated paths ($\Sigma = \{\sigma^i \mid i \in [R]\}$) are *simultaneously* sent to the robots which follow them *synchronously* to cover the cells on the paths (lines 4-5). While following the paths, the robots use their laser rangefinders to detect whether an obstacle occupies any cell newly visible to them. Consequently, each robot r^i updates its *local view* $W^i = (W^i_u, W^i_o, W^i_g, W^i_c)$ of the workspace W, where W^i_u, W^i_o, W^i_g, and W^i_c are the sets of unexplored cells, obstacle cells, goal cells, and the covered cells according to r^i. Upon finishing, the robots directly send their local views to the planner (line 5) so that the planner can generate the *global view* of W (line 7) as follows:

$$W_c = \bigcup_{i \in [R]} W^i_c, \qquad W_g = \Big(\bigcup_{i \in [R]} W^i_g\Big) \setminus W_c,$$
$$W_o = \Big(\bigcup_{i \in [R]} W^i_o\Big) \setminus W_c, \quad W_u = W \setminus (W_c \cup W_g \cup W_o).$$

Note that a robot may appear as an obstacle to another robot (e.g., when two robots are on two neighboring cells) as it cannot distinguish between obstacles and other robots. But, the planner can while generating the global view. Finally, the initial states of the robots for the next horizon are set to their final states in the current horizon (line 8). The entire procedure is repeated in a **while** loop (lines 2-8) if there is at least one goal left to be visited.

A. *Computing Collision-Free Paths in Each Horizon*

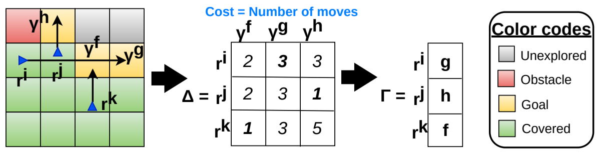
In each horizon, the problem of optimally visiting W_g by R robots is a *Multiple Traveling Salesman Problem* [38], which is *NP-hard*. Thus, solving the problem with large R and/or $G = |W_g|$ is computationally challenging. So, we propose a goal assignment-based method that generates collision-free paths, but without guaranteeing optimality.

GAMRCPP_Horizon binds the robots to distinct goals in a cost-optimal manner (lines 1-2) and then generates their paths that avoid $W_u \cup W_o$ (line 3). Next, if needed, the paths are adjusted to avert inter-robot collisions (line 4), and accordingly, the *horizon length* λ gets determined *dynamically* (line 5). Finally, all the paths are made of equal length λ (line 6), and returned to GAMRCPP. We now describe the steps of GAMRCPP_Horizon in further detail.

1) Optimal costs (Δ)*:* First, we construct a *state transition graph* $\mathcal{G}_\Delta = (V, E)$, where V denotes all possible states of the robots such that $\forall u \in V$, $\mathcal{L}(u) \in W_g \cup W_c$. There exists a *directed* transition edge $e \in E$ from state $u \in V$ to state $v \in V$ if it is possible to reach state v by applying a motion primitive $\mu \in M$ to state u. The weight of the edge e is equal to $\mathtt{cost}(\mu)$. Let $V_{goals} \subseteq V$ be the set

Algorithm 2: GAMRCPP_Horizon (W, R, S, M)

```
   Result: Equal length collision-free paths $(\Sigma)$
1  $\langle \Delta, L_\Delta \rangle \leftarrow$ compute_optimal_costs$(W, S, M)$
2  $\Gamma \leftarrow$ compute_optimal_assignments$(\Delta)$
3  $\Phi \leftarrow$ get_optimal_paths$(L_\Delta, \Gamma)$
4  $\Sigma' \leftarrow$ GAMRCPP_CFP$(R, \Gamma, \Phi)$              // Algo. 3
5  $\lambda \leftarrow$ compute_horizon_length$(\Sigma')$
6  $\Sigma \leftarrow$ get_equal_length_paths$(\Sigma', \lambda)$
```



	y^f	y^g	y^h
r^i	2	**3**	3
r^j	2	3	**1**
r^k	**1**	3	5

r^i	g
r^j	h
r^k	f

Fig. 1: Cost optimal assignments of the goals to the robots

of states that are associated with goals, i.e., $\forall u' \in V_{goals}$, $\mathcal{L}(u') \in W_g$. Now, for each r^i, we compute the optimal costs of reaching every $u' \in V_{goals}$ from the start state s^i_0 using the A* algorithm [39], [40]. We capture the costs in a matrix $\Delta \in \mathbb{R}^{R \times G}$, where $\Delta[i][j]$ stores the optimal cost for robot r^i to reach the j $(\in [G])$-th goal, denoted by $\gamma^j \in W_g$. In addition, while computing $\Delta[i][j]$, we store the corresponding *predecessor information* of states into a list L_Δ, which is later queried during the generation of optimal paths. Note that L_Δ takes $\mathcal{O}(|E| \cdot G \cdot R)$ of space.

2) Optimal assignments (Γ)*:* Each robot now needs to be assigned a *distinct* goal such that the total path cost of all the robots is optimal. To achieve this, first we construct a weighted bipartite graph $\mathcal{G}_\Gamma = ([R], [G], \Delta)$, whose edge weights are the optimal path costs. Then, we run the *Hungarian algorithm* [41], [42] on $\mathcal{G}_\Gamma$ to solve this combinatorial optimization problem. It produces a goal assignment given by $\Gamma \colon [R] \to [G] \cup \{\mathsf{NULL}\}$. Note that a robot r^i may not be assigned any goal in a horizon. In that case, its assigned goal id, denoted by $\Gamma[i]$, contains NULL, and it is called an *inactive* robot.

Example 2: An example of goal assignment is shown in Figure 1. In this example and all subsequent examples, we consider Turtlebots and the cost to be the number of moves, including turns.

3) Optimal paths (Φ)*:* For each *active* r^i, its optimal cost path φ^i gets generated from L_Δ. Such a φ^i originates from s^i_0 and passes through $W_g \cup W_c$, before ending at the assigned goal $\gamma^{\Gamma[i]}$. The path φ^j of an *inactive* robot r^j, however, contains s^j_0 only. The paths are collectively captured in $\Phi = \{\varphi^i \mid i \in [R]\}$, where φ^i can be mathematically characterized as follows:

$$\varphi^i = \begin{cases} s^i_0, & \text{if } \Gamma[i] = \mathsf{NULL} \\ s^i_0 s^i_1 \cdots s^i_{|\varphi^i|} \text{ s.t. } \mathcal{L}(s^i_{|\varphi^i|}) = \gamma^{\Gamma[i]}, & \text{otherwise.} \end{cases} \tag{III.1}$$

where $|\varphi^i|$ denotes the length of φ^i.

4) Collision Free Paths (Σ')*:* All the paths of Φ may not be feasible for the robots to traverse due to inter-robot collisions at cells or cell borders. In that case, the paths need some adjustments to make them collision-free. So, we invoke GAMRCPP_CFP (Algorithm 3), which returns the set

Algorithm 3: GAMRCPP_CFP (R, Γ, Φ)

```
Result: Collision free paths of the robots ($\Sigma'$)
1  while true do
2      $\langle \Gamma, \Omega \rangle \leftarrow$ get_feasible_paths($R, \Gamma, \Phi$)  // Algo.4
3      $\Theta_r \leftarrow$ compute_relative_precedences($\Omega$)
4      $\Theta_a \leftarrow$ compute_absolute_precedence($\Theta_r$)
5      if $\Theta_a$ is valid then
6          break
7      else
8          $\Gamma \leftarrow$ break_precedence_cycles($\Gamma, \Theta_r$)
9          $\Phi \leftarrow$ adjust_paths($\Gamma, \Omega$)
10 $\Upsilon \leftarrow$ compute_start_time_offsets($\Omega, \Theta_a$)
11 $\Sigma' \leftarrow$ get_collision_free_paths($\Omega, \Upsilon$)
```

of collision-free paths ($\Sigma' = \{\sigma'^i \mid i \in [R]\}$). We defer the description of GAMRCPP_CFP to the following Section III-B.

5) Computing the horizon length (λ)*:* The paths of Σ' could be of different lengths. So, we determine the length of the horizon as the minimum length of paths of the active robots, i.e., $\lambda = \min_{i \in [R]} \{|\sigma'^i| > 0\}$. Had we chosen max instead of min, it could have made most of the active robots *idle* once they reach their respective goals.

6) Equal Length Paths (Σ)*:* First, we initialize $\Sigma = \{\sigma^i \mid i \in [R]\}$ to Σ'. Notice that $|\sigma'^i| < \lambda$ for an inactive r^i. So, we insert its s_0^i at the end of σ^i for $\lambda - |\sigma'^i|$ times. Similarly, for an active r^j whose $|\sigma'^j| > \lambda$, we trim σ^j from the end for $|\sigma'^j| - \lambda$ times. Thus, we make collision-free paths of equal length λ.

B. Inter-robot Collision Avoidance

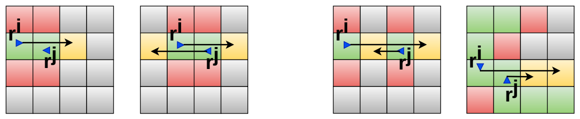
Prioritized planning (e.g., [43], [44], [45], [46]), which first assigns priorities to the mobile robots and then plans their collision-free paths one by one in the order, has been widely used for collision avoidance. In [46], a cost-optimal goal assignment method precedes the prioritized planning, where the cost metric did not consider *turns* because it deals with only aerial robots having symmetrical motion in any direction. But, [13] has recently shown that the number of *turns* taken by ground robots influences the mission time significantly. We take a similar approach of [46] in GAMRCPP_CFP, but with the difference that our Δ can also consider *turns* depending on the type of robots used. As a result, two types of infeasible paths may appear that prioritized planning fails to handle. We first identify those infeasible paths and then adjust them (line 2) so that prioritized planning can handle them. Subsequently, priorities are dynamically computed (lines 3-4) based on their movement constraints. Finally, time-parameterized collision-free paths are computed (lines 10-11). Note that cycles in priority need to be broken (lines 8-9) if they appear.

1) Removal of infeasible paths (Ω)*:* There are two types of infeasible paths in Φ - *crossover paths* and *nested paths*, which are defined later by using the following function:

$$\texttt{in\_path}(s, p) = \begin{cases} 1, & \exists k \in [|p|]_0 \text{ s.t. } \mathcal{L}(s) = \mathcal{L}(p[k]) \\ 0, & \text{otherwise.} \end{cases} \tag{III.2}$$

The function in_path takes a robot state s and a path p as inputs, and outputs 1 only if p passes through the location corresponding to s. Here, $p[k]$ denotes the k-th state in p. Now,

- A pair of distinct paths (φ^i, φ^j) is called a *crossover path pair iff* either of the following two conditions hold:



(a) Crossover paths (b) Nested paths

Fig. 2: Infeasible paths

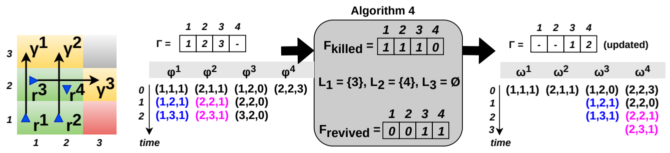


Fig. 3: Removal of infeasible paths

  Cond. i: Inactive r^i sits on the path φ^j of active r^j, i.e., $\texttt{in\_path}(s_0^i, \varphi^j)$.
  Cond. ii: The start locations of the active robots r^i and r^j are on each other's path φ^j and φ^i, respectively, i.e., $\texttt{in\_path}(s_0^i, \varphi^j) \wedge \texttt{in\_path}(s_0^j, \varphi^i)$.
- A path φ^j of active r^j is said to be *nested* in another path φ^i of active r^i *iff* the following condition holds:
  Cond. iii: Both the start and the goal locations of r^j are on φ^i, i.e., $\texttt{in\_path}(s_0^j, \varphi^i) \wedge \texttt{in\_path}(s_{|\varphi^j|}^j, \varphi^i)$.

Example 3: Four simple examples of infeasible paths involving two robots r^i and r^j are shown in Figure 2.

We design Algorithm 4 to make some infeasible paths of Φ feasible by adjusting them *locally* through goal reassignments. It is a two-step procedure, where the first step (lines 5-22) detects infeasible paths of the active robots and subsequently marks corresponding robots as killed in $\mathcal{F}_{killed}$, and the last step (lines 23-48) revives some of the inactive robots and the killed robots in $\mathcal{F}_{revived}$ so that some goals of the killed robots can be visited. While lines 8-12 kill active robot pairs that form either crossover paths or nested paths, lines 13-17 kill active robots that form crossover paths with either an inactive robot or a killed robot. Note that any active robot index i can be inserted into Ξ at most once as $\mathcal{F}_{killed}[i]$ gets set as soon as r^i is killed for the first time (lines 19-22).

In the revival step, inactive robots and killed robots are not assigned any goal initially (lines 28-30), whereas the paths of the active robots, which did not get killed, remain unaltered (lines 31-33). Now, for each killed robot i, its L_i (line 36) contains indices of inactive robots and killed robots, which are on its path φ^i, and are yet to get revived. In case L_i turns out to be empty, we do not assign the goal of r^i to any other robot. Otherwise, we choose the robot having index $j \in L_i$ such that r^j is the nearest to the goal of r^i (line 38), thus eliminating the chance of further forming any crossover path with inactive/killed robots. Then, if $j \neq i$, the potential path ω of r^j is generated from φ^i, otherwise φ^i is used (lines 39-42). Now, ω needs to be checked to see whether it forms any infeasible path with the paths of the revived robots. If it does not, then r^j is set to visit the goal of r^i by following its feasible path ω^j, thereby reviving r^j (lines 43-46). Finally, Γ gets updated, and $\Omega = \{\omega^i \mid i \in [R]\}$ is returned (lines 47-48).

Example 4: We show an example in Figure 3, where not all optimal paths of Φ, computed as per Γ, are feasible. In fact, the Hungarian algorithm can find such Γ because it only acts on Δ without knowing underlying paths. So, we use

Algorithm 4: Remove infeasible paths

```
1  Function get_feasible_paths($R, \Gamma, \Phi$)
2      $\mathcal{F}_{killed} \leftarrow$ kill_robots($R, \Gamma, \Phi$)
3      $\Omega \leftarrow$ revive_robots($R, \Gamma, \Phi, \mathcal{F}_{killed}$)
4      return $\Omega$
5  Function kill_robots($R, \Gamma, \Phi$)
6      $\Xi \leftarrow \{i \mid \Gamma[i] = \text{NULL}\}$                // Temp. set
7      $\mathcal{F}_{killed}[i] = 0,\ \forall i \in [R]$               // Flag array
8      for $i \leftarrow 1$ to $R$ do
9          for $j \leftarrow 1$ to $R$ do
10             if $i \neq j \wedge \Gamma[i] \neq \text{NULL} \wedge \Gamma[j] \neq$
               $\text{NULL} \wedge$ (Cond. ii || Cond. iii) then
11                 insert_item($i, \Xi, \mathcal{F}_{killed}$)
12                 insert_item($j, \Xi, \mathcal{F}_{killed}$)
13     while $\Xi \neq \emptyset$ do
14         $i \leftarrow$ extract_item($\Xi$)               // $\Xi \leftarrow \Xi \setminus \{i\}$
15         for $j \leftarrow 1$ to $R$ do
16             if $i \neq j \wedge \Gamma[j] \neq \text{NULL} \wedge$ Cond. i then
17                 insert_item($j, \Xi, \mathcal{F}_{killed}$)
18     return $\mathcal{F}_{killed}$
19 Function insert_item($i, \Xi, \mathcal{F}_{killed}$)
20     if $\mathcal{F}_{killed}[i] = 0$ then
21         $\mathcal{F}_{killed}[i] \leftarrow 1$                        // Kills $r^i$
22         $\Xi \leftarrow \Xi \cup \{i\}$
23 Function revive_robots($R, \Gamma, \Phi, \mathcal{F}_{killed}$)
24     $\Gamma_{new}$          // New goal assignment array
25     $\Omega \leftarrow \emptyset$          // The set of feasible paths
26     $\mathcal{F}_{revived}[i] = 0,\ \forall i \in [R]$             // Flag array
27     for $i \leftarrow 1$ to $R$ do
28         if $(\Gamma[i] = \text{NULL})\ \|\ (\mathcal{F}_{killed}[i] = 1)$ then
29             $\Gamma_{new}[i] \leftarrow \text{NULL}$               // Initialized
30             $\omega^i \leftarrow s^i_0$                            // $\omega^i \in \Omega$
31         else
32             $\Gamma_{new}[i] \leftarrow \Gamma[i]$                 // Unaltered
33             $\omega^i \leftarrow \varphi^i$
34     for $i \leftarrow 1$ to $R$ do
35         if $\mathcal{F}_{killed}[i] = 1$ then
36             $L_i \leftarrow \{j \mid (\Gamma[j] = \text{NULL}\ \|\ \mathcal{F}_{killed}[j] = 1)$
               $\wedge$ in_path($s^j_0, \varphi^i$) $\wedge\ \mathcal{F}_{revived}[j] = 0\}$
37             if $L_i \neq \emptyset$ then
38                 $j \leftarrow \underset{l\ \in\ L_i}{\arg\max}\ k \in [|\varphi^i|]_0 \mid \mathcal{L}(s^l_0) = \mathcal{L}(s^i_k)$
39                 if $j = i$ then
40                     $\omega \leftarrow \varphi^i$          // Potential path
41                 else
42                     $\omega \leftarrow$ generate_path($s^j_0, \varphi^i$)
43                 if test_path($j, \omega, \Gamma_{new}, \Omega$) then
44                     $\mathcal{F}_{revived}[j] \leftarrow 1$          // Revived
45                     $\Gamma_{new}[j] \leftarrow \Gamma[i]$          // New goal
46                     $\omega^j \leftarrow \omega$                    // New path
47     $\Gamma \leftarrow \Gamma_{new}$                             // Itemwise copy
48     return $\Omega$
```

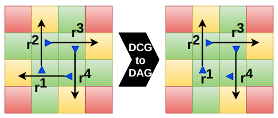


Fig. 4: Removal of cyclic precedence

Algorithm 4 to get feasible paths. Notice that the number of revived robots is less than the number of killed robots. Although this is a weakness of Algorithm 4, we formally prove in Theorem 4.2 that it ensures the revival of at least one inactive or killed robot. Moreover, we evaluate its strength in terms of the percentage of robots that get revived w.r.t. the number of killed robots, and goals of the killed robots that the revived robots visit (Figure 5). At first glance, it may look like an ordering among the start locations can help Algorithm 4 to get rid of the weakness, but Figure 2a (right) suggests that the ordering may not be always possible. Also notice that one extra *turn* got introduced while generating ω^4 from φ^2. An alternative to Algorithm 4 is to keep finding the next cost optimal assignment Γ, until it guarantees no infeasible path, as done in [47]. But, this is computationally expensive.

2) Relative precedence (Θ_r)*:* Based on Ω, $\Theta_r \in \mathbb{B}^{R\times R}$ is populated, that captures relative precedence for every pair of distinct robots r^i and r^j having paths ω^i and ω^j, respectively, using the following equation:

$$\Theta_r[i][j] = \begin{cases} 1, & \text{if } \texttt{in\_path}(s^i_0, \omega^j)\ \|\ \texttt{in\_path}(s^j_{|\omega^j|}, \omega^i) \\ 0, & \text{otherwise.} \end{cases} \tag{III.3}$$

The *if* clause states that if the start location of r^i is on ω^j, then r^i must depart from its start location before r^j reaches there. Similarly, if the goal location of r^j is on ω^i, then r^i must pass r^j's goal location before r^j arrives there. Likewise, $\Theta_r[j][i]$ is computed. Essentially, distinct robots of each pair are assigned relative priorities to avert mutual collisions.

Example 5: For the example in Figure 1, relative precedences are found to be $\Theta_r[j][i] = 1$ and $\Theta_r[i][k] = 1$.

3) Absolute precedence (Θ_a)*:* From the directed graph $\mathcal{G}_\Theta = ([R], \Theta_r)$, we now find $\Theta_a \in [R]^{R\times 1}$ among the robots using the *topological sort*. As the presence of cyclic precedence leads to *invalid* Θ_a, such cycles need to be broken by inactivating suitable robots before proceeding further. In such cases, we would like to remove a minimum feedback arc set (MFAS) [48] from $\mathcal{G}_\Theta$. However, this is an *NP-hard* problem. So, we employ a simple algorithm [49] to generate two acyclic subgraphs of $\mathcal{G}_\Theta$. We choose the subgraph with more edges for the next iteration, i.e., the subgraph having more active robots. As inactivating an active robot may introduce a crossover path pair, the resultant paths Φ are re-examined in a **while** loop (lines 1-10) for the existence of infeasible paths.

Example 6: For the example in Figure 1, the absolute precedence is found to be j, i, k.

Example 7: In Figure 4, we provide an example of cyclic precedences involving 4 robots, where $\Theta_r[1][4] = \Theta_r[4][3] = \Theta_r[3][2] = \Theta_r[2][1] = 1$ forms a cycle. Notice that r^4, which was earlier active, has become inactive, creating a crossover path.

4) Start-time offsets (Υ)*:* As Θ_a is valid at this stage, the penultimate step is to *incrementally* calculate $\Upsilon \in \mathbb{N}_0^{R\times 1}$ for the robots starting from the highest priority robot to the lowest one. The start-time offset of r^i, denoted by $\Upsilon[i]$, determines how long the corresponding robot needs to *halt* at its start location to avert collisions with the robots having higher priorities.

Example 8: For the example in Figure 1, $\Upsilon[j] = \Upsilon[i] = 0$ as both r^j and r^i can start moving simultaneously at the 0-th time step without colliding with each other. However, $\Upsilon[k] = 2$ as any value less than that would put r^k in ω^i, thereby causing an inter-robot collision.

5) Collision free paths (Σ')*:* First, the resultant $\Sigma' = \{\sigma'^i | i \in [R]\}$ is initialized to Ω. Then, for each r^i, its start state, i.e. s^i_0, is inserted at the beginning of σ'^i for $\Upsilon[i]$ times. So, the length of σ'^i becomes $\Upsilon[i] + |\omega^i|$.

Example 9: For the example in Figure 1, $|\omega^i|, |\omega^j|, |\omega^k|$ are $3, 1, 1$, respectively, but $|\sigma'^i|, |\sigma'^j|, |\sigma'^k|$ are $3, 1, 3$, respectively. Also, active r^j's path σ'^j has the minimum length.

IV. Theoretical Analysis

In this section, first, we formally prove that Algorithm 4 is correct. Then, we formally prove that GAMRCPP ensures complete coverage of W assuming (i) W_{free} is *strongly connected* and (ii) the robots are failure-free. Finally, we analyze its time complexity.

A. Correctness of Algorithm 4

Lemma 4.1: The potential path ω of any r^j does not form any infeasible path with the active robots that are not killed.

Proof: Let us assume r^k be an active robot that is not killed, i.e., $k \in \{p \mid \Gamma[p] \neq \mathsf{NULL} \wedge \mathcal{F}_{killed}[p] = 0\}$.
First, if ω forms a crossover path pair with ω^k,
$\Rightarrow$ `in_path`$(s_0^j, \omega^k) \wedge$ `in_path`(s_0^k, ω) (by **Cond. ii**)
$\Rightarrow$ `in_path`$(s_0^j, \varphi^k) \wedge$ `in_path`(s_0^k, ω) (by line 33)
$\Rightarrow k \in \{p \mid \Gamma[p] \neq \mathsf{NULL} \wedge \mathcal{F}_{killed}[p] = 1\}$ (by lines 16-17) - contradiction.
Next, if ω is nested in ω^k,
$\Rightarrow$ `in_path`$(s_0^j, \omega^k) \wedge$ `in_path`$(s_{|\omega|}^j, \omega^k)$ (by **Cond. iii**)
$\Rightarrow$ `in_path`$(s_0^j, \varphi^k) \wedge$ `in_path`$(s_{|\omega|}^j, \varphi^k)$ (by line 33)
$\Rightarrow k \in \{p \mid \Gamma[p] \neq \mathsf{NULL} \wedge \mathcal{F}_{killed}[p] = 1\}$ (by lines 16-17) - contradiction.
Finally, if ω^k is nested in ω,
$\Rightarrow$ `in_path`$(s_0^k, \omega) \wedge$ `in_path`$(s_{|\omega^k|}^k, \omega)$ (by **Cond. iii**)
$\Rightarrow$ `in_path`$(s_0^k, \omega) \wedge$ `in_path`$(s_{|\varphi^k|}^k, \omega)$ (by line 33)
$\Rightarrow$ `in_path`$(s_0^k, \varphi^i) \wedge$ `in_path`$(s_{|\varphi^k|}^k, \varphi^i)$ (by line 42)
$\Rightarrow k \in \{p \mid \Gamma[p] \neq \mathsf{NULL} \wedge \mathcal{F}_{killed}[p] = 1\}$ (by lines 10-12) - contradiction. ■

Theorem 4.2: At least one inactive/killed robot gets revived if at least one active robot gets killed.

Proof: No active robot gets killed means Φ is feasible. Otherwise, for the first killed robot r^i (line 35), $L_i \neq \emptyset$, and $j \neq i$ because of lines 10-12 and 16-17. Moreover, there is no revived robot. So, the check (line 43) succeeds trivially (Lemma 4.1), and therefore r^j gets revived. ■

B. Proof of Complete Coverage

Lemma 4.3: GAMRCPP_Horizon ensures that at least one goal gets visited in each horizon.

Proof: In each horizon, after Γ gets initialized (line 2 in GAMRCPP_Horizon), it may get updated in GAMRCPP_CFP while computing Ω (line 2) or breaking precedence cycles in $\mathcal{G}_\Theta$ (line 8) to settle collisions. Theorem 4.2 guarantees that there is at least one active robot after computing Ω. Moreover, the technique [48], [49] used to break precedence cycles in $\mathcal{G}_\Theta$ ensures that *at least half as many edges as optimum* are preserved. Here, each preserved edge corresponds to an active robot. So, GAMRCPP_CFP keeps at least one robot active. Further, computed horizon length λ ensures that at least one active robot visits its goal. ■

Theorem 4.4: GAMRCPP eventually stops, and when it stops, it ensures complete coverage of W.

Proof: Lemma 4.3 ensures that at least one goal of W_g gets covered in each horizon, increasing W_c. It results in exploring some of W_u as W_{free} is strongly connected, and thus, adding new goals (if any) into W_g. Hence, GAMRCPP eventually stops when $W_g = \emptyset$, that entails $W_c = W_{free}$. ■

Notice that, for a given W whose W_{free} consists of many strongly connected *components*, GAMRCPP can still guarantee complete coverage, provided the deployment of robots ensures at least one robot in each component.

C. Time Complexity

Lemma 4.5: Algorithm 4 takes $\mathcal{O}(R^2)$ time.

Proof: `insert_item` (lines 19-22) runs in $\mathcal{O}(1)$. In `kill_robots` (lines 5-18), initializations of Ξ and $\mathcal{F}_{killed}$ take $\mathcal{O}(R)$. Subsequently, both the **for** and the **while** loops (lines 8-17) take $\mathcal{O}(R^2)$ as each robot index can be inserted into, and therefore extracted from Ξ *at most* once. Now, in `revive_robots` (lines 23-48), initialization of $\mathcal{F}_{revived}$, and execution of the following **for** loop (lines 27-33) take $\mathcal{O}(R)$. In the next **for** loop (lines 34-46), computing L_i (line 36) and j (line 38) take $\mathcal{O}(R)$ as $|L_i| = \mathcal{O}(R)$. Checking feasibility of any potential path ω (line 43) takes $\mathcal{O}(R)$ too. Therefore, `get_feasible_paths()` overall takes $\mathcal{O}(R^2)$. ■

Theorem 4.6: GAMRCPP takes $\mathcal{O}(|W|^4)$ time.

Proof: First, we analyze GAMRCPP_Horizon, which gets invoked from GAMRCPP in each horizon. Creating $\mathcal{G}_\Delta$ takes $\mathcal{O}(|V|)$ under the assumption that $|M|$ is constant, and running the A* algorithm on this for a robot and goal pair takes $\mathcal{O}(|E|)$. So, computing Δ and L_Δ takes, overall $\mathcal{O}(|V| + |E| \cdot G \cdot R)$. Computing Γ takes $\mathcal{O}(\max(G, R)^3)$ [41]. Accordingly, generating Φ takes $\mathcal{O}(|E| \cdot R)$. In GAMRCPP_CFP, the body of the **while** loop takes $\mathcal{O}(R^2)$ as follows: computing Ω (Lemma 4.5), Θ_r, Θ_a take $\mathcal{O}(R^2)$, additionally, breaking precedence cycles and adjusting paths of inactive robots take $\mathcal{O}(R^2)$ and $\mathcal{O}(R)$, respectively. As per Lemma 4.3, in a horizon, this **while** loop iterates at most $R - 1$ times inactivating one robot in each iteration. Therefore, this **while** loop takes total $\mathcal{O}(R^3)$. Consequently, computing Υ and Σ' take $\mathcal{O}(R^2)$ and $\mathcal{O}(R)$, respectively. So, GAMRCPP_CFP takes $\mathcal{O}(R^3)$. Finally, computing λ and Σ take $\mathcal{O}(R)$. So, GAMRCPP_Horizon takes $\mathcal{O}(|V| + |E| \cdot G \cdot R + \max(G, R)^3 + R^3)$ where $|V| = \mathcal{O}(|W|)$ and $|E| = \mathcal{O}(|V|)$ under the assumption that $|M|$ is constant. Now, as per Theorem 4.4, the **while** loop of GAMRCPP iterates at most $|W_g|$ times covering one goal in each horizon where $|W_g| = \mathcal{O}(|W|)$, amounting to total $\mathcal{O}(|W|(|W| \cdot G \cdot R + \max(G, R)^3 + R^3))$. As both G and R are $\mathcal{O}(|W|)$, the overall time complexity can be written as $\mathcal{O}(|W|^4)$. ■

Note that the worst-case time complexity derived here is a loose bound. As our experimental results will reveal, our algorithm does scale well for large workspaces and a large number of robots.

V. Evaluation

A. Implementation and Experimental Setup

We implement[1] our CP and the model of r^i in two ROS [50] packages that run in a computer having Intel® Core™ i7 − 4770 CPU@3.4 GHz and 16 GB of RAM. For experimentation, we consider 6 benchmark workspaces, viz.: Boston, Paris, Den, Mansion, and Room from [51], and Hospital from [22]. For coverage, we use *Quadcopters* for the first two workspaces, and *Turtlebots* for the rest. For each experiment, we *incrementally* deploy $R \in \{16, 32, 64, 128\}$ robots and repeat it for 15 times with different initial deployments to report their mean and standard deviation in the performance metrics. We take the cost of a path as the number of moves the corresponding robot performs, assuming $\tau = 1$s.

[1]Source code at https://github.com/iitkcpslab/GAMRCPP

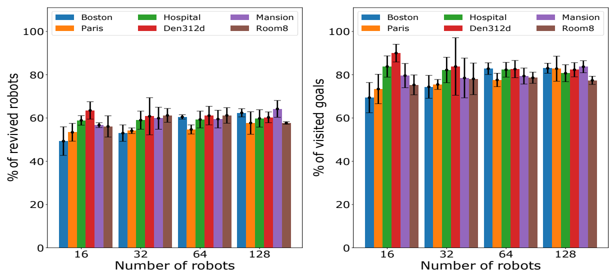


Fig. 5: Performance of Algorithm 4 for GAMRCPP

We compare GAMRCPP with BoB [35] and $\texttt{GAMRCPP}_{\texttt{MAX}}$ w.r.t. the *mission time* (T_m), where $\texttt{GAMRCPP}_{\texttt{MAX}}$ is a variant of GAMRCPP in which the horizon length λ (line 5 of Algorithm 2) is set to the maximum path length, i.e., $\lambda = \max_{i\in[R]} |\sigma'^i|$. In BoB, the robots move in *boustrophedon* motion. Whenever a robot gets surrounded by obstacles and covered cells, it backtracks to its nearest unvisited region to start another boustrophedon motion.

In each horizon of GAMRCPP, CP starts generating equal-length paths once it receives the local views from all the robots. Subsequently, all the robots start moving simultaneously after they receive the generated paths from the CP. Hence, T_m can be defined, while ignoring the *total communication time*, as follows:

$$T_m = T_c + T_p,$$

where the *total computation time* (T_c) and the *total path execution time* (T_p) are the sum of times spent on CPP and on traversing the paths, respectively, across all horizons. Note that T_p takes $\Lambda \times \tau$ seconds as all π^is have the same length Λ, which in our case is the number of moves each robot makes. We also validate our CP by performing Gazebo [52] simulations in 5 2D benchmark workspaces from [51] and 1 3D workspace with 10 quadcopters and a real experiment in a 2D workspace with 2 four-wheel ground robots. A video of our experiments is available as a supplementary material. The video is also available at `https://youtu.be/4TuIOoKlztU`.

B. Results and Analysis

1) Performance in Keeping the Robots Active: First, we measure the strength of Algorithm 4 in terms of the *percentage of revived robots* and the *percentage of visited goals* w.r.t. the number of killed robots (Figure 5). Note that each killed robot was associated with a goal earlier, and each path of the revived robots may pass through multiple such goals, thereby may increase the number of visited goals. For example, in Figure 2b (right), only r^j gets revived, but its ω^j passes through both the old goals of the killed robots r^i and r^j. Figure 5 indicates the revived robots ($\approx 60\%$ of the number of killed robots) visit $\approx 80\%$ of the goals of the killed robots, which is beneficial from the energy consumption perspective.

2) Comparison with BoB and $\texttt{GAMRCPP}_{\texttt{MAX}}$: We present our main experimental results in Table I. The *total computation time* (T_c) increases as R increases since the cost-optimal assignment of the goals to the robots, and later, collision-free path generation becomes intensive in each horizon. Unlike GAMRCPP, the robots choose their next moves independently in BoB. Although BoB performs better than GAMRCPP in sparse workspaces like in Boston and Paris city maps, it scales poorly in dense workspaces as it takes many more rounds to complete the coverage. On the other hand, in each horizon of $\texttt{GAMRCPP}_{\texttt{MAX}}$, all the active robots reach their goals, which reduces the number of horizons to complete the coverage, thereby reducing T_c.

The *path length* (Λ) decreases as R increases since deploying more robots expedites simultaneous coverage. In BoB, the collision-free backtracking path of a robot avoids not only the starting points of other robots but also the backtracking points of the other backtracking robots. It results in *detours*, increasing Λ. Moreover, backtracking points are reserved on a *first-come first-serve* basis, which may result in reserving a backtracking point for a robot that is farther. In contrast, GAMRCPP assigns goals to the robots in a cost-optimal way and accordingly generates paths that pass through $W_g \cup W_c$. If infeasible paths appear, then only paths are adjusted and may lose optimality. Thus, GAMRCPP yields better Λ. In each horizon of $\texttt{GAMRCPP}_{\texttt{MAX}}$, however, most of the active robots become *idle* after reaching their goals, as the next horizon cannot begin until the active robots with the maximum path length reach their goals. It increases λ, and so Λ.

Thus, despite larger T_c, GAMRCPP outperforms BoB and $\texttt{GAMRCPP}_{\texttt{MAX}}$ in terms of T_m because of its gain in Λ. The results are more prominent in cluttered workspaces with a large number of robots, establishing that the scalability of GAMRCPP is far better than that of BoB and $\texttt{GAMRCPP}_{\texttt{MAX}}$.

VI. Conclusion

We have proposed a goal assignment-based online homogeneous multi-robot coverage planner that guarantees complete coverage of an unknown workspace. We have considered two different types of robots to illustrate and experimentally evaluate our algorithm, establishing our algorithm's usability for a wide range of robots and applications. Experimental results further establish the scalability of our approach w.r.t. the number of robots and the size of the workspace. Though we have evaluated our algorithm only for 2D workspaces, we can seamlessly apply our algorithm to a 3D coverage mission with robots capable of performing 3D navigation. In the future, we would extend our algorithm to deal with unreliable communication and various faults that the robots may encounter during their operations, such as mechanical failure and discharge of the battery.

References

[1] A. Barrientos, J. Colorado, J. del Cerro, A. Martinez, C. Rossi, D. Sanz, and J. Valente, "Aerial remote sensing in agriculture: A practical approach to area coverage and path planning for fleets of mini aerial robots," *J. Field Robotics*, vol. 28, no. 5, pp. 667–689, 2011.

[2] S. Dogru and L. Marques, "Improved coverage path planning using a virtual sensor footprint: a case study on demining," in *ICRA*, 2019, pp. 4410–4415.

[3] A. Bircher, M. Kamel, K. Alexis, H. Oleynikova, and R. Siegwart, "Receding horizon "next-best-view" planner for 3D exploration," in *ICRA*, 2016, pp. 1462–1468.

[4] J. S. Lewis, W. Edwards, K. Benson, I. M. Rekleitis, and J. M. O'Kane, "Semi-boustrophedon coverage with a dubins vehicle," in *IROS*, 2017, pp. 5630–5637.

[5] M. Salaris, A. Riva, and F. Amigoni, "Multirobot coverage of modular environments," in *AAMAS*, 2020, pp. 1178–1186.

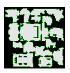
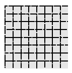
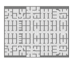
[6] M. R. Nimmagadda, S. Dattawadkar, S. Muthukumar, and V. Honkote, "Adaptive directional path planner for real-time, energy-efficient, robust navigation of mobile robots," in *ICRA*, 2020, pp. 455–461.

[7] S. Agarwal and S. Akella, "Line coverage with multiple robots," in *ICRA*, 2020, pp. 3248–3254.

[8] R. K. Ramachandran, L. Zhou, J. A. Preiss, and G. S. Sukhatme, "Resilient coverage: Exploring the local-to-global trade-off," in *IROS*, 2020, pp. 11 740–11 747.

[9] G. Hardouin, J. Moras, F. Morbidi, J. Marzat, and E. M. Mouaddib, "Next-best-view planning for surface reconstruction of large-scale 3D environments with multiple uavs," in *IROS*, 2020, pp. 1567–1574.

TABLE I: Experimental results

				T_c (s)			T_p (s)			T_m (min)		
M	*Workspace*		R	BoB	GAMRCPP$_{\text{MAX}}$	GAMRCPP	BoB	GAMRCPP$_{\text{MAX}}$	GAMRCPP	BoB	GAMRCPP$_{\text{MAX}}$	GAMRCPP
Quadcopter	Boston 256x256		16	53.9 ± 8.6	78.5 ± 11.1	119.0 ± 6.0	5436.0 ± 349.3	11831.6 ± 863.9	4092.8 ± 237.1	91.5 ± 6.0	198.5 ± 14.4	70.2 ± 4.0
			32	63.1 ± 9.8	72.2 ± 12.6	140.7 ± 13.3	2928.6 ± 166.9	13224.2 ± 2598.5	2407.6 ± 139.9	49.9 ± 2.9	221.6 ± 43.2	42.5 ± 2.6
			64	112.3 ± 28.1	67.6 ± 6.5	153.7 ± 12.9	1932.8 ± 246.4	10951.4 ± 278.6	1318.6 ± 111.2	34.1 ± 4.6	183.7 ± 4.7	24.5 ± 1.8
			128	197.7 ± 37.4	86.4 ± 13.7	223.4 ± 13.5	1364.2 ± 168.5	9821.6 ± 248.0	688.8 ± 31.7	26.0 ± 3.4	165.1 ± 4.2	15.2 ± 0.7
	Paris 256x256		16	36.8 ± 11.5	65.8 ± 49.9	122.4 ± 18.5	4279.0 ± 428.3	10368.2 ± 2869.2	3862.6 ± 115.6	71.9 ± 7.3	173.9 ± 48.1	66.4 ± 2.2
			32	45.5 ± 13.1	92.9 ± 44.9	116.2 ± 20.2	2355.4 ± 253.6	12398.4 ± 2051.4	2212.8 ± 109.2	40.0 ± 4.4	208.2 ± 33.9	38.8 ± 1.9
			64	90.8 ± 27.8	81.9 ± 42.6	147.8 ± 26.3	1634.0 ± 275.5	12055.0 ± 2326.6	1271.8 ± 86.1	28.7 ± 5.1	202.3 ± 39.3	23.7 ± 1.7
			128	151.5 ± 60.5	87.0 ± 9.1	264.9 ± 35.2	1120.0 ± 290.2	11441.8 ± 851.2	766.8 ± 73.2	21.2 ± 5.8	192.1 ± 14.2	17.2 ± 1.7
Turtlebot	Den 65x81		16	0.7 ± 0.1	0.4 ± 0.0	2.3 ± 1.5	413.4 ± 44.3	1671.2 ± 175.0	353.4 ± 29.0	6.9 ± 0.7	27.9 ± 2.9	5.9 ± 0.5
			32	1.2 ± 0.1	0.4 ± 0.0	2.7 ± 1.6	308.2 ± 25.8	1345.4 ± 276.5	193.4 ± 19.2	5.2 ± 0.4	22.4 ± 4.6	3.3 ± 0.3
			64	2.3 ± 0.4	0.5 ± 0.0	2.0 ± 0.1	258.8 ± 36.3	905.6 ± 124.5	114.4 ± 14.5	4.4 ± 0.6	15.1 ± 2.1	1.9 ± 0.2
			128	4.5 ± 1.1	0.8 ± 0.1	2.9 ± 0.5	231.8 ± 50.1	468.0 ± 73.2	69.6 ± 13.4	3.9 ± 0.9	7.8 ± 1.2	1.2 ± 0.2
	Mansion 133x270		16	7.5 ± 0.9	7.4 ± 0.2	23.8 ± 2.5	1185.8 ± 79.2	5271.6 ± 1034.3	1049.4 ± 70.8	19.9 ± 1.3	88.0 ± 17.2	17.9 ± 1.2
			32	13.7 ± 3.6	7.6 ± 0.2	21.5 ± 2.3	845.0 ± 114.9	4260.0 ± 415.5	618.4 ± 34.9	14.3 ± 2.0	71.1 ± 6.9	10.7 ± 0.6
			64	23.1 ± 7.8	7.7 ± 0.3	22.7 ± 6.1	610.6 ± 148.5	4137.4 ± 1478.6	380.6 ± 70.6	10.6 ± 2.6	69.1 ± 24.6	6.7 ± 1.3
			128	37.4 ± 7.3	8.7 ± 0.5	30.6 ± 3.6	432.4 ± 70.5	2403.8 ± 407.0	187.8 ± 23.8	7.8 ± 1.3	40.2 ± 6.8	3.6 ± 0.5
	Room 64x64		16	2.3 ± 1.3	0.9 ± 0.3	1.5 ± 0.2	550.6 ± 54.2	2322.0 ± 296.0	487.2 ± 48.8	9.2 ± 0.9	38.7 ± 4.9	8.1 ± 0.8
			32	5.3 ± 2.1	0.8 ± 0.2	1.7 ± 0.3	447.8 ± 19.5	2087.6 ± 422.4	266.4 ± 28.6	7.6 ± 0.3	34.8 ± 7.0	4.5 ± 0.5
			64	9.1 ± 5.4	0.8 ± 0.2	2.2 ± 0.3	360.6 ± 122.8	1304.0 ± 148.1	153.0 ± 9.4	6.2 ± 2.1	21.7 ± 2.5	2.6 ± 0.2
			128	8.1 ± 3.3	1.3 ± 0.1	4.5 ± 0.4	234.0 ± 18.8	797.8 ± 116.0	84.4 ± 6.3	4.0 ± 0.3	13.3 ± 1.9	1.5 ± 0.1
	Hospital 96x96		16	2.8 ± 0.6	2.9 ± 1.4	7.1 ± 0.4	932.8 ± 84.2	3882.6 ± 266.5	862.1 ± 45.6	15.6 ± 1.4	64.8 ± 4.4	14.5 ± 0.8
			32	4.7 ± 1.1	2.8 ± 1.2	7.6 ± 0.7	617.7 ± 83.6	3534.1 ± 410.6	510.3 ± 37.3	10.4 ± 1.4	58.9 ± 6.8	8.6 ± 0.6
			64	7.5 ± 1.8	2.9 ± 1.1	8.7 ± 1.2	440.9 ± 65.5	2353.9 ± 289.1	290.2 ± 31.1	7.5 ± 1.1	39.3 ± 4.8	6.7 ± 0.7
			128	18.0 ± 3.6	3.7 ± 0.6	12.3 ± 2.0	442.8 ± 90.2	1545.4 ± 213.9	163.9 ± 15.1	7.7 ± 1.6	25.8 ± 3.6	5.0 ± 0.5

[10] W. Luo, C. Nam, G. Kantor, and K. P. Sycara, "Distributed environmental modeling and adaptive sampling for multi-robot sensor coverage," in *AAMAS*, 2019, pp. 1488–1496.

[11] T. Charrier, A. Queffelec, O. Sankur, and F. Schwarzentruber, "Reachability and coverage planning for connected agents," in *IJCAI*, 2019, pp. 144–150.

[12] A. Özdemir, M. Gauci, A. Kolling, M. D. Hall, and R. Groß, "Spatial coverage without computation," in *ICRA*, 2019, pp. 9674–9680.

[13] I. Vandermeulen, R. Groß, and A. Kolling, "Turn-minimizing multirobot coverage," in *ICRA*, 2019, pp. 1014–1020.

[14] S. N. Das and I. Saha, "Rhocop: receding horizon multi-robot coverage," in *ICCPS*, 2018, pp. 174–185.

[15] R. Bormann, F. Jordan, J. Hampp, and M. Hägele, "Indoor coverage path planning: Survey, implementation, analysis," in *ICRA*, 2018, pp. 1718–1725.

[16] E. Galceran and M. Carreras, "A survey on coverage path planning for robotics," *Robotics Auton. Syst.*, vol. 61, no. 12, pp. 1258–1276, 2013.

[17] T. M. Cabreira, L. B. Brisolara, and P. R. Ferreira Jr, "Survey on coverage path planning with unmanned aerial vehicles," *Drones*, vol. 3, no. 1, p. 4, 2019.

[18] K. Yu, J. M. O'Kane, and P. Tokekar, "Coverage of an environment using energy-constrained unmanned aerial vehicles," in *ICRA*, 2019, pp. 3259–3265.

[19] A. Riva and F. Amigoni, "A GRASP metaheuristic for the coverage of grid environments with limited-footprint tools," in *AAMAS*, 2017, pp. 484–491.

[20] M. Dharmadhikari, T. Dang, L. Solanka, J. Loje, H. Nguyen, N. Khedekar, and K. Alexis, "Motion primitives-based path planning for fast and agile exploration using aerial robots," in *ICRA*, 2020, pp. 179–185.

[21] G. Sharma, A. Dutta, and J. Kim, "Optimal online coverage path planning with energy constraints," in *AAMAS*, 2019, pp. 1189–1197.

[22] E. A. Jensen and M. L. Gini, "Online multi-robot coverage: Algorithm comparisons," in *AAMAS*, 2018, pp. 1974–1976.

[23] X. Chen, T. M. Tucker, T. R. Kurfess, and R. W. Vuduc, "Adaptive deep path: Efficient coverage of a known environment under various configurations," in *IROS*, 2019, pp. 3549–3556.

[24] A. Kleiner, R. Baravalle, A. Kolling, P. Pilotti, and M. Munich, "A solution to room-by-room coverage for autonomous cleaning robots," in *IROS*, 2017, pp. 5346–5352.

[25] A. Mannucci, S. Nardi, and L. Pallottino, "Autonomous 3D exploration of large areas: A cooperative frontier-based approach," in *MESAS*, vol. 10756. Springer, 2017, pp. 18–39.

[26] A. Howard, L. E. Parker, and G. S. Sukhatme, "Experiments with a large heterogeneous mobile robot team: Exploration, mapping, deployment and detection," *Int. J. Robotics Res.*, vol. 25, no. 5-6, pp. 431–447, 2006.

[27] W. Burgard, M. Moors, C. Stachniss, and F. E. Schneider, "Coordinated multi-robot exploration," *IEEE Trans. Robotics*, vol. 21, no. 3, pp. 376–386, 2005.

[28] N. Hazon and G. A. Kaminka, "Redundancy, efficiency and robustness in multi-robot coverage," in *ICRA*, 2005, pp. 735–741.

[29] X. Zheng, S. Jain, S. Koenig, and D. Kempe, "Multi-robot forest coverage," in *IROS*, 2005, pp. 3852–3857.

[30] W. Burgard, M. Moors, D. Fox, R. G. Simmons, and S. Thrun, "Collaborative multi-robot exploration," in *ICRA*, 2000, pp. 476–481.

[31] Z. J. Butler, A. A. Rizzi, and R. L. Hollis, "Complete distributed coverage of rectilinear environments," in *Algorithmic and Computational Robotics*. AK Peters/CRC Press, 2001, pp. 61–68.

[32] N. Hazon, F. Mieli, and G. A. Kaminka, "Towards robust on-line multirobot coverage," in *ICRA*, 2006, pp. 1710–1715.

[33] I. M. Rekleitis, A. P. New, E. S. Rankin, and H. Choset, "Efficient boustrophedon multi-robot coverage: an algorithmic approach," *Ann. Math. Artif. Intell.*, vol. 52, no. 2-4, pp. 109–142, 2008.

[34] E. A. Jensen and M. L. Gini, "Rolling dispersion for robot teams," in *IJCAI*, 2013, pp. 2473–2479.

[35] H. H. Viet, V. Dang, S. Y. Choi, and T. Chung, "BoB: an online coverage approach for multi-robot systems," *Appl. Intell.*, vol. 42, no. 2, pp. 157–173, 2015.

[36] L. Siligardi, J. Panerati, M. Kaufmann, M. Minelli, C. Ghedini, G. Beltrame, and L. Sabattini, "Robust area coverage with connectivity maintenance," in *ICRA*, 2019, pp. 2202–2208.

[37] Turtlebot. [Online]. Available: https://www.turtlebot.com/

[38] P. Oberlin, S. Rathinam, and S. Darbha, "A transformation for a heterogeneous, multiple depot, multiple traveling salesman problem," in *ACC*, 2009, pp. 1292–1297.

[39] S. M. LaValle, *Planning Algorithms*. Cambridge University Press, 2006.

[40] S. J. Russell and P. Norvig, *Artificial Intelligence - A Modern Approach, Third International Edition*. Pearson Education, 2010.

[41] H. W. Kuhn, "The hungarian method for the assignment problem," *Nav. Res. Logist.*, vol. 2, no. 1-2, pp. 83–97, 1955.

[42] F. Bourgeois and J. Lassalle, "An extension of the munkres algorithm for the assignment problem to rectangular matrices," *Commun. ACM*, vol. 14, no. 12, pp. 802–804, 1971.

[43] M. A. Erdmann and T. Lozano-Pérez, "On multiple moving objects," in *ICRA*, 1986, pp. 1419–1424.

[44] D. Silver, "Cooperative pathfinding," in *AIIDE*, 2005, pp. 117–122.

[45] J. P. van den Berg and M. H. Overmars, "Prioritized motion planning for multiple robots," in *IROS*, 2005, pp. 430–435.

[46] M. Turpin, K. Mohta, N. Michael, and V. Kumar, "Goal assignment and trajectory planning for large teams of aerial robots," in *RSS*, 2013.

[47] W. Hönig, S. Kiesel, A. Tinka, J. W. Durham, and N. Ayanian, "Conflict-based search with optimal task assignment," in *AAMAS*, 2018, pp. 757–765.

[48] R. Hassin and S. Rubinstein, "Approximations for the maximum acyclic subgraph problem," *Inf. Process. Lett.*, vol. 51, no. 3, pp. 133–140, 1994.

[49] S. Skiena, *The Algorithm Design Manual, Second Edition*. Springer, 2008.

[50] Robot Operating System. [Online]. Available: https://www.ros.org/

[51] R. Stern, N. R. Sturtevant, A. Felner, S. Koenig, H. Ma, T. T. Walker, J. Li, D. Atzmon, L. Cohen, T. K. S. Kumar, R. Barták, and E. Boyarski, "Multi-agent pathfinding: Definitions, variants, and benchmarks," in *SOCS*, 2019, pp. 151–159.

[52] N. P. Koenig and A. Howard, "Design and use paradigms for gazebo, an open-source multi-robot simulator," in *IROS*, 2004, pp. 2149–2154.